FACHHOCHSCHULE KIEL
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN | UNIVERSITY OF APPLIED SCIENCES

Semantic Segmentation of Aquatic Organisms in Underwater Videos Using Deep Learning Methods

MASTER THESIS

by

Veeramalli Rajesh

Matriculation Number - 932249

submitted to obtain the degree of

MASTER OF SCIENCE (M.SC.)

at

FACHHOCHSCHULE KIEL (KIEL UNIVERSITY OF APPLIED SCIENCES)
DEPARTMENT OF COMPUTER SCIENCE AND ELECTRICAL ENGINEERING

Degree Program

INFORMATION ENGINEERING

First Reviewer: Prof. Dr. Hauke Schramm
Kiel University of Applied Sciences

Second Reviewer: Gordon Böer
Kiel University of Applied Sciences

Kiel, May 2021

Contact details: VEERAMALLI RAJESH
Steenbeker weg 22
24106, Kiel, Germany.
veeramalli.rajesh@yahoo.com
Phone: +49 176 66336422

Abstract

Semantic segmentation is a computer vision technique that works at the pixel level to, gain precise knowledge about an image. It labels every single pixel in an image with the category it belongs to and gives an idea about all the objects present in an image. In object detection, only the label and the location of the object are extracted but the exact shape of the object is not known. Segmentation provides rich information about the image i.e the object's shape and boundary. There are several applications of semantic segmentation in a wide range of areas, like robotics, mapping, medical image analysis, etc, in which pixel-level labels are of primary importance. In recent years, state-of-the-art deep neural networks have shown impressive results and have become go-to solutions for several recognition tasks. In this thesis, the use of deep neural networks for semantic image segmentation of fishes in underwater videos is thoroughly investigated.

The main goal of this research work is to develop a deep neural network for the segmentation of fishes for the underwater fish observatory (UFO). The UFO is an autonomous system to continuously monitor the diversity and abundance of the fish population which has mainly been operated within the Bay of Kiel region, so far. The segmentation network is expected to correctly identify the shape of the fishes and separate them from the background in the underwater videos. This automatic and optimized fish segmentation helps to build an application to identify the size and weight of the species with higher statistical certainty. In this way, an automated, non-invasive, and cost-effective alternative to ship-based monitoring of fish stocks will be developed and tested in Kiel Bight as a contribution to a reformed, evidence-based fisheries policy. The efficient segmentation networks, specifically PSPNet, U-Net, and DeepLabv3 are trained and tested on the fish image data provided by the UFO in this research work.

Abstrakt

Die semantische Segmentierung ist eine Computer-Vision-Technik, die auf Pixelebene arbeitet, um genaue Erkenntnisse über ein Bild zu gewinnen. Sie kennzeichnet jedes einzelne Pixel in einem Bild mit der Kategorie, zu der es gehört, und gibt einen Überblick über alle in einem Bild vorhandenen Objekte. Bei der Objekterkennung werden nur die Klasse und die Position des Objekts extrahiert, aber die genaue Form des Objekts ist nicht bekannt. Die Segmentierung liefert umfangreiche Informationen über das Bild, d. h. über die Form und den Umriss des Objekts. Es gibt mehrere Anwendungen der semantischen Segmentierung in einer Vielzahl von Bereichen, wie z. B. Robotik, Kartierung, medizinische Bildanalyse usw., in denen Markierungen auf Pixelebene von primärer Bedeutung sind. In den letzten Jahren haben hochmoderne tiefe neuronale Netze beeindruckende Ergebnisse gezeigt und sind zu bevorzugten Lösungen für verschiedene Erkennungsaufgaben geworden. In dieser Arbeit wird die Verwendung von tiefen neuronalen Netzen für die semantische Bildsegmentierung von Fischen in Unterwasservideos eingehend untersucht.

Das Hauptziel dieser Forschungsarbeit ist die Entwicklung eines tiefen neuronalen Netzes zur Segmentierung von Fischen für das Unterwasser-Fisch-Observatorium (UFO). Das UFO ist ein autonomes System zur kontinuierlichen Überwachung der Vielfalt und Abundanz der Fischpopulation wleches bisher vorwiegend in Kieler Bucht eingesetzt wurde. Das Segmentierungsnetzwerk soll die Form der Fische korrekt erkennen und sie in den Unterwasservideos vom Hintergrund abgrenzen. Diese automatische und optimierte Fischsegmentierung hilft, eine Anwendung zu erstellen, die Größe und Gewicht der Arten mit hohrer statistischer Sicherheit identifiziert. Auf diese Weise wird eine automatisierte, nicht-invasive und kostengünstige Alternative zur schiffsbasierten Überwachung der Fischbestände entwickelt und in der Kieler Bucht als Beitrag zu einer reformierten, evidenzbasierten Fischereipolitik getestet. Die effizienten Segmentierungsnetzwerke, insbesondere PSPNet, U-Net und DeepLabv3, werden in dieser Forschungsarbeit auf den von der UFO zur Verfügung gestellten Fischbilddaten trainiert und getestet.

Acknowledgements

I would first like to express my deep thanks and gratitude to my supervisor, Prof. Dr. Hauke Schramm for his guidance and continuous support during this research. His insightful and invaluable feedback pushed me to sharpen my thinking and brought my work to a higher level. I would also thank Gordon Böer for his enormous support throughout my master thesis. He answered all my questions with utmost patience. His continuous feedback and suggestions helped me to implement new techniques in my work. I also thank the Kiel University of Applied Sciences for being so helpful and providing me with the necessary computing power to train the networks. I would also thank all the research assistants and my colleagues in the working group who helped me with data annotations. I would like to thank my University who helped me with a stipend during my research work.

In addition, I am extremely thankful to my parents and grandparents for boosting my spirits and encouraging me in all aspects of my life. They always gave me courage and guidance to face challenges and pat my back when I succeed. Finally, I could not have completed this dissertation without the support of my near and dear friends, they gave me emotional support as well as happy distractions to rest my mind outside of my research. Thank you to everyone and forgive me if I forgot to mention anyone specifically.

Contents

List of Figures

List of Tables

List of Abbreviations

UFO	**U**nderwater **F**ish **O**bservatory
ANN	**A**rtifical **N**eural **N**etwork
DCNN	**D**eep **C**onvolutional **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
FCN	**F**ully **C**onvolutional **N**etwork
FCr	**F**ully **C**onnected
ReLU	**R**ectified **L**inear **U**nit
TL	**T**ransfer **L**earning
PSPNET	**P**yramid **S**cene **P**arsing **N**etwork
PR	**P**recision-**R**ecall
CVPR	**C**omputer **V**ision and **P**attern **R**ecognition
TP	**T**rue-**P**ositives
FP	**F**alse-**P**ositives
TN	**T**rue-**N**egatives
FN	**F**alse-**N**egatives
SOTA	**S**tate of the **A**rt

Chapter 1

Introduction

Machine learning and Deep learning attracted a lot of researchers in the past few decades due to their impeccable abilities. Stock market prediction [1], cancer prediction [2], self-driving cars [3], etc. have incorporated deep learning processes to be better and faster.

> "***Computer Vision is building algorithms that can understand the content of images and use it for other applications. [4]***"

Computer Vision, which deals with images and videos has emerged as one of the top branches of machine learning. Computers can be trained to understand images at various levels defined in terms of granularity. It ranges from a coarse-grained level to a more fine-grained level of understanding an image. In the Computer Vision domain, there is a problem defined at every level of granularity to detect and understand different features of an image.

1.1 Image Classification

The process of assigning a class, or label for something that is defined by a set of data points is defined as the classification problem. Image classification is a subset of the classification problem, where a computer analyzes and identifies the class of an image. A class is essentially a label, for instance, 'flower', 'animal', 'van', and so on. Image classification helps in many ways but for some tasks, it requires a deeper image analysis. Semantic Segmentation helps to do deep image analysis at a pixel level, by clustering the parts of the image that belong to the same class. An image is a collection of pixels and analyzing an image at pixel level alleviates the prediction capabilities of a machine learning model. The pixel grouping is carried out in three steps, they are:

- **Image classification -** Identifies the class of the object present in the image.
- **Object Detection -** Detecting which object is present and where it is present in an image by highlighting its location via a bounding box. In object detection the exact shape of the object is unknown.

- **Image segmentation -** Finding all the pixels that belong to a specific class inside the bounding box in an image. It identifies the exact shape of the object.

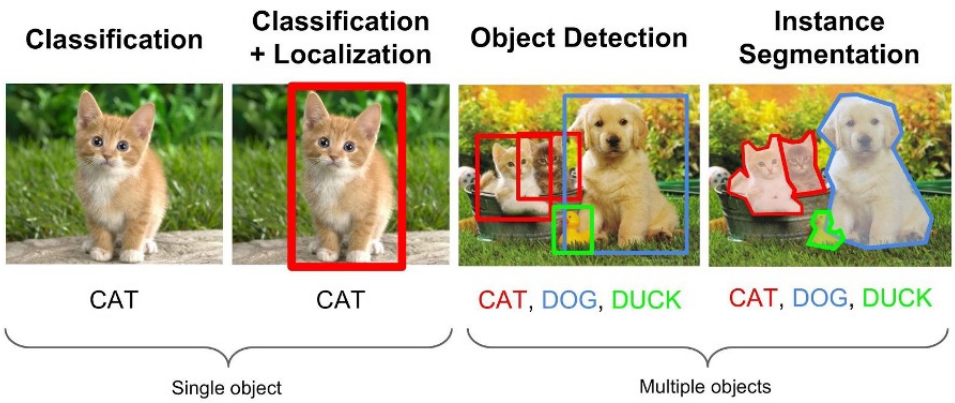

Figure 1.1: Image classification, Object detection and Instance segmentation. Source: [5]

In image classification, the process is limited to assigning a single class label based on the objects present in an entire image. In contrast, object detection not only classifies what individual objects are present in an image but also localizes the part where the object is present in the whole image using bounding boxes. Object detection is very feasible solution when there are multiple objects in an image. Image segmentation is an enhanced detection and classification technique that works on every pixel of an image and assigns it with a corresponding class. The current work focuses mainly on image segmentation, and other topics are not explained in detail in this paper.

1.2 Image Segmentation

Image segmentation is a computer vision technique that converts raw input images to masked images with highlighted regions of interest. Since segmentation works on every pixel of an image it can also be called as "full-pixel semantic segmentation". Earlier computer vision techniques were used to extract edges, colors, gradients, etc. but they could not classify images at the pixel level. Semantic segmentation helps to understand an image more precisely. Semantic segmentation is applied in several fields like autonomous driving [6], facial segmentation [7], fashion [8], etc. Image segmentation can be internally classified into two major types, based on how different objects of the same class get their label namely, semantic segmentation, and instance segmentation.

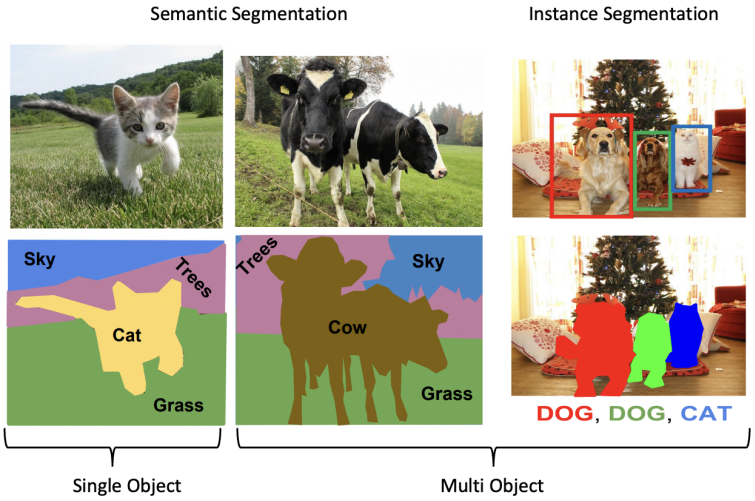

Figure 1.2: Example of Semantic Segmentation and Instance segmentation. Source: [9]

- **Semantic segmentation:** In semantic segmentation each pixel in an image is assigned a specific class label. It detects all object instances and assigns a class to it, but does not differentiate between multiple instances that belong to the same object. For example, as shown in figure 1.2 when there are two cows in the image they are classified as cow and they are portrayed as a single instance.
- **Instance segmentation:** Instance segmentation not only classifies each pixel to a class ID but also assigns a unique label to every instance of an object in the image. As shown in figure 1.2, there are two dogs and one cat in the image, and each dog is assigned a different color, to differentiate one from the other. It treats every object as an individual.

There are two types of semantic segmentation methods based on the number of class of objects to be classified in an image.

Binary Segmentation: This method aims at detecting objects from a single class and attempts to classify each pixel of the objects of one class and the remaining pixels are classified as a background.

Figure 1.3: Binary Semantic Segmentation example from Carvana dataset. Source: [10].

As shown in figure 1.3, the pixels of the car was the target pixels to be detected by the model and all the remaining pixels were masked as background pixels. The binary segmentation is useful in applications where there is only one object class of interest in the images.

Multi-class Segmentation: This method aims at multi-class object detection and attempts to classify all pixels in an image. In this case, the model has to assign a class label to each pixel, where the class ids have multiple options, usually more than two.

Figure 1.4: Image from Cityscapes Segmentation dataset. Source: [11].

In figure 1.4, there are several objects that belong to multiple classes like a car, road, traffic sign, person, vegetation, motorbike, etc., all those objects are classified individually. Each class is assigned a different color for better visualization. The image is taken from the "Cityscapes dataset", a very famous set of images for benchmarking semantic segmentation algorithms.

A core semantic segmentation architecture consists of an encoder network. and a corresponding decoder network. The architecture is explained in detail in section 6.6. Semantic segmentation projects the discriminative features learned by the encoder onto the pixel space and passes them into the decoder. There are many approaches to achieve segmentation. But before going through all this, the main question is: **Does one really need semantic segmentation, is classification not enough for any ML model and how does it help in real-life scenarios?**

Classification helps in many ways, it can give an overview of what is present in an image, but in some tasks, we need precise information like what objects are present, where they are, and the exact shapes of them. This can be achieved by image segmentation. Consider the example of autonomous vehicles, they receive data from many sensory input devices like cameras, radar, and lidars. A car cannot decide a path just by knowing what objects are in front of the vehicle, it also needs to

know where they are. Once the vehicle gets information like what and where the objects are, it becomes possible to create a digital map for the travel route. Object detection and image segmentation are the core elements for autonomous driving.

Before deep learning, image segmentation was made by using classical machine learning techniques like Support Vector Machines [12], Random Forest [13], K-means clustering [14] etc. Once deep learning was introduced, various convolutional neural networks like VGG16 [15], UNet [16], ResNet [17], etc., were used and it showed impressive results. All these state-of-the-art networks opened doors for many applications in many fields like medical, fashion, automobile, geology, etc. In recent years, researchers are using Convolutional Neural Networks (CNNs) extensively for semantic segmentation. For semantic segmentation, either existing pre-trained CNN models are taken as a baseline using transfer learning, or new CNN architectures are designed and trained from scratch. When existing models are used, the previously learned parameters are considered as prior information and the model must be trained again according to application.

Semantic segmentation is also a classification task where every pixel of an image is labeled with the class of the corresponding enclosing object. Segmentation can be a single-step or multi-step process. When segmentation is carried out in a single step, only the pixel classification is the final result. Whereas, in a multi-step segmentation process, there are some additional post-processing steps. For example, the network output can be passed through conditional random fields (CRFs) and ensemble approaches. CRFs provide statistical modeling to obtain a structured prediction. CRFs help semantic segmentation to improve the capability of the network to represent the object along the boundaries. In ensemble approaches the strengths and predictions of multiple algorithms are used to achieve better predictive performance.

In this thesis, deep learning models are used to apply semantic segmentation on images of fish extracted from underwater videos and this application can be called as **Fish segmentation**. Similar to the existing stationary UFO system, newly built mobile, and portable underwater fish observatories will also combine optical methods and modern acoustic for monitoring fish stocks being provided as a coupled hybrid system. The hybrid system is a coupling of near field (optics) and far-field (acoustics). The coupled UFO system tests in the Kiel Bight region ensure low logistics costs, as they are performed close to the coast. This project, enables the underwater fish observatory (UFO), to monitor the fish stocks, analyze the biomass and the health of the aquatic organisms. The state-of-the-art segmentation models like U-Net [16], PSPNet [18] and DeepLabv3 [19] are used as the baseline networks for the purpose of segmentation in this work.

1.3 Outline of the Master Thesis

The rest of the work is organized in different chapters as follows: Chapter 2 lists previous attempts to image segmentation, different methods of image segmentation, and fish segmentation. Chapter 3 gives a higher-level introduction to Artificial Intelligence and Machine learning and Deep learning. Chapter 4 explains convolution neural networks and various extensions of the standard convolutional layer used in the current work. Chapter 5 explains the data, the annotation process, data preprocessing and post-processing techniques, and the data augmentation techniques used in this work. Chapter 6 describes the hardware and software, training configurations, various hyper-parameters, and various network architectures used in this work. Chapter 7 thoroughly explains the evaluation metrics and their importance. Chapter 8 clearly explains the results of the experiments and compares the performance of all the networks used in this work. The results are summarized and further discussed in Chapter 9.

Chapter 2

Related Work

As discussed in the introduction the main goal of image segmentation is to represent an image in a meaningful way to ease the analyzing process. When image segmentation just started, it was used to locate boundaries, lines, curves, etc from images. The quality and capability of image segmentation has been growing rapidly. Many innovations have been made in the field of computer vision encouraging the application of image segmentation on almost any kind of data (medical, human, animals, videos, etc.). Some applications of image segmentation are :

- Tumor prediction, fracture detection, etc in medical imaging.
- Pedestrian detection, traffic light detection, vehicle detection, etc. in autonomous driving.
- Video surveillance, face detection, etc..

2.1 Statistical Segmentation Techniques

Image segmentation started developing in the early 1980*s* [20] and many innovations have been made since then. In statistics, there are many segmentation techniques based on the segmentation approach.

2.1.1 Threshold Segmentation

The process of segmenting an image by thresholding, based on the pixel values of an image is called "Threshold Segmentation". It converts a grayscale input image to a binary output image using a threshold constant **T**. The image intensity of each pixel is compared with a known fixed threshold value, the original pixel is replaced with a black pixel if the image intensity of the given pixel is less than the threshold value **T** and it is replaced with a white pixel if the image intensity of the given pixel is greater than the threshold value. The threshold value is either given by the user or can be automatically calculated (for example the mean 8-bit value of the original image). There are many methods to calculate the threshold value automatically. According to Sankur et. al [21] there are six automatic threshold calculating methods based on various features. Due to the limited number of calculations, threshold

segmentation can be executed very quickly. Choosing an initial threshold value is quite a challenge, it is often selected by taking the mean value of the grayscale image. If a histogram is plotted for the pixel intensity of both black and white pixels, the average of those peaks can be considered as an initial threshold value. When the threshold is calculated from the histogram of pixel intensity, there is a possibility for faster convergence of the algorithm.

2.1.2 Region-Based Segmentation

The region-based segmentation concentrates on the similarities between adjacent pixels. Pixels with similar attributes such as pixel value, image intensity, variance, color, etc. are grouped together into regions. When the adjacent pixel's attribute values differ less than a specific threshold value, they form a cluster called "regions". These regions are assigned with a unique integer label. Region segmentation works well even on complex images because, it is less susceptible to image attributes like noise, ambiguous borders, etc. Firstly grayscale analysis is performed on the image and then each region is assigned with a reference pixel. Each pixel is compared to its neighboring pixels using the reference pixel of a region and the pixel is added to that corresponding region if the difference of grayscale values is less than or equal to the threshold value. For example for every pixel p(i) in a region, there exists another pixel p(j) in the region such that the absolute value of its difference must be less than a chosen threshold value.

$$abs(I[p(i)]) - I[p(j)] < Threshold \tag{2.1}$$

This segmentation works well for color images. Region growing can be applied even to the image background areas by defining a lower limit on the acceptable pixel values. This can be used where there is a clear difference in foreground/ background threshold values, and when the areas of interest are only in the foreground. Region growing must also be limited, so that it does not cross over obvious edges in the image and get out of the image scope. This method can be combined with other segmentation methods like edge-based to get a more specific understanding of an image. Mueller et.al [22] used a combination of region and edge-based segmentation techniques to extract information and analyze shapes of fields, large man-made structures, etc., from satellite images.

2.1.3 Edge-Based Segmentation

Edges are the boundaries of objects in a scene or image. Edge detection is a process that tries to find edges based on the pixel intensity changes. In any image, edges represent a path of rapid change in image intensity. Edges provide information such as the topology and structure of the objects in an image. Edge detection can be used for various purposes like region segmentation, feature extraction, and object or boundary description. An edge detector was first built by Marr and Hildreth [23], it

was based on Laplacian of Gaussian(LOG). The images were firstly blurred by using a Gaussian filter and then the edges were enhanced and extracted using a Laplacian filter. The Edge localization is mainly done by finding zero-crossings. The process of edge detection in LOG is first to Convolve the image with a two-dimensional Gaussian function, then Laplacian (L) of the convolved image is computed and at the last edge, pixels are identified for those pixels where there is a zero-crossing in L. The Gaussian and Laplacian values in a 2D image is calculated by:

$$Gaussian : G_\theta(x,y) = e^{(\frac{-r^2}{2\sigma^2 0})}, \qquad Where \;\; r^2 = x^2 + y^2 \tag{2.2}$$

$$Laplacian : G(x,y) = (\frac{r^2 - \sigma^2}{\sigma^4})e^{(\frac{-r^2}{2\sigma^2})} \tag{2.3}$$

The width of the filter is determined by σ so that the filter is tuned to detect edges at different scales. σ controls the smoothing produced by the Gaussian component.

Later in 1986, The Canny edge detector was introduced by John Canny in his article "A Computational Approach to Edge Detection" [24], it addresses the fact that for edge detection, there is a trade-off between noise reduction (smoothing) and edge localization. The canny edge detector is a form of optimal edge detector. Firstly the smooth image is extracted with a Gaussian filter, then the gradient magnitude and orientation are computed. A non-maximal suppression is applied to the gradient magnitude image and the hysteresis thresholding is used to detect and link edges. The standard deviation σ, determines the width of the filter and hence the amount of smoothing. A filter with a large σ will suppress much of the noise, but also smooth away the weakest edges.

The Shen-Castan edge detector was introduced by Shen J and Castan in their article "An optimal linear operator for step edge detection" [25]. This detector uses an Infinite Symmetric Exponential Filter (ISEF) function. It gives better signal-to-noise ratios and localization than the Canny detector. Firstly, the image is convolved using the ISEF then the edges are localized by finding zero-crossings of the Laplacian (similar to the Marr-Hildreth algorithm). The quality of the edge pixels is improved by using false zero-crossing suppression in the Shen-castan edge detector. Zero crossings exist at the location of an edge pixel, this means the gradient is either maximum or minimum. If the change is +ve to –ve, it indicates positive zero-crossing. If the change is –ve to +ve then it is negative zero-crossing. The +ve zero-crossing is allowed to have a +ve gradient, and –ve zero-crossing is allowed to have a –ve gradient. All other zero-crossings are suppressed, highlighting only the correct edges. Later, many more edge detectors have been introduced for obtaining precise edges and the work continues.

2.1.4 Texture Segmentation

The texture is an image characteristic that is used to identify objects in an image. In images like an aerial photograph such as satellite, image objects are not precise so the classification can be done by examining its textures. The assumption is that textures belonging to the same class possess the same visual appearance. The image class can be identified by finding to which class does the considered texture belongs, this process is called texture recognition. Haralick et al. explained the importance of textural features in image classification in an article in 1990 [26]. This paper describes how textural features can be computed easily based on gray-tone spatial dependencies and illustrates how they can be applied to tasks like category identification. They applied texture segmentation on three types of image data namely photomicrographs, panchromatic aerial photographs of eight land-use categories, and Earth Resources Technology Satellite (ERTS) multispectral imagery containing seven land-use categories. Later Haralick et. al worked on the statistical approaches of textural segmentation [27]. Texture segmentation was used for letter recognition by Clark in 1987 [28] and Tuceryan in 1994 [29].

The textural feature extraction is carried out, firstly by describing the texture with some features which are related to its appearance and then the Feature Vectors (FV) are extracted.

- Texture $\rightarrow$ class $\rightarrow w_k$
- Subband statistics $\rightarrow$ Feature Vectors (FV) $\rightarrow x_{i,k}$

The distance measure for FV is defined such that it reflects the similarities and dissimilarities among textures. Classification rule must be chosen and the class is assigned to the texture based on which is the closest class in the feature space. The class that occurs maximum in an image becomes the image class.

2.1.5 Clustering

Clustering refers to organizing data points into groups based on similarities. Each cluster is different from the other in one or the other way. Clustering is used in many fields like data analysis, image analysis, pattern recognition, etc. It is not just an algorithm that binds objects together, in fact, it is a combination of various algorithms to efficiently find the similar qualities of various objects. Cluster analysis was firstly used in anthropology by Driver and Kroeber in 1932 [30] and then used in psychology by Joseph Zubin in 1938 [31] and later by many in psychology. Clustering has been used by various researchers in various fields. A.K. Jain et. al [32] provides a clear overview of how clustering algorithms can be applied to image segmentation, object recognition, etc.

There are various clustering algorithms namely Hierarchical clustering, Centroid-Based clustering, K-means clustering, Grid-Based clustering, etc. Every algorithm uses a different approach to define how relevant the objects are. Algorithms are chosen according to the type of application. The main steps of clustering are, first choosing the number of clusters, then random points are selected mostly the centroids (not necessarily from the dataset) of the clusters. Later each data point is assigned to the closest centroid that forms clusters (this step varies in all algorithms, the data point assignment is carried by calculating the distance between the point and cluster centroid in various ways in different algorithms). Then the new centroid of the cluster is computed and each data point is assigned to the new closest centroid. This reassignment and computation take place until there is no more change. Once the computation is completed, the model is ready.

All these statistical models showed a great effect on all fields, despite many advances and positive results, these statistical approaches consume an ample amount of computation time. An image consists of numerous annotated pixels and on average a statistical model requires around 30 minutes to go through one image. It is extremely stressful and time taking for models to go through hundreds of images. To reduce computation time and enhance the results, researchers have opted for machine learning and deep learning models.

2.2 Impact of Deep learning on Image Segmentation

Deep learning models give much precise information about an image, in much less time compared to the statistical models. Some standard deep network models made significant contributions in computer vision and their architecture is often used as a baseline in segmentation tasks. Networks like ALexNet [33], VGG16 [15], GoogleNet [34], ResNet [17] etc. showed award-winning results on ImageNet dataset. AlexNet is a Toronto model and it won the 2012 ImageNet competition with 84.6% test accuracy. VGG16 is an Oxford model, which won the 2013 ImageNet competition with 92.7% accuracy. GoogLeNet is a Google network, it won the 2014 ImageNet competition with an accuracy of 93.3%. ResNet is a Microsoft's model, it won the 2016 ImageNet competition with 96.4% accuracy. All these pre-trained models can be used for any computer vision task by using transfer learning. The aim of Transfer learning is to apply the knowledge and prediction ability gained from one task (source task) to a different but similar task (target task). Transfer learning is further explained in section 6.5. The CNN models like MobileNetV2 (refer section 6.8.1), PSPNet (refer section 6.8.2), U-Net (refer section 6.8.3) and Deep Lab V3 (refer section 6.8.4) are used in this thesis for fish segmentation and are clearly explained later in section 6.

2.3 Fish Segmentation

Usage of computer vision techniques like semantic segmentation has been rapidly increasing in many fields. Recently, underwater researchers are also introducing computer vision in their research about aquatic animals and their behavior. Chuang Yu et. al [35] proposed a scheme for segmenting fish images to measure morphological features of fish using Mask R-CNN. Previously used measurement methods were mainly manual, which was operationally complex, has low efficiency, and high subjectivity. So, the automated accurate measurement of the fish morphological features is of great significance. Rafael Garcia et. al [36] used Mask R-CNN to detect unwanted fish to overcome the problem of overfishing. This research helped to understand the population of each fish species so that undersized fish species can be targeted less in commercial trawling. The image nonlinearities of the camera response were corrected in pre-processing. Each individual fish was localized and segmented using Mask R-CNN. An accurate estimate of the boundary of every fish was obtained by using local gradients. The main advantage of this research was the ability to successfully deal with cluttered overlapping fish images. Several other researchers like Amanullah Baloch et. al [37] used camera images, Christensen et.al [38] used sonar data for the development of deep learning models for fish segmentation for various purposes. The usage of deep learning models is rapidly increasing day by day due to its performance and it is opening new doors for many researchers to experiment and experience.

Alshdaifat et. al [39] conducted research on underwater videos to track and monitor fish movements so that they can analyze the health and well-being of the marine Eco-system. They recognized that other existing methods during their research focused mainly on static underwater images and had several issues when applied to images extracted from videos. They proposed methods to improve the capability of deep learning networks for multi-fish instance segmentation. Sanchez Torres et. al [40] used a combination of 2D saliency maps, edge detection, and thresholding for fish segmentation. The length and weight of a fish are approximated from collected underwater images. The fishes were caught to manually measure length and weight which created great stress on the fishes, hindering their growth. This research was mainly conducted to reduce stress on fishes and to automate the process. Kumar Rai et. al [41] proposed a method to enhance the quality of underwater images using the adaptive histogram equalization method, then the segmentation was made using histogram thresholding. This research was performed to overcome the problems in images like low contrast, light scattering, noise, etc. since they are making the segmentation process difficult. Alzayat et. al [42] proposed a fully supervised segmentation model to monitor the fish stocks in fisheries. They used LCFCN loss and named their method as "Affinity-LCFCN" to improve the segmentation process. They could annotate a fish in a single click and it took on an average of 1 second per

fish. Jahidul et. al[43] proposed a fully convolutional encoder-decoder model named SUIM-Net to segment underwater images into 8 categories namely fish, coral reefs, aquatic plants, wrecks/ruins, human divers, robots, and sea-floor. The data used in the research was collected by humans and robots in underwater. This research gave a scope to increase underwater human-robot cooperative applications to analyze and segment aquatic life. They stated in their paper that the existing methods were either too application-specific or outdated, which created a need for an efficient, flexible (on data) and automated approach. They stated that their model was considerably faster and gave competition to the SOTA approaches.

Chapter 3

Background

3.1 Artificial Intelligence

The ability of machines to perform tasks similar to humans, by machine intelligence is considered as "Artificial Intelligence". The name "**ARTIFICIAL INTELLIGENCE**" was introduced by John Mccarthy [44] in the 1950*s*. Andreas Kaplan and Michael Haenlein defined AI as a

> "***System's ability to correctly interpret external data, to learn from such data, and to use those learnings to achieve specific goals and tasks through flexible adaptation.***"[45]

AI can be described in two categories "Narrow AI" and "Artificial General Intelligence (AGI)". When a machine performs a single task very well, but with so many constraints and limitations it is referred to as "Narrow AI". Narrow AI is often considered as weak AI and it is much less than human intelligence. Artificial General Intelligence is a very strong AI, these machines work much like human beings or try to overrule them. AI is applied to many areas like smart assistants (Siri, Alexa, Cortana, etc.), health care, banking, finance, manufacturing, defense, etc. AI has many subsets which deal with several domains namely pattern recognition [46], speech recognition [47], machine learning [48], deep learning [49], natural language processing (NLP) [50], robotics [51], etc. where each one of them deals with a specific task. "Machine learning" and "Deep learning" are often considered the backbone of Artificial Intelligence.

3.2 Machine Learning

Machine learning (ML) is a branch of Artificial Intelligence based on the idea that computers can learn from data, identify new data, and automatically improve from experience. ML models are those models, that provide statistical tools to explore and analyze the data, to perform required tasks. These machine learning models are used to solve real-world problems. In classical programming (figure. 3.1(A)) an algorithm should be explicitly coded using known features to generate the output. In contrast to this, ML (figure. 3.1(B)) uses subsets of data and the corresponding

output to generate an algorithm that may use novel or different combinations of features and weights to produce inference on completely new data. The learning strategy in ML is divided into four categories, each useful for solving different tasks: supervised, unsupervised, semi-supervised, and reinforcement learning.

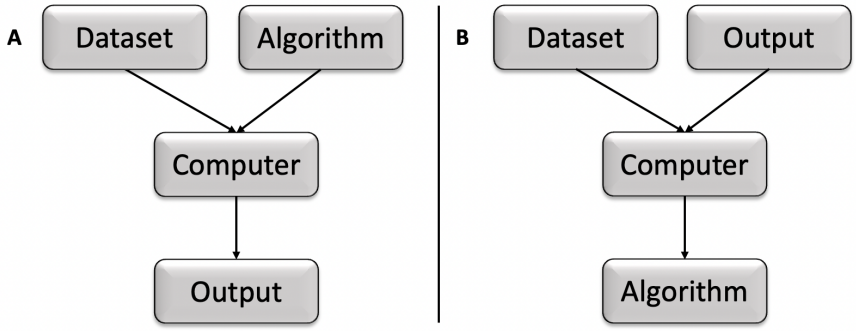

Figure 3.1: Classical programming versus machine learning paradigm. (A) In classical programming, both a dataset and an algorithm are provided to the computer. The computer learns from the algorithm provided and creates outputs. (B) In machine learning, a computer expected to learn the algorithm provided both the dataset and the corresponding outputs. The trained algorithm can be used for inference on future datasets. Inspired from: [52].

Logistic regression, Linear regression, Decision trees, and Random forests are types of machine learning models used for different tasks. Decision trees are flow charts like structures that contain nodes and branches where a node specifies a test for the variable, a leaf determines the class variable and each branch represents an outcome of the given test. Decision trees can be used even for solving tasks like classification and regression. Random forest is a collection or ensemble of decision trees, where each decision tree represents a scenario and gives a class prediction. The class that gets predicted by most of the decision trees is considered as the model final prediction. The random forest can also be used for predicting classification and regression outcomes.

3.3 Artificial Neural Networks and Deep Learning

Deep Learning is a subset of Machine learning where Deep Neural networks are a major part of it. A artificial neural networks (ANN's) [53] is a kind of neural network that is built on the working mechanism of biological neural networks. In ANN, nodes act as cell bodies to communicate with other nodes in the network via connections similar to axons and dendrites. These networks are able to learn to solve specific tasks by providing relevant data. The two major types of artificial neural networks are feedforward neural networks and recurrent neural networks (RNN) with a difference in the layer connections in the network structure. Feed-Forward Networks have a sequential layer structure, the layers can be segregated into three

major parts input layer, hidden layers, and an output layer as shown in the figure 3.2.

By adding more hidden layers sequentially to the ANN's architecture we can form a Deep Artificial Neural Network (DNN). The advantage of adding more hidden layers is that each layer adds its own level of non-linearity on the input thereby improving the efficiency of the model. The number of hidden layers must be optimized during the learning process of the network.

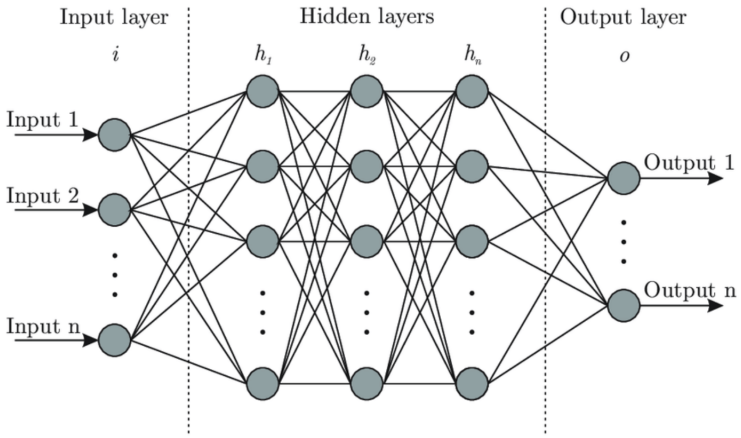

Figure 3.2: Basic Neural Network structure. Source: [54]

In feedforward neural networks the inputs are fed to the network through the input layer, passed through the sequence of one or more hidden layers where the model learns features from the inputs by doing complex calculations, then the output is obtained from the output layer. The processing of the inputs takes place in one direction and there are no feedback connections. RNN's has recurrent connections or feedback connections between the nodes where the output from the model is fed back to the network. The explanation of this architecture is out of the scope of this thesis.

The learning strategy in ML is divided into four categories, each useful for solving different tasks: supervised, unsupervised, semi-supervised, and reinforcement learning. Based on the research question at hand and the data available, the data scientist has to choose the specific learning strategy to train an algorithm.

- **Supervised Learning:** Learning a task under supervision is called Supervised Learning. In this method, the learning algorithm is provided with the data and their corresponding ground-truth labels during training. Every time an algorithm makes a prediction, it is compared with the ground truth label to evaluate the performance. So, the primary requirement for a supervised learning task is that one should have both the data and the labels. The widely used supervised learning tasks are regression and classification.

- **Unsupervised Learning:** The problem with supervised learning is that the algorithm requires a sufficient amount of labeled data to learn the task. In reality, the labeled datasets are not easy to generate. The process of annotating the data is a time-consuming, resource-intensive, and expensive task. That is where Unsupervised Learning comes into the picture. In this method, the ML algorithm is provided with the dataset without any explicit instructions on what to do with it. The algorithm attempts to automatically find similar patterns in the data by analyzing data distribution and by extracting useful features from the data.

- **Semi-supervised Learning:** Semi-supervised learning is a combination of both Supervised and Unsupervised learning strategies. This strategy is particularly useful for datasets that contain a mixture of labeled and unlabeled data. This situation arises when labeling images are constrained with time and cost. This method is often used for medical images, where a radiologist or a physician labels a small subset of images. The network is trained on this small subset and used to classify the rest of the unlabeled images in the dataset thereby increasing the size of the training data. The network is retrained on the new data to produce more accurate classifications. This process is repeated iteratively until an effective model is produced. In theory, the models trained with semi-supervised strategy should outperform unsupervised models.

- **Reinforcement Learning:** Finally, reinforcement learning is the technique of training an algorithm based on the policy of rewarding desired behaviors and punishing undesired ones. The algorithm has to predict an outcome from the multiple possible values based on the reward. The reinforcement learning agent is able to interpret its environment by taking actions learned from trial and error. It is considered as one of the closest attempts at modeling the human learning experience because it also learns from trial and error rather than data alone.

A Supervised learning strategy is followed to train the segmentation models in this thesis work.

Chapter 4

Convolution Neural Networks (CNN)

In machine learning, a convolutional neural network (CNN) is a class of deep artificial neural networks, more specifically multi-layer feedforward networks that have successfully been applied to analyze visual images. Similar to ANN's, CNN's also take inspiration from the human brain, more specifically from the visual cortex. Apparently, different neurons in the specific regions of the visual cortex of the brain are sensitive to different patterns or colors and respond to specific actions. These groups of neurons are distributed in an hierarchical fashion in the brain. The neurons at the bottom of the hierarchy detect very simple patterns such as lines while the neurons at the top of the architecture detect much more complex patterns. The core building blocks of CNN's architecture are a special type of layer called the convolutional layer. They have been pioneered by Yann LeCun in the 1990*s* [55] and successfully applied to tasks like handwritten digit recognition, general image classification, object detection, and semantic segmentation.

The basic architecture of a CNN or ConvNet is a combination of an input layer, hidden layers, and an output layer. The middle layers in the feed-forward neural network are called hidden layers because their outputs are the estimation of some kind of activation functions. In a convolutional neural network, the hidden layers are the combination of convolutions, pooling operations, and some activation function. Typically, this includes a layer that does dot product, and its activation function is commonly ReLU [56]. These are followed by other convolution layers such as fully connected layers, and normalization layers. Each layer in a CNN is characterized by a set of filters that is applied to the data received from the previous layer (in this context called feature map). Filters are spatially smaller than the input image but are more in-depth. They are able to detect different spatial patterns in the input image and can determine where each pattern occurs by sliding across the input. From the mathematical point of view, the operation that describes the application of the filters is the convolution. At each location of the filter, the product between the filter element and the input element it overlaps is computed and the results are summed up to obtain the output in the current location. The process is repeated to produce

all the output values. Therefore, the parameters to be learned during training are the filters, unlike in fully connected layers where the network learns all interconnections between neurons. Additionally, another advantage of CNN's is that by "sliding" the same filters across the feature maps, we can deal with inputs of arbitrary spatial size. On the other hand, fully connected layers require the input of a fixed size, because the number of connections to be learned has to be determined in advance. The problem of applying these networks is one of the supervised learning tasks, and mathematically can be formalized as follows:

$$F(\mathbf{w}) = \min_{\mathbf{w} \in \mathbb{R}^n} \{\frac{1}{|X|} \sum_{x \in X} f(x, \mathbf{w})\}$$

where F(w) is a loss function, w is the vector of weights being optimised, n is the dimension of the weight vector w, X is a labeled training set, and f(x,w), is the loss computed from samples $x \in X$ and their labels y. The process of optimising the loss function F(w) is also called training of the network. Stochastic Gradient Descent (SGD) and its extrensions are often used for training convolutional networks.

4.1 Convolutional Layers

A convolution layer applies a convolution operation on an input image to generate a number of feature maps. A sliding window approach computes the features over localized regions of an image. This makes the transformations more flexible and the computation efficient without much negative impact on spatial dependencies of the features. The convolutional kernels are defined as $height \times width \times channels$, where channels correspond to the depth of the kernel (for instance an RGB image has 3 channels). There are several hyperparameters in the convolution layers of a CNN namely:

1. **Kernel size:** The kernel size defines the field of view of the convolution operation on the input volume. It is defined as the matrix that moves over the sub-region of pixels in an input image. A group of kernels stacked together is often defined as a filter and they are used to extract various features from the images. The number of kernels controls the depth of the output.

2. **Input Depth:** "*Input depth refers to the number of stacked feature maps received from the previous layer* [57]." For RGB color images, the depth is defined by 3 color channels, whereas grayscale images have a depth of 1.

3. **Output Depth:** Output depth is defined as the number of filters contained in a layer. Each kernel will end up generating a single feature map. All the generated feature maps are stacked together and passed forward to the next layers in the network. As a result, the input depth in the next layer is equal to the number of filters in the current one.

4. **Stride:** Stride is the parameter used to control the movement of the kernel over the input volume. If the stride is 1, the filter convolves the input volume by shifting one block of window (defined by the size of the kernel) at a time. This parameter is by default set to 1.

5. **Padding:** Padding is the process of adding zeros at the borders of our input images to prevent them from shrinking. By padding, one can preserve the height and width of the input volume.

All together, these parameters define a convolution operation. For example, figure 4.1 represents a 2D convolution with a kernel size of 3, the stride of 1, and padding.

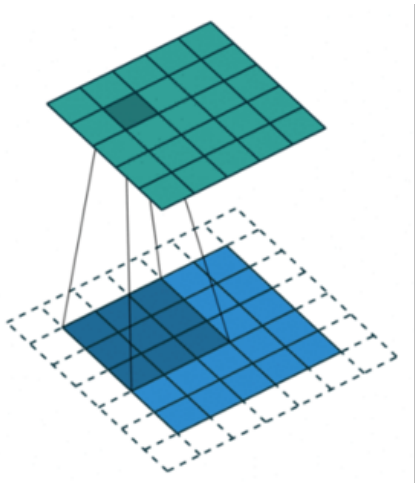

Figure 4.1: Example of 2D convolution. Source: [58]

The shape of the output after any given convolution layer is calculated by the formula:

$$\frac{W - K + 2P}{S} + 1 \tag{4.1}$$

Where,

- W is the input volume
- K is the kernel size
- P is the padding and
- S is the stride.

4.1.1 Understanding Standard Convolution Operation

Consider a typical RGB image of $(12 \times 12 \times 3)$ pixels where (12×12) is the resolution of the image in terms of height and width and 3 is the number of color channels. The number of color channels can also be considered as the depth of the input image.

Let us apply a (5×5) convolution operation on this input image with no padding and stride 1. As the input image has 3 channels our convolution kernel needs to have 3 channels as well. The output of the convolution operation is $(8 \times 8 \times 1)$ after applying the formula as in 4.1.

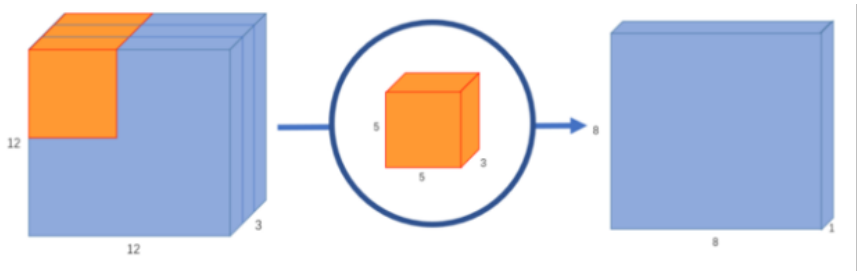

Figure 4.2: A normal convolution with single kernel. Source: [59]

If the desired size of the output is $(8 \times 8 \times 256)$, it is necessary to include 256 kernels, so that each kernel creates an output of size $(8 \times 8 \times 1)$ and finally, stack all the outputs together to form $(8 \times 8 \times 256)$ image output.

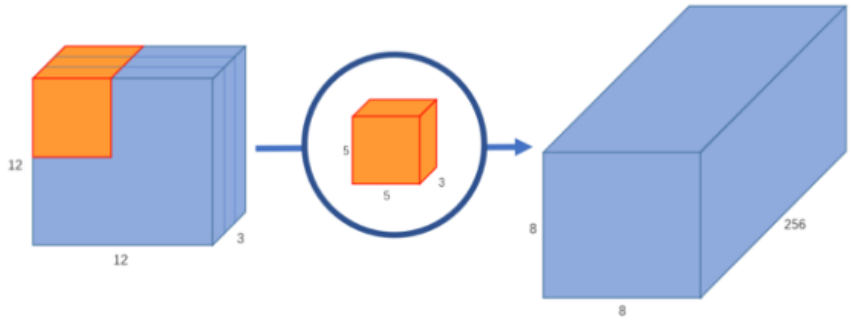

Figure 4.3: A normal convolution with 256 kernels. Source: [59]

There are many extensions to the standard convolution layers and it is necessary to understand those extensions as they were used in training the deep neural networks to achieve the aim of this thesis work. Every Convolution technique has its own influence on predicting the output. The various types of convolution layers used in this thesis are Transposed Convolution, Atrous Convolution, and Depthwise Separable Convolution.

4.1.2 Transposed Convolution

The convolution operation creates an abstract representation of an input image by decreasing the spatial dimensions as we go deeper down the network. However, there are few instances where it requires to upsample feature maps from low resolution to high resolution. This is especially important in tasks like semantic segmentation where we want to reconstruct the input image as a segmented image of the same dimension. Figure 4.4 shows some techniques that can be used to do upsampling of a feature map that does not require trainable parameters. These are traditional interpolation techniques. Generally, upsampling operation is the opposite of standard

pooling operation and these techniques only depend on the content of the feature maps and the desired factor of upsampling.

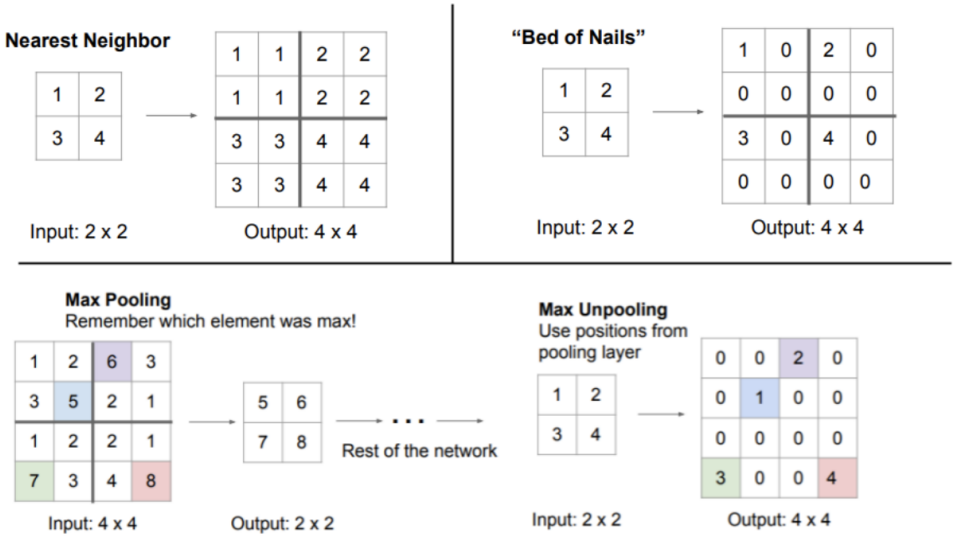

Figure 4.4: Techniques for upsampling images that don't require trainable parameters. Source: [9]

Upsampling the inputs using interpolation techniques is not always flexible as they simply compute the estimate of the interpolated pixel values. It is not a good choice when there is a need to make images larger without losing small details in the image. To solve this problem, transposed convolution also known as fractionally-strided convolution comes into the picture. It can be used to upsample input feature maps to desired output feature maps. These convolutions have trainable parameters that can be optimized during the training process. In other words, it tries to learn the optimal, or most correct, upsampling while the model is being trained. In equation 4.2 the standard convolution (1D) is expressed as a matrix multiplication:

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix} \tag{4.2}$$

Source: Inspired from [9]

In equation 4.3 the transposed convolution (1D) is expressed as a matrix multiplication:

$$\vec{x}^T * \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix} \tag{4.3}$$

Source: Inspired from [9]

A transposed convolution operation is simply a convolution operation using a transposed convolution matrix, X^T. The weights in X and X^T are not the same. The Eq: 4.3 merely shows that upsampling can be done using convolution operation. The important part is the connectivity pattern of X^T. It describes how individual pixels in the input is contributing to multiple pixels in the larger output. A many-to-one relationship is seen in transposed convolution, unlike the one-to-many relationship we see during a standard convolution operation.

4.1.3 Atrous Convolutions

Atrous Convolutions can also be referred to as dilated convolution. They introduce a new parameter to the convolutional layer called the dilation rate. Dilated convolutions incorporate larger context in the filters by enlarging their field of view (or receptive field). The filter's field-of-view is enlarged by upsampling the filter kernels by inserting holes between filter weights. Originally, researchers invented a hole ('trous' in French) algorithm for wavelet transformation [60] and it is later used by researchers for convolution in the deep neural network. Its effective mechanism controls the field-of-view of the filter by finding the best trade-off between accurate localization (small field-of-view) and context assimilation (large field-of-view). The resolution at which feature responses are computed with DCNN without requiring learning extra parameters can be controlled using atrous convolutions. These type of convolutions are very popular in real-time segmentation as they eliminate the bottleneck of adding extra parameters to achieve wider field of view.

$$y[i] = \sum_{\mathbf{k}} x[i + r.\mathbf{k}]w[\mathbf{k}] \tag{4.4}$$

In the Eq. 4.4 atrous convolution is applied over the input feature map x with dilation rate r. The location on the input x is represented by i and the filter is represented by w. The dilation rate indicates number of zeros to be inserted between two consecutive filter values along each spatial dimension. If the dilation rate is r,

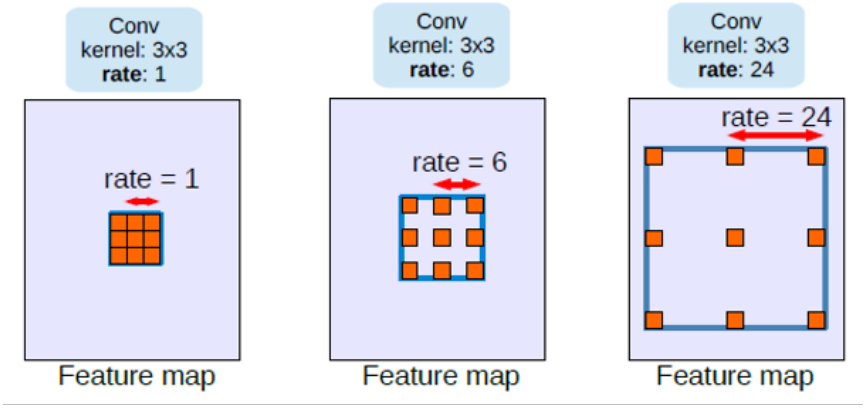

Figure 4.5: Atrous convolution with different rates. Source: [19]

$r-1$ zeros are inserted. When the dilation rate r is set to 1, it behaves as a standard convolution operation.

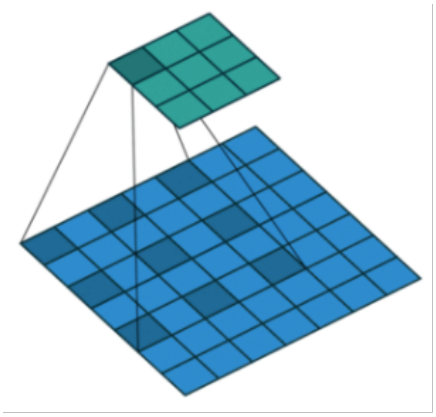

Figure 4.6: Example of 2D dilated convolution. Source: [58]

In figure 4.6 a 2D filter of size (3×3) with dilation rate $r = 2$ is applied on the input size (7×7). The field-of-view of the filter in this case is (5×5) instead of (3×3) as it is a dilated convolution.

4.1.4 Depthwise Separable Convolutions

The main idea of depth-wise separable convolution is to separate the full convolution operator into two parts, depthwise convolution and pointwise convolution.

- **Depthwise convolution:** In depthwise operation, each channel of the input is operated with a single convolution filter. This is in contrast to standard CNN's in which the operation is applied on all the channels of the input. Let us consider the same example input RGB image of size $(12 \times 12 \times 3)$ and kernels of $(5 \times 5 \times 1)$ shape. As the kernel is applied on the single channel of the input image it outputs an $(8 \times 8 \times 1)$ image for each channel and stacking these images together creates an $(8 \times 8 \times 3)$ output. Depthwise convolution uses 3 kernels to transform an RGB image of $(12 \times 12 \times 3)$ to an $(8 \times 8 \times 3)$ image.

As only one kernel is applied per input channel, the depthwise operation is considered as a lightweight operation than standard convolution operation.

Figure 4.7: Depthwise convolution, uses 3 kernels to transform image of $(12 \times 12 \times 3)$ to a $(8 \times 8 \times 3)$ image. Source: [59]

- **Pointwise convolution:** As discussed in section 4.1.1, the standard convolution transformed a $(12 \times 12 \times 3)$ image to an $(8 \times 8 \times 256)$ image. Unlike standard convolution, the depthwise convolution transformed the image of size $(12 \times 12 \times 3)$ to an image of size $(8 \times 8 \times 3)$. Both the outputs vary in their output depth. So, the depth of each image must be increased to make the outputs equivalent. This is where pointwise convolution operation comes into the picture. The pointwise convolution iterates a (1×1) kernel through every single point of the input. This kernel can have a depth of however many channels the input image has; in this case, it is 3. Therefore, a $(1 \times 1 \times 3)$ kernel must be iterated through our $(8 \times 8 \times 3)$ image, to get an $(8 \times 8 \times 1)$ image.

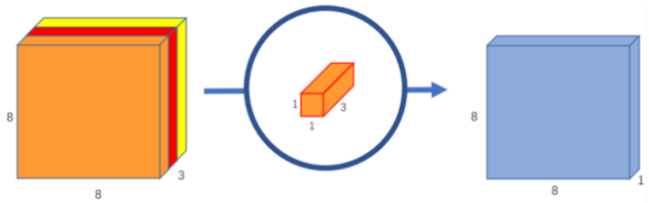

Figure 4.8: Pointwise convolution transforming image of 3 channels to image of 1 channel. Source: [59]

To obtain a final output image shape of $(8 \times 8 \times 256)$, it requires 256 $(1 \times 1 \times 3)$ kernels that output an $(8 \times 8 \times 1)$ image each.

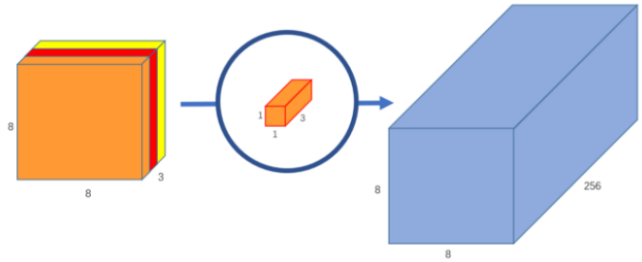

Figure 4.9: Pointwise convolution with 256 kernels, outputting an image of 256 channels. Source: [59]

As described above the convolution is separated into 2 parts, a depthwise convolution and a pointwise convolution. In more simple terms, the transformation of input by standard convolution is illustrated as $(12 \times 12 \times 3) \rightarrow (5 \times 5 \times 3 \times 256) \rightarrow (12 \times 12 \times 256)$. The transformation by the new convolution is expressed as $(12 \times 12 \times 3) \rightarrow (5 \times 5 \times 1 \times 1) \rightarrow (1 \times 1 \times 3 \times 256) \rightarrow (12 \times 12 \times 256)$.

4.2 Activation Layer

An activation layer is used to infuse some non-linearity in an input-output relationship, making CNN a universal function approximator. The most common form of an activation function in a CNN is the Rectified Linear Unit (ReLU) which suppresses the negative values in the features by clipping them at 0. ReLU can be defined as,

$$f(x) = max(0, x)$$

Where x is a value in the input feature map to the activation function. Visually the ReLU activation function looks like the following:

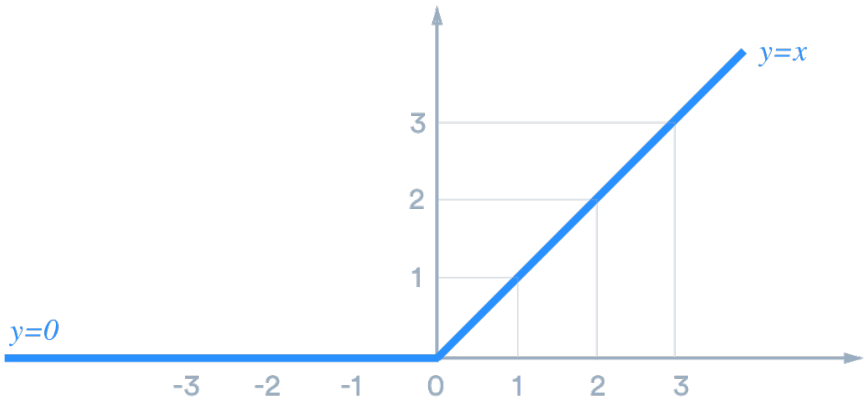

Figure 4.10: Representation of ReLU Activation Function. Source: [61].

From the figure 4.10, it should be noted that for any value in the input feature map ReLU is linear (identity) if it is a positive value and zero if negative. Due to this effect, the computations become cheap as there is no complicated math involved. Therefore, the model can take less time to train or infer with the application of ReLU activation.

4.3 Pooling Layers

The concept of pooling refers to the downsampling of features by using a consensus like average or max over localized regions. The most common form of this is max-pooling which selects the maximum value over a small kernel window and slides this over the feature map to create a smaller-sized representation. There are

however other forms of pooling like averaging and they can also be applied over the entire feature map to form a global representation. Figure 4.11 shows a sample for average pooling and max-pooling operation over a 2-dimensional matrix with a (2×2) kernel and slide length of 2.

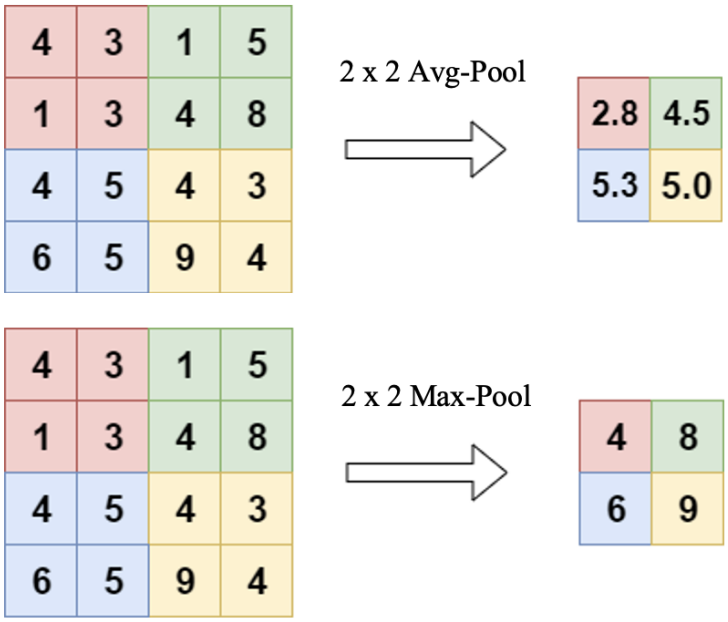

Figure 4.11: Example showing effect of Average pooling and Max-pooling operations. Source: [62].

The pooling layers generate the summary of the features present in a sub-region of the input. The downsampling of feature maps has an important advantage in CNN. It reduces the number of parameters in the network thereby considerably reducing the computation time of the model.

4.4 Fully Connected Layers

A Fully Connected (FC) layer is a linear layer that connects every neuron between two consecutive layers. The activations between two FC layers can be computed as matrix multiplication and by adding bias offset. After the feature extraction from the convolutional layers, the data needs to be classified into different classes. This can be done using FC layers. Although FC layers work efficiently in classifying the data by extracting relevant features from it, they are not useful in practice to deal with complex data such as images, video, or audio. There are two main reasons for this. Primarily, because these types of data are generally represented in a very high dimensional space and the number of neurons and consequently computational power required even to train a relatively shallow network is prohibitive. The secondary reason is that there is a much more efficient approach to exploit the intrinsic spatial structure of the data using Convolutional Neural Networks (CNN's).

A Fully Connected (FC) layer is a linear layer that connects every node or neuron to every node of the output. This method allows feature learning over the entire input and the architecture is supposed to emulate the neural connections in a human's brain. Mathematically it can be represented by a matrix multiplication as,

$$f(X, \theta \sim) = W.X \sim + b$$

where X is a vector input of size D, W is the weight matrix i.e. the values for each node of the linear layer of dimensions D×O with O being the output dimension and b as a bias term that is added to the feature.

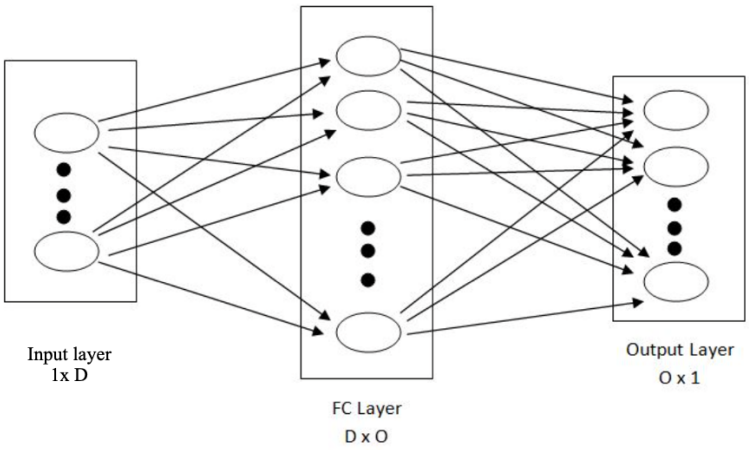

Figure 4.12: Example for single Fully Connected layer mapping an input to an output.

In classification based CNNs, the final layers tend to be a collection of one or more FC layers. A softmax [63] operation can be utilized on the output of the last layer to generate probabilities over the number of classes.

4.5 Receptive Field

When dealing with high-dimensional data such as images, it is impractical and computationally expensive to connect neurons in a fully connected fashion. Instead, we connect each neuron to only a small local region of the input volume specified by the filter size. The small local region is called the receptive field. The size of the receptive field roughly indicates how much context information is used for computing the activations and generating the feature maps. The size of the effective receptive field is defined by the kernel size. There is a linear increase in the size of the receptive field as we stack more convolutional layers and an exponential increase as we stack more atrous convolutions. During the training of the neural network the size of the effective receptive field acts as a hyperparameter.

4.6 Output Layer

Output layers are generally the last layers in a neural network and are responsible for generating the final result from the model based on the use case for which the network has been designed. In most classification-based CNN architectures the last layer is a fully connected layer as explained in chapter 4.4. The FC layer takes input in the form of flattened data from the previous layer and accumulates all the signals from each dimension to introduce generalization ability. However, in most segmentation tasks, this layer is not an appropriate choice as it highly increases the computational cost. In the semantic segmentation task, the shape of the output from the model is the same as the shape of the input. In our case, the input and output shapes are (512×512). One can use either softmax or sigmoid activation for the output layer to predict the probabilities for each of the neurons in the output layer. In general, the sigmoid activation function is used for the segmentation or classification of images with only 2 classes (binary), and the softmax activation is used for the tasks with multiple classes.

Sigmoid activation function: A Sigmoid function is a mathematical function, returns a value in the range of 0 to 1 for any input given. It has a characteristic "S"-shaped curve or also called a sigmoid curve. This function is a very good choice where one has to predict the probability as the network output as the probability of anything exists only between the range 0 to 1.

A typical example of a sigmoid function is a logistic function shown in the figure 4.13 and it is defined by the formula:

$$S(x) = \frac{1}{1 + e^{-x}}$$

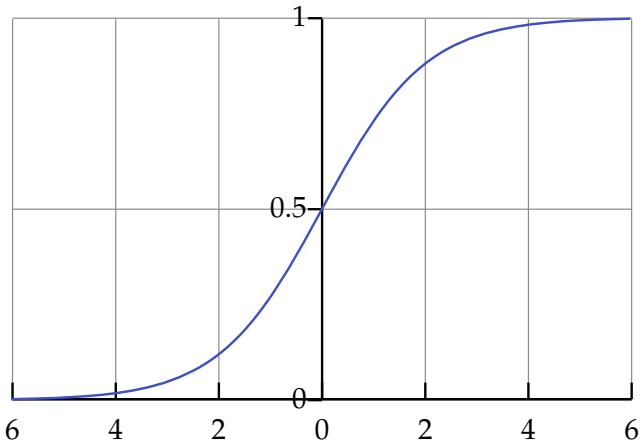

Figure 4.13: Logistic function an example of Sigmoid function.
Source: Inspired from [64]

The curve in the figure 4.13 explains that if the value of x approaches the positive infinity then the predicted value of S(x) will be 1 and if it approaches the negative infinity the predicted value of S(x) will be 0.

Chapter 5

Data

Some of the public datasets for underwater fish detection are Fish4Knowledge [65], DeepFish [66], QUT [67], Rockfish [68], etc. The tasks that these datasets address are limited to classification i.e. to distinguish between fish species. Fish4Knowledge and Rockfish also address the task of detection where the goal is to draw a bounding box around the fish [66]. However, there is a lack of well-annotated open-source datasets that are suitable for the segmentation of fishes in underwater videos. Therefore, an underwater fish dataset was created specifically for semantic segmentation for this study. This chapter presents the images used for segmentation and how they were obtained. It also describes the segmentation masks for the corresponding images and how they were annotated.

5.1 Data Collection

A joint project "**UFOTriNet**" is carried out by German universities and marine engineering companies to observe and collect underwater sensor data. The sensor data is collected by a Underwater Fish Observatory (UFO) prototype. The project is publicly funded by the Federal Ministry for Food and Agriculture in Germany. The UFO sensor platform was deployed at seafloor level in the North Sea, about 45 nautical miles east of the island of Sylt, and in the Kiel Fjord, an inlet of the Baltic Sea in northern Germany. The primary goal of deploying the prototype is to continuously record videos and sonar data for various oceanic parameters. The raw data was approximately collected for about 240 days only in the daytime, which is about 3000 hours of video footage. The dataset used in this thesis is a subset of the complete optical data suited for the development and evaluation of computer vision algorithms, i.e. no abiotic measurements or sonar data are included in the dataset.

The videos were recorded using two Photonis "Nocturn XL" monochrome CMOS image sensors, these are specially designed for underwaters with a flat view port. The sensors have an optical resolution of (1024×1280) pixels and a sampling rate of up to 20 frames/second. The cameras are well designed to work in low-light scenarios. No extra lighting was required during the daytime, even on cloudy days. It

could record data up to a maximum depth of 22m. The extra lighting was purposely avoided to be as less invasive as possible and to record the natural behavior of fish.

5.2 Images

The underwater videos were split into many frames of images. All the frames in the dataset are of the resolution 1024×1280 $(height \times width)$. The frames only have a single color channel which means all the frames are gray-scale images.

Figure 5.1: Sample images from the dataset.

5.3 Masks

Supervised learning of semantic segmentation task requires annotations for all individual pixels of an image. The annotations are needed for the training, validation, and testing of the developed model. The provided images initially did not have any ground-truth per-pixel segmentation labels. The images were annotated with the help of the Computer Vision Annotation Tool (CVAT) and the binary segmentation masks were generated for 400 images. The mask of the images is a binary mask which means each of the pixels in the image is labeled as a fish (white pixels) or no fish (Black pixels). In terms of pixel intensity values, all the pixels with a value of 255 indicate a fish and all the pixels with a value of 0 indicate no fish.

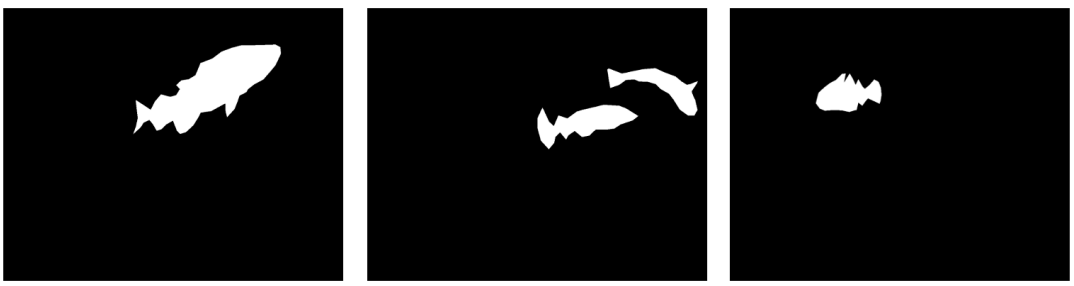

Figure 5.2: Sample masks from the dataset generated by the annotation tool.

5.4 Bounding Box Labels

For each of the images in the dataset, corresponding bounding box labels for fishes in the images are presented in the form of text files. The text file contains the top-left and the bottom right coordinates of the rectangular bounding box region of the object. If there are multiple objects in the image then they are represented with multiple rows in the labels file where each row corresponds to a single object.

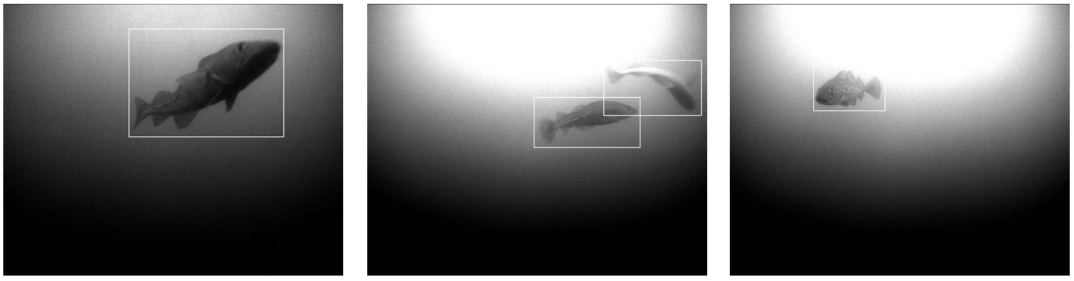

Figure 5.3: Sample images with bounding box regions from the dataset.

5.5 Image Annotation Tool

A supervised learning algorithm requires an adequate amount of labeled or annotated samples to effectively learn the features. As the aim of this thesis was to perform classification at the pixel level (Segmentation), every single pixel in the image has to be labeled with its corresponding class label. Compared to annotating the image at the object level (for classification task), this is particularly time-consuming. To facilitate this work I have used a free, open-source, web-based tool called "Computer Vision Annotation Tool". Users can create segmentation maps from the tool by clicking on the selected object regions of the images.

CVAT:

It is an interactive tool designed for labeling image and video data used for computer vision algorithms. One can label using CVAT [1] for all the primary tasks of supervised machine learning like image classification, object detection, and image segmentation (semantic and instance). CVAT provides flexibility to the annotators by its powerful features. It provides features to include the interpolation of shapes between keyframes and shortcuts for most critical actions during the annotation process. One can also make use of the ability for semi-automatic labeling [2] using deep learning models. It has a dashboard interface showing the users with a list of ongoing annotation projects and tasks, LDAP (Lightweight Directory Access Protocol), and a basic access authentication facility. CVAT is mainly written using scripting

[1] https://github.com/openvinotoolkit/cvat

[2] https://www.youtube.com/watch?v=9HszWP$_q$sRQ

languages like Python, React, TypeScript, CSS, Ant Design, and Django. It is an open-source project distributed under the MIT License and its source code is available online for free of cost.

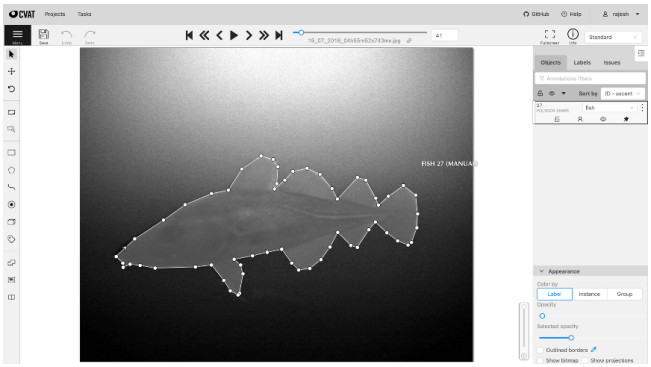

Figure 5.4: CVAT interface for annotating images.

Categories considered during the annotation process of the segmentation task are:

- Fish/foreground.
- No fish/background.

In figure. 5.4, an example of the annotation of the single fish object in the image is visualized. All the points forming a polygon indicate the fish object and are distinguished from the background pixels. Once the annotation is saved, the labels can be exported in multiple formats like Extensible Markup Language (XML), Common Objects in Context (COCO), Portable Networks Graphics (PNG), etc. We exported the binary masks per image in the form of PNG images.

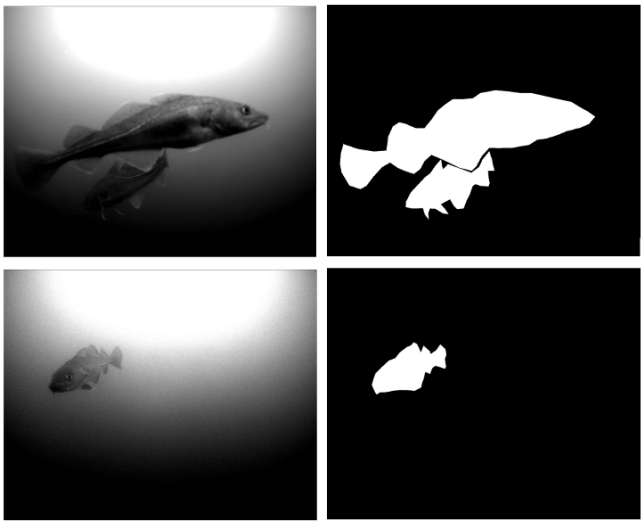

Figure 5.5: Samples of images from the dataset and their corresponding segmentation masks from the annotation tool.

Some of the labeled images (with corresponding masks) from the CVAT tool are shown in figure 5.5

5.6 Input Pipeline

This section explains the type of pre-processing applied to the input images before passing them through the network for training.

5.6.1 Data Pre-processing

All the images are resized to a target resolution of (512×512) pixels to train the deep neural networks which is discussed later in the chapter. The method used for resizing the images is **bounding-box cropping.**

Bounding-Box-Cropping

It is a pre-processing technique used in the current work. As discussed in section 5.4, for each image in the dataset there are corresponding bounding box annotations for the fish objects in the images. The height and width of the bounding box are calculated from the given label coordinates and if they are not equal to the required input shape then the images are cropped. If cropping of the images is not possible (in case if the crop size is beyond the image boundary) then they are either upsampled or downsampled.

Let, $(x_1, y_1), (x_2, y_2)$ are top-left and bottom-right coordinates of the object bounding box, the width, and height of the box are represented as below:

$$width = (x_2 - x_1) \tag{5.1}$$

$$height = (y_2 - y_1) \tag{5.2}$$

If there are multiple fishes in the image, then a new bounding box is created enclosing all these objects into one bounding box region. Let, $(x_{11}, y_{11}), (x_{12}, y_{12}), (x_{21}, y_{21}), (x_{22}, y_{22})$ be the bounding box coordinates of two fishes in an image. The bounding that is enclosing these two objects is calculated by:

$$(x_1, y_1) = (min(x_{11}, x_{21}), min(y_{11}, y_{21}))$$

$$(x_2, y_2) = (max(x_{12}, x_{22}), max(y_{12}, y_{22}))$$

Afterwards, width and height are calculated by Eq.: 5.1, 5.2.

If there are n fishes in a single frame of the video, the height and width of the bounding box region are calculated as the following:

Algorithm 1 Bounding box region estimation

function fetch_bbox_coordinates($bbox\_file$)
  for Every object in the $bbox\_file$ **do**
    Read (x_1, y_1) and (x_2, y_2)
    if $n\_objects$ **then**
      $(x_1, y_1) = (min(x_{11}, x_{21}, ..., x_{n1}), min(y_{11}, y_{21}, ..., y_{n1}))$
      $(x_2, y_2) = (max(x_{12}, x_{22}, ..., x_{n2}), max(y_{12}, y_{22}, ..., y_{n2}))$
    end if
  end for
  $width = (x_2 - x_1)$
  $height = (y_2 - y_1)$
return $width, height$
end function

Method for bounding box pre-processing:

- Size of the original image (1024×1280).
- Size of the target image (512×512).
- Get the bounding box coordinates for the fish objects in the image samples using the labels file.
- Get height and width of the bounding box region using equation 5.2, 5.1.
- If the calculated height or width greater than the target dimensions, resize the image to the target shape.
- If the height and width less than the target dimensions, then take the difference of height and width and divide it by 2.
- Extend the image in both the X and Y directions equally by increasing the pixel coordinates in the X direction and decreasing the pixel coordinates in the Y direction, to make sure the object lies in the center of the crop region.
- If the extension of the crop region goes beyond the boundary in either of the direction, take the maximum crop region until the boundary edge and then resize to the target dimension.

The algorithm describing the above pre-processing method is shown below. The terms *bw* and *bh* in the algorithm represent bounding box width and bounding box height.

Algorithm 2 Bounding box cropping

function preprocess_input_with_bb(*image*, *mask*)
  Size of the original images and masks in the dataset $H = 1024, W = 1280$
  Size of the target images and masks $h = 512, w = 512$
  for Every image, mask in the dataset **do**
    Read (x_1, y_1) and (x_2, y_2) (using the algorithm 1).
    Get bw and bh from Eq.:5.1 and Eq.:5.2.
    if $bw < w$ **then**
      Increase the width of the box by enlarging (x_1, x_2) in both directions
    end if
    if $bh < h$ **then**
      Increase the height of the box by enlarging (y_1, y_2) in both directions
    end if
    Crop the input image and mask with the updated coordinates (x_1, y_1, x_2, y_2).
    if $bw > w$ **then** or $bh > h$
      Resize the input image and mask to (h, w)
    end if
  end for
return *image*, *mask*
end function

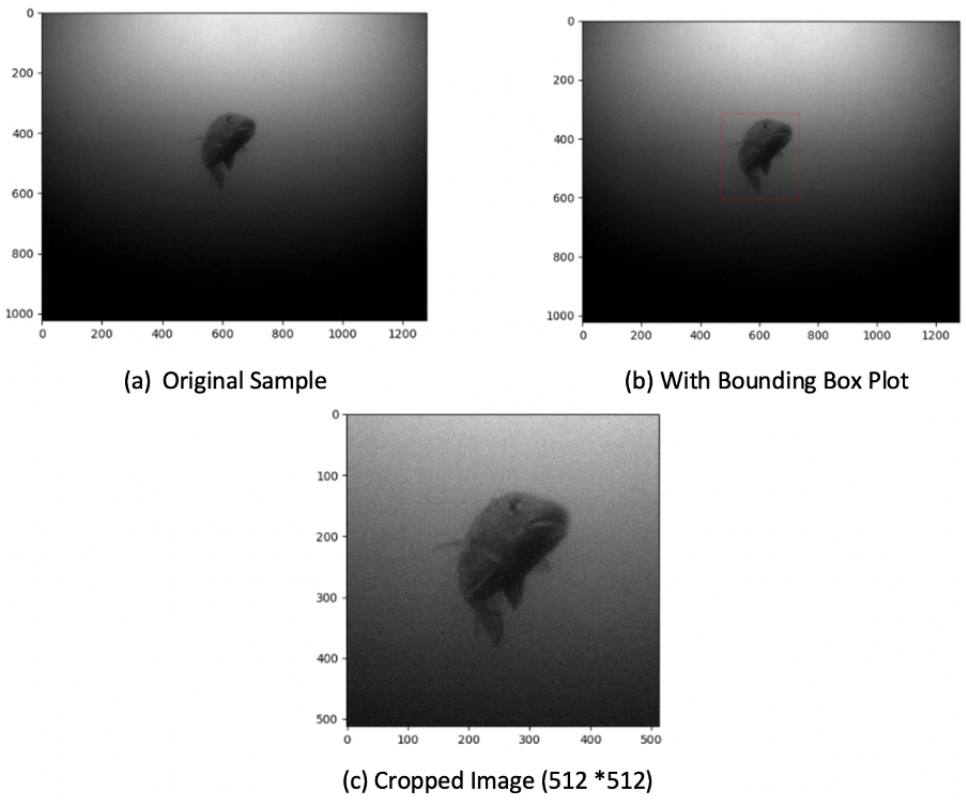

Figure 5.6: a) Fish image sample from the dataset, b) Bounding box plot of fish and c) Final cropped image after pre-processing.

5.6.2 Output Post-processing

The segmentation output from the trained network is a 512×512 matrix with each pixel being a floating-point value between 0 to 1. This is because the output layer uses a Sigmoid activation function as defined in section 4.6. The values are clipped to either 0 or 1 based on the threshold setting (discussed in Chapter 8). In addition to this, bounding box post-processing is also applied to the predicted segmentation output which is similar to input pre-processing. Here, the new bounding box coordinates are calculated from the original coordinates by taking into account the number of pixel coordinates that are added or subtracted during pre-processing 5.6.1. This new bounding box region is the Region of Interest (ROI) and all the pixels out of this region are deactivated or modified to be pixels with a value of 0. This process reduces the false positives because if there are any pixels that are outside the ROI classified as fish, they are re-classified to be background after post-processing.

5.7 Data Augmentation

The process of synthesizing new data by adding slight modifications to the available data is called data augmentation. It is a strategy that enables the machine learning developers to significantly increase the diversity and the amount of the training data without actually collecting the new data. As in real-world scenarios, the collection of high amounts of data is not always feasible or sometimes very expensive and in such cases, the data augmentation technique is more useful.

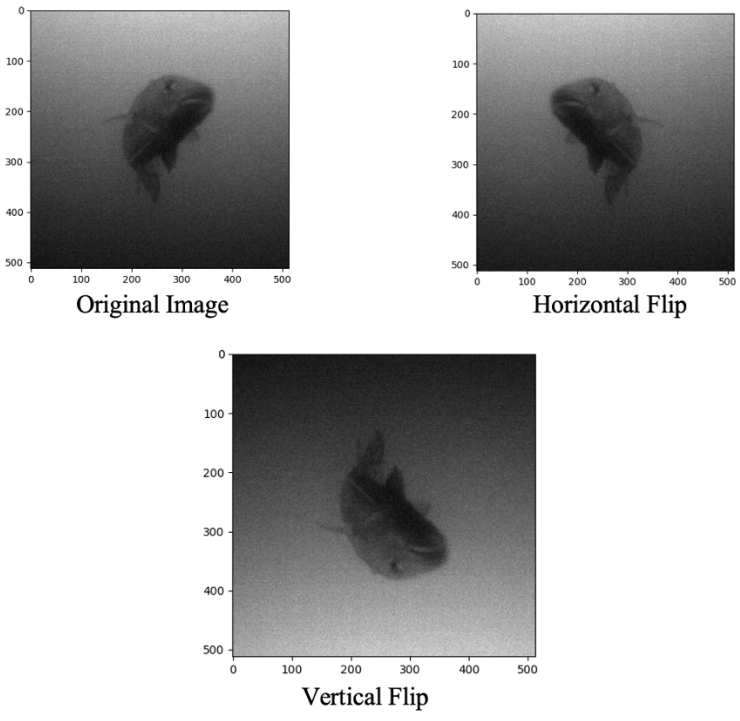

Figure 5.7: Real-time augmented data samples.

During the re-training and fine-tuning phases of the DCNN networks, real-time data augmentation is performed. In particular, at the beginning of each epoch, training images and corresponding masks in a mini-batch are randomly distorted to obtain more diverse data that would help prevent overfitting of the model. The usage of data augmentation was limited to random horizontal flipping and random vertical flipping in this work. They were applied with probabilities of 0.5 which means, each sample from the training set has a 50% probability of being flipped in the horizontal and vertical directions. The resulting images after data augmentation are shown in figure 5.7. This process of applying augmentations to images that change the position of objects in the images by changing the position of the pixels is called as **Position augmentation**. There are many more other augmentation techniques available for images both in **position**: scaling, cropping, padding, rotation, translation and in **color**: brightness, saturation, contrast, hue. The augmentations that modify the color properties of an image by changing the pixel values are called as **Color augmentation**. Out of all these techniques, only the flipping approaches are chosen because most of these techniques include resizing or interpolating the input images and the idea is to avoid such techniques as they distort or modify the shape and dimensions of the fishes in the images. No color augmentation is performed in this work.

Chapter 6

Implementation Methodology

6.1 Problem Statement

There is not enough research in the field of underwater fish segmentation which could help Underwater Observatory Systems in automating the monitoring process of fishes. In this Master's thesis work, the problem of fish segmentation is solved and it can be used in many applications like predicting the biomass of various fishes from underwater videos. State-of-the-art binary semantic segmentation models are developed in this thesis that could segment fishes in the images from the background.

6.2 Hardware and Software

The development and training of all deep networks were done using PyTorch [69] as the main machine learning framework. The training of the networks was done on computing machines, property of the Fachhochschule Kiel, using NVIDIA TITAN XP and GeForce RTX 2080 TI GPUs.

In this thesis, PyTorch a deep learning framework is used for the training and development of the segmentation networks. It is an open-source machine learning library based on the Torch library, primarily developed by Facebook's AI research lab (FAIR) [70]. The main reasons to choose this framework are, it is more pythonic, it has a massive research community and it provides fast and efficient GPU support. Torch is constantly being developed and is being used by various companies such as Facebook, Google, and Twitter. Project development is done using the high-level programming language Python (version 3.7.7), using PyCharm as an Integrated Development Environment in macOS. The code versioning and tracking are done using GitHub and the model versioning is done using Data Version Control (DVC) [1]. For all the configuration settings in this work, an open-source python framework Hydra [2] is used. The source code of the project developed in this thesis is available on GitHub [3].

[1] https://github.com/iterative/dvc
[2] https://github.com/facebookresearch/hydra
[3] https://github.com/veeramallirajesh/ufo-segmentation

6.3 Training Configurations

This section explains the settings with which the CNN networks were trained. With the help of these settings, one could reproduce the results described in chapter 8.

6.3.1 Train-Validation-Test Split

As discussed earlier, 400 images are annotated with segmentation masks. Out of these images 80% of the samples were chosen for training, 20% of the samples were chosen for validation and the remaining 20% were chosen for testing the model. With this 80-20-20 split total number of images in the training, validation, and test splits are:

- Train: 320 image samples
- Validation: 40 image samples and
- Test: 40 image samples

6.3.2 Hyperparameters

The process of training a CNN or any deep neural network involves a lot of parameters to be set up and adjusted during the training process. Some of the parameters and their corresponding settings used in this work are described in the section.

- Input shape $(BatchSize \times Channels \times Height \times Width)$: $(1 \times 1 \times 512 \times 512)$
- Loss function: Dice loss
- Batch-size: 4
- Optimizer: Adam
- Learning rate: $1e-3$
- Number of epochs: 30
- Early stopping: Yes, with patience 7.
- Output activation: Sigmoid

6.3.3 Loss Functions

Deep Learning algorithms are trained to map a set of inputs to a set of outputs from the training data. To obtain a good mapping, the algorithm needs to find optimal parameters of the network by optimizing some objective function. The objective function is usually termed as the loss function that measures the error between the prediction and the ground-truth observation. It must be ensured that the mathematical representation loss function, is able to cover all the possible edge cases to learn

the objective (mapping between inputs and outputs) accurately and faster.

"The introduction of loss functions has roots in traditional machine learning, where these loss functions were derived on basis of the distribution of labels. For example, Binary Cross Entropy is derived from Bernoulli distribution and Categorical Cross Entropy from Multinoulli distribution [71]."

A widely used loss function for semantic segmentation tasks is pixel-wise cross-entropy loss. This loss function calculates the error for each pixel vector individually, comparing the class predictions to a one-hot encoded target vector. The computed loss is then averaged over all the pixels. In a binary segmentation task, this loss function is defined as binary cross-entropy where each prediction is compared against 2 classes of pixels.

Binary Cross Entropy:

Cross-entropy is defined as a measure of the difference between two probability distributions, a true distribution, and an estimated distribution. It outputs a probability value between 0 and 1.. It is also named as *logarithmic loss* or *logistic loss*. It is a popular loss function and widely used for classification objectives. It works well in segmentation as it is a pixel-level classification. Binary Cross-Entropy loss is defined as:

$$L_{BCE}(y, \widehat{y}) = -(ylog(\widehat{y}) + (1-y)log(1-\widehat{y})) \tag{6.1}$$

In Eq. 6.1, y is the true value and $\widehat{y}$ is the predicted value from the prediction model. The binary cross-entropy is very helpful to train a model if each classification can be reduced to a binary choice (i.e. 0 or 1, yes or no, background or foreground, fish or no fish, etc.).

Dice Loss

Dice loss, which is originated from the Sørensen-Dice coefficient is a widely used metric by computer vision researchers to calculate the similarity between two images. The loss is calculated at pixel level and it is defined as:

$$DL(y, \widehat{p}) = 1 - \frac{2y\widehat{p} + 1}{y + \widehat{p} + 1} \tag{6.2}$$

In Eq. 6.2, both numerator and denominator are added with 1 to prevent the function from being undefined (to avoid division by zero problem) in edge case scenarios such as when $y = \widehat{p} = 0$.

6.3.4 Batch Size

This parameter represents a number of training samples from the training set that will be used during the training in order to make one update to the network parameters. Specifically, the batch size is used when fitting the model, and it controls the number of predictions to be made at a time. It impacts the CNN training both in terms of the time to converge and the amount of over-fitting.

6.3.5 Optimizer

Optimization in deep learning involves minimizing an objective function termed a loss function. Gradient descent is an optimization algorithm that finds the local minimum of an objective function in an iterative fashion. The main idea is that it takes repeated steps in the opposite direction of the computed gradient of the function at the current point to reach the steepest point in that region. In deep learning, gradients are computed at each layer and are propagated backward to update the parameters, which is called backpropagation. The gradient descent is defined as,

$$\begin{aligned} L(\theta) &= -\frac{1}{N}\sum_{i} N f(\theta_T, x_i), \\ \theta_(t+1) &= \theta_t - \eta \delta L(\theta) \end{aligned} \tag{6.3}$$

where $L(\theta)$ is the loss function over the parameters θ, N is the size of the batch, t is the time step and η is a multiplication factor called learning rate which dictates the speed at which the function steps towards the minima. Smaller learning rates result in small changes in network weights while larger learning rates result in rapid updates. So, this hyperparameter should be carefully selected.

The selection of the algorithm to optimize a neural network's objective function is one of the most important steps in training a network. In machine learning, there are three main kinds of optimization strategies based on gradient descent. The first one is called *batch* or *deterministic* gradient methods which use the entire training set to make one update to the network parameters. It processes all training samples simultaneously in one large batch. The second one is called *stochastic* or *online* methods that use only one example at a time to make one update to the entire network parameters. Most of the algorithms today are a mix of the two methods using only a part of the training set indicated by batch size at each epoch. These algorithms are called *mini-batch* methods. In the deep learning era, mini-batch methods are mostly preferred for two important reasons. Firstly, the convergence speed is accelerated. Secondly, as the mini-batches are selected randomly and they are independent, an unbiased estimate of the expected gradient can be computed.

Stochastic gradient descent (SGD) maintains a constant learning rate (termed alpha) for backpropagating the error gradient during the entire training process. Sometimes, it is necessary to change the learning rate depending on how close or far we move from the global minima. Adaptive Moment Estimation (Adam) [72] is a type of optimization algorithm that computes adaptive learning rates for individual network parameters. Adam combines the advantages of two other extensions of SGD, Specifically:

- **Adaptive Gradient Algorithm** In this method, the algorithm performs smaller updates for the parameter associated with the most frequent features and larger updates for the parameters associated with the most infrequent features. Thus, the algorithm adapts the learning rate per parameter eliminating the need to control the learning rate manually. This method is highly efficient for problems with sparse gradients (e.g. computer vision).
- **Root Mean Square Propagation** (RMSProp) [73] also adapts per-parameter learning rates by dividing the gradient by running average of recent magnitudes.

Adam acquires the strengths and benefits of both AdaGrad and RMSProp. Adam additionally makes use of the average of the second moments of the gradients along with the average of the first moment (the mean) as in RMSProp for computing the learning rate for individual parameters. Specifically, the algorithm updates the exponential moving averages of the gradient (m_t) and the squared gradient (v_t), and the hyperparameters β_1 and β_2 control the exponential decay rates of these moving averages.[72]

$$m_t = \beta_1 m_(t-1) + (1-\beta_1) g_t \tag{6.4}$$

$$v_t = \beta_2 v_(t-1) + (1-\beta_2) g_t^2 \tag{6.5}$$

m_t and v_t are estimates of the first and second moments, specifically the mean and the uncentered variance of the gradients respectively.[72]

6.3.6 Number of Epochs:

One epoch refers to one complete pass of the entire training data (in batches) in the forward direction and the propagation of error gradients in the backward direction. In general, it is not enough to pass the entire dataset only once through the network, we need to pass it multiple times and update the parameters of the network iteratively. Depending on the number of epochs the network may fall under *underfitting* or *optimal* or *overfitting* scenario.

Underfitting is the situation where the trained network has poor performance both on the training set and on a completely new test set. This scenario is observed when the network is trained with only one epoch.

Overfitting is the situation where the trained network has good performance on the training set but it has poor generalization ability on the new data.

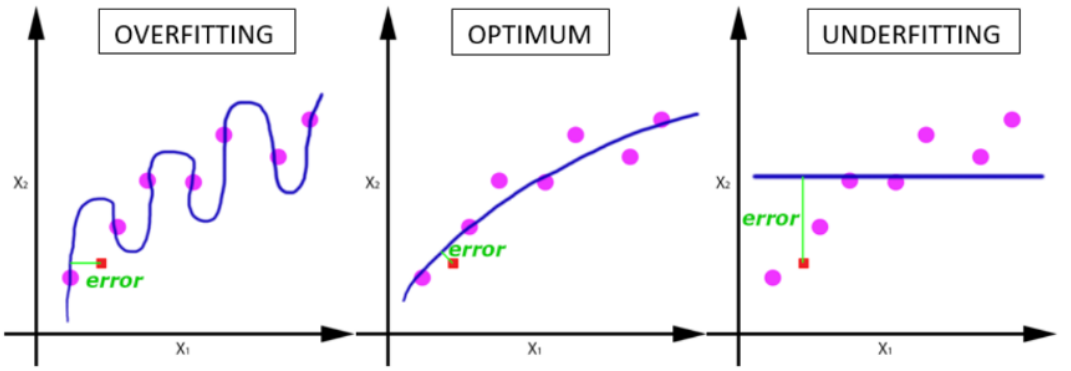

Figure 6.1: Graph depicting the effect of epochs on the network training. Source: [74]

So, it is important to select the optimum number of epochs to train a deep neural network to learn the features of the data effectively. This number can be fine-tuned depending on the network, dataset, and other hyperparameters chosen.

6.3.7 Early Stopping:

This is one of the methods to prevent the model from being overfitting. To do so, fine-tuning is performed by continuously monitoring the validation loss after each epoch. This process of terminating the training of the algorithm when the loss on the validation set starts to increase or there is no improvement while the loss on the training set still continuously going down is called Early Stopping. This hyperparameter can be set with the patience value which is the number of consecutive epochs after which the training should be terminated if there is no decrease in the validation loss. The patience value is set to 7 in this work.

6.4 Training Pipeline

The pipeline to train the deep networks in this thesis study is a 6 steps approach as in figure 6.2. In the first step, the input grayscale images of size (1024×1280) and their corresponding binary masks of the same size are fetched from the dataset. In the second step, the images are masks are pre-processed to the target shape of (512×512). The pre-processing of the inputs is done using the bounding box labels as described in 5.6.1. Then in the next step, the data augmentations are applied to

the input batch of data. The data augmentation is only applied at the runtime by applying random modifications to the original images without actually increasing the training set. This will make sure that in each training iteration, the model sees a different batch of images. The data augmentations are explained in chapter 5.

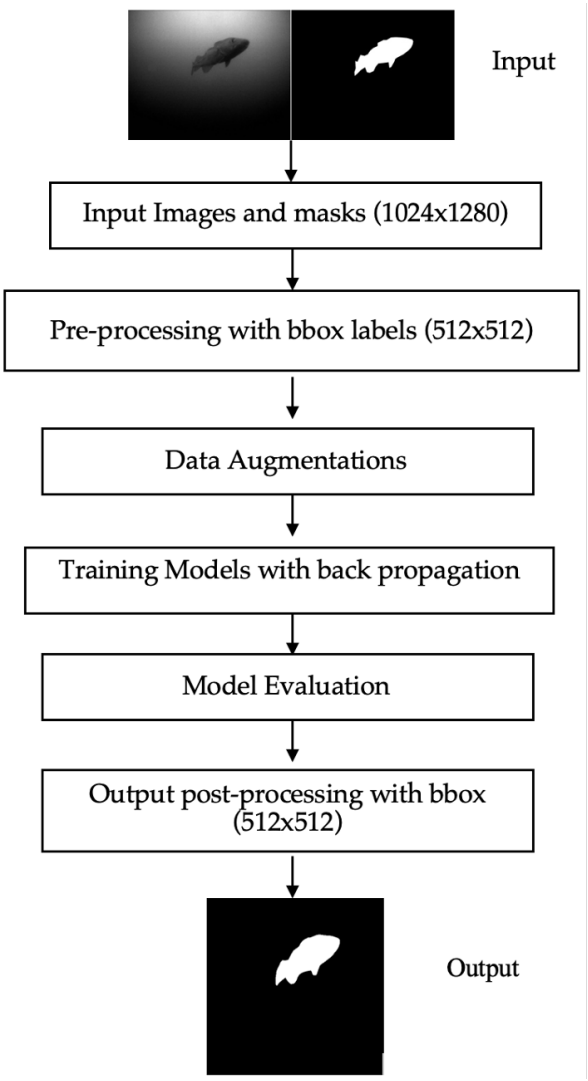

Figure 6.2: Training Pipeline for segmentation networks.

In the next step, the models that are chosen for the task of semantic segmentation are trained with the training data. The networks are trained in epochs and in each training epoch, the network weights are fine-tuned with back-propagation. The trained models are evaluated on the test set for measuring the efficiency of the model on the unknown data. Our task is a binary semantic segmentation task, so we have a sigmoid output from the model in the range of 0 to 1. The outputs after thresholding and post-processed with the bounding box labels to remove any false positives outside the region of interest.

6.5 Transfer Learning

Training a model from scratch is never a good idea unless we have a very huge labeled dataset for the specific task. Since there was a limited labeled dataset in our case, it was sensible to use the "Transfer Learning" approach and make use of the existing pre-trained models.

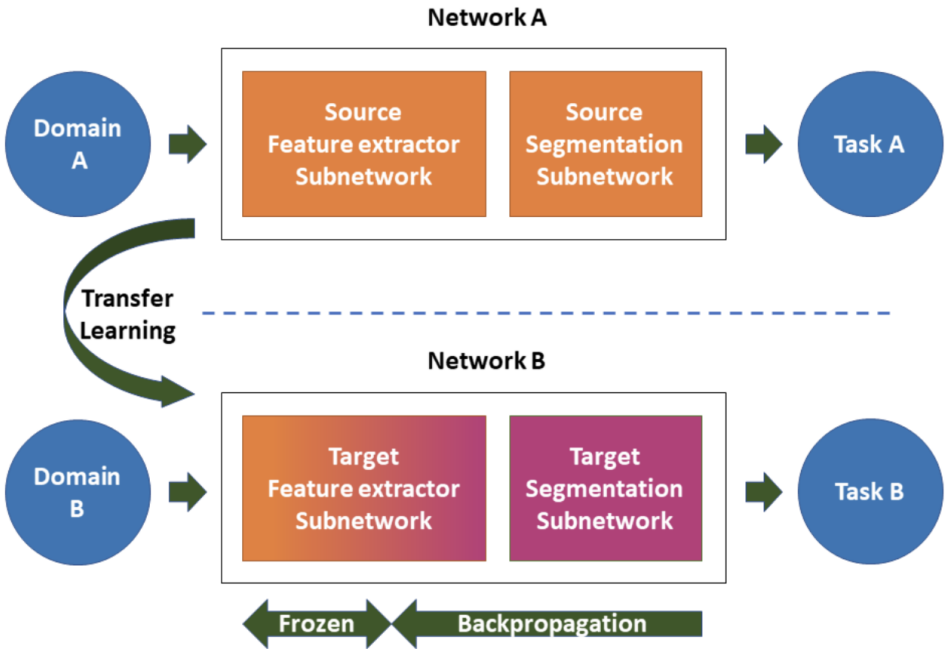

Figure 6.3: Transfer Learning of model trained on Domain A to Domain B. Source: [75]

As shown in the figure.6.3, Network A is trained on the dataset in domain A for the purpose of solving task A. The trained network A is fine-tuned on the dataset in domain B using backpropagation, by keeping the initial layers in a frozen state to solve task B. In this thesis work, the feature extractor trained on the ImageNet [76] dataset, is fine-tuned on the fish dataset. A transfer learning approach called fine-tuning is used in this thesis. The lower layers of a CNN extract information about general features of an image namely edges, basic shapes, color, etc. The information extracted by the higher layers of a CNN is more dataset-specific. So, the weights learned from the original trained model are kept in a frozen state, to exploit the features learned by them. Since the general features in the ImageNet dataset and fish dataset are very similar to each other, the same weights can be used. Only a few higher layers must be modified and trained on the fish dataset, the hyper-parameters of these layers are optimized during training.

6.6 Encoder-Decoder Architecture

Most of the segmentation networks follow encoder-decoder architecture. The goal of an image segmentation task is to generate a prediction map of a size equivalent to the

size of the input image. Hence, it is very important to retain the spatial information of the input image. Therefore, no fully connected layers are used in the network and these networks are called as **Fully Convolutional Networks (FCN)**. The sequence of convolutional layers and downsampling (pooling) layers produces a low-resolution image tensor containing high-level information. By taking this low-resolution spatial tensor with high-level information in it, we have to produce high-resolution segmentation outputs. To achieve this we add a sequence of more convolutional layers followed by upsampling layers (refer section 4.1.2) which gradually increase the size of the spatial features. During the decoding path, as we increase the resolution of the image, we decrease the number of channels in the convolution filter as we are getting back to the low-level information.This is called an **encoder-decoder architecture**.

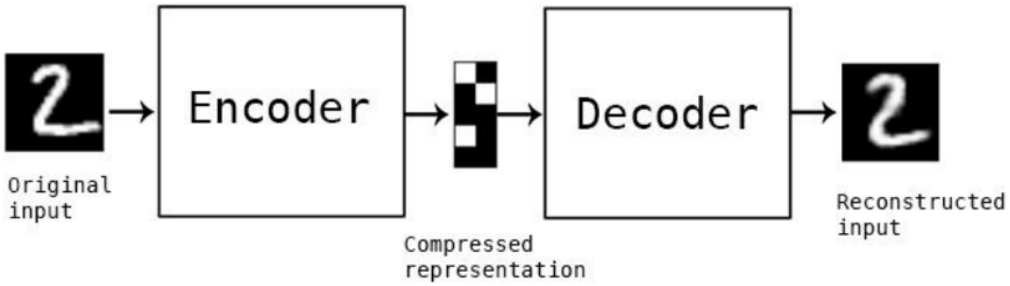

Figure 6.4: An encoder takes the image as input and output a compressed representation while decoder take the compressed representation as an input and reconstruct the input image. Source: [77].

As shown in figure 6.4 the layers that correspond to downsampling the input image makes up the encoder and the layers that correspond to upsampling makes up the decoder. All the segmentation networks trained in the thesis follow encoder-decoder architecture.

6.7 Skip Connections

Skip connections are a special type of connections in a neural network that carries information between 2 non-consecutive layers. By simply stacking the encoder and decoder layers, there is a high risk of losing the low-level details and fine-grained information. This might result in inaccurate boundaries in segmentation maps produced by the decoder. Skip connections solve this problem by letting the decoder to access the low-level features produced by the encoder layers. With the help of these connections, intermediate outputs of the encoder are concatenated with the inputs to the intermediate layers of the decoder at the same level.

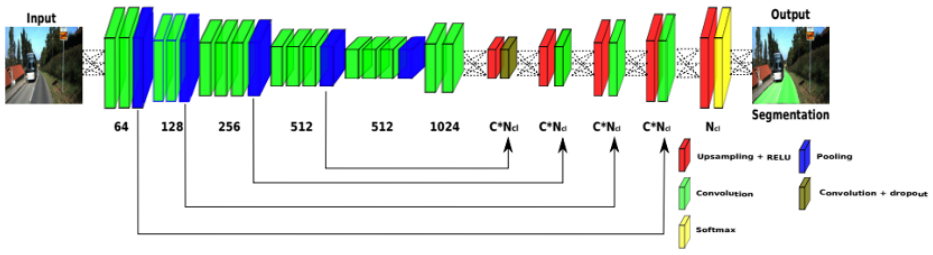

Figure 6.5: Encoder-Decoder architecture with skip connections to resolution refinement. Source: [77].

As in figure 6.5, the skip connections from the earlier layers of the encoder are connected to the appropriate layers in the decoder to pass the necessary low-level information which is required for creating accurate object boundaries in the segmentation output.

6.8 Model Architectures

As explained in previous sections, many different network architectures were proposed in recent years for solving the problem of semantic segmentation within the scope of deep learning. Some of the most relevant ones like PSPNet, U-Net, and DeepLabv3 are used in this master thesis for underwater fish segmentation. This section describes the architecture of the models used in this thesis in detail.

6.8.1 MobileNetV2

MobileNetV2 [78] was introduced in the year 2017 by a group of researchers from Google. It is an optimized network for mobile devices and it is used as the backbone architecture for all the segmentation networks used in this work. It acts as an encoder part of the network and extracts all the necessary lower-level features from the images and passes them through the decoder part of the network. The state-of-the-art performance and benchmarks of mobile models on multiple tasks can be effectively improved by the MobileNetV2 network.

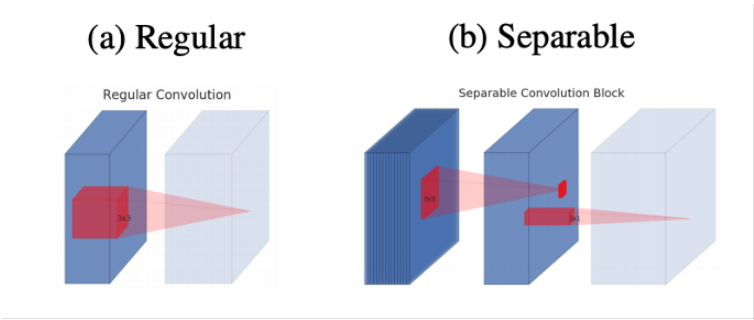

Figure 6.6: Regular vs Depthwise Separable Convolutions in MobileNetV2. Source: [78]

Depthwise Separable Convolutions and inverted residuals are key building blocks of the MobileNetV2 architecture. The basic idea of depthwise convolutions is to reduce the number of computations by replacing a full convolutional operator with a combination of depthwise and pointwise convolutions. This operation is further explained in section 4.1.4. The authors of the paper claim that depthwise convolutions reduce the computation cost by a factor of k^2 compared to traditional layers. MobileNetV2 uses $k = 3(3 \times 3$ depthwise separable convolutions) so the computational cost is *8* to *9* times smaller than that of standard convolutions at only a small reduction in the accuracy.

The beginning and end of the convolution blocks are connected with the residual blocks (using skip connections). With the help of these connections, the network can access activations from the earlier layers. The original residual blocks follow a wide-narrow-wide approach concerning the depth of input and in MobileNetV2 it is reversed as narrow-wide-narrow. So the authors named it inverted residual blocks.

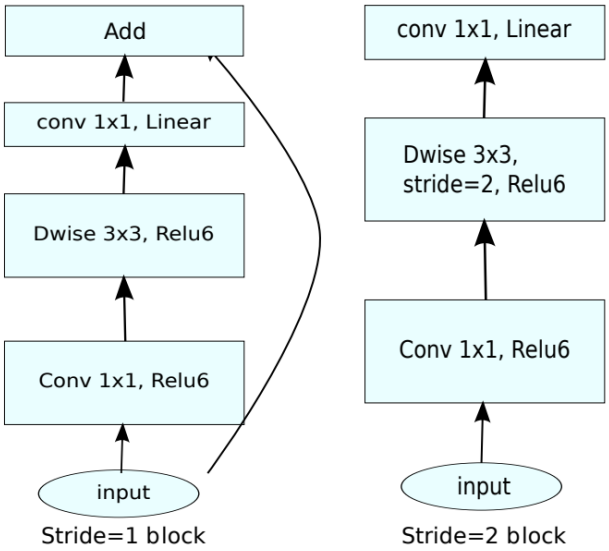

Figure 6.7: MobileNetV2 architecture with depthwise convolutions and skip connections. Source: [78].

The original MobilenetV2 is trained on the ImageNet dataset with the following settings: The developers of this network trained the model using TensorFlow. They used the standard RMSProp Optimizer with both decay and momentum set to 0.9. The network also has batch normalization after every layer, and the standard weight decay is set to 0.00004. Following MobileNetV1 setup they use an initial learning rate of 0.045, and a learning rate decay rate of 0.98 per epoch. They used 16 asynchronous GPU workers to train the network with a batch size of 96.

6.8.2 PSPNet: Pyramid Scene Parsing Network

PSPNet network is another well-recognized semantic segmentation network that follows encoder-decoder architecture. This paper was published in CVPR 2017. This network provides a superior framework for the task of scene understanding and generating pixel-level predictions. The authors of the paper claim that their proposed architecture yields a new record of mIoU of 85.4% on PASCAL VOC 2012. The overview of the proposed PSPNet architecture is shown in the figure 6.8

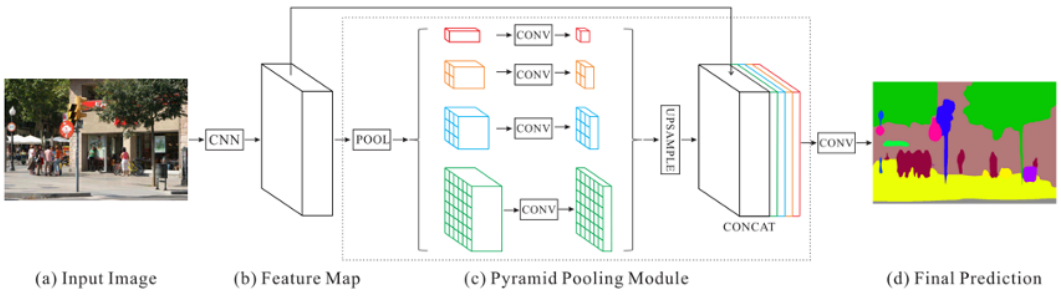

Figure 6.8: Overview of the proposed **PSPNet** model. Source: [18].

Given an input in figure 6.8 (a), the developers used the pre-trained ResNet model with the dilated network strategy to extract the feature maps. The feature maps from the encoder are of size $1/8^{th}$ of the input image as shown in figure 6.8 (b). In this work, pre-trained MobileNetV2 is used as an encoder to extract the feature maps instead of ResNet. (b). On the extracted feature maps the pyramid pooling module (discussed in the next section) is processed as shown in the figure 6.8 (c) to gather contextual information. The feature maps from the pyramid pooling module are integrated and concatenated with the original feature map in the final part of figure 6.8 (c). It is followed by a decoder network to generate the final prediction map as in the figure 6.8 (d).

Pyramid pooling module:

The pyramid pool module can collect information at different levels, more representative than the global pooling module. It is the heart of the PSPNet architecture as it helps the model to capture the global context in the image which is used to classify the individual pixels. In PSPNet a four-level pyramid is used to obtain global context from the image. The feature map from the pre-trained backbone 6.8.1 is pooled at four different sizes (to capture multi-scale information) and passed through a 1×1 convolution layer. The pooled feature maps are integrated and upsampling (refer section 4.1.2) layer is operated to make the resolution of the pooled features equivalent to the original feature map. Finally, the upsampled feature maps are concatenated with the original feature map to be passed to the decoder part of the network to generate final predictions. This technique of fusing the features at different scales

provides the aggregation of the overall context.

For example, in the figure 6.8 (c) we can notice a four-level pyramid pooling module. The four colors green, blue, orange, and red represent scale sizes of 6, 3, 2, and 1 respectively. The feature map from the CNN is pooled at these scales after which they are convolved with 1x1 filters to reduce the depth of the features. The feature maps with varied sizes are integrated and upsampled at the size of the original feature map. Finally, the original feature map is connected with the upsampled feature with a skip connection. The features pooled by 1×1 filter are coarse (low resolution) in nature as it captures information from 1×1 spatial locations of the input. As the pyramid pool size increases, the resolution of the feature maps increases as well (as a case of 6×6 pyramid size).

PSPNet Decoder:

The PSPNet with a pyramid pooling module does not complete the architecture of a segmentation model. It is just an encoder, which makes up the first half of the image segmentation network. To obtain the final predictions of input resolution, the feature maps must be passed through a decoder. The decoder is responsible for converting the feature maps received from the encoder into a final prediction map. The decoder achieves this by passing the feature maps from its layers. The decoder is yet another network that takes in low-resolution features and results in predictions of desired resolution.

A convolutional layer followed by a bilinear upsampling layer with a scale of 8 is one of the most common decoders used in various implementations of PSPNet. It is used in the current work as well. As discussed in the section 4.1.2 the upsampling layers do not have learnable parameters in them which makes the final segmentation results look blobby with some sense of details missing. This could also result in the network generating smoother boundaries of the objects in the image.

6.8.3 U-Net:

The U-Net is another state-of-the-art semantic segmentation model that follows an encoder-decoder architecture which was developed by Ronnenberger et. al in 2015 [16]. U-Net, evolved from the traditional convolutional neural network, was first designed and applied in 2015 to process biomedical images. This network is investigated for the task of fish segmentation in this work.

The U-Net architecture contains two main paths the contraction path followed by the expansion path. The contraction path (also called encoder) i.e the path on the left side of the figure 6.9 is responsible to capture the context (*what* information) in the image by reducing the spatial resolution of the image. The expansion path (also

called decoder) i.e. the path on the right side of the figure 6.9 is the symmetric path to the encoder which is responsible to bring back the spatial resolution to enable accurate localization (*where* information) using transposed convolutions (refer section 4.1.2). As the U-Net contains only convolution layers without any dense layers both in the encoder and the decoder part, it is an end-to-end fully convolutional network (FCN). Due to this reason, it can accept images of any size. Actually, the network is named U-Net because the encoder and the decoder are symmetric and the architecture looks similar to the alphabet letter "U".

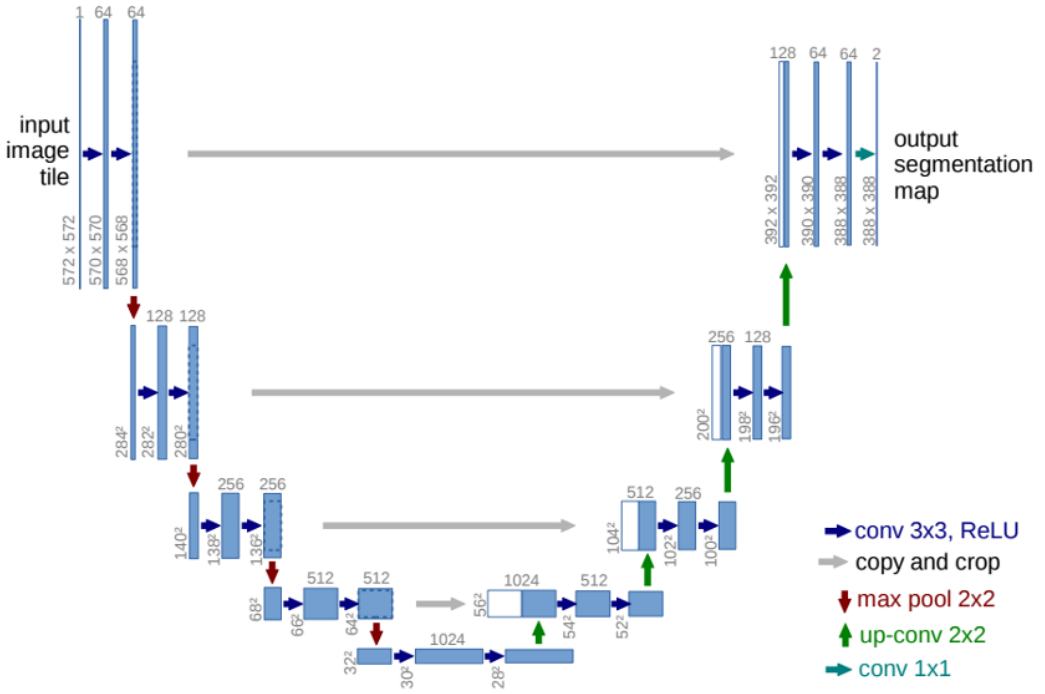

Figure 6.9: Overview of the proposed U-Net model. Source: [16].

U-Net also has skip connections at several points along the decoding path that connect the feature maps from the same stage at the encoder. These shortcut connections are also helpful to skip some layers when backpropagating the error during training to avoid problems like vanishing gradients. The connected feature maps are concatenated with the upsampled feature maps at the decoding layers. Furthermore, each convolution or transposed convolution along with the encoder and the decoder is followed by a rectified linear unit (ReLU) activation function 4.2.

As in most classification models (VGG, ResNet, etc.), the encoder in U-Net has layers alternating between convolutions and max-pooling operations. The input size of the model is changed from (572×572) in the original paper to (512×512) in this master thesis. As mentioned in section 6.8.1 MobileNetV2 acts as an encoder for the U-Net in this work.

6.8.4 DeepLabv3

DeepLabv3 is the successor of DeepLab and DeepLabv2 [79] and it is published in the paper called **"Rethinking Atrous Convolution for Semantic Image Segmentation"**[19]. This model also follows encoder-decoder architecture. The key points about this model architecture are:

- Atrous Convolution
- Going Deeper with Atrous Convolution Using Multi-Grid
- Atrous Spatial Pyramid Pooling (ASPP)

The overview of the DeepLabv3 architecture showing all the key elements of the network is shown in figure 6.10:

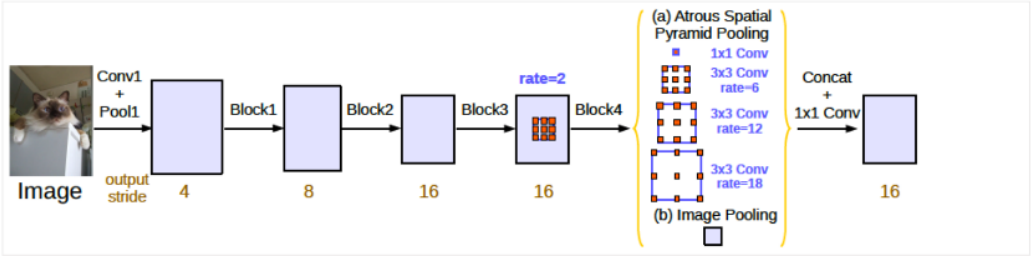

Figure 6.10: Overview of the proposed **Deep Lab V3** model. Source: [19]

Deep Convolutional Neural Networks (DCNNs) designed and trained in a fully convolutional fashion have shown to be very effective for the task of semantic segmentation. However, there is a significant reduction in the spatial resolution of the output feature maps when a combination of max-pooling and striding layers are applied repeatedly in the network. Transposed convolution (refer section 4.1.2) has been employed to recover the spatial resolution but at the cost of increasing the number of parameters and computation time. Instead, this network advocates the use of 'atrous convolution'. Atrous Convolutions (refer section 4.1.3) are the key building blocks of the DeepLabv3 model architecture. They are applied at varying dilation rates at various depths of the architecture.

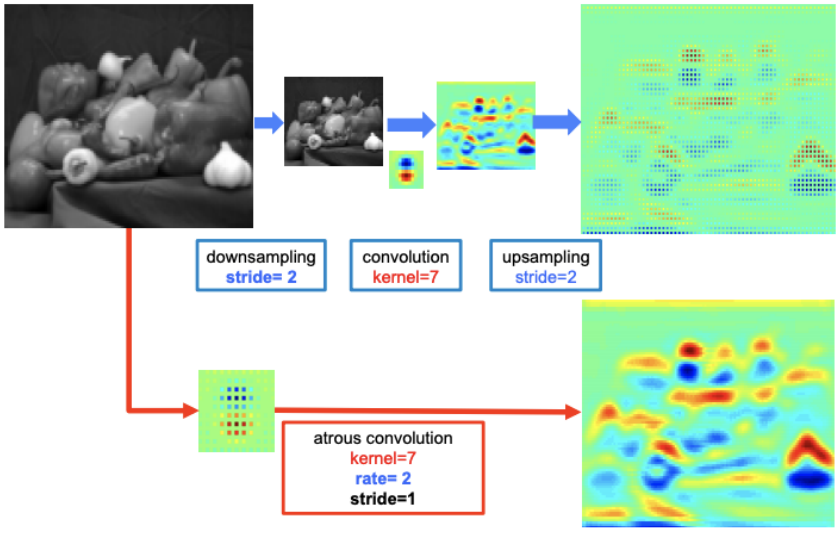

Figure 6.11: Illustration of atrous convolution in 2-D. The upper part of the image denotes sparse feature extraction with standard convolution on a low resolution input feature map. The bottom part of the image denotes dense feature extraction with atrous convolution with rate r = 2, applied on a high resolution input feature map. Source: [79]

Going Deeper with Atrous Convolution Using Multi-Grid

In DeepLabv3 architecture, a series of 3×3 atrous convolutions are designed in cascade. These sequential modules capture long-range information from the inputs. In the original paper, the last block of the ResNet encoder is duplicated 3 times and represented as block 4 in figure 6.10. After duplication of the block, the total number of blocks in the network are 7. Similarly in the current work the last blocks of the MobileNetV2 encoder are duplicated.

(a) Without Atrous Conv: When a sequence of standard convolution and pooling is performed on the input image, it makes the output stride increasing while gradually decreasing the output feature map. However, the application of strides consecutively is unfavorable for semantic segmentation since spatial information is decimated at the deeper layers. Moreover, this approach increases the number of parameters in the network.

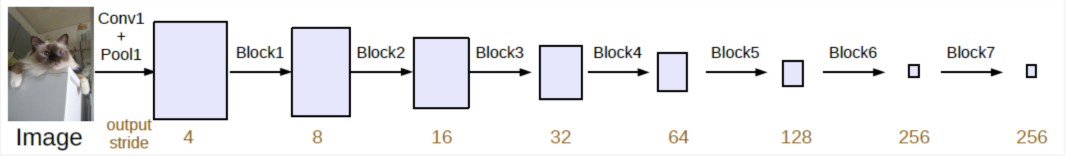

Figure 6.12: Cascaded modules without atrous convolutions. Source:[19]

(b) With Atrous Conv: To address the increasing computation time problem, atrous convolutions are used. With atrous convolution, one can incorporate a larger field-of-view by keeping stride size and number of parameters in the network constant. Finally, We can achieve a larger output feature map which is good for semantic segmentation without affecting the computation time or number of parameters.

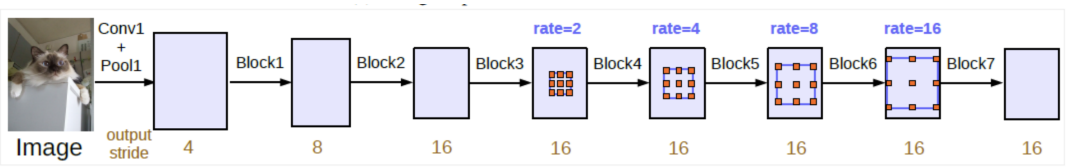

Figure 6.13: Cascaded modules with atrous convolutions. Atrous convolution with dilation rate *rate>1* is applied after block3. Source:[19]

Atrous Spatial Pyramid Pooling (ASPP)

This module was initially proposed in [79] and reused by the authors in DeepLabv3 by adding batch normalization into it. One of the important abilities of DCNNs is to detect objects of multiple sizes. By training DCNNs on datasets that contain objects of varying size they show exceptional ability to implicitly represent scale. However, the authors believed that the DCNN's ability to handle objects of varying scales could be further improved by explicitly accounting for object scale. In the mentioned paper the authors explained about two approaches to handle scale variability in semantic segmentation.

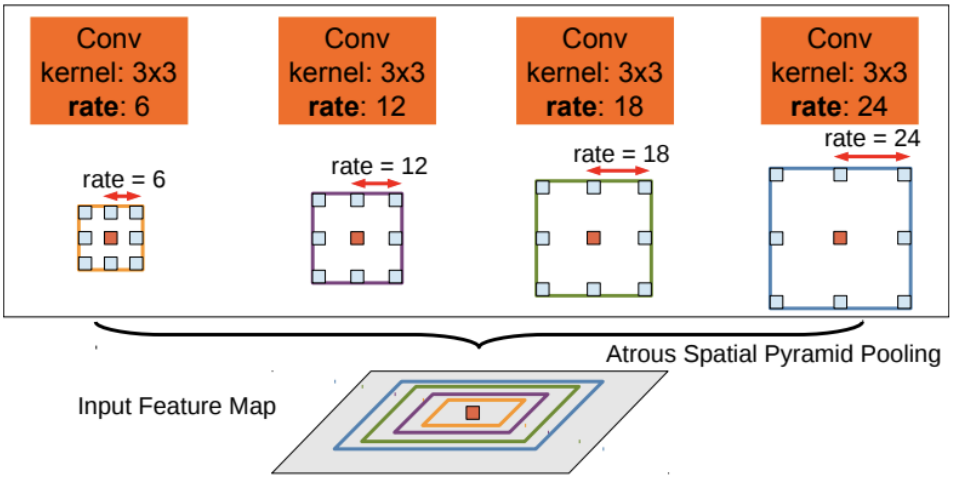

Figure 6.14: Atrous Spatial Pyramid Pooling (ASPP). To classify the center pixel (orange), ASPP exploits multi-scale features by employing multiple parallel filters with different rates. The effective Field-Of-Views are indicated in various colors. Source: [79]

The first approach was standard multiscale processing [80] [81]. The approach was to extract feature maps at multiple scales of the original image using parallel

DCNN branches that share the same weights. The feature maps from all the DCNN branches were bilinearly interpolated to the original image resolution. Finally, to generate the result the outputs were fused, by taking the maximum response at each position across the different scales. This processing was done both during the training and testing phases. Multi-scale processing significantly improved the performance of the DCNN but at the cost of growing computation time.

The second approach from them was inspired by the success of the R-CNN spatial pyramid pooling method of [82], which eliminated the existing CNNs constraint of fixed size input image by resampling convolutional features extracted at a single scale. With this idea objects or regions of arbitrary size can be accurately classified. They have modified this scheme and implemented a new variant that uses multiple parallel atrous convolutional layers with different dilation rates. The features extracted for each dilation rate are further processed in separate parallel branches and integrated to generate the final result. The proposed "atrous spatial pyramid pooling" (DeepLabASPP) approach generalizes their DeepLab-LargeFOV variant and is illustrated in the figure 6.14.

Chapter 7

Evaluation Metrics

Metrics are the tools used to analyze the quality of any statistical or machine learning model. One has to wisely choose the metrics to evaluate the model based on the input distribution and the aim of the task. As already mentioned in section 6.3.1, I have evaluated the final and intermediate results on a relatively small validation and test set. It is, therefore, appropriate to interpret them carefully. The efficiency of semantic segmentation networks cannot be described in a comprehensive and accurate way by a single evaluation metric. Each evaluation metric has its own pros and cons, so the combination of two or more evaluation metrics must be used to get a precise idea about the network's efficiency. The evaluation metrics used in this thesis are explained in this section.

7.1 Confusion Matrix

Image segmentation is all about how accurately the model has classified each pixel correctly. A confusion matrix explains how many times the classifier predicted correctly and how many times it was wrong. The horizontal axis indicates the actual values, whereas the vertical axis indicates the predicted values. For a two-class classification problem, the confusion matrix looks like table 7.1.

	Actual Value	
Predicted Value	**Class 0**	**Class 1**
Class 0	True Negative	False Negative
Class 1	False Positive	True Positive

Table 7.1: Confusion Matrix

True Positive (TP): If the actual value is positive (class 1) and the predicted value is also positive (class1), the classification is right and falls under true positive.

False Positive (FP): If the actual value is negative (class 0) and the predicted value is positive (class 1), then the results fall under false positive.

False Negative (FN): If the actual value is positive (class 1) and the predicted value is negative (class 0), then the result goes into false negative.

True Negative (TN): If the actual value is negative (class 0) and the predicted value is also negative (class 0), the classification is right and falls under true negative.

For example, the confusion matrix for gender classification looks like in Table 7.2.

	Truly Male	Truly Female
Classifier calls it male	196	4
Classifier calls it female	0	0

Table 7.2: Example for Confusion Matrix

The confusion matrix in Table 7.2 says that there are 200 samples in which all 200 are classified as male but in that only 196 samples are truly male samples and 4 samples are truly female. Therefore, only 4 images are wrongly classified by the classifier. If we just calculate the accuracy of the classifier it is 98% accurate but it does not make any sense because the classifier is giving output only as a male. Thus, the confusion matrix helps to get insights into a classifier irrespective of class-imbalance, but it can be applied only on supervised models as the labels are already known. It cannot be applied to the model with unknown labels.

7.2 Class Imbalance

In a binary semantic segmentation problem class imbalance is referred to one class of pixels dominating the other class in an input image. If this problem is consistent over all the images in the dataset, then the model is expected to suffer from a class imbalance issue. To deal with this issue one has to choose the right metrics and the loss function to train a deep learning model.

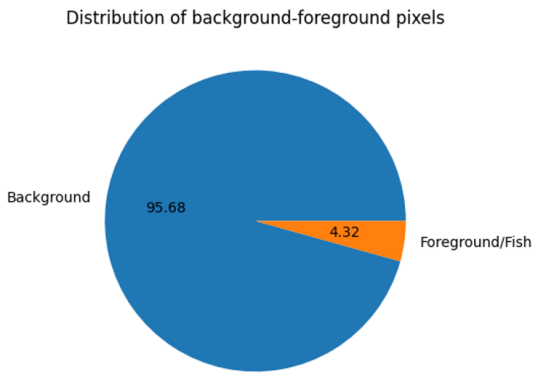

Figure 7.1: Plot showing average distribution of pixels-per-class in the training set.

Figure 7.1 is a pie chart describing the average distribution of background and foreground pixels of the images in the training set. From the plot, one could notice that background pixels are dominating the fish pixels, the pixels that we are interested in. There exists a high class imbalance problem here and one has to carefully deal with this to properly train the models.

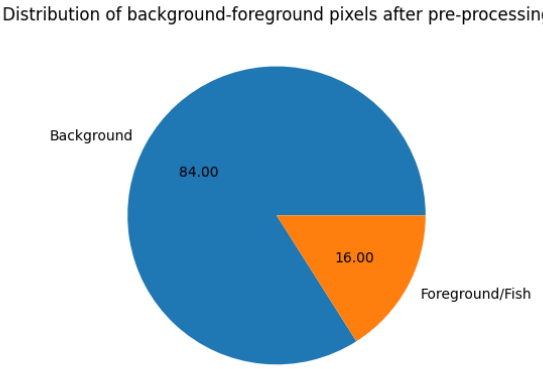

Figure 7.2: Plot showing average distribution of pixels-per-class in the training set after pre-processing.

Figure 7.2 is a pie chart describing the average distribution of background and foreground pixels of the images in the training set after pre-processing (as described in section 5.6.1). From the plot, one could notice that cropping the images with bounding box labels has improved the ratio between background and foreground from 4.32% to 16%. However, the problem of class imbalance still persists.

7.3 Pixel Level Accuracy

This is the most common evaluation metric for a semantic segmentation task. The overall per-pixel accuracy measures the ratio of correctly classified pixels to total pixels. This metric makes use of a pixel-level confusion matrix for evaluating the formula defined in 7.1. The main disadvantage of this metric is that it's strongly influenced by a class imbalance in the dataset (refer to section 7.2). A classifier constantly betting on the majority class can achieve high-accuracy rates. Here, the "background" or "no fish" class covers approximately 16% (from figure 7.2) of an entire image. So in this case, a high PA score from the model does not necessarily mean that the model is equally confident about all the classes.

$$\textbf{Pixel Accuracy(PA)} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

In this case, for a given class C:

- **True Positive (TP):** indicates pixel classified correctly as C

- **False Positive (FP):** indicates pixel classified incorrectly as C
- **True Negative (TN):** indicates pixel classified correctly as not C
- **False Negative (FN):** indicates pixel classified incorrectly as not C

PA is not the appropriate choice of metric when there is a high class-imbalance (as shown in figure 7.1) in the input dataset but can act as a supportive metric with other metrics that can potentially deal with this issue.

7.4 Intersection-Over-Union (IoU)

This metric is often referred to as Jaccard Index and it is the most commonly used metric in semantic segmentation and object detection problems. The great advantage of this metric is that both the false positives and the false negatives are taken into consideration. Nevertheless, this metric has a drawback. It does not explicitly measure how precise the predicted segmentation boundaries are.

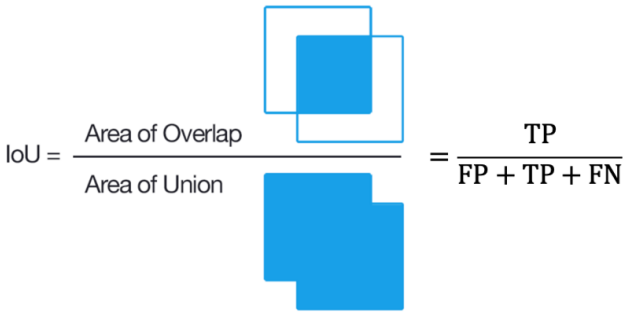

Figure 7.3: Visual representation of Intersection over Union. Source: [83].

where, TP, FP, and FN denote the true positive, false positive, and false negative counts, respectively.

It is measured as the area of overlap between the predicted segmentation and the ground-truth segmentation, divided by the area of union between the predicted segmentation and the ground truth as shown in the figure 7.4. This metric ranges from 0-1 (0-100%) where an IoU of 0 denotes no overlap and an IoU of 1 denotes a perfect overlapping segmentation.

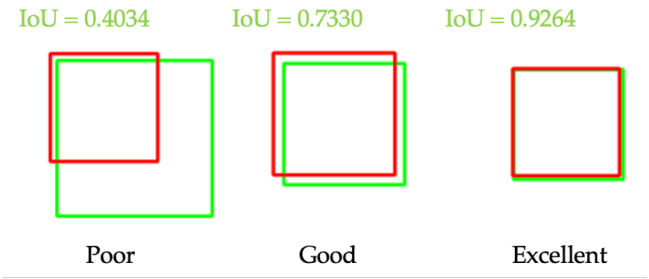

Figure 7.4: Examples of how IoU is calculated. Left: poor performance= 0.4034, Middle: Good performance = 0.7330, Right: Excellent performance = 0.9264. Source: [83].

7.5 Precision Recall Curves

Precision recall curves are used to evaluate the results of binary classification problems. As the segmentation itself is a classification problem at the pixel level, it is a good metric to evaluate the performance of the models. Moreover, Precision-Recall curves act as a good metric when there is a moderate to large class imbalance issue in the input dataset. The precision is plotted on the y-axis and the recall on the x-axis at varying probability thresholds.

Precision: Precision refers to number of positive predictions which are actually correct. It is calculated by the ratio of the number of true positives divided by the sum of the true positives and false positives as expressed in Eq. 7.2. The curve shows how good the model is in predicting a positive class. The other name for Precision is **Specificity**.

$$\textbf{Precision} = \frac{TP}{TP + FP} \tag{7.2}$$

Recall: Recall is the ratio of the number of true positives by the sum of the true positives and false negatives as expressed in Eq. 7.3. It indicates the proportion of actual positives identified by the model. The other name for Recall is **Sensitivity**.

$$\textbf{Recall} = \frac{TP}{TP + FN} \tag{7.3}$$

To understand more about true-positives, true-negatives and false-negatives please refer to section 7.1.PR curve majorly concentrates on the prediction of the minority class. Since true negatives are not involved in any calculations, PR curves should be used only when specificity is not a concern for the classifier. The ratio of a total number of positive results divided by the sum of a total number of positive results and negative results is called the 'no-skill line'. There are composite scores that are used to review precision and recall they are:

- **F score or F_1 score:** this calculates the harmonic mean of the precision and recall. This measure is used to find the optimal threshold that results in the best balance of precision and recall.

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{7.4}$$

- **Area Under Curve:** this summarizes the integral of the area under the precision-recall curve.

All these methods mentioned above can be used on machine learning models to calculate their efficiency.

Chapter 8

Experimental Results

In this chapter the performance of all the 3 segmentation models investigated in this thesis are discussed in detail, to give a complete overview to the reader. Various performance metrics are used to evaluate and understand the performance of the segmentation models. To know more about the evaluation metrics used in this thesis work, please refer to chapter 7.

8.1 Results Comparison Table

As discussed in chapter 6 the models PSPNet, U-Net and DeepLabv3 are compared and tabulated in Table 8.1. The performance metrics Pixel accuracy, IoU, Precision, Recall, and F_1 Score are used for comparison. The metrics PA (Pixel Accuracy) and IoU are corresponding to each model on the validation corpus of the dataset. Precision, Recall, and F1 score are measured on the test corpus of the dataset and tabulated values are at a threshold of 0.6. All the models are trained with the hyperparameter settings discussed in section 6.3.2.

	Metric (in percentage)				
Model	Pixel Accuracy	IoU	Precision	Recall	$F_1 Score$
PSPNet	97.8%	88.7%	96.2%	92.2%	94.40%
U-Net	98.6%	91.0%	97.6%	91.1%	94.32%
DeepLabv3	**98.6**%	**91.3**%	**96.0**%	**93.1**%	**94.66**%

Table 8.1: Table to compare the metrics of all the three models on validation corpus.

From table 8.1 one can infer the performance metrics of all the networks with the same training configurations explained in chapter 6. The metrics pixel-accuracy (PA) and IoU are tabulated at 30^{th} epoch. It can be noticed that the DeepLabv3 model has the best metrics compared to the other two models. However, all the models have a very small difference in their metrics. To compare the accuracy of all the 3 models at various thresholds, precision-recall curve gives a better understanding of the same. Figure 8.1 is the precision-recall curve plotted for all the 3 models at varying probability thresholds between 0.0 to 1.0. The following plot shows the Precision-Recall curve of all the networks on the validation. data:

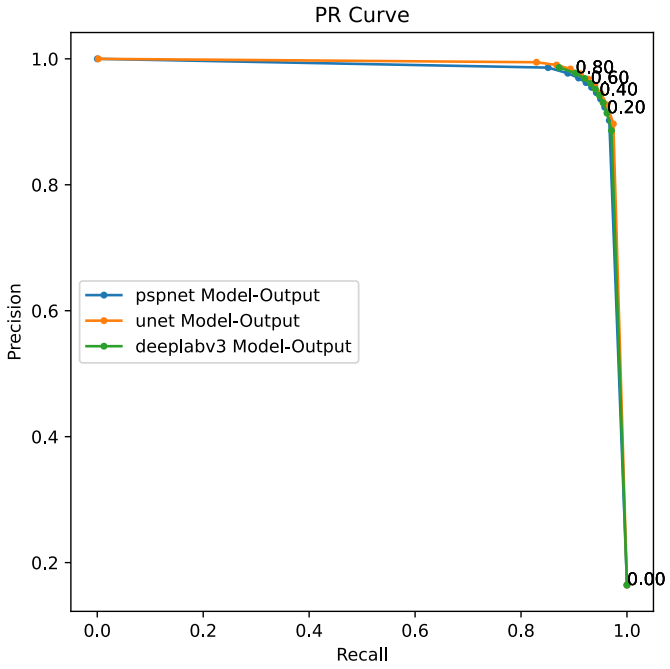

Figure 8.1: Precision-Recall curve for the three models at varying probability thresholds.

The plot shown in figure 8.1 describes the precision, recall values of all the 3 models. From the graph, we could notice that the area under the PR-Curve for all the models is almost same. The F_1-score is used to determine the optimal probability threshold for all the models. At the probability threshold of 0.60, it is observed that there is a good balance between precision and recall. Therefore, 0.60 is considered as the optimal threshold for post-processing of the segmentation output. F1-score values at 0.6 threshold are formulated in table 8.1.

8.2 Loss and Metrics curves

This section presents the loss and metrics curves for all the models. The graphs are plotted for 30 epochs using TensorBoard. Smoothing is applied to the curves, to give a general idea of relatively slow changes in the metrics and for better visualization. The curve in light shade represents the true values of the metrics and the curve in the darker shade represents the metrics after smoothing.

8.2.1 PSPNet accuracy, loss and IoU curves

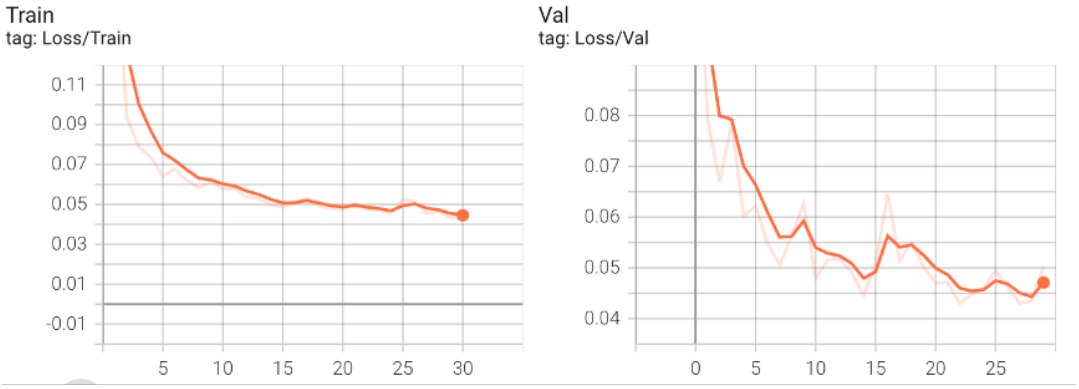

Figure 8.2: PSPNet Training/Validation Loss Curves

The plots shown in 8.2 are the loss curves for the PSPNet model. The loss curves are plotted for both training and validation set. Both training loss and validation loss fall below 0.05 by the end of 30^{th} epoch.

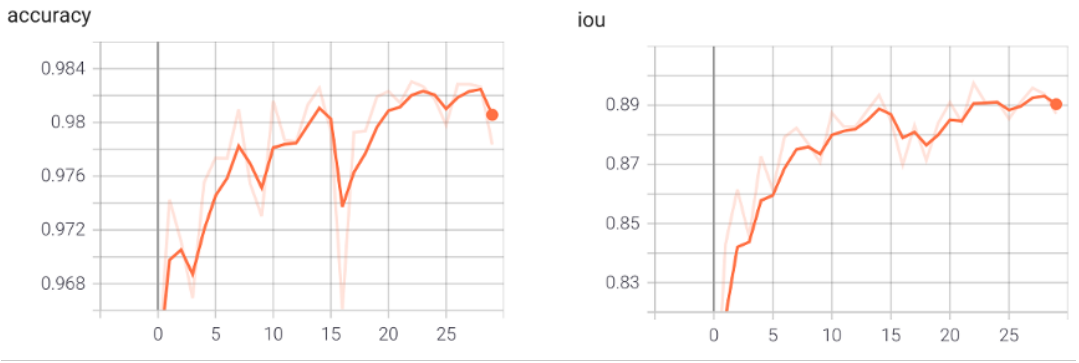

Figure 8.3: PSPNet validation metrics curves

The plots shown in 8.3 are the accuracy and IoU metrics curves for the PSPNet model. The metric curves are plotted for the validation set. PSPNet has a pixel-accuracy of 97.8% and IoU of 88.7% by the end of 30^{th} epoch.

8.2.2 U-Net accuracy, loss and IoU curves

The plots shown in 8.4 are the loss curves for the U-Net model. The loss curves are plotted for both training and validation set. Training loss settled around 0.04 and the validation also settles around 0.045 by the end of 30^{th} epoch.

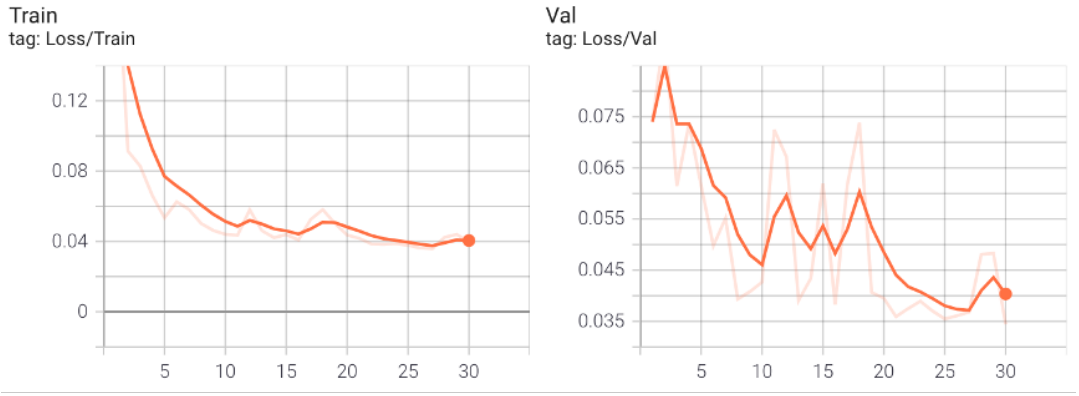

Figure 8.4: U-Net Training/Validation Loss Curves

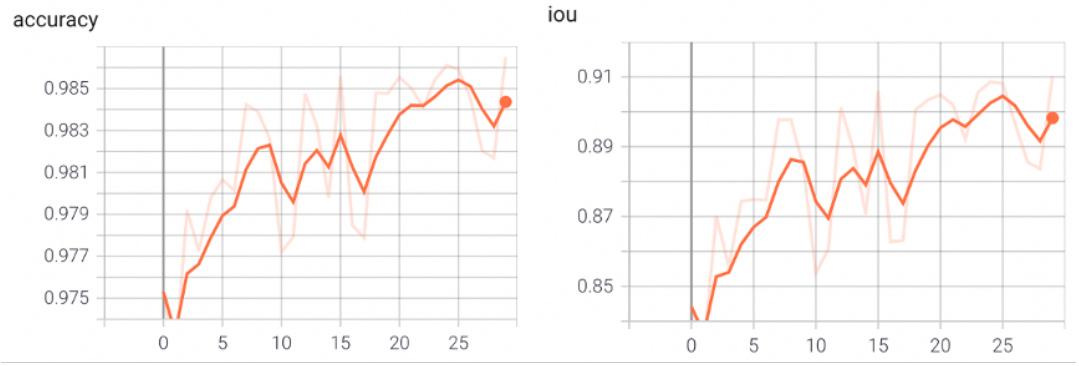

Figure 8.5: U-Net validation metrics curves

The plots shown in 8.5 are the accuracy and IoU metrics curves for the U-Net model. The metric curves are plotted for the validation set. U-Net has a pixel-accuracy of 98.6% and IoU of 91.0% by the end of 30^{th} epoch.

8.2.3 DeepLabv3 accuracy, loss and IoU curves

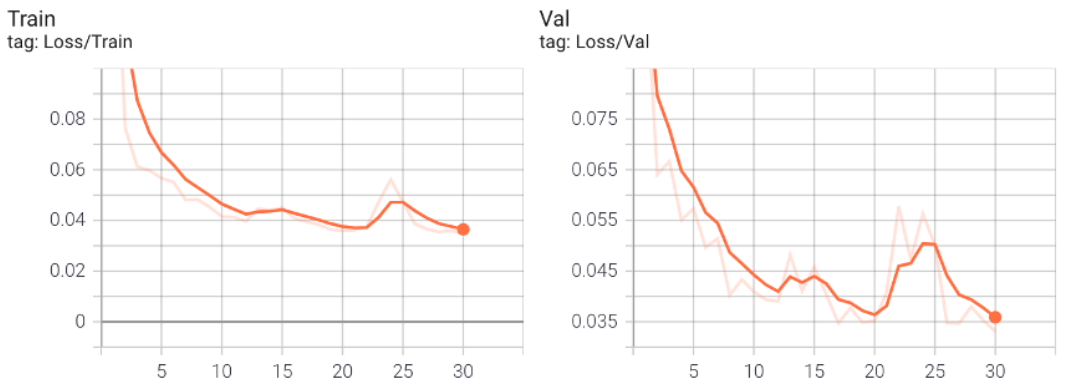

Figure 8.6: DeepLabv3 Training/Validation Loss Curves

The plots shown in 8.6 are the loss curves for the DeepLabv3 model. The loss curves are plotted for both training and validation set. Both training loss and validation loss is less than 0.05 by the end of 30^{th} epoch.

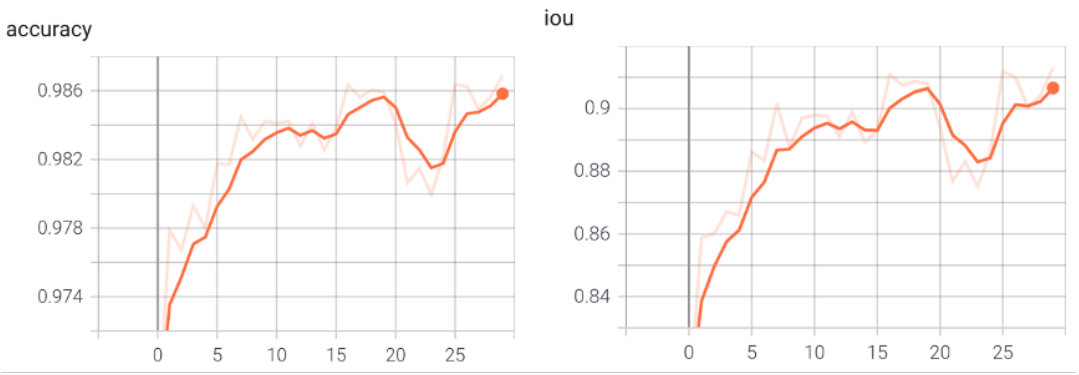

Figure 8.7: DeepLabv3 validation metrics curves

The plots shown in 8.7 are the accuracy and IoU metrics curves for the DeepLabv3 model. The metric curves are plotted for the validation set. DeepLabv3 has a pixel-accuracy of 98.6% and IoU of 91.3% by the end of 30^{th} epoch.

8.3 Visual Assessment of Segmentation versus Ground-truth

In this section the outputs from the model are visualized and compared with the ground-truth mask. The IoU score is a well established quantitative measure of segmentation quality. However, a visual assessment should be conducted to get an idea of what types of fishes the networks struggled to segment. Visualizing the predicted segmentation masks of all the fishes is impractical. Each of the visualized output has 4 images and they are: original image, original mask (ground truth mask), predicted mask and the difference between original and the predicted mask(referred to as "Difference" in the plots). In the "***Difference plot***" of the output visualizations, the color **"yellow"** represents all the pixels that are correctly classified by the model and the color **"purple"** represents all the pixels of the image that are incorrectly classified. Below are some of the sample images and the outputs from all the models are visualized. For a better understanding of the results, 2 samples are explained with the help of a confusion-matrix.

8.3.1 PSPNet Output Visualisations

	Actual Value			
	Sample 1		**Sample 2**	
Predicted Value	**No Fish**	**Fish**	**No Fish**	**Fish**
No Fish	231263	593	180289	1354
Fish	1561	28727	3306	77195

Table 8.2: Confusion Matrix for samples in figure 8.8 for PSPNet Model

In table 8.2 the confusion matrix for the two samples (same samples shown in output visualizations in figure 8.8) is tabulated for the PSPNet model. As samples

are of size 512 × 512, the total number of pixels is 262144. So, the confusion matrix gives an overview of the model performance for all the pixel values. In figure 8.8 the outputs from the PSPNet model are visualized. The two sample images of size 512 × 512 are chosen from the test corpus.

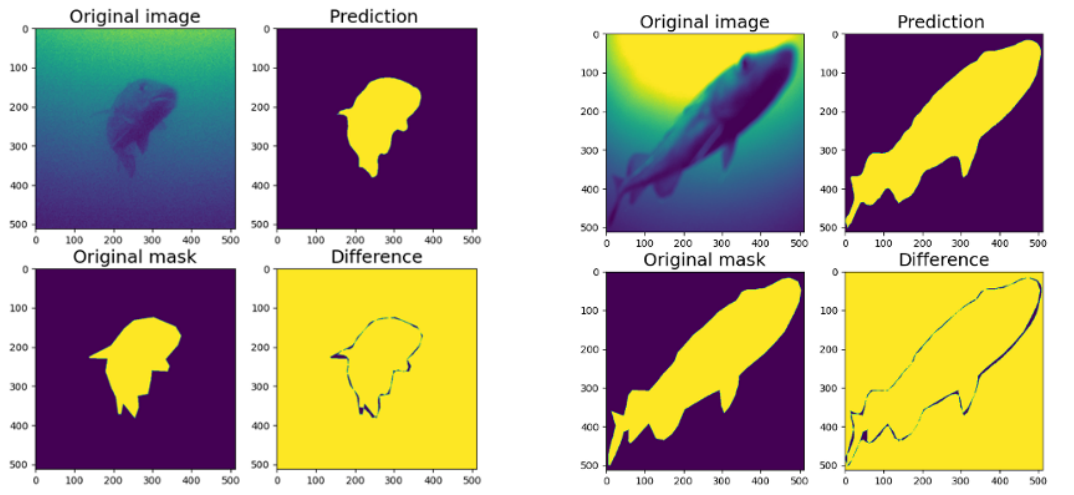

Figure 8.8: PSPNet output visualizations: Sample 1 on left and Sample 2 on right.

8.3.2 U-Net Output Visualisations

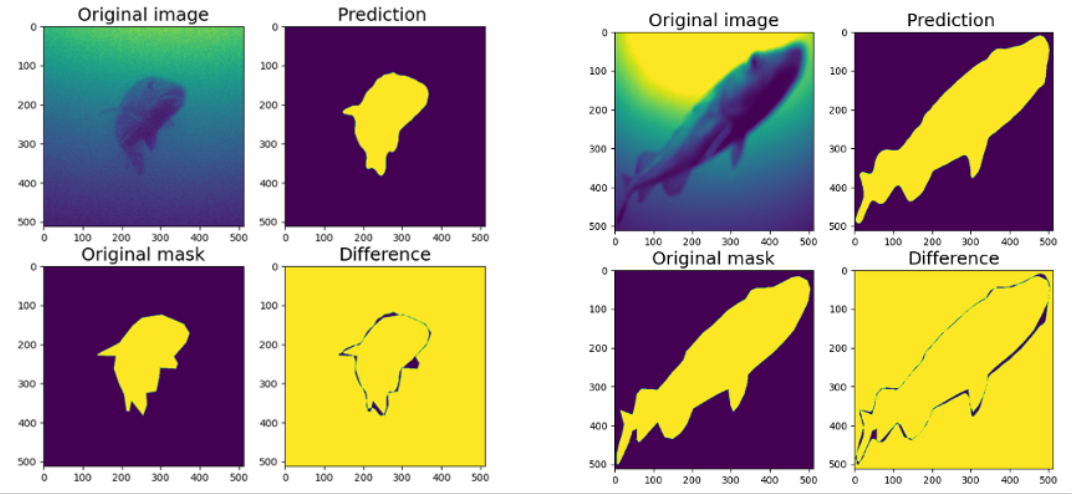

Figure 8.9: U-Net output visualizations: Sample 1 on left and Sample 2 on right.

In figure 8.9 the outputs from the U-Net model are visualized. The two sample images of size 512 × 512 are chosen from the test corpus. In table 8.3 the confusion matrix for the same two samples is tabulated. As samples are of size 512 × 512, the total number of pixels is 262144. So, the confusion matrix gives an overview of the model performance for all the pixel values.

	Actual Value			
	Sample 1		Sample 2	
Predicted Value	**No Fish**	**Fish**	**No Fish**	**Fish**
No Fish	231432	1646	179010	878
Fish	1392	27674	4585	77671

Table 8.3: Confusion Matrix for samples in figure 8.9 for U-Net Model

8.3.3 DeepLabv3 Output Visualisations

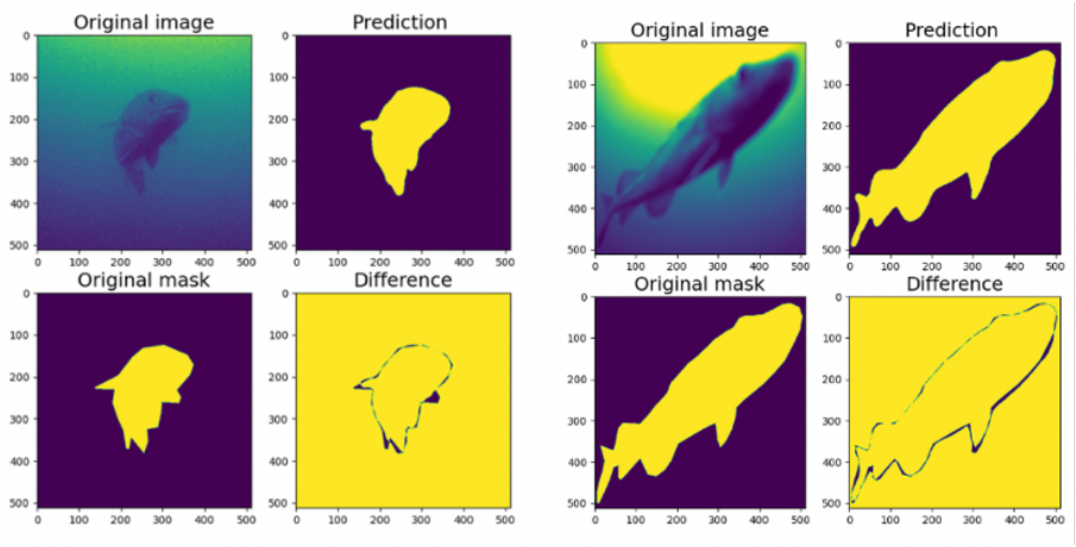

Figure 8.10: DeepLabv3 output visualizations: Sample 1 on left and Sample 2 on right.

In figure 8.10 the outputs from the DeepLabv3 model are visualized. The two sample images of shape 512×512 from the test corpus are chosen for output visualization. In table 8.4 the confusion matrix for the same two samples is given. The total number of pixels is 262144. Confusion matrix gives an overview of the model performance for all the pixel values.

	Actual Value			
	Sample 1		Sample 2	
Predicted Value	**No Fish**	**Fish**	**No Fish**	**Fish**
No Fish	231990	1828	179450	1391
Fish	834	27492	4145	77158

Table 8.4: Confusion Matrix for samples in figure 8.10 for DeepLabv3 Model

8.4 Error Analysis

In this section, a thorough investigation is carried out, to understand in what cases the model is producing more errors, either false-positives or false-negatives?. All three models are making most of the errors at the object boundaries. From a careful investigation, it is understood that the ground-truth segmentation masks have sharper edges and the trained deep learning models are generating smoother boundary edges. The possible assumptions for the models generating smoother boundary edges are:

- PSPNet decoder has an $8\times$ bilinear upsampling layer at the end of the network which retains the input resolution. The authors explained that the results are expected to be blobby with the sense of details missing. Similarly, DeepLabv3 also uses a bilinear upsampling layer after the ASPP layer (refer 6.14) which is also the possible reason for smoother boundary edges in the segmentation.
- The chosen threshold value for post-processing the segmentation output. It is assumed that the trained networks are less confident in predicting the pixels accurately at the fish boundaries. All the pixels that are less than the threshold value segmented to be the background.

However, further investigation is required to understand this effect in detail. In general the ground-truth segmentation masks are expected to be more smoother, rather being sharp edges. So, by properly improving the ground-truth annotations with more smoother labels, the network errors at the boundaries could be possibly mitigated.

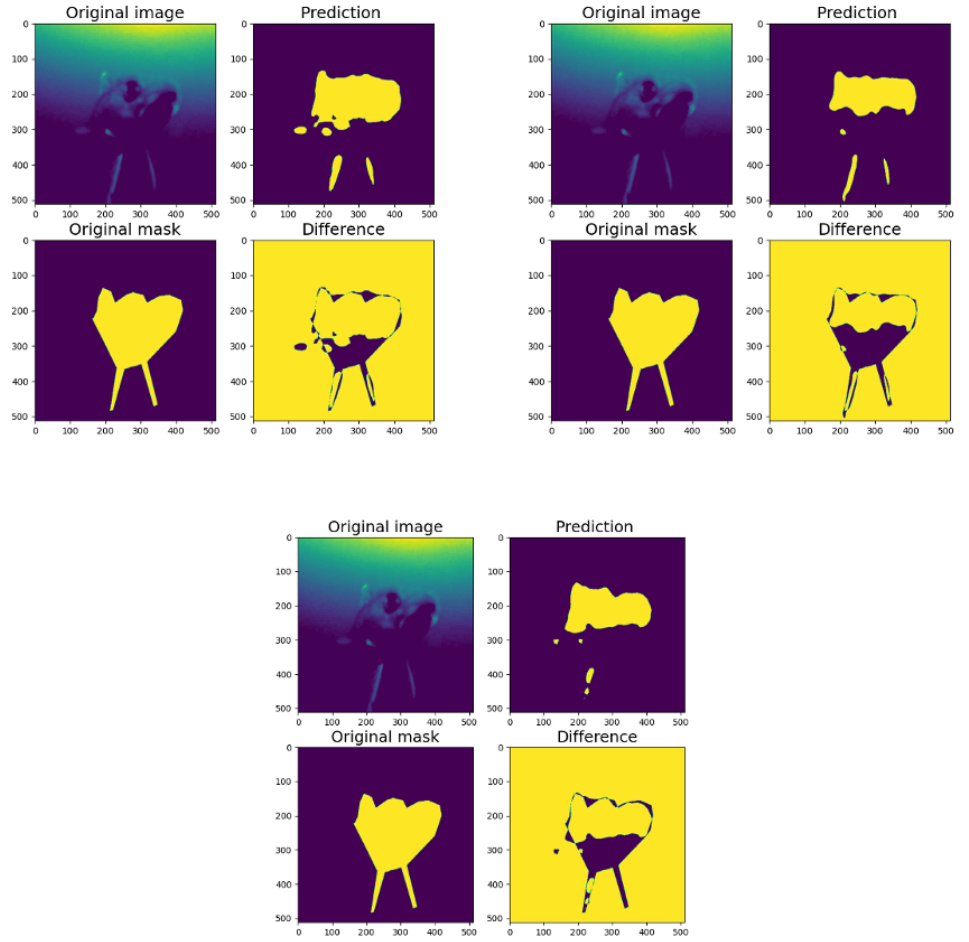

Figure 8.11: Output visualizations of single sample from all the models.

In the results shown in figure 8.11, some parts of the fish are hardly visible and all the networks produced errors in those regions. With proper data collection in good lighting conditions and with quality annotations these errors could be reduced.

Chapter 9

Conclusion and Future work

Successful methods were developed for segmenting fishes in underwater videos in this thesis work. For training deep networks, 400 image samples were annotated with the help of the CVAT annotation tool. Data augmentation was used to generate new samples from the existing samples during training. As the size of the labeled dataset is small, transfer learning was used to solve the segmentation problem. MobileNetV2 trained on the ImageNet dataset was used as the baseline feature extractor for all the segmentation networks in this work. The networks PSPNet, U-Net, and the DeepLabv3 with the light-weight MobileNetV2 encoder were trained and had produced state-of-the-art results on the validation data. DeepLabv3 is the best among all these networks on this fish segmentation task with a F1 score of 94.66% and PSPNet with the second-best figures of 94.40% F1 score. U-Net also successfully solved the task with an F1 score of 94.32%. All the segmentation networks struggled to correctly classify the pixels in the boundary regions of the fish. Also, in some samples where the fish is directly facing the camera, the visibility of the fish is very low and the number of such samples in the training set is less. The errors for this cases potentially could be reduced by inducing more similar examples in the training set. These models have shown promising results when tested on a completely new batch of data.

The potential future work is to extend this framework to multi-class segmentation of the fishes and other aquatic organisms. As the team is already working on producing more quality annotations (for instance adding smoother fish boundary edges, occluded fishes, etc.), one could retrain the networks on the new batch of data to see if the performance still improves. As the labeled data in this work is very less and it is a time-consuming task to create a larger labeled dataset, it is a very good idea to explore more semi-supervised learning algorithms that could learn from a mixture of labeled and unlabeled images. One could also iteratively improve the training data with semi-supervised learning methods and active learning.

Bibliography

[1] O. Hegazy, O. Soliman, and M. A. Salam. "A Machine Learning Model for Stock Market Prediction". In: *ArXiv* abs/1402.7351 (2014).

[2] Joseph A Cruz and David S Wishart. "Applications of machine learning in cancer prediction and prognosis". In: *Cancer informatics* 2 (2006), p. 117693510600200030.

[3] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* 37.3 (2020), pp. 362–386.

[4] R Krishna. *Computer Vision Foundations and Applications*. 2017.

[5] Arthur Ouaknine. "Review of deep learning algorithms for object detection". In: *Medium. February* 5 (2018), p. 2018.

[6] Bi-ke Chen, Chen Gong, and Jian Yang. "Importance-Aware Semantic Segmentation for Autonomous Driving System." In: *IJCAI*. 2017, pp. 1504–1510.

[7] Khalil Khan, Massimo Mauro, and Riccardo Leonardi. "Multi-class semantic segmentation of faces". In: *2015 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2015, pp. 827–831.

[8] Alexander Vezhnevets, Vittorio Ferrari, and Joachim M Buhmann. "Weakly supervised semantic segmentation with a multi-image model". In: *2011 international conference on computer vision*. IEEE. 2011, pp. 643–650.

[9] Serena Yeung Fei-Fei Li Justin Johnson. "Detection and Segmentation". In: (2017), pp. 18–43. URL: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf.

[10] Kaggle. *Carvana Image Masking Challenge*. URL: https://www.kaggle.com/c/carvana-image-masking-challenge/data.

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[12] Thorsten Joachims. "Svmlight: Support vector machine". In: *SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund* 19.4 (1999).

[13] Mahesh Pal. "Random forest classifier for remote sensing classification". In: *International journal of remote sensing* 26.1 (2005), pp. 217–222.

[14] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. "An efficient k-means clustering algorithm". In: (1997).

[15] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: (2015). arXiv: 1505.04597 [cs.CV].

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[18] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. *Pyramid Scene Parsing Network*. 2017. arXiv: 1612.01105 [cs.CV].

[19] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. *Rethinking Atrous Convolution for Semantic Image Segmentation*. 2017. arXiv: 1706.05587 [cs.CV].

[20] King-Sun Fu and JK Mui. "A survey on image segmentation". In: *Pattern recognition* 13.1 (1981), pp. 3–16.

[21] Mehmet Sezgin and Bülent Sankur. "Survey over image thresholding techniques and quantitative performance evaluation". In: *Journal of Electronic imaging* 13.1 (2004), pp. 146–165.

[22] Marina Mueller, Karl Segl, and Hermann Kaufmann. "Edge-and region-based segmentation technique for the extraction of large, man-made objects in high-resolution satellite imagery". In: *Pattern recognition* 37.8 (2004), pp. 1619–1628.

[23] David. Marr; Ellen. Hildreth. "Theory of edge detection". In: *Proceedings of the Royal Society of London. Series B, Biological Sciences* 207 (1167 1980), 187–217. DOI: 10.1098/rspb.1980.0020.

[24] J. Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[25] J. Castan S.; Zhao and J. Shen. "New edge detection methods based on exponential filter". In: *Pattern Recognition, Proceedings 10th International Conference* 1 (16 June,1990), pp. 709–711.

[26] R. M. Haralick, K. Shanmugam, and I. Dinstein. "Textural Features for Image Classification". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-3.6 (1973), pp. 610–621. DOI: 10.1109/TSMC.1973.4309314.

[27] R. M. Haralick. "Statistical and structural approaches to texture". In: *Proceedings of the IEEE* 67.5 (1979), pp. 786–804. DOI: 10.1109/PROC.1979.11328.

[28] Marianna Clark, Alan C. Bovik, and Wilson S. Geisler. "Texture segmentation using Gabor modulation/demodulation". In: *Pattern Recognition Letters (PRL)* 6.4 (1987), pp. 261–267. URL: http://dx.doi.org/10.1016/0167-8655(87)90086-9.

[29] M. Tuceryan. "Moment based texture segmentation". In: *PPattern Recognition Letters* 15 (1994), pp. 695–668.

[30] Harold Driver and Alfred Kroeber. "Quantitative Expression of Cultural Relationships". In: *Quantitative Expression of Cultural Relationships* (1932), pp. 211–256. URL: http://digitalassets.lib.berkeley.edu/anthpubs/ucb/text/ucp031-005.pdf.

[31] J. Rubin. "A technique for measuring like-mindedness". In: *The Journal of Abnormal and Social Psychology* 33 (4 1938), 508–516. DOI: https://doi.org/10.1037/h0055441.

[32] A.K. Jain, M.N. Murty, and P.J Flynn. ""Data Clustering: A Review"". In: *ACM Computing Surveys* 31 (3 1999), pp. 264–332.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[35] Chuang Yu, Xiang Fan, Zhuhua Hu, Xin Xia, Yaochi Zhao, Ruoqing Li, and Yong Bai. "Segmentation and measurement scheme for fish morphological features based on Mask R-CNN". In: *Information Processing in Agriculture* 7.4 (2020), pp. 523–534. ISSN: 2214-3173. DOI: https://doi.org/10.1016/j.inpa.2020.01.002. URL: https://www.sciencedirect.com/science/article/pii/S2214317319301714.

[36] Rafael Garcia, Ricard Prados, Josep Quintana, Alexander Tempelaar, Nuno Gracias, Shale Rosen, Håvard Vågstøl, and Kristoffer Løvall. "Automatic segmentation of fish using deep learning with application to fish size measurement". In: *ICES Journal of Marine Science* 77.4 (2020), pp. 1354–1366.

[37] Dr A. Baloch, Mushstaq Ali, Faqir Gul, Sadia Basir, and Ibrar Afzal. "Fish Image Segmentation Algorithm (FISA) for Improving the Performance of Image Retrieval System". In: (2017).

[38] H. Christensen Jesper, V. Mogensen Lars, and Ravn Ole. "Deep Learning based Segmentation of Fish in Noisy Forward Looking MBES Images". In: (2020). arXiv: 2006.09034 [cs.CV].

[39] Nawaf Farhan Funkur Alshdaifat, Abdullah Zawawi Talib, and Mohd Azam Osman. "Improved deep learning framework for fish segmentation in underwater videos". In: *Ecological Informatics* 59 (2020), p. 101121.

[40] G Sanchez-Torres, A Ceballos-Arroyo, and S Robles-Serrano. "Automatic measurement of fish weight and size by processing underwater hatchery images". In: *Engineering Letters* 26.4 (2018).

[41] Rajesh kumar Rai, Puran Gour, and Balvant Singh. "Underwater image segmentation using clahe enhancement and thresholding". In: *International Journal of Emerging Technology and Advanced Engineering* 2.1 (2012), pp. 118–123.

[42] Alzayat Saleh, Issam Laradji, Pau Rodriguez, Derek Nowrouzezahrai, Mostafa Rahimi Azghadi, and David Vazquez. "Weakly Supervised Underwater Fish Segmentation Using Affinity LCFCN". In: (2021).

[43] Md Jahidul Islam, Chelsey Edge, Yuyang Xiao, Peigen Luo, Muntaqim Mehtaz, Christopher Morse, Sadman Sakib Enan, and Junaed Sattar. "Semantic segmentation of underwater imagery: Dataset and benchmark". In: *arXiv preprint arXiv:2004.01241* (2020).

[44] John McCarthy. "Artificial intelligence, logic and formalizing common sense". In: *Philosophical logic and artificial intelligence*. Springer, 1989, pp. 161–190.

[45] Andreas Kaplan and Michael Haenlein. "Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence". In: *Business Horizons* 62.1 (2019), pp. 15–25.

[46] WR Howard. "Pattern recognition and machine learning". In: *Kybernetes* (2007), p. 275.

[47] Lawrence Rabiner. "Fundamentals of speech recognition". In: *Fundamentals of speech recognition* (1993).

[48] Tom M Mitchell et al. "Machine learning". In: (1997).

[49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning. nature (2015)". In: *May; 521 (7553): 436 10.1038/nature14539* (2015).

[50] Yoav Goldberg. "A primer on neural network models for natural language processing". In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 345–420.

[51] Juan Manuel Ramos Arreguin. *Automation and robotics*. Citeseer, 2008.

[52] Rene Y. Choi, Aaron S. Coyner, Jayashree Kalpathy-Cramer, Michael F. Chiang, and J. Peter Campbell. "Introduction to Machine Learning, Neural Networks, and Deep Learning". In: *Translational Vision Science Technology* 9.2 (Feb. 2020), pp. 14–14. ISSN: 2164-2591. DOI: 10.1167/tvst.9.2.14. eprint: https://arvojournals.org/arvo/content_public/journal/tvst/938366/i2164-2591-226-2-2007.pdf. URL: https://doi.org/10.1167/tvst.9.2.14.

[53] Kishan Maladkar. *6 Types of Artificial Neural Networks Currently Being Used in Machine Learning*. 2018.

[54] Shukla Lavanya. *Designing Your Neural Networks*. 2019.

[55] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[56] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. 2019. arXiv: 1803.08375 [cs.NE].

[57] Albert Bou. *Deep Learning models for semantic segmentation of mammography screenings*. 2019.

[58] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285* (2016).

[59] Chi-Feng Wang. "A Basic Introduction to Separable Convolutions". In: *Towards Data Science https://towardsdatascience. com/a-basic-introduction-to-separable-convolutions-b99ec3102728 Aug* 13 (2018), p. 15.

[60] M. Holschneider, R. Kronland-Martinet, J. Morlet, and Ph. Tchamitchian. "A Real-Time Algorithm for Signal Analysis with the Help of the Wavelet Transform". In: *Wavelets*. Ed. by Jean-Michel Combes, Alexander Grossmann, and Philippe Tchamitchian. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 286–297. ISBN: 978-3-642-75988-8.

[61] Sonish Sivarajkumar. *ReLU — Most popular Activation Function for Deep Neural Networks*. May 2019. URL: https://medium.com/@sonish.sivarajkumar/relu-most-popular-activation-function-for-deep-neural-networks-10160af37dda.

[62] Chris. *What are Max Pooling, Average Pooling, Global Max Pooling and Global Average Pooling*. 2020. URL: https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/#references.

[63] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 2018. arXiv: 1811.03378 [cs.LG].

[64] Don R Hush and Bill G Horne. "Progress in supervised neural networks". In: *IEEE signal processing magazine* 10.1 (1993), pp. 8–39.

[65] Robert B Fisher, Yun-Heh Chen-Burger, Daniela Giordano, Lynda Hardman, Fang-Pang Lin, et al. *Fish4Knowledge: collecting and analyzing massive coral reef fish video data*. Vol. 104. Springer, 2016.

[66] Alzayat Saleh, Issam H Laradji, Dmitry A Konovalov, Michael Bradley, David Vazquez, and Marcus Sheaves. "A realistic fish-habitat dataset to evaluate algorithms for underwater visual analysis". In: *Scientific Reports* 10.1 (2020), pp. 1–10.

[67] Yonatan Adiwinata, Akane Sasaoka, I Putu Agung Bayupati, and Oka Sudana. "Fish Species Recognition with Faster R-CNN Inception-v2 using QUT FISH Dataset". In: *Lontar Komputer: Jurnal Ilmiah Teknologi Informasi* 11.3 (), pp. 144–154.

[68] George Cutter, Kevin Stierhoff, and Jiaming Zeng. "Automated Detection of Rockfish in Unconstrained Underwater Videos Using Haar Cascades and a New Image Dataset: Labeled Fishes in the Wild". In: *2015 IEEE Winter Applications and Computer Vision Workshops*. 2015, pp. 57–62. DOI: 10.1109/WACVW.2015.11.

[69] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[70] Serdar Yegulalp. "Facebook brings GPU-powered machine learning to Python". In: *InfoWorld* 19 (2017).

[71] S. Jadon. "A survey of loss functions for semantic segmentation". In: *arXiv:2006.14822* (2020). eprint: arXiv:2006.14822.

[72] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[73] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8 (2012).

[74] Sagar Sharma. *Epoch vs Batch Size vs Iterations*. Sept. 2017. URL: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

[75] Manpreet Singh Minhas. *Transfer Learning for Semantic Segmentation using PyTorch DeepLab v3*. Sept. 2019. URL: https://github.com/msminhas93/DeepLabv3FineTuning.

[76] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[77] Gabriel Leivas Oliveira. "Encoder-decoder Methods for Semantic Segmentation: Efficiency and Robustness Aspects". PhD thesis. Albert-Ludwigs-Universität Freiburg, 2019.

[78] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[79] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[80] Liang-Chieh Chen, Yi Yang, Jiang Wang, Wei Xu, and Alan L Yuille. "Attention to scale: Scale-aware semantic image segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3640–3649.

[81] Iasonas Kokkinos. "Pushing the boundaries of boundary detection using deep learning". In: *arXiv preprint arXiv:1511.07386* (2015).

[82] K He, X Zhang, S Ren, et al. "Spatial pyramid pooling in deep convolution networks or visual classification". In: *ECCV*. 2014.

[83] Adrian Rosebrock. *Intersection over Union (IoU) for object detection*. Nov. 2016. URL: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection.

Chapter 10

Appendix

Code snippet for horizontal flip augmentation:

```
class RandomHorizontallyFlip(object):
    def __init__(self, p):
        self.p = p
    def __call__(self, img, mask):
        if random.random() < self.p:
            return (
                img.transpose(Image.FLIP_LEFT_RIGHT),
                mask.transpose(Image.FLIP_LEFT_RIGHT),
            )
        return img, mask
```

Code snippet for vertical flip augmentation:

```
class RandomVerticallyFlip(object):
    def __init__(self, p):
        self.p = p

    def __call__(self, img, mask):
        if random.random() < self.p:
            return (
                img.transpose(Image.FLIP_TOP_BOTTOM),
                mask.transpose(Image.FLIP_TOP_BOTTOM),
            )
        return img, mask
```

Both of these functions take single image and the corresponding mask from the training set and returns the augmented samples.

PR Tables:

PSPNet			
Threshold	Precision	Recall	$F_1 Score$
0.0	0.1644	1.00	0.2823
0.1	0.9022	0.9666	0.9333
0.2	0.9232	0.9580	0.9403
0.3	0.9362	0.9502	0.9431
0.4	0.9463	0.9421	0.9442
0.5	0.9547	0.9332	0.9438
0.6	0.9633	0.9256	0.9440
0.7	0.9695	0.9086	0.9381
0.8	0.9771	0.8883	0.9306
0.9	0.9860	0.8511	0.9136
1.0	1.0000	0.0002	0.0004

Table 10.1: Precision-Recall metrics for PSPNet

UNet			
Threshold	Precision	Recall	$F_1 Score$
0.0	0.1644	1.0000	0.2823
0.1	0.8967	0.9743	0.9339
0.2	0.9269	0.9624	0.9443
0.3	0.9447	0.9511	0.9479
0.4	0.9576	0.9394	0.9484
0.5	0.9678	0.9268	0.9468
0.6	0.9765	0.9119	0.9431
0.7	0.9839	0.8931	0.9363
0.8	0.9902	0.8671	0.9245
0.9	0.9946	0.8292	0.9044
1.0	1.0000	0.0020	0.0039

Table 10.2: Precision-Recall metrics for U-Net

DeepLabv3			
Threshold	Precision	Recall	$F_1 Score$
0.0	0.1644	1.0000	0.2823
0.1	0.8862	0.9707	0.9265
0.2	0.9138	0.9622	0.9374
0.3	0.9301	0.9550	0.9424
0.4	0.9421	0.9478	0.9449
0.5	0.9520	0.9403	0.9461
0.6	0.9607	0.9330	0.9466
0.7	0.9689	0.9201	0.9439
0.8	0.9772	0.9028	0.9385
0.9	0.9863	0.8718	0.9256
1.0	NaN	0.000	NaN

Table 10.3: Precision-Recall metrics for DeepLabv3

Declaration of Authorship

I hereby declare that I have prepared the present work independently and without the use of resources other than those indicated. Thoughts that are taken literally or meaningfully from other works are indicated with the text. I hereby certify that I have not submitted any examination papers with the same or similar subject to an examination authority or other university.

Kiel, 01.May.2021

Place, date

Signed: VEERAMALLI RAJESH