Warsaw University of Technology

Faculty of

Power and Aeronautical Engineering

Institute of Aeronautics and Applied Mechanics

Master's diploma thesis

In the field of study Automation and robotics

And specialization Robotics

Segmentation of LIDAR and camera data from an autonomous vehicle

Eyob Azene Yigezu

Student record book number 317238

Thesis supervisor: Andrzej Kordecki (Ph.D.)

Warsaw 2022/2023

Table of Contents

List of Figures

Acronyms

LIDAR	Light detection and ranging
AI	Artificial Intelligence
ANNs	Artificial Neural Networks
TanH	Hyperbolic Tangent
ReLU	Rectified Linear Unit
ELU	Exponential Linear Units
CNNs	Convolutional Neural Networks
FCN	Fully Convolutional Neural Networks
RNN	Recursive Neural Networks
DNNs	Deep Neural Networks
R-CNN	Region Based Convolutional Neural Networks
DCNNs	Deep Convolutional Neural Networks
CRF	Conditional Random Field
CAM	Context Aggregation Module
MRFs	Markov Random Network
RBMs	Restricted Boltzmann Machines
LSTM	Long Short-Term Memory networks
IU	Intersection over Union
PSPNet	Pyramid Scene Parsing Network
ASPP	Atrous spatial pyramid pooling
FPN	Feature pyramid network
CAM	Channel attention module

Chapter 1

Introduction

As the level of vehicle automation increases, research on autonomous vehicles has become a hot topic. The key technologies for autonomous vehicles include perception, decision-making, and control. Among these, the environmental perception system, which converts physical world information into digital signals, is the foundation of the hardware architecture for autonomous vehicles.

In autonomous vehicles, perception is achieved using various sensors, such as LIDARs (Light Detection and Ranging), radars, cameras, and ultrasound sensors. Perception is an important aspect of path finding in robotics because it allows the robot to gather information about the environment and to build a map of its surroundings. This information is used to identify features that the robot can use to navigate, such as walls, obstacles, landmarks, or other reference points. In this paper, we will use data from both cameras and LIDARs for path finding of the autonomous vehicle. Cameras have the advantage of high resolution, fast speed, and rich information transmission, as well as low cost, while LIDARs have the advantage of accurate 3D perception and a wealth of information [1].

LIDAR and cameras are both sensors that can be used for path finding in robotics. LIDAR works by emitting laser pulses and measuring the time it takes for the pulses to bounce back after hitting an object. The LIDAR can then use this time delay to calculate the distance to the object. This information can be used to create a 2D or 3D map of the environment, which can then be used for path planning. Cameras on the other hand, capture visual information in the form of images or video. This visual information can be used to identify features in the environment, such as lines, corners, or objects. The robot can then use this information to navigate around the environment and find a path to its destination [2] [3].

There are various techniques for learning from the surrounding environment, depending on the desired outcome and required level of accuracy. These techniques can be used for tasks such as image classification, object detection, object tracking, or image segmentation. In the field of computer vision, image segmentation is the process of dividing a digital image into multiple segments, or groups of pixels. Image segmentation can be either semantic, which involves understanding and classifying each pixel into multiple classes (labels) at the pixel level, or instance, which involves identifying and separating different instances of objects within an image [4].

Classical computer vision techniques provide the necessary tools for tasks such as image classification, object detection, and image segmentation, but these tools are often inflexible and limited in their ability to handle variations in external and internal parameters, such as illumination, occlusion, depth, camera resolution, disturbances, or noise. Deep learning has proven to be an effective approach for computer vision applications that operate in dynamic environments. As an example, Google employs LIDAR technology in their self-driving cars to calculate the distance and time of each laser pulse, providing precise height and distance data for nearby objects [5]. In their research, Denis et.al [6] demonstrate a sophisticated indoor positioning and navigation technique for autonomous robots, which combines LIDAR and GPS sensors within the Robot Operating System framework, ensuring accurate navigation in restricted spaces. Moreover, in agriculture, Jawad et.al [7] present a study on a ROS-based mobile field robot that uses a nodding 2D LiDAR for effective phenotyping tasks, adopting a combined GPS-LiDAR navigation approach for obscured crop rows, promoting agricultural automation, and addressing the phenotyping bottleneck. In this article, we will apply deep learning to conduct semantic image segmentation for autonomous vehicles, utilizing data from both camera and LIDAR sensors.

1.1 Goal of the thesis

To be written . . .

1.2 Motivations

In the age of automation, it is not far-fetched to imagine a future where transportation is fully driverless. There has been significant progress in autonomous driving since the first successful demonstration in the 1980s [8]. It is expected that autonomous vehicles will be safe and reliable by 2025 and will be commercially available in many areas by 2030 [9]. Autonomous vehicles have the potential to reduce traffic congestion, improve road safety, and lower carbon emissions [10]. However, developing reliable autonomous vehicles is still a very challenging task. This is because driverless cars are intelligent agents that must perceive, predict, decide, plan, and execute their actions in the real world, often in uncontrolled or complex environments. Even a small error in the system can result in serious accidents.

Currently, the lack of maturity in environmental perception technology is the main factor holding back the overall performance of autonomous vehicles and preventing their widespread commercialization.

Perception systems in autonomous vehicles need to be accurate, robust, and real-time in order to be effective. Specifically, they need to provide precise information about the driving environment, be able to function properly in adverse weather and unexpected situations (open-set conditions) and be able to process data quickly when the vehicle is driving at high speeds. To achieve these goals, autonomous vehicles are often equipped with multiple types of sensors (e.g., cameras, LIDARs, radars), and these sensing modalities are fused to take advantage of their complementary properties. Research has shown that deep learning methods are currently the most effective for these tasks, and methods that fuse camera and LIDAR data often produce better results than those using just one of these modalities [11].

Deep learning has recently shown great potential and achieved impressive results in a variety of computer vision tasks. A deep neural network can effectively learn hierarchical feature representations with a large dataset [12]. Semantic segmentation is one research area that has benefited from the power of deep learning. This paper will focus on the fundamental perception problem of semantic segmentation using data from sensors, such as LIDARs and cameras.

Chapter 2

Deep Learning

Deep learning is a technique within the field of machine learning, which is a branch of artificial intelligence concentrated on multilayer artificial neural networks. These networks are intended to learn and extract complicated features and patterns from huge datasets without the need for explicit feature engineering [13] [14].

Deep learning has its origins with the introduction of the perceptron [15], a type of single-layer neural network. However, it wasn't until the 1990s and early 2000s that significant progress was made in the development of deep learning algorithms, due to advances in computing power and the availability of large amounts of data.

2.1 Artificial neural network (ANN)

Deep learning algorithms, which are a type of artificial intelligence that involves the use of algorithms to learn from data and make predictions or decisions, are based on artificial neural networks (ANNs). ANNs are modeled after the structure and function of the brain and are composed of interconnected processing nodes, or neurons, that communicate with one another to solve complex problems. Each neuron receives input from other neurons, processes it using a mathematical function, and sends the result to other neurons in the network. This allows the neural network to learn and adapt to new information, making it a powerful tool for various applications including image and speech recognition, natural language processing, and decision making.

The perceptron is an algorithm for binary classification that was developed as a model of a biological neuron in the field of artificial neural networks. It was designed to classify inputs, such as visual data or labels, into one of two classes by dividing them with a boundary line. The perceptron was an early example of a supervised machine learning algorithm and has been used in a variety of applications [15].

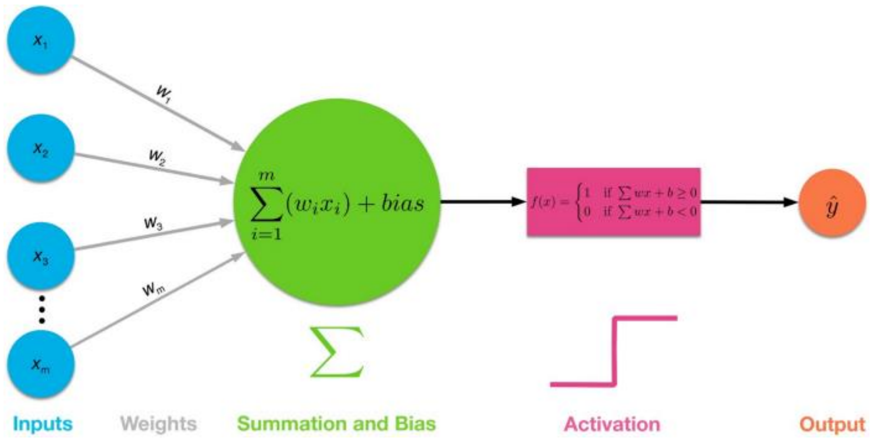
The classification of objects and patterns is a crucial aspect of machine learning, particularly in image processing. Machine learning algorithms use various techniques to identify and analyze patterns. In classification tasks, perceptron algorithms analyze classes and patterns in order to achieve linear separation between different classes of objects and corresponding patterns derived from numerical or visual input data.

2.2 Model of Perceptron

The perceptron model was developed in 1958 at Cornell Aeronautical Laboratory by Frank Rosenblatt for machine-implemented image recognition. It was one of the first artificial neural networks ever created [15]. At the time, the perceptron algorithm was thought to be a breakthrough in artificial intelligence and there were high hopes for it. However, technical limitations were discovered, and it was found that the single-layered perceptron model could only be applied to linearly separable classes. Later, it was discovered that multi-layered perceptron algorithms could classify nonlinearly separable groups.

2.2.1 Single-layered perceptron model

A single layer perceptron is a type of artificial neural network that consists of only one layer (Fig. 1). It calculates the sum of the input vector multiplied by the corresponding weight vector and passes the result through an activation function to produce an output [16]. A single-layer perceptron model as shown in figure 1 includes a feed-forward network that uses a threshold transfer function in its model [17] and is capable of analyzing linearly separable objects with binary outcomes, such as 1 and 0.



x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output

Figure 1 Architectural graph of a single layered perceptron [18].

Neuron mode can be written as equation 1.

$$y = F\left(\sum_{i=1}^{n} w_i x_i + b = 1\right) \tag{1}$$

This is the activation function for a simple model of neuron that receives multiple input signals x_i and computes the weighted w_i sum of those inputs, then checks the activation function F and outputs the result y. The transfer function is a critical uncertain component in this model. It defines the properties of the artificial neuron and can take any mathematical function. The transfer function chosen is determined by the specific challenge that the artificial neural network must solve. It is usually selected from a set of functions such as the Step function, Linear function, and Non-linear (Sigmoid) function.

The single-layered perceptron model has a type of algorithm that does not use previous information. Therefore, the weights are initially assigned randomly. The algorithm calculates the sum of the weighted inputs. If this sum is greater than a predetermined threshold value, the perceptron is activated and produces an output of +1. In simpler terms, the perceptron model receives multiple input values, processes them, and produces an output. If the output matches the desired result, the model is considered to be performing satisfactorily and the weights do not need to be modified. However, if the model does not produce the desired output, the weights are adjusted in order to minimize errors. A single perceptron is not sufficient to learn complex systems, so it is necessary to expand the perceptron to create a multi-layered perceptron model.

2.2.2 Multi-layered perceptron model

The multi-layered perceptron (MLP) model shown in (Fig 2), is similar to a single-layered perceptron model, but it has more hidden layers [17]. The input layer is the first layer of a neural network, and its main function is to receive input data from external sources. This data can come in different forms, such as data files or web services. After the input data has been received, it is passed on to the subsequent layers of artificial neurons for further processing.

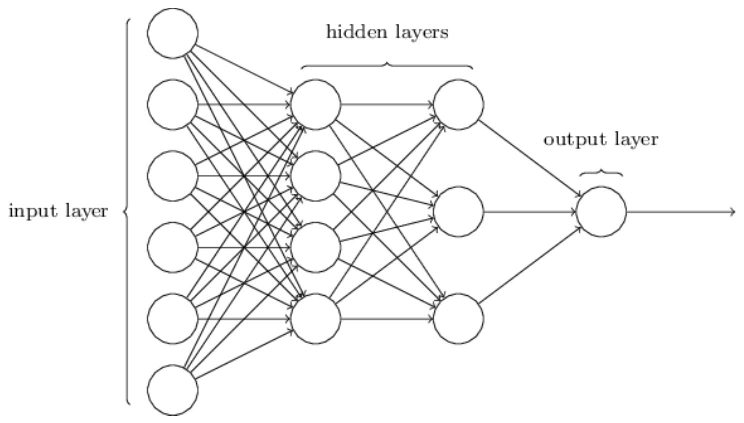


Figure 2 Architectural graph of a multilayer perceptron with two hidden layers [19].

The hidden layers of a neural network are located between the input layer and the output layer. These layers are responsible for processing the input data by applying non-linear functions to it. The hidden layers are essential for enabling a neural network to learn complex tasks and achieve good performance. They perform multiple functions, such as data transformation and automatic feature creation. The hidden layers are private to the neural network, meaning that they are not visible to external systems. The number of hidden layers in a neural network can have a big impact on its performance and the time it takes to produce output. Typically, the more hidden layers a neural network has, the longer it will take to produce output.

The output layer is the final layer in a neural network, and it produces the output of the problem. The output layer receives inputs from the previous layers, performs calculations through its neurons, and produces the final output. In complex neural networks, the output layer typically receives inputs from the hidden layers.

In simple terms, a multi-layered network can be thought of as a network of artificial neurons organized into multiple layers. Unlike a single-layered network, which uses a linear activation function, a multi-layered network uses non-linear activation functions, such as sigmoid, TanH, or ReLU functions, to process the input data. These non-linear activation functions allow the network to learn more complex patterns and make more accurate predictions.

2.3 Activation function

In the same way that our brains receive input from the outside world and use neurons to process that information and make decisions, neural networks take an input, such as images, sounds, or numbers, and use artificial neurons and algorithms to process that information and generate results. The activation of the correct final layer of neurons in a neural network is crucial for producing the desired output [20].

The activation function in a neural network is responsible for determining whether or not a neuron should be activated, based on its input. This means that it uses simple mathematical operations to determine whether the input to the neuron is relevant for making predictions. The activation function also introduces non-linearity to the network, which allows it to generate output from a collection of input values that are fed into a layer [21]. In other words, the activation function is what enables a neural network to learn and make predictions based on the input it receives.

2.3.1 Linear Activation Function

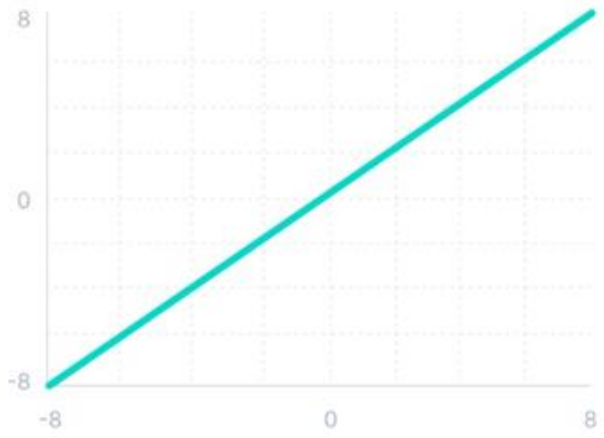


Figure 3 Linear activation function [20].

The linear activation function (Fig 3), also known as the identity activation function, is a type of activation function that is directly proportional to the input it receives. This means that the range of the linear activation function is from negative infinity to infinity. Essentially, the linear activation function simply adds up the weighted total of the inputs it receives and returns that value as the output. It is called a "linear" activation function because it produces a straight line when graphed.

Mathematically it is represented as in equation 2:

$$f(x) = x \tag{2}$$

2.3.2 Binary Step Activation Function

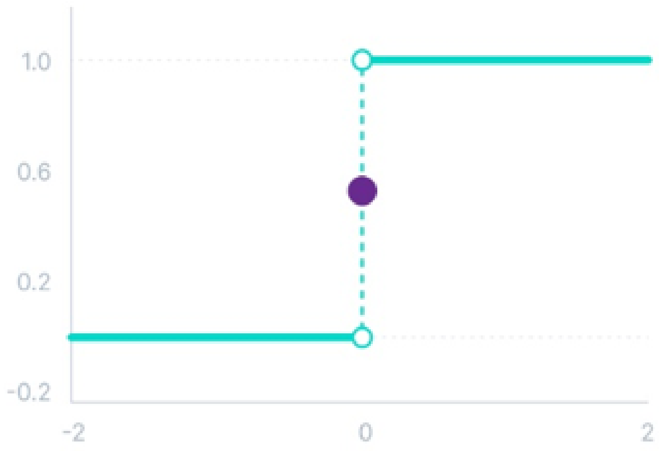


Figure 4 Binary activation function [20].

In a binary step activation function (Fig 4), a threshold value is used to determine whether a neuron should be activated or not. If the input value is greater than the threshold, the neuron is activated, and its output is sent on to the next or hidden layer. However, if the input value is less than the threshold, the neuron is not activated, and its output is not sent on to the next layer.

Mathematically it is represented as equation 3:

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geq 0 \end{cases} \tag{3}$$

2.3.3 Sigmoid

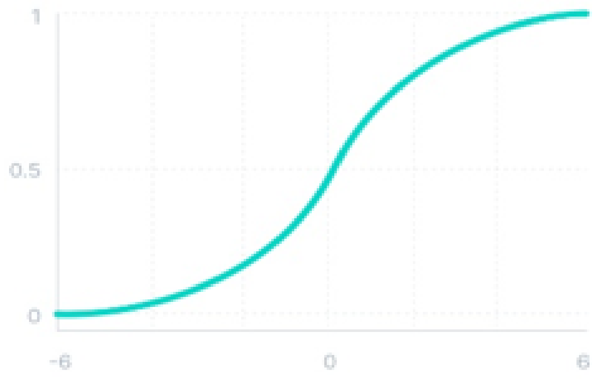


Figure 5 Sigmoid activation function [20].

Sigmoid is a simple activation function (Fig 5) that takes a number as input and returns a value between 0 and 1. This function is easy to use and has several desirable properties, including nonlinearity, continuous differentiation, monotonicity, and a fixed output range. These properties make sigmoid a useful activation function in many machine learning models. Sigmoid is a useful activation function in binary classification problems because it provides a probability of the existence of a particular class.

Mathematically it is represented as equation 4:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

2.3.4 TanH (Hyperbolic Tangent)

TanH (Fig 6) is an activation function that compresses real-valued numbers to the range [-1, 1]. Unlike sigmoid, which maps inputs to a range of [0, 1], TanH maps negative inputs to the range [-1, 0] and positive inputs to the range [0, 1]. This makes the output of TanH zero-centered, which can be advantageous in some situations. TanH is a non-linear function, which means that it can model complex relationships between inputs and outputs. Additionally, the strong negative mapping of TanH means that negative inputs will have a strong impact on the output, while zero inputs will have almost no impact. These properties make TanH a useful activation function in many machine learning models.

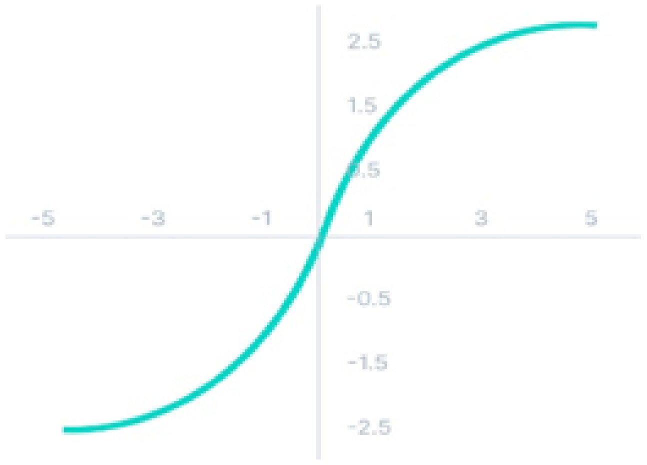


Figure 6 TanH activation function [20].

Mathematically it is represented as equation 5:

$$f(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{5}$$

2.3.5 ReLU (Rectified Linear Unit)

ReLU or Rectified Linear Unit (Fig 7) is one of the most commonly used activation functions in machine learning applications. It is widely used because it solves the problem of vanishing gradients, which can cause training problems in some neural networks. This is because the maximum gradient of a ReLU function is always 1, which means that the function never saturates. Additionally, the slope of a ReLU function is never zero, which means that it does not suffer from the problem of saturating neurons. The range of a ReLU function is between 0 and infinity, which makes it a useful function for modeling many real-world phenomena.

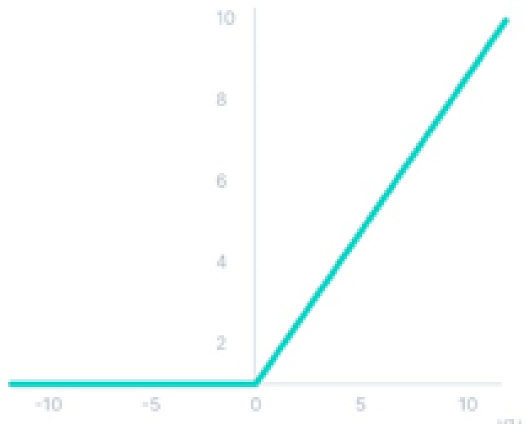


Figure 7 ReLU activation function [20].

Mathematically it is represented as equation 6:

$$f(x) = max\ (0\ ,x) \tag{6}$$

2.3.6 Leaky ReLU

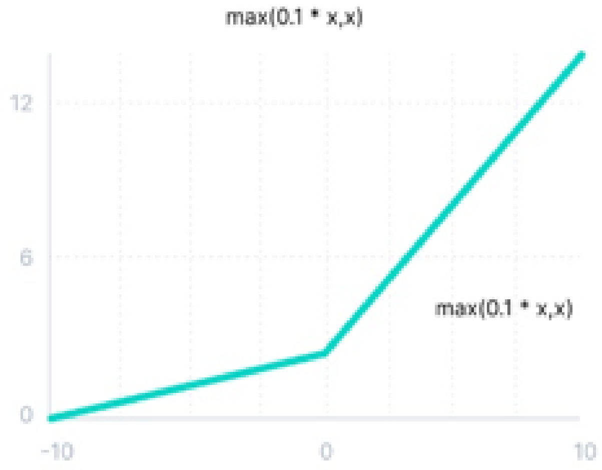


Figure 8 Leaky ReLU function [20].

Leaky ReLU (Fig 8) is a variant of the standard ReLU activation function. Like ReLU, Leaky ReLU is a non-linear function that can be used in machine learning models to introduce non-linearity and improve the model's ability to capture complex relationships between inputs and outputs. The key difference between Leaky ReLU and ReLU is that Leaky ReLU allows a small, non-zero gradient when the input is less than 0. This can help to alleviate the problem of "dying neurons" in which a neuron becomes inactive and ceases to have an impact on the model's output. Leaky ReLU can therefore improve the performance of some neural network models by preventing the vanishing or saturation of gradients.

Mathematically it is represented as equation 7:

$$f(x) = max\ (0.1x, x) \tag{7}$$

2.3.7 ELU (Exponential Linear Units)

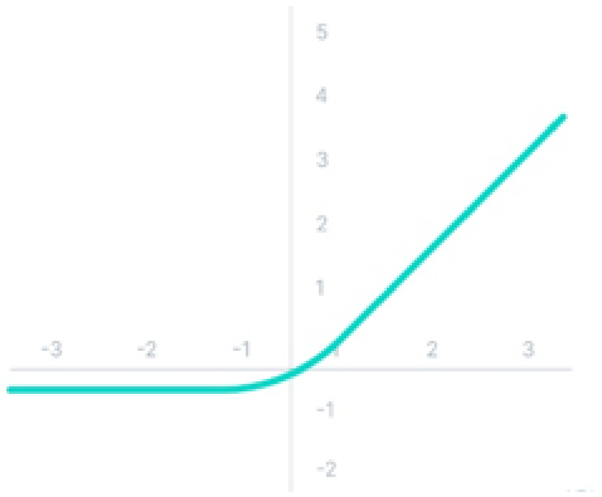


Figure 9 ELU function [20].

Like other activation functions, ELU (Fig 9) introduces non-linearity into the model, which can improve the model's ability to capture complex relationships between inputs and outputs. Unlike other activation functions, ELU uses an exponential function to map inputs to outputs. This means that ELU has a smooth gradient and can therefore improve the convergence of some neural network models. Additionally, ELU has a small negative value for negative inputs, which can help to alleviate the problem of "dying neurons" and improve the performance of the model. Overall, ELU is a useful activation function that can improve the performance of some machine learning models.

Mathematically it is represented as equation 8:

$$f(x) = \begin{cases} x & for\ x \geq 0 \\ \alpha(e^x - 1) & for\ x < 0 \end{cases} \tag{8}$$

2.3.8 Softmax

The Softmax function is a combination of multiple sigmoid functions. It is often used in multi-class classification problems, where it can determine the relative probability of each class. In a neural network, Softmax is typically used in the last layer of the network, where it provides a probability distribution over the possible classes. This means that the output of the Softmax function considers the possibility of each class and provides a probability for the current class relative to the other classes. Overall, Softmax is a useful activation function that can improve the performance of some machine learning models.

Mathematically, it can be represented as equation 9:

$$softmax(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{n} exp\ (x_j)} \tag{9}$$

Where x_i is the input vector of softmax function and x_i is the i^{th} element of input vector.

2.3.9 Swish

One key property of the Swish activation function (Fig 10) is that it allows for the propagation of negative weights, whereas the ReLU activation function sets all non-positive weights to zero. This property is important for the success of smooth, non-monotonic activation functions. Because Swish allows for the propagation of negative weights, it can provide a more accurate mapping of inputs to outputs in some situations, which can improve the performance of the model. Additionally, the smooth gradient of Swish can help to improve the convergence of the model, which can make training more efficient. Overall, the ability of Swish to propagate negative weights is a key property that makes it a useful activation function in some machine learning models.

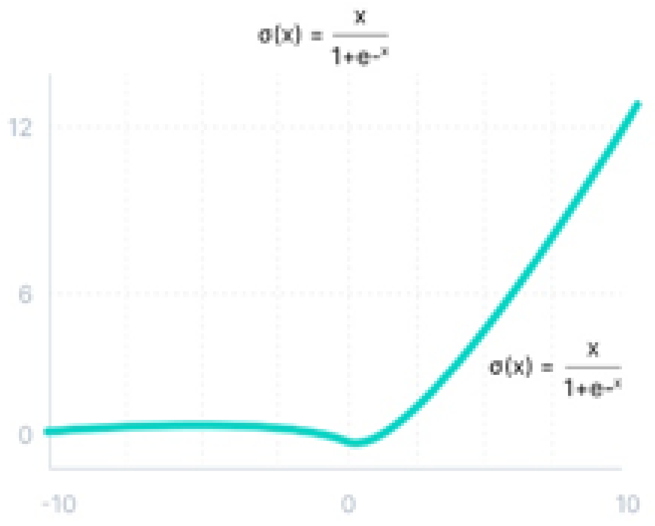


Figure 10 Swish function [20].

Mathematically, it can be represented as equation 10:

$$\sigma(x) = \frac{x}{1 + e^{-x}} \tag{10}$$

The most-used activation functions in artificial neural network models are non-linear activation functions. These functions make it easier for the model to adapt to a variety of data and to differentiate between outputs. Non-linear activation functions also enable the stacking of multiple layers of neurons because the output of these functions is a non-linear combination of inputs that have been passed through multiple layers. This allows any output to be represented as a functional computation output in a neural network. Non-linear activation functions can be divided into different categories based on their range and the shape of their curve.

2.4 Loss Function

In deep learning, a loss function is a measure of how well a machine learning model is able to predict the expected outcome. The loss function calculates the difference between the predicted output of the model and the desired output and provides a way to update the model's weights and biases in order to improve its predictions. The goal of training a deep learning model is to minimize the loss function, so that the model can make more accurate predictions [22].

In supervised learning, there are two main types of loss functions that correspond to the two major types of neural networks: regression and classification loss functions. Regression loss functions are used in regression neural networks, where the model predicts a continuous output value given an input value. Examples of regression loss functions include Mean Squared Error (Eq. 11) and Mean Absolute Error (Eq. 12). Classification loss functions are used in classification neural networks, where the model predicts the probability of an input belonging to one of several pre-defined categories. Examples of classification loss functions include Binary Cross-Entropy loss (Eq. 13) [23].

$$\text{Mean Squared Error } (MSE) = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{11}$$

$$\text{Mean Absolute Error } (MAE) = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{12}$$

$$\text{Cross} - \text{Entropy Loss (CEL)} = -\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right) \tag{13}$$

Where n - Number of training examples. i - ith training example in a data set. y_i - Ground truth label for ith training example. $\hat{y}_i$- Prediction for ith training example.

2.5 Backpropagation

Backpropagation, in short for "backward propagation of errors," is a technique for the supervised training of artificial neural networks using the gradient descent approach. When provided with an artificial neural network and a corresponding error function, this method computes the error function's gradient with respect to the network's weights. The approach optimizes the network weights by reducing the difference between the predicted and actual output. The main steps in this algorithm are [24]:

Forward phase: The input is sent through the network to generate predictions. To generate its output, each neuron computes a weighted sum of its inputs and applies an activation function. Sigmoid, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU) are examples of common activation functions.

Error calculation: Calculate the difference (loss) between the projected and actual output (target). This error aids in the updating of the connection weights. Mean squared error (MSE) and cross-entropy are two often used loss functions for regression and classification problems, respectively.

Backward phase: Propagate the error backward through the network to adjust the weights. The chain rule from calculus is used in this method to compute the error gradient for each weight. The gradients are used to update the weights in the following manner:

a. Calculate the output layer error for each neuron.
b. Calculate the hidden layer errors for each neuron, starting from the last hidden layer and working backward.
c. Compute the gradients of the error with respect to each weight by multiplying the error of the receiving neuron by the output of the sending neuron.
d. Iterate: The forward pass, error calculation, and backward pass steps are repeated for multiple epochs (iterations over the entire dataset) until the error converges to an acceptable level

Iteration: The steps of the forward pass, error estimation, and backward pass are executed multiple times (epochs) over the whole dataset until the error reaches an acceptable threshold or meets a predetermined stopping condition.

2.6 Optimization Algorithms

Optimization algorithms are a type of optimization strategy that improves the performance of a deep learning model. These optimization methods, or optimizers, have a substantial impact on the deep learning model's training speed and accuracy.

To minimize the loss function, the weights of a deep learning model must be adjusted at each epoch during training. An optimizer is a function or algorithm that changes the neural network's properties, such as weights and learning rates, with the goal of lowering total loss and increasing accuracy [25]. Given that deep learning models sometimes involve millions of parameters, choosing the optimal weights for a model can be difficult. This implies choosing the best optimization algorithm for the individual application. Below are commonly used optimization algorithms.

2.6.1 Gradient Descent

Gradient Descent is the most well-known optimizer. This optimization algorithm employs calculus to consistently adjust the parameters and reach the local minimum. Gradient descent begins with some coefficients, calculates their cost, and looks for a cost value that is less than what it is presently. It then switches to the lowest weight and changes the coefficient values. Finally, the method is repeated indefinitely until the local minimum is obtained. A local minimum is a point beyond which no further progress may be made [25].

2.6.2 Stochastic Gradient Descent (SGD)

Instead of using the entire dataset, SGD updates the weights based on the gradient obtained for an individual or a small number of training samples. The technique begins with determining the starting parameters w and learning rate n. Then, in each iteration, randomly shuffle the data to arrive at an estimated minimum. This results in faster convergence and better management of large datasets [25].

2.6.3 Momentum

This approach expedites the convergence of gradient descent by incorporating a momentum component to the weight adjustment, enabling the algorithm to progress more rapidly in the right direction while minimizing fluctuations [25].

2.6.4 Root Mean Square Propagation (RMSprop)

The RMSprop algorithm is an optimization technique with adaptable learning rates. This technique changes the learning rate for each weight separately, increasing the robustness. RMSprop promotes a more effective optimization process in various contexts by catering to gradients with different scales. This makes the algorithm more robust and capable of dealing with diverse gradient scales, resulting in faster and more stable convergence. Because of its simplicity and effectiveness, RMSprop has been widely embraced in the deep learning community. It frequently results in faster convergence and higher performance when compared to ordinary gradient descent, especially when the gradients have varied magnitudes [25].

2.6.5 Adam (Adaptive Moment Estimation)

This optimization methodology combines momentum with adaptive learning rates (similar to RMSprop) to develop an optimization strategy that is appropriate for deep learning problems. The Adam method takes advantage of both momentum and RMSprop, allowing for faster convergence and improved stability during the training phase. It is frequently recommended as the default optimization approach and has been widely used as a benchmark in deep learning research. In addition, Adam is simple to construct, has a faster execution time, requires less memory, and requires less fine-tuning than other optimization methods [25].

2.7 Deep Neural Network Architectures

A wide number of architectures have been developed in the field of deep learning, each suited to solving certain tasks or difficulties. Among the notable examples are:

2.7.1 Convolutional Neural Networks (CNNs)

A CNN (Fig. 11) is a multilayer neural network that was inspired by the animal visual cortex. These networks were created primarily for image processing and computer vision tasks. CNNs are made up of convolutional layers that apply a series of filters to the input in order to detect local patterns in the data. These filters may identify edges, corners, textures, and other elements, which are subsequently integrated into higher-level representations via subsequent layers. CNNs have demonstrated exceptional performance in image classification, object detection, and semantic segmentation [26] [27].

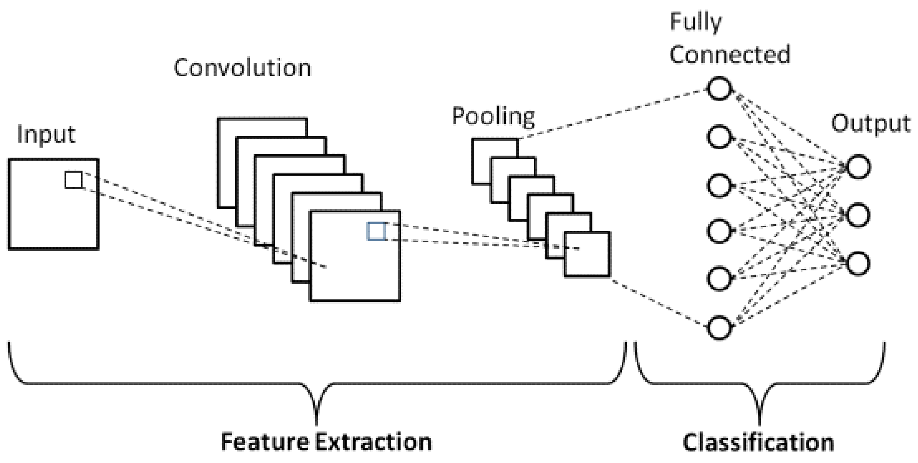


Figure 11 Basic CNN architecture [28].

2.7.2 Recurrent Neural Network (RNN)

RNN serves as a fundamental network architecture upon which other deep learning structures are developed. The main distinction between a standard multilayer network and a recurrent network is the presence of feedback connections in the latter, as opposed to exclusively feed-forward connections. These feedback connections enable RNNs to retain memories of previous inputs, thereby allowing them to model temporal problems effectively [27]. The basic RNN network shown below in Fig. 12.

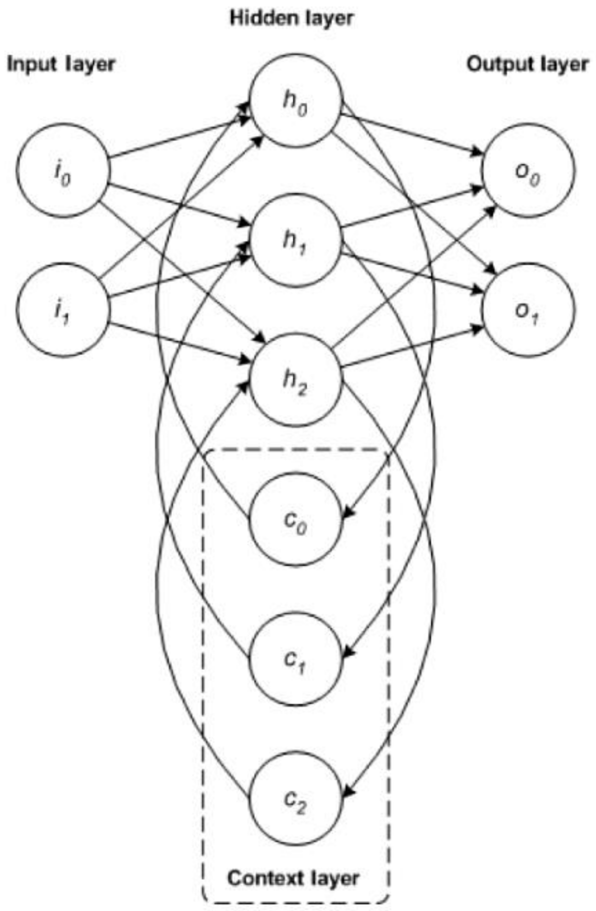


Figure 12 Basic RNN network architecture [27].

2.7.3 Long Short-Term Memory (LSTM) network

LSTM invented by Hochreiter and Schimdhuber in 1997, has grown in prominence as an RNN design for a wide range of applications in recent years. LSTMs are used in common devices such as cellphones, and IBM has used them in their Watson system for ground-breaking conversational voice recognition [27].

By introducing the concept of a memory cell, LSTM deviated from traditional neuron-based neural network designs. This memory cell can keep its value for short or extended periods of time based on its inputs, allowing it to recall important information rather than merely the most recently computed value. The LSTM memory cell is made up of three gates (Fig. 13) that control the flow of data into and out of the cell. The input gate controls when new data can enter the memory, while the forget gate controls when old data is deleted to make place for new data. The output gate determines when the information contained in the cell contributes to the output of the cell. Each gate also has accompanying weights that are controlled during training. These weights are optimized by a popular training procedure, such as Backpropagation Through Time (BPTT), based on the inaccuracy in the network output [27] [29].

2.7.4 Autoencoders

This artificial neural network architecture is made up of three unique layers: the input layer, the hidden layer, and the output layer. Initially, the input layer goes through an encoding procedure, which results in the development of the hidden layer. When compared to the input layer, the hidden layer has a substantially lesser number of nodes. It does, however, function as a compressed representation of the original input. The output layer, on the other hand, is in charge of reconstructing the input layer by implementing a decoder function [29].

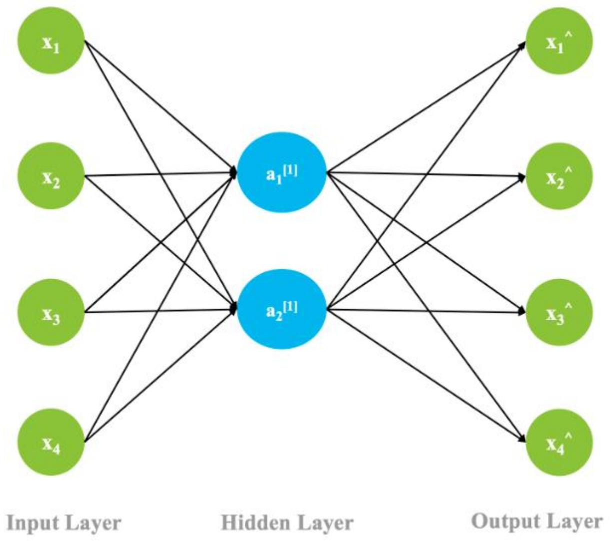


Figure 13 Autoencoders architecture [27].

2.8 Image segmentation

Image segmentation is a technique used to divide a digital image into smaller subgroups called image segments [30]. This process simplifies the image and allows for further processing or analysis of each image segment. In technical terms, segmentation involves assigning labels to pixels to identify objects, people, or other important elements in the image.

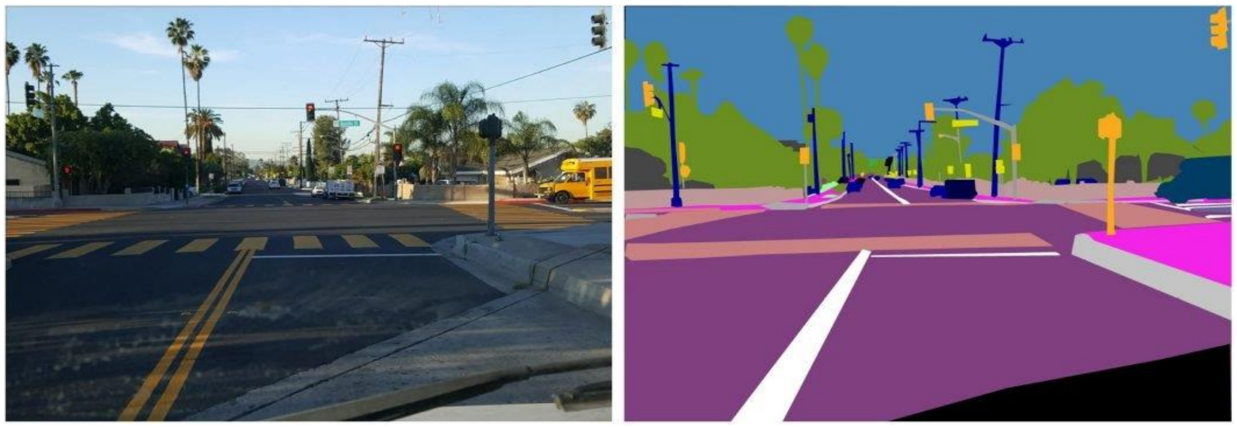


Figure 14 Semantic image segmentation from autonomous vehicle [31].

The image on the left shows ground truth image from camera and on the right annotated image for semantic image segmentation in autonomous vehicle.

Image segmentation has a wide range of applications, including medical image analysis (e.g., extracting tumor boundaries and measuring tissue volumes), autonomous vehicles (Fig. 15) (e.g., detecting navigable surfaces and pedestrians), video surveillance, and augmented reality [31].

2.8.1 Types of image segmentation

Image segmentation can be split into two main categories: instance segmentation, and semantic segmentation. When people think of image segmentation, the latter category is often what comes to mind. It involves identifying, grouping, and labeling the pixels in an image that form a complete object, based on the fundamental definition of image segmentation discussed earlier. Recently one additional type of segmentation was introduced which is called Panoptic Segmentation. It is frequently expressed as a combination of semantic and instance segmentation; this approach involves predicting the identity of each object and separating each instance of each object in the image.

Below, instance segmentation and semantic segmentation will be presented. While this paper is focused on the later one, it will be discussed in detail in the later chapter.

Semantic image segmentation

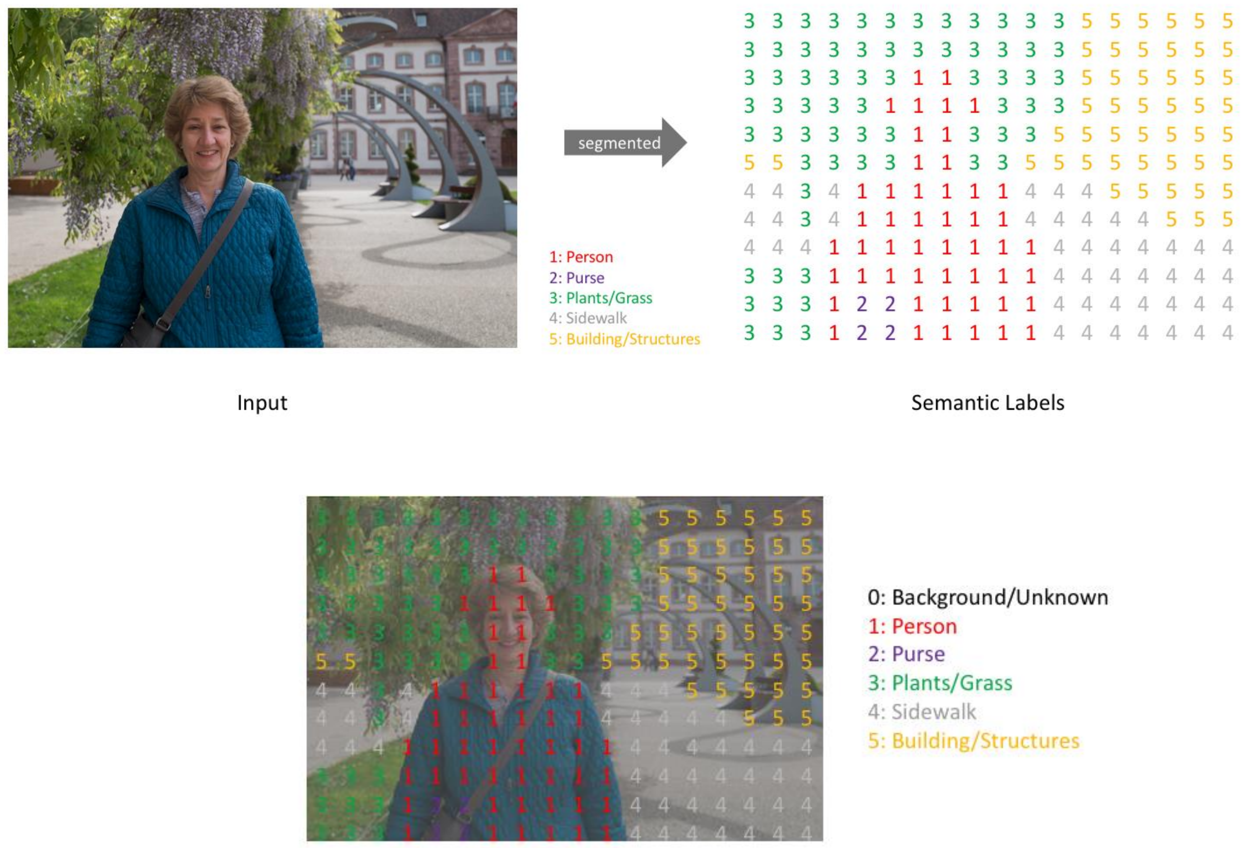
Semantic segmentation is a technique in deep learning that involves assigning a class label to each pixel in an image. It is a form of image segmentation that aims to identify and classify the objects and regions in an image into predefined categories.

For example, in a semantic segmentation task, an image of a cityscape might be labeled with classes such as road, building, sky, and vegetation. Each pixel in the image would be assigned to one of these classes, resulting in a segmentation map that can be used to identify and distinguish different objects and regions in the image.

Semantic segmentation can be performed using a variety of deep learning techniques, including convolutional neural networks (CNNs). In this approach, the input to the network is an image and the output is a map of class labels for each pixel in the image (figure 12) [32].

One of the key challenges in semantic segmentation is to accurately identify and classify objects and regions that may have complex shapes or overlap with other objects. To address this challenge, semantic segmentation models often employ techniques such as skip connections and upsampling to preserve spatial information and improve the resolution of the output segmentation map [33].

Semantic segmentation is commonly used in a variety of applications, such as autonomous vehicle navigation, medical image analysis, and satellite image processing. It is an important tool for understanding and interpreting the content of images and video and can be used to extract useful information for a wide range of tasks. Generally, it is useful to answer the question "What objects or elements are present in this image, and where are they located within the image?" (Figure 13).



```
3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5
3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5
3 3 3 3 3 3 1 1 3 3 3 3 5 5 5 5 5 5
3 3 3 3 3 1 1 1 1 3 3 3 5 5 5 5 5 5
3 3 3 3 3 3 1 1 3 3 3 5 5 5 5 5 5 5
5 5 3 3 3 3 1 1 3 3 5 5 5 5 5 5 5 5
4 4 3 4 1 1 1 1 1 1 4 4 4 5 5 5 5 5
4 4 3 4 1 1 1 1 1 1 4 4 4 4 4 5 5 5
4 4 4 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4
3 3 3 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4
3 3 3 1 2 2 1 1 1 1 1 4 4 4 4 4 4 4
3 3 3 1 2 2 1 1 1 1 1 4 4 4 4 4 4 4
```

Figure 15 Representation of semantic segmentation task [32].

Instance image segmentation

Instance segmentation is a process of classifying pixels based on the specific instances of an object rather than object classes. This type of segmentation algorithm does not identify which class each region belongs to, but rather separates similar or overlapping regions based on the boundaries of objects (Figure 17). For example, if an instance segmentation model processes an image of a crowded street, it should ideally locate and identify the individual objects within the crowd, but it cannot predict the region or object (e.g., a "person") for each instance.

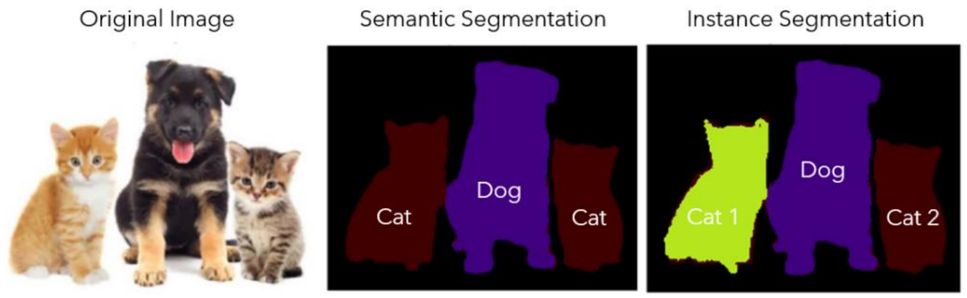


Figure 16 Representation of image segmentations [34].

As shown in the image semantic segmentation can mark out the dog and cats pixels however, there is no indication of how many dogs and cats are there in the image. With Instance Segmentation, each instance is numbered which shows how many cats and dogs are there.

2.8.2 Image segmentation techniques

Image segmentation involves breaking an image into various segments or regions, each corresponding to a different object or background in the image. There are various approaches to achieving image segmentation, including classical and less traditional methods. Some of the most common techniques include region-based segmentation, edge detection segmentation, thresholding, clustering, and Watersheds segmentation [35]. Each of these techniques has its own unique approach to producing the final output of an image or video. Let's examine some of these techniques in more detail.

Region-based segmentation: Region-based segmentation is a technique that involves identifying similarities between the pixels of adjacent segments in an image [36]. This technique takes into account the fact that pixels that are close to each other are more likely to belong to the same object. To determine the boundaries of an object, this technique analyzes the similarities and differences between neighboring pixels. However, one of the limitations of this technique is that lighting and contrast within the image can affect the accuracy of defining the object's parameters.

Edge detection segmentation: Edge detection segmentation is a technique that is designed to address the limitations of region-based techniques by focusing on the edges of objects in an image [35]. This technique involves identifying and classifying certain pixels as "edge pixels" in order to achieve reliable results. Edge detection is particularly useful for images with objects that have well-defined outlines and is relatively simple to implement compared to other techniques that may be more time-consuming.

Thresholding segmentation: Threshold segmentation is an image processing technique that separates regions of interest from the background by comparing the intensity of each pixel to a predefined threshold value. Pixels with intensities equal to or greater than the threshold are in the forefront, while those with lower intensities are in the background, resulting in a binary image. When there is a clear contrast between foreground and background intensities, this technique is useful but it may encounter challenges when intensities overlap, the image has irregular lighting, or noise is present [37].

Cluster-based segmentation Unsupervised classification algorithms known as clustering algorithms can be used to uncover hidden information in images. These algorithms work by dividing images into groups of pixels that have similar features, helping to identify patterns and structures in the data. Clustering algorithms can aid human vision by isolating clusters and shading within an image. They do this by grouping similar elements together into clusters, which allows for the separation of data elements [36].

Watersheds segmentation: Watersheds involve the transformation of a grayscale image. These algorithms operate by treating images as if they are topographic maps, with pixel brightness representing elevation [37]. The watershed segmentation process involves the detection of lines that form ridges and basins, which are used to mark the boundaries between different regions in the image. This technique divides an image into multiple regions based on the gray value of the pixels, grouping pixels with similar brightness together.

Chapter 3

State of the art

This section illustrates the related work of semantic segmentation using LIDAR and camera data. Cameras and LIDARs are the two main sensors used in autonomous cars for semantic segmentation. Both camera-based and LIDAR-based semantic segmentation use various techniques and strategies. In order to compare camera and LIDAR semantic segmentation for autonomous vehicles, it is necessary to assess how well each modality performs when it comes to correctly identifying and categorizing things in the environment. Whereas LIDAR-based semantic segmentation makes use of point cloud data gathered by LIDAR sensors, camera-based semantic segmentation depends on visual data obtained by cameras [1].

When trained on big datasets, recent research has demonstrated that camera-based semantic segmentation can achieve excellent accuracy, but it may struggle in poor illumination and weather situations. On the other side, LIDAR-based semantic segmentation can work well in low light and bad weather, but it might have trouble distinguishing thin structures or certain item types. To comprehend their strengths and shortcomings and choose the optimal strategy for particular cases, it is crucial to compare these two semantic segmentation methods.

Segmentation can be broadly categorized into traditional and deep neural network (DNN) methods [38]. Hand-crafted features and mathematical operations such as thresholding, clustering, and edge detection are commonly used in traditional image segmentation methods. These methods frequently necessitate expert knowledge and manual parameter tuning, and they may not generalize well to new data or complex scenes.

Deep neural network (DNN) segmentation methods, on the other hand, use neural networks to learn features and representations from input images automatically. These methods have achieved cutting-edge performance on a variety of segmentation tasks, including semantic, instance, and panoptic segmentation. The accuracy of segmentation has increased significantly since the re-emergence of DNN (Deep Neural Network). This paper focuses primarily on DNN methods for semantic segmentation using camera and LIDAR data.

3.1 Traditional segmentation method

Prior to the development of Deep Neural Networks (DNN), the most important areas in computer vision and image processing were thought to be feature extraction and classification. A feature in this context refers to a piece of information that is required to solve computational tasks, similar to the concept of feature in machine learning and pattern recognition. For semantic segmentation, a variety of features have been used, including pixel color and Histogram of oriented gradients [38].

Thresholding is a common traditional method that is especially effective for grayscale images. Grayscale images are frequently used in the medical field, where imaging equipment such as X-ray CT scanners or MRI (Magnetic Resonance Imaging) machines are used. Several studies [39] [40] have demonstrated that thresholding methods can be quite effective in this context.

K-means clustering is an unsupervised clustering method in which each data point is assigned to one of a predefined number of clusters. The algorithm begins by randomly placing K centroids in the feature space and requires the number of clusters to be specified beforehand. Each data point is then assigned to the nearest centroid, which is then moved to the cluster's center. The procedure is repeated until a stopping criterion is met [41].

Edge detection is a traditional method for semantic image segmentation. It entails detecting the boundaries of objects in an image by locating the edges, or areas of high contrast, between various regions. While edge detection can be a useful pre-processing step for object detection and segmentation, its ability to accurately segment complex objects or scenes with overlapping edges is frequently limited. To achieve more robust and accurate segmentation results, edge detection is typically used in conjunction with other segmentation methods such as region growing or clustering [42].

MRFs (Markov Random Fields) are widely used in image processing and computer vision applications, including semantic segmentation. MRFs provide a principled probabilistic framework for modeling the spatial relationships between neighboring pixels, which can help to improve segmentation accuracy and robustness. Let x represent the input and y represent the output. MRF learns the P(y, x) distribution [38].

3.2 DNN segmentation method

DNNs are a type of artificial neural network (ANN) composed of many layers of interconnected neurons, each of which performs a linear or nonlinear transformation on its inputs. ANNs can be configured in a variety of ways to perform specific tasks. Auto-encoders, Restricted Boltzmann Machines (RBMs), Recurrent/Recursive Neural Networks (RNNs), Convolutional Neural Networks, Long Short-Term Memory (LSTM) networks, and other types of ANNs can be created through various neuron stacking [43]. DNNs are distinguished by their ability to learn complex and abstract data representations using multiple layers of nonlinear transformations.

DNNs are used in semantic segmentation to directly learn features and representations from input images, as well as to generate pixel-wise predictions of object labels or segmentation masks. When compared to traditional methods that rely on hand-crafted features and mathematical operations, the use of DNNs has enabled significant improvements in segmentation accuracy.

DNN-based semantic segmentation methods have grown in popularity in recent years, owing to the availability of large-scale datasets and powerful computing resources, which have enabled deep and complex networks to be trained on massive amounts of data.

As stated earlier in the chapter, this section is devoted to the current state of the art in DNN-based semantic segmentation. To achieve the goal of this paper, the segmentation approaches will be divided into camera-based and LIDAR-based semantic segmentation, which will be discussed in detail in the following section.

It should be noted that this does not imply that the methods listed in camera-based semantic segmentation are not applicable in LIDAR-based semantic segmentation. These methods can be used interchangeably depending on the problem's requirements and needs. Here they are presented based on their common usage.

3.3 Camera-based Semantic Segmentation

Cameras are inexpensive and provide a wealth of visual data in the form of high-resolution images. Deep learning techniques, particularly Convolutional Neural Networks, have dominated recent advances in camera-based semantic segmentation (CNNs). There have been several prominent models proposed. Deep learning-based methods have shown great success in camera-based semantic segmentation tasks, such as Fully Convolutional Networks (FCNs) [44], U-Net [45], and DeepLab [46]. These methods rely on the spatial and appearance information in the images to distinguish between different object classes. Camera-based systems, on the other hand, have limitations in terms of their sensitivity to lighting conditions, such as shadows, glare, or low-light environments, which may affect semantic segmentation accuracy [43] .

3.3.1 Fully Convolutional Networks (FCNs)

Long et al. [44] introduced FCNs, which marked a significant advancement in semantic segmentation. The authors proposed an end-to-end trainable CNN that can handle any size input image and produce dense pixel-wise output. Their approach's key innovation was to replace fully connected layers in neural networks with fully convolutional layers. The network was able to maintain the same input and output sizes by including interpolation layers, which is critical for accurate segmentation. The authors also used skip connections to collect multi-scale data and refine the segmentation output. Another significant aspect of their work was the network's end-to-end training, which allowed it to handle arbitrary-sized inputs and produce correspondingly sized outputs with efficient inference and learning.

At the time of their introduction, Fully Convolutional Networks (FCNs) were implemented in VGG-Net and achieved state-of-the-art semantic segmentation results on the PASCAL VOC dataset. The approach increased mean Intersection over Union (IU) performance by 20% compared to 2012, resulting in a mean IU of 62.2%. For a typical image, the inference time was less than one fifth of a second [38]. Figure 32 depicts the main architecture of the approach.

FCNs have been successfully applied to a range of image segmentation tasks, such as segmenting brain tumors [47], performing instance-aware semantic segmentation [48], identifying skin lesions [49] , and segmenting irises [50]. While these models have demonstrated the ability of deep neural networks (DNNs) to perform semantic segmentation in a fully automatic way on images of varying

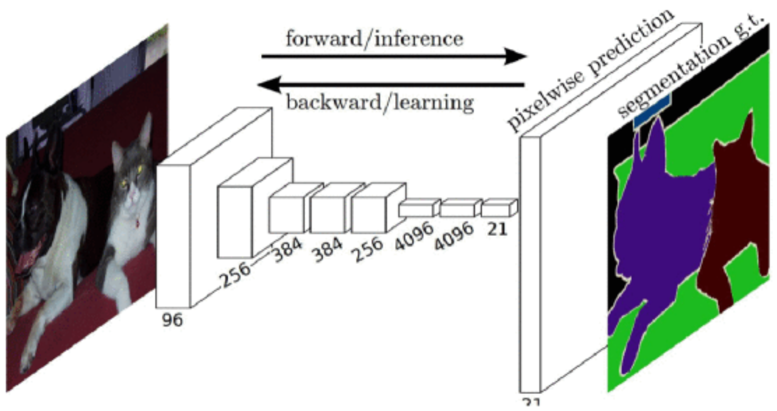


Figure 17 Architecture of Fully convolutional network [44].

sizes, they do have some limitations. For example, they can be computationally expensive for real-time inference, they do not effectively incorporate global context information, and they are not easily adaptable to 3D images. To address some of these limitations, researchers have proposed various modifications to the basic FCN model. For instance, Liu et al. [51]introduced ParseNet which enhances FCNs by adding global context using a context vector. This vector is created by pooling the feature map for a layer over the entire image, normalizing it, and then unpooling it to produce new feature maps of the same size as the original ones. These new feature maps are concatenated with the output of the FCN's convolutional layers, effectively replacing the convolutional layers in the FCN model with the described module.

Deconvolution layers have been used in semantic segmentation alongside the Fully Convolutional Network (FCN) architecture. A deconvolution network consisting of deconvolution and un-pooling layers was used in the approach proposed by Noh et al. [52]. This network was used to identify pixel-level, class labels and predict segmentation masks. Unlike the FCN architecture proposed by Long et al. [44], the network in Noh et al. [52] was applied to individual object proposals to obtain instance-wise segmentations, which were then combined for the final semantic segmentation.

3.3.2 Pyramid Scene Parsing Network (PSPNet)

Hengshuang Zhao et al. [53] introduced PSPNet (Pyramid Scene Parsing Network), a deep learning-based semantic segmentation approach, in 2017. The main idea behind PSPNet is to use global context information in addition to local context to improve semantic segmentation accuracy. To accomplish this, the authors proposed a pyramid pooling module that captures global context at multiple scales, as well as a skip architecture that combines low- and high-level features to improve segmentation accuracy.

The pyramid pooling module (Fig. 33) in PSPNet consists of four convolutional components, each of which processes the feature map produced by the model's convolutional layers in a different way. The first component produces a single bin output, while the other three divide the feature map into different regions and form pooled representations for different locations. These outputs are then up sampled to the same size and concatenated to form the final feature representation. Because the convolutional network uses filters of different sizes (i.e., 1×1, 2×2, 3×3, and 6×6), it is able to extract both local and global context information [53]. The concatenated, up-sampled result from the pyramid module is then processed by the rest of the CNN to produce the final prediction map.

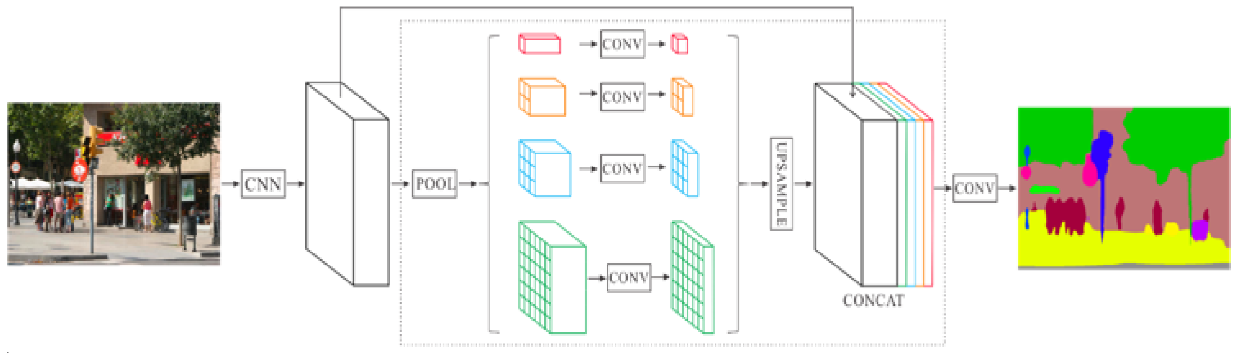


Figure 18 PSPNet with Pooling Module [53].

PSPNet employs a skip architecture in addition to the pyramid pooling module to integrate low-level and high-level features for more accurate segmentation. To capture both low-level details and high-level semantic information, the network combines feature maps from earlier layers with feature maps from later layers.

PSPNet has been used as a baseline architecture for many semantic segmentation tasks since its introduction and has been improved in a variety of ways. To improve segmentation accuracy, Fu et al. [54] proposed a dual attention network that incorporates spatial and channel attention mechanisms into PSPNet. Qin et al. [55] used a similar pyramid pooling module in their BASNet salient object detection approach, which produced state-of-the-art results on several benchmark datasets. Li et al. [56]proposed DFANet, a real-time semantic segmentation approach that uses a modified version of the PSPNet pyramid pooling module for feature aggregation.

3.3.3 U-Net

U-Net is a well-known convolutional neural network architecture that was introduced in 2015 for medical image segmentation [45]. The main idea behind U-Net is to combine low-level and high-level features using an encoder-decoder architecture with skip connections for improved segmentation accuracy.

Ronneberger introduced the U-Net (Fig.34) for effectively segmenting biological microscopy images. The U-Net architecture consists of two parts: the left part - a contracting path to capture context and the left part - a symmetric expanding path that allows for precise localization. The use of the contracting path is to capture context, on the other hand the role of the expansive path is to aid in precise localization [57].

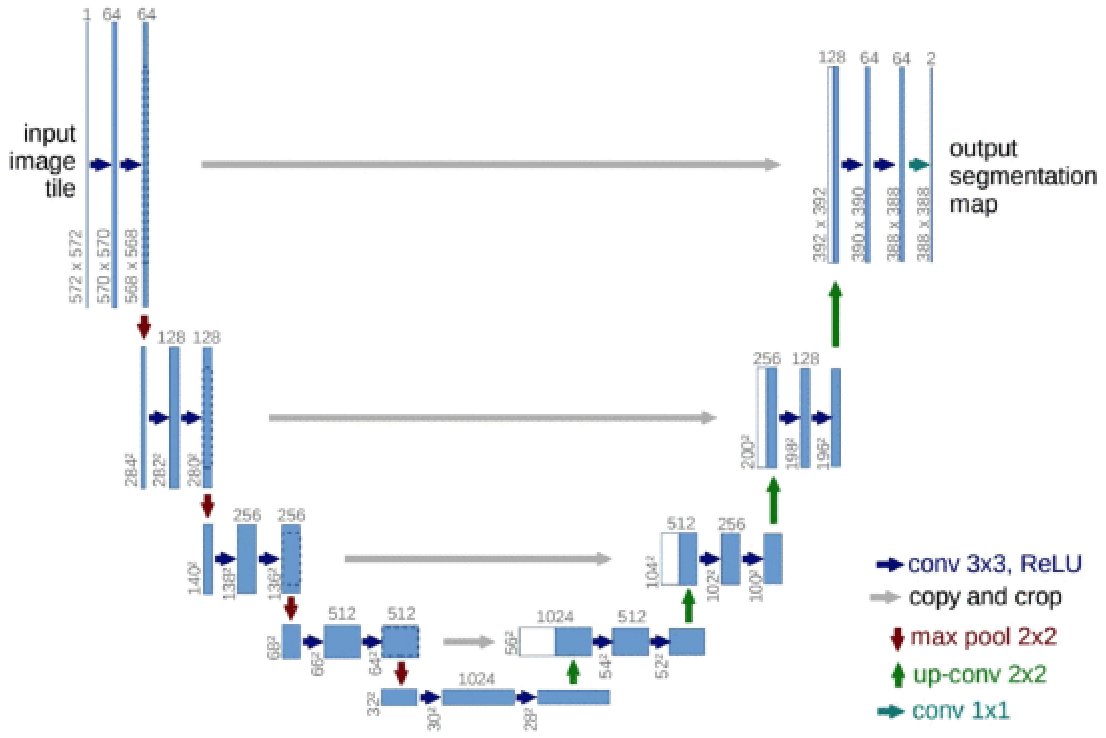


Figure 19 The U-net model [45].

U-Net is widely used in semantic segmentation applications such as medical image segmentation, road segmentation, and building segmentation. Many variations and improvements to the U-Net architecture have been proposed since its introduction. Ronneberger et al. [45], for example, used a 3x3 convolution kernel for all convolutional layers, but subsequent work has investigated the use of other kernel sizes and architectures, such as dilated convolutions [58] and attention mechanisms [59]. Some works, such as the Mobile-UNet architecture [60], have also focused on reducing the number of parameters in the network while maintaining segmentation accuracy.

Yu et al. [58] proposed Dilated Residual Networks. This paper investigates the use of dilated convolutions in the U-Net architecture to broaden the network's receptive field and capture more spatial context. The authors demonstrate the effectiveness of this approach on the Cityscapes dataset for the task of semantic segmentation, achieving state-of-the-art performance at the time of publication.

Schlemper et al. [59] published "Attention U-Net: Learning Where to Look for the Pancreas." Attention U-Net, a variant of U-Net that introduces attention gates to selectively propagate informative features for improved segmentation accuracy, is proposed in this paper. The authors demonstrate the utility of Attention U-Net for segmenting the pancreas in abdominal CT scans.

Howard et al. [60] published "Mobile-UNet: Efficient Convolutional Neural Network for Mobile Vision Applications." This paper proposes Mobile-UNet, a U-Net variant designed to be computationally efficient and well-suited for deployment on mobile devices. The authors accomplish this by reducing the number of network parameters while maintaining segmentation accuracy, allowing the network to be trained and deployed more efficiently.

3.3.4 DeepLab

Google developed DeepLab, a neural network architecture that is based on convolutional neural networks (CNNs). Unlike U-net, which combines features from every convolutional block with the corresponding deconvolutional block, DeepLab uses only the features generated by the last convolutional block before upsampling, similar to the approach used by FCN. DeepLab also utilizes Atrous convolution for upsampling [46].

The first version of DeepLab is DeepLab V1 which addresses issues related to reduced spatial resolution and localization accuracy in DCNN. This is achieved by eliminating down-sampling operators in the final max-pooling layers, introducing Atrous convolutions, and using a Conditional Random Field (CRF) to smooth labels and capture global contextual relationships between object classes [33].

To improve the representation of things at different scales, the DeepLab V1 was enhanced by adding Atrous Spatial Pyramid Pooling. This extension of the pyramid pooling module used in PSPNet helps in producing more accurate semantic image segmentation at multiple scales. DeepLab V2 builds on this technique by incorporating Atrous spatial pyramid pooling, which boosts its performance [33] [46].

Later DeepLab V3 [61] was created to enhance the ability of semantic image segmentation to capture precise object boundaries. To achieve this, an encoder-decoder architecture with Atrous convolution was utilized [62]. Additionally, DeepLab V3 includes depth-wise separable convolution in its encoder-decoder network to enhance computational efficiency.

Currently DeepLab V3+ is the latest version of this approach and it has become a popular deep learning architecture for semantic segmentation in computer vision tasks. Chen et al. [63] introduced this architecture in 2018 and expands on previous DeepLab models by incorporating an encoder-decoder structure with atrous spatial pyramid pooling (ASPP) modules. The ASPP modules allow the network to capture multi-scale features and effectively handle images of varying resolutions. DeepLab V3+ utilizes a novel decoder module defined as the "decoder refinement module," which refines the encoder's feature maps to produce more accurate segmentation results [63].

DeepLab V3+ also employs a feature pyramid network (FPN) to combine features from various scales and resolutions, allowing the network to capture both fine-grained details and high-level semantic information. The FPN is integrated with the ASPP module, which improves the network's ability to capture multi-scale data [64].

Recent research has proposed several enhancements to DeepLab V3+ in order to improve its performance even further. For example, Li et al. [65] proposed a "inverted residuals with linear bottleneck" (IRB) dilation scheme that reduces the number of parameters while maintaining accuracy. Meanwhile, Zhang et al. [66] presented a modified version of DeepLab V3+ that uses an attention mechanism known as "channel attention module" (CAM) to capture more informative features.

DeepLab V3+ has demonstrated cutting-edge performance on a variety of benchmark datasets, including PASCAL VOC 2012, Cityscapes, and ADE20K. It's also been used in real-world applications like medical image segmentation, self-driving cars, and remote sensing [63]. In this paper, the DeepLab V3+ model is used for semantic segmentation of the autonomous vehicle using camera data and it will be discussed in detail in the later chapter.

3.4 LIDAR-based semantic segmentation

LIDAR sensors generate precise 3D point cloud data for semantic segmentation. Deep learning-based approaches for LIDAR-based semantic segmentation have gained popularity in recent years. FCNs have also been applied to LIDAR point clouds, where the network learns to predict a dense output of semantic labels for each point in the point cloud. Other deep learning-based models proposed for LIDAR-based semantic segmentation include SqueezeSeg [67] [68], PointNet++ [69], and FIDNet [70]. These models can handle the irregularity and sparsity of point cloud data and have demonstrated promising results in a variety of autonomous driving applications. However, LIDAR-based semantic segmentation also has limitations, such as the limited range of LIDAR sensors and their sensitivity to occlusions caused by objects in the scene [70].

3.4.1 PointNet++

PointNet++ [69] is a well-known deep learning-based model for semantic segmentation using LIDAR data. It is an extension of the original PointNet model, which is intended to deal with unordered point cloud data. PointNet++ introduces a hierarchical neural network architecture capable of capturing local and global features from point clouds, allowing it to handle larger and more complex datasets.

The original PointNet learns global point cloud features by applying a series of fully connected layers and max-pooling operations to unordered point clouds. However, because it does not take into account local context, it is limited in capturing local structures. PointNet++ overcomes this limitation by employing a hierarchical learning strategy. PointNet++'s main components are:

- Set Abstraction (SA) Layer: The SA layer divides the input point cloud into overlapping local regions, which are then processed by a smaller PointNet to learn local features. This process is repeated at various scales to capture both local and global features.
- Feature Propagation (FP) Layer: The FP layer propagates the features learned at each level of the hierarchy back to the original point cloud, allowing the model to refine its predictions using a combination of local and global features.

PointNet++ represents a significant advancement in 3D point cloud processing. It effectively addresses the limitations of the original PointNet and outperforms it on various benchmarks by incorporating a hierarchical learning approach. PointNet++ lays the groundwork for future research in 3D point cloud analysis and its applications in a variety of domains.

3.4.2 FIDNet

FIDNet [70] is a deep learning-based model designed for semantic segmentation of LiDAR point clouds in autonomous driving scenarios. It is made up of two main parts: a PointNet-based encoder network and a fully interpolation decoding network. The encoder network extracts point-wise features from the raw LiDAR point cloud data, and the decoding network restores the feature maps to their original resolution using a novel fully interpolation scheme.

FIDNet uses fully interpolation decoding, which allows the network to predict semantic labels for all points in the point cloud, is a key innovation [70]. The fully interpolation decoding approach enables FIDNet to achieve higher accuracy and better preserve the fine-grained details of the input point cloud.

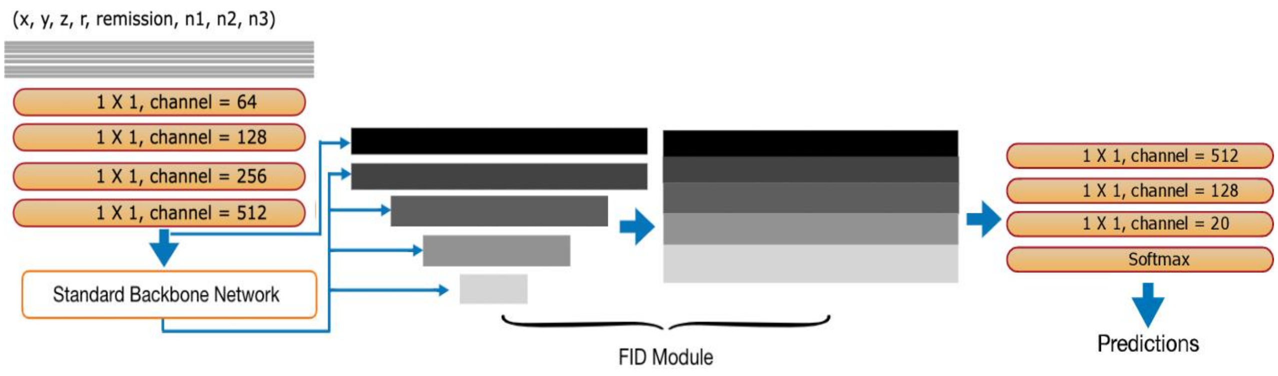


Figure 20 Network structure of FIDNet [70].

As shown in figure 35, the input module consists of two 1×1 layers that map individual points to a high-dimensional space. For the backbone, a regular standard network like ResNet-34 can be used. The FID module, which is responsible for upsampling all low-resolution feature maps to their original size, combines the resulting feature maps by concatenating them. Finally, the last classification head receives the merged large tensor as input and produces a label for each point.

On several benchmark datasets, including SemanticKITTI, the FIDNet model outperformed state-of-the-art methods in terms of segmentation accuracy and efficiency. FIDNet is an efficient and lightweight architecture, which makes it suitable for real-time applications, and is one of its key strengths. Furthermore, FIDNet is intended to be adaptable to various LiDAR sensors and configurations.

3.4.3 SqueezeSeg

SqueezeSeg is a deep learning-based model family designed for semantic point cloud segmentation in autonomous driving scenarios. The main idea behind SqueezeSeg is to use the sparsity and irregularity of LiDAR point cloud data to reduce the number of parameters and computational complexity of the model [67] [68]. SqueezeSeg has been proposed in several versions, each with its own distinct architecture and performance characteristics.

Wu et al. (2018) [67] proposed the first version of SqueezeSeg, which is a single-stage fully convolutional network (FCN) architecture. The network begins with raw LiDAR point cloud data and predicts a dense output of semantic labels for each point in the point cloud. The model achieves cutting-edge performance on the KITTI dataset, which serves as a benchmark dataset for LiDAR-based semantic segmentation.

Wu et al. (2019) [68] proposed SqueezeSegV2, which extends the original SqueezeSeg architecture with several key improvements. The use of dilated convolutions, which allows the network to capture multi-scale contextual information without increasing the number of parameters, is the main improvement. The network also includes skip connections between the encoder and decoder, allowing it to use both low-level and high-level features to achieve more accurate segmentation. On both the KITTI and SemanticKITTI datasets, SqueezeSegV2 outperforms the original SqueezeSeg.

Xu et al. (2020) [70] proposed SqueezeSegV3, which improves the SqueezeSegV2 architecture with several novel techniques. The use of a dynamic convolutional layer, which adaptively adjusts the kernel size and stride of the convolution operation based on the local density of LiDAR points, is the main improvement. This enables the network to collect more accurate and detailed data in areas with a high point density. SqueezeSegV3 also includes a modified loss function and a multi-scale feature fusion module, among other improvements. On the SemanticKITTI dataset, SqueezeSegV3 achieves state-of-the-art performance.

In general, the SqueezeSeg family of models has performed great success in LiDAR-based semantic segmentation tasks, with each version improving on the previous one. Because the models are efficient and lightweight, they are well suited for real-time applications in autonomous driving scenarios. SqueezeSeg V2 model is used in this paper for semantic segmentation of autonomous vehicles using LIDAR data and it will be presented in detail in the next chapter.

Chapter 4

SqueezeSeg V2 and DeepLab V3+

SqueezeSegV2 and DeepLabV3+ are both deep learning models designed for semantic segmentation tasks, but they target different types of data and employ different techniques. In SqueezeSegV2, semantic segmentation works by converting LiDAR point cloud data into a 2D grid and using a CNN with a context aggregation module to predict point-wise class labels. In DeepLabV3+, semantic segmentation is performed by employing an encoder-decoder structure with atrous convolutions and spatial pyramid pooling, capturing multi-scale context, and generating pixel-wise class labels for images. In this section the details of both structures will be discussed.

4.1 SqueezeSegV2

Based on convolutional neural networks (CNN), in SqueezeSeg the CNN accepts a converted LiDAR point cloud as input and directly outputs a point-wise label map, which is subsequently improved by a conditional random field (CRF) implemented as a recurrent layer. Conventional clustering algorithms are then used to get instance-level labels. To feed 3D point clouds to a 2D CNN, a spherical projection is used to convert sparse, unevenly distributed 3D point clouds to dense, 2D grid representations [67]. The model architecture of SqueezeSeg V2 is illustrated below in Fig.28.

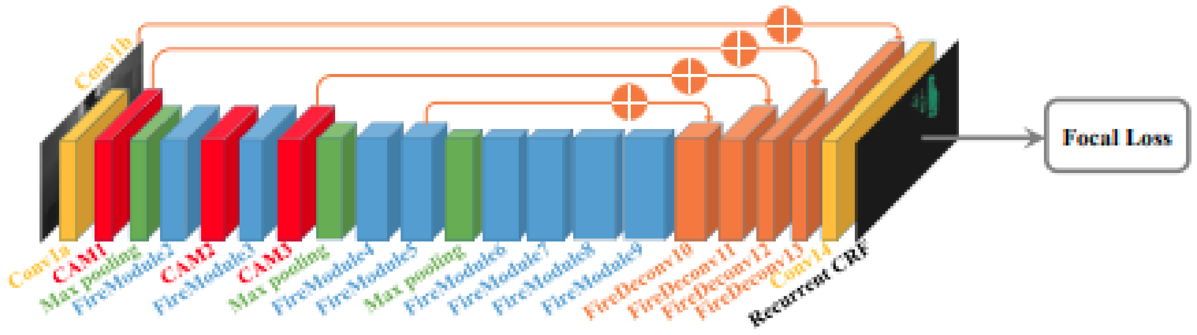


Figure 21 SqueezeSegV2 model for road-object segmentation from 3D LiDAR point clouds [68].

4.1.1 Point Cloud Transformation

Traditional CNN models operate on images that can be represented as 3-dimensional tensors with H × W × 3 dimensions. The first two dimensions, H and W, correspond to the spatial position of the image's height and width. The last dimension is often used to express features such as RGB values. A 3D LIDAR point cloud, on the other hand, is composed of Cartesian coordinates (x, y, z) and may include extra properties such as intensity or RGB values. In contrast to picture pixels, which have uniform distributions, LiDAR point clouds have sparse and uneven distributions. Converting 3D space to voxels results in a high number of empty voxels. Handling such sparse data is inefficient and wastes computer resources [67].

To achieve a more efficient representation, the LiDAR point cloud is projected onto a sphere, resulting in a densely arranged grid-based format as depicted below (Eq. 14).

$$\theta = \arcsin\frac{z}{\sqrt{x^2+y^2+z^2}}, \tilde{\theta} = [\theta/\Delta\theta]$$
$$\varphi = \arcsin\frac{y}{\sqrt{x^2+y^2}}, \tilde{\varphi} = [\varphi/\Delta\varphi] \tag{14}$$

Where, φ and θ are *azimuth and zenith* angles, as shown in Fig. 29 (A). $\Delta\theta$ and $\Delta\varphi$ are resolutions for discretization and (θ, φ) denotes the position of a point on a 2D spherical grid. It is possible to obtain a 3D tensor with dimensions H ×W×C by using equation (14) for each point in the cloud. Fig. 29 illustrates example of point cloud projection to 2D and comparison to camera view.

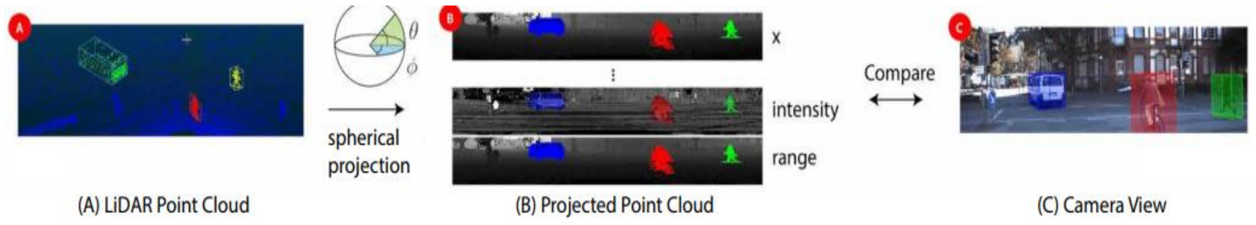


Figure 22 LiDAR Projections [67].

4.1.2 Network structure

The main components of the SqueezeSegV2 structure are as follows:

I. **Input Layer:**

A LiDAR point cloud, which is a collection of 3D points representing the environment, is received by the input layer. Each point has unique characteristics such as distance, intensity, and laser return values. To efficiently process this data, the point cloud is turned into a 2D grid representation using spherical projection. The matching point characteristics are filled into the grid cells, and empty cells can be padded with zero values or masked out.

II. **SqueezeNet Backbone:**

SqueezeNet [71] is a lightweight CNN architecture that serves as the SqueezeSegV2 backbone. It is incorporated with a set of *FireModules* and *FireDeconvs* that are designed to be memory and computing resource efficient. The structure of each Fire module is as follows:

- Squeeze layer: A convolutional layer with 1×1 filters that lowers the number of input feature maps' channels. This layer aids in the compression of the feature representation, resulting in a reduction in the number of parameters in the network.

- Expand layer: There are two sets of convolutional filters (1×1 and 3×3) in this layer. The squeeze layer's output feature maps are run through both sets of filters, and the outputs are concatenated channel-wise. This topology allows the network to collect both local and global data.

III. Context Aggregation Module (CAM)

Dropout noise is caused by several missing points in LiDAR point cloud data. Dropout noise has a considerable impact on SqueezeSeg, particularly in early network levels. Missing points in a local neighborhood can significantly affect the filter's output at early stages because the receptive field of the convolution filter is very small [68].

To address this issue, Context Aggregation Module (CAM) is used which reduces sensitivity to dropout noise. As illustrated in Fig. 30, CAM begins with max pooling and rather large kernel size. With a substantially broader receptive field, max pooling accumulates contextual information around a pixel and is less sensitive to missing data inside its receptive field. Furthermore, even with a larger kernel size, max pooling can be computed effectively. The max pooling layer is then followed by two cascaded convolution layers separated by a ReLU activation. Sigmoid function is used to normalize the module's output and element-wise multiplication to merge it with the input [68].

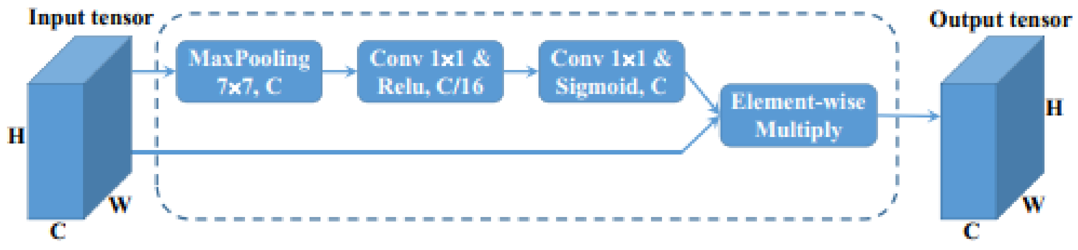


Figure 23 Structure of Context Aggregation Module [68].

IV. Upsampling Layers:

Following the SqueezeNet backbone and CAM, the model employs a succession of deconvolution (*FireDeconvs*) layers to upsample the feature maps. These layers increase the spatial resolution of the feature maps, allowing the model to generate per-point segmentation predictions. The upsampling layers can also be paired with skip connections from earlier layers in the network, which helps to keep fine-grained features in the output segmentation.

V. Output Layer / Conditional random field (CRF)

Label maps predicted by CNN models have indistinct boundaries when it comes to image segmentation. This is due to the loss of low-level features in downsampling techniques such as max pooling. SqueezeSeg model exhibits a similar effect.

To refine the label map created by CNN, a CRF is used [46]. A CRF model employs the energy function (Eq. 15) for a given point cloud and a label prediction $\boldsymbol{c}$, where c_i is the projected label of the *i-th* point.

$$E(\boldsymbol{c}) = \sum_i u_i(c_i) + \sum_{i,j} b_{i,j}(c_i, c_j) \tag{15}$$

The unary potential expression $u_i(c_i) = logP(c_i)$ considers the CNN classifier's predicted probability $p(c_i)$. The binary potential terms determine the "penalty" for labeling a pair of similar locations differently and are defined as $b_{i,j}(c_i, c_j) = \mu(c_i, c_j) \sum_{m=1}^{M} w_m k^m(f_i, f_j)$ where $\mu(c_i, c_j) = 1$ if $c_i \neq c_j$ and 0 otherwise, k^m is the *m-th* Gaussian kernel (Eq.16) that depends on features $\boldsymbol{f}$ of point *i* and *j*, and w_m is the corresponding coefficient. Here, 2 Gaussian kernels are used.

$$w_1 \exp\left(-\frac{\left\|\boldsymbol{P_i} - \boldsymbol{P_j}\right\|^2}{2\sigma_\alpha^2} - \frac{\left\|\boldsymbol{X_i} - \boldsymbol{X_j}\right\|^2}{2\sigma_\beta^2}\right) + w_2 \exp\left(-\frac{\left\|\boldsymbol{P_i} - \boldsymbol{P_j}\right\|^2}{2\sigma_\gamma^2}\right) \tag{16}$$

The first term is influenced by both the angular position $P(\tilde{\theta}, \tilde{\varphi})$ and the Cartesian coordinates $X(x, y, z)$ of two points. In contrast, the second term relies solely on the angular positions. $\sigma_\alpha, \sigma_\beta, and\ \sigma_\gamma$ represent three hyperparameters that are determined empirically. Additional features, such as intensity and RGB values, can also be incorporated [68].

Reducing the CRF energy function improves label assignment. Although precise minimization of equation (15) is not possible, [71] proposed a mean-field iteration technique to estimate and efficiently solve the problem. This algorithm is formulated for a use as a recurrent neural network (RNN) [72].

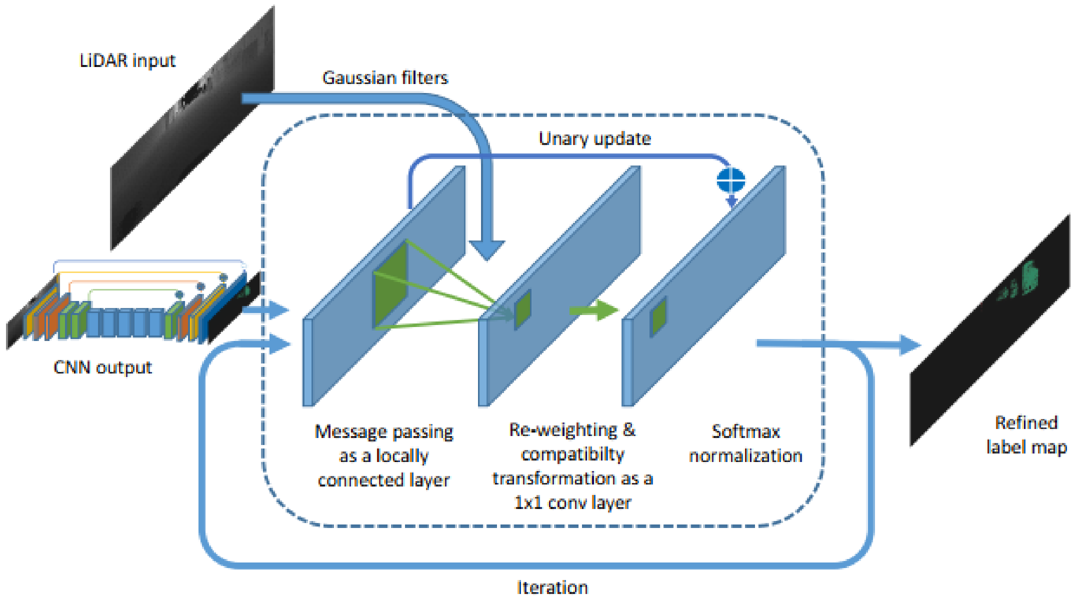

LiDAR input
Gaussian filters
Unary update
CNN output
Message passing as a locally connected layer
Re-weighting & compatibilty transformation as a 1x1 conv layer
Softmax normalization
Refined label map
Iteration

Figure 24 Conditional Random Field (CRF) as an RNN layer [67].

As shown in (Fg.31) the initial probability map, the CNN output is sent into the CRF module. As indicated in equation (16), Gaussian kernels are computed depending on input features. The values of these kernels fall fast as the distance between points in 3D Cartesian and 2D angular spaces increases. The initial probability map is filtered using Gaussian kernels on the input tensor. Message passing is a step that collects probability of surrounding points and can be implemented as a locally connected layer. The aggregated probability is then re-weighted, and a compatibility transformation is used to determine changes in the distribution of each point. During training, a 1x1 convolution with learnable parameters is used. By adding it to the 1x1 convolution output and normalizing it using softmax, the original probability is updated. Using this approach, the refined probability map can be enhanced iteratively [67].

4.1.3 Focal loss

LiDAR point clouds offer a severely skewed distribution of point types. There are far more background points than foreground objects like automobiles and humans. Because of this skewed distribution, the model focuses on easy-to-classify background points that give no meaningful learning signals, while foreground items are not appropriately handled during training [68].

To address this issue, focal loss is used [73] . The loss contribution from distinct pixels is adjusted by focal loss, which focuses on challenging examples. For a given pixel label t and expected probability p_t, focal loss modifies the cross-entropy loss by a modulating factor $(1 - p_t)^\gamma$, resulting in the focus loss for that pixel is thus:

$$FL(p_t) = -(1 - p_t)^\gamma \log(p_t) \tag{17}$$

When a pixel is misclassified and p_t is small, the modulating factor approaches to one, and the loss remains constant. As p_t approaches 1, the factor decreases to 0 and the loss for well-classified pixels is down weighted. The focusing parameter controls the rate at which well-classified examples are down weighted in a smooth way. Focal Loss is equal to Cross Entropy Loss when γ = 0. As γ grows, so does the modifying factor's impact [68].

4.1.4 LiDAR Mask

A LiDAR Mask is an extra channel added to LiDAR data that represents a binary mask that specifies whether each pixel is present or not [68]. In this case, in addition to the original (x, y, z, intensity, depth) channels, -a binary mask is added that indicates whether each pixel is missing or present. This additional information can aid in the segmentation accuracy of specific objects, such as bikers, in point cloud processing tasks such as those handled by semantic segmentation models.

4.1.5 Batch normalization

Batch Normalization (BN) is a technique introduced by Sergey Ioffe and Christian Szegedy in 2015 to improve the training of deep neural networks [74]. It addresses the issue of internal covariate shift, which occurs when the distribution of input data changes during training, causing the network to continuously adapt to these changes and slowing down the learning process.

BN works by normalizing the activations or outputs of a layer in a neural network, ensuring they have a mean of zero and a standard deviation of one. This normalization is performed across each mini batch of data during training. By doing so, BN stabilizes the distributions of layer outputs, allowing the network to be trained with higher learning rates and reducing the sensitivity to weight initialization. BN is added after every convolution layer in this model. And it addresses the issue of internal covariate shift, a common problem when training deep neural networks, by normalizing layer outputs and promoting stable learning [68].

4.2 DeepLab V3+

DeepLab V3+ extends DeepLab V3 by including an efficient decoder module that recovers object boundaries, as seen in Fig (25). DeepLab V3 is the model's encoder, and it uses atrous convolution to encode rich semantic information. The lack of a decoder module in DeepLab V3 can, however, result in the loss of spatial information in the segmentation mask. DeepLab V3+ addresses this by including a decoder module, which allows for more precise object segmentation. An upsampling module and skip connections connect the encoder feature maps to the decoder, and a final convolutional layer provides the segmentation mask. This enhancement improves the accuracy and quality of segmentation findings, particularly for objects with complex shapes or borders [63] [75].

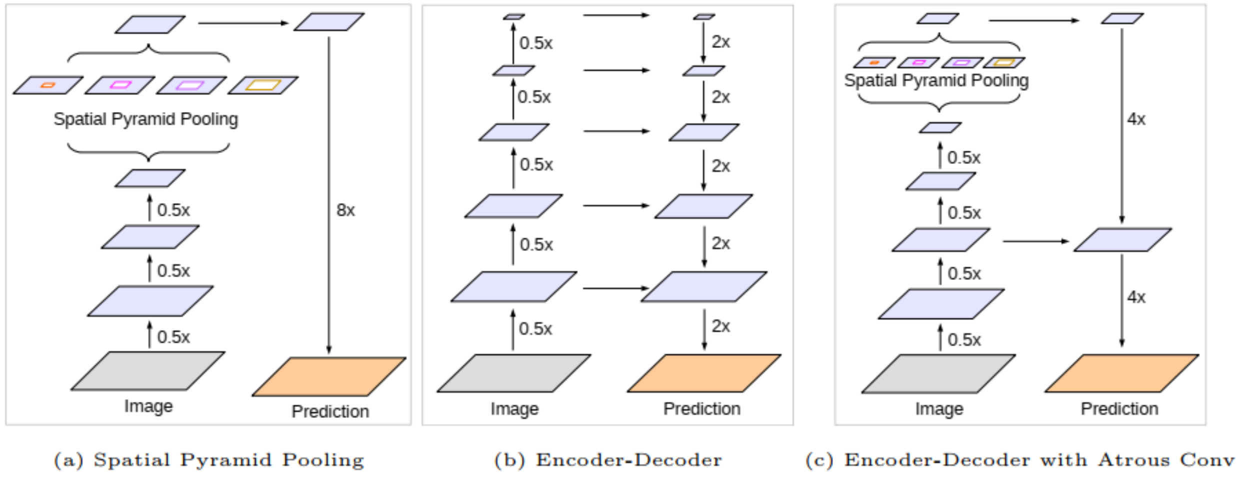


Figure 25 Improved DeepLab V3 model [63], with a) Spatial pyramid pooling, and b) Encoder-Decoder.

As shown in the above (Figure 25, c) the model explores two different types of neural network designs for semantic segmentation: spatial pyramid pooling modules and encoder-decoder architectures [63]. The first type effectively collects contextual data by pooling features at varying resolutions, while the latter provides clear object boundaries. DeepLab V3 [61] uses Atrous Spatial Pyramid Pooling (ASPP) to capture contextual information at different scales, but loses some features related to object boundaries due to pooling or striding operations within the network backbone. Atrous convolution can help extract denser feature maps, but it can be computationally challenging for models that require denser output features.

Encoder-decoder models, on the other hand, offer faster computation by avoiding dilated features in the encoder path and gradually recovering crisp object boundaries in the decoder path. The DeepLab V3+ model increases the encoder module within encoder-decoder networks to incorporate multi-scale contextual information by leveraging the strengths of both methodologies [63].

4.2.1 Encoder-Decoder network structure

The DeepLab V3+ model uses an encoder-decoder design as shown in fig 26, to effectively handle semantic segmentation issues. This design captures high-level semantic information and low-level spatial data to produce precise and fine-grained segmentation maps.

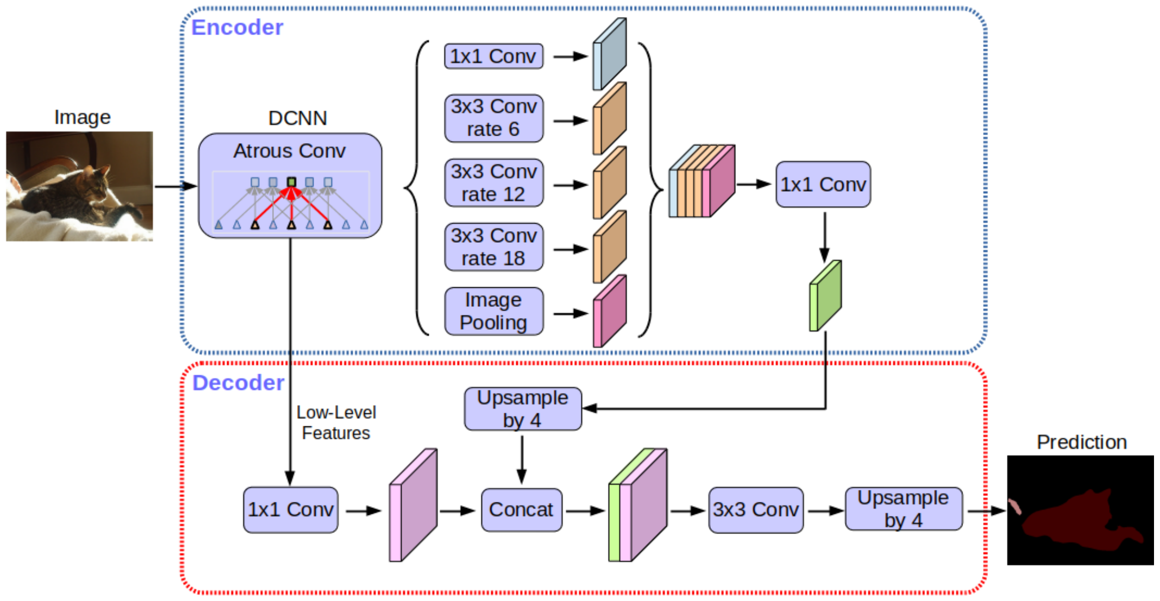


Figure 26 Model of DeepLabV3+ [63].

4.2.2 Encoder

The encoder part of this model uses a deep convolutional neural network (CNN) backbone to extract information from the input image. Backbones that are utilized in this model include Xception and ResNet-101 which provide a balance of computing efficiency and accuracy. These backbone networks are pre-trained CNN that are commonly trained on large-scale classification tasks like ImageNet. The choice of backbone greatly impacts the model's performance and efficiency.

I. Backbone

Modified-Xception model: demonstrated promising image classification results on ImageNet [76] while maintaining fast computation speeds [77]. Motivated by these findings, the DeepLab V3+ model attempted to adapt the Xception model for semantic image segmentation tasks by making a few minor changes. These modifications include: (1) a deeper Xception without changing the entry flow network structure, ensuring fast computation and memory efficiency; (2) replacing all max pooling operations with depthwise separable convolution with striding, allowing the use of atrous separable convolution to extract feature maps at any resolution (another option is to extend the atrous algorithm to max pooling operations); and (3) incorporating extra batch normalization [74] and ReLU activation following each 3×3 depthwise convolution.

ResNet-101: The DeepLab V3+ model uses ResNet-101 [78]as a network backbone for extracting important features from input images that are necessary for semantic segmentation. By utilizing ResNet-101, DeepLab V3+ can quickly and effectively extract features. Residual connections within the network help address the vanishing gradient problem commonly found in deep neural networks, leading to more accurate learning and gradient flow throughout all layers. This ultimately improves performance in the semantic segmentation task. To ensure precise segmentation masks while maintaining object boundaries and spatial information, the ResNet-101 backbone is combined with atrous convolution, spatial pyramid pooling, and an encoder-decoder structure within DeepLab V3+.

II. Atrous convolution

One of the significant challenges with the FCN approach is the excessive downsizing caused by multiple consecutive pooling operations. The input image is downsized by 32x through these pooling operations, and then upsampled to obtain the segmentation result. This level of downsizing leads to a loss of valuable information, which can impact the quality of the segmentation. Additionally, the deconvolution required to upsample by 32x is computationally and memory-intensive, as it involves additional parameters for learning the upsampling process.

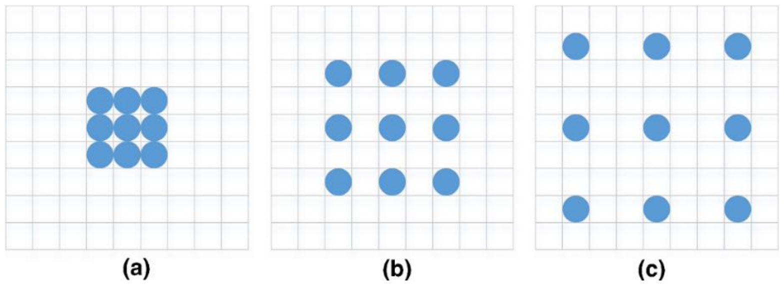


Figure 27 Dilated convolution example [79].

Dilated convolution (also known as Atrous convolution or hole convolution) is illustrated in (Fig. 22). It shows a convolutional layer with a kernel size of 3x3, along with three different convolution operations using different dilation rates: a normal convolution with dilation rate 1 (a), a dilated convolution with dilation rate 2 (b), and a dilated convolution with dilation rate 3 (c).

Dilated convolution is a technique that increases the size of the convolutional filter by inserting zeros (called holes) between the filter parameters [33]. The number of zeros inserted is referred to as the dilation rate. When the dilation rate is equal to 1, it is equivalent to standard convolution. When the rate is increased to 2, one zero is inserted between every other parameter, resulting in a filter with the capacity to capture the context of a 5x5 convolution using only 3x3 convolutional parameters. Similarly, when the rate is set to 3, the receptive field of the filter becomes 7x7 [80] [61].

To reduce the downsampling rate in DeepLab V3, the last pooling layers are replaced with ones that have a stride of 1 instead of 2, resulting in a down sampling rate of only 8x. The model then uses a series of Atrous convolutions to capture larger contextual information. During training, the output labeled mask is downsampled by 8x to facilitate pixel-level comparison. For inference, bilinear upsampling is used to produce an output of the same size as the input image, which provides acceptable results at lower computational and memory costs since bilinear upsampling does not require any additional parameters, unlike deconvolution-based upsampling [61].

Consider two-dimensional signals in which each point i on the output y interacts with a filter w. Over the input feature map x, atrous convolution is used:

$$y[i] = \sum_{k} x[i + r \cdot k] w[k] \tag{18}$$

The stride utilized to sample the input signal corresponds to the atrous rate r in this situation. This is equal to convolving the input x with upsampled filters generated by adding $r - 1$ zeros between successive filter values along each spatial dimension (thus the French term atrous convolution, as "trous" translates to "holes" in English). Standard convolution occurs when the rate $r = 1$, but atrous convolution allows for adaptive change of the filter's field-of-view by varying the rate value [61].

III. Spatial Pyramid Pooling

Spatial Pyramid Pooling (SPP) is a technique that takes features of different sizes and combines them into fixed-length feature vectors. To do this, the input feature map is divided into sub-regions of varying sizes, and the features in each sub-region are pooled. This pooling method creates a feature vector that encodes the presence or absence of specific features in each sub-region. SPP is beneficial for object recognition and semantic segmentation because it captures both fine-grained and coarse-grained information about the input image by pooling features of different sizes [81].

The DeepLabv3+ model uses an Atrous Spatial Pyramid Pooling (ASPP) module with various SPP layers that have different dilation rates. This helps the model capture multi-scale contextual information and improve semantic segmentation accuracy by identifying fine-grained features [63].

IV. Atrous Spatial Pyramid Pooling (ASPP)

The ASPP module captures multi-scale contextual information by applying parallel atrous convolutions with varying dilation rates. To construct the final high-level features, the output feature maps from these parallel branches are concatenated and then run through a 1×1 convolution as shown in Fig 28. This module assists the model in adapting to objects of various sizes and shapes.

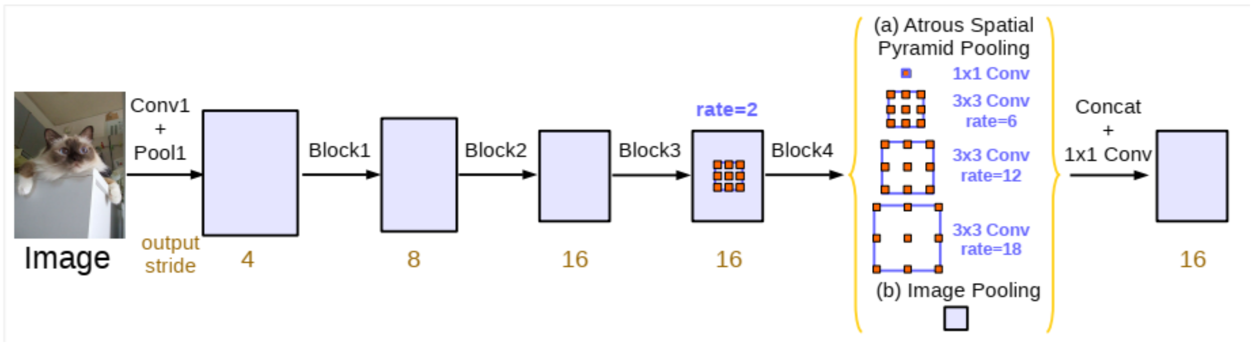


Figure 28 Parallel modules with atrous convolution (ASPP), augmented with image-level features [61].

V. Depthwise separable convolution

In the DeepLab V3+ model, the Atrous Spatial Pyramid Pooling (ASPP) module uses depthwise separable convolution in the encoder [63]. This technique is specifically applied in the parallel atrous convolution branches that make up the ASPP. Depthwise separable convolution is a more efficient version of normal convolution. It involves splitting the process into two parts (Fig. 29): depthwise convolution and pointwise convolution. Depthwise convolution uses a single filter per input channel, while pointwise convolution combines the depthwise convolution outputs using 1×1 filters. This factorization reduces the number of parameters and computational complexity, making the model more efficient while maintaining its performance.

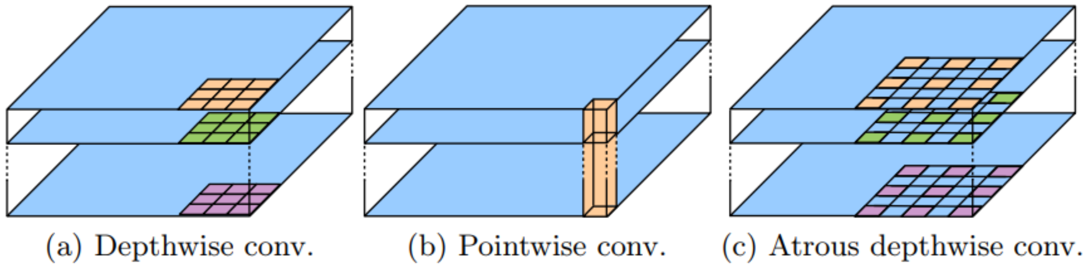


Figure 29 Depthwise separable convolution [63].

As shown in Fig. 29, 3x3 depthwise separable convolution divides a normal convolution into two parts: (a) a depthwise convolution (using a separate filter for each input channel) and (b) a pointwise convolution (merging the depthwise convolution outputs across channels). (c) depthwise separable convolution with rate 2.

In summary, DeepLab V3+ generalizes the DeepLab V3 model by using it as an encoder, inheriting its core components described above for feature extraction and context processing while adding a decoder to refine the segmentation results.

4.2.3 Decoder

DeepLab V3+ decoder's major function is to recover spatial information lost during the encoding phase and refine segmentation results. To accomplish this, as shown in Fig. 26, a simple yet effective decoder module is incorporated that bilinearly upsamples the encoder's high-level feature maps and concatenates them with low-level features using the skip technique [63].

I. Bilinear Upsampling

The decoder first uses bilinear interpolation to upsample the encoder's high-level feature maps. Bilinear interpolation is a low-cost method for enhancing the spatial resolution of feature maps. The upsampling procedure recovers part of the lost spatial information by interpolating pixel values based on their surrounding neighbors, resulting in more accurate segmentation output [63].

II. Skip connection

While the phrase skip connection is not explicitly used in the original DeepLab V3+ study, the concept of integrating high-level and low-level features in the decoder corresponds to the idea of skip connections, which will be used to refine the segmentation boundaries. These skip connections combine the upsampled high-level features with the encoder's corresponding low-level features. The low-level features give fine-grained spatial information that aids in the recovery of object boundaries and the generation of more exact segmentation results. The merging of high-level and low-level information improves overall segmentation quality by combining the encoder's semantic understanding with the spatial details preserved in the network's lower layers.

Bibliography

[1] C. L. J. L. M. B. a. Y. L. X. Li, "A Multi-Sensor Environmental Perception System for an Automatic Electric Shovel Platform," *IEEE,* p. 4335, Jun. 2021.

[2] W. M. I. P. Dan Barnes, "Find Your Own Way: Weakly-Supervised Segmentation of Path Proposals for Urban Autonomy," *arXiv,* vol. abs/1610.01238, 17 Nov 2017.

[3] S. Manhart, "Autonomous Path Planning with LIDAR and Motion Capture," 21 May 2021. [Online]. Available: https://www.hackster.io/manhart3/autonomous-path-planning-with-lidar-and-motion-capture-6ff36f. [Accessed 2022].

[4] B. Sahu, "The Evolution of Deeplab for Semantic Segmentation," 12 July 2019. [Online]. Available: https://towardsdatascience.com/the-evolution-of-deeplab-for-semantic-segmentation-95082b025571. [Accessed 2022].

[5] D. Krambeck, "Tesla vs Google: Do LIDAR Sensors Belong in Autonomous Vehicles?," 24 July 2016. [Online]. Available: https://www.allaboutcircuits.com/news/tesla-vs-google-do-lidar-sensors-belong-in-autonomous-vehicles/. [Accessed 2022].

[6] N. C. ∗. S. C. ∗. A. C. Denis Chikurtev ∗, "Mobile robot localization and navigation using LIDAR and indoor GPS," *ScienceDirect,* vol. 54, no. 13, pp. 351-356, 2021.

[7] R. X. S. S. C. L. Jawad Iqbal, "Simulation of an Autonomous Mobile Robot for LiDAR-Based In-Field Phenotyping and Navigation," *robotics,* Vols. 9, no. 2, p. 46, 21 June 2020.

[8] E. D. a. B. Mysliwetz, "Recursive 3-D road and relative ego-state recognition," *IEEE Trans. Pattern Anal. Mach,* Vols. 14, no. 2, pp. 199-213, Feb. 1992.

[9] T. Litman, " Autonomous Vehicle Implementation Predictions Implications for Transport Planning," 6 November 2022. [Online]. Available: https://www.vtpi.org/avip.pdf. [Accessed 2023].

[10] R. Berger, " Autonomous Driving," 2014. [Online]. Available: http://www.rolandberger.ch/media/pdf/Roland_Berger_TABAutonomousDriving%final20141211. [Accessed 2022].

[11] C. H.-S. L. R. H. H. C. G. F. T. W. W. K. D. Di Feng, "Deep Multi-modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges," *CoRR, eprinttype arXiv,* vol. abs/1902.07830, 821Feb 2019.

[12] Y. B. a. G. H. Y. LeCun, "Deep learning," *Nature,* Vols. 521, no. 7553, p. 436, 2015.

[13] A. Burkov, "The hundred-page machine learning book," 2020. [Online]. Available: http://themlbook.com/wiki/doku.php. [Accessed 2022].

[14] K. D. Foote, "A Brief History of Deep Learning,," 4 February 2022. [Online]. Available: https://www.dataversity.net/brief-history-deep-learning. [Accessed 2022].

[15] F. Rosenblatt, "The perceptron - a perceiving and recognizing automaton.," Cornell Aeronautical Laboratory, New York, 1957.

[16] "Understanding single layer Perceptron and difference between Single Layer vs Multilayer Perceptron," 6 September 2019. [Online]. Available: https://www.i2tutorials.com/what-is-single-layer perceptron-and-difference-between-single-layer-vs-multilayer-perceptron/. [Accessed 2022].

[17] N. Tyagi, "Understanding the Perceptron Model in a Neural Network," 27 January 2020. [Online]. Available: https://medium.com/analytics-steps/understanding-the-perceptron-model-in-a-neural-network-2b3737ed70a2. [Accessed 2022].

[18] "Single Layer Perceptron in TensorFlow," [Online]. Available: https://www.javatpoint.com/single-layer-perceptron-in-tensorflow. . [Accessed 2022].

[19] "Multilayer Perceptron," [Online]. Available: https://github.com/d-r-e/multilayer-perceptron/blob/main/README.md. [Accessed 2022].

[20] P. Baheti, "Activation Functions in Neural Networks [12 Types & Use Cases]," 21 October 2022. [Online]. Available: https://www.v7labs.com/blog/neural-networks-activation-functions#h3. [Accessed 2022].

[21] V. Jain, "Everything you need to know about "Activation Functions" in Deep learning models," 30 December 2019. [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models. [Accessed 2022].

[22] O. G. I. F. a. J. K. H. Zhao, "Loss Functions for Image Restoration With Neural Networks," *IEEE Transactions on Computational Imaging,* Vols. 3, no. 1, pp. 47-57, March 2017.

[23] V. Yathish, "Loss Functions and Their Use in Neural Networks," 4 Aug 2022. [Online]. Available: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks. [Accessed 2023].

[24] A. Gad, "A Comprehensive Guide to the Backpropagation Algorithm in Neural Networks," 27 January 2023. [Online]. Available: https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide. [Accessed 2022].

[25] A. Gupta, "A Comprehensive Guide on Optimizers in Deep Learning," 7 October 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/. [Accessed 2022].

[26] L. B. Y. B. a. P. H. Y. Lecun, "Gradient-based learning applied to document recognition," *in Proceedings of the IEEE,* Vols. 86, no. 11, pp. 2278-2324, Nov. 1998.

[27] M. T. J. By Samaya Madhavan, "Deep learning architectures," 7 September 2017. [Online]. Available: https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/. [Accessed 2023].

[28] M. Gurucharan, "Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network," 28 July 2022. [Online]. Available: https://www.upgrad.com/blog/basic-cnn-architecture/. [Accessed 2023].

[29] A. G. Josh Patterson, "Major Architectures of Deep Networks," [Online]. Available: https://www.oreilly.com/library/view/deep-learning/9781491924570/ch04.html. [Accessed 2023].

[30] R. Szeliski, "Computer Vision: Algorithms and Applications," Berlin, Germany, Springer, 2010.

[31] N. Klingler, "Image Segmentation with Deep Learning," [Online]. Available: https://viso.ai/deep-learning/image-segmentation-using-deep-learning/ . [Accessed 2023].

[32] J. Jordan, "An overview of semantic image segmentation," 21 May 2018. [Online]. Available: https://www.jeremyjordan.me/semantic-segmentation/. [Accessed 2022].

[33] N. Barla, "The Beginner's Guide to Semantic Segmentation," 16 September 2021. [Online]. Available: https://www.v7labs.com/blog/semantic-segmentation-guide.

[34] H. Bandyopadhyay, "The Definitive Guide to Instance Segmentation," 22 February 2022. [Online]. Available: https://www.v7labs.com/blog/instance-segmentation-guide. [Accessed 2022].

[35] "Introduction to image segmentation for machine learning," Super Annotate, 1 December 2021. [Online]. Available: https://www.superannotate.com/blog/image-segmentation-for-machine-learning.

[36] P. Sharma, "Computer Vision Tutorial: A Step-by-Step Introduction to Image Segmentation Techniques," 1 April 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation techniques-python/. [Accessed 2022].

[37] "Image segmentation," [Online]. Available: https://datagen.tech/guides/image-annotation/image-segmentation/. [Accessed 2022].

[38] Z. D. Y. Y. Xiaolong Liu, " Recent progress in semantic image segmentation," *Artificial Intelligence Review, Springer Science and Business Media (LLC),* Vols. 22, no-2, pp. 1089--1106, June 2018.

[39] L. Z. a. G. L. a. Y. Bao, "Improvement of grayscale image 2D maximum entropy threshold segmentation method," *2010 International Conference on Logistics Systems and Intelligent Management (ICLSIM),* vol. 1, pp. 324-328, 2010.

[40] H. E. R. J. Hu S, "Automatic lung segmentation for accurate quantitation of volumetric X-ray CT images," *IEEE Trans Med Imaging,* Vols. 20, no-6, pp. 490-8, June 2001.

[41] H. J. Hartigan JA, "Clustering algorithms," 1975. [Online]. Available: https://toaz.info/doc-view-2.. [Accessed 2022].

[42] L. Barghout, "Image Segmentation Using Fuzzy Spatial-Taxon Cut: Comparison of Two Different Stage One Perception," *Electronic Imaging,* vol. 2016, pp. 1-6, 2016.

[43] M. O. M. R. S. R. T. E. M. B. R. F. U. R. S. &. S. B. Cordts, "The Cityscapes Dataset for Semantic Urban Scene Understanding.," *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 3213-3223, June 2016.

[44] E. S. a. T. D. J. Long, "Fully convolutional networks for semantic segmentation," *IEEE Conf. Comput. Vis. Pattern Recognition,* pp. 3431-3440, 2015.

[45] O. R. a. P. F. a. T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *CoRR,* vol. abs/1505.04597, 2015.

[46] G. P. I. K. K. M. A. L. Y. Liang-Chieh Chen, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *CoRR,* vol. abs/1606.00915, p. 1606.00915, 12 May 2017.

[47] W. L. S. O. a. T. V. G. Wang, "Automatic brain tumor segmentation using cascaded anisotropic convolutional neural networks," *Proc. Int. MICCAI Brainlesion Workshop,* vol. 13, pp. 178-190, 2019.

[48] H. Q. J. D. X. J. a. Y. W. Y. Li, "Fully convolutional instance-aware semantic segmentation," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit,* vol. abs/1611.07709, pp. 2359-2367, 2017.

[49] M. C. a. Y.-C. L. Y. Yuan, "Automatic skin lesion segmentation using deep fully convolutional networks with Jaccard distance," *IEEE Trans. Med. Imag.,* Vols. 36, no. 9, pp. 1876-1886, Sep. 2017.

[50] H. L. M. Z. J. L. Z. S. a. T. T. N. Liu, "Accurate iris segmentation in non-cooperative environments using fully convolutional networks," *International Conference on Biometrics (ICB),* pp. 1-8, 2016.

[51] W. a. R. A. a. B. A. Liu, "ParseNet: Looking Wider to See Better," *CoRR,* vol. abs/1506.04579, June 2015.

[52] S. H. a. B. H. H. Noh, "Learning Deconvolution Network for Semantic Segmentation," *2015 IEEE International Conference on Computer Vision (ICCV),* pp. 1520-1528, 2015.

[53] J. S. X. Q. X. W. J. J. Hengshuang Zhao, "Pyramid Scene Parsing Network," *CoRR,* vol. abs/1612.01105, 27 Apr 2017.

[54] J. L. J. J. Y. &. X. X. Fu, "Dual attention network for scene segmentation," *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* Vols. Fu, J., Liu, J., Jiang, Y., & Xue, X., pp. 3146-3154, 2018.

[55] X. Z. Z. H. C. D. A. Z. O. R. &. J. M. Qin, "BASNet: Boundary-aware salient object detection," *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 7479-7489, 2019.

[56] P. X. H. F. J. S. Hanchao Li, "DFANet: Deep feature aggregation for real-time semantic segmentation," *In Proceedings of European Conference on Computer Vision (ECCV),* vol. abs/1904.02216, 2019.

[57] K. (. L. Derrick Mwiti, "Image Segmentation: Architectures, Losses, Datasets, and Frameworks," 26 April 2023. [Online]. Available: https://neptune.ai/blog/image-segmentation. [Accessed 2023].

[58] F. K. V. &. F. T. Yu, "Dilated residual networks," *In Conference on Computer Vision and Pattern Recognition,* vol. abs/1705.09914, pp. 472-480, 2017.

[59] O. a. S. J. a. F. L. a. L. M. a. H. M. a. M. K. a. M. K. a. M. S. a. H. N. a. K. B. a. G. B. a. R. D. Oktay, "Attention U-Net: Learning Where to Look for the Pancreas," *CoRR,* vol. abs/1808.08114, April 2018.

[60] M. Z. B. C. D. K. W. W. T. W. M. A. H. A. Andrew G. Howard, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.048,* vol. abs/1704.04861, 17 Apr 2017.

[61] G. P. F. S. H. A. Liang-Chieh Chen, "Rethinking Atrous Convolution for Semantic Image Segmentation," *CoRR,* vol. abs/1706.05587, 5 Dec 2017.

[62] B. Sahu, "The Evolution of Deeplab for Semantic Segmentation," 12 July 2019. [Online]. Available: https://towardsdatascience.com/the-evolution-of-deeplab-for-semantic-segmentation-95082b025571.. [Accessed 2023].

[63] G. P. F. S. H. A. Liang-Chieh Chen, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," *CoRR,* vol. abs/1802.02611, 22 Aug 2018.

[64] T. Y. D. P. G. R. H. K. H. B. &. B. S. Lin, "Feature pyramid networks for object detection," *In Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 2117-2125, 2017.

[65] Y. Q. H. D. J. J. X. &. W. Y. Li, "Fully convolutional instance-aware semantic segmentation," *In Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 2359-236, 2018.

[66] H. D. K. S. J. Z. Z. W. X. T. A. &. A. A. Zhang, "Context encoding for semantic segmentation," *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 7151-7160, 2018.

[67] A. W. X. Y. a. K. K. Bichen Wu, "SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud," *CoRR,* vol. abs/1710.07368, 19 Oct 2017.

[68] X. Z. S. Z. X. Y. K. K. Bichen Wu, "SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from LiDAR Point Clouds," *CoRR,* vol. abs/1809.08495, 22 Sep 2018.

[69] L. Y. H. S. a. L. J. G. C. R. Qi, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," *in Advances in Neural Information Processing Systems,* pp. 5099-5108, 2017.

[70] L. B. a. X. H. Yiming Zhao, "FIDNet: LiDAR Point Cloud Semantic Segmentation with Fully Interpolation Decoding," *CoRR,* vol. abs/2109.03787, 8 Sep 2021.

[71] S. H. M. W. M. K. A. W. J. D. a. K. K. F. N. Iandola, "SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," *CoRR,* vol. abs/1602.07360, p. arXiv:1602.07360, 2016.

[72] P. K. ¨. u. a. V. Koltun, " "Efficient inference in fully connected ¨ crfs with gaussian edge potentials," *in Advances in neural information processing systems,* vol. abs/1210.5644, p. 109–117, 2011.

[73] P. G. R. G. K. H. P. D. Tsung-Yi Lin, "Focal Loss for Dense Object Detection," *CoRR,* vol. abs/1708.02002, 2018.

[74] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep Network Training by Reducing," *CoRR,* vol. abs/1502.03167, 2015.

[75] J. S. L. W. S. L. a. L. L. C. Liu, "LA-DeepLab V3+: A Novel Counting Network for Pigs," *Agriculture,* Vols. 12, no. 2, p. 284, Feb. 2022.

[76] J. D. H. S. J. K. S. S. S. M. Z. H. A. K. A. K. M. B. A. C. B. L. F.-F. Olga Russakovsky, "ImageNet Large Scale Visual Recognition Challenge," *CoRR,* vol. abs/1409.0575, 2014.

[77] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR,* vol. abs/1610.02357, 4 April 2017.

[78] K. Z. X. R. S. S. J. He, "Deep residual learning for image recognition," *CoRR,* vol. abs/1512.03385, 10 December 2016.

[79] X. D. Z. &. Y. Y. Liu, "Recent progress in semantic image segmentation," *Artif Intell Rev 52,* p. 1089–1106, 2019.

[80] A. C. N. Matcha, "A 2021 guide to Semantic Segmentation," 2020. [Online]. Available: https://nanonets.com/blog/semantic-image-segmentation-2020/. [Accessed 2022].

[81] C. S. a. J. P. S. Lazebnik, "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categorie," *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), New York, NY, USA,* p. 216, 2006.