UNIVERSITÀ DEGLI STUDI DI UDINE
hic sunt futura

BELLANTONI STANISLAO

COMPUTER VISION TECHNIQUES FOR FOOD SEGMENTATION

MASTER THESIS

submitted in fulfilment of the requirement for the degree of[1]
Diplom-Ingenieurin/Diplom-Ingenieur – Laurea Magistrale

In the framework of the Double Degree Programme

"Information and Communications Engineering (ICE)",
Alpen-Adria-Universität Klagenfurt
and
"Multimedia Communications and Information Technology",
University of Udine

Supervisor / Supervisors

Piciarelli Claudio
University of Udine
Department of Mathematics, Computer Science and Physics

Kyamakya Kyandoghere
Alpen-Adria-Universität
Department of Intelligent System Technologies

Academic year: 2024/2025

[1] The completed study programme is a Double-Degree Programme. The graduate obtains the right to hold one Master's Degree, in Austria "Diplom-Ingenieurin" or "Diplom-Ingenieur", in Italy "Laurea Magistrale".

Acknowledgements

Per aspera ad astra.

Abstract

Food is not just sustenance, it's part of our lives and our culture. Our relationship with food goes beyond its nutritional value; it includes cultural, social, and personal dimensions. In today's digitally connected world, the ability to analyze and understand food images has far-reaching implications. Whether it's for dietary planning, food quality assessment or restaurant recommendations, the accurate segmentation and recognition of food in images hold enormous importance.

Food recognition and segmentation is an important and challenging task in computer vision. With the proliferation of food photos across social media and recipe sites, there is a growing need for intelligent systems that can automatically analyze food content in images. Accurate food analysis has many potential applications such as dietary assessment and monitoring, food journaling apps, automated meal planning and recommendation systems.

Recent advances in deep learning and convolutional neural networks have led to promising improvements in food image recognition and segmentation accuracy. However, identifying and isolating food items from cluttered backgrounds remains difficult due to issues like food texture variation, complex plating arrangements, and occlusion. In particular, semantic segmentation, or pixel-level labeling of food images, is a hard problem that state-of-the-art models still struggle with.

This work presents a research of deep learning techniques for food segmentation and recognition in images. The deep learning models studied include classification networks like ResNet34 for food category prediction, and segmentation models like PSPNet and DeepLabv3+ for pixel-wise food labeling. These models utilize convolutional neural network architectures that learn rich feature representations from large, labeled datasets.

Table of contents

1. INTRODUCTION

Image segmentation [2] is an important task in computer vision, the goal is to divide an image into different segments that represent various objects or parts of objects. This process is essential for many applications, including medical imaging, self-driving cars, robotics, and digital image processing. Essentially, image segmentation helps us break down an image into more manageable parts, making it easier to analyze and understand.

In recent years, deep learning and convolutional neural networks have significantly improved our approach to image segmentation. Unlike traditional methods that depended on manually crafted features and rules, deep learning techniques learn features directly from the data. This shift has led to better performance and more generalizable models.

A particularly challenging area within image segmentation is semantic segmentation. This involves assigning a class label to every pixel in an image, which means accurately distinguishing between different objects and their boundaries. This level of detail is especially important in fields like medical imaging, where pinpointing the exact location and size of a tumor can influence treatment options.

Segmenting food images is a specific application of semantic segmentation that is challenging and important. Accurately segmenting food items in images has many practical applications, such as dietary assessment, automated food logging, nutritional analysis, and others. For example, in dietary assessment, correctly identifying and segmenting food items can help in calculating nutritional intake, which is important for health monitoring and meal planning.

However, this task is not easy due to several challenges. One of the main issues is the high variability in how food looks. Foods can have different colors, textures, and shapes, even if they belong to the same category. This variability can be due to different cooking

methods, lighting conditions, and presentation styles. For instance, a salad can look very different depending on the ingredients and how they are arranged on the plate.

Another challenge is the complex arrangement of food items on plates. Unlike other objects that might have more predictable shapes, food items can be piled up, layered, or mixed together, making it hard to draw clear boundaries. Occlusion, where one food item partially covers another, also adds to the difficulty.

Research into food image segmentation aims to resolve these challenges and develop algorithms that can handle the diversity and complexity of food images. Improving food image segmentation can enhance applications in health and nutrition and provide better tools for food recognition and analysis. [2]

This work aims to explore these challenges by studying and evaluating state-of-the-art deep learning techniques for food image segmentation. By using advanced segmentation models and comprehensive datasets, this research aims to find insights that are accurate and reliable.

1.1. Thesis Objectives

The main objective of this research is to explore and evaluate the effectiveness of deep neural networks for semantic segmentation of food images.

Specifically, this work aims to:

- Compare different deep learning segmentation models, including DeeplabV3, UNet, FPN, etc. in the context of food image segmentation.
- Utilize a substantial dataset, UEC-FoodPixComplete, with 9000 training images and 1000 test images, divided into 103 classes representing various types of food.
- Assess the models' performance using a range of metrics, including accuracy, precision, recall, and Jaccard score.
- Explore the capability of using pre-trained ResNet34 encoders on ImageNet as feature extractors for food image segmentation.
- Analyze the results and provide insights into the effectiveness of deep neural networks for food image segmentation.

1.2. Methodological Approach

To address these objectives, a thorough review of existing literature on segmentation algorithms and datasets was conducted.

This research uses Python 3.8, PyTorch and the "segmentation_models.pytorch" library to implement and test 6 segmentation algorithms (UNet, Unet++, PSPNet, FPN, DeepLabV3, DeepLabV3+), leveraging pre-trained models and a range of encoders to ensure flexibility and efficiency in the experimentation process.

1.3. Thesis Structure

This thesis is structured into four main chapters each contributing to a comprehensive understanding of food image segmentation:

- Chapter 2 provides an introduction of the fundamentals of machine learning.
- Chapter 3 provides an overview of the state-of-the-art techniques in machine learning, with a focus on their applications to food image segmentation tasks.
- Chapter 3 describes the dataset, tools, methods and model configurations used in the research; this chapter sets the stage for the practical implementation of the research.
- Chapter 4 presents the results of experiments and model performances.
- Chapter 5 analyzes the results and explores practical applications in gastronomy.

By following this structure, this thesis aims to provide a detailed and systematic exploration of food image segmentation using deep learning techniques, contributing valuable insights.

2. MACHINE LEARNING FUNDAMENTALS

This chapter covers an introduction of the techniques in machine learning, with a focus on their applications to image segmentation tasks.

We begin by introducing the three main machine learning paradigms:

- **Supervised Learning:** algorithms that learn to map input features to output labels/targets from labeled training data. This includes classification for predicting discrete categories and regression for predicting continuous values.
- **Unsupervised Learning:** algorithms that discover patterns and structures in unlabeled data, such as clustering algorithms and dimensionality reduction techniques.
- **Reinforcement Learning:** algorithms where an agent learns an optimal policy through trial-and-error interactions with an environment, aimed at maximizing cumulative rewards.

We then explore deep learning, a powerful subfield involving artificial neural networks, with concepts like network architectures, activation functions, backpropagation, and convolutional neural networks (CNNs).

Finally, we focus on the most used deep learning models and techniques specifically designed for image segmentation tasks, which partition images into meaningful segments or objects. Their underlying principles and applications are discussed.

Throughout, we provide theoretical foundations, mathematical frameworks, and implementation details of these techniques.

2.1. Introduction to Machine Learning

Machine learning is a field of artificial intelligence that focuses on developing systems and **algorithms that can learn from data and make accurate predictions or decisions without being explicitly programmed.** It has rapidly emerged as one of the most transformative and widely adopted technologies across a multitude of domains, revolutionizing areas such as computer vision, natural language processing, predictive analytics, and autonomous systems.

At its core, machine learning involves creating mathematical models from sample data, known as training data, in order to make predictions or decisions on new, unseen data. Depending on the availability and nature of the training data, as shown in Fig. 1 machine learning can be categorized into three main types: **Supervised, Unsupervised and Reinforcement learning. [1]**

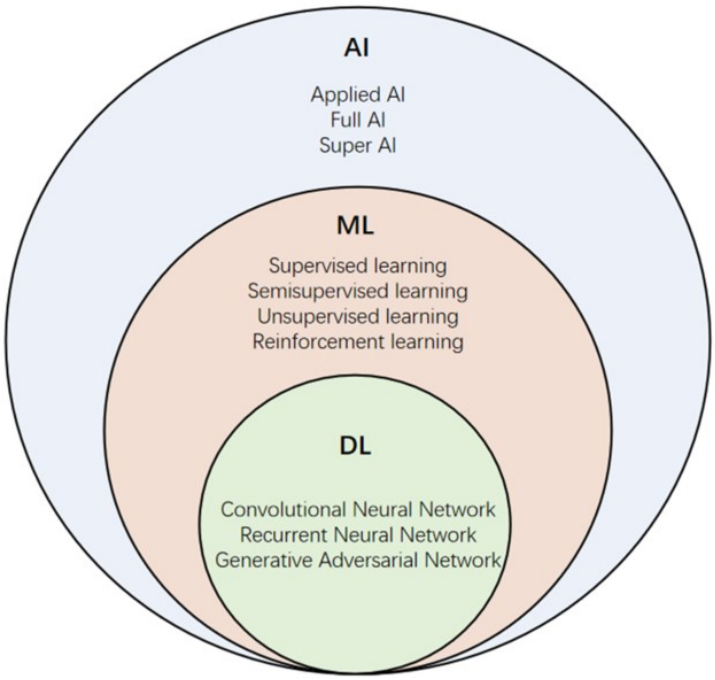

Figure 1 - The relationship and main types of artificial intelligence, machine learning and deep learning. Credits Wikipedia Commons

2.1.1. Supervised Learning

Supervised learning [1] is a machine learning paradigm where the training data consists of **input-output** pairs. As shown in Fig. 2 the goal is to learn a function or mapping from the input features (independent variables) to the output labels or targets. This mapping is achieved by exposing the learning algorithm to a set of labeled examples, allowing it to discover patterns and relationships between the inputs and outputs.

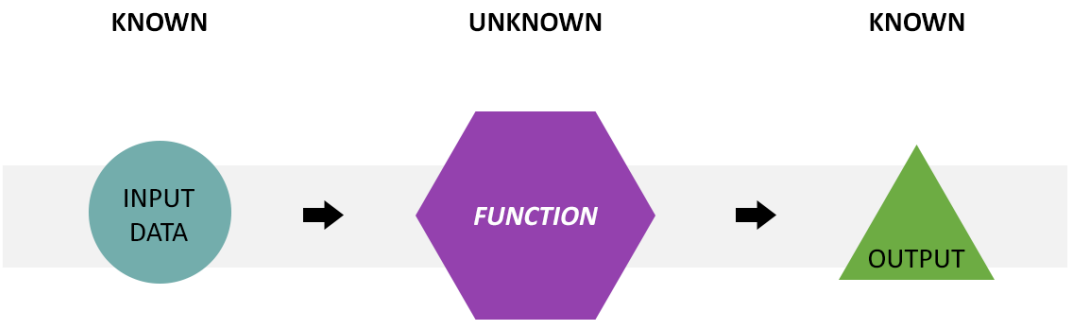

Figure 2 - Supervised Learning

The key strength of supervised learning lies in its ability to learn complex patterns and relationships directly from labeled examples, without the need for explicit programming of rules or feature engineering. This is particularly powerful in domains where it is challenging to specify rules or extract meaningful features manually.

The supervised learning process typically involves the following steps:

1. **Data Collection and Preprocessing:** obtaining a labeled dataset and performing necessary preprocessing steps, such as handling missing data, feature scaling, and data normalization.
2. **Train-Test Split:** splitting the labeled dataset into two subsets: a training set used to train the model and a test set used for evaluating the model's performance on unseen data.
3. **Model Selection and Training:** choosing an appropriate machine learning algorithm and training the model on the training set. During training, the algorithm learns to map input features to output labels by minimizing a predefined loss or error function.
4. **Model Evaluation:** assessing the performance of the trained model on the test set using appropriate evaluation metrics, such as accuracy, precision, recall, and

F1-score for classification, or mean squared error (MSE) or R-squared for regression.

5. **Model Tuning and Validation:** optimizing the model's performance by tuning its hyperparameters (e.g. learning rate, regularization strength) and validating the model on a separate validation set or using techniques like cross-validation.
6. **Model Deployment:** once the model is satisfactory, deploying it to a production environment for making predictions on new, unseen data instances.

Classification

Classification is a core task in supervised learning that deals with assigning discrete categorical labels or classes to input data instances. Unlike regression, where the output is a continuous numerical value, classification aims to predict the class or category that an input instance belongs to.

In a classification problem, the output variable is a categorical or nominal variable that can take on a finite set of values or labels. For example, in an image classification task, as shown in Fig. 3 the input could be an image, and the output could be a label identifying the object or scene in the image (e.g. "rectangle," "circle," "hexagon"). Similarly, in a spam detection problem, the input could be an email, and the output would be a binary label indicating whether the email is spam or not.

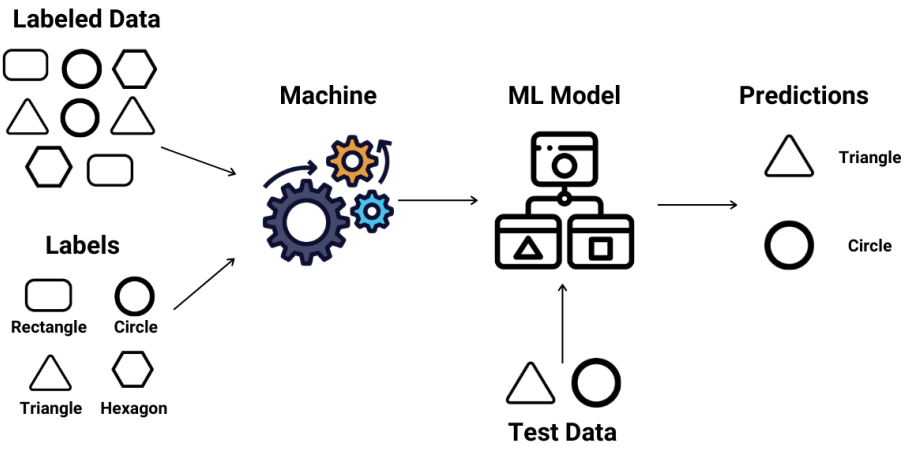

Figure 3 - Supervised learning classification
Credits enjoyalgorithms.com

The learned mapping function in classification problems takes the form:

$$\mathbf{y = f(x; \theta)}$$

where **x** represents the input feature vector, **y** is the predicted class label, and **θ** represents the model parameters or weights that need to be learned from the training data.

During the training process, the classification algorithm aims to find the optimal set of parameters θ* **that minimizes a predefined loss function** L(y, f(x; θ)), which measures the discrepancy between the predicted class labels f(x; θ) and the true class labels y in the training data.

Common loss functions for classification include:

- **Cross-Entropy Loss**
- Hinge Loss (for Support Vector Machines)

The optimization problem involves finding the parameters θ* that minimize the overall loss function over the entire training dataset. This optimization is typically performed using techniques like gradient descent or its variants, or specialized algorithms like the Sequential Minimal Optimization (SMO) algorithm for Support Vector Machines (SVMs).

Depending on the assumed form of the mapping function f(x; θ) and the number of output classes, classification algorithms can be further categorized into:

- **Linear Classifiers:** the mapping function is a linear combination of the input features, making it suitable for separating classes with a linear decision boundary (e.g. Logistic Regression, Linear Discriminant Analysis).
- **Non-linear Classifiers:** the mapping function can take more complex non-linear forms, such as decision trees, neural networks, or kernel-based models, enabling the classification of instances that are not linearly separable.
- **Binary Classifiers:** these classifiers handle problems with two output classes (e.g. spam vs. non-spam).
- **Multi-class Classifiers:** these classifiers can handle problems with more than two output classes (e.g. image classification with multiple object categories).

Popular classification algorithms include logistic regression, decision trees, random forests, support vector machines (SVMs), naive Bayes classifiers, and neural networks (for deep learning-based classification). [1]

The performance of classification models is typically evaluated using metrics such as **accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC).** These metrics quantify the model's ability to correctly classify instances into their respective classes, providing insights into the model's performance and potential biases or imbalances in the data.

Classification has numerous applications across various domains, including image and speech recognition, text categorization, fraud detection. The choice of classification algorithm depends on factors such as the complexity of the problem, the size and quality of the available data, the desired level of interpretability, and computational constraints.

Regression

Regression is a core task in supervised learning that focuses on modeling the relationship between a set of independent input variables (features) and a continuous dependent output variable. The goal of regression is to learn a mapping function that can accurately predict the numerical output value given new input data.

In a regression problem, the output variable is a real-valued, continuous quantity. For example, in a salary prediction task, as shown in Fig. 4, the output variable could be the expected annual salary, while the input features may include variables such as years of experience, education level, job title, industry, and location.

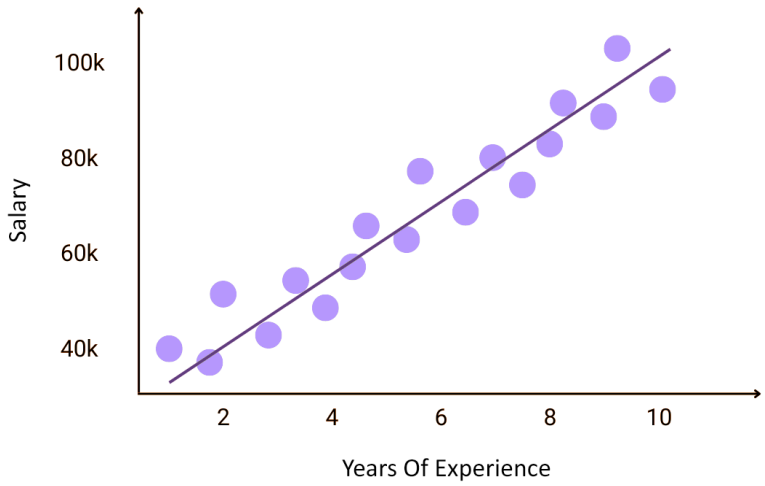

Figure 4 - Supervised learning classification

The learned mapping function in regression problems takes the form:

$$\mathbf{y = f(x; \theta)}$$

where **x** represents the input feature vector, **y** is the predicted output (e.g. salary), and **θ** represents the model parameters or weights that need to be learned from the training data.

During the training process, the regression algorithm aims to find the optimal set of parameters θ* that minimizes a predefined loss function L(y, f(x; θ)), which measures the discrepancy between the predicted output f(x; θ) and the true output y in the training data. Common loss functions for regression include:

- **Mean Squared Error (MSE)**
- Mean Absolute Error (MAE)

The optimization problem involves finding the parameters θ* that minimize the overall loss function over the entire training dataset. This optimization is typically performed using techniques like **gradient descent** or its variants.

Depending on the assumed form of the mapping function f(x; θ), regression algorithms can be further categorized into:

- **Linear Regression:** the mapping function is a linear combination of the input features, making it suitable for modeling linear relationships.
- **Polynomial Regression:** the mapping function includes polynomial terms, allowing it to capture non-linear relationships up to a certain degree.
- **Non-linear Regression:** the mapping function can take more complex non-linear forms, such as decision trees, neural networks, or kernel-based models, enabling the modeling of highly intricate relationships.

Regularization techniques, such as **L1 (Lasso) or L2 (Ridge) regularization**, are often employed in regression to prevent overfitting and improve the generalization performance of the learned models, especially when dealing with high-dimensional or correlated input features.

Regression models are widely used in various domains, including finance, economics, engineering , and scientific research [1].

2.1.2. Unsupervised Learning

Unlike supervised learning, where the data is labeled with corresponding output values or classes, unsupervised learning [1] deals with finding patterns and extracting insights from unlabeled data. As shown in Fig. 5, the goal of unsupervised learning is to discover the inherent structure, relationships, or underlying representations within the input data without relying on predefined target variables or labels.

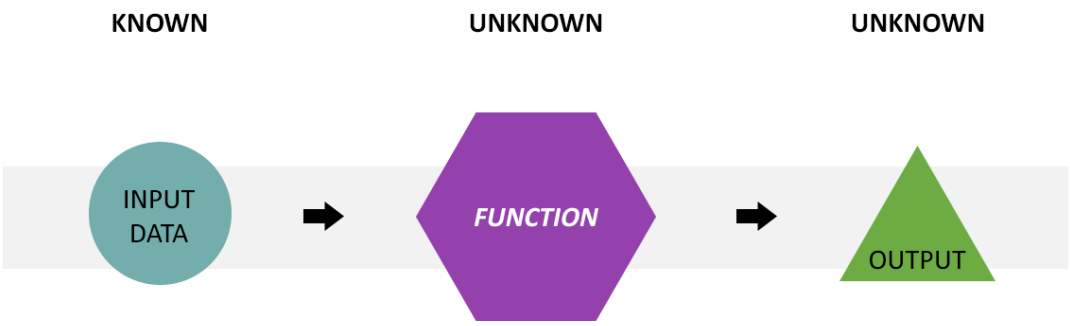

Figure 5 - Unsupervised Learning

Unsupervised learning algorithms can be broadly categorized into two main tasks: clustering and dimensionality reduction.

Clustering

Clustering is the process of grouping similar data instances together based on their inherent characteristics or features. The goal is to partition the data into distinct clusters or groups, such that instances within the same cluster are more similar to each other than to instances in other clusters.

Some widely used clustering algorithms include:

- **K-Means:** this algorithm partitions the data into K clusters by iteratively assigning instances to the nearest cluster centroid and updating the centroids based on the instances in each cluster.

- **Hierarchical Clustering:** this technique builds a hierarchical tree-like structure by merging or splitting clusters based on their proximity or dissimilarity, resulting in a dendrogram that can be used for data exploration and visualization.
- **DBSCAN:** density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a density-based algorithm that identifies clusters as dense regions separated by low-density areas, making it suitable for detecting arbitrary-shaped clusters and handling noise or outliers.

Clustering has numerous applications, including customer segmentation, anomaly detection, image segmentation, and exploratory data analysis.

Dimensionality Reduction

Dimensionality reduction techniques aim to reduce the number of features or dimensions in the input data while preserving the most essential information or structure (Fig. 6). This is particularly useful when dealing with high-dimensional data, where visualization and analysis become challenging, and redundant or irrelevant features can negatively impact the performance of machine learning models.

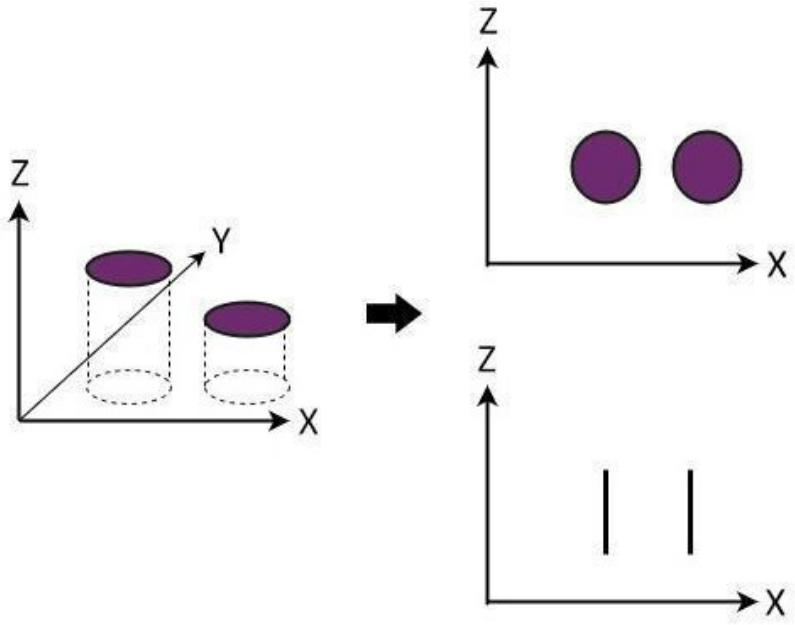

Figure 6 - Dimensionality reduction example

Popular dimensionality reduction algorithms include:

- **Principal Component Analysis (PCA)**: is a linear technique that projects the data onto a lower-dimensional subspace by identifying the directions (principal components) that capture the maximum variance in the data. (Fig. 7)
- **t-Distributed Stochastic Neighbor Embedding (t-SNE):** is a non-linear technique that maps high-dimensional data into a lower-dimensional space while preserving the local structure and similarity relationships between instances.
- **Autoencoders:** are a type of neural network architecture that learns to reconstruct the input data by first encoding it into a lower-dimensional representation and then decoding it back to the original space. The learned code can be used as a lower-dimensional representation of the data. (Fig. 7)

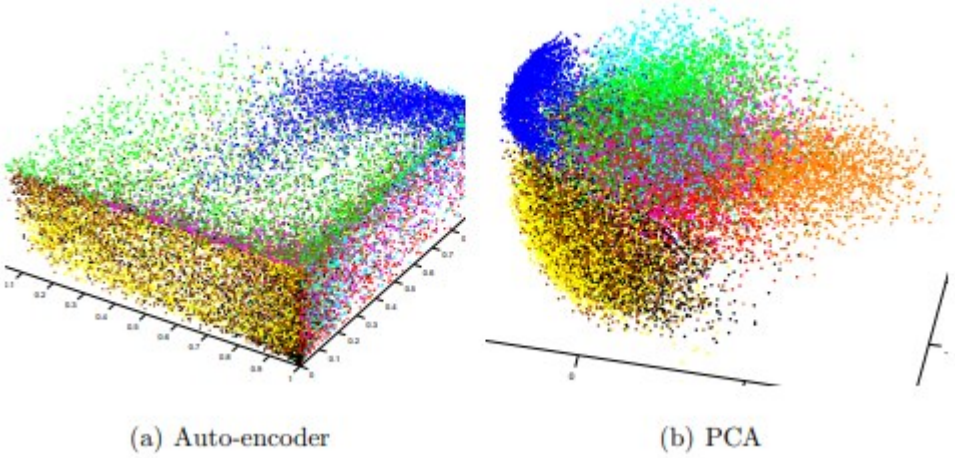

Figure 7 - Autoencoders vs PCA - Credits Wikipedia Commons

Dimensionality reduction techniques are valuable for data visualization, feature selection, and preprocessing for other machine learning tasks.

Unlike supervised learning, where the objective is to learn a mapping function from input features to output labels, unsupervised learning algorithms aim to discover patterns, structures, or representations within the data itself. The lack of labeled data makes unsupervised learning particularly useful in exploratory data analysis, data preprocessing, and scenarios where obtaining labeled data is challenging or expensive.

However, unsupervised learning algorithms can be sensitive to the choice of hyperparameters, such as the number of clusters or the dimensionality of the reduced space and may require domain knowledge or additional validation techniques to interpret and evaluate the results.

In many real-world applications, a combination of unsupervised and supervised learning techniques is often employed, where unsupervised methods are used for data exploration, feature extraction, or preprocessing, followed by supervised learning for prediction or classification tasks [1].

2.1.3. Reinforcement Learning

Reinforcement learning [1] is a paradigm of machine learning that is inspired by how humans and animals learn from their interactions with the environment. Unlike supervised and unsupervised learning, reinforcement learning does not rely on labeled data or predefined outputs. Instead, it focuses on learning an optimal policy or behavior for an agent to achieve a specific goal or maximize a reward signal.

As shown in Fig. 8 in a reinforcement learning problem, an agent interacts with an environment by taking actions and receiving rewards or penalties based on the outcomes of those actions. The goal of the agent is to learn a policy or strategy that maximizes the cumulative reward over time.

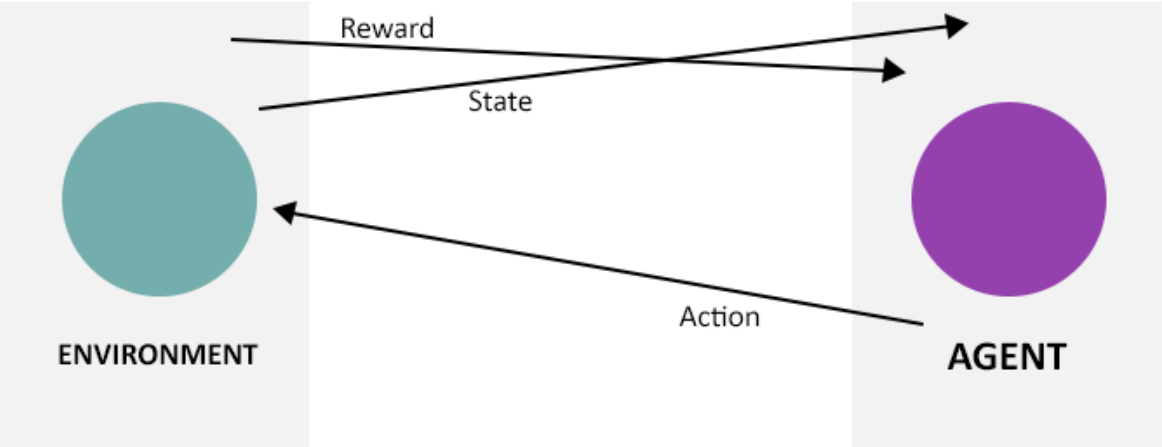

Figure 8 - Reinforcement learning

The reinforcement learning framework typically consists of the following components:

- **Environment:** is the setting in which the agent operates and interacts. It defines the possible states, actions, and rewards.
- **Agent:** the agent is the decision-making entity that takes actions based on the current state of the environment and the learned policy.
- **State:** represents the current condition or configuration of the environment, which is observed by the agent.
- **Action:** is the decision or behavior taken by the agent in a given state, which can influence the subsequent state of the environment.
- **Reward:** is a numerical signal that evaluates the desirability or quality of the action taken by the agent in a particular state. The agent's goal is to maximize the cumulative reward over time.
- **Policy:** is the strategy or decision-making function that maps states to actions, defining the behavior of the agent in the environment.

The reinforcement learning process involves the agent repeatedly interacting with the environment, taking actions, and receiving rewards or penalties. The agent then updates its policy based on these experiences, aiming to learn the optimal behavior that maximizes the expected cumulative reward.

Two main approaches to reinforcement learning are:

1. **Value-based methods:** these methods estimate the value function, which represents the expected cumulative reward for being in a particular state or taking a specific action.
2. **Policy-based methods:** these methods directly learn the optimal policy that maps states to actions, without estimating value functions.

Reinforcement learning has been successfully applied to various domains, including robotics, game playing (e.g. chess, Go) and autonomous systems (e.g, self-driving cars).

However, reinforcement learning can be challenging due to issues such as the exploration-exploitation trade-off (balancing exploration of new actions and exploitation of known good actions), credit assignment (determining which actions led to positive or negative rewards), and the curse of dimensionality (dealing with high-dimensional state and action spaces).

Advancements in deep learning and the use of neural networks as approximators have greatly improved performance of reinforcement learning algorithms [1].

2.2. Introduction to Deep Learning

Deep learning [1] is a subfield of machine learning that has gained significant attention and achieved remarkable success in various domains, including computer vision, natural language processing, and speech recognition. At the core of deep learning lies the concept of artificial neural networks, which are inspired by the structure and function of the human brain.

Deep learning models, particularly neural networks, are capable of automatically learning hierarchical representations and extracting complex patterns from raw data, such as images, text, or audio. This ability to learn rich representations directly from data has been a key factor in the success of deep learning in solving challenging problems that were previously difficult or intractable with traditional machine learning methods.

Deep learning architectures are characterized by their depth, which refers to the number of hidden layers in the neural network. These deep architectures allow the network to learn increasingly abstract and complex representations of the input data as the information propagates through the layers. The depth of the network, combined with the use of non-linear activation functions, enables deep learning models to capture intricate patterns and relationships within the data, leading to superior performance in various tasks.

Deep learning has revolutionized various fields, including computer vision, where convolutional neural networks (CNNs) have become the de facto standard for image-related tasks. CNNs are designed to leverage the spatial and local correlation present in image data, making them highly effective for tasks such as image classification, object detection, and image segmentation [1].

In the following sections, we will delve deeper into the core concepts of deep learning, including artificial neural networks, activation functions, the backpropagation algorithm and convolutional neural networks, which form the foundation for image segmentation tasks.

2.2.1. Artificial Neural Networks

Artificial neural networks (ANNs) are computational models inspired by the structure and function of the biological neural networks found in the human brain. They are composed of interconnected nodes, called neurons, which are organized into layers. These layers include an input layer, one or more hidden layers, and an output layer.

As shown in Fig. 9 the basic unit of an ANN is the neuron, which performs a simple mathematical operation on its inputs. Each neuron receives inputs from the previous layer, multiplies them by corresponding weights, and then sums these weighted inputs. This summation is then passed through an activation function, which introduces non-linearity into the network, allowing it to model complex relationships within the data.

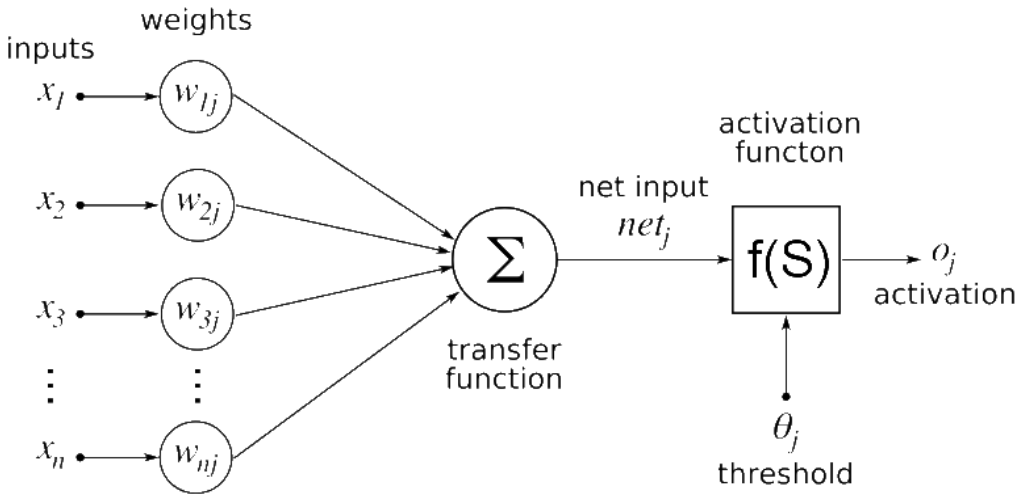

Figure 9 - Neuron - Credits Wikipedia Commons

The connections between neurons are represented by weights, which are automatically learned during the training process. The goal of training an ANN is to adjust these weights in such a way that the network can accurately map input data to the desired output.

The architecture of an ANN can be seen in Fig. 10 and described as follows:

- **Input Layer:** this layer receives the input data, which can be numerical values, images, or any other form of data representation.
- **Hidden Layers:** these layers perform the main computational work of the neural network. Each neuron in a hidden layer receives weighted inputs from the previous layer, applies an activation function, and propagates the resulting output to the next layer.
- **Output Layer:** produces the final output of the neural network, which can be a classification label, a numerical value, or any other desired output representation.

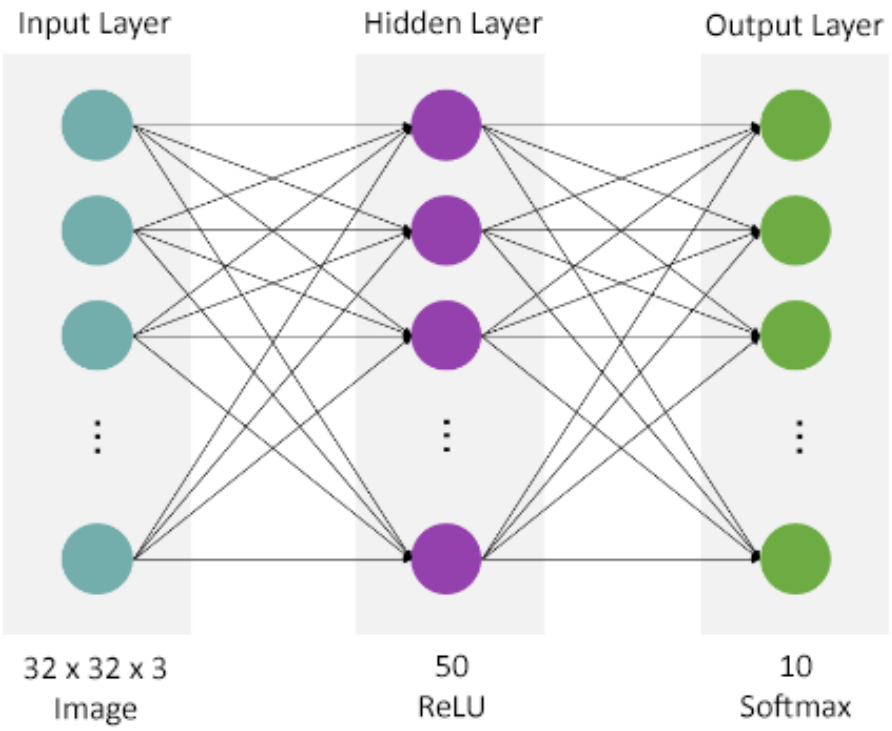

Figure 10 - ANN Layers

The power of neural networks lies in their ability to learn complex, non-linear relationships between the input data and the desired output by adjusting the weights of the connections between neurons during the training process. This training process is typically performed using an optimization algorithm, such as gradient descent or its variants, in conjunction with the backpropagation algorithm, which will be discussed later.

Neural networks can be composed of different types of layers and architectures, each designed to capture specific types of patterns or relationships in the data. For example, convolutional neural networks (CNNs) are specialized architectures particularly well-suited for processing image data, which will be discussed in section 1.2.2.

Artificial neural networks have proven to be powerful models for a wide range of tasks, including image recognition, natural language processing, speech recognition, and many others. Their ability to learn complex representations directly from data has been a driving force behind the success of deep learning in various domains.

Activation Function

Activation functions play a very important role in introducing non-linearity into artificial neural networks, allowing them to model complex, non-linear relationships within the data. Without activation functions, neural networks would be limited to representing linear functions, which would significantly restrict their ability to learn and generalize. [1]

In a neural network, each neuron receives weighted inputs from the previous layer, and these inputs are summed together. The activation function is then applied to this summation, introducing non-linearity and determining the output of the neuron. The choice of activation function can significantly impact the performance and convergence of the neural network during the training process.

Some commonly used activation functions include:

- **Sigmoid Function:** this function maps input values to the range (0, 1), making it suitable for binary classification tasks. (Fig. 11)
  It is defined as: $\sigma(x) = 1 / (1 + e\text{^}(-x))$
- **Rectified Linear Unit (ReLU):** the ReLU function is defined as: $f(x) = max(0, x)$ which means it outputs the input directly if it is positive, and 0 otherwise. ReLU functions are computationally efficient and have been shown to improve the training performance of deep neural networks, especially in computer vision tasks. (Fig. 11)
- **Softmax:** the softmax function is often used in the output layer of neural networks for multi-class classification tasks. It normalizes the output of the network into a probability distribution over the classes, ensuring that the sum of the output probabilities is equal to 1.

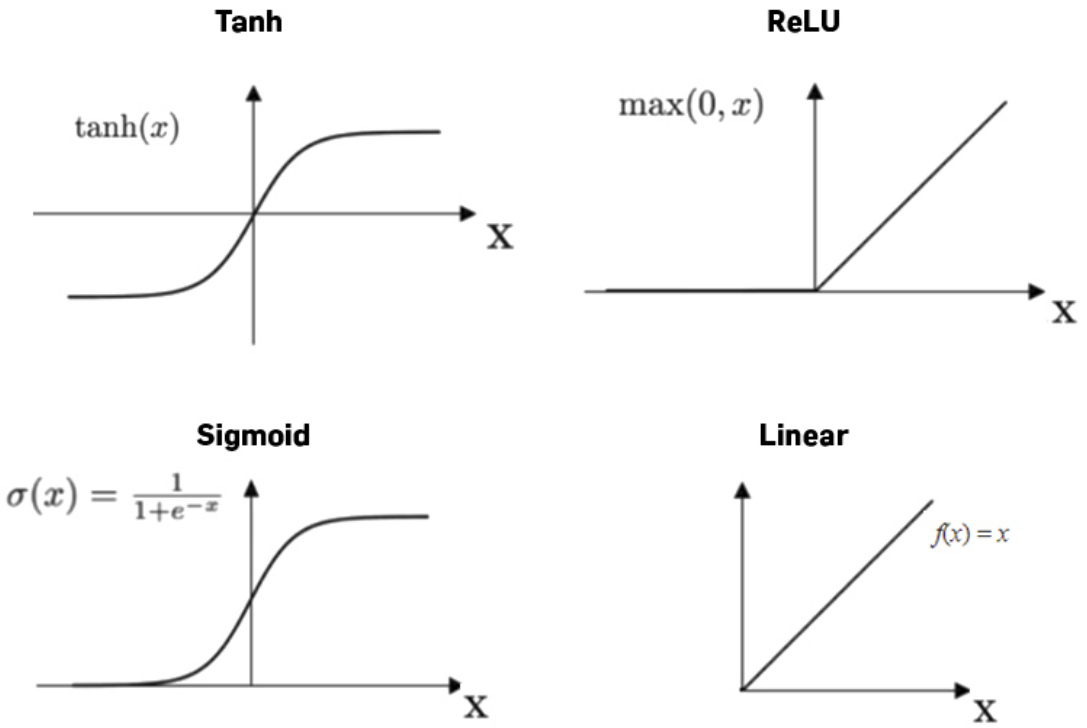

Figure 11 - Different Activation Functions
Credits: machine-learning.paperspace.com/wiki/activation-function

The choice of activation function depends on the specific task and the characteristics of the data. For example, the sigmoid and tanh functions are commonly used in the output layer for binary and multi-class classification tasks, respectively, while ReLU and its variants are widely used in the hidden layers of deep neural networks, especially in computer vision applications.

It is important to note that the nonlinearity introduced by activation functions is essential for neural networks to learn complex, non-linear relationships within the data. Without non-linearity, neural networks would be limited to learning linear functions, which would severely limit their expressive power and ability to generalize to unseen data.

The choice of activation function can significantly impact the overall performance of the neural network, and it is an active area of research in the field of deep learning.

Backpropagation Algorithm

The backpropagation algorithm is a fundamental technique used to train artificial neural networks. It is an optimization algorithm that adjusts the weights of the network based on the error between the predicted output and the true output, with the goal of minimizing this error. The backpropagation algorithm is a vital component of deep learning, as it enables the training of complex neural networks with multiple hidden layers, which would be intractable to train using other optimization techniques.

The backpropagation algorithm consists of two main phases:

1. **Forward Propagation:**
   - 1.1. Input data is fed into the input layer and propagated through the network.
   - 1.2. At each hidden layer, the inputs undergo an affine transformation (linear combination with weights and bias) followed by a nonlinear activation function (e.g. ReLU, sigmoid).
   - 1.3. The final layer computes the output using a suitable activation function (e.g. sigmoid for classification, linear for regression).
   - 1.4. The loss between the predicted output and the ground truth is calculated using a loss function (e.g. cross-entropy for classification, mean squared error for regression).

2. **Backward Propagation (Backpropagation Fig. 12):**
   - 2.1. The error signal (derivative of loss w.r.t output) is propagated backward through the network layers using the chain rule.
   - 2.2. At each layer, the gradients of the loss with respect to the weights are computed, quantifying the contribution of each weight to the error.
   - 2.3. The weights are updated using an optimization algorithm (e.g. stochastic gradient descent, Adam) to minimize the loss by moving in the opposite direction of the gradients.
   - 2.4. Regularization techniques (e.g. L1/L2 regularization, dropout) may be applied to prevent overfitting and improve generalization.

This forward and backward propagation process is repeated over multiple epochs, allowing the network to iteratively adjust its weights and learn to map inputs to desired outputs effectively.

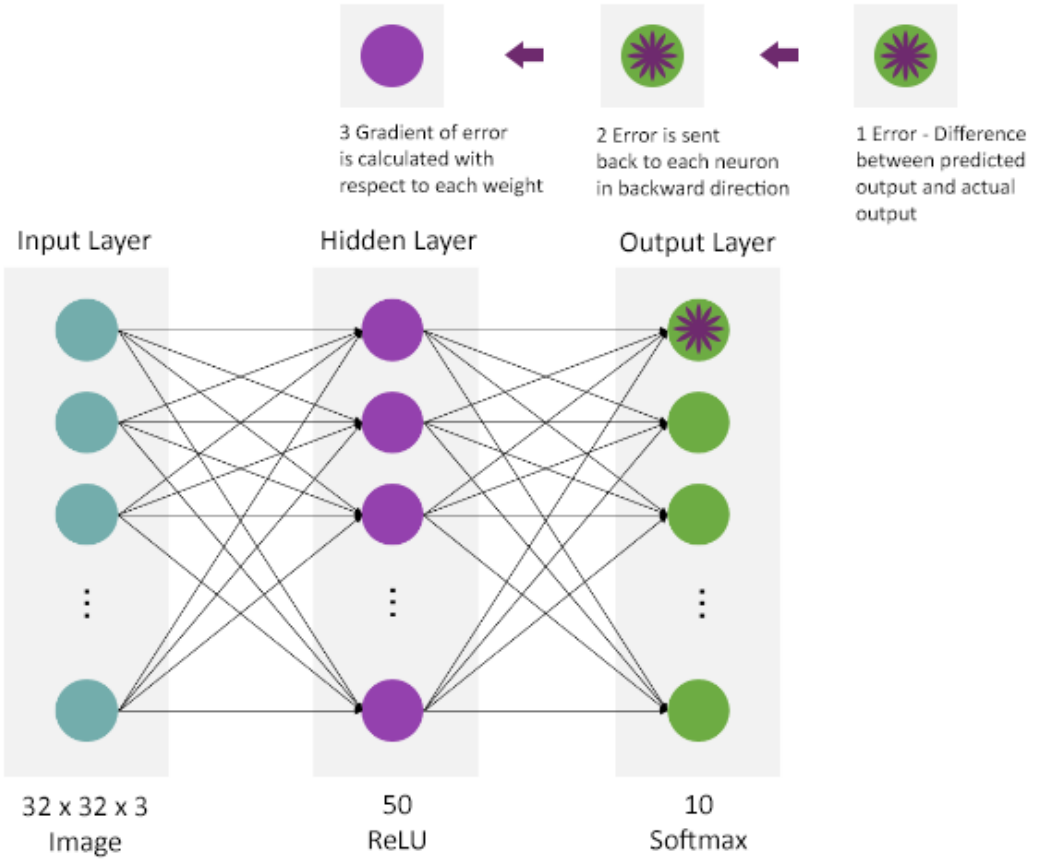

Figure 12 - Backpropagation algorithm

The weight update rule in the backpropagation algorithm is based on the principle of gradient descent, which adjusts the weights in the direction of the negative gradient of the loss function with respect to the weights. This ensures that the weights are updated in a way that reduces the error and improves the performance of the neural network.

The backpropagation algorithm can be summarized as follows:

1. **Initialize the weights** of the neural network with small random values.
2. Forward Pass: compute the predicted output using the current weights.
3. **Compute the error** between the predicted output and the true output.
4. Backward Pass: **propagate the error backward** through the network and **compute the gradients of the loss function** with respect to the weights.
5. **Update the weights** using an optimization algorithm (e.g. gradient descent) and the computed gradients.
6. Repeat steps 2-5 for x epochs until the error is minimized or a stopping criterion is met.

The backpropagation algorithm is an iterative process that gradually adjusts the weights of the neural network to minimize the error and improve its performance. It has been a key enabler for the successful training of deep neural networks, which have proven to be highly effective in various domains, including computer vision, natural language processing, and speech recognition. [1]

In practice, the backpropagation algorithm is often combined with techniques such as regularization, dropout, and batch normalization to improve the generalization performance of the neural network and prevent overfitting. Additionally, variations and optimizations of the backpropagation algorithm, such as the use of momentum or adaptive learning rate methods, have been developed to improve convergence and training efficiency.

2.2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [1] are a specialized type of neural network architecture that has been highly successful in various computer vision tasks, including image classification, object detection, and image segmentation. They are a powerful class of deep learning models that have revolutionized the field of computer vision and have achieved state-of-the-art performance on numerous challenging tasks.

CNNs are designed to take advantage of the spatial and local correlation present in image data, making them particularly well-suited for processing and learning from visual data. Unlike traditional fully connected neural networks, CNNs exploit the hierarchical pattern in data and assemble patterns to build increasingly complex representations of the input data.

The key advantages of CNNs lie in their ability to automatically learn rich and complex features directly from raw pixel data, without the need for manual feature engineering. This is achieved through a series of convolutional and pooling layers, which progressively extract higher-level features from the input data. These learned features are then fed into fully connected layers for classification or regression tasks.

CNNs have been instrumental in advancing fields such as autonomous vehicles, medical imaging, robotics, and many others, where the ability to accurately interpret and understand visual data is necessary.

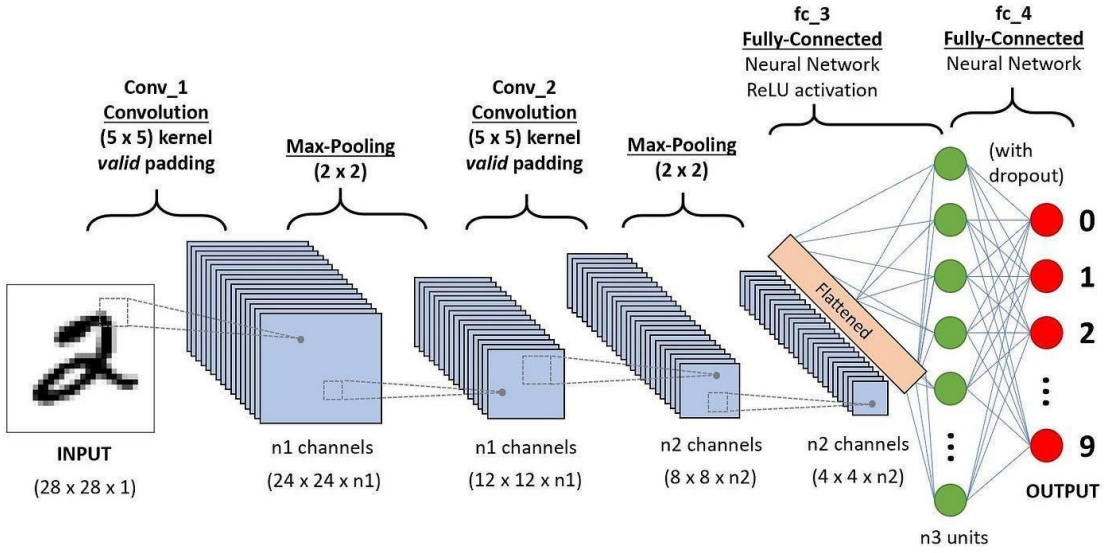

Figure 13 - CNN Example - Credits: Everton Gomede

The key components (Fig. 13) of a CNN architecture are:

1. **Convolutional Layers:** these layers apply a set of learnable filters (kernels) to the input image, performing a convolution operation that extracts local features and patterns from the image. The convolutional operation is spatially invariant, meaning it can detect the same feature regardless of its position in the image. Each filter produces a feature map, which represents the activation of that filter across the entire image.
2. **Pooling Layers:** these layers perform a downsampling operation, reducing the spatial dimensions of the feature maps while retaining the most relevant information. Common pooling operations include max pooling, which selects the maximum value within a local region, and average pooling. Pooling layers help to reduce the computational complexity and introduce spatial invariance, allowing the network to be robust to small translations or distortions in the input data.
3. **Fully Connected Layers:** after several convolutional and pooling layers, the CNN typically includes one or more fully connected layers, which are similar to the layers in a traditional neural network. These layers combine the learned features and perform the final classification or regression task.

As shown in Fig. 14 the architecture of a CNN can be summarized as follows:

1. Input Image
2. Convolutional Layer + Activation Function (e.g. ReLU)
3. Pooling Layer
4. Fully Connected Layer(s)
5. Output (e.g. classification or regression)

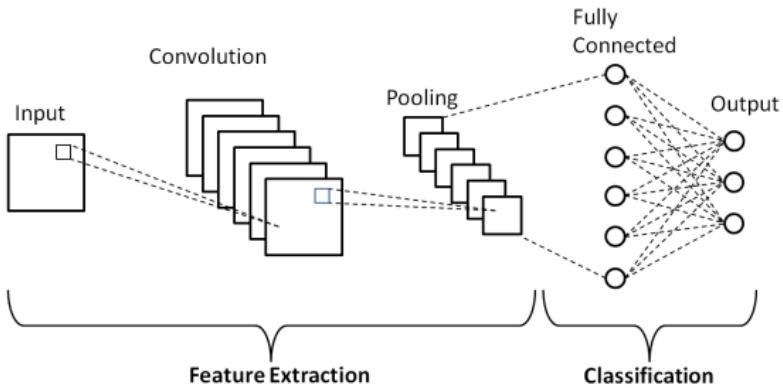

Figure 14 - Another example of CNNs - Credits Sonia Calvari

The strength of CNNs lies in their ability to learn hierarchical representations of the input data, starting from low-level features (e.g. edges, shapes) and progressively building up to higher-level, more abstract features (e.g. objects, patterns). This hierarchical learning process is facilitated by the alternating convolutional and pooling layers, which extract and condense the relevant features at different spatial resolutions.

CNNs have achieved remarkable success in various computer vision tasks, outperforming traditional machine learning techniques and setting new state-of-the-art benchmarks. They have been widely adopted in applications such as image classification, object detection, facial recognition, medical image analysis, and, most importantly for this thesis, image segmentation. [1]

In the context of image segmentation, CNNs are often used as the backbone architecture, with additional layers and specialized architectures designed to capture the spatial and contextual information necessary for accurate pixel-level segmentation. Many state-of-the-art image segmentation models, such as U-Net, Mask R-CNN, and DeepLabV3+, are based on CNN architectures and have demonstrated impressive performance in various domains, including the food image segmentation task addressed in this thesis.

2.3. Deep Learning Images Segmentation

Image segmentation [2] is a fundamental task in computer vision that aims to partition an image into multiple meaningful segments or objects. It serves as a required step in various applications, including object detection, scene understanding, medical image analysis, and autonomous systems. Traditional machine learning approaches for image segmentation often relied on hand-crafted features and algorithms, which could be time-consuming and ineffective in capturing complex patterns and relationships within the image data.

Deep learning models, such as CNNs, have demonstrated the remarkable ability to learn hierarchical representations and extract complex features directly from raw image data, enabling accurate and robust segmentation results. These models can automatically learn to identify and distinguish between different objects, textures, and patterns within an image, without the need for manual feature engineering.

Deep learning-based image segmentation approaches can be broadly categorized into two main types: semantic segmentation and instance segmentation.

Semantic Segmentation

As shown in Fig. 15 in semantic segmentation, the goal is to assign a class label to each pixel in the image, effectively partitioning the image into meaningful regions or objects. This approach is particularly useful in applications such as scene understanding, autonomous driving, and medical image analysis, where it is crucial to accurately identify and delineate different objects or structures within the image.

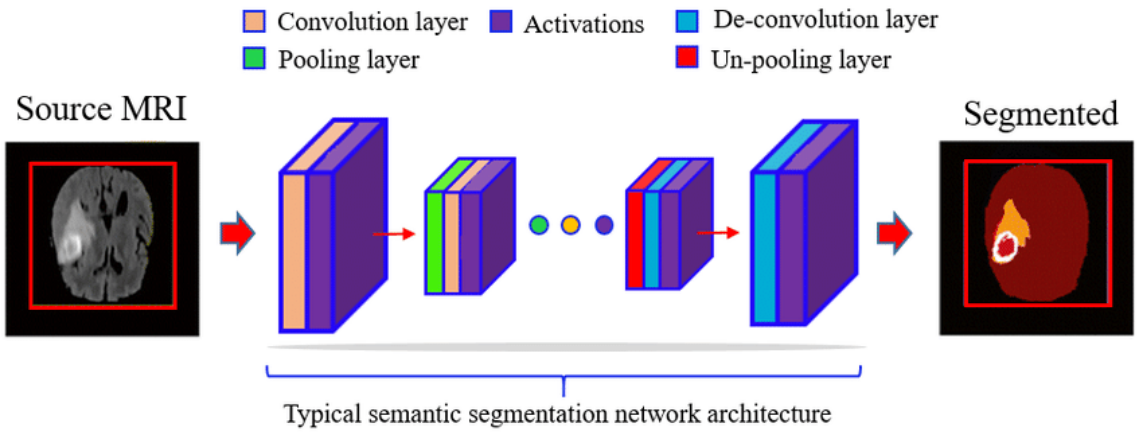

Figure 15 - Typical semantic segmentation network architecture - Credits Fouzia Altaf

Several deep learning architectures have been specifically designed and adapted for semantic segmentation tasks, including:

- **Fully Convolutional Networks (FCNs):** were among the first deep learning models proposed for semantic segmentation. They replace the fully connected layers in traditional CNNs with convolutional layers, allowing for dense predictions and preserving spatial information. This enables the model to produce pixel-wise predictions for the entire image, rather than a single classification output.
- **U-Net:** this architecture is particularly popular for biomedical image segmentation tasks. It follows an encoder-decoder structure, where the encoder part extracts features from the input image, and the decoder part upsamples and combines these features to produce precise segmentation maps. The U-Net architecture incorporates skip connections, which allow the model to capture and combine both low-level and high-level features, resulting in more accurate and detailed segmentation results.
- **DeepLabV3+:** is a powerful semantic segmentation model that employs atrous convolutions (dilated convolutions) to capture multi-scale information and incorporate context for accurate segmentation. It also incorporates encoder-decoder structures and spatial pyramid pooling to effectively segment objects at multiple scales, making it well-suited for tasks like scene segmentation and food image segmentation.

These semantic segmentation models often employ various techniques to improve their performance, such as skip connections, attention mechanisms, multi-scale feature aggregation, and post-processing methods like conditional random fields (CRFs) or fully connected CRFs (FCRFs) [2].

Instance Segmentation

As shown in Fig. 16 instance segmentation extends semantic segmentation by not only classifying each pixel into a specific class but also differentiating between individual instances of the same class. This is very important in object detection and tracking, where it is necessary to identify and segment each individual object instance accurately.

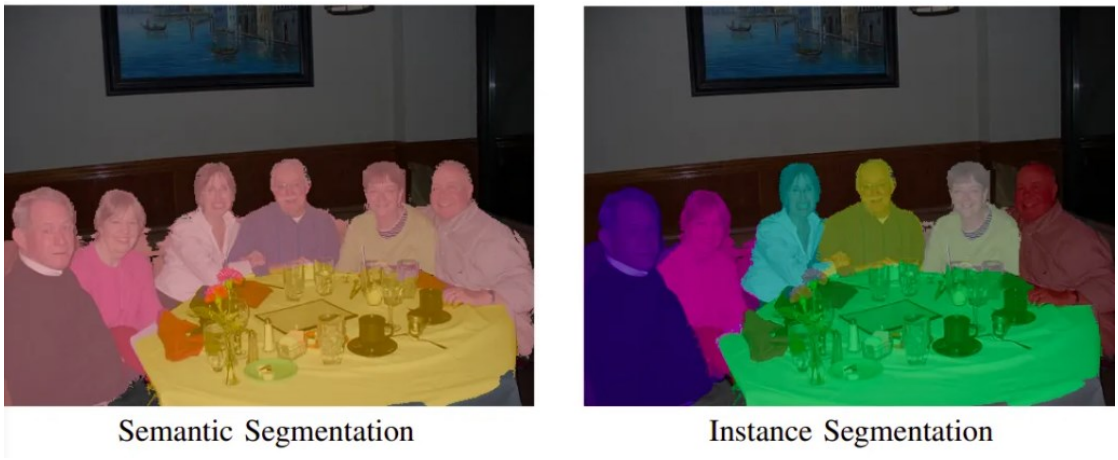

Figure 16 - Semantic vs Instance Segmentation - Credits Creative Commons Attribution 4.0

One of the most notable deep learning models for instance segmentation is Mask R-CNN. Mask R-CNN is a state-of-the-art model that extends the Faster R-CNN object detection architecture by adding a parallel branch for predicting segmentation masks for each detected object instance. This allows the model to not only detect and classify objects but also precisely delineate their boundaries and shapes within the image.

The success of deep learning in image segmentation can be attributed to the ability of these models to learn rich, hierarchical representations from raw image data, capture spatial and contextual information, and effectively leverage large-scale annotated datasets for training. These models have achieved impressive performance on various image segmentation benchmarks and have been widely adopted in numerous applications, ranging from autonomous vehicles and robotics to medical imaging and satellite imagery analysis. [2]

In the context of this thesis, deep learning techniques for image segmentation will be explored and applied to the specific task of food image segmentation. Advanced deep learning models and architectures tailored for image segmentation, such as DeepLabV3+ and U-Net, will be investigated and evaluated for their effectiveness in this domain.

3. STATE OF THE ART

The analysis of food images through techniques like segmentation and recognition is an area of growing interest and importance. However, this domain presents challenges due to the inherent complexities of food items. Some key challenges include high intra-class variability, occlusions, acquisition artifacts like varying illumination or blur, and the difficulty in accurately segmenting and recognizing individual ingredients within meal images.

This state of the art section aims to provide an overview of the latest research and advancements in the field of food image segmentation and recognition. It covers the development of specialized datasets tailored to address the challenges of food images, as well as the evolution of methodologies employed for tasks such as food classification, semantic segmentation, and nutritional value estimation.

By examining the existing literature, this section highlights the strengths and limitations of various approaches, datasets, and techniques used in food image analysis. It also identifies challenges and potential areas for further research and innovation.

Understanding the current state of the art is essential for advancing the field and developing more robust and accurate systems for food image understanding.

One significant advancement in food image segmentation involves the development of specialized datasets that resolve the challenges presented by food images. Ciocca et al. [3] introduced a dataset called UNIMIB2016 [3] specifically designed for this purpose, featuring images of canteen trays with multiple food items. Each item in these images was manually segmented with polygonal boundaries to provide accurate ground truth data for segmentation and classification algorithm development.

In their research, Ciocca et al. evaluated three distinct classification strategies to analyze the efficacy of different approaches in food image recognition. These strategies utilized a combination of global and local features extracted from the food images. As shown in Fig. 17 the global features approach involves analyzing the entire image or large regions of interest to capture overarching characteristics, while the local features strategy focuses on smaller, specific areas within the image to capture detailed textural and color information. The third approach combined these two methods, aiming to leverage both broad and detailed perspectives for improved classification accuracy.

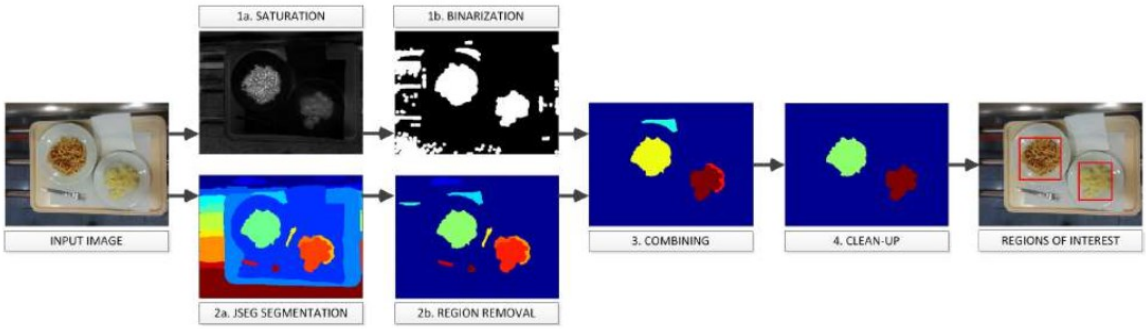

Figure 17 - Processing pipeline for the food segmentation, image from [3]

To facilitate these classification strategies, various visual descriptors were employed. The descriptors included traditional methods such as color histograms and texture features, as well as more sophisticated techniques like Convolutional Neural Networks. CNNs, in particular, have shown great promise in the field due to their ability to learn rich, hierarchical feature representations that are highly effective in complex image recognition tasks. In their study, Ciocca et al. achieved a classification accuracy of approximately 79% using CNN-based features, highlighting the potential of deep learning methods in the domain of food image analysis. [3]

Building on the foundation set by Ciocca et al. , S. Aslan and her team introduced a substantial enhancement to food image analysis through the development of a comprehensive dataset for the evaluation of food localization and semantic segmentation algorithms. In their paper, Aslan et al. [4] addressed the inherent complexities of food image segmentation which is compounded by the high intra-class variability (Fig. 18) and the presence of acquisition artifacts that can significantly distort the performance of segmentation algorithms.

Figure 18 - Top row: examples of high intra-class variability in visual appearance in food images. Bottom row: examples of low intra class variability, Image from [4]

The researchers constructed a new dataset starting with a base dataset called “Food50Chen” from Chen et al. [5] with 5,000 images across 50 food categories, each annotated with pixel-wise precision. Recognizing the need to assess algorithm performance under varied conditions, they expanded this dataset by artificially introducing common acquisition distortions such as changes in illumination, JPEG compression, gaussian noise, and gaussian blur. This augmentation brought the dataset to a total of 120,000 images. [4]

Using this dataset, Aslan and her team evaluated ten segmentation algorithms across two critical tasks: food localization and semantic food segmentation. Their findings not only shed light on the specific strengths and weaknesses of each algorithm but also set new benchmarks for future research in this area. The study highlighted the importance of robust dataset design and rigorous evaluation metrics in developing food analysis systems capable of operating effectively in real-world conditions.

This advancement underscores the evolving nature of machine learning applications in food analysis, building upon the earlier work by Ciocca et al. [3] to utilize both global and local features for improved classification accuracy. Aslan et al.'s work [4] demonstrates the role of specialized datasets in overcoming the challenges of food image segmentation.

Another research called “A Large-Scale Benchmark for Food Image Segmentation” by Wu et al. [6], published in May 2021, represents a significant contribution to the field of food image segmentation. It addresses two limitations: the lack of large-scale, high-quality datasets with fine-grained ingredient labels and pixel-wise masks, and the complex appearance of food items that makes it challenging to localize and recognize individual ingredients. The authors introduce FoodSeg103 [6] and its extension, FoodSeg154, with 9,490 images annotated with 154 ingredient classes and over 60,000 segmentation masks.

A notable strength of this work is the data collection and annotation process, which involved data selection, iterative refinement of labels and masks, and engagement with a professional data annotation company. Fig. 19 shows an example of this rigorous approach that ensures high-quality ground truth data, which is essential for training and evaluating segmentation models.

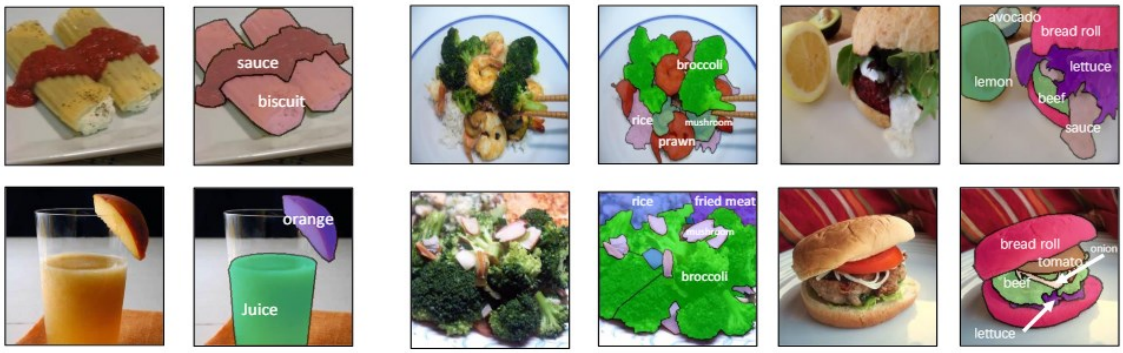

Figure 19 - Foodseg103 examples: source images (left) and annotations (right). Image from [6]

Furthermore, as shown in Fig. 20 the authors propose a multi-modality pre-training approach called ReLeM [6] (Recipe Learning Module) that incorporates recipe information in the form of language embeddings into the visual representation of food images. This technique aims to reduce the intra-variance of ingredients caused by different cooking methods, addressing this challenge in food image segmentation.

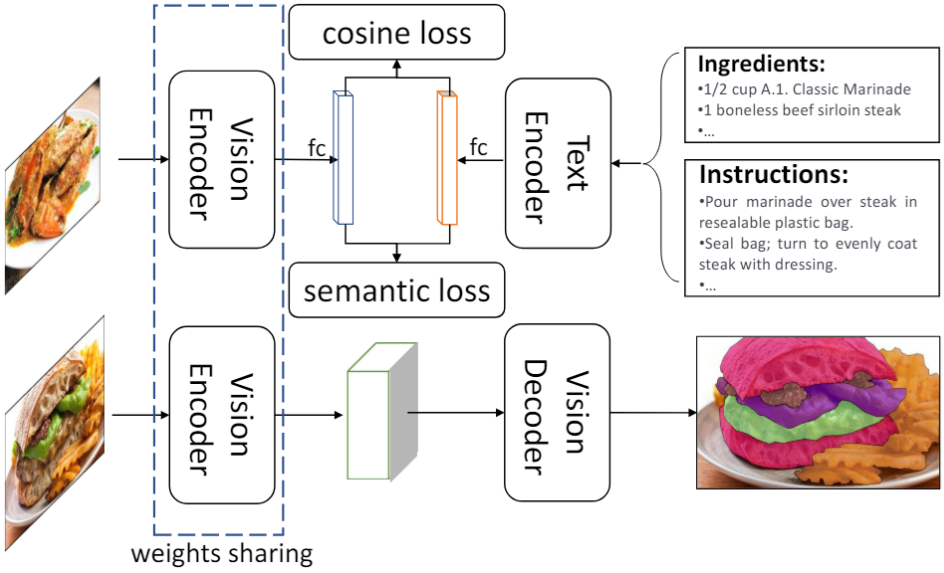

Figure 20 - Food image segmentation framework. Image from [6]

The experiments conducted by the authors, including evaluations on different semantic segmentation methods and cross-domain adaptability assessments, provide valuable insights into the challenges of food image segmentation and validate the effectiveness of the proposed ReLeM.

While the work presents advancements, it is essential to acknowledge some limitations. The dataset construction process used images from the Recipe1M dataset [7], which may introduce biases or limitations inherent to that dataset. Additionally, the computational complexity and resource requirements of the ReLeM approach might be challenges for real-world deployments, particularly on low-resource devices.

Overall, the paper by Wu et al. [6] represents a contribution to the field, providing a large-scale benchmark dataset and a multi-modality approach for enhancing

segmentation performance. Their findings and methodologies are useful for further research and advancements in food image understanding.

While the previous studies focused on specific aspects of food image analysis, such as classification and segmentation, Sultana et al. [8] provided an overview on automated estimation of nutritional values from food images. As shown in FIg. 21 the authors provide a taxonomic categorization of the reviewed studies into three major groups: food image classification, volume/weight estimation and nutrition estimation.

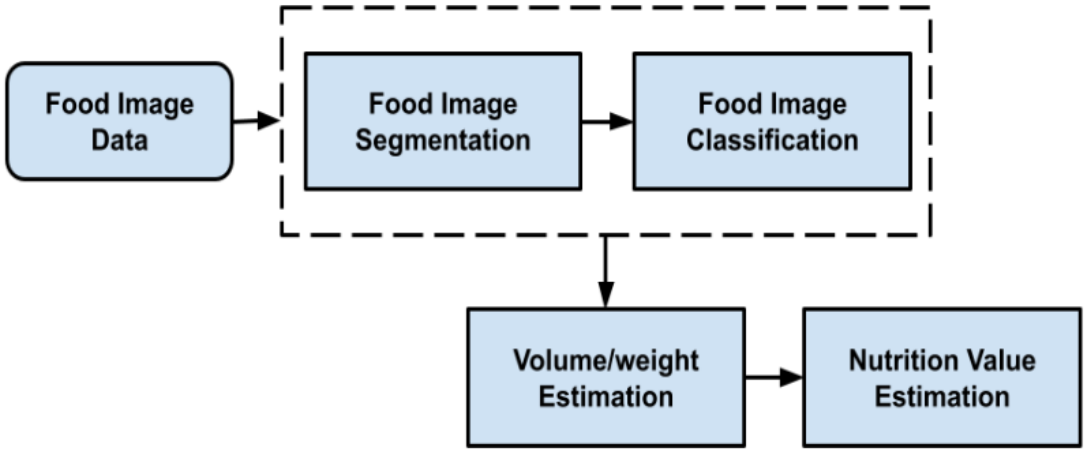

Figure 21 - A generalized framework for food nutrition estimation from food images. Image from [8]

A strength of this paper is the methodology employed, which involved a research across multiple databases published between 2011 and 2023, followed by a process to ensure relevance and quality. The author's critical analysis of the existing methodologies and datasets used in each stage of the pipeline offers insights for the current state-of-the-art [8].

The authors also highlight the transition from traditional Machine Learning techniques, like Support Vector Machines and K-Nearest Neighbors, which rely on handcrafted features like color and texture, to the more recent adoption of Deep Learning techniques, particularly Convolutional Neural Networks. The authors acknowledge the performance of Deep Learning methods in food classification tasks, as evidenced by the high accuracy achieved by various CNN architectures on benchmark datasets like Food-101 and UEC-Food100.

For volume and weight estimation, they cover diverse approaches, including the use of 3D food images, specific shape templates, multiple-view images and other deep learning techniques [8].

Some studies have placed standard accessories to correctly measure the approximate size of the foods from the food images. One example is shown in Fig. 22.

Figure 22 - Food images with reference cards. Image from [8]

The work critically analyzes the strengths and limitations of each method, highlighting the challenges associated with estimating volume from 2D food images without reference objects.

The authors also acknowledge several challenges and limitations in the field. Like difficulties in segmenting and classifying food items in images with multiple food classes, estimating volume and calorie for irregularly shaped foods, and the already known lack of large, high-quality food datasets.

For these challenges, the authors provide recommendations for future research, including the construction of large standard food datasets and the development of robust and generalized food recognition systems [8].

4. DATASETS AND TOOLS

This chapter outlines the dataset, tools, and methodologies used in this research to compare and evaluate food segmentation models. Descriptions include the dataset used, the preprocessing steps and the implementation of the deep learning models.

Tools such as PyTorch and specific model architectures like DeepLabV3+ are also discussed. The aim is to provide an understanding of the processes and technologies utilized to achieve the study's objectives.

4.1. Datasets

The choice of dataset for any deep learning application significantly determines and influences the performance of the model. In the context of this thesis on binary image segmentation for food images, the UEC-FoodPixComplete [9] dataset was chosen as the primary data source.

This decision is for several key factors:

1. **Segmentation Masks:** unlike many food image datasets that predominantly provide bounding boxes or labels without segmentation details, the UEC-FoodPixComplete offers manually refined segmentation masks. This attribute is needed for binary segmentation tasks where precise delineation between food and background is required.
2. **Variety:** the dataset has 10,000 images derived from the UECFood-100 dataset. It includes a diverse array of food items, providing a good spectrum of visual characteristics and segmentation challenges. This diversity is beneficial for training robust models capable of generalizing across various foods.

3. **Improvement:** this dataset is an enhancement of the earlier FoodPix, where segmentation masks were semi-automatically and contained inaccuracies. As shown in Fig. 23 the complete version resolves these inaccuracies through manual corrections, ensuring higher fidelity in training data which is important for better model performance.
4. **Related Research and Applications:** previous studies utilizing the UEC-FoodPixComplete have demonstrated its utility in applications beyond segmentation, such as calorie estimation and food image synthesis, indicating its versatility and efficacy. [9]

Figure 23 - The differences on segmentation masks in both datasets.
(1st column: food images image, 2nd column: segmentation masks in the original UECFoodPix, 3rd column: segmentation masks in the renewed UEC-FoodPixComplete.) Image from [9]

4.1.1. Image Preprocessing

The preprocessing is important for enhancing the quality of input data and improving the performance of the segmentation models.

The preprocessing steps applied to the images in this study include:

1. **Resizing:** Images were resized to 512x512 pixels. This standardization reduces computational complexity and ensures consistent dimensions across the dataset.

2. **Normalization:** Pixel values were normalized to match the requirements of the pretrained weights from ResNet34, using a specific preprocessing function from the SMP library.

   ResNet34 is a deep convolutional neural network architecture that is 34 layers deep. It is known for its use of residual learning to ease the training of deep networks by allowing gradients to flow through shortcut connections directly, bypassing one or more layers. This architecture was introduced by He et al. [10]

   ImageNet is a large visual database designed for use in visual object recognition research. More than 14 million images have been hand-annotated with information about the objects contained in each image. The database has been used to train and benchmark various image recognition models. [11]

   Normalization scales the pixel values to a range that is more suitable for training neural networks. This process improves convergence and stability during training, because the network can learn more from data that is on a similar scale.

The preprocessing pipeline ensures that the dataset is well-prepared for training, facilitating the development of accurate food segmentation models. By standardizing the image dimensions and normalizing the pixel values, the preprocessing enhances the quality of the input data, for an improved model performance and generalization.

4.2. Tools and Libraries

This section explores the key tools and libraries utilized in this study for developing and training deep learning models for image segmentation. The focus will be on PyTorch [12], which is a widely used machine learning library, and the “Segmentation Models library [13]” which extends PyTorch's capabilities by providing pre-trained models and utilities designed for image segmentation tasks. Understanding these tools is essential for comprehending the methodology and implementation details of the models developed in this study.

4.2.1. PyTorch

PyTorch [12] is an open-source machine learning library developed by Facebook's AI Research lab. It is used for a variety of tasks, including computer vision, natural language processing, and other deep learning applications.

PyTorch provides tensor computation similar to NumPy with strong support for GPU acceleration, enhancing computational efficiency. Unlike static graph frameworks, it uses dynamic computational graphs, allowing developers to modify the graph on-the-fly. The library includes an automatic differentiation system called Autograd, which facilitates gradient computation essential for training neural networks through backpropagation.

TorchScript, a subset of PyTorch, enables the transition between execution mode and graph mode, making it easier to optimize and deploy models. PyTorch also supports distributed training, allowing for efficient model training across multiple GPUs and machines, which is important for handling large datasets and complex models. [12]

These features make PyTorch a powerful tool for developing state-of-the-art machine learning models.

4.2.2. Segmentation Models Library

The Segmentation Models library [13] provides an interface for creating and training segmentation models. This library simplifies the process of developing complex image segmentation models by offering pre-implemented architectures and utilities.

Some features are:

- **Pre-trained Models:** includes various pre-trained models such as U-Net, FPN, DeepLabV3, and DeepLabV3+. These models can be fine-tuned for specific tasks, using pre-trained weights to improve convergence and accuracy.
- **Customizability:** as shown in Fig. 24 users can customize model architectures by choosing different backbones, activation functions, and other parameters.
- **Integration with PyTorch:** this library integrates seamlessly with PyTorch, utilizing its tensor computation and GPU acceleration capabilities.
- **Efficient Training:** the library provides utilities for data loading and preprocessing which improve the training process. These tools help prepare the data in a way that maximizes the performance of the models.
- **Encoder Variety:** it supports a wide range of encoders with pre-trained weights from various sources such as ImageNet.

```
model = smp.DeepLabV3Plus(
    encoder_name='resnet34',
    encoder_depth=5,
    encoder_weights='imagenet',
    encoder_output_stride=16,
    decoder_channels=256,
    decoder_atrous_rates=(12, 24, 36),
    in_channels=3,
    classes=1,
    upsampling=4,
    activation='sigmoid',
    aux_params=None
)
```

Figure 24 - Customizability of smp Library. Image from Google Colab

For these reasons this segmentation models library was chosen and is a valuable tool for researchers working on image segmentation tasks.

4.3. Analysis of Segmentation Algorithms and Models

This section presents an analysis of various deep learning models used for image segmentation, with a particular focus on their application to food images. Using the segmentation_models library, a series of experiments were conducted to evaluate the performance of several segmentation algorithms.

The analysis includes an overview of each model's architecture and the methodological approaches that differentiate them, providing insights into how they handle the complexities of image segmentation.

This comparison offers an overview of the different image segmentation models and offers insights into their performance.

4.3.1. U-NET and U-NET++

U-NET [14] was primarily designed for medical image segmentation. As shown in Fig. 25 its architecture is characterized by the symmetry, consisting of a contracting path to capture context and a symmetric expanding path that enables precise localization. The contracting path is a typical convolutional network that consists of repeated application of convolutions, each followed by a rectified linear unit (ReLU) and a max pooling operation for down-sampling. At each down-sampling step, the number of feature channels is doubled. In the expanding path, feature maps are up-sampled followed by a convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two more convolutions and ReLU operations.

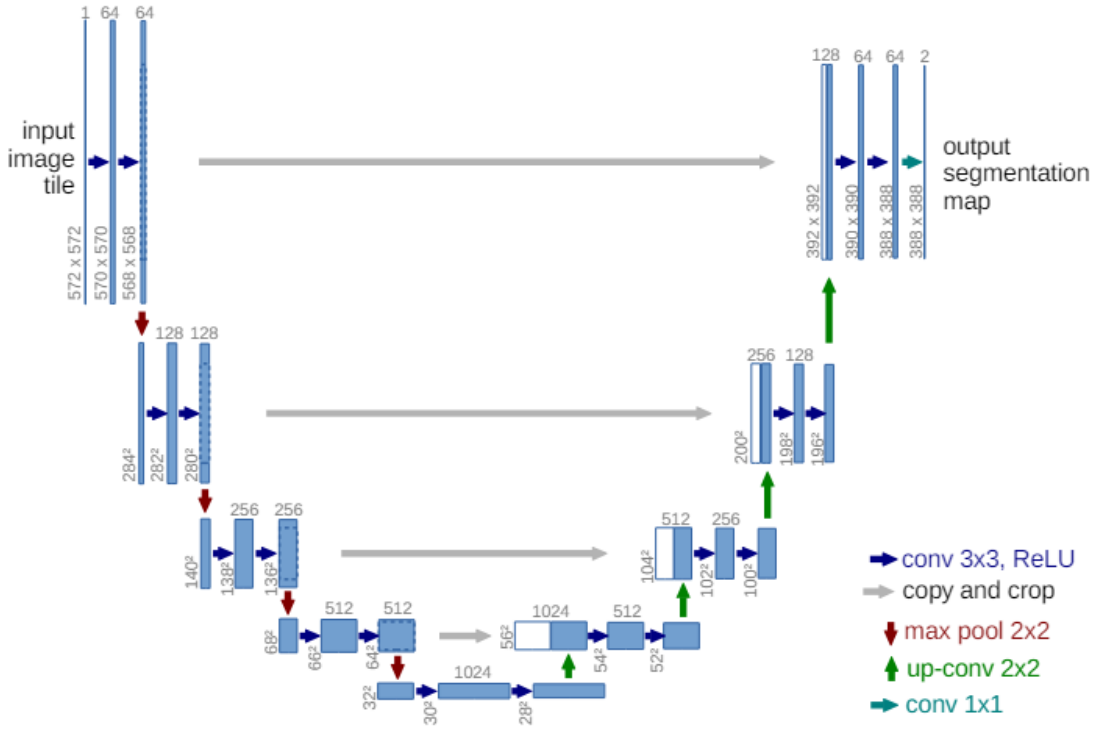

Figure 25 - U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. Image from [14]

As shown in Fig. 26 U-NET++ [15] enhances the U-NET architecture by introducing dense skip pathways that offer improvements over the standard U-net structures for capturing fine-grained details more effectively. These modifications address the need for more precise segmentation in applications requiring detailed contextual recognition, such as segmenting individual food items from a complex background. U-Net++ reduces the semantic gap between the feature maps of the encoder and decoder sub-networks through a series of dense skip pathways and deep supervision.

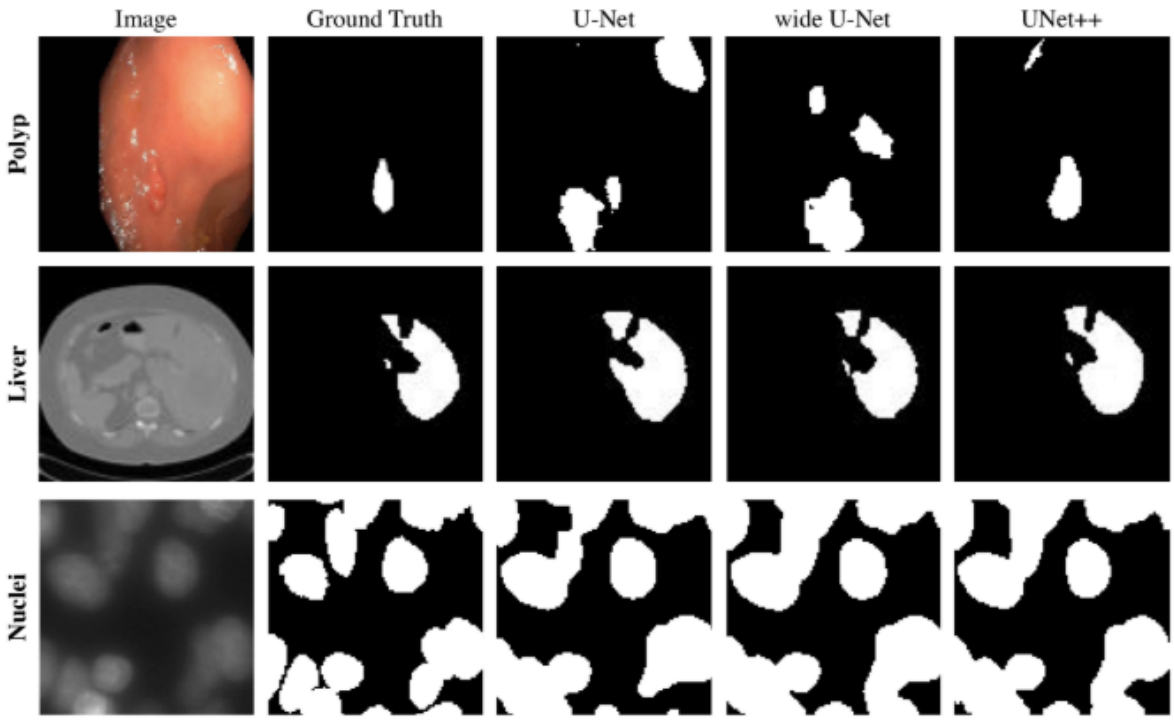

Figure 26 - Qualitative comparison between U-Net, wide U-Net, and UNet++, showing segmentation results for polyp, liver, and cell nuclei datasets. Image from [15]

4.3.2. FPN (Feature Pyramid Network)

Feature Pyramid Network FPN [16] introduces a top-down architecture with lateral connections for building semantic feature maps at all scales. (See Fig. 27)

This model improves segmentation by applying high-level semantic features across the different scales through a pyramid of predictions, which is effective in identifying objects at different scales, this is an essential feature for segmenting food images which can vary in size and shape in the same scene.

Strengths:

- Good at handling multi-scale object detection.
- Enhances feature representation across different scales, improving segmentation accuracy.

Weaknesses:

- Can be computationally intensive due to the pyramid structure.
- Complexity in implementation and tuning compared to simpler models.

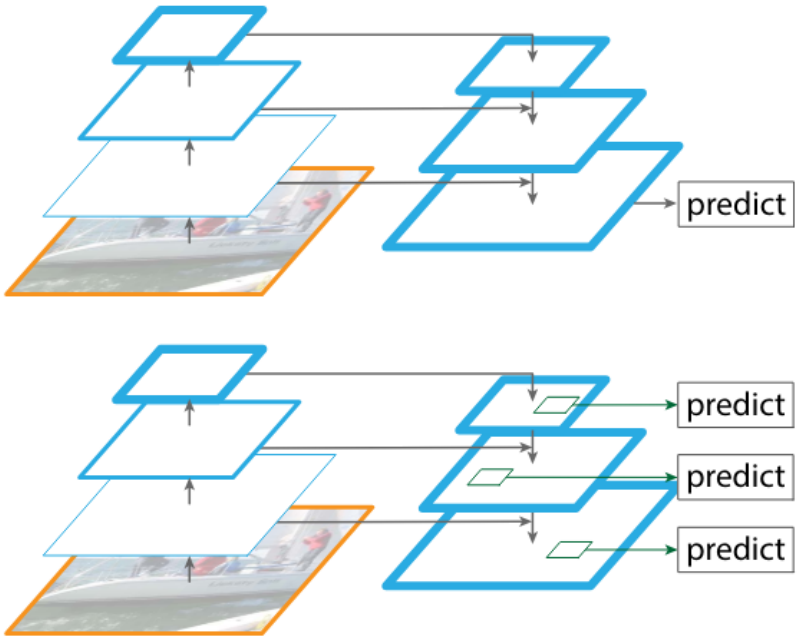

Figure 27 - Top: a top-down architecture with skip connections, where predictions are made on the finest level. Bottom: FPN model that has a similar structure but leverages it as a feature pyramid, with predictions made independently at all levels. Image from [16]

4.3.3. PSPNet (Pyramid Scene Parsing Network)

PSPNet [17] resolved some complex scene parsing challenges by implementing a pyramid pooling module to gather different sub-region representations, followed by up-sampling and concatenation layers. (see Fig. 28)

This architecture captures global context and precise local cues without requiring excessively large inputs. It harnesses the capability of global context information by different-region-based context aggregation, which is important for complex images where different food and background details need to be distinguished accurately.

Strengths:

- Captures both local and global context.
- Good in complex scenes where distinguishing between fine details and broader context is crucial.

Weaknesses:

- High computational demands due to extensive pooling operations.
- May require significant memory resources for training and inference.

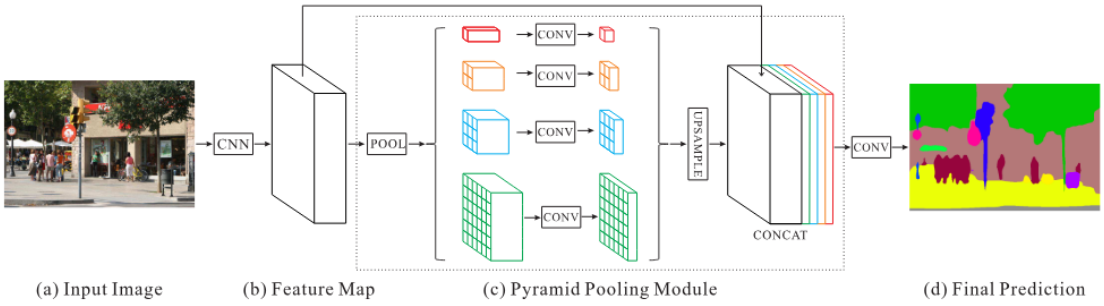

Figure 28 - Overview of PSPNet. “Given an input image (a), we first use CNN to get the feature map of the last convolutional layer (b), then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information in (c). Finally, the representation is fed into a convolution layer to get the final per-pixel prediction (d)”. Image from [17]

4.3.4. DeepLabV3 and DeepLabV3+

DeepLabV3 [18] integrates **atrous convolutions** (also known as dilated convolutions, they expand the receptive field of the convolutional filters without increasing the number of parameters) to capture multi-scale context by adopting multiple atrous rates, including an atrous spatial pyramid pooling (ASPP) module to examine convolutional features with filters at multiple sampling rates. (See Fig. 29)

As shown in Fig. 30 these features enable the model to capture sharp object boundaries and textural features that are important for accurately segmenting food items.

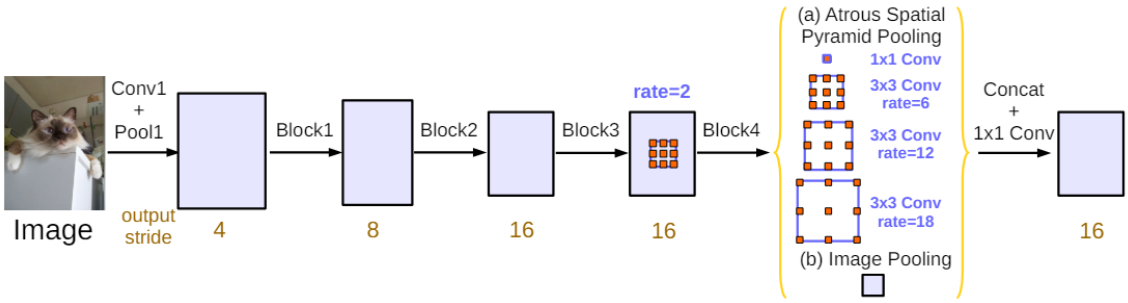

Figure 29 - Parallel modules with atrous convolution (ASPP), augmented with image-level features. Image from [18]

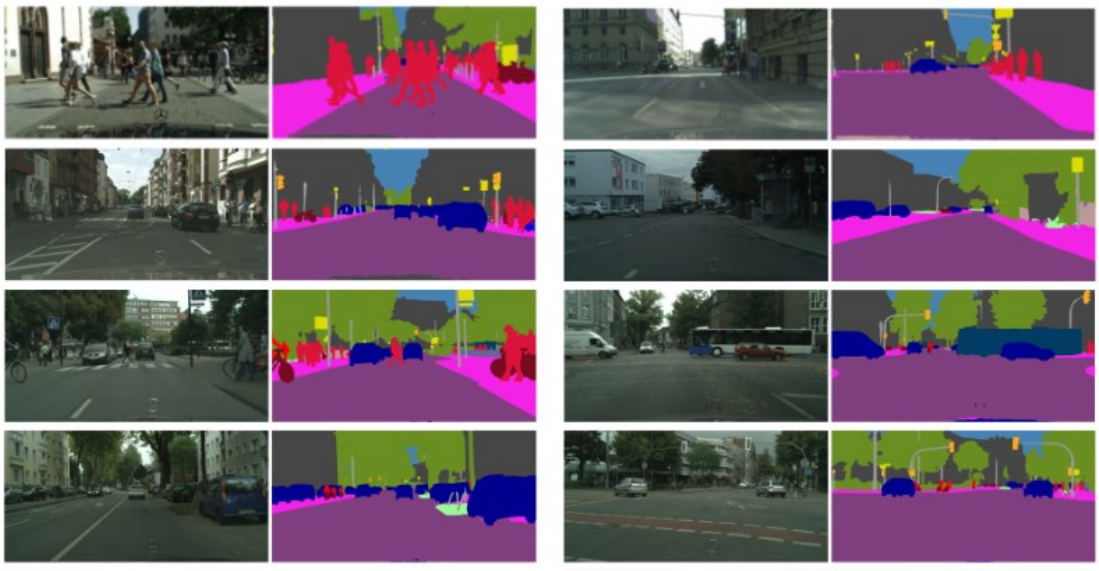

Figure 30 - Visualization results of DeepLabV3. Image from [18]

DeepLabV3+ [19] extends DeepLabV3 by adding an encoder-decoder structure (see Fig. 31) that refines segmentation results, especially with object boundaries. The decoder uses features from earlier layers to enhance recovery of object boundaries, particularly important for segmenting detailed objects such as foods. This adjustment significantly improves the sharpness of the segmentation maps and is effective with multiple overlapping items.

The encoder in DeepLabV3+ continues to employ atrous convolutions to capture multi-scale contextual information, as seen for DeepLabV3.
The real advancement here is the decoder module, it takes advantage of features extracted from earlier layers of the encoder, which contain higher resolution information crucial for precise boundary recovery. By fusing these high-resolution features with context-rich features, the decoder improves the sharpness and clarity of the segmentation.

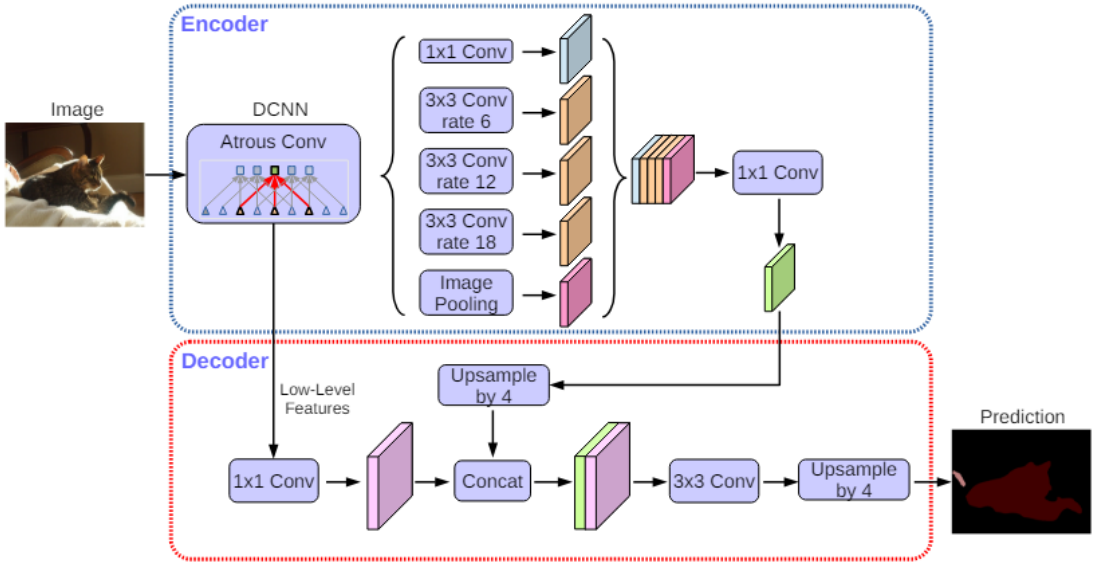

Figure 31 - The proposed DeepLabv3+ extends DeepLabv3 by employing a encoder decoder structure. The encoder module encodes multi-scale contextual information by applying atrous convolution at multiple scales, while the simple yet effective decoder module refines the segmentation results along object boundaries. Image from [19]

4.4. Model Configuration

This section details the configuration of the deep learning models used for comparison in this study. All the models were configured and tested using the defaults parameters in SMP documentation [13].

Model Definition

As shown in Fig. 32 the model used in this example is DeepLabV3Plus:

```python
# Model Definition
model = smp.DeepLabV3Plus(
    encoder_name='resnet34',
    encoder_depth=5,
    encoder_weights='imagenet',
    encoder_output_stride=16,
    decoder_channels=256,
    decoder_atrous_rates=(12, 24, 36),
    in_channels=3,
    classes=1,
    upsampling=4,
    activation='sigmoid',
    aux_params=None
)

# Define loss function, and optimizer
loss = smp.losses.DiceLoss(mode="binary")

# Optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Figure 32 - Code: DeepLabV3Plus configuration.

Here is a short description of this model's parameters:

- **Encoder Name:** utilizes the ResNet34 architecture for the encoder.
- **Encoder Depth:** sets the depth of the encoder, this typically means the encoder has 5 stages of convolutional layers. This depth allows for capturing hierarchical features from the input image, ranging from low-level edges to high-level object parts.
- **Encoder Weights:** uses weights pretrained on the ImageNet dataset.
- **Encoder Output Stride:** controls the downsampling factor of the encoder output. An output stride of 16 balances between reducing the spatial resolution and retaining enough detail for accurate segmentation, which is a common choice in segmentation tasks.
- **Decoder Channels:** specifies the number of channels in the decoder. Setting the number of channels to 256 allows the decoder to handle a sufficient amount of feature maps, enabling it to reconstruct spatial details effectively.
- **Decoder Atrous Rates:** defines the atrous (or dilated) convolution rates used in the decoder. These rates allow the decoder to capture multi-scale context by enlarging the receptive field without losing spatial resolution, which is particularly useful for semantic segmentation.
- **In Channels:** number of input channels, which indicates that the input images have 3 channels (RGB).
- **Classes:** specifies the number of classes in the output segmentation mask. In this case, it means the segmentation task is binary, with one class of interest (foreground) and the background.
- **Upsampling:** defines the upsampling factor to obtain the final output resolution. An upsampling factor of 4 is used to scale the decoder output back to the original input resolution, ensuring the final segmentation map aligns with the input image size.
- **Activation:** sets the activation function for the output layer, suitable for binary classification tasks. The sigmoid activation function outputs values between 0 and 1, representing the probability of a pixel belonging to the foreground class.
- **Aux Params:** indicates that no auxiliary output is used, meaning the model focuses on the main segmentation task without additional auxiliary tasks.

Loss Function

The loss function chosen is the Dice Loss, which is particularly effective for segmentation tasks. Here the goal is to maximize overlap between the predicted and true masks.

Mode: "binary" indicates that the dice loss is used for binary classification.

Optimizer

The Adam optimizer is used to train the model with a learning rate of 0.001.

The learning rate (lr) determines the step size at each iteration while moving towards the minimum of the loss function. It controls how much to change the model parameters with respect to the gradient of the loss function.
A learning rate of 0.001 is a commonly used starting point for Adam, it provides a good balance between convergence speed and stability.

Early Stopping

Early stopping is used to prevent overfitting and to ensure that the model generalizes well to unseen data. By monitoring the validation loss during training, early stopping stops the training process once the validation loss stops improving for a predefined number of epochs (patience). This approach helps in avoiding the scenario where the model learns noise and irrelevant details from the training data, which can degrade its performance on new data. The patience parameter is set to 3 epochs, meaning if the validation loss does not improve for 3 consecutive epochs, the training will be stopped. This ensures that the model is saved at the point where it performs best on the validation set.

These configurations ensure that the model is well-prepared for the segmentation task, leveraging the pretrained capabilities of ResNet34 while being fine-tuned on the specific dataset used in this study.

4.5. Methodology

For the experimental results, the following methodology was employed:

1. **Data Preparation:** all images from the dataset were pre-processed to meet the input requirements of the neural network models. This involved resizing images, normalizing pixel values, and converting segmentation masks into a binary format (foreground-background).
2. **Model Selection:** multiple deep learning models, specifically those proficient in semantic segmentation tasks, were selected for evaluation. These models are well-regarded for their efficiency and effectiveness in handling complex image segmentation tasks.
3. **Training:** each model was trained independently on the dataset for 15 epochs with an early stopping based on the F1 score, using a patience of 3.
4. **Evaluation:** these 6 models were evaluated using standard segmentation metrics. Comparisons were made to determine which model best handled the binary segmentation of food images.
5. **Analysis of Results:** the best-performing model was selected, and the results were discussed.

5. EXPERIMENTAL RESULTS

This chapter presents the experimental results of the study. The training and testing process are detailed, including the relative code.

Following this, the performances of the models are analyzed through the presentation of various metrics and visual results. Finally, a comparison of different models is conducted to determine which approach yields the best results in segmenting food images.

5.1. Dataset Description

This section provides a description of the datasets used for training and testing, including images and ground truth masks. Understanding the datasets and the expected outcomes is essential for evaluating the performance of the models.

5.1.1. Dataset Splitting

The dataset used in this study was split into training, validation, and test sets to ensure an accurate evaluation of the models performance.

- **Training Set:** 7200 images (70%) were used to train the models. This set includes a wide variety of food images that allow the models to learn the features and variabilities of different foods.
- **Validation Set:** 1800 images (20%) were reserved for validation during the training process. This set is used to monitor the model performance and prevent overfitting.
- **Test Set:** 1000 images (10%) were used to test the final performance of the trained models. This set allows the evaluation of the model on unseen data.

5.1.2. Data Sources

As mentioned previously in Section 4.1, the dataset utilized in this study is the UEC FoodPixComplete, which includes 10,000 images with 102 different food types. (Some examples in Fig. 33)

Figure 33 - Images from the UEC dataset (left: sliced bread, middle: sandwich, right: hamburger)

5.1.3. Ground Truth Mask

The ground truth masks in the UEC-FoodPixComplete dataset are manually annotated to ensure high accuracy. These masks serve as the benchmark for evaluating the models performance, providing a clear reference for the expected segmentation of food items. (See Fig. 34)

Figure 34 - Mask from the UEC dataset (left: sliced bread, middle: sandwich, right: hamburger)

5.1.4. Model Input and Outputs

The images are of different sizes and in RGB format, meaning they have 3 color channels, capturing the color spectrum necessary for food recognition. Before the images are given to the models, they are resized to 512×512 pixels, this ensures that all images have a uniform size, which is necessary for batch processing. The resized images are then converted to tensors, this conversion is essential because PyTorch models require input data in tensor format. Finally, as shown in Fig. 35 the images are preprocessed to match the distribution of the dataset used to pretrain ResNet encoder.

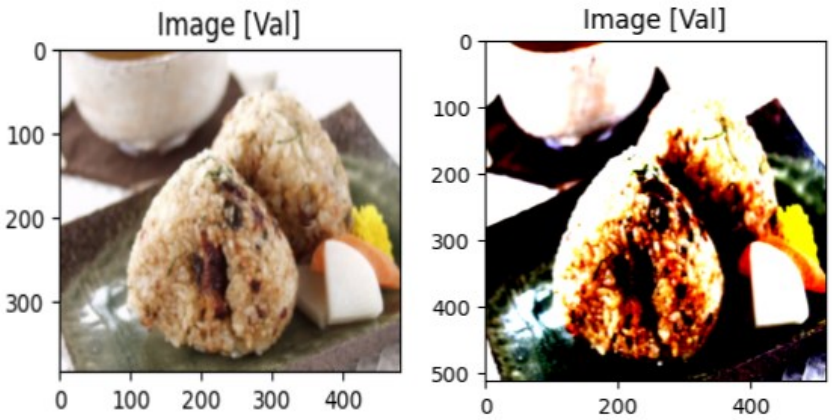

Figure 35 - Left: original dataset image. Right: preprocessed and normalized.

In addition to the images, the corresponding segmentation masks are provided as input. These masks are needed for training the models to learn the boundaries of food items, as shown in Fig. 36 they are preprocessed to extract the red channel and converted into binary format, where each pixel is food (1) or background (0).

```python
def load_mask(self, mask_path):
        mask = Image.open(mask_path).convert("RGB")
        red_channel = mask.split()[0]  # Estract red channel
        binary_mask = np.where(np.array(red_channel) > 0, 1, 0)
        binary_mask_image = Image.fromarray((binary_mask *
        255).astype(np.uint8), mode="L")  # Convert to binary

        return binary_mask_image
```

Figure 36 - Code: mask loader.

The output produced by the models is a segmentation mask for each input image. This segmentation mask is a binary map where each pixel is classified as either food (1) or background (0). The mask has the same spatial resolution as the input image, 512×512 pixels, ensuring that the segmentation aligns perfectly with the original image dimensions. (Fig. 37)

Figure 37 - Middle: predicted mask generated by the model.

To convert the model's predicted probabilities in binary, a threshold step is applied (see Fig. 38). If the probability of a pixel being food is greater than 0.5 the pixel is classified as food (1), otherwise, it is classified as background (0). This step ensures that the output mask provides a clear segmentation of food items in the image.

```
# Compute predicted labels
predicted_labels = torch.argmax(outputs, dim=1)

# Treshold predicted
predicted_labels = torch.where(outputs > 0.5, torch.tensor(1), torch.tensor(0))
```

Figure 38 - Middle: predicted mask generated by the model.

To facilitate the process of training, validation and testing, the dataset is loaded into dataloaders, they are used for batching the data, shuffling it if necessary, and loading it on the appropriate device (CPU or GPU). As shown in Fig. 39 the training dataloader is created by initializing the CustomDatasetUEC class with images_names_file, image directory, and mask directory. This dataloader is configured with a batch size of 8 and shuffles the data at each epoch to ensure that the model does not learn the order of the images.

```
# Create train dataset/dataloader
dataset = CustomDatasetUEC(image_names_file_7200, image_dir,
mask_dir)
data_loader = torch.utils.data.DataLoader(dataset,
batch_size=8, shuffle=True)

# Create test dataset/dataloader
test_dataset = CustomDatasetUEC(test_image_names_file,
test_image_dir, test_mask_dir)
test_data_loader = torch.utils.data.DataLoader(test_dataset,
batch_size=8, shuffle=False)

# Create val dataset/dataloader
val_dataset = CustomDatasetUEC(val_image_names_file_1800,
val_image_dir, val_mask_dir)
val_data_loader = torch.utils.data.DataLoader(val_dataset,
batch_size=8, shuffle=False)

# Access the data
image, mask = dataset[1]
test_image, test_mask = test_dataset[1]
val_image, val_mask = val_dataset[1]
```

Figure 39 - Code: dataloaders.

Similarly, the test and validation dataloaders are created using their respective dataset files, image directories, and mask directories. These dataloaders are also configured with a batch size of 8 but do not shuffle the data, because shuffling is unnecessary for evaluation purposes.

This setup allows for seamless and effective training, validation, and testing of the models used in this study for food segmentation.

5.2. Model Training

This section outlines the training process for the deep learning models used in the study. The training phase is crucial as it determines the effectiveness and accuracy of the models in performing food image segmentation.

The models were trained using the following parameters:

- Pre-processing: Resnet34
- Loss Function: Dice Loss
- Optimizer: Adam with Learning Rate: 0.001
- Early Stopping: F1 Score, with a patience parameter set to 3 epochs.
- Number of Epochs: 15

The training was conducted on Google Colab Pro, which provides a Jupyter Notebook environment with Python 3.8. And the NVIDIA L4 Tensor Core GPU with 22.5GB of RAM, 63GB of System RAM and 200GB of Disk space.

5.2.1. Training Procedure

The training procedure involves some steps to ensure learning and accurate segmentation of food images. During each epoch batches of images and corresponding masks are processed and transferred to the GPU to accelerate computation.

As shown in Fig. 33 other steps running for each epoch are:

1. Gradients are reset and a forward pass is conducted to obtain the outputs.
2. Masks are rounded and converted to long tensors.
3. Loss is calculated and accumulated for the single epoch.
4. Backpropagation updates the model parameters.
5. Batch loss is periodically printed.

At the end of each train epoch, as shown in Fig. 34 performance metrics are computed to monitor the model. Predicted labels are obtained by applying a 0.5 threshold to the outputs. Accuracy, precision, recall, F1 score, and Jaccard Score are calculated based on the true and predicted labels, with these metrics rounded and printed for each epoch. The average loss for the epoch is also calculated and stored.

After computing the performance metrics on the first epoch of the train set, as shown in Fig. 35 it is important to set the model to evaluation mode. This ensures that the model behaves correctly during validation by disabling certain features like dropout and batch normalization, which are only used during training.

1. Validation batches are processed similarly to training batches.
2. Validation loss is calculated and accumulated.
3. Predicted labels are thresholded and validation metrics computed.
4. Average validation loss and metrics are stored.
5. Early stopping is triggered if there is no improvement in the F1 score for 3 consecutive epochs.

Visualization of the training progress includes plotting training and validation losses across epochs, an overfitting line is drawn to indicate the point at which validation loss starts increasing while training loss continues to decrease. Validation F1 scores are also plotted to ensure that early stopping is triggered correctly. (See Fig. 36)

This training procedure ensures that the models are trained effectively while monitoring overfitting and early stopping. The performance metrics and visualizations provide insights into the model's learning process.

5.2.2. Code

```python
# TRAIN
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
loss.to(device)

# Early Stopping
best_metric = 0.0
no_improvement_epochs = 0
patience = 3  # Maximum number of epochs without improvement to be tolerated
val_metrics = []

# Epochs
num_epochs = 15
epoch_loss = 0

for epoch in range(num_epochs):
    print(f"Epoch {epoch+1}/{num_epochs}")
    print("-" * 10)

    model.train()  # Set model to training mode

    for batch_idx, (images, masks) in enumerate(data_loader):

        # To GPU
        images = images.to(device)
        masks = masks.to(device)
        # Optimizer
        optimizer.zero_grad()
        # Forward pass
        outputs = model(images)
        # Convert masks to index tensor
        masks = masks.round().long()
        # Calculate loss
        loss_value = loss(outputs, masks)
        # Accumulate loss
        epoch_loss += loss_value.item()
        # Compute predicted labels
        predicted_labels = torch.argmax(outputs, dim=1)
        # Treshold predicted
        predicted_labels = torch.where(outputs > 0.5, torch.tensor(1), torch.tensor(0))
        # Backward pass and optimization
        loss_value.backward()
        optimizer.step()

        if batch_idx % 100 == 0:
            print(f"Batch {batch_idx}/{len(data_loader)} - Loss: {loss_value.item()}")
```

Figure 33 - Code: 1st part of training.

After the initial part of training, which generally includes the steps in Fig. 33 and, after printing the loss for each batch, the metrics are calculated using the variables “y_pred” (the predicted mask generated by the model) and “y_true” (the ground truth mask).

This work utilized the “sklearn.metrics” module from the scikit-learn library to evaluate and measure the performance of the models. This facilitated the implementation of various performance metrics with minimal code. (See Fig. 34)

```
# Compute metrics after each epoch
    y_pred = predicted_labels.cpu().flatten()
    y_true = masks.cpu().flatten()

    accuracy = accuracy_score(y_true, y_pred)
    confusion = confusion_matrix(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    jaccard = jaccard_score(y_true, y_pred)
    train_jaccard.append(jaccard)

    # Print or store the metrics
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
    print("Jaccard Score:", jaccard)
    print(" ")

    # Compute average loss
    epoch_loss /= len(data_loader)
    train_losses.append(epoch_loss)
```

Figure 34 - Code: 2nd part of training: compute metrics after each epoch.

Next, as shown in Fig. 35 the model transitions to the evaluation phase after each training epoch. During this phase, the model's predictive performance is evaluated. Additionally, the validation loss is monitored to ensure continuous improvement of the model over time. The early stopping mechanism halts the training process if the F1 score does not improve for three consecutive epochs, in this case the code also prints the best score achieved.

```python
# Early stopping
    model.eval()
    with torch.no_grad():
        print(f"Validation Set")
        print("-" * 10)
        epoch_val_loss = 0
        for batch_idx, (val_image, val_mask) in enumerate(val_data_loader):

            # To GPU
            val_image = val_image.to(device)
            val_mask = val_mask.to(device)
            # Forward pass
            outputs = model(val_image)
            # Convert masks to index tensors
            val_mask = val_mask.round().long()
            # Calculate loss
            loss_value = loss(outputs, val_mask)
            # Save validation loss
            epoch_val_loss += loss_value.item()
            # Calculate the predicted labels
            predicted_labels = torch.argmax(outputs, dim=1)
            # Treshold predicted
            predicted_labels = torch.where(outputs > 0.5, torch.tensor(1), torch.tensor(0))

            if batch_idx % 100 == 0:
              print(f"Batch {batch_idx}/{len(val_data_loader)} - Loss: {loss_value.item()}")

        # Compute metrics after each epoch
        y_pred = predicted_labels.cpu().flatten()
        y_true = val_mask.cpu().flatten()
        # Compute average loss
        epoch_val_loss /= len(val_data_loader)
        val_losses.append(epoch_val_loss)
        # Compute Jaccard score
        val_jacc = jaccard_score(y_true, y_pred)
        val_jaccard.append(val_jacc)
        # Compute F1 score
        f1 = f1_score(y_true, y_pred)
        val_f1_scores.append(f1)

        # Set early stopping metric
        current_metric = val_f1_scores[-1]
        print("F1 Score:", current_metric)
        print("Jaccard Score:", val_jacc)
        print(" ")
        # Check for improvement
        if current_metric > best_metric:
            best_metric = current_metric
            no_improvement_epochs = 0
        else:
            no_improvement_epochs += 1
        # Check if number of epochs without improvement reached the patience threshold
        if no_improvement_epochs >= patience:
            print(f"Early stopping: F1 score has not improved for {patience} epochs")
            print(f"Best F1 score: {best_metric}")
            print(f" ")
            break
```

Figure 35 - Code: 3rd part of training: model.eval with early stopping.

At the end, as shown in Fig. 36, three plots are generated to overview loss trends, evaluate potential overfitting and monitor combined metrics behavior.

```
# Plot Loss
train_epochs = [i+1 for i in range(len(train_losses))]
val_epochs = [i+1 for i in range(len(val_losses))]

plt.plot(train_epochs, train_losses, label='Train Loss')
plt.plot(val_epochs, val_losses, label='Validation Loss')

# Find min and max of losses and trace overfit line
min_epoch = val_epochs[val_losses.index(min(val_losses))]
min_loss = min(min(train_losses), min(val_losses))
max_loss = max(max(train_losses), max(val_losses))
plt.vlines(min_epoch, min_loss, max_loss, linestyles='--', colors='red', label='Overfitting')
plt.title('Train and Val Losses')
plt.xlabel('Epochs')
plt.xticks(val_epochs)
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot validation F1 score
plt.plot(val_epochs, val_f1_scores, label='F1 Score')
plt.title('Validation F1 Score')
plt.xlabel('Epochs')
plt.xticks(val_epochs)
plt.ylabel('F1 Score')
plt.legend()
plt.show()

# Plt Jaccard
train_epochs_jacc = [i+1 for i in range(len(train_jaccard))]
val_epochs_jacc = [i+1 for i in range(len(val_jaccard))]
plt.plot(train_epochs_jacc, train_jaccard, label='Train Jaccard')
plt.plot(val_epochs_jacc, val_jaccard, label='Validation Jaccard')
plt.title('Jaccard Score')
plt.xlabel('Epochs')
plt.xticks(val_epochs_jacc)
plt.ylabel('Jaccard Score')
plt.legend()
plt.show()

print(" ")
best_epoch = train_jaccard.index(max(train_jaccard))
print(f'Best Train Epoch: {best_epoch+1}')

print(" ")
if min_epoch == len(val_losses):
    print(f'')
else:
    print(f'Validation Overfit in Epoch: {min_epoch} with value: {val_losses[min_epoch]}')
```

Figure 36 - Code: 4th part of training: Loss plot, Val F1 plot and Jaccard plot.

5.3. Model Testing

This section shows the code of the testing phase, the final metrics and the qualitative images examples. The evaluation process involves some steps to ensure an assessment of the model's performance.

The code used for model evaluation on the test set (see Fig. 37) is important for understanding how the models predictions are compared against the ground truth data. To provide a qualitative evaluation, 32 results will be also displayed. As shown in Fig. 38 each result will include the original image, the ground truth mask, and the mask predicted by the model. These visual comparisons will help in comparing the models ability to accurately segment food items.

Following the qualitative evaluation, the final metrics will be printed. (See Fig. 39) These metrics are essential for a quantitative comparison of different models. The core of this comparison will focus on the combined metrics: F1 score and Jaccard Score, as mentioned in previous chapters these metrics provide a robust measure of the models performance, capturing precision and recall in the case of the F1 score, and the overlap between the predicted and ground truth masks in the case of the Jaccard Score.

By analyzing these results, valuable insights can be gained into the strengths and weaknesses of the various models.

The evaluation process ensures that the most effective model is identified based on quantitative criteria, providing a foundation for further improvements and applications.

```python
# TEST
model.eval()

with torch.no_grad():
    for batch_idx, (test_image, test_mask) in enumerate(test_data_loader):
        # To GPU
        test_image = test_image.to(device)
        test_mask = test_mask.to(device)
        # Forward pass
        outputs = model(test_image)
        # Convert masks to index tensors
        test_mask = test_mask.round().long()
        loss_value = loss(outputs, test_mask)
        # Calculate the predicted labels
        predicted_labels = torch.argmax(outputs, dim=1)
        # Treshold predicted
        predicted_labels = torch.where(outputs > 0.5, torch.tensor(1), torch.tensor(0))

        # Some example images
        if batch_idx < 4:  # Only 2 batches
            for i in range(test_image.size(0)):
                image = test_image[i].cpu().numpy().transpose(1, 2, 0)  # Image to ndarray
                predicted_mask = predicted_labels[i].cpu().numpy()  # Predicted mask to ndarray
                original_mask = test_mask[i].cpu().numpy()  # Original mask to ndarray
                image = image / image.max()
                plt.figure()
                plt.subplot(1, 3, 1)
                plt.imshow(image)
                plt.title("Image")
                plt.axis('off')
                plt.subplot(1, 3, 2)
                plt.imshow(predicted_mask.squeeze(), cmap='gray')
                plt.title("Predicted mask")
                plt.axis('off')
                plt.subplot(1, 3, 3)
                plt.imshow(np.squeeze(original_mask), cmap='gray')
                plt.title("Original mask")
                plt.axis('off')
                plt.show()

    # Compute metrics after each epoch
    y_pred = predicted_labels.cpu().flatten()
    y_true = test_mask.cpu().flatten()

    accuracy = accuracy_score(y_true, y_pred)
    confusion = confusion_matrix(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    jaccard = jaccard_score(y_true, y_pred)
    # Print the metrics
    print("--- METRICS ---")
    print("Accuracy:", accuracy)
    print("Confusion Matrix:")
    print(confusion)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
    print("Jaccard Score:", jaccard)
```

Figure 37 - Code: Model evaluation on test set.

Two of the 32 resulting images are displayed below in Fig. 38. These thirty-two images correspond to four batches from the test set. As said previously qualitative results are important because they allow the researcher to see how the model segments the food.

Figure 38 - 1st row: original image, 2nd row: predicted mask, 3rd row: original mask.

At the end of the testing phase, the model evaluation results are printed (see Fig. 39).

```
--- METRICS ---
Accuracy: 0.92
Confusion Matrix:
[[827516 107580]
 [ 10885 528579]]
Precision: 0.831
Recall: 0.98
F1 Score: 0.899
Jaccard Score: 0.817
```

Figure 39 - Performance metrics example.

5.4. Evaluation Metrics

To evaluate the performance of the various deep learning models, several metrics were used. These metrics provide different perspectives on the model effectiveness in classifying and segmenting the data. Specifically, the evaluation metrics used include accuracy, confusion matrix, precision, recall, F1 score, and Jaccard score. Each metric captures aspects of the model's performance, offering a view of its strengths and weaknesses. The following section details the computation and significance of each metric used in this study. [20]

Key Terms

Understanding the terms related to false positives and false negatives in classification is essential:

- **FP (False Positive):** occurs when a model incorrectly predicts a positive result when it is actually negative. i.e. diagnosing a disease in a healthy patient.
- **FN (False Negative):** occurs when a model incorrectly predicts a negative result when it is actually positive. For example, not diagnosing a disease in a sick patient.
- **FPR (False Positive Rate):** percentage of false positives out of all actual negative results. $FPR = \frac{FP}{FP + TN}$
- **FNR (False Negative Rate):** percentage of false negatives out of all actual positive results. $FNR = \frac{FN}{TP + FN}$
- **TPR (True Positive Rate or Recall):** percentage of true positives out of all actual positive results. $TPR = \frac{TP}{FP + FN}$
- **TNR (True Negative Rate):** percentage of true negatives out of all actual negative results. $TNR = \frac{TN}{TN + FP}$

Confusion Matrix

The confusion matrix is a table used to describe the performance of a classification model. It summarizes the counts of true positive, true negative, false positive and false negative predictions.

Accuracy

Accuracy is the ratio of correctly predicted labels to the total number of predictions made. It is an aggregate measure but does not represent performance on specific classes.

$$Accuracy = \frac{Total\ Number\ of\ Prediction}{Number\ of\ Correct\ Predictions}$$

Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It measures the accuracy of the positive predictions made by the model.

$$Precision = \frac{True\ Positive}{True\ Positive\ +\ False\ Positive}$$

Recall

Recall (also known as sensitivity or true positive rate) is the ratio of correctly predicted positive observations to all observations in the actual class. It measures the ability of the model to capture all the positive cases.

$$Recall = \frac{True\ positive}{True\ Positive\ +\ False\ Negative}$$

Combined metrics, such as the F1 Score and Jaccard Score, are used to provide a more comprehensive evaluation of a model's performance by integrating multiple individual metrics into a single measure.

F1 Score

The F1 Score is the harmonic mean of precision and recall. It balances the two metrics, giving a single performance measure that accounts for both false positives and false negatives. The F1-score ranges from 0 to 1, where 1 indicates perfect precision and recall. It is sensitive to class imbalance and may overestimate performance on underrepresented classes, and it does not directly account for the spatial quality of segmentation. “Spatial quality” refers to the model's ability to produce segmentation masks that accurately match the shape and contours of objects in the image.

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Jaccard Score

The Jaccard score (also known as Intersection over Union “IoU”) is the size of the intersection divided by the size of the union of the predicted and actual labels. It provides a measure of how similar the predicted set is to the actual set.

$$Jaccard\ Score = \frac{Area\ of\ Intersection}{Area\ of\ Union}$$

“Area of intersection” is the number of correctly predicted positive pixels (TP), and “Area of union” is the total number of pixels present in either the predicted positives or the actual positives (i.e. the pixels of TP, FP and FN).

The Jaccard Score ranges from 0 to 1 and better captures the spatial quality of segmentation compared to the F1-score.

Optimization Considerations

Using a single metric like precision or recall can lead to imbalanced optimizations.

- Focusing only on precision might result in a very conservative model that returns few results with high precision but low recall.
- Instead, focusing only on recall might produce an aggressive model that returns many relevant results but also many false positives, leading to low precision.

To avoid these imbalances, it is better to optimize precision and recall simultaneously, for example, using the F1-score which combines both metrics, or use the Jaccard Score. These combined metrics should be also adopted as the early stopping criterion during the training phase.

Loss Function

In the Fundamentals Chapter (Section 2.1.1), the discussion was focused on common loss functions for general classification tasks, such as Cross-Entropy Loss, which is used in binary and multi-class classification problems. These functions are well-suited for tasks where the goal is to predict discrete class labels. However, the Dice Loss was not mentioned because it is a specialized loss function used for segmentation tasks.

In this study, the Dice Loss is used to train the models. The Dice Loss is particularly good for image segmentation tasks because it handles class imbalance. It measures the overlap between the predicted segmentation mask and the ground truth mask, providing a meaningful evaluation for tasks where the foreground and background classes are highly imbalanced.

The Dice coefficient, which ranges from 0 to 1 is used to calculate the Dice Loss. A value of 1 indicates perfect overlap between the predicted and ground truth masks, while a value of 0 indicates no overlap.

The Dice Coefficient is defined as:

$$Dice\ Coefficient\ = \frac{2|X \cap Y|}{|X| + |Y|}$$

Where X is the set of predicted pixels, and Y is the set of ground truth pixels.

The Dice Loss is defined as:

$$Dice\ Loss\ =\ 1\ -\ Dice\ Coefficient$$

This means that a Dice coefficient of 1 (perfect overlap) results in a Dice Loss of 0, instead a lower Dice coefficient (less overlap) results in a higher Dice Loss.

By focusing on the overlap, the Dice Loss optimizes the segmentation quality, which is critical in applications such as food segmentation where precise boundary detection is essential.

5.5. Model Performance Comparison

This section focuses on comparing the training performance and evaluation metrics of the six different image segmentation models. Each model's training performance is evaluated based on the Dice Loss during the training.

At the end of the training phase, a loss graph was generated for each model:

1. **Loss Graphs:** these graphs plot the training loss and validation loss over the epochs. The training loss indicates how well the model is fitting the training data, a decreasing trend suggesting effective learning. The validation loss reflects how well the model generalizes to unseen data, with a lower and stable validation loss indicating good generalization. An increase in validation loss after a period of decline suggests overfitting, where the model starts to memorize the training data instead of learning patterns.

The final metrics obtained from the test phase, including accuracy, precision, recall, F1 score and Jaccard Score are used in section 5.5.1 to determine which model performs best.

UNet

Below are the performance graphs for the UNet model, highlighting the loss and Jaccard Score through the training and validation phases.

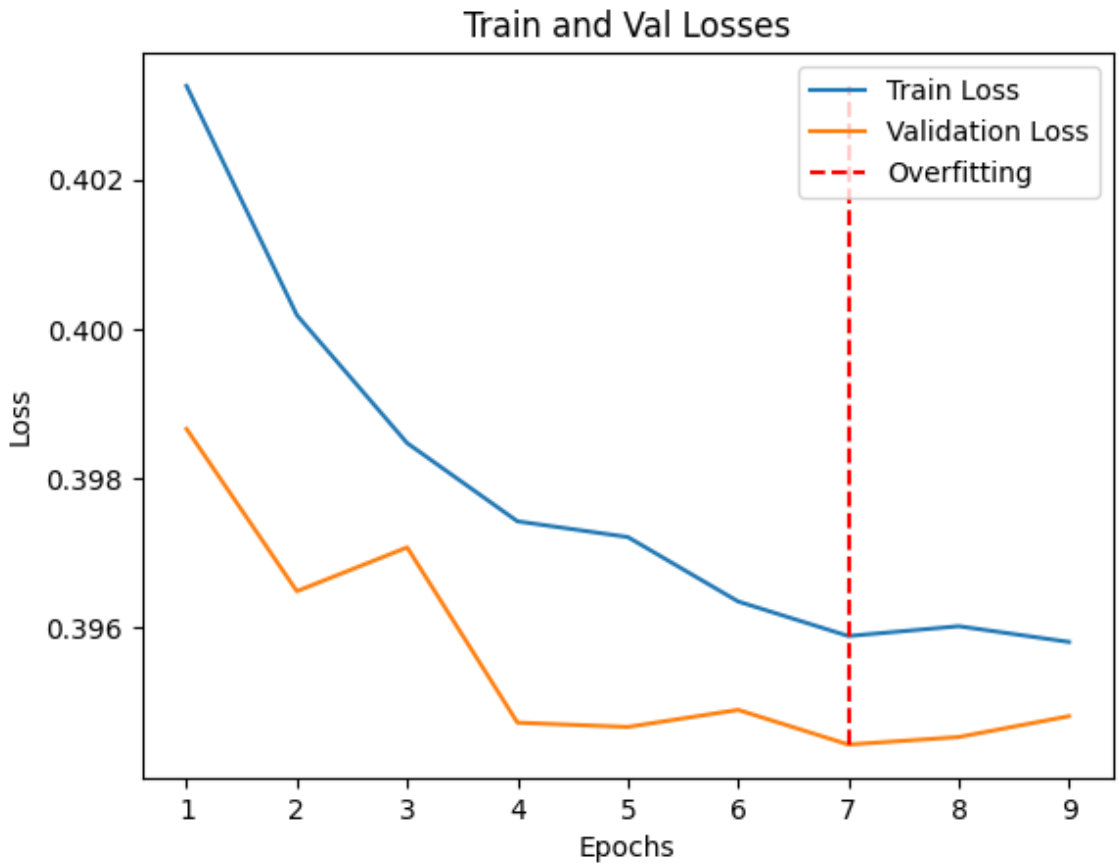

Figure 40 - UNet train and validation losses plot.

As shown in Fig. 40, the model demonstrates a consistent decrease in training loss over the epochs, indicating effective learning from the training set. Initially, the validation loss also decreases, showing the model's ability to generalize. However, around epoch 7, the validation loss starts to increase, suggesting the onset of overfitting. This increase indicates that the model begins to memorize the training data rather than learning general patterns, reducing its performance on unseen data. Early stopping was triggered appropriately around this point (at epoch 9) because the F1 score did not improve for 3 consecutive epochs, preventing further degradation in performance.

UNet++

Below are the performance graphs for the UNet++ model, highlighting the loss and Jaccard Score throughout the training and validation phases.

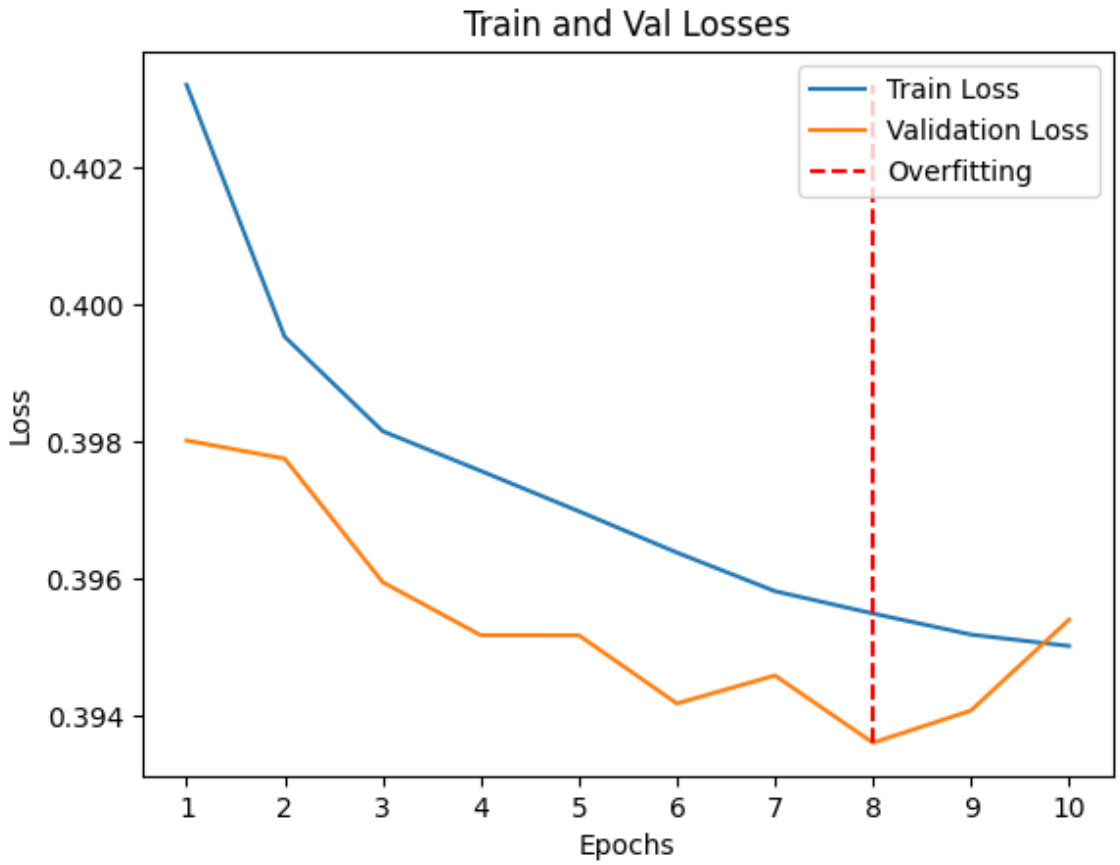

Figure 41 - UNet++ train and validation losses plot.

As shown in Fig. 41, the training loss for the UNet++ model steadily decreases over the epochs, indicating effective learning from the training data. The validation loss initially decreases as well, demonstrating good generalization. However, around epoch 8, the validation loss starts to rise, indicating the onset of overfitting. This suggests that the model begins to memorize. The increase in validation loss around this point makes early stopping an appropriate measure to prevent further performance degradation, in fact it halts the training at epoch 10.

Feature Pyramid Network

Below are the performance graphs for the FPN model, highlighting the loss and Jaccard Score throughout the training and validation phases.

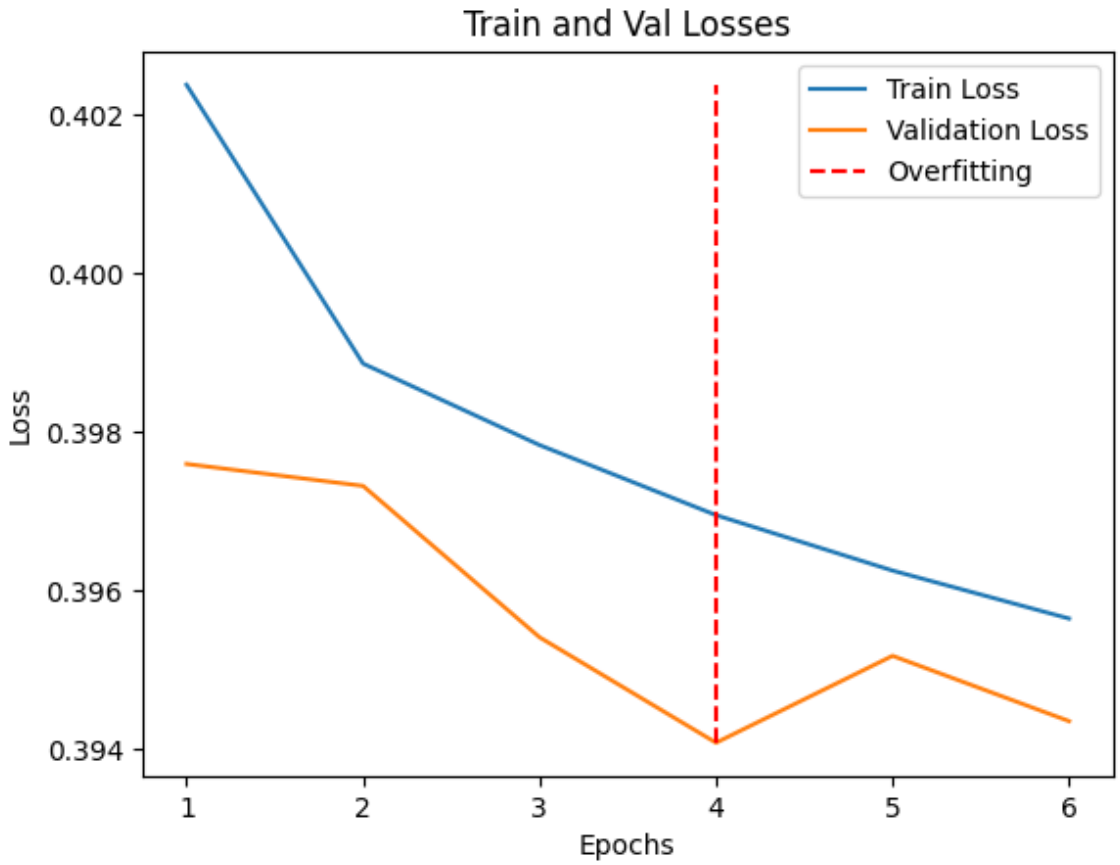

Figure 42 - FPN train and validation losses plot.

As depicted in Fig. 42, the training loss for the FPN model decreases over the epochs, indicating effective learning from the training data. The validation loss initially decreases as well, demonstrating good generalization capabilities. However, around epoch 4, the validation loss starts to increase signaling the onset of overfitting.

Early stopping was triggered around epoch 6 to prevent further degradation in performance.

PSPNet

Below are the performance graphs for the PSPNet model, highlighting the loss and Jaccard Score throughout the training and validation phases.

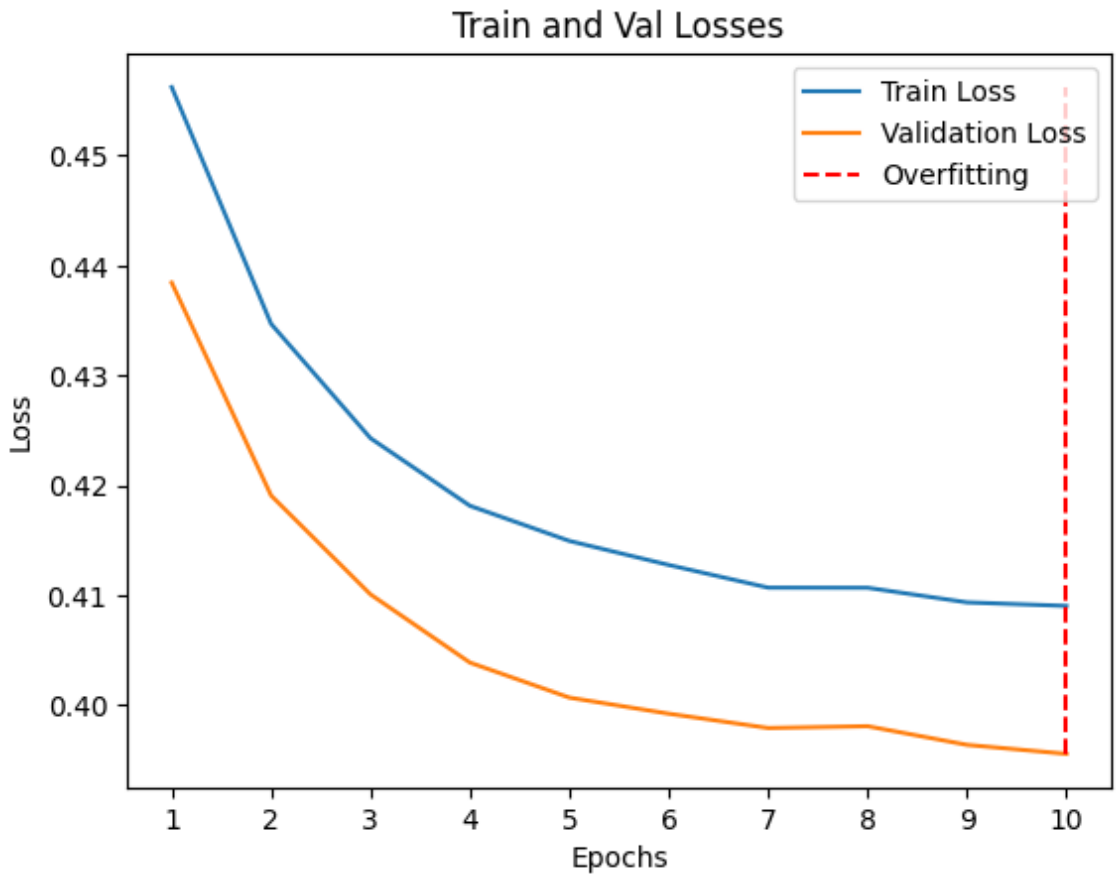

Figure 43 - PSPNet train and validation losses plot.

In Fig. 43, the training loss for the PSPNet model shows a stable decline over the epochs, suggesting that the model is effectively learning from the training dataset. The validation loss also decreases, which indicates good generalization to the validation data. Only in epoch 8, the validation loss increases a little bit, but then stabilizes again below 0.40.

DeepLabV3

Below are the performance graphs for the DeepLabV3 model, highlighting the loss and Jaccard Score throughout the training and validation phases.

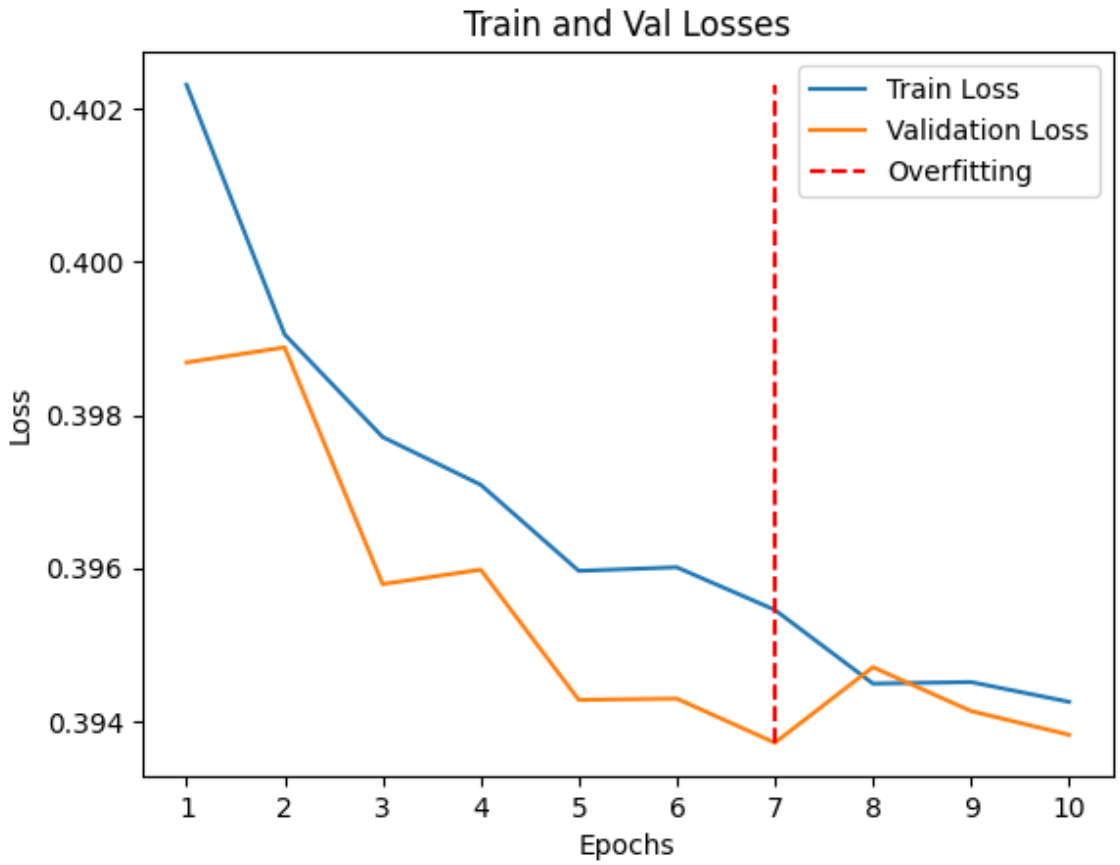

Figure 44 - DeepLabV3 train and validation losses plot.

In Fig. 44, the loss for the DeepLabV3 model decreases over the epochs, which indicates effective learning from the training dataset. The validation loss initially follows a decreasing trend, suggesting good generalization. However, around epoch 7, the validation loss begins to increase, marking the start of overfitting.

Early stopping at epoch 10 was appropriate to prevent further overfitting and degradation in performance.

DeepLabV3+

Below are the performance graphs for the DeepLabV3+ model, highlighting the loss and Jaccard Score throughout the training and validation phases.

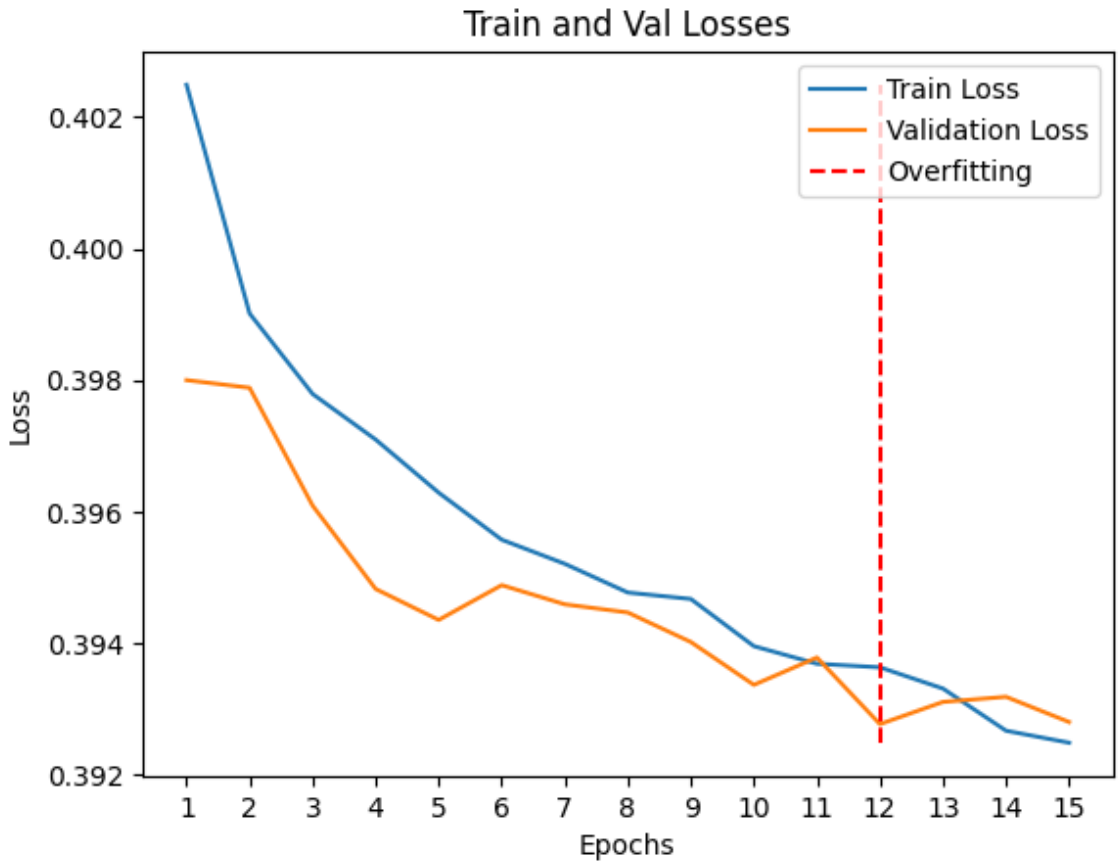

Figure 45 - DeepLabV3+ train and validation losses plot.

In Fig. 45, the training loss for the DeepLabV3+ model shows a consistent decline over the epochs, suggesting effective learning from the training dataset. The validation loss initially decreases as well, indicating good generalization. However, around epoch 12, the validation loss starts to increase again, marking the beginning of overfitting.

The early stopping near this point was not activated as the F1 score metric did not worsen for 3 consecutive epochs.

5.5.1. Comparison of Test Metrics

This section presents tables and graphs for a comparative analysis of the test metrics for the different models evaluated in this study.

The results are summarized in Table 1, this analysis facilitates the determination of the most effective model for the binary image segmentation task.

Model	Accuracy	Precision	Recall	F1 Score	Jaccard Score
UNET	0.904	0.809	0.963	0.88	0.785
UNET PLUS	0.895	0.80	0.951	0.869	0.769
PSPNET	0.923	0.853	0.952	0.90	0.818
FPN	0.928	0.86	0.959	0.907	0.829
DLV3	0.917	0.831	0.97	0.895	0.81
DLV3+	0.929	0.853	0.974	0.909	0.834

Table 1 - Test results comparison

The performance table offers a comparison of the six models across our critical metrics: Accuracy, Precision, Recall, F1 Score, and Jaccard Score.

Notably, DeepLabV3+ excels in most metrics, showcasing its better performance. FPN and PSPNet also demonstrate high precision and balanced F1 scores, indicating their robustness in identifying relevant segments and maintaining precision. DeepLabV3, while effective, has slightly lower precision and Jaccard scores compared to DeepLabV3+, highlighting areas for improvement in reducing false positives. UNET and UNET Plus, though strong in recall, indicate potential for improvement in precision, F1 score, and Jaccard score.

This table serves as a reference to discern each model's strengths and suitability for different binary segmentation tasks.

The bar charts below compare the single metrics of the different models used in this study. Each bar represents the score achieved by a specific model on the test set.

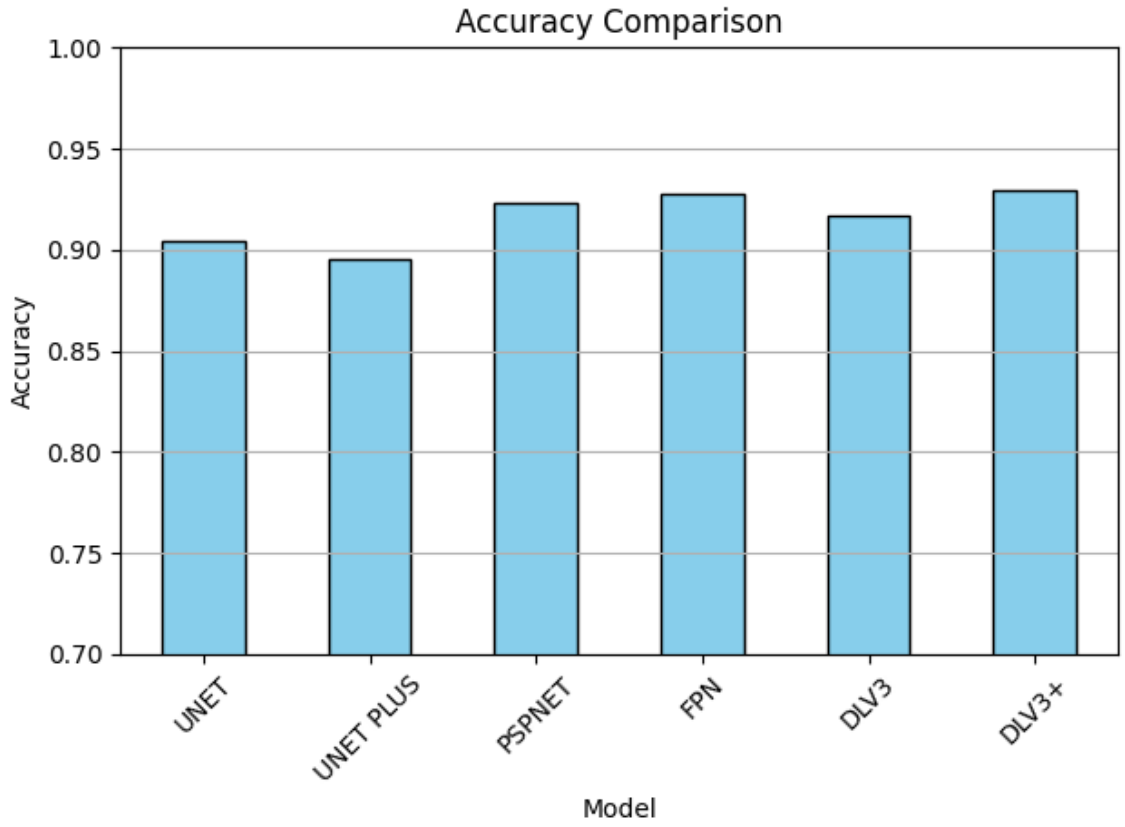

Figure 46 - Accuracy comparison plot.

As observed from Fig. 46, the DeepLabV3+ model achieved the highest accuracy, closely followed by FPN and PSPNet. This indicates that DeepLabV3+ is particularly effective in correctly classifying pixels as either food or background. The accuracy values for all models are relatively high, indicating good overall performance.

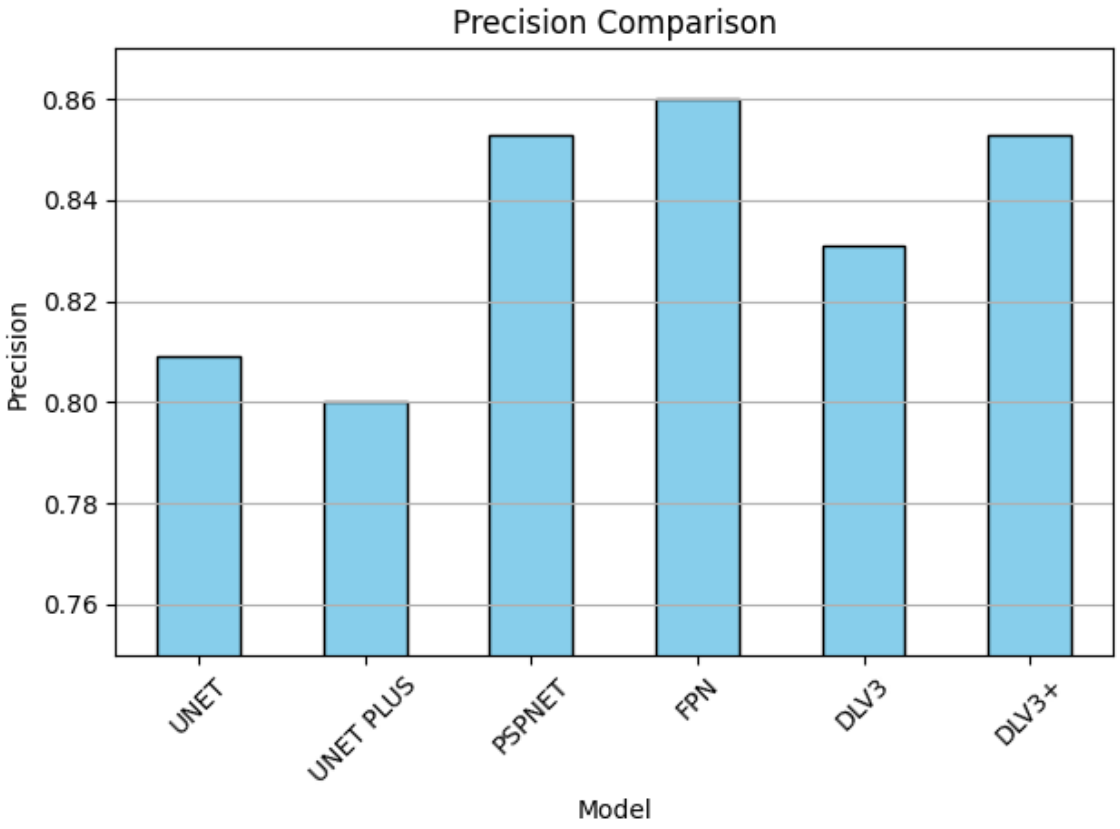

Figure 47 - Precision comparison plot.

The precision comparison plot (Fig. 47) provides a clear visual representation of the relative performance of each model, highlighting that FPN is the most precise for food image segmentation in this study.

This suggests that also PSPNet and DeepLabV3+, that achieve a similar score are particularly good at classifying food pixels while reducing the false positives.

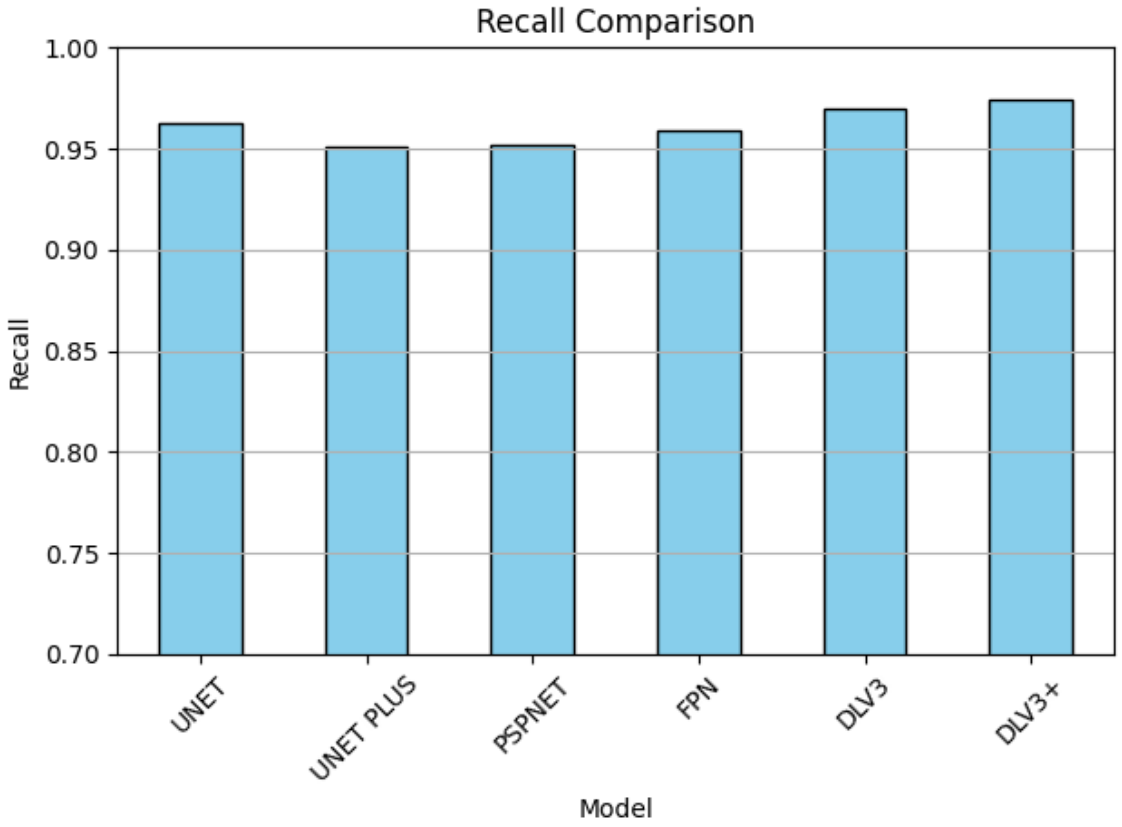

Figure 48 - Recall comparison plot.

This bar chart (Fig. 48) compares the recall of various models. Each bar represents the recall score of a specific model, indicating the proportion of true positive food pixels correctly identified among all actual food pixels.

All models demonstrate high recall values, with DeepLabV3 and DeepLabV3+ achieving the highest scores, indicating their effectiveness in capturing the majority of food pixels.

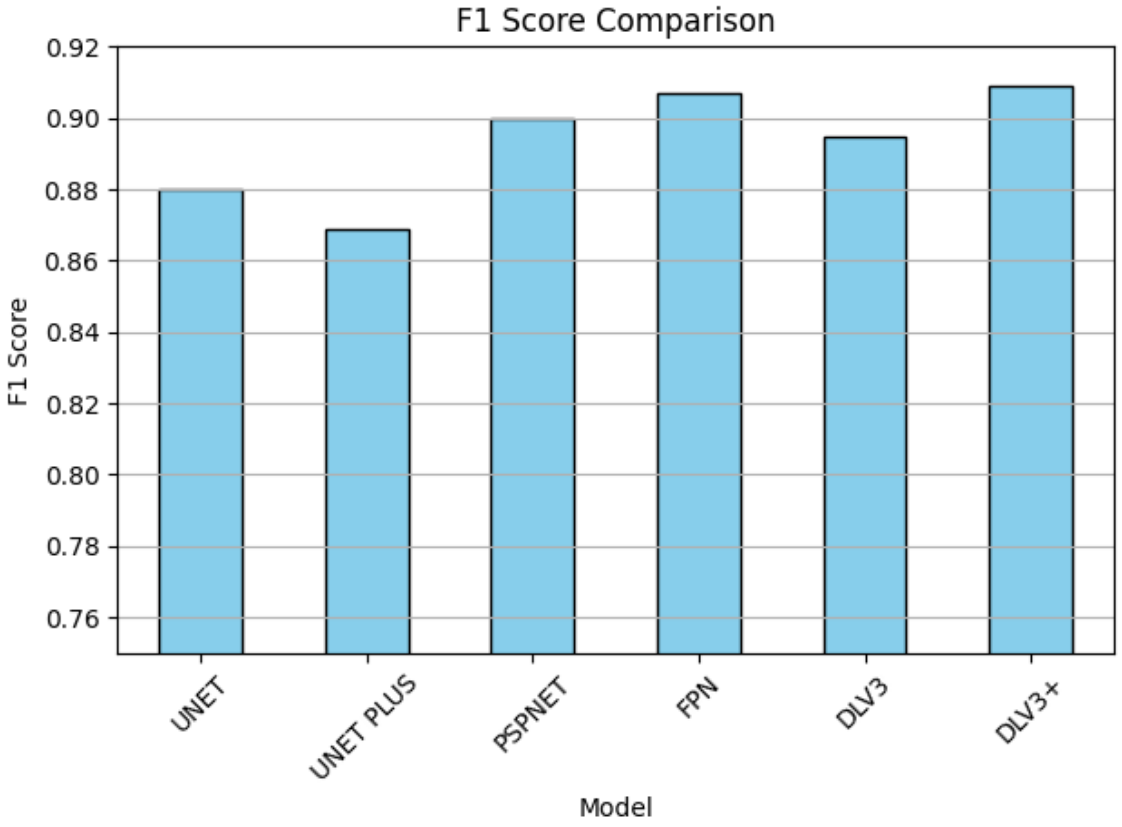

Figure 49 - F1 Score comparison plot.

In Fig. 49 each bar represents the F1 score of a model, DeepLabV3+ and FPN achieve the highest F1 scores. Also PSPNet and DeepLabV3 achieve good scores, indicating the harmonic mean of precision and recall. UNET and UNET++ have relatively lower F1 scores.

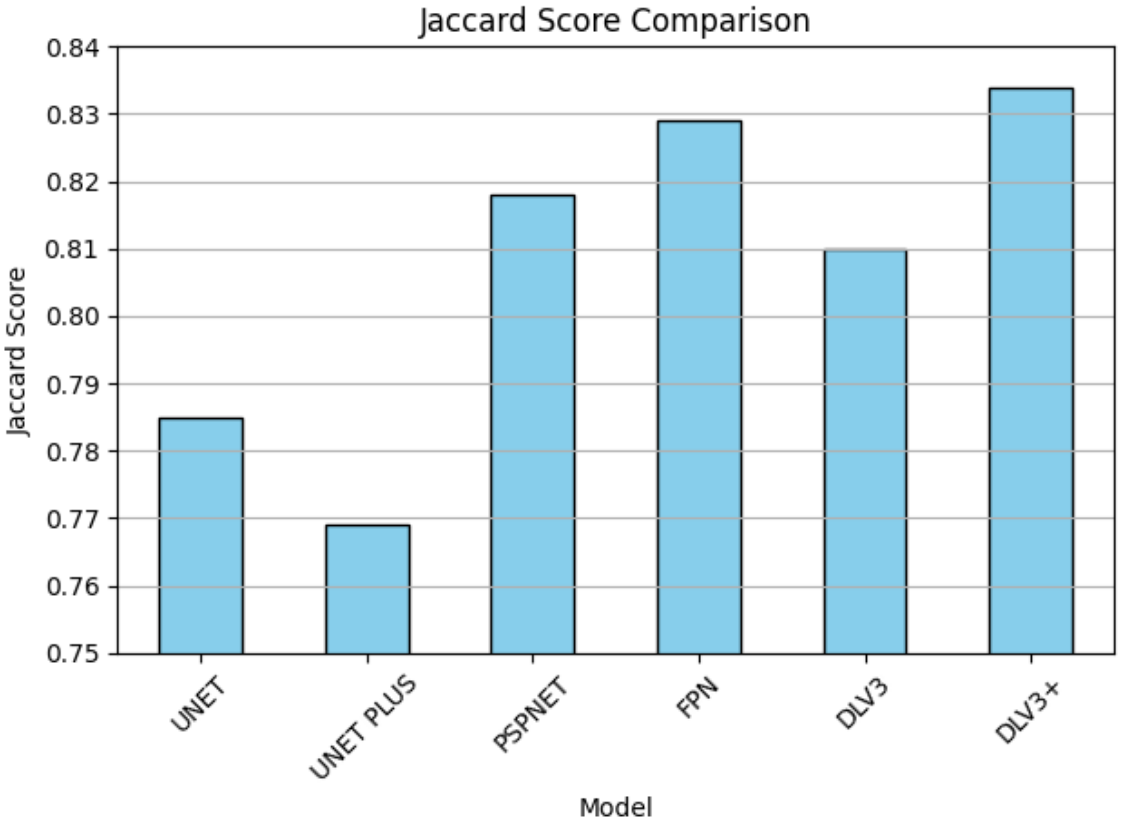

Figure 50 - Jaccard comparison plot.

The Jaccard score plot (Fig. 50) indicates that FPN and DeepLabV3+ are the top performers, highlighting their capability to match the predicted segmentation with the ground truth. UNET and UNET++ show relatively lower Jaccard scores, suggesting less accurate segmentation performance.

Figures 51-56 shows the segmentation outputs from the various models using the same image of sliced bread, illustrating in a qualitative way how each model performs.

Figure 51 - Unet segmentation.

Figure 52 - Unet++ segmentation.

Figure 53 - FPN segmentation.

Figure 54 - PSPNet segmentation.

Figure 55 - DeepLabV3 segmentation.

Figure 56 - DeepLabV3+ segmentation.

As seen from these qualitative results, DeepLabV3+ and FPN show the best segmentation performance, demonstrating their ability to accurately delineate food items. These models excel in capturing fine details and maintaining high precision, which is crucial for accurate segmentation tasks.

6. DISCUSSION

This chapter provides an analysis and interpretation of the experimental results. By examining both quantitative metrics and qualitative observations, this discussion aims to underscore the strengths and limitations of the models, providing insights into their performance and practical applications.

6.1. Results Analysis

The comparative analysis of the segmentation models shown in Section 5.5.1 demonstrated that DeepLabV3+ and FPN excel in most metrics compared to other models in this binary segmentation on the UEC-FoodPixComplete dataset.

The results highlight some strengths and limitations of each model:

- **DeepLabV3+** outperformed other models in terms of accuracy (0.92), recall (0.97), F1 score (0.90) and Jaccard Score (0.83).
- **FPN** exhibited high precision (0.86) and Jaccard Score (0.83), indicating its strength in accurately identifying food pixels while minimizing false positives.
- **DeepLabV3** showed good performance, especially in terms of recall (0.97), indicating its efficiency in capturing the majority of food pixels. This high recall is essential for applications that require comprehensive food detection, such as dietary assessment tools.
- **PSPNet** also performed well, especially in maintaining a balance between precision and recall. Its pyramid pooling module effectively captures different sub-region representations, enhancing its performance in diverse and cluttered food images.
- **UNet** demonstrated a reasonable segmentation capability. It suffered from overfitting after a few epochs of training, resulting in diminished precision (0.80) on complex food images.

- **UNet++** improves the Jaccard Score (0.78) compared to UNet, but still shows similar overfitting tendencies. It has the same recall and precision compared with UNet, but still was outperformed by models such as FPN and DeepLabV3+ on more complex tasks.

DeepLabV3+'s best performances can be attributed to his atrous convolutions, which allow the model to capture context at multiple scales without losing resolution of fine details. This feature is critical for accurately segmenting food items which may vary greatly in appearance and size.

FPN instead utilizes a pyramid structure to integrate semantic information from different resolutions, which effectively addresses the challenges posed by the diverse dimensions of food items in the dataset.

Some reasons that explain the performance of the models:

- **Architecture and Multi-scale:** FPN leverages a feature pyramid network that enhances object detection across various scales. This feature is particularly beneficial for this dataset, which encompasses a wide array of food types varying in size and proportions. Similarly, DeepLabV3+ with its atrous pooling handles scale and shape variations effectively, ensuring detailed segmentation even in complex visual contexts.
- **Capturing Details:** both models are designed to retain high-resolution details through their pooling and upsampling operations, essential for recognizing specific food characteristics such as texture and contours.
- **Context Variations:** accurate segmentation in images with varied backgrounds and lighting conditions is crucial, and both DeepLabV3+ and FPN show considerable adaptability to these variations.

These findings align with the existing literature that recognise DeepLabV3 and FPN for their effectiveness in complex segmentation tasks [16][18][19]. The best performance of these models over others, particularly in scenarios with high variety and complexity, is consistent with their capabilities to manage multi-resolution contexts effectively.

Qualitative Analysis

Qualitative results revealed that DeepLabV3+, DeepLabV3, and FPN maintained high segmentation accuracy and detail preservation across various food images. Their ability to delineate fine boundaries and handle occlusions was evident in the visual outputs, reinforcing their suitability for practical food segmentation applications.

The predicted masks from UNet and UNet++ instead often lacked detail precision and exhibited a higher occurrence of false positives. In this study, these models generated less accurate segmentations compared to the others (Fig. 57), highlighting their limitations in handling complex contexts.

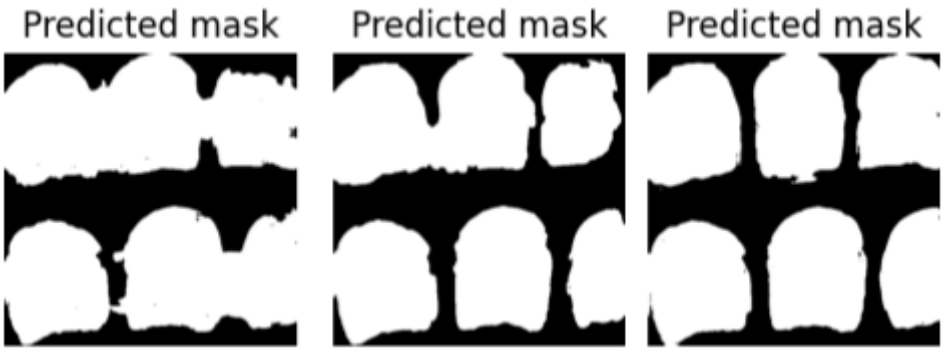

Figure 57 - Left: UNet segmentation, Middle: UNet++ segmentation, Left: DeepLabV3+ segmentation.

Limitations

Major limitations of this study include the dataset size, the variation in types of food and lighting conditions in the images, and the use of a single image pre-processing method. These limitations underscore the need to explore data augmentation techniques and diverse pre-processing approaches to further improve model performance and generalizability.

6.2. Potential Applications

Food image segmentation has several applications that extend beyond the academic field and into production fields, particularly the food and catering industries.

Accurate segmentation technology would be applied innovatively to increase productivity and make better customer impressions. The most common application could be related to digital menu systems, for example, restaurants could create interactive digital menus where each dish is presented with high accuracy. This will let the customers easily find and perceive the ingredients and portions, thereby improving their experience of ordering.

Another critical application is the nutritional analysis, segmentation of food items on a plate allows restaurants to deliver detailed nutrition information. It will be beneficial with dietary restrictions or for people wanting to make healthy choices.

Food segmentation by intelligent cooking appliances in the kitchen could help the chef use it and prepare dishes. They can identify ingredients, recommend recipes and ensure proper portion sizing.

It can also have a significant role in inventory management, this system will be effective in tracking the usage of various ingredients in preparations conducted by the company to reduce waste and optimize the stock.

Implementing food segmentation technology in restaurants and food operations may result in higher efficiency, cost reduction and improved customer experience.

7. CONCLUSIONS

This study conducted a comparative analysis of deep learning techniques for the segmentation of food images, evaluating advanced models such as DeepLabV3, FPN and PSPNet. Through this analysis, it was demonstrated how convolutional neural networks can overcome various challenges intrinsic to food image segmentation, including variability in shapes, textures and lighting conditions.

Among the models examined FPN and DeepLabV3 show the best performance, excelling due to their technical features like atrous convolutions and pyramid structures, as well as their ability to capture details and maintain contextual coherence in food images. This highlights the importance of the model architecture in adapting to the variables present in food images.

The comparative evaluation of different methodologies underscored the need to identify the most effective solutions for semantic segmentation of food images based on their performance and potential applications in specific contexts.

The study revealed the potential of deep learning techniques to improve the accuracy and reliability of food image segmentation, offering opportunities for practical applications in nutrition and health, such as dietary monitoring systems and meal planning applications.

Overall, this study provided insights into the strengths and limitations of the various deep learning models for food segmentation, contributing to the understanding of this complex task in the field of computer vision and food applications.

8. REFERENCES

[1] Z.H. Zhou, "Machine Learning", Springer Nature, 2021. ISBN: 9811519676, 9789811519673

[2] S. Ghosh, N. Das, I. Das, U. Maulik, "Understanding Deep Learning Techniques for Image Segmentation", ACM Computing Surveys, vol. 52, no. 4, Article 67, August 2019. https://doi.org/10.1145/3329784

[3] G. Ciocca, P. Napoletano, R. Schettini, "Food Recognition: A New Dataset, Experiments, and Results", IEEE Journal of Biomedical and Health Informatics, May 2017. http://dx.doi.org/10.1109/JBHI.2016.2636441

[4] S. Aslan, G. Ciocca, D. Mazzini, R. Schettini, "Benchmarking algorithms for food localization and semantic segmentation", Int. J. Mach. Learn. & Cyber. 11, June 2020. https://doi.org/10.1007/s13042-020-01153-z

[5] M.-Y. Chen, Y.-H. Yang, C.-J. Ho, S.-H. Wang, S.-M. Liu, E. Chang, C.-H. Yeh, and M. Ouhyoung, “Automatic chinese food identification and quantity estimation,” in SIGGRAPH Asia 2012 Technical Briefs, pp. 1–4, 2012. http://dx.doi.org/10.1145/2407746.2407775

[6] X. Wu, X. Fu, Y. Liu, E. Lim, S.C.H. Hoi, Q. Sun, "A Large-Scale Benchmark for Food Image Segmentation", MM 2021: ACM Multimedia Conference, pages 506-515. http://dx.doi.org/10.1145/3474085.3475201

[7] J. Marin, “Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images”, Oct. 2018. https://doi.org/10.48550/arXiv.1810.06553

[8] J. Sultana et al. "A Study on Food Value Estimation From Images: Taxonomies, Datasets, and Techniques," in IEEE Access, vol. 11, pp. 45910-45935, 2023. https://doi.org/10.1109/ACCESS.2023.3274475

[9] K. Okamoto, K. Yanai, "UEC-FoodPix Complete: A Large-scale Food Image Segmentation Dataset", Springer, Cham, February 2021. https://doi.org/10.1007/978-3-030-68821-9_51

[10] K. He, X. Zhang, S. Ren, J. Sun, "Deep Residual Learning for Image Recognition", arXiv:1512.03385, December 2015. https://doi.org/10.48550/arXiv.1512.03385

[11] O. Russakovsky Et al. “ImageNet Large Scale Visual Recognition Challenge,” arXiv.org, September 2014. https://doi.org/10.48550/arXiv.1409.0575

[12] Wikipedia contributors, "PyTorch," Wikipedia, accessed May 16, 2024. https://en.wikipedia.org/w/index.php?title=PyTorch&oldid=1223171021

[13] Pavel Iakubovskii Et al. “Python library with Neural Networks for Image Segmentation based on PyTorch”, Github Repository, 2019. https://github.com/qubvel/segmentation_models.pytorch

[14] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation" in arXiv:1505.04597, 2015. https://doi.org/10.48550/arXiv.1505.04597

[15] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation" in arXiv:1807.10165, 2018. https://doi.org/10.48550/arXiv.1807.10165

[16] T. Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection" presented at the COCO 2017 Stuff Segmentation Task, 2017.
http://dx.doi.org/10.1109/CVPR.2017.106

[17] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network" in arXiv preprint arXiv:1612.01105, 2016. https://doi.org/10.48550/arXiv.1612.01105

[18] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Rethinking atrous convolution for semantic image segmentation," in arXiv arXiv:1706.05587, 2017. https://doi.org/10.48550/arXiv.1706.05587

[19] L. C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in arXiv preprint arXiv:1802.02611, 2018. https://doi.org/10.48550/arXiv.1802.02611

[20] Vujović, Ž. "Classification model evaluation metrics.", International Journal of Advanced Computer Science and Applications, 12(6), 599-606, 2021 http://dx.doi.org/10.14569/IJACSA.2021.0120670