Automated machine learning plankton taxonomy pipeline

I.C. du Toit

2022

Automated machine learning plankton taxonomy pipeline

Ian Charles du Toit

215087410

April 2022

Submitted in fulfilment of the requirements for the Master of Engineering (Mechatronics) degree in the Faculty of Engineering and the Built Environment at the Nelson Mandela University.

Supervisor: Prof. Barend van Wyk

Declaration

In accordance with Rule G5.6.3, I, Ian Charles du Toit (215087410), hereby declare that this dissertation for the degree of Master of Engineering (Mechatronics) is my work and that it has not previously been submitted for assessment or completion of any postgraduate qualification to another university or another qualification.

Ian Charles du Toit

September 2021

NELSON MANDELA UNIVERSITY

PERMISSION TO SUBMIT FINAL COPIES

OF TREATISE/DISSERTATION/THESIS TO THE EXAMINATION OFFICE

Please type or complete in black ink

FACULTY: Engineering, Built Environment and Technology

SCHOOL/DEPARTMENT: Mechatronics

I, (surname and initials of supervisor) Prof Barend J van Wyk

and (surname and initials of co-supervisor) N/A

the supervisor and co-supervisor respectively for (surname and initials of candidate) I C du Toit

(student number) 215087410 a candidate for the (full description of qualification)

Master of Engineering (Mechatronics)

with a treatise/dissertation/thesis entitled (full title of treatise/dissertation/thesis):

Automated machine learning plankton taxonomy pipeline

It is hereby certified that the proposed amendments to the treatise/dissertation/thesis have been effected and that **permission is granted to the candidate to submit** the final copies of his/her treatise/dissertation/thesis to the examination office.

18 March 2022

SUPERVISOR **DATE**

And

N/A N/A

CO-SUPERVISOR **DATE**

Abstract

Plankton taxonomy is considered a multi-class classification problem. The current state-of-the-art developments in machine learning and phytoplankton taxonomy, such as MorphoCluster, include using a convolutional neural network as a feature extractor and Hierarchical Density-Based Clustering for the classification of plankton and identification of outliers. These convolutional feature extraction algorithms achieved accuracies of 0.78 during the classification process. However, these feature extraction models are trained on clean datasets. They perform very well when analysing previously encountered and well-defined classes but do not perform well when tested on raw datasets expected in field deployment.

Raw plankton datasets are unbalanced; whereas some classes only have one or two samples, others can have thousands. They also exhibit many inter-class similarities with significant size differences. The data can also be in the form of low-resolution, noisy images. Phytoplankton species are also highly biodiverse, meaning that there is always a higher chance of a network encountering unknown sample types. Some samples, such as the various body parts of organisms, are easily confused with the species itself. Marine experts classifying plankton tend to group ambiguous samples according to the highest order to which they are confident they belong. This system leads to a dataset containing conflicting classes and forces the feature extraction network to overfit when training.

This research aims to address these spatial issues and present a feature extraction methodology built upon existing research and novel concepts. The proposed algorithm uses feature extraction methods designed around real-world sample sets and offers an alternative approach to optimizing the features extracted and supplied to the clustering algorithm. The proposed feature extraction methods achieved scores of 0.821 when tested on the same datasets as the general feature extractor. The algorithm also consists of Auxiliary SoftMax classification branches which indicate the class prediction obtained by the feature extraction models. These branches allow for autonomous labelling of the clusters formed during the HDBSCAN algorithm being performed on the extracted features. This results in a fully automated semi-supervised plankton taxonomy pipeline which achieves a classification score of 0.775 on a real-life sample set.

Acknowledgements

I want to take this opportunity to thank those who have encouraged, supported, and been with me throughout my studies. It has been a testing period with all that's gone on in the world, along with the compounded emotional toll of losing my mother.

To Prof Barend van Wyk, you have been inspiring to work with. Your wealth of knowledge, passion for learning, and attention to detail are infectious traits that have not only motivated me but everyone who works alongside you. Thank you for your unwavering support, understanding, and management during this study. You brought clarity to a very dark time in my life whilst dealing with your own uncertain circumstances.

To my father, your never-waning love and compassion for me and my future has set me up on a path for which I am so grateful. What you have invested emotionally and financially has outdone anything I could ever have attained by myself. Growing up watching someone who truly loves as you do has been a blessing, and I hope to be half the man you are.

Finally, to my late mother, thank you for your guidance thus far. I will continuously strive to make you proud. You've paved the way for me, and your memory will live on through my works.

Contents

List of Figures

List of Tables

List of abbreviations

Abbreviation	Term
UVP	Underwater Vision Profiler
LED	Light Emitting Diode
ROI	Region(s) of Interest
PkID	Plankton Identifier
WoRMS	World Register of Marine Species
SVC	Support Vector Classifier
COM	Co-occurrence matrices
CNN	Convolutional Neural Networks
ReLU	Rectified Linear Activation Unit
NDSB	National Data Science Bowl
FC	Fully Connected
DBSCAN	Density-based spatial clustering of applications with noise
PCA	Principle Component Analysis
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
GUI	Graphical User Interface

1. Introduction

1.1 Introduction

Plankton taxonomy is a field of research where marine experts collect, analyse, and annotate various samples of particles freely floating in aquatic systems around the globe. Plankton makes up almost 70% of the earth's oxygen supply. Monitoring the movement and occurrence patterns of this organism is crucial to understanding the world in which humans live. Plankton samples occur in abundance, along with other floating particles in the same environment. There are many methods used to observe these planktonic trends including in-field and in-laboratory observation techniques (Bi et al., 2015). Although effective for obtaining samples, these methods all suffer from the lack of an efficient autonomous class identifier.

Recently, deep neural networks such as convolutional neural networks have taken over from traditional computer vision techniques for plankton feature extraction and identification. These networks are trained under supervision and find inherent and underlying features of plankton without the need to hand-engineer the features as was previously done (Correa et al., 2016). Even more modern methodologies, such as MorphoCluster proposed by Schröder et al (2020), utilize an unsupervised clustering method on top of a convolutional feature extractor to find underlying patterns within the data, thus aiding researchers with a semi-automated, semi-supervised approach (Schröder, Kiko and Koch, 2020). While many plankton taxonomic methods have been proposed, all the investigated methods suffer from a single drawback: they are not trained to handle real-life plankton sample sets.

The need for an automated plankton classification algorithm robust enough to handle real-life sample sets led to the research objective presented in Section 1.2. Section 1.3 provides a discussion of the research objectives for this study followed by an outline of the contributions envisioned in Section 1.4. Section 1.5 provides an overview of the structure of this dissertation, highlighting the topics in each proceeding chapter.

1.2 Problem Statement

Deep learning algorithms for plankton taxonomy are usually trained on clean datasets and do not meet satisfactory performance levels when tested on real-life sample sets. This is because plankton datasets are noisy, unbalanced and contain many classes that exhibit similar features. Training a single convolutional network as a feature extractor on raw plankton datasets results in overfitting and poor testing performance. Convolutional neural networks hard classify all considered samples and therefore do not provide an indication of which samples the classifier does not recognize. Current systems that attempt to solve these issues are inefficient and require human intervention throughout the entire process. To solve these issues a multi-feature extraction method with an autonomous clustering and identification algorithm is required.

This dissertation proposes, implements, and evaluates an automated semi-supervised plankton taxonomy pipeline for real-world applications and conditions by analysing existing plankton taxonomy pipelines and algorithms, determining their underlying shortcomings, and combining existing and novel techniques to improve performance.

1.3 Research Objectives

The purpose of this investigation is to develop an autonomous classification algorithm for plankton taxonomy that is robust enough to be used on real-life sample sets.

Before any research can be conducted to investigate an automated semi-supervised plankton taxonomy pipeline for real-world applications, it is vital to first investigate existing systems used to perform the same task and analyse why they don't perform well in real-life situations. The first research objective is, therefore:

1.3.1 Research objective one: Investigate existing plankton taxonomy pipelines

Research on plankton taxonomy systems is increasing exponentially. Grasping the concepts that have been applied, and their advantages and relevant pitfalls, provides deep insight into the mismatch between reported network accuracies in training versus real-life application. Objective two is built upon this understanding.

1.3.2 Research objective two: Determine the underlying performance lapses in modern plankton taxonomy techniques

While modern plankton taxonomic techniques showcase relatively high laboratory results, a reality gap is present that renders the real-life application of the investigated automatic algorithms impractical. There are several algorithms available to increase the performance of these techniques so that they can be applied effectively. These algorithms need to be implemented and evaluated on real-life datasets. This leads to research objective three.

1.3.3 Research objective three: Propose, implement, and evaluate a binary classification CNN for detritus identification

Identifying samples belonging to classes that exhibit large intra-class differences in real-life plankton sample sets would allow for the practical implementation of general plankton classification networks. Decreasing the number of learnable sample classes from a network's training process increases the accuracy and speed of the network in dealing with specific subsets. This leads to research objective four.

1.3.4 Research objective four: Propose, implement, and evaluate a biological group-based classification CNN for plankton identification

Breaking the real-life dataset into biologically related subsets would increase the ability of deep learning methods to distinguish between samples belonging to classes with high spatial similarities and those with minor inter-class differences. The combination of research objectives three and four is the fifth research objective.

1.3.5 Research objective five: propose, implement, and evaluate the combination models trained with binary and subclass groupings for plankton identification

Once the data-driven classification networks outperform the general architecture on their respective classes, these models need to be combined to deliver an enhanced feature extraction methodology. This feature extractor is envisaged to supply features with a clustering algorithm and will be used to supervise cluster labelling. The final research objective, therefore, is:

1.3.6 Research objective six: Propose, implement, and evaluate automated semi-supervised plankton taxonomy pipeline for real-world application

Finally, the extensive feature extraction network would be used to cluster the resultant features and use auxiliary classification branches and their associated probability scores to label the resulting clusters. The final outcome should be a real-life, data-driven, automated plankton taxonomy pipeline.

1.4 Contributions

In addition to the overall objectives, this research sought to bridge the gaps in related research where similar applications in the field of plankton taxonomy failed to perform efficiently. The research contributed towards the furtherment of previous literature as follows:

1. Created and restructured a dataset comprised of many different plankton classes, including all particles found in a real-life data sample. The noisy samples are the reason all automated plankton studies fail when used in real-life deployment.
2. Trained multiple neural networks specifically designed to extract more relevant features from classes previously disregarded in other studies. This allows for distinguishable patterns even for non-plankton samples occurring in abundance in real-life sample sets.
3. Stacked all the trained models into a single feature extraction model and clustered the resultant feature vector, similarly to how the MorphoCluster system uses a generalized feature extractor trained on a cleaner dataset.
4. Utilized the classification class score obtained within the individual specific feature extractors to label the resultant clusters automatically. This resulted in a semi-supervised automatic plankton taxonomy pipeline for real-life applications.

Publication:

du Toit, I. (2021) ‘Enhanced Deep Learning Feature Extraction for Plankton Taxonomy’, *Proceedings of the International Conference on Artificial Intelligence and its Applications*, pp. 1–8. doi: 10.1145/3487923.3487930.

1.5 Dissertation Layout

This section provides a brief overview of each chapter. A graphical representation of the structure of this dissertation is shown in Figure 1.1.

1.5.1.i Chapter 2 – Background

Chapter 2 presents a study on plankton. It introduces the importance of plankton and how researchers collect, analyse, annotate, and store the various samples. The chapter then investigates the different methodologies applied in the field of plankton taxonomy as well as research results stemming from the turn of the century up until today.

1.5.1.ii Chapter 3 – Literature Review

Chapter 3 presents a review of the machine learning topics relevant to the research investigated. It highlights various supervised and unsupervised methodologies in a way that shows how they build off one another.

1.5.1.iii Chapter 4 – Experimental Methodology

Chapter 4 discusses the thought process behind each investigation undertaken in this study. It presents brief overviews of the individual studies, how they link to each other, the dataset

used, and the evaluation criterions used to validate the performance of the relevant investigations.

1.5.1.iv Chapter 5 – Investigation A

Chapter 5 presents results of a binary classification network for classes exhibiting large intra-class spatial differences.

1.5.1.v Chapter 6 – Investigation B

Chapter 6 presents results of multiple classification networks trained on specific subsets of the original dataset containing classes with high inter-class similarities.

1.5.1.vi Chapter 7 – Investigation C

Chapter 7 presents results of clustering performances on the original dataset, comparing the multi-network feature extractor to a general feature extractor trained on a cleaned dataset.

1.5.1.vii Chapter 8 – Investigation D

Chapter 8 presents results of using the multi-network feature extractor auxiliary classification branches to automatically label the obtained clusters.

1.5.1.viii Chapter 9 – Conclusions

The final chapter draws conclusions from the investigations and provides recommendations for future research.

1.5.1.ix Appendix A

Appendix A provides an overview and breakdown of the original real-life dataset.

1.5.1.x Appendix B

Appendix B shows extracts of the confusion matrices obtained during testing.

1.5.1.xi Appendix C

Appendix C contains various samples of the code used throughout the investigation.

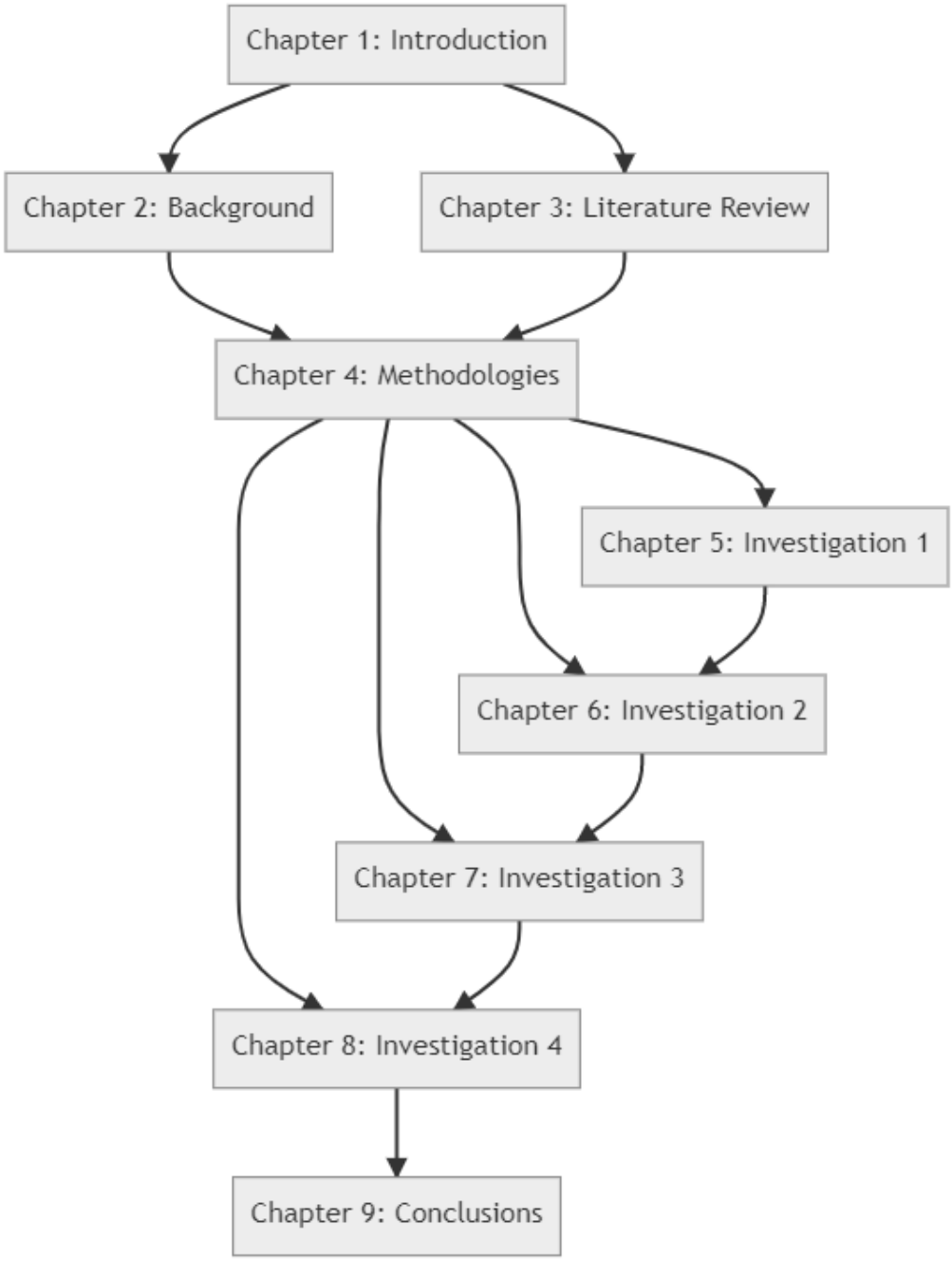

Figure 1.1: Dissertation layout

2 Background

2.1 Plankton Imaging and Acquisition Methods

The acquisition of plankton image data remains a focus of marine researchers and technologists. Plankton occurs in abundance and is generally omnipresent in all ecological aquatic systems. Studying these organisms, no matter what technology is used, takes relatively long periods. Marine researchers have, however, developed numerous technologies to acquire plankton image data such as the ZooSCAN images shown in Figure 2.1.

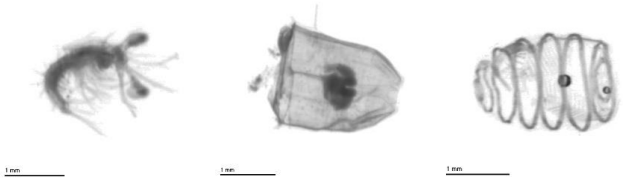

Figure 2.1: ZooSCAN plankton images

These technologies generally fall within one of two categories: In situ and ex situ systems. In situ methods, such as the Underwater Vision Profiler (UVP) system shown in Figure 2.2, allow researchers to analyse plankton samples in their natural environment. Section 2.1.1 highlights the most commonly used In situ plankton observation technologies available.

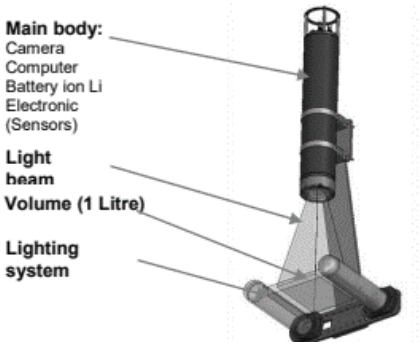

Figure 2.2: Underwater Vision Profiler (Underwater Vision Profiler (UVP) - OceanNet, 2020)

Ex situ methodologies are the main consideration of technologies used in this research. Ex situ technology such as the ZooSCAN, shown in Figure 2.4, allows researchers to analyse

plankton samples outside their natural environment, for example, in laboratories. These methods are discussed in more detail in Section 2.1.2.

Figure 2.3: ZooSCAN (ZooSCAN | EMBRC France, 2010)

2.1.1 In situ methods

In situ technologies are at the forefront of plankton image data acquisition and analysis studies. These methods allow for in-place monitoring of plankton, which ultimately saves a lot of time (Bi *et al.,* 2015). The systems briefly discussed in this section include FlowCytobot and the UVP (Underwater Vision Profiler).

The FlowCytobot, shown in Figure 2.4, generates images of particles using flow cytometry and video technology. Once deployed, the FlowCytobot pushes a constant stream of water through a thin tube. Using flow cytometry technology (McKinnon, 2018), it employs a laser beam to create fluorescent light signals. These light signals cause live plankton cells to illuminate due to the presence of chlorophyll (Matz *et al*., 1999). A camera records the illuminated cells, and the video data is sent to shore for analysis.

The Underwater Vision Profiler, shown in Figure 2.2, uses red light emitting diodes (LED) and computerized optical technology to analyse plankton data at depths reaching up to 6000 metres. The UVP makes use of 100 µs flashes to illuminate an area of 4 x 20 cm with custom red LED lighting and takes an image using a high-quality camera. The UVP returns a sample image representation equivalent to 1 litre of water (Ramondenc *et al*., 2016). The image data can then be monitored almost in real-time on the ship deploying the UVP. Image data is stored on hard drives for later analysis in a laboratory.

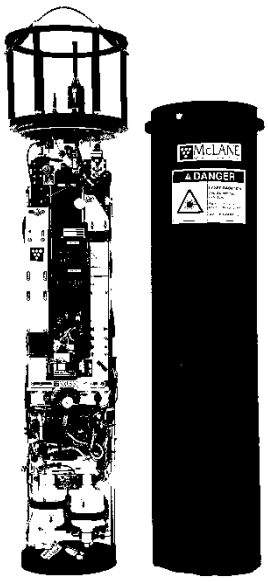

Figure 2.4: FlowCytobot

In situ methods provide a less time-consuming way to analyse plankton data. However, these techniques are still relatively new, so their effectiveness is diminished by image quality. Low resolution negatively affects how well machine learning algorithms perform in the identification of plankton classes.

2.1.2 Ex situ methods

Ex situ methods such as the ZooSCAN system provide a better means of collecting plankton samples for the training and testing of machine learning algorithms. This is because the samples being processed are not moving and are in a closed environment allowing for high resolution still images.

2.1.2.i Sample Acquisition

For ex situ methods, marine researchers must manually catch plankton samples using various techniques. The most popular technique involves using a plankton net, shown in Figure 2.5.

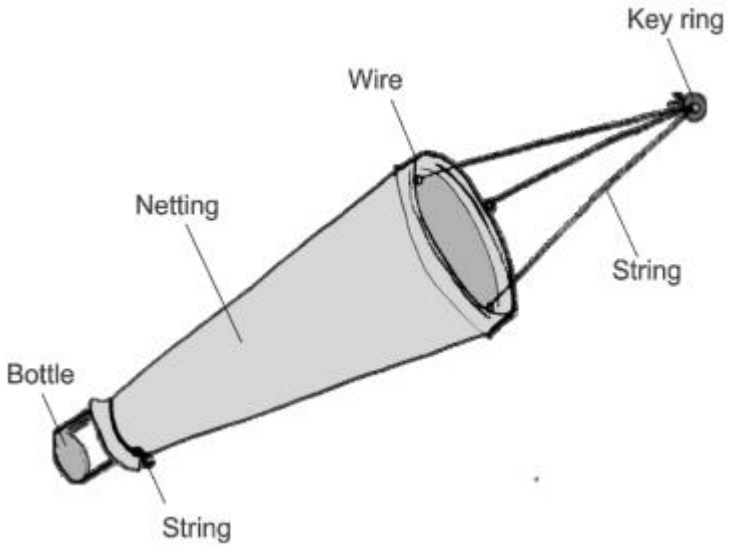

Figure 2.5: Plankton net ('Estuary Education Resources Catching Plankton Estuary Concept', 2012)

The netting is usually made of nylon, a material that allows water to pass through whilst being fine enough to capture the organisms. The net is moved through a body of water of interest by hand or with assistance from some form of vessel. The samples are stored in vials with a volume of 1 litre. Researchers in a marine science laboratory then process the vials one by

one using systems like the ZooSCAN. How the ZooSCAN system processes the samples is explained in more detail in the following section.

2.1.2.ii ZooSCAN and ZooProcess

ZooSCAN and ZooProcess form an integrated system for ex situ digital plankton image acquisition. The ZooSCAN, shown in Figure 2.3, is the hardware component of the system and is comprised of two waterproof elements: the top cover and the base. The top cover is responsible for even illumination of the sample under consideration and measurement of the sample medium's resistance to the transmission of light through an optical density reference cell. The base of the ZooSCAN is where the sample is loaded. It contains a high-resolution imaging device and a drainage passage for sample recovery (Gorsky *et al*., 2010).

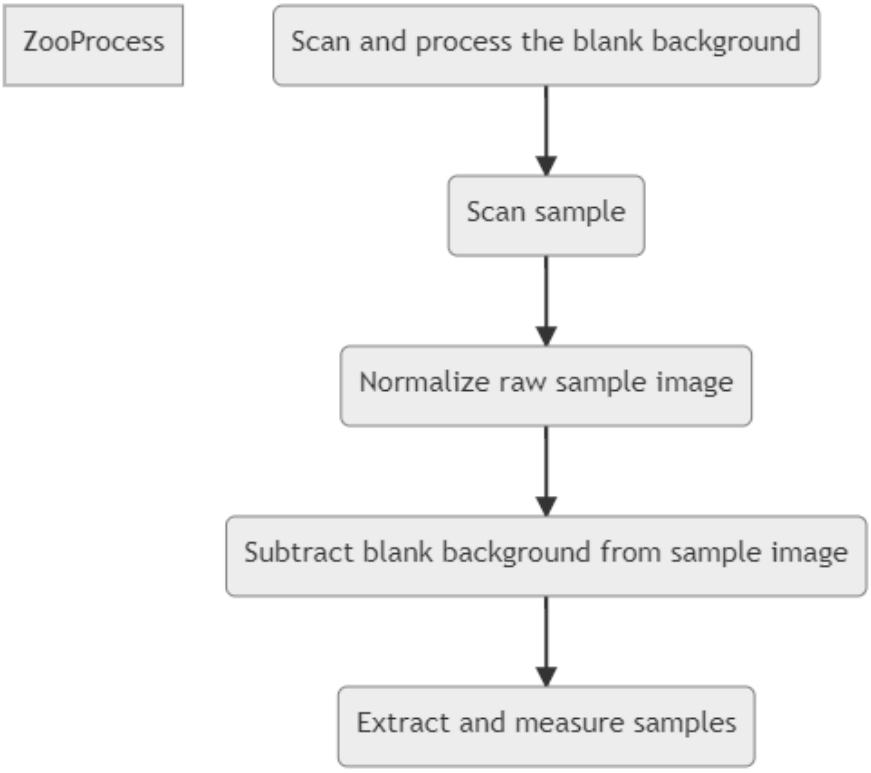

Figure 2.6: Overview of ZooProcess

The ZooProcess is the software component of the system. It is responsible for the scanning, normalization, and object detection of plankton samples. A breakdown of the process is shown in Figure 2.6.

The ZooProcess starts by calibrating and determining the grey level of the background image. This should be performed daily so that researchers can calibrate the ZooSCAN instrument by comparing new background images to those obtained previously. Once a sample is loaded into the ZooSCAN system, the ZooProcess measures the grey levels and compares them to the calibrated value. Overlapping organisms cause problems for vision algorithms, so when samples are loaded, researchers separate and move them using a small stick-like apparatus until all they are visually isolated. Particles along the sides of the frame are discarded by the vision algorithm.

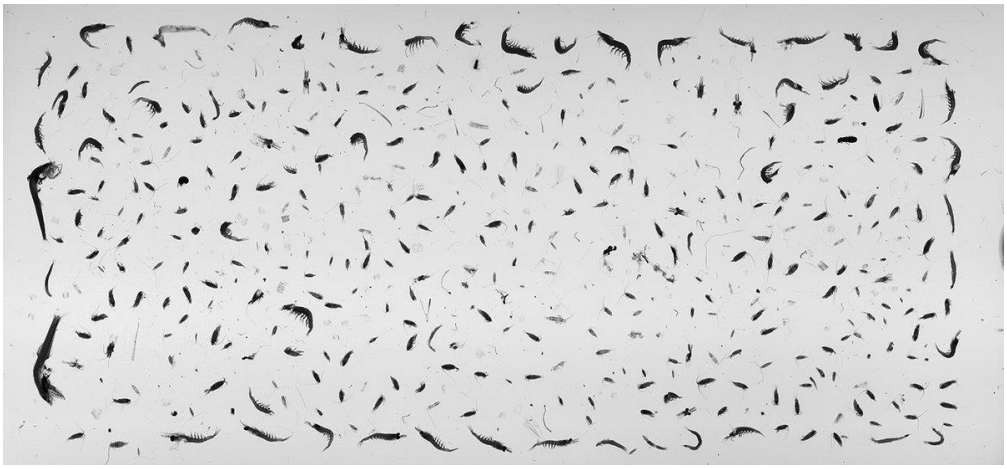

Figure 2.7: Example of raw ZooSCAN image

ZooProcess proceeds to extract the regions of interest (ROI) from the sample and archives them individually in a folder. It also associates the relevant metadata defined by the researcher conducting the scanning process. This metadata contains information about the entire sample set, including location and time data. The ZooProcess also measures the extracted ROI and includes this data along with the metadata in a Logfile. An example of the extracted ROI is shown in Figure 2.7.

The ZooProcess then feeds a standalone application called Plankton Identifier (PkID). This software makes use of more traditional computer vision techniques to predict the class of plankton. These techniques and a more in-depth description of the plankton classes are

discussed in the following sections. An overview of the ZooSCAN process is shown in Figure 2.8.

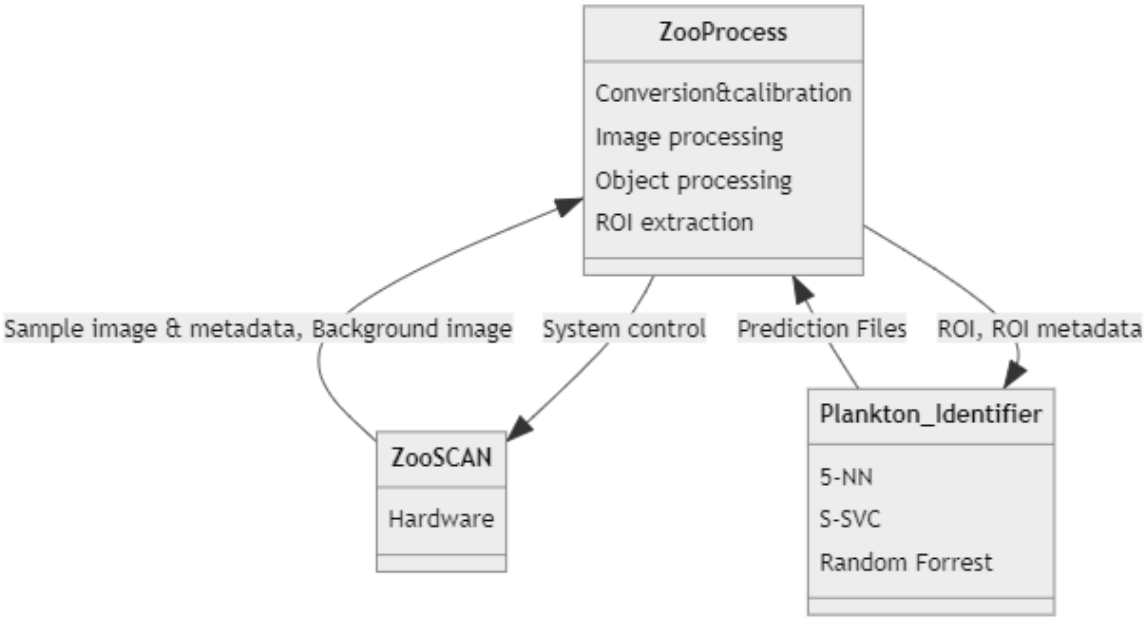

Figure 2.8: ZooSCAN dataflow overview

2.2 What are Plankton?

As an ecological contributor, plankton plays a vital role in aquatic ecosystems and falls at the base of the food chain. Plankton also drives carbon and nutrient cycles, thus influencing global biochemical processes (Keister *et al*., 2012). Dating back 2.4 billion years ago, they are the original contributors to our oxygen-rich atmosphere, giving rise to every living organism on Earth (Falkowski *et al*., 2004).

Plankton, like plants, make their energy through photosynthesis, consuming carbon dioxide and producing oxygen. Their mass consumption of carbon dioxide maintains low acidity levels

of oceans and thus the normal function of all associated systems. They are still the number one contributors to the Earth's oxygen level, contributing to an estimated 80 per cent of the planet's total oxygen supply. Plankton also drives other global nutrient cycles such as the nitrogen, iron, sulphur, and phosphorus cycles (Falkowski, 2012).

Plankton is the organic matter that most oceanic organisms use as their energy supply. Thus, it directly or indirectly affects every aquatic ecosystem. Since 1899, there has been a cumulative loss estimated at 40 per cent of plankton worldwide. This loss could be attributed to many different contributing factors, including increasing ocean surface temperatures (Loeb *et al*., 2021).

Some plankton has a direct impact on all other creatures on earth. Dense blooms of these organisms can drain the oxygen from aquatic systems, in turn suffocating other organisms that share the same environments. Some of these blooms of plankton, known as harmful algal blooms (HAB), can expose humans, whales, and aquatic creatures to fatal toxins. This results in significant economic loss every year in the seafood industry and tourist communities (Schmale *et al*., 2019). The analysis of plankton demographics is crucial to understanding the current state of our world. Various methods have been proposed to acquire and analyse plankton samples. These methods are discussed in Section 2.1. Section 2.3 provides an introduction on how the various plankton organisms are categorized.

2.3 Plankton Taxonomy

Researchers in most biological fields tend to order living organisms into a Linnaean system of classification. In other words, researchers group living organisms based on species the organisms are closely related to. This grouping is called a taxon. Taxa are assigned a taxonomic rank and then split into smaller groups that form a tree-like structure. The overview of the taxonomic ranking is shown in Figure 2.9 and starts with the kingdom at the top, which for all multicellular living organisms on earth is the kingdom Animalia. As one progresses down the pyramid, the diversity of organisms at each level increases. At the base of the pyramid shown in Figure 2.10 is the group called species. Plankton from the same species could have evolved independently with different ecological adaptions. Where plankton researchers are still

unsure if well-defined species exist, they tend to divide them into groups at higher hierarchical levels.

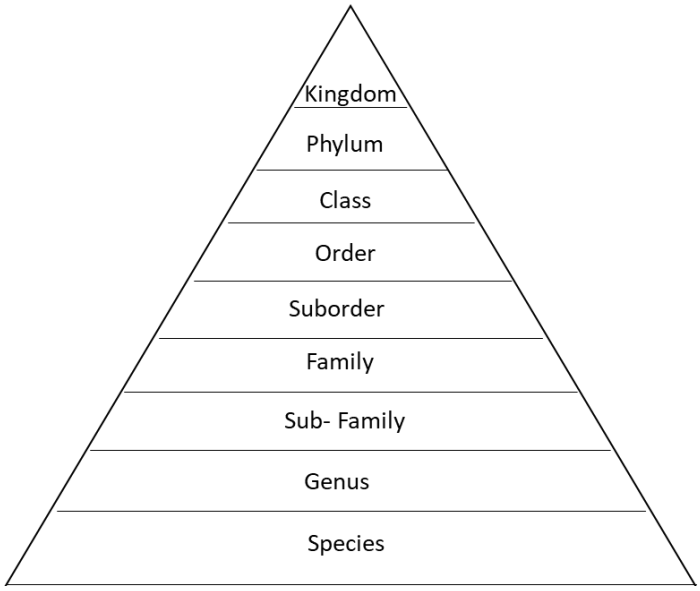

Figure 2.9: Taxonomic ranking (top-down view)

Plankton is divided broadly into three different phylum . These are phytoplankton, zooplankton and bacterioplankton. The phytoplankton group represents approximately ten different classes from four different kingdoms and can vary in sizes from one millimetre to one micron in length. They also exhibit various shapes and textures, all of which are used to identify which species of phytoplankton they belong to. Zooplankton and bacterioplankton are also classified based on their varying features. The World Register of Marine Species (WoRMS) provides a comprehensive list of all marine organisms, including their scientific names and household names, to guide the interpretation of most taxonomic literature. WoRMS is maintained by taxonomic experts, and new information is added daily to keep the system up to date.

Marine specialists utilize technology like the ZooSCAN system described in the sections above to collect, identify, and count plankton samples. These sample classifications are validated using the WoRMS platform and then stored along with their metadata in an online database called EcoTaxa.

2.3.1 EcoTaxa

EcoTaxa is an extensive online database and web application dedicated to the visual exploration and taxonomic annotation of planktonic biodiversity (Picheral, Colin and Irisson, 2017). The database houses over 160 million plankton images from researchers and facilities all over the world. The web app allows marine researchers and general enthusiasts to explore, download and contribute to the database in multiple research projects. An example of the web app interface is shown in Figure 2.10.

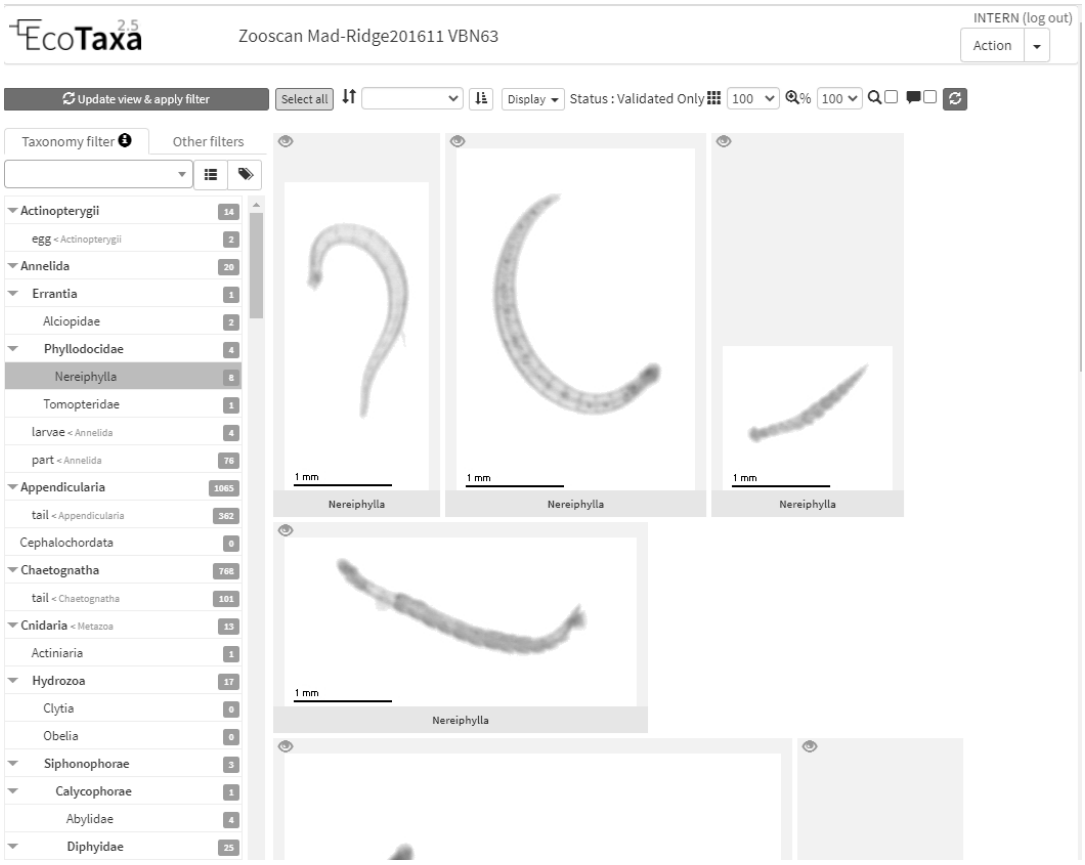

Figure 2.10: EcoTaxa Interface

The EcoTaxa web application being utilized along with the ZooSCAN system is shown in Figure 2.11. Large databases with many contributors, such as EcoTaxa, are prone to mislabelling and human error. Therefore, EcoTaxa and ZooSCAN come equipped with their own plankton identification components, which use machine learning algorithms to classify plankton ROI

into their respective classes. The machine learning methodologies used in these applications are based on more traditional hand-selected feature engineering techniques and achieve accuracies, on average, of about 50 per cent. These low accuracies result in researchers having to validate all the prediction results, essentially rendering the entire automatic annotation process unnecessary. This process of annotation validation is the bottleneck in plankton analysis related to processing time.

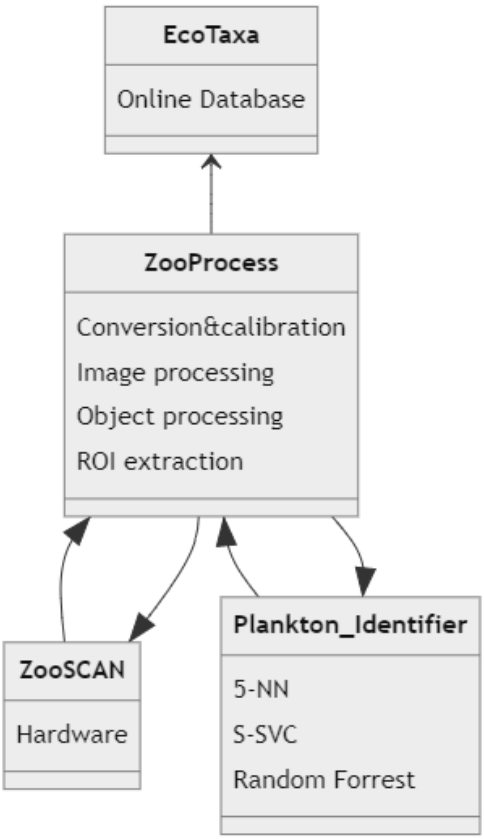

Figure 2.11: ZooSCAN Workflow using EcoTaxa

The following sections investigate various computer vision techniques used to classify plankton into their relative taxonomic groupings. Starting with the more traditional hand-selected feature-based methodologies, Section 2.4 gives insight into the algorithm architectures and their relative performances. Section 2.5 then introduces the newer deep learning and clustering methods used in current state-of-the-art systems and how they measure up in terms of performance. A breakdown of how modern machine learning techniques work is discussed further in the literature review (Chapter 3).

2.4 Feature Engineering-Based Plankton Classification Techniques

Before the wide adoption of modern deep learning methodologies, marine researchers used more traditional computer vision techniques to extract visual features from plankton image data. This section briefly introduces these techniques and the machine learning models that produce predictions based on the engineered features. This section also highlights the results achieved by these machine learning algorithms to compare their ability to classify plankton samples with modern deep learning techniques.

To apply more traditional machine learning methods to plankton images, a comprehensive amount of feature engineering must be performed. This includes extracting geometric, greyscale and texture features. Some geometric or morphological feature extraction methods include corner detection, curve fitting, and edge detection, which allow size, area, and elongation calculations. Greyscale feature extraction methods utilize image pixel values that range between 0 and 256 in intensity. Greyscale features, otherwise known as statistical features, include the pixel values' sum, mean, and standard deviation. Texture feature extraction methods, such as the Gabor filter shown in Figure 2.12, act as a bandpass filter that convolves an image highlight and extracts texture patterns within the original image.

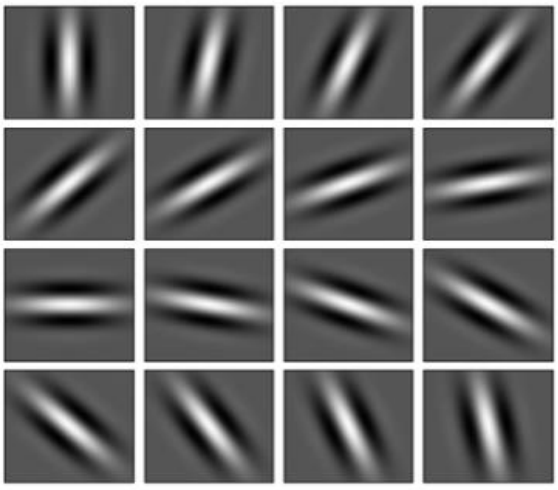

Figure 2.12: 16 Gabor filters for texture detection at different angles (Shah, 2018)

The above-mentioned feature extraction techniques, although highly effective, require extensive parameter selection and are susceptible to changes in the environment processing

the images. Machine learning algorithms that have been used along with these hand-selected features in plankton classification include support vector machines (SVM) and random forest.

2.4.1 SVM

Support vector machines are versatile machine learning algorithms that perform very well in regression and anomaly detection as well as linear and nonlinear binary classification tasks. The SVMs applied in most plankton classification instances use the hand-engineered feature extractors as described in Section 2.4 and perform nonlinear classification on the relevant data points representing each sample (Hu and Davis, 2005). These data points are in the form of a vector representing either one of two classes. SVM is a binary nonlinear classifier trained to determine a nonlinear hyperplane between the two different sample types. This is accomplished by a generalized dot product of the two vectors projecting the nonlinearly separable data into a higher-dimensional space where, according to Cover's theorem, even nonlinearly separable datasets have a high probability of becoming separable. This generalized dot product is also known as kernel tricks, where different kernel functions make use of these dot products to solve for the SVM hyperplane optimization.

Hu and Davis proposed implementing an SVM classifier for plankton features extracted using co-occurrence matrices (COM). COM extractors use pixel brightness to localize features from images. Their study involved a cleaned dataset comprised of 20 000 plankton images from seven different categories and achieved an accuracy of 0.74 (Hu and Davis, 2005).

Lue *et al*. (2005) introduced a system called SIPPER, which took advantage of a one vs all style SVM. Their technique accomplished an accuracy of 0.90 on six classes (Luo *et al*., 2005).

Sosik and Olson (2007) proposed combining geometric, texture, orientation invariant moments, diffraction pattern sampling, and co-occurrence matrix features. An SVM was used to classify the resultant features and achieved an accuracy of 0.88 on 22 different categories (Sosik and Olson, 2007).

2.4.2 Random Forest

Decision trees, such as that shown in Figure 2.13 are the building blocks of the random forest algorithm. Decision trees aim to build models using training data to predict the value or class of target by learning simple choice rules. Decision trees comprise leaf nodes, internal nodes, and branches. Leaf nodes correspond to the class label or outcome. Internal nodes represent the features of the dataset, and branches represent the decision rules. The first decision node in the tree is called the root node.

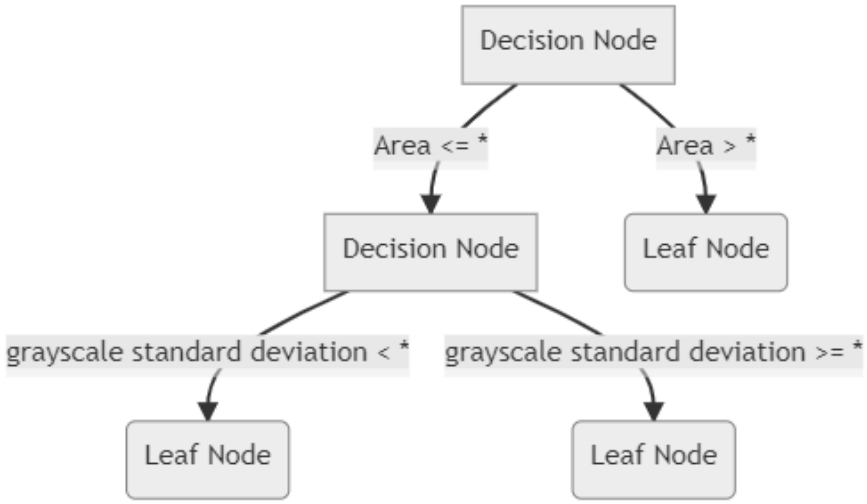

Figure 2.13: Example decision tree

Decision trees are built in a way that minimizes their size. To achieve this, the algorithm uses information gain to decide which features to split at each step, thus producing the purest child nodes. Information gain measures the importance of certain features in the identification of a specific class. A standard measure of information gain is known as Gini Impurity.

As shown in Figure 2.14, a random forest is a supervised machine learning algorithm that builds an ensemble of decision trees. The difference between random forest and decision tree algorithms is that the random forest algorithm randomly establishes root notes and leaf nodes r and uses the bagging method to render predictions. The bagging method takes advantage of a training dataset comprised of features with labels and uses multiple decision

trees that respond differently depending on the data fed to them. The root of each decision tree corresponds to a specific set of features from the dataset. For example, a single tree in a random forest algorithm could predict samples based on only geometric features. The final prediction of the random forest algorithm is an amalgamation of the predictions of all the decision trees that it comprises.

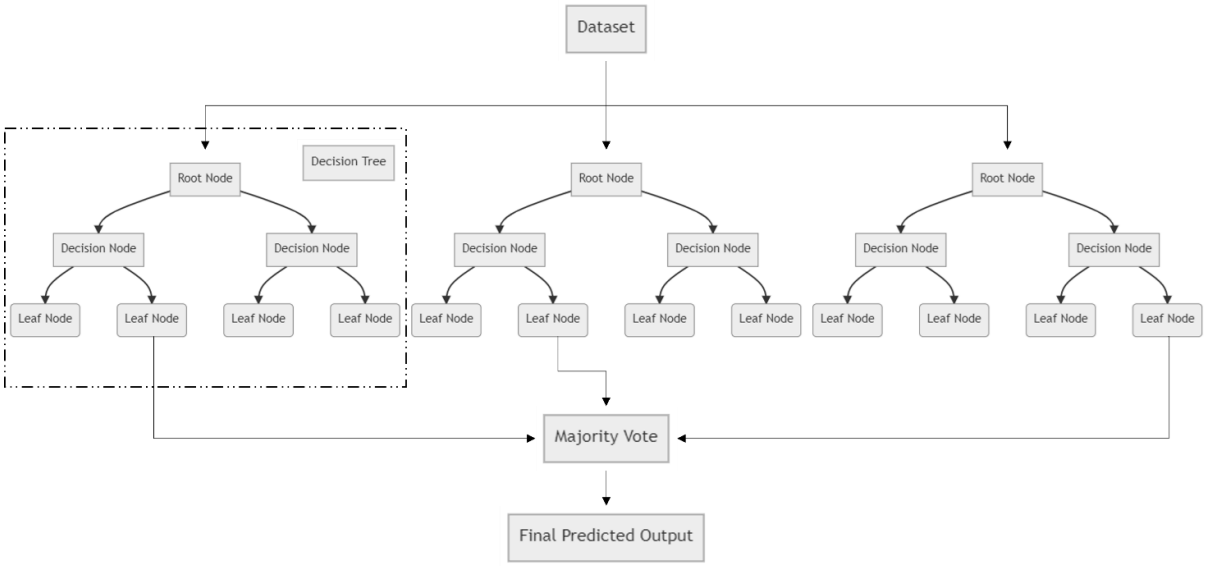

Figure 2.14: Example of a random forest algorithm

Grosjean et al. (2004) proposed the implementation of a random forest algorithm trained on two datasets of 1000 objects each. One of the datasets was divided into eight simplified classes, and the other dataset was divided into 29 more detailed classes. A supplementary algorithm was proposed to identify which samples had produced low confidence scores and needed manual validation from marine experts. The algorithm classified the geometric and greyscale features extracted from zooplankton with an accuracy of 0.75.

ZooImage, a random forest plankton image analysis software, was proposed by Bell and Hopcroft (Bell and Hopcroft, 2008). The algorithm achieved a recall accuracy of 0.817 when classifying a cleaned dataset split into 53 different classes. The authors noted that the ZooImage algorithm did not accurately identify the underlying subphylum classes within the test sample's numerically dominant taxon called copepods. The best performance the

algorithm could achieve using hand-engineered features was classifying copepods as either large, medium or small (Bell and Hopcroft, 2008).

Fernandes et al. (2009) made use of ZooImage to determine how different geometric and greyscale features affect the algorithm. The study showed that features such as the shape of samples are not indicative features. On closer inspection of the confusion matrix produced in the study, the shape characteristic attributed to large amounts of misclassification within the testing dataset.

The studies conducted by Fernandes et al. (2009), Bell and Hopcroft (2008) indicated that the performance of machine learning algorithms used for plankton classification was highly dependent on the feature sets used. The feature engineering process requires much effort and poses challenges to research involved with introducing the system to new environments or introducing new classes to an existing system. Modern machine learning algorithms such as convolutional neural networks (CNN) have been successfully used to overcome the issue of feature engineering. Section 2.5 provides an overview of the performances achieved in studies that utilize these algorithms (Bell and Hopcroft, 2008; Fernandes *et al*., 2009).

2.5 Existing Deep Learning Plankton Classification Systems

This section highlights the performances of various automated deep learning plankton classification approaches. Most of the studies investigated have convolutional neural networks at their core. The literature review chapter discusses a deeper insight into how these networks and the unsupervised methodologies that use them work.

2.5.1 Convolutional Neural Network Algorithms

In 2017 Iago Correa *et al*. proposed using convolutional neural networks to classify microalgae. The study achieved an accuracy of 0.886 on a dataset acquired using the FlowCAM device. The dataset contained 29 449 samples belonging to 19 different classes. The CNN architecture used in the study was built using five convolutional layers and three fully

connected layers. The architecture also made use of ReLU activation functions, max pooling and dropout. (Correa *et al*., 2016).

The success garnered in early studies using CNNs and further advancements in CNN architectures propelled large amounts of research into using CNNs to solve the plankton classification problem. In the same year, Dai *et al*. proposed a CNN architecture called ZooplanktoNet (Dai *et al*., 2016), which outperformed AlexNet and CaffeNet on a dataset made up of 9460 samples from 13 different classes. Dai *et al*. also proposed a hybrid CNN architecture that utilized three different CNN architectures simultaneously. The architectures were trained on the WHOI-Plankton dataset, which consisted of 30 classes with 1000 samples each. Using an ensemble of GooLeNet CNN architectures connected via their fully connected layers resulted in an accuracy of 0.963 (Dai *et al*., 2017).

Li and Cui proposed a deep residual network based on VGGNet. The network achieved an accuracy of 0.73 on a dataset consisting of 30 336 samples from 121 different classes (Li and Cui, 2016). Lumini *et al*. performed a study to test which networks performed best on the three most well-known public plankton datasets. The results of the ZooSCAN dataset, which contains 3771 images belonging to 20 classes, are shown in Table 1 (Lumini, Nanni and Maguolo, 2019). Other research conducted by Cheng *et al*. also compared the performance of various CNN architectures on the NDSB dataset, and the result is shown in Table 2 (Cheng *et al*., 2019).

Table 1: Comparison of CNN architectures performance on ZooSCAN dataset (Lumini, Nanni and Maguolo, 2019)

	Lumini, ZooSCAN dataset, architecture benchmark	No of Classes
AlexNet	80%	20
GoogLeNet	84%	20
VGG16	85%	20
VGG19	84%	20
ResNet50	85%	20
ResNet101	85%	20
DenseNet	88%	20

Table 2: Comparison of CNN architectures performance on National Data Science Bowl dataset (Cheng *et al*., 2019)

	Cheng, National Data Science Bowl dataset, architecture benchmark	No of Classes
AlexNet	85%	7
GoogLeNet	87%	7
VGG16	87%	7
VGG19	87%	7
ResNet50	88%	7

Training these CNN networks for classification can be done in many ways, but the last layer CNN is usually flattened and used as a classification layer at the end of the network. The convolutional layer weights are updated based on the loss function result achieved in the classification layer; this eventually yields various filters specifically trained to distinguish between the different phytoplankton taxa. Once the model is trained, the output layer shown in Figure 2.15 is removed or replaced with an Identity Matrix.

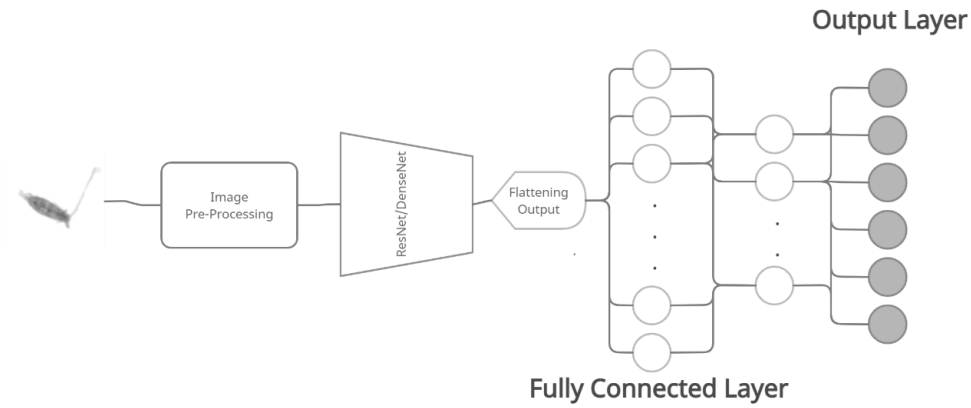

Figure 2.15: Basic overview of a fine-tuned CNN architecture

The model is then used as a feature extractor outputting a higher dimensional vector representing each ROI. These features can be used to cluster ROI data into groups. The MorphoCluster algorithm discussed in the Section 2.5.2 utilizes a CNN model as a feature

extractor. It applies Hierarchical Density-Based Clustering with Noise (DBSCAN) and marine specialist annotation to classify plankton samples.

2.5.2 MorphoCluster

Schröder et al. (2020) proposed an efficient and accurate plankton image annotation software tool called MorphoCluster, shown in Figure 2.16. This sub-section provides an overview of how the MorphoCluster system works. The literature review chapter provides a more in-depth breakdown of how the algorithms and techniques used within the MorphoCluster system (like CNN, HDBSCAN, and transfer learning) work.

As shown in all the investigated methodologies for plankton classification, supervised learning methods are only as effective as the data used to train them. In the field of plankton taxonomy, there are over 4000 different species. Plankton is very biodiverse with many different classes and minor intraclass differences. If utilizing only supervised techniques, the algorithm could process classes of plankton not previously encountered by your network. This presents the need for clustering data features extracted using convolutional networks and identifying outliers (Salmaso, Naselli-Flores and Padisák, 2015).

Various clustering algorithms have been used to group the features extracted from the CNN models. This is accomplished by training the model to classify the dataset, putting the model into inference mode, and removing the final output layer of the model to yield a vector of features extracted by the CNN. This high dimensionality can be reduced by methods such a PCA or used as is with a higher dimensional clustering algorithm.

The MorphoCluster study conducted by Schröder et al. (2020) used a large dataset of 1 million unlabelled images and 584 thousand labelled images. To train the feature extractor CNN, the 584 thousand labelled images were sorted into 65 classes. The labelled dataset was also separated into 392 thousand samples for training and 192 thousand samples for validation. The chosen CNN architecture used was a ResNet18 network, which was trained on the ImageNet dataset and then fine-tuned to the plankton classification task. The ResNet18 architecture achieved an accuracy of 0.738 on the validation set.

The MorphoCluster algorithm uses the above-mentioned feature extractor to cluster the entire dataset. The clustering performed is HDBSCAN, which by its nature and the selected clustering parameters, initially only identifies the densest cluster regions whilst rejecting most samples as noise. The initial detected dense regions of the feature space are used as the cluster seeds for the next step of the algorithm.

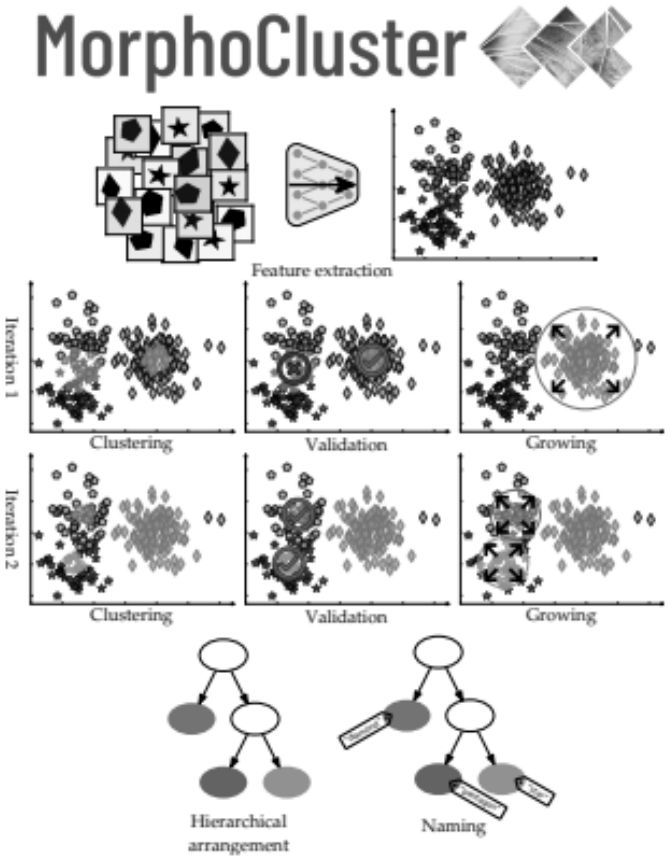

Figure 2.16: Overview of the MorphoCluster algorithm (Schröder, Kiko and Koch, 2020)

The next part of the algorithm is called the cluster validation stage. Users are prompted by a graphical user interface (GUI) to validate the cluster purities. This is accomplished by presenting the user with each identified cluster, one by one. The samples within the detected cluster are presented so that the two most dissimilar samples within the cluster are shown next to one another. The user can then choose to validate the cluster as pure or impure.

Samples within the impure cluster are returned to the unclustered set of data. An example of the GUI is shown in Figure 2.17.

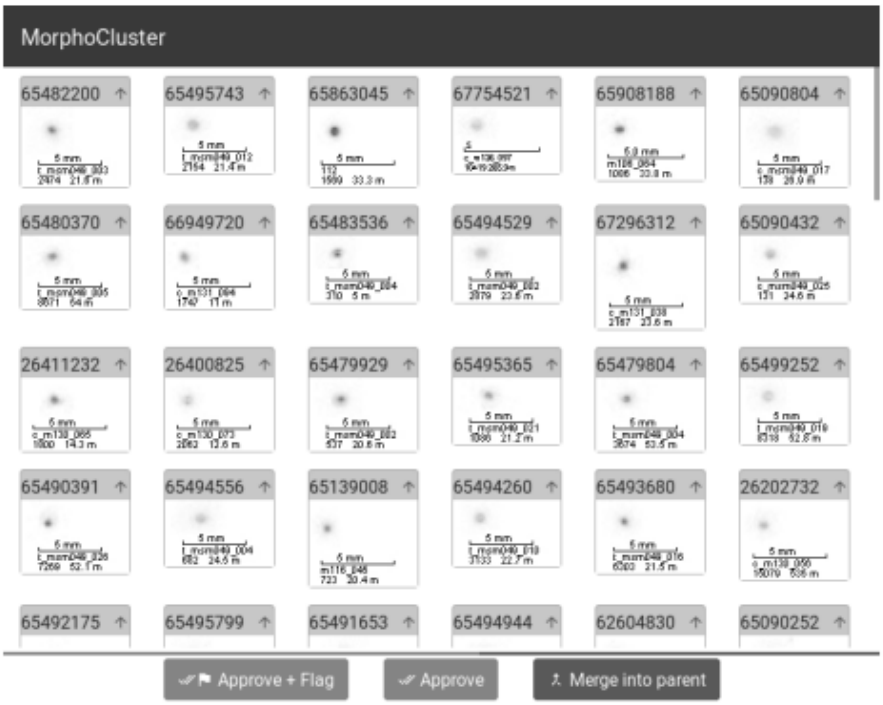

Figure 2.17: MorphoCluster validation GUI (Schröder, Kiko and Koch, 2020)

The cluster growing step allows users to investigate the boundaries of the identified pure clusters. Objects deemed to exhibit similar features to the cluster seeds are then displayed to the user in descending order of similarity. The user then moves through the list of recommended samples until the samples no longer represent the clusters. As there are potentially many samples the user would have to traverse, the algorithm uses a binary search to speed up the process. Once the user encounters and indicates that the proposed samples no longer belong to the cluster, the algorithm moves to the next pure cluster seed. The samples associated with a cluster are removed from the unclustered set.

In the next iteration of the algorithm, the HDBSCAN parameters are reduced to find smaller dense regions of data points. This more conservative clustering and the lower amount of samples present allow the algorithm to find more fine-grained details between the clusters. This process is repeated until all the samples belong to either a cluster or are deemed as noise.

The algorithm then presents the clusters to the user to be named and ranked hierarchically. The clusters with the same name are merged.

MorphoCluster is a highly effective method for plankton annotation and achieves 95% accuracy on the validation set. This method outperforms any plankton classification methods discussed. However, it is not fully autonomous and therefore still requires the laborious task of researchers having to spend hours manually classifying samples (Schröder, Kiko and Koch, 2020).

3 Literature Review

This chapter provides an in-depth literature review of machine learning concepts researched in Chapter 2 and implemented in Chapter 4. Section 3.1 introduces convolutional neural networks. Here, the general workings, various layers, and loss functions are discussed of a CNN are discussed. Section 3.2 presents the problems encountered by a CNN and the optimization techniques used to solve them. Section 3.3 provides an overview of the current state-of-the-art CNN architectures used in most of the considered studies. Section 3.4 discusses how CNNs are evolved to extract spatial features from the datasets used to train them. This section also highlights the effectiveness of CNNs to extract features and introduces the various techniques used to augment and transform image data into more consumable objects. Section 3.5 introduces the use of unsupervised machine learning techniques, including dimension reduction and clustering. Section 3.6 concludes the chapter and brings together all the concepts from chapters 2 and 3 to provide a basis for the investigations undertaken in the proceeding chapters.

3.1 Convolutional Neural Networks

3.1.1 Introduction

Convolutional neural networks (CNN) are different to the more traditional feed-forward neural networks in that they are not only made up of fully connected (FC) layers. FC layers are explained in detail later in this section. Their basic principle is that the output of every neuron within a layer is connected to the input of every other neuron in the next layer. An example of a multilayer perceptron composed only of FC layers is shown in Figure 3.1. A CNN is defined as a neural network where at least one of the FC layers is replaced with a specialized convolutional layer. A typical CNN is constructed by first using convolutional layers and then ending the network with FC layers. These layers are also explained in further detail in Section 3.1.2.

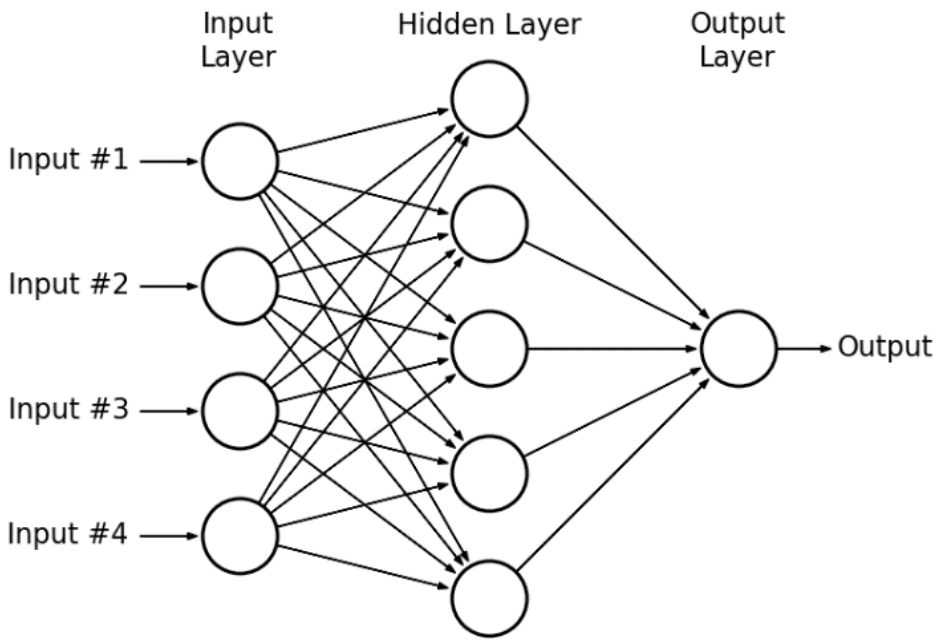

Figure 3.1: Example of a multilayer perceptron with FC layer (Zahran, 2021)

In the field of computer vision, researchers attempt to bridge the gap between the human visual cortex and the abilities of machines to mimic this characteristic. This is done by visualizing all images as just a set of pixels with varying intensities, as shown in Figure 3.2. Initially, researchers would convert the image of interest into its relative pixel values, flatten the resultant matrix into a single row vector representation of the image, and then train multilayer neural networks on these as two-dimensional pixel vectors. The loss of spatial information when performing this sort of image transformation results in only simple binary images being classified at relatively poor levels of accuracy. In the case of images that have RGB (red, green and blue) channels representing the pixel values and larger dimensions, using multilayer perceptrons to process these images becomes exponentially more expensive to compute.

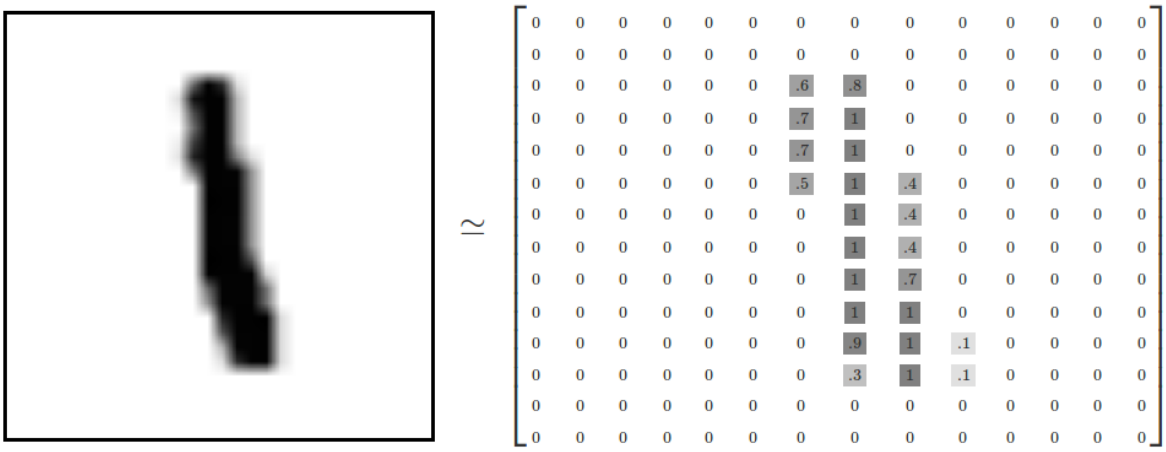

Figure 3.2: Visualizing an image of a "1" as pixel values

The CNN, also known as ConvNets, was first introduced by Yann LeCun in 1980. The challenges experienced by the founders of the idea were linked to the limited computing power available at the time. The first real advancement toward achieving state-of-the-art results using ConvNets was when Krizhevsky et al. (2012) proposed AlexNet ConvNets, unlike the multilayer perceptron, make use of filters to identify features of the input image. Each CNN applies up to thousands of different filters and feeds them to the next layer (Krizhevsky, Sutskever and Hinton, 2012). An example CNN is shown in Figure 3.3.

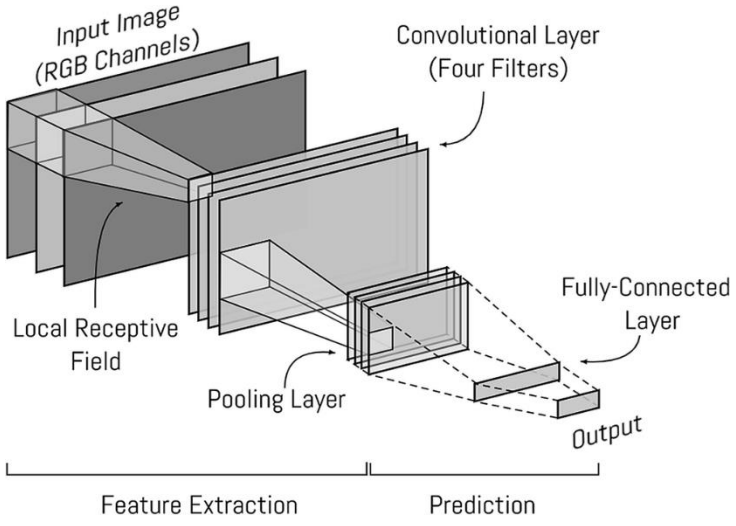

Figure 3.3: Convolutional neural network example (Millar *et al.*, 2019)

The goal of ConvNets using this filtering methodology is to reduce the input image into a much simpler, processable form without losing any crucial information. ConvNets use convolutional layers, which use convolution operations to generate representative feature maps of the original image. The main advantage of using ConvNets is that they automatically learn the values of these filters by using the low-level features to detect higher-level features, eventually allowing the CNN to make predictions regarding the input image. This a known as compositionality. Another key benefit to the application of ConvNets is their local invariance. No matter the location of the object of interest within the image, the ConvNets will still be able to identify the regions with high responses to a certain filter. The next section explains what convolutions are and how the ConvNets convolutional layers work.

3.1.2 Convolutional Layers

Convolutional layers are the building blocks of ConvNets. They are made up of small square templates called convolutional kernels. These kernels are slid across the input image to detect certain patterns in the image's pixel values, as shown in Figure 3.3 and Figure 3.4. To explore the working of convolutional layers, this section starts by covering two essential topics: kernels and convolutions.

Convolutions are generally denoted by the $\star$ operator. The mathematical expression representing a convolution of a two-dimensional image X and a two-dimensional kernel Y is shown in equation 1.

$$Z(i,j) = (X \star Y)(i,j) \sum_{m} \sum_{n} Y(i+m, j+n) X(m,n) \quad (1)$$

Where i and j represent the actual position of the centred pixel and m and n represent the size of the image. As shown in Equation 1, images convolved with kernels produce new representations of the original image. This can be done in many ways. Traditionally, hand-crafted kernels are used to perform various image processing functions. Some of these functions include blurring, sharpening and edge detection. Figure 3.4 shows an example of

how convoluting a two-dimensional image of a cross with a two-dimensional kernel is built to detect horizontal lines.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \star \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.4: Convoluting an image of a cross with a horizontal line-detecting kernel

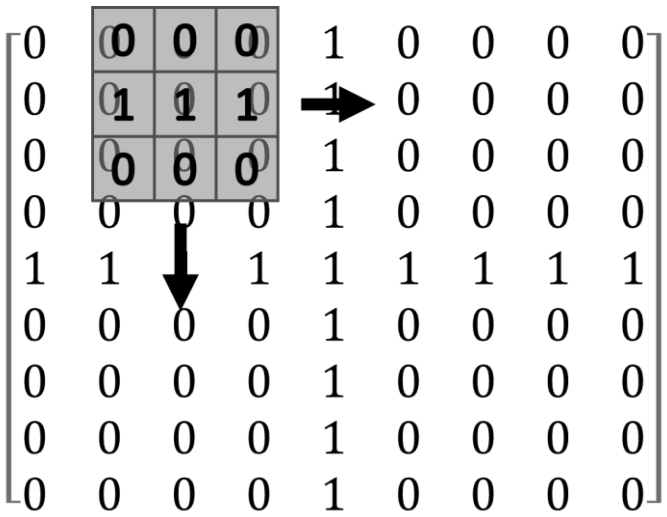

Figure 3.5: Example showing the sliding kernel

As shown in Figure 3.4, areas within the image (larger matrix on the left) that match the convolutional kernel return increasingly larger values based on their similarity. Areas of the input image that do not match the convolutional kernel template output small values or even zero. From Equation 1, Figures 3.4 and Figure 3.5 convolutions are the sum of element-wise matrix multiplication between a convolutional kernel and the area that the kernel is currently occupying within the target image.

To ensure that the spatial dimension of the input image is kept in the resultant output, a concept called padding is applied. Pixels sitting on the edge of the input image, under normal

circumstances, can never be considered as the centre point of the sliding kernel. This ultimately leads to ever decreasing output sizes depending on the size of the kernel filter being applied. Padding is a technique that replicates pixels on the border of the image so that the relevant kernel can be applied to the edges of the original input.

The stride of a kernel is defined as the step size of the kernel when sliding through the image. Equation 2 shows how the parameters such as stride, padding, input image volume and kernel size affect the output volume of the convolution function.

$$Size(Z) = \frac{Size(X) - Size(Y) + 2P}{S} + 1 \qquad (2)$$

Where S represents the stride, and the padding is shown as P. In general, setting the padding as shown in Equation 3 and the stride as 1 ensures that the output size of Z is the same as the input size of X.

$$P = \frac{(Size(Y) - 1)}{2} \qquad (3)$$

Convolutional layers are made up of the kernel filters described above, except that the kernels used in these layers are called learnable kernel filters. Each filter has a width and a height, and as the kernels and images are usually square, the resultant filter is also square. A set of filters is also known as the layer's depth. An example of this is an input image to a CNN that has three channels: Red, Green and Blue (RGB) has a depth of 3. For filters deeper in the network, the depth is dependent on the number of filters from the previous layer. Figure 3.6 shows how the depth of the activation map is equal to the number of learnable filters in the current layer.

During a forward pass of the network, every filter is convolved across the input volume, and as shown in Equation 1, a two-dimensional output activation map of the filter is produced. Every output activation map is essentially a neuron that inspects small regions of the input, learning new filters that activate when they encounter certain features. Earlier in the network, the filters may start when they encounter more rudimental features such as corners or edges. Deeper layers are dependent on previous layers and may activate more detailed features such as certain identifiable parts of the image, including the tails or legs of plankton samples.

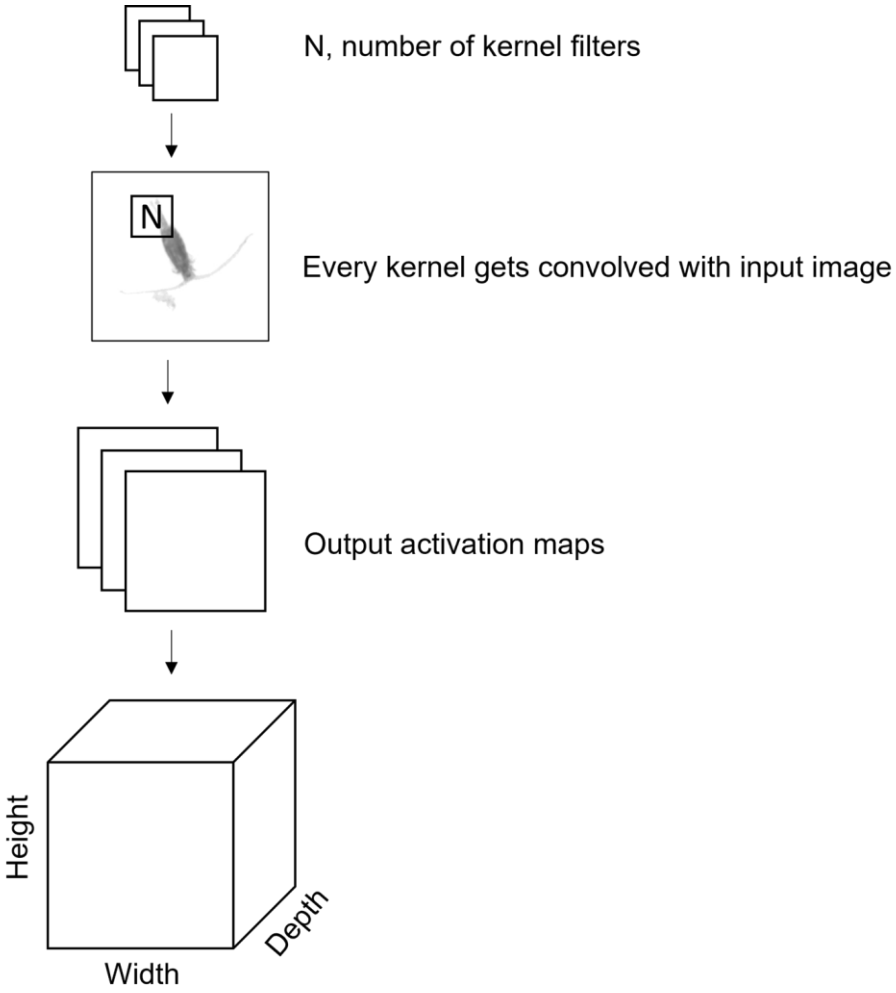

Figure 3.6: Process of convolutional layers

ConvNets are built by stacking these convolutional layers within sequence with pooling layers, activation layers, and FC layers. When training the ConvNet, a sample labelled image is passed through the network. These other types of layers are described in the next section along with network optimization layers, namely dropout and batch normalization layers.

3.1.3 Pooling Layers

As described in Section 3.1.2, the convolutional layers are essentially just a stack of feature maps. The deeper the network, the more detailed features can be identified by the filters. Training ConvNets on more complicated datasets with many classes will require more filter

maps; each filter map is responsible for finding a certain pattern in the image. Increasing the number of filters increases the size of the convolutional layer stack, which subsequently increases the dimensionality of the convolutional layers. This higher dimensionality means that more parameters are necessary, and this can lead to overfitting the data. The concept of overfitting is described in Section 3.1.8. Pooling layers are used to reduce the dimensionality of deeper networks. There are different types of pooling layers, the most common being the max pooling layer. Max pooling layers take the resultant feature maps produced by the convolutional layers and reduce them to smaller representative feature maps, as shown in Figure 3.7.

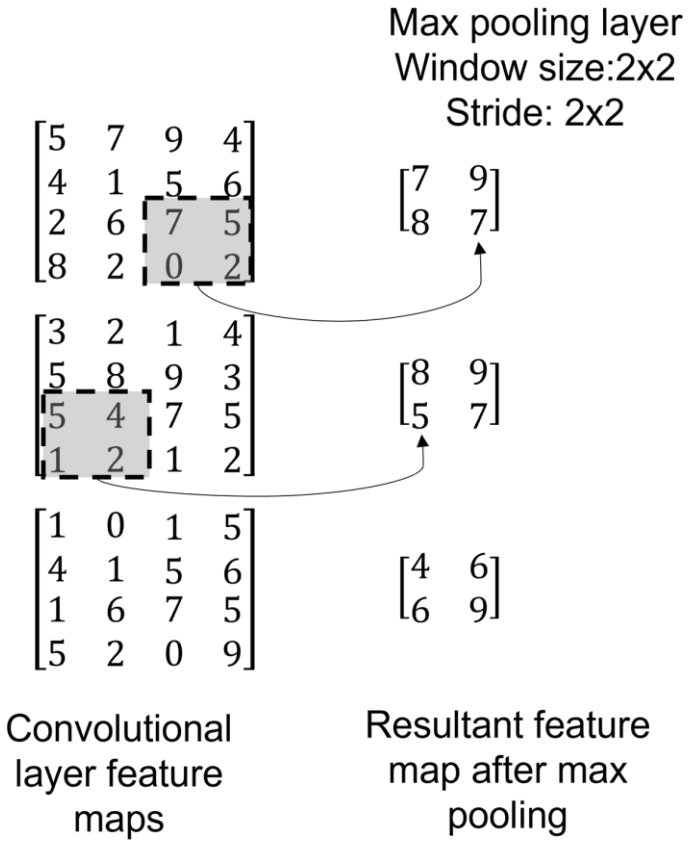

Figure 3.7: Max pooling with a 2 x 2 window size and a stride of 2

As with convolutional layers, max pooling layers also use a window size (like kernel size in convolutional layers) and a stride parameter. Starting at the top left of the input feature map, the max pooling layer slides a kernel over the feature map, in each step taking the maximum value of the containing pixels. In the case of the example shown in Figure 3.7, the resultant feature map is half the width and height of the original input. Other types of pooling layers include global pooling and average pooling. In global pooling, the average value of the entire

input feature map is used. Average pooling uses the same idea as max pooling except that instead of using the maximum value, an average value of the kernel contents is used.

3.1.4 Activation Layers

Activation layers are not actual layers like convolutional layers or pooling layers, as no weights or parameters are learnt inside these layers. However, researchers tend to include them in network architecture diagrams to clarify which type of activation function is being used. Activation functions are used to determine the output of a layer within a neural network and constrain the resulting value within a specified range depending on the function. ConvNets use nonlinear activation functions, including a rectified linear unit (ReLU), as shown in Figure 3.8, and Leaky ReLU, as shown in Figure 3.9. Generally, these functions are applied immediately following convolutions. Nonlinear activation functions allow models to adapt better and generalize a diverse amount of data as well as differentiate between outputs (Millar *et al*., 2019).

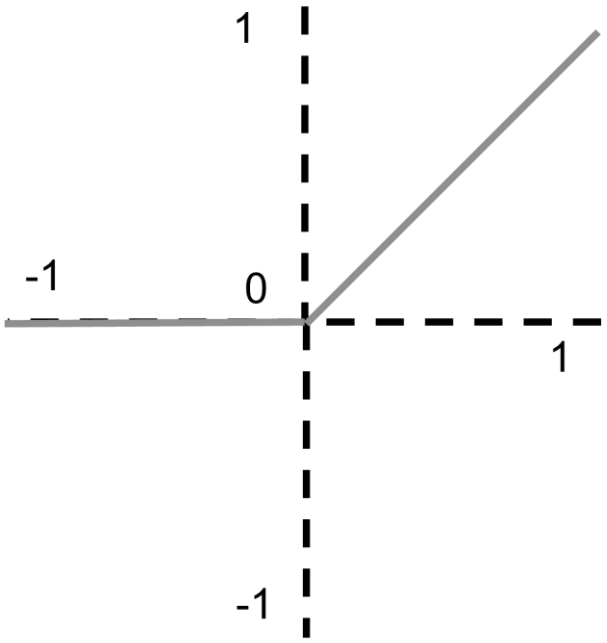

Figure 3.8: ReLU activation function

The Relu activation function shown in Figure 3.8 is the most used activation function in deep learning, with a range from 0 to infinity, as shown in Equation 4.

$$R(z) = \max(0, z) \quad (4)$$

The activation function forces any negative values to become zero. However, sometimes this characteristic decreases the ability of a model to fit the dataset. The Leaky ReLU shown in Figure 3.9 was proposed to counteract this problem.

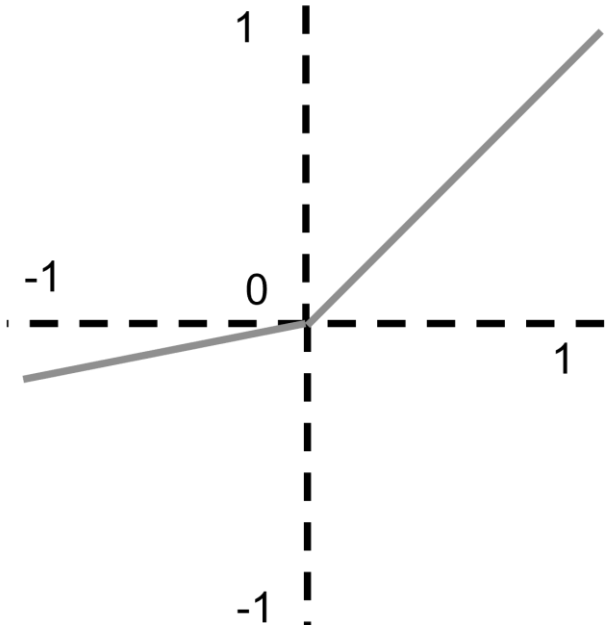

Figure 3.9: Leaky ReLU activation function

The leaky ReLU function allows for a small positive gradient when the unit is not active, as shown in Equation 5.

$$R(z) = \max(0.1 * z, z) \quad (5)$$

The activation layer takes an input volume of feature maps and applies its designated activation function. The output of the activation layer is always the same size as the original input.

3.1.5 Fully Connected Layers

A fully connected layer's job is to connect the input it sees to a desired form of output. In the case of ConvNets, this means converting a matrix of image features into a feature vector whose dimensions are 1 x C where C is the number of classes. As an example, consider using an FC layer to sort images into ten classes. Given a set of pooled and activated feature maps as input, the FC layer uses a combination of these features (multiplying them, adding them, combining them, etc.) to output a 10-item-long feature vector. This vector compresses the information from the feature maps into a single feature vector. An example of a fully connected layer is shown in Figure 3.1. FC layers are typically found at the end on ConvNets and are used to relate the extracted feature maps from the convolutional part of the network to a certain output prediction. Using loss functions, the result of the last fully connected layer backpropagates the error between the true value and the predicted value back along with the entire network.

3.1.6 Loss Functions

Loss functions are used in supervised neural networks to provide algorithms with a method of dealing with deviations in the predicted output from the expected output. There are many different loss functions, all suited to various situations.

The SoftMax function can take any vector of values as input and returns a vector of the same length whose values are all in the range (0, 1) and, together, these values will add up to 1. This function is often seen in classification models that must turn a feature vector into a probability distribution. The SoftMax loss function is also known as Cross-Entropy Loss and is very effective at binary and multiclass classification tasks.

The selection of which layers to use with which loss functions are the building blocks of successful deep learning networks. However, neural networks still need to be optimized because, in most instances, there is no “one size fits all” model. Some of the significant problems experienced by neural networks and the techniques available to optimize them are discussed in the Section 3.1.6.

3.1.7 Vanishing and Exploding Gradients

Backpropagation is the technique used in multi-layer neural networks to recursively calculate the contribution of each weight in every layer of the network to the total error calculated by the loss function in the output layer of the network. As shown in Figure 3.10, backpropagation uses partial derivatives to calculate the associated error with the node under consideration.

In larger neural networks with more than one hidden layer, the error associated with the first nodes in the network is calculated with respect to all the other nodes later in the network that feed it. This is accomplished by multiplying these partial derivatives with one another. The use of activation functions on every neuron saturates the output values to within certain ranges. Sigmoid activation functions, for example, saturate the outputs to a value between 1 and -1. Calculating the error using the dot product of partial derivatives with values within that range eventually results in extremely small error gradients. This is called the vanishing gradient problem.

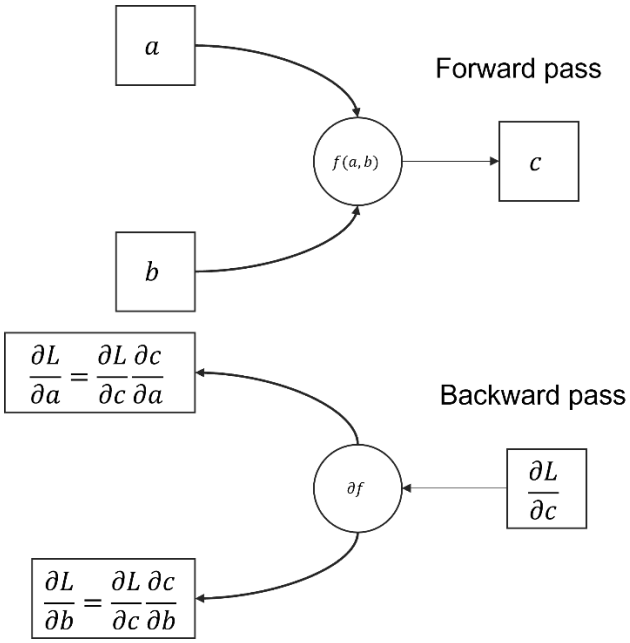

Figure 3.10: Principles of forward pass vs backward pass in neural networks

ConvNets typically use ReLU or Leaky ReLU activation functions to counteract the vanishing gradient problem. However, these functions still suffer from a similar issue as, in some cases,

these activation functions result in some neurons dying out, and the output of the neuron stays 0. Leaky ReLU negative values are saturated between 0 and -1, therefore making them susceptible to the vanishing gradient problem.

The exploding gradient problem is exactly the opposite of the vanishing gradient problem. Activation functions such as ReLU and Leaky ReLU have a range up to infinity. Magnitudes of the gradients calculated during training have the potential to become unstable, as large numbers multiplied with large numbers result in significantly large numbers. This leads to poor prediction results and can sometimes render a model useless.

Algorithmic methods such as batch normalization proposed by Ioffe and Szegedy (2015) apply the mean and variance of the current training batch to the output or input of a neuron's activation function (Ioffe and Szegedy, 2015). The effect of using batch normalization is that networks perform faster and converge better. Batch normalization layers also allow for much higher learning rates without affecting the convergence of the network in training. Batch normalization allows deep, traditionally unstable networks to be trainable.

Another vital issue supervised neural networks are faced with is the overfitting or underfitting of the training data. This problem is discussed in Chapter 3.1.8.

3.1.8 Overfitting and underfitting

Using machine learning for classification tasks aims to create and train models that can generalize well on unseen data of the same class as its training data. Many factors influence how well a model performs, most of which are due to the architecture of the model, the hyperparameters selected, or the data used to train the model. An overview of how these characteristics affect the performance of a model is shown in Figure 3.11.

Models that train for too long or have an overly complex architecture for the task at hand tend to start learning irrelevant information. Noise, overly complex, and incorrectly labelled data in the training set can also cause a model to overfit the dataset. This means the model loses its ability to generalize on unseen samples as it has lost understanding of the task and instead tries to memorize samples from the training data. A test for overfitting models is to

use validation sets within the training process. If the model tends to perform well whilst training but underperforms on the unseen validation set, then the model in question is deemed to have overfitted the training data.

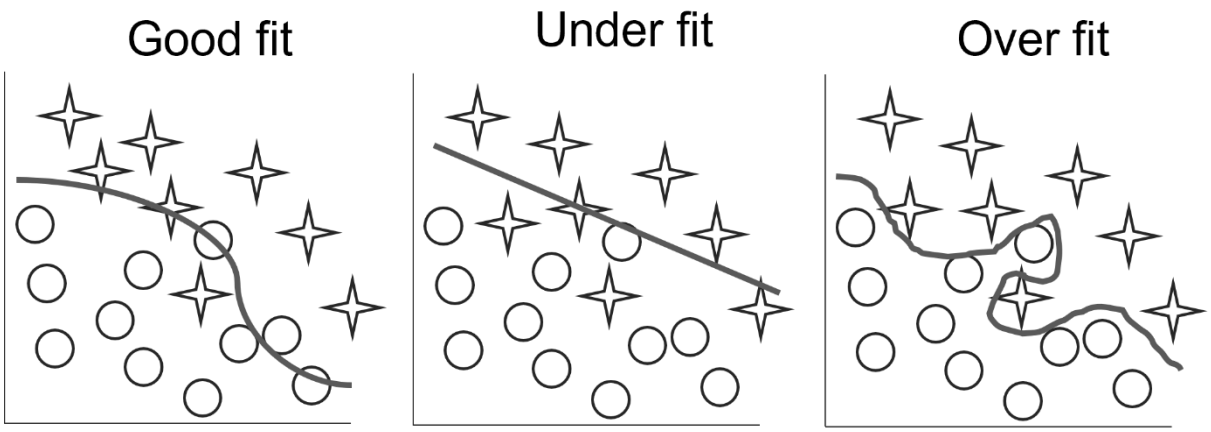

Figure 3.11: Good fit vs over and underfitting

Reducing the training time and architecture complexity are the go-to methods for preventing overfitting. However, if the training is stopped too early or the network is not complex enough to retrieve a deep understanding of the data, the model is susceptible to underfitting the data. Underfitting also results in a model performing poorly when classifying unseen samples from the dataset.

There are methods used to prevent overfitting and underfitting in the training process of deep neural networks. These methods include dropout, reduced data complexity, data augmentation, image transforms and ensembling networks. The concept of dropout is discussed in Section 3.1.9.

3.1.9 Dropout

Dropout randomly turns off perceptrons that make up the layers of a network, with some specified probability. It may seem counterintuitive to throw away a connection in a neural network, but as the network trains, some nodes can dominate others or end up making big mistakes. Dropout provides a way to balance our network so that every node works equally towards the same goal, and if one makes a mistake, it won't dominate the behaviour of the

model. Dropout can be seen as a technique that creates network resilience; it makes all the nodes work well as a team by ensuring no node is too weak or too strong. An example of dropout is shown in Figure 3.12.

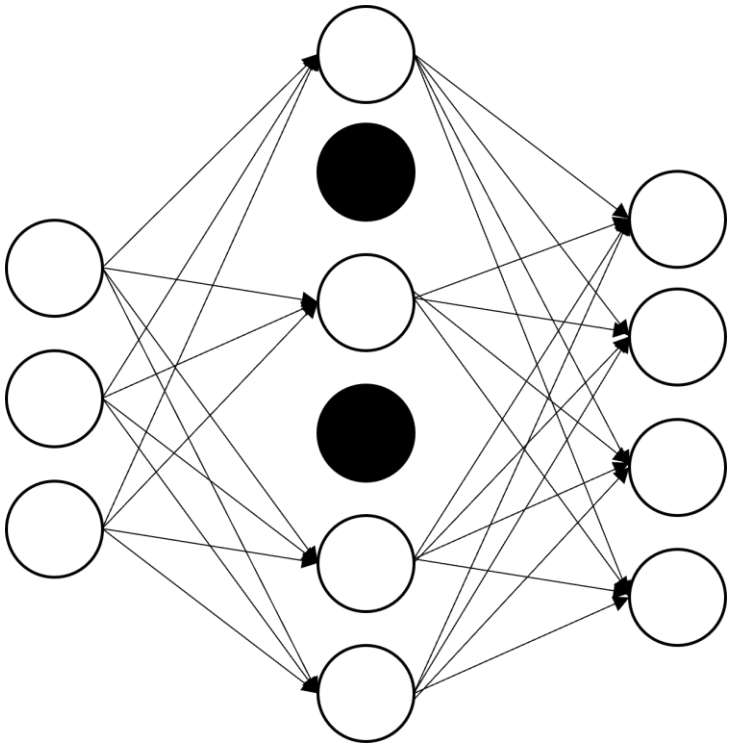

Figure 3.12: Example of dropout on an FC layer

Dropout is an effective way to reduce the ConvNet's ability to overfit the data, thus increasing the network's likelihood of performing better when presented with unseen data. Another method for improving convolutional neural networks' performance is by analysing and transforming the data used to train the network.

3.1.10 Image Transforms and Augmentations

The importance of the data used to train a deep neural network cannot be understated and is usually the difference between excellent models and poor ones. One way of boosting the performance of a neural network is to increase the exposure of the network to more variations of a certain class. When adding more samples to the dataset is unfeasible, researchers use artificially created images with variances from the original image in the

dataset (Yaroslavsky, 2014). This is called image transformation and image augmentation. Training networks with more images that contain different variations produce models that are more robust in identifying specific classes. Some of the techniques used to accomplish this are shown in Figure 3.13.

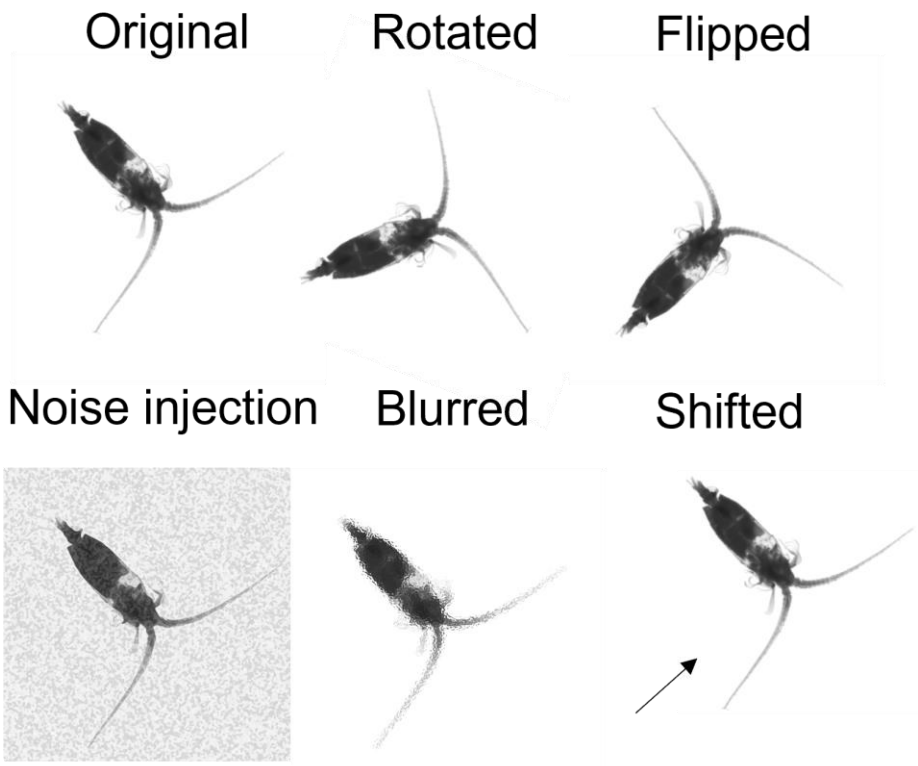

Figure 3.13: Various image transformations performed on a plankton sample

Image rotation is a common technique used to augment images. The benefit of applying this sort of transform onto a dataset is that the model learns to identify certain classes regardless of their orientation. A model that achieves this feature is seen to be rotationally invariant (Chidester *et al.*, 2019). Other transforms such as flipping also increase the rotational invariance of the network. Another geometric transform typically used is known as a shifting transform. This transform changes the position of where the object of interest appears in the image and trains networks to be shift-invariant, thus increasing the robustness of the resulting model(Chen *et al.*, 2019).

Images collected from real-life sources are of different quality and resolutions. Training networks to identify the objects of interest and classify them in various quality samples increases the model's generalisation ability. Noise injection is a transform applied to images where samples encourage the model to learn how to separate the region of interest from the injected noise. The blurring transform is done by applying various filters onto the original image resulting in images with smoothed pixel values (Hua *et al*., 2006).

Applying multiple image transforms onto a single sample image essentially converts it into a new image. New images can be used to increase the size of datasets and are especially useful in situations where a dataset has unbalanced classes. An unbalanced dataset is one where some classes occur in abundance while other classes have fewer samples. Transforms not mentioned in this section include resizing, cropping, and stretching. ConvNets use these to standardize the inputs to the networks to a consistent shape and size.

Image transforms create new images from sample images and are used to increase the robustness and generalization abilities of the neural networks trained on them. Another technique used to mitigate the effects of overfitting and underfitting a given dataset is called ensembling. This method involves combing multiple simpler networks to achieve a larger goal and is discussed in the Section 3.1.11.

3.1.11 Ensemble Networks

Ensembling is a method that combines more than one machine learning algorithm to obtain a collective understanding of the underlying features within the data. Figure 3.14 shows an overview of how most ensemble networks work. There are many different techniques used

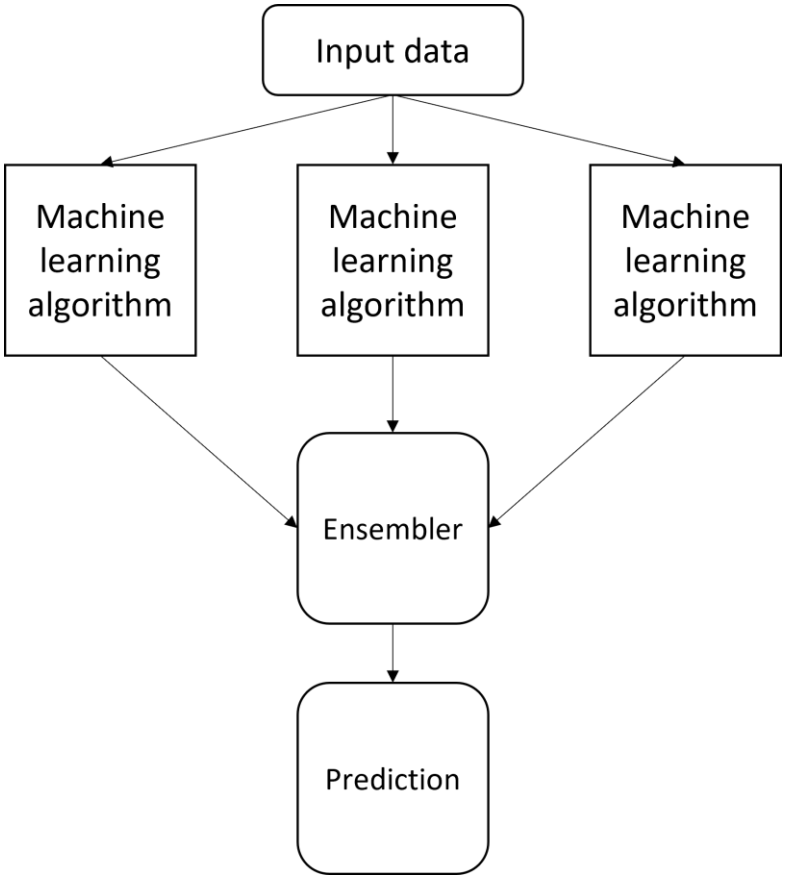

Figure 3.14: Overview of general ensemble algorithm

to ensemble learning algorithms together, and for classification tasks these techniques include bagging, boosting, and stacking.

Bagging or bootstrap aggregating is the same method used in random forest algorithms discussed in Section 2.4.2. Each homogenous model in the ensemble is trained on a different subset of the data to encourage variance in what the algorithms learn. The output of each algorithm is then used to vote on the final output of the ensemble (Zhou, Huang and Wang, 2019).

Boosting has been shown to produce better results than bagging. Boosting is an ensemble method that enables stronger machine learning algorithms using by combining the results of

weaker ones. This is achieved by combining the weak classification rules determined by predecessor algorithms, as shown in Figure 3.15.

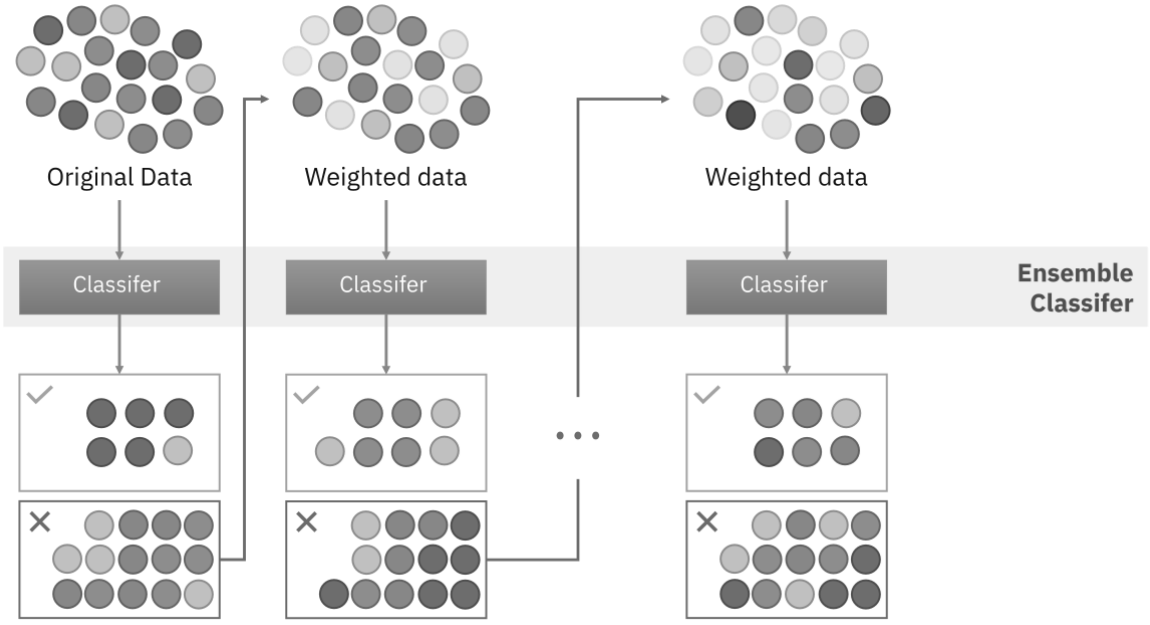

Figure 3.15: Boosting algorithm (File:Ensemble Boosting.svg - Wikimedia Commons, 2020)

Another ensembling method is stacking, where several weaker heterogeneous machine learning models feed into one meta-model. This meta-model then returns a prediction based on the results from the several weaker models.

Ensembling different types of models allow for aggregated decision-making resulting in more confident predictions. More traditional ensembles combine engineered feature extractors, whereas more modern techniques use different CNN architectures to extract diverse features from the dataset. Chapter 3.2 dives into some of the state-of-the-art CNN architectures available.

3.2 CNN Network Architectures

This section provides insight into some of the most used and well-constructed CNN architectures. It also highlights some of the key techniques discovered during the inception of the relevant architecture. The first of the CNN algorithms of great significance developed was AlexNet.

3.2.1 AlexNet

AlexNet is a deep learning architecture consisting of five convolutional layers and three FC layers with varying sizes of convolutional kernel filters. It makes use of Rectified Linear Unit (ReLU) activation functions which are designed to train neural networks faster and do not suffer from the vanishing gradient problem. AlexNet introduced Local Response Normalization and makes use of Dropout, whereby every epoch, a random number of neurons are not allowed to fire. This reduces overfitting and local minima convergence (AlexNet: The Architecture that Challenged CNNs - Towards Data Science, 2018).

3.2.2 VGG

VGGNet shows many similarities to AlexNet. The one major difference is that VGG uses much smaller and fixed-sized kernel filters. Stacking these smaller filters increases network depth, allowing the algorithm to learn more complex features. Smaller kernel size also increases the performance of the network. VGGNet makes use of convolutional kernels of size 3 x 3 with a stride of one and uses max pooling kernels of size 2 x 2 with a stride of two. There are many variants of VGGNet, including VGG16 and VGG19, which indicate the total number of layers in the network.

3.2.3 GoogLeNet

GoogLeNet builds on the previous VGG16 network. However, GoogLeNet has an inception module that removes redundant activations within the network. This improves the efficiency of the algorithm without affecting the result at all (Szegedy *et al*., 2015). Inception modules are used in CNNs to minimize computational expense and to add width but not depth to networks. Inception module designs are generally based on the specific objective at hand. Inception modules, as shown in the GoogLeNet inception module example shown in Figure 3.16, consist of four layers on the same level, namely (1x1, 3x3, 5x5 convolutional layers and a 3x3 max pooling layer).

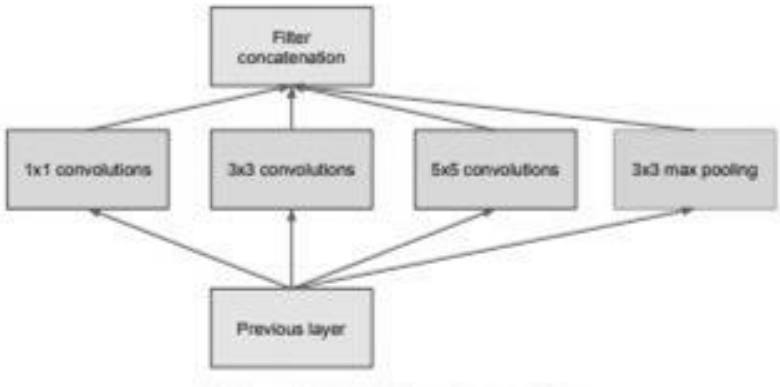

Figure 3.16: Basic inception module (Szegedy *et al*., 2014)

Adding dimension reduction, as shown in Figure 3.17, reduces the computational expense by using smaller 1x1 convolutional filters, increasing the model's speed.

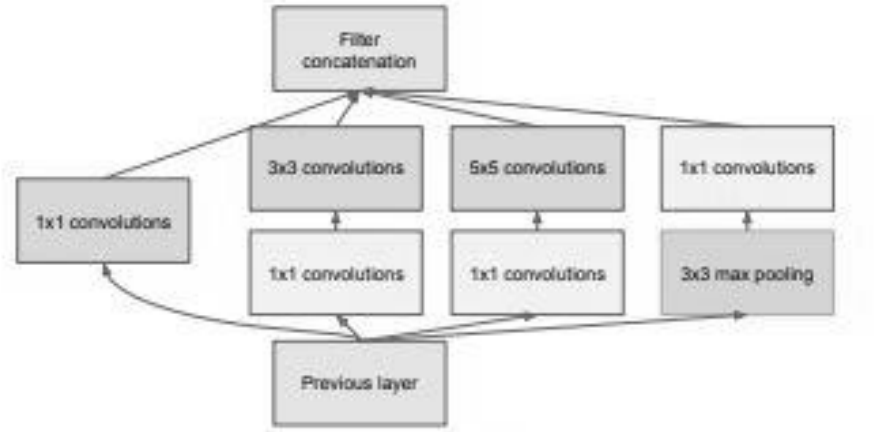

Figure 3.17: Inception module with dimension reduction (Szegedy *et al*., 2014)

GoogLeNet has 9 of these inception modules stacked onto one another. The architecture is shown in Figure 3.18. The network is 27 layers deep which leaves it susceptible to the vanishing gradient problem. To address this issue, the network introduced an auxiliary loss function used in conjunction with the real loss function. The new loss function is a weighted sum of the auxiliary loss function and the real loss function.

type	patch size/ stride	output size	depth
convolution	7×7/2	112×112×64	1
max pool	3×3/2	56×56×64	0
convolution	3×3/1	56×56×192	2
max pool	3×3/2	28×28×192	0
inception (3a)		28×28×256	2
inception (3b)		28×28×480	2
max pool	3×3/2	14×14×480	0
inception (4a)		14×14×512	2
inception (4b)		14×14×512	2
inception (4c)		14×14×512	2
inception (4d)		14×14×528	2
inception (4e)		14×14×832	2
max pool	3×3/2	7×7×832	0
inception (5a)		7×7×832	2
inception (5b)		7×7×1024	2
avg pool	7×7/1	1×1×1024	0
dropout (40%)		1×1×1024	0
linear		1×1×1000	1
softmax		1×1×1000	0

Figure 3.18: GoogLeNet architecture (Szegedy, Liu, Sermanet, *et al*., 2014)

Newer implementations of the inception module, Inception v2 and v3, introduce convolution factorization, which factorizes down convolutional filers of larger sizes into smaller, less computationally expensive convolutional filters.

3.2.4 ResNet

Advancements in developing deep and converging neural networks have exposed another key issue called the degradation problem, where increasing the depth of a network no longer increases but decreases the accuracy of the network. Residual Networks (Resnet) address this by introducing Identity Mapping by Shortcuts whereby residual learning is applied every few stacks of layers. This is shown in Figure 3.19 and Equation 6 (He *et al.*,2014).

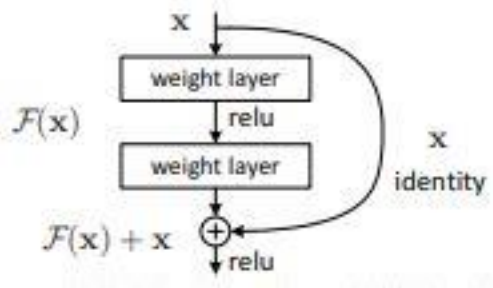

Figure 3.19: Residual learning (He *et al.*, 2015)

Sometimes x and F(x) will not have the same dimension due to convolutional operations and thus, identity mapping is multiplied by a linear projection W to match the residual.

$$y = \mathrm{F}(x, \{W_i\}) + W_s x \qquad (6)$$

This allows for an input x and the resultant F(x) to be combined and passed onto the next stage of the network. The comparison in Tables 1 and 2 compare results attained by different types of research in the field of phytoplankton classification, indicating which CNN architectures performed the best and the number of different phytoplankton classes the algorithm was trained on. Resnet outperformed other algorithms in most cases, indicating that this architecture could be better suited to phytoplankton classification datasets.

3.2.5 DenseNet

Densely Connected Convolutional Neural Network (DenseNet) works similarly to the ResNet architecture explained in Section 3.2.4 by utilizing shortcut connections to pass on potentially important information to the proceeding layers. However, in DenseNet, every layer is connected to every proceeding layer. The problem with connecting one layer to all the proceeding layers is that the feature maps are usually downsampled throughout the various layers to reduce the dimensionality of the network. Therefore, feature maps from all layers feeding each other need to be of a consistent size. DenseNet makes use of dense blocks shown in Figure 3.20, where all the layers in the block are connected to every other layer proceeding it in the block. This allows for the size of the feature maps to be kept constant as no downsampling is applied within the block (Huang *et al*., 2016).

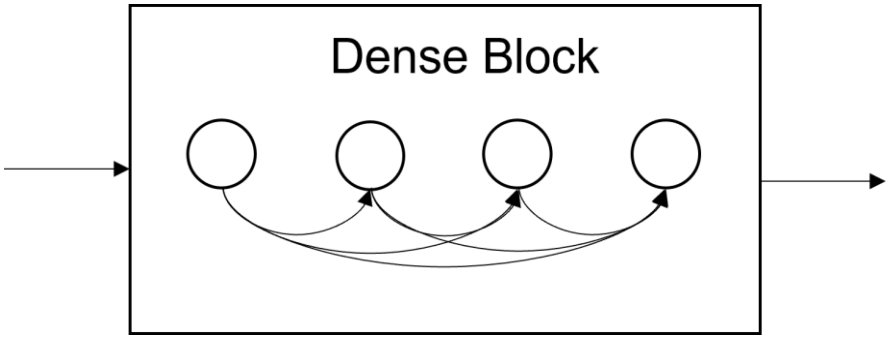

Figure 3.20: Dense block

Figure 3.21 shows a three dense block dense net and how pooling is performed between the dense blocks in layers known as transition layers.

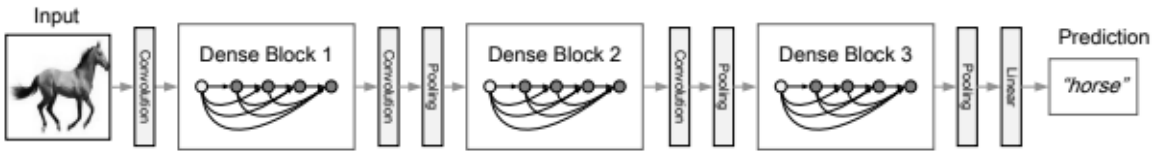

Figure 3.21: Overview of DenseNet architecture

3.3 CNN as Classifiers and Feature Extractors

ConvNets have been shown in numerous studies to outperform manual feature extraction techniques for image classification. This is due to the ability of ConvNets to automatically generate invariant features using convolutions on images and filters. It removes the tediousness and the influence of human bias from the feature extraction process and relies purely on the data at hand. A key feature of ConvNets is that they utilize parameter sharing. This means that filters generated in earlier layers directly or indirectly influence the filters generated in the proceeding layers, reducing the network's dimensionality (Guérin *et al*., 2021). This is illustrated in Figure 3.22, where the deeper than network goes, the smaller the feature maps become.

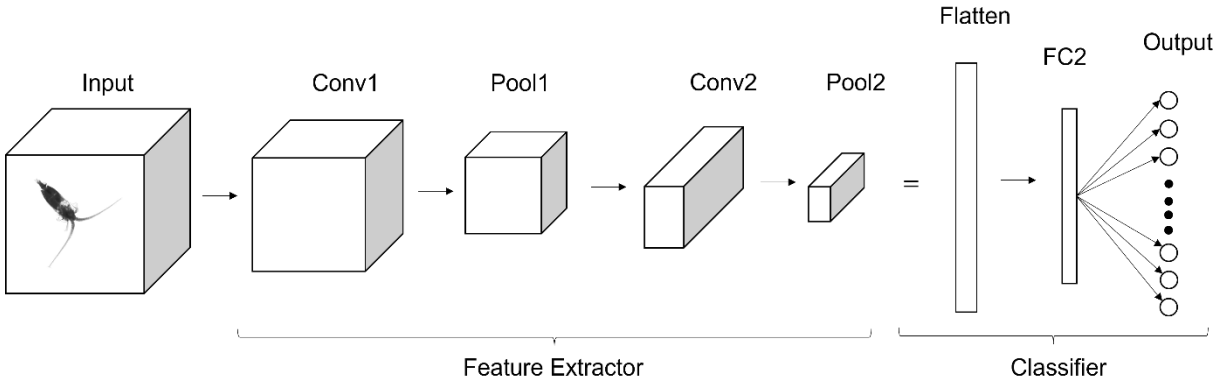

Figure 3.22: Basic CNN architecture

A CNN can ultimately be broken up into two parts, as shown in Figure 3.22. The first part is known as the feature extraction network. It is made up of all the convolutional, pooling, activation, and batch normalization layers. The second part of the network is known as the classifier, which usually houses the FC layers. In classification tasks, the last layer of the classifier network is used to indicate the network's class prediction based on the outputs of all the other layers that come before it.

In most neural network architectures, as discussed in Section 3.2, the classifier layer is only made up of a single fully connected layer to lower the number of trainable parameters, thus reducing the training time of the feature extraction part of the network. However, once the network is trained, it is common for researchers to include more fully connected layers to the classification network to increase the depth of understanding of the outputs. This is done by retraining the network with the new FC layers while placing the feature extraction network into inference mode, where no backpropagation takes place. This process is known as fine-tuning.

After a network has been fine-tuned, the FC layers in the classifier network can be seen as lower-dimensional representations of the features of the entire network. In studies such as the MorphoCluster system discussed in Section 2.5.2, the output layer is removed, and its predecessor FC layer is used as a higher dimensional output feature vector. Once the entire network is trained, fine-tuned, and the output layer removed and placed into inference mode, the entire network acts as a feature extractor which unsupervised algorithms can then use to find inherent structures and grouping within the data. The next part of this chapter discusses these unsupervised methods.

3.4 Unsupervised Learning

In supervised learning, models are trained to find a mapping function between an input and an expected output. In cases where labelled data is limited, or there is uncertainty in the dataset, supervised methods are not viable. Unsupervised models provide a way to solve this

issue. Rather than explicitly being told what to learn, the algorithm find patterns within the input data itself (Lehr et al., 2021). This section highlights some of the key unsupervised algorithms used today. Section 3.4.1 introduces an unsupervised dimension reduction technique called principal component analysis (PCA). Section 3.4.2 discusses another branch of unsupervised algorithms called clustering algorithms as well as various clustering methodologies

In the final part of the chapter the understanding of these clustering methodologies is brought together to better understand the underlying working of HDBSCAN (Hierarchical Density-Based Clustering for Applications with Noise).

3.4.1 Dimension Reduction

Mathematician R. Bellman introduced the term “the curse of dimensionality” (Bittner, 1962), which proposed that for arbitrary functions to achieve high levels of accuracy, more features are required. This increase of features exponentially increases the potential for associated error. Dimension reduction techniques are used to mitigate this need for higher dimensionality by reducing the number of features required by an arbitrary function without losing the relative accuracy (Weng and Young, 2017). A common method used for dimension reduction is known as principal component analysis (PCA).

3.4.1.i Principle Component Analysis

Principle Component Analysis (PCA) is a linear dimension reduction method and seeks to extract features from a dataset. It involves reducing the dimensions of a dataset to its principal components, as shown in Figure 3.23.

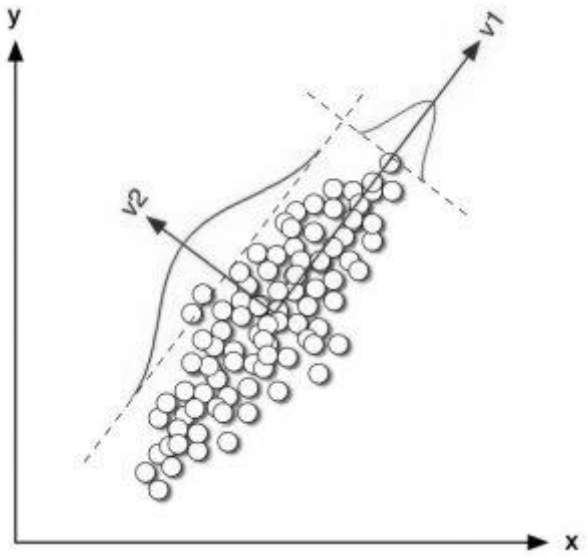

Figure 3.23: Diagram showing dimension reduction using PCA (Kwak, 2008)

PCA involves reducing the dimensions of the dataset while retaining as much information about the dataset as possible. The following equations show how PCA can be used in linear dimension reduction.

Let:

$$\boldsymbol{x} \in R^D \qquad (7)$$

Where:

x is a vector in a high dimensional space D

The goal of PCA is to reduce this vector x to lower dimensional space. There we want to project it to vector z, where:

$$\boldsymbol{z} \in R^M \text{ (M << D)} \qquad (8)$$

Because PCA is a method of linear dimension reduction, we use a linear transform such that:

$$\boldsymbol{z} = U^T \boldsymbol{x} \qquad U \in R^{D \times M} \tag{9}$$

Where on some input x we want to reduce it to some output z in a reduced dimensionality space. This is, however, for one point only. Consider matrices X and Z composed of these points where:

$$\mathrm{X} = [\mathbf{x_1}\ \mathbf{x_2}\ \mathbf{x_3} \ldots\ \mathbf{x_N}]^{\mathrm{T}} \quad X \in R^{N \times D} \tag{10}$$

$$\mathrm{Z} = [\boldsymbol{z_1}\ \mathbf{z_2}\ \mathbf{z_3} \ldots\ \mathbf{z_N}]^{\mathrm{T}} \quad Z \in R^{N \times M} \tag{11}$$

$$Z = UX \tag{9}$$

Information of the dataset is represented by the covariant matrix S_Z where:

$$S_Z = \frac{1}{N}(Z^T Z) \qquad S_Z \in R^{M \times M} \tag{10}$$

Because the goal is to minimize the number of dimensions while maintaining the information, optimization can therefore be performed:

$$\begin{aligned} &\max_U S_Z \\ &= \max_U \frac{1}{N} Z^T Z \\ &= \max_U \frac{1}{N} (XU)^T (XU) \\ &= \max_U \frac{1}{N} U^T X^T X U \\ &= \max_U U^T S_X U \end{aligned} \tag{11}$$

This maximization, however, has no upper bound on U, adding a condition that every vector in the matrix has a unit vector magnitude:

$$U^T U = I \tag{12}$$

Lagrange multipliers can be used to solve for U as there exists an optimization problem with equality constraints where:

$$\max_x f(x) \tag{13}$$

Where f is the function, we are trying to minimize with respect to a variable x under a set of constraints:

$$g_i(x) = 0 \tag{14}$$

The new set of variables, called the Lagrangian multipliers, is a set of λ_i where:

$$\begin{gathered} L(x, \{\lambda_i\}) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) \\ \lambda_i \geq 0 \\ \frac{dL}{dx} = 0 \\ x = h(\{\lambda_i\} \end{gathered} \tag{15}$$

The original optimization can be rewritten in terms of the Lagrangian Optimization as:

$$\begin{gathered} L(U, \lambda) = U^T S_x U + \lambda(I - U^T U) \\ \frac{dL}{dU} = 0 \end{gathered} \tag{16}$$

$$S_X U = \lambda U$$

Because of matrix equality, both sides of the equation are equal and can be rewritten as a set of equations:

$$S_X u_i = \lambda_i u_i \quad (17)$$

This structure represents that of an eigenvector equation set on which eigen decomposition is applied and Matrix Diagonalization determines the eigenvectors. This results in a matrix of eigenvectors which are then sorted in descending order of eigenvalues. Only the top M pairs of eigenvectors are selected as this is the smallest set of the eigenvectors that can retain the most information about the original data. Each of these eigenvectors corresponds to u_i and therefore the matrix U = [u1 u2 u3 … um] ^{T} is a matrix of the top M eigenvectors of S_x. The transformation is then:

$$z = U^T x \quad (18)$$

(Jolliffe and Cadima, 2016)

3.4.2 Clustering

Clustering algorithms are a form of unsupervised learning that seek to leverage the patterns within the data manifolds to uncover groups of data points that share similarities. In other words, data points in the same group should show highly similar properties, whilst data points in other groups should exhibit very dissimilar properties. An example of desirable clustering is shown in Figure 3.24.

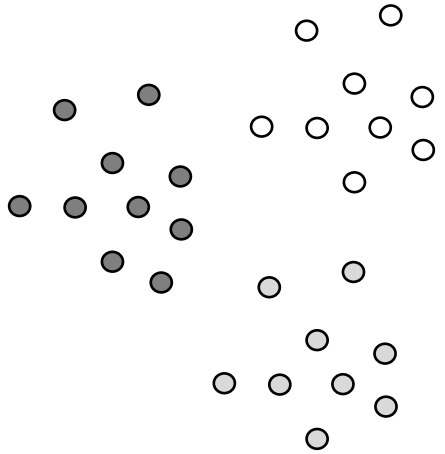

Figure 3.24: Example of ideal clustering

Clustering algorithms can provide deeper insight into the features of the dataset under analysis. Certain clustering algorithms are better suited to certain datasets than others. The clustering algorithms discussed in this section form part of two groups, namely Partial Clustering and Hierarchical Clustering, shown in Figure 3.25. K-Means clustering is the most popular clustering algorithm for more clearly separated manifolds such as the example shown in Figure 3.24.

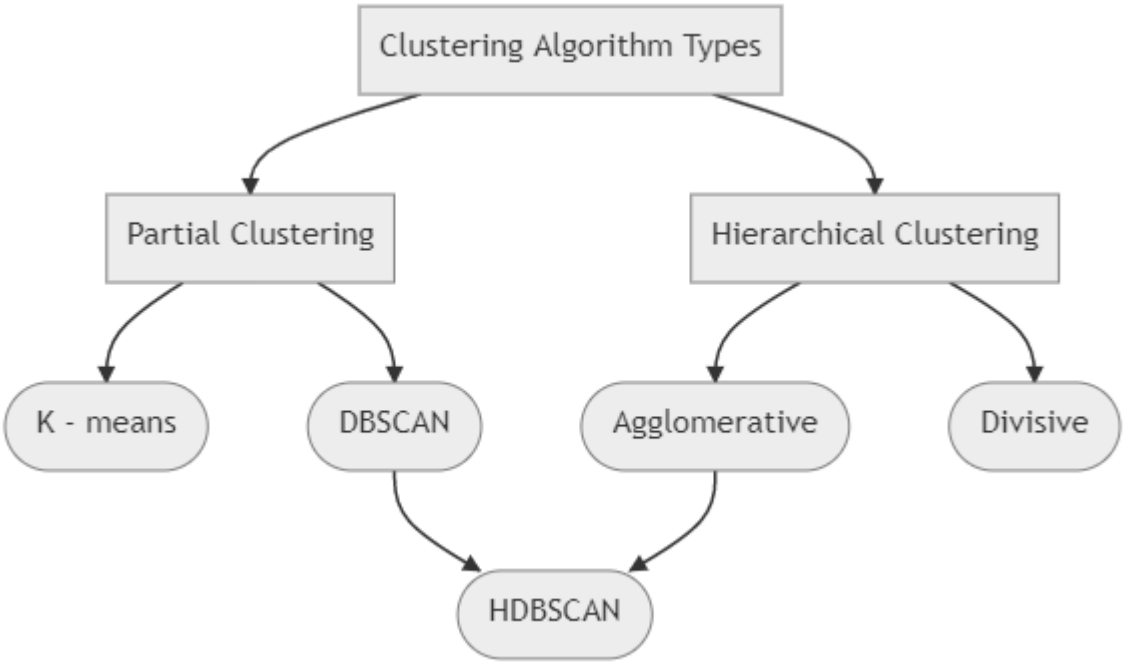

Figure 3.25: Considered clustering algorithms

3.4.2.i K-Means

K-Means clustering is a method originally used for compression in audio signal processing known as vector quantization. The algorithm aims to place samples into groups, where each sample belongs to the group based on the distance between the sample points and the cluster centroids. There are many variations of K-Means. However, they all stem from naïve K-Means. The most important part of all K-Means algorithms is how they start initializing the clusters. This can be done in one of three ways: Forgy Initialization, Random Partition, and the K-Means ++ method.

Forgy is a fast Initialization method for K-Means where, if there are N number of desired clusters, the method randomly selects N points from the dataset and uses their value as the initial cluster means. The K-Means algorithm then assigns each data point to a cluster. If each designated cluster centroid does not have at least one data point assigned to it then Forgy initialization is applied again, this time picking new random points as cluster means. This process is repeated until every cluster has at least one assigned data point (Peña, Lozano and Larrañaga, 1999).

In the Random Partition method, every data point is assigned to a random cluster. For each cluster, the initial centroid is then determined by taking the average value of points assigned to that cluster. This initialization results in cluster centroid values near the mean value of all the data points (Ahmad and Khan, 2019). Both the Random Partition and Forgy Initialization methods are susceptible to forcing the K-Means algorithm into finding Local Optima. David Arthur and Sergei Vassilvitskii (2007) proposed a different initialization method called K-Means ++, which makes use of careful cluster seeding.

K-Means ++ Initialization starts by selecting a random point from the data then computes the distance of every point in the dataset from the selected point. The next point is then selected from a probability distribution representing the absolute distance of all points. We select the new point such that the distance of the point from the initial centroid is relatively large. The next point is then selected based on its relative distance from a probability distribution representing the distance of every point from the two previously selected points. This process of continuously choosing relatively distant points from one another is repeated until the desired number of clusters has been proposed. K-Means++ is a recommended way of seeding for K-Means clustering and assists the algorithm in converging on the Global Minimum (Arthur and Vassilvitskii, 2007).

A common approach to measuring the distances is using the Euclidean distance between the points and the cluster centroids. The goal of K-Means is to group a set of elements x in a D – dimensional space into K number clusters such that the Sum of Squared Error (SSE) of data points to associated cluster centre (C_i) is minimized, as shown in Equations 19 and 20.

$$\boldsymbol{SSE(C) = \sum_{i=1}^{K} \sum_{x_i \in C_i} (x_i - C_i)^2} \qquad (19)$$

$$\boldsymbol{C_i = \frac{x_i \in C_i}{|C_i|}} \qquad (20)$$

Algorithm 1 shows the pseudocode for the general implementation of K-Means initialized with K-Means ++ method. After initialization of the centroids, the algorithm reassigns all the points to their nearest cluster centroid. Once all the points are assigned to a cluster, the algorithm recalculates the new cluster centroids. The algorithm terminates once the cluster centroid locations between iterations stay the same.

Algorithm 1 – K-Means

```
1   Choose K points
2   Initialize centroids using k -means++ method
3   while previous centroids!= new centroids
4       Assign all points to nearest cluster centroid
5       Calculate new cluster centroids
    end
```

Variations of the algorithm have been implemented. These variations include K-Medoids Clustering, K-Modes Clustering, X-Means Clustering and Kernel K-Means Clustering, all of which seek to optimize the clusters using the same general approach shown in Algorithm 1. Some of these variations of K-Means include automated methods on how the K value is selected, and others use different measures of the distance between points and the cluster centroids.

The performance of the K-Means algorithm and all its variants wanes with increasing data complexity and noise. This decrease in performance can be attributed to many features of the algorithm's rudimental cluster identification methods. Problems include the spherical assumption of the data, outliers skewing data centroid calculation, and hard clustering to a specific K number of clusters. Figure 3.26 shows cluster centroids (indicated by the black star) selected undesirably. The dataset shown has two distinct clusters, but because the number of clusters (K) was selected as 4, and the data doesn’t have a spherical structure, the resultant cluster centroids are placed in undesirable locations.

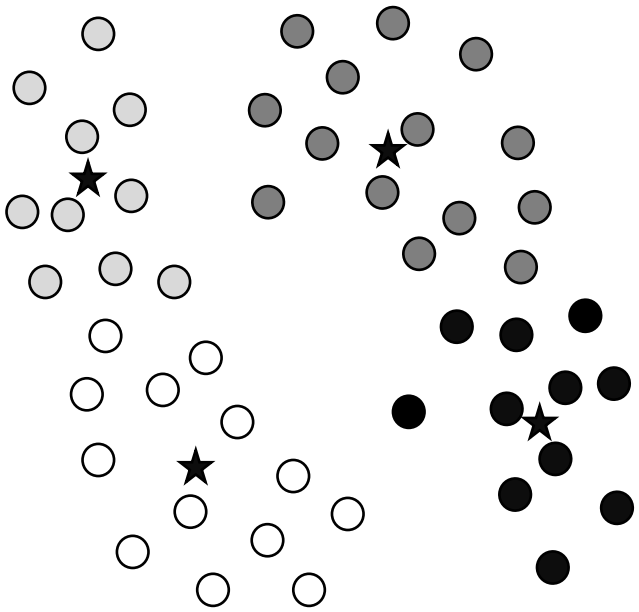

Figure 3.26: Dataset clustered undesirably using K-Means.

The infamous trait that most well-known clustering algorithms have, including K-Means, is that they directly or indirectly assume that the data under consideration is from a particular type of probability distribution. This assumption leads to the clustering algorithms producing spherical groupings, whereas most real-life spatial datasets contain groups of irregular shapes. This phenomenon, coupled with the algorithm's lack of outlier consideration, led to the development of clustering algorithms more suited to spatial data. These algorithms, which include Density-Based and Hierarchical clustering algorithms, are discussed in the following sections.

3.4.2.ii Density-Based Clustering

Density-Based Clustering algorithms such as DENCLUE (Density Based Clustering) and DBSCAN (Density Based Spatial Clustering for Applications with Noise) form part of the group of

clustering algorithms deemed density-based. These algorithms were developed because of the limitations of centroid based clustering techniques such as K-Means. As discussed in the K-Means section, these limitations include the inability to deal with arbitrarily shaped data manifolds and anomalous points.

Density-based algorithms use the ε -neighbourhood, which is defined as the set of all points within a certain distance (ε) from a given point. Figures 3.27 and 3.28 show the effect of varying the ε value in a two-dimensional space. Figure 3.27 shows the ε-neighbourhood of a point P with ε = 0.5, which contains eight other points.

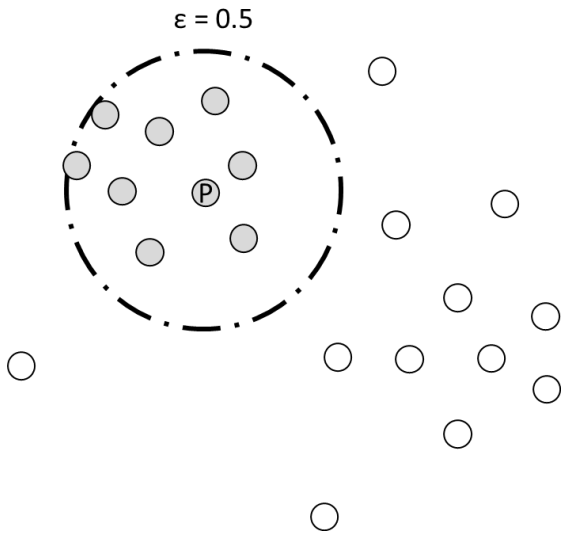

Figure 3.27: ε-neighbourhood with ε=0.5

In the two-dimensional case of point P in Figures 3.27 and 3.28, the volume is defined as the area of the resulting circular neighbourhood shape, and the mass is defined as the number of points inside point P's ε-neighbourhood. The local density approximation of the point P is then defined as this mass divided by this volume.

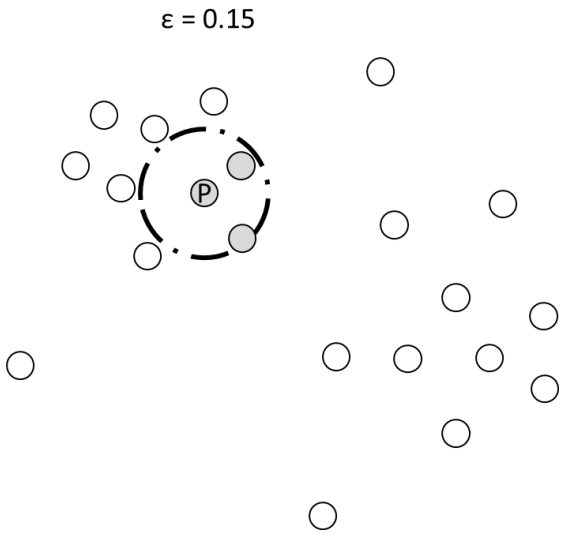

Figure 3.28: ε-neighbourhood with ε=0.15

Suppose the local density approximation is calculated for all the points in the dataset. In that case, points within the same neighbourhood and with like local density approximations could be considered as part of the same cluster. This definition of local density approximation and ε neighbourhood make up the key elements of how all density-based clustering algorithms operate.

3.4.2.iii DBSCAN (Density-Based Clustering for Applications with Noise)

In 1996 Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu (1996) proposed Density-Based Clustering for Applications with Noise (DBSCAN). The algorithm that makes use of non-parametric density-based clustering won the SIGKDD (Special Interest Group on Knowledge Discovery and Data Mining) Test of Time award and is one of the most used clustering algorithms available (Ester *et al*., 1996). The task of DBSCAN is to find clusters in the dataset with respect to two parameters. These parameters are ε, or the ε-neighbourhood, explained in the Density-Based Clustering section, and Min Points, which is the minimum number of points required within the ε-neighbourhood for the point under consideration to be deemed a core point.

DBSCAN introduced two key properties between points: density reachable and density connected. These properties are shown in Figure 3.29, where two points X and Y are considered density reachable as point X is within the ε-neighbourhood of core point Y. Point Z is also considered density reachable from core point Y. The two points X and Z are considered density connected. This is because there is a point Y in which both points X and Z are density reachable. Therefore, a cluster in DBSCAN can be understood as a set of density connected points that is maximal with respect to density reachability (Jungan *et al*., 2018).

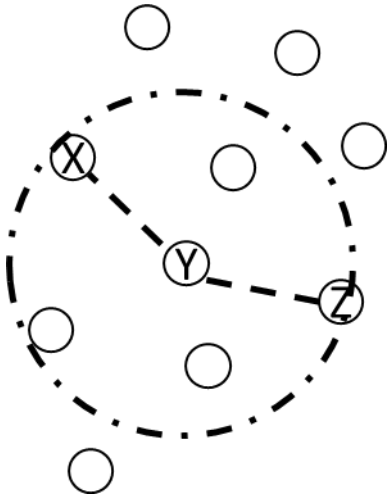

Figure 3.29: Density reachability and connection properties

The DBSCAN algorithm will eventually deem all points within the dataset as one of three types of points illustrated in Figure 3.30. These points are known as core points, border points and noise points. A point is identified as a core point if the number of points within the ε-neighbourhood is greater than the threshold Min Points value. A point is deemed a border point if it is density reachable from a core point but does not have enough points within its ε-neighbourhood to meet the Min Points threshold. Noise points are points that are not associated with any cluster.

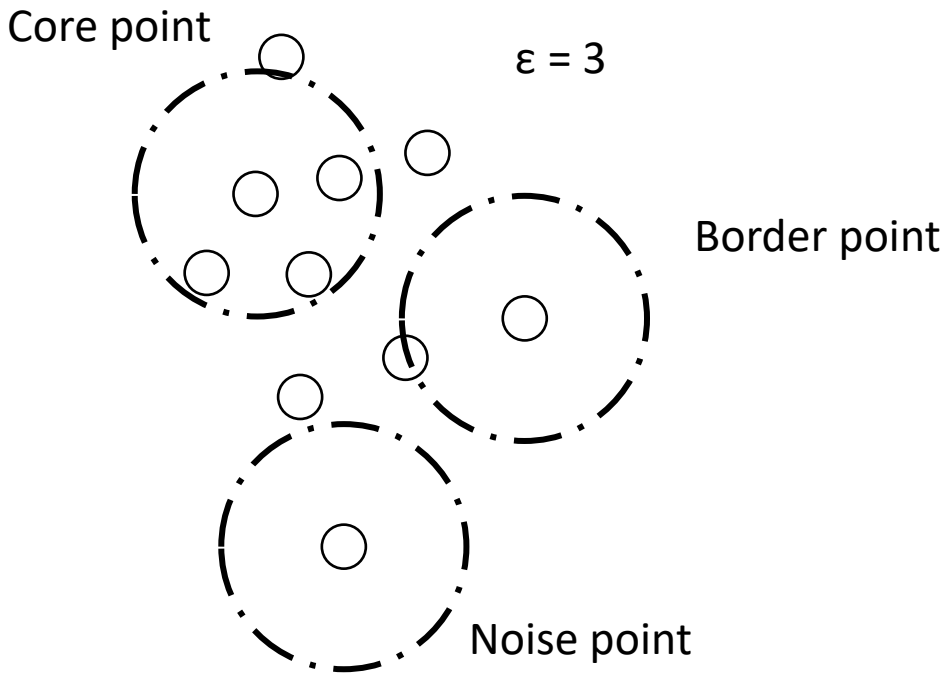

Figure 3.30: Identifying core, border, and noise points

The DBSCAN algorithm, as shown in Algorithm 2, starts by selecting a random point p from the dataset. The algorithm proceeds to identify all points density reachable from p and decides, based on Min_Points, whether point p is a core point. The algorithm then moves through all the density reachable points from point p and determines their point type with respect to ε and Min_Points. If the randomly selected point p is not a core point, the algorithm classifies it as a noise point due to not having any density reachable neighbouring points. The algorithm is complete once all the data points are assigned to either a cluster or a noise point.

Algorithm 2 - DBSCAN

```
1   Choose ε value (ε-neighbourhood size)
2   Choose Min Points (minimum number of points in ε-neighbourhood to be classified
    as a core point)
3   Initialize Set of Identified Clusters (Ci) where (i=0)
4   For each unvisited point p in dataset
5       Select arbitrarily and mark p as visited
6       Identify all points p' density reachable from p (including p itself), such that p' is
        an element of N
7       If number of points in N < Min_Points
8           mark p as a noise point
9       Else
10          Select next cluster Ci, where I = (i + 1)
11          Add p to cluster Ci
12          For each point p' in N
13              If p' unvisited
14                  Mark up as visited
15                  Identify all points p'' density reachable from p' (including p' itself), such
                    that p'' is an element of N'
16                  If number of points in N' >= Min_Points
17                      N = N + N'
18              If p' not contained in any Ci then add p' to current Ci
19  Until all points belong to at least one cluster
```

The DBSCAN algorithm works well for datasets containing desired clusters of arbitrary shapes and noise but has some drawbacks. DBSCAN is non-deterministic, meaning that because of the arbitrary point selection, as shown in line 5 of Algorithm 4, the algorithm can produce

varying outputs even on the same dataset. The algorithm also struggles with differing cluster densities and high dimensionality due to using a constant ε-neighbourhood size.

3.4.2.iv Hierarchical Clustering

Hierarchical Clustering aims to produce a dataset visualized as nested clusters (for example Figure 3.31) or as a hierarchical tree known as a dendrogram (for example Figure 3.32.) The most popular of the hierarchical clustering techniques is agglomerative clustering.

Agglomerative clustering, shown in Algorithm 3, starts by assuming every point as part of a cluster, calculates the proximity between the clusters, and then merges the closest clusters. This process is repeated until only one maximal cluster remains. This one remaining cluster represents the apex of the dendrogram shown in Figure 3.32. There are different types of proximity measures used in agglomerative hierarchical clustering, the two main types being single link and complete link clustering.

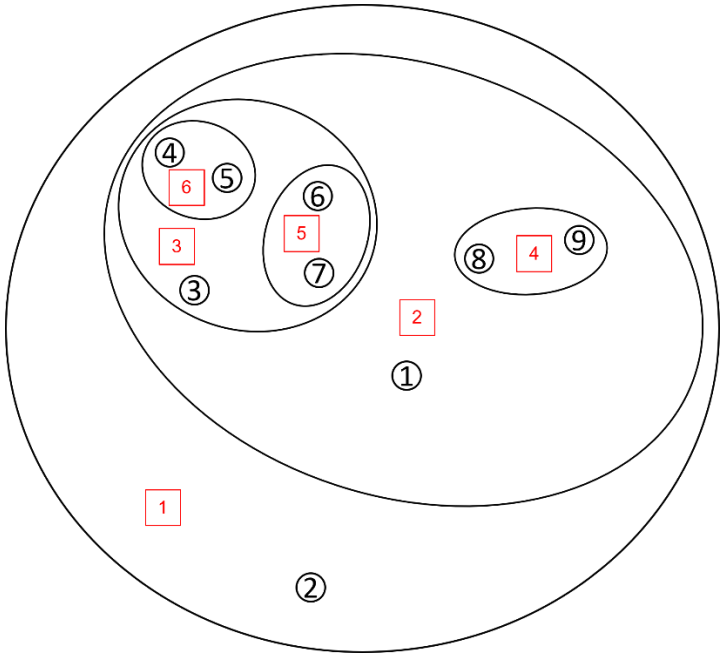

Figure 3.31: Example of resultant clusters using agglomerative clustering relating to the dendrogram in Figure 3.32

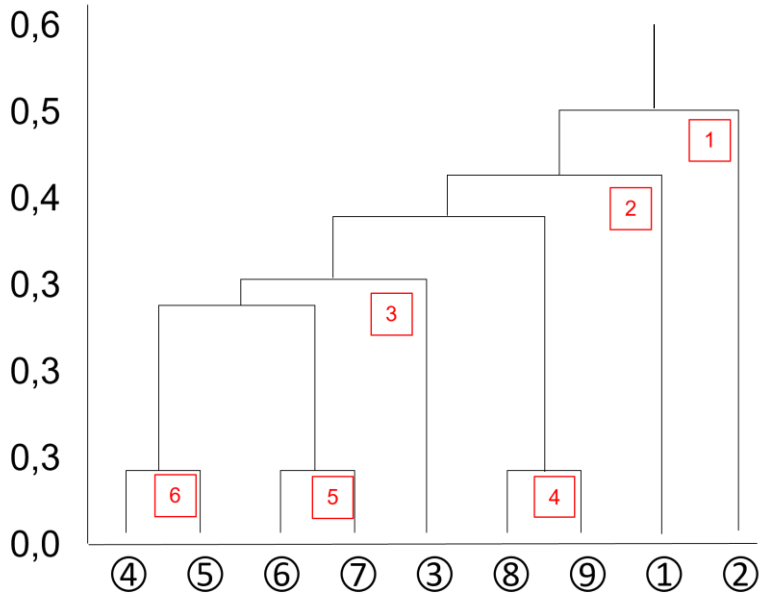

Figure 3.32: Dendrogram produced because of agglomerative clustering the points shown in Figure 3.31

Algorithm 3 – Agglomerative Hierarchical Clustering	
1	Choose proximity criterion
2	Calculate and assign every data point as a cluster
3	**while** the number of clusters > 1
4	Calculate cluster distances using proximity criterion
5	Merge the nearest clusters
	end

Complete link clustering measures the distance between clusters as the distance between the relative clusters' most dissimilar members. This type of clustering results in smaller, more compactly shaped clusters.

Single link clustering is a form of local similarity-based clustering. This type of clustering considers the distance between clusters as the distance between the separate clusters two most similar members. Single link clustering pays more attention to the distances between

members in the separate clusters than the actual structure of the clusters themselves. The advantage of this form of clustering is that it lends itself to arbitrary shaped clusters. However, the algorithm is highly susceptible to noise. The dendrogram shown in Figure 3.32 makes use of single-link clustering.

The performance of hierarchical clustering algorithms such as agglomerative single link clustering suffers from the inability of the algorithm to simultaneously satisfy efficiency and accuracy. This, combined with the disregard for cluster shapes, leads to decreased performance. The K-Means algorithm described earlier in this chapter assumes that the underlying data patterns are in the shape of a sphere. The previous section introduced the DBSCAN algorithm and highlighted that the parameters used to efficiently use the algorithm are difficult to determine. Lv *et al*. (2018) proposed using the concept of a minimum spanning tree (MST) to alleviate the problems experienced by all the classical clustering algorithms. An MST is invariant to changes in the geometric boundaries of the clusters, meaning the shape of clusters have an insignificant impact on the performance of clustering algorithms (Lv *et al*., 2018). The next part of this chapter explains one such method called Prims Algorithm.

3.4.2.v Prims Algorithm

Prim's algorithm efficiently finds the minimum spanning tree (MST) of an undirected and connected graph. An MST is a minimum weight, connected graph with no circles. An example of an MST can be seen in Figure 3.33 (Berenbaum, 1998).

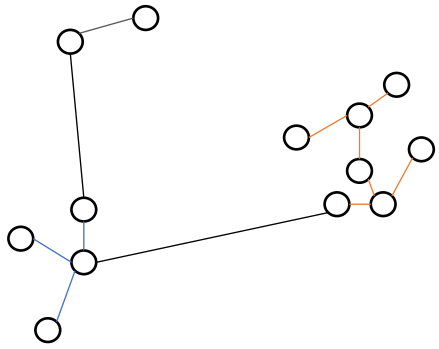

Figure 3.33: Examples of an MST

Prim's algorithm aims to build up an MST from an empty tree while maintaining two sets of nodes. One set includes all the nodes currently in the MST, and the other set contains nodes not yet considered for the MST. The algorithm iteratively picks the minimum weight distance connecting the two sets and includes the endpoint node into the MST set (Marpaung and Arnita, 2020).

Prim's algorithm starts by creating an empty list that keeps track of the nodes encountered by the algorithm. Starting with a random node, the node is added to the list, and all nodes reachable from the starting node are then considered. An example of this step is shown in Figure 3.34.

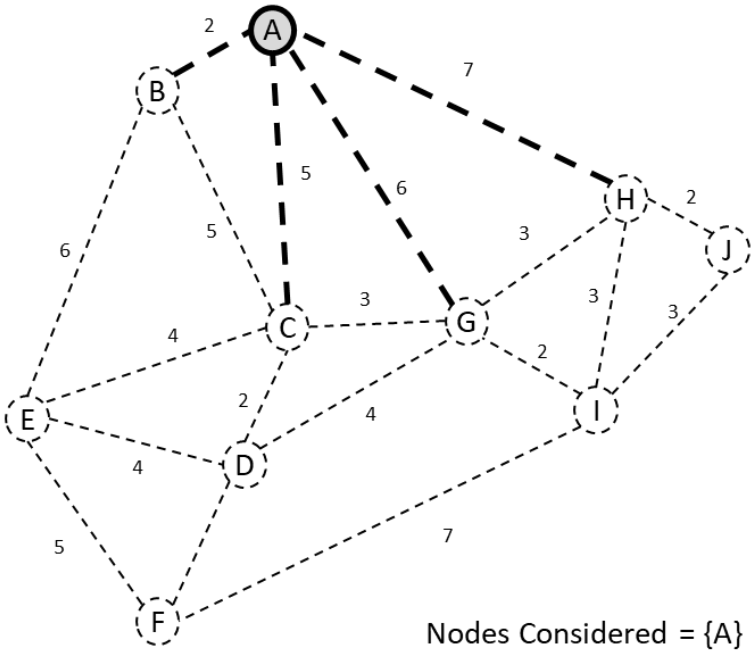

Figure 3.34: Prim's algorithm first step

Prim's algorithm is a greedy algorithm, which means that the algorithm selects the smallest edge connecting to an unvisited node. In the case of the example shown in Figure 3.34, the shortest edge is the one that connects A to B. B is then added to the visited list forming the start of the MST, as shown in Figure 3.35.

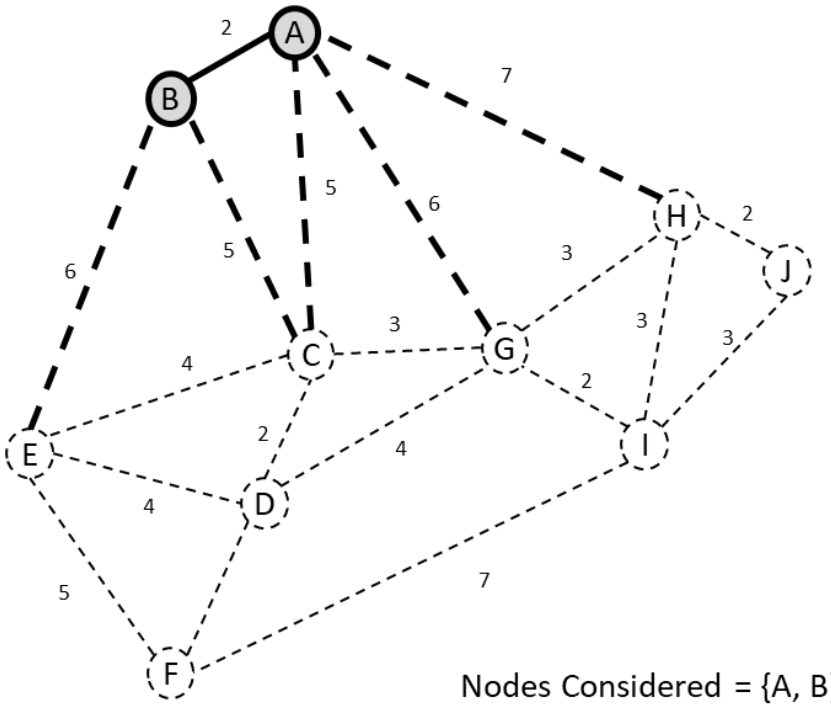

Figure 3.35: Prim's algorithm second step

When two edges have the same length, the algorithm picks one at random. Prim's algorithm continues in this manner until all the nodes have been connected, resulting in the MST shown in Figure 3.36.

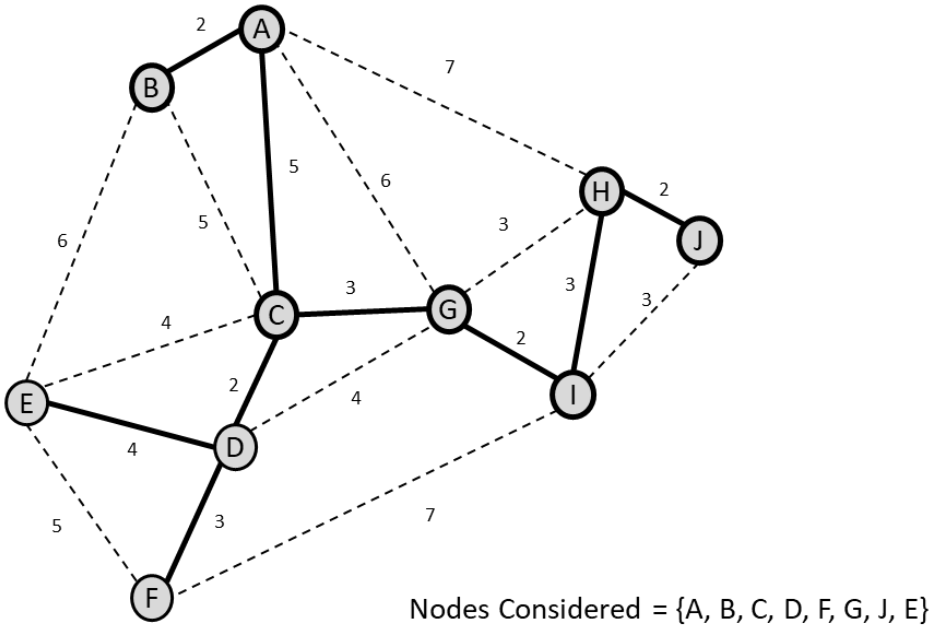

Figure 3.36: Result of Prim's algorithm

Clustering algorithms use minimum spanning trees as they can detect clusters of data with irregular cluster boundaries. They increase the performance of all algorithms which suffer from variations in cluster boundaries. The next section introduces Hierarchical Density-Based Clustering for Applications with Noise (HDBSCAN), which utilizes the MST in combination with DBSCAN and agglomerative single-link clustering.

3.4.2.vi Hierarchical Density-Based Clustering for Applications with Noise (HDBSCAN)

HDBSCAN was first introduced in 2013 by Campello *et al*. (Campello, Moulavi, and Sander, 2013). It combines the ideas of DBSCAN and agglomerative clustering to analyse feature sets by extracting clusters of arbitrary shape, different sizes, varying densities, and noise.

HDBSCAN, just like DBSCAN, transforms the data into density space. The HDBSCAN algorithm then builds an MST using Prim's algorithm and performs single link clustering on the MST to get it into the form of a dendrogram, as shown in Figure 3.37 (b) and (c).

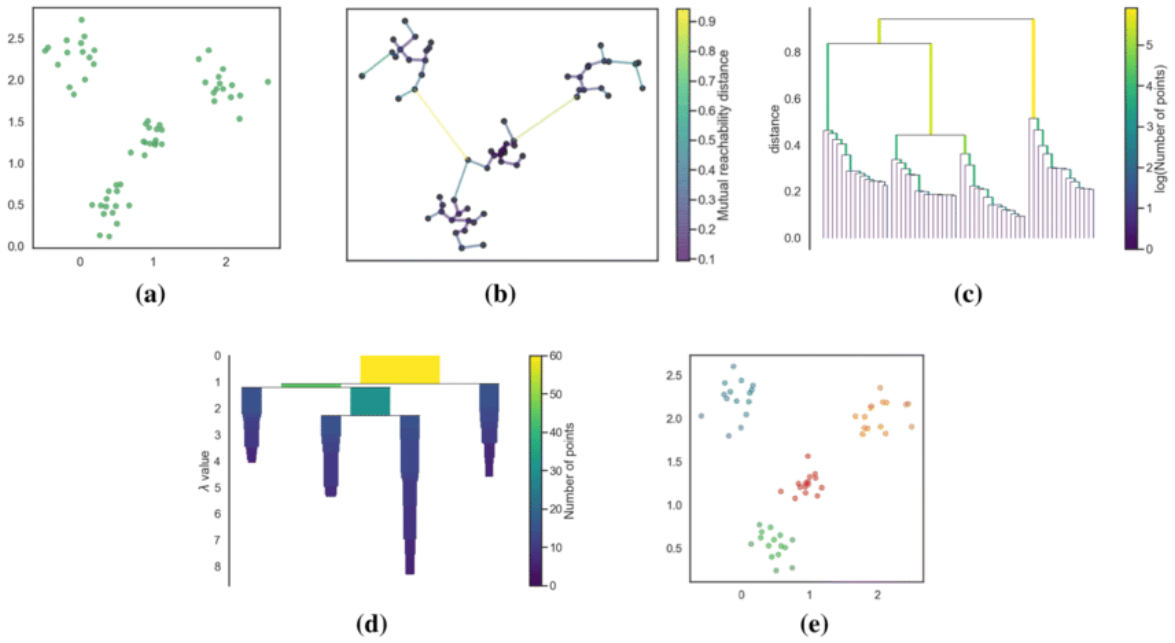

Figure 3.37: HDBSCAN steps (Graphical representation of data output at key stages in the HDBSCAN... | Download Scientific Diagram, 2020*)*

The difference between HDBSCAN and DBSCAN is that instead of using the constant ε-neighbourhood value from the DBSCAN algorithm as the cut value (cluster identifier) for the dendrogram produced by the agglomerative clustering method, another method is used. The dendrogram is pruned by classifying the small groups of points which branch off as falling out of a cluster. This results in a smaller tree representing fewer clusters. In this way, the dendrogram can be utilized to extract the most indefatigable and solid clusters. This can be seen in Figure 3.37 (d). This methodology removes the need for ε-neighbourhood parameters and allows for variable density clusters. The only parameter left from the DBSCAN algorithm is Min_Points, which determine whether points are forming a new cluster or falling out of one.

The studies undertaken in this research use most of the methodologies described in this chapter. The first set of studies make use of ConvNets to classify specific subsets of a broader plankton dataset. These trained ConvNets are then ensembled and used as a novel feature extractor network to feed an HDBSCAN algorithm and extract underlying groupings within the data. Finally, an automated cluster identification model is proposed and tested on the entire dataset. A breakdown of the various investigations is presented in the next chapter, Chapter 4.

4 Experimental Methodology

The literature review conducted in Chapters 2 and 3 provide insight into critical features in the field of computer vision, specifically focused on the classification and identification of plankton taxonomy. The purpose of this chapter is to discuss the investigations undertaken in this research. Section 4.2 provides an overview of the broader dataset created and used during the investigations. All the investigations use different types of classes. The difference between these class types is also discussed. All the investigations make use of the same image processing techniques to augment and transform the dataset. These image transforms are discussed in Section 4.3. The computer vision and machine learning libraries used are briefly discussed in Section 4.4. Section 4.5 through Section 4.8 presents an outline for each investigation to be considered. These definitions include a short introduction to the investigation, how the original dataset is augmented to train and test the respective algorithms, and the parameters used to compare the results. The neural network and clustering algorithms presented in this research are trained using an Nvidia RTX2080 Super GPU.

4.1 Introduction

Extensive research on using automated methods for plankton taxonomy has been conducted since the turn of the century. With exponential enhancements in technology, humans are now able to analyse not only samples in a laboratory but also in situ. Humans can also implement complex algorithms, such as the MorphoCluster algorithm, to assist marine experts in annotating plankton samples. While the combining these systems can drastically improve the way plankton data is collected, the job still requires an immense amount of time and effort. This is due to the inherent noise associated with monitoring any natural process which causes semi-supervised methods to require a lot of human input. In the case of plankton analysis, this noise comes from increasing levels of dead matter and other artefacts that share the

same world as the plankton. Environmental conditions, unseen species and poor image resolutions also plague the field of plankton taxonomy, decreasing the ability of humans and machines to interpret real-life sample sets. Chapter 2 shows that even when automated computer vision algorithms are applied in laboratory environments, they still don't perform up to levels where researchers would not have to validate most of the samples. Most automated machine learning plankton taxonomic research proves that the proposed models are best employed using highly pre-processed and cleaned datasets and that they underperform when subjected to real-life sample sets. The models used in the studies shown in Chapter 2 use a small number of distinct classes and avoid spatially challenging samples such as detritus.

This research presents a data-driven approach utilizing multiple validated, real-life datasets obtained using the ZooSCAN. The aim of this research was to build a plankton taxonomic pipeline that is a less human involved version of the successful MorphoCluster system. The pipeline uses an enhanced feature extractor designed to detect minor intraclass differences between subphylum classes and separate large noisy samples, such as detritus, from the dataset. It then clusters these features using the HDBSCAN algorithm and automatically labels the resulting clusters using an auxiliary classifier network structure.

4.2 Broader Dataset Overview and Class Assignment

The broader dataset used in this study is an amalgamation of real-life plankton datasets collected and validated by the Nelson Mandela University Ocean Sciences Campus. A summary of the entire dataset used is found in Appendix A. This research presents an ensemble method utilizing multiple models built to process the subsets of the larger dataset. It proposes that no samples be discarded during the data cleaning process unless the class does not contain at least 10 ROI. This is because classes with less than 10 ROI in the combined raw dataset are deemed to be insignificant in this study and could force the relevant network to overfit those samples and underfit others. If discarded classes eventually contain enough samples, then the class in consideration can be added to the feature extraction network. Samples that belong to classes deemed “overall” in the class column, found in Appendix A, are removed from the original set and placed within their own overview subset set.

With large plankton datasets, there are a few elements to consider. These elements are used to separate the dataset into three different types of classes, the first of which is based on an abundance of dead matter, otherwise known as detritus. Detritus is a very prominent feature of real-life plankton datasets, and the ability to correctly identify ROI of this type greatly increases the ability of neural network models to identify the rest of the samples. As shown in the dataset summary in Appendix A, detritus makes up almost a quarter of the total samples in the dataset. In situ methodologies have a vast amount more detritus to take into consideration as the net used to capture samples in ex situ methods tend to destroy a significant number of samples. Identifying these samples is a complex task as detritus samples exhibit significant spatial differences between them, occurring in any shape or form, as shown in Figure 4.1.

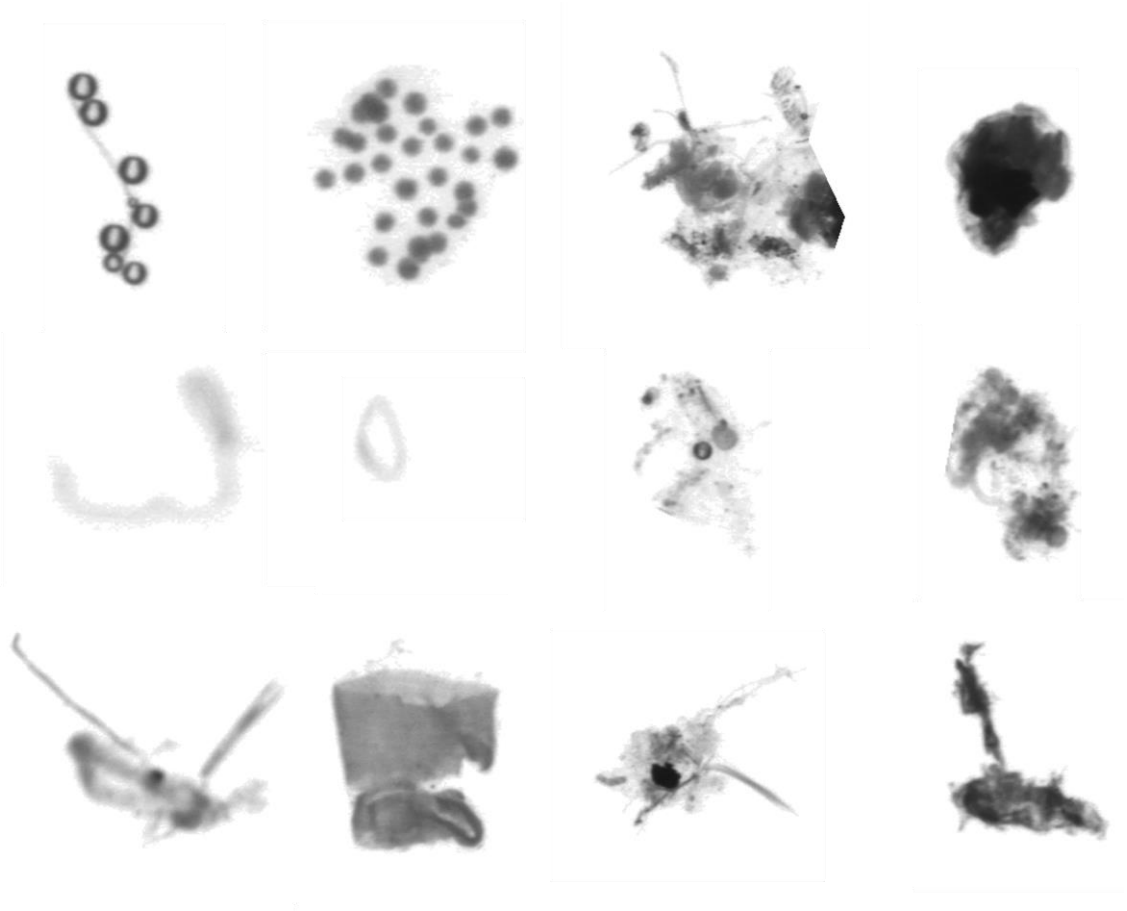

Figure 4.1: Detritus class (Type 1) examples

Type 2 classes do not share close biological and visual similarities to other classes, but exhibit consistency within their own class. These classes can be in the form of living organisms such as Appendicularia or Cyphonaute shown in Figure 4.2 and Appendix A. They can also be in the form of non-living artefacts such as bubbles.

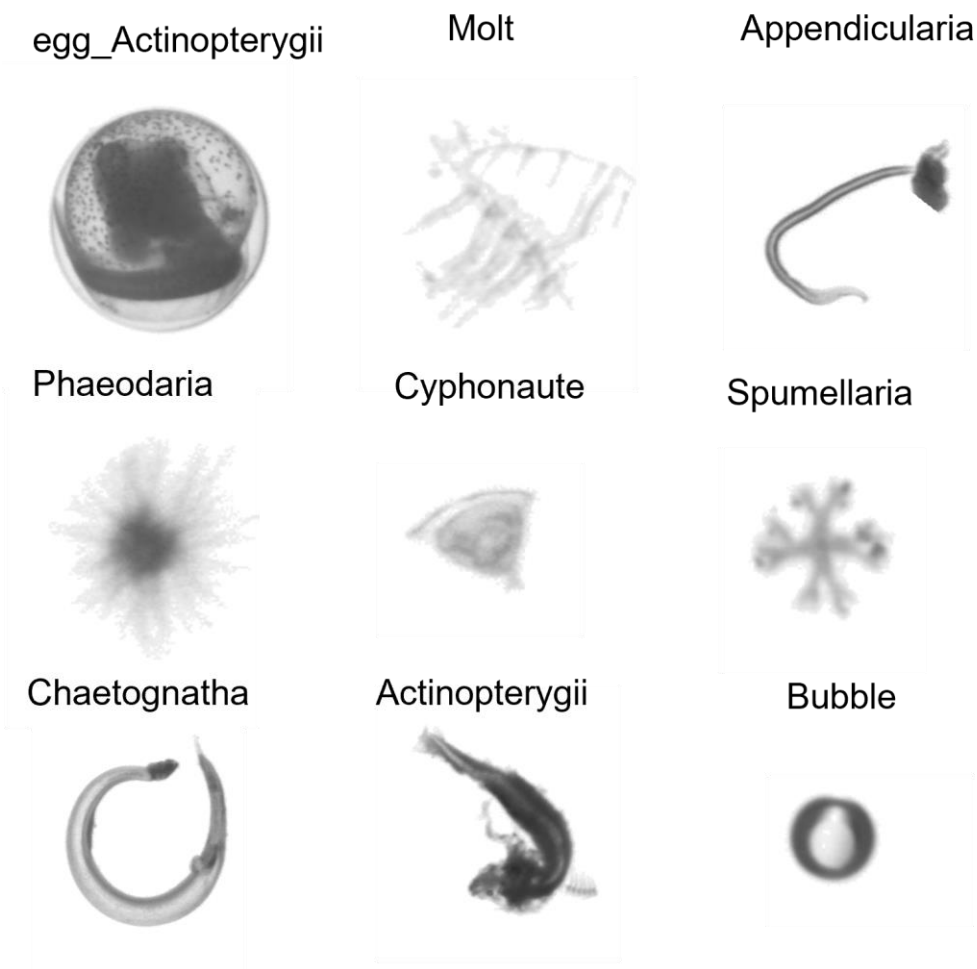

Figure 4.2: Type 2 Classes

Other classes that occur in abundance and make up the type 2 class are the immature plankton classes, various body parts of different classes, and ROI containing multiples of one class, as shown in Appendix A.

The third and final type of class considered are classes that ultimately form the plankton hierarchical structure researchers use to organise and validate the ROI. These samples belong to a subphylum, with other classes of the same subphylum exhibiting high spatial similarities. These classes are indicated in Appendix A and shown in Figure 4.3.

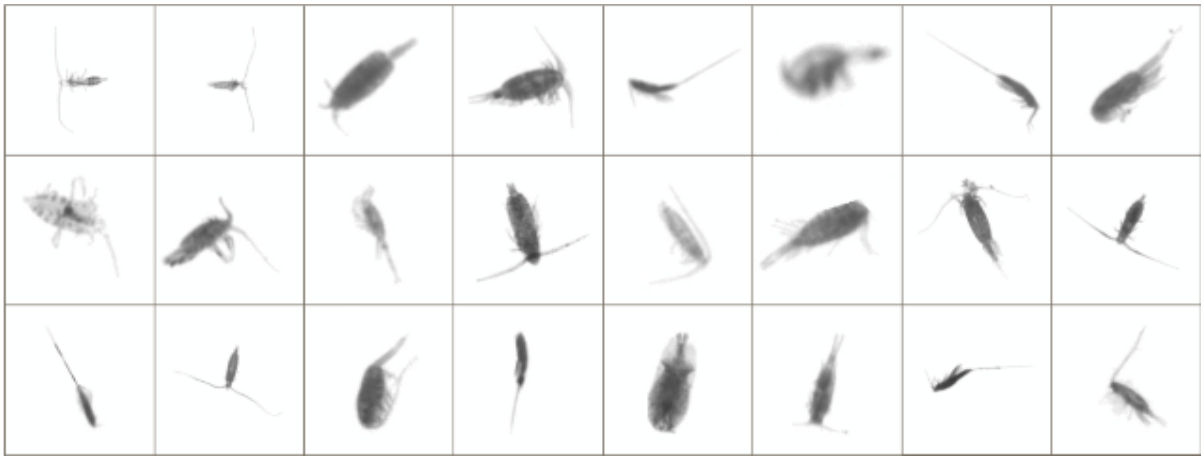

Figure 4.3: Crustacean Subphylum showing high similarities between different classes

The first of the three types of classes are considered in Investigation A. The second and third type of class is utilized in Investigation B. Investigations C and D use all the described classes.

Another general dataset is constructed, where objects that belong to noisy particles and ROI not considered in most plankton studies are removed. This cleaner dataset is comprised only of samples that belong to a specific subphylum.

All machine learning algorithms require input standardization to effectively compute predictions in training and testing. In computer vision, this means performing image pre-processing techniques on the ROI before they are fed to the neural network for training or inference. Techniques used throughout training and testing of the investigations are presented in Section 4.3.

4.3 Image Processing

To successfully train and test multiple deep learning models on the amalgamated plankton dataset obtained from the original ZooSCAN, the ROI must go through various stages of pre-processing. The first of these stages removes the original size scale markings on the images. The marking is shown in Figure 4.4 and always occurs at the same relative distance and size in every ROI. This marking is removed by cropping every image by a pixel value of 10 from the bottom of the images.

Figure 4.4: Original ZooSCAN ROI

The predefined neural network architectures used in the investigations are described in Section 3.2. These architectures mostly take in different input sizes. The ResNet architectures typically accept an input size of 224 x 224, whereas the Inception v3 network takes in input images of size 299 x 299. As some models perform better on various datasets, different architectures are used, and the input images must be of an acceptable size. In these investigations, the images are resized to 3 values higher than the required input and then centre cropped down to the correct size. Where images are smaller than the required size rescaling, they could cause skewing and bad resolution versions of the original image. Instead, these smaller images have their original image padded with the same colour as the background to a size 10 values greater than the desired input size. The resizing and cropping

are then performed the same way as before. All the input images have their channels reduced from 3 (red, green, and blue) to 1 (greyscale).

Due to imbalance in the dataset, each investigation has its own sampling method described in their relevant chapters. However, one way in which the image processing pipeline assists with this matter is by flipping the various samples at random.

Lastly, the image is converted into a matrix called a tensor containing values between 0 and 256 depending on pixel intensities. The image tensor is then normalized to represent the same grayscale pixel ratios as values between 0 and 1. The whole image processing pipeline presented in this section provides the network with an easily consumable matrix representation of the respective plankton ROI.

4.4 Computer vision and machine learning libraries

Python was the language used to implement the algorithms throughout all the investigations undertaken in this research. Python is an OOP (object-oriented programming) language with many open-source libraries available for the implementation of data manipulation, computer vision and machine learning algorithms. This chapter briefly introduces some of the essential libraries utilized.

Pytorch

Pytorch is an open-source machine learning library created by Facebook's AI Research Lab (FAIR). It is based on the Torch Library, which is a scripting language derived from Lua programming (Léonard *et al*., 2015). Pytorch allows for processing tensor-based matrices with a special focus on graphical processing unit (GPU) acceleration. Where other popular machine learning libraries like TensorFlow work on a pre, user-defined, static graph computation of the model, Pytorch is based on a dynamic graph allowing users to manipulate the graph on the go, i.e. implementing dynamic learning rates. Pytorch is used to define and train the supervised learning architectures proposed in this research.

Scikit-learn

Scikit-learn is also an open-source machine learning library that provides various implementations of supervised and unsupervised methods. Scikit-learn is used in this research to investigate the unsupervised learning methods researched in this paper.

HDBSCAN

The HDBSCAN library is an open-source library designed purely for density-based clustering for applications with noise.

Pandas Data frame

Pandas is a Python package that was built to be the fundamental building block for real-life data manipulation and analysis in Python. Pandas is a fast and malleable data structure allowing easy and intuitive use of relational and annotated datasets. Pandas, like most data structures, is made up of data viewed in rows and columns. Pandas is used in these investigations to manage and store data for model analysis purposes. It also provides the structure for moving data from the supervised Pytorch models to Scikit-learn and HDSCAN libraries.

This research makes use of many Python libraries for dataset interpretation and manipulation. Those mentioned above underline the essential ones utilized. Section 4.5 introduces the first of four studies undertaken to fulfil the research objectives.

4.5 Investigation A: CNN Binary Classifier and Feature Extractor for Classes Exhibiting Low Intra Class Similarities

4.5.1 Introduction

One problem encountered by the current real-life implementations of vision algorithms for plankton classification is the abundance of detritus samples. These samples often make up to 70% of the total sample set under consideration. Detritus does not have an exact or predictable shape and exhibits large intra-class differences, as shown in Figure 4.1, as opposed to more standard plankton classes, as shown in Figure 4.3. These low intraclass differences pose a challenge to traditional techniques such as random forest, but more so for general convolutional neural networks trained on many different classes. The nature of backpropagation within these general deep learning convolutional neural networks causes them to have decreased performance across all classes when classes such as detritus are included in their training set. Investigation A proposes training a CNN to identify the presence of a specific class in a binary classification manner, rather than classifying between more than two different classes.

4.5.2 Dataset

The classes under consideration from the original training set are shown in black in Appendix A. These classes are considered type 1 classes. Setting up a specialized structure for binary classification, as shown in Figure 4.5, allows for the training of a binary classifier to identify the presence of a certain class. The sampling algorithm for Investigation A is shown in Algorithm 5. The sampling algorithm creates a new dataset containing two classes. The first is the class of interest (i.e. detritus). The second is a new class made of randomly sampled ROI from the original plankton dataset, excluding the first class. The algorithm makes sure the classes are balanced by sampling the same number of random ROI as there are samples in the class of interest. This sampling is performed for the training, testing and validation subsets of the original dataset.

Algorithm 5 – Binary folder creation

```
Choose class of interest
While size of output < size of interest class
    for each element in the list of other classes (!= class of interest)
        Pick a random sample
        Send sample to collective folder
end
```

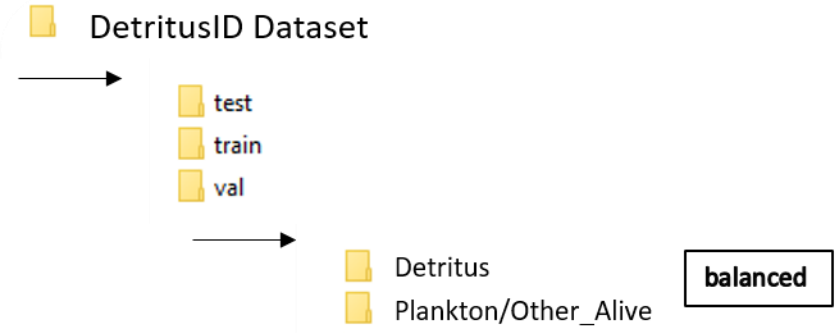

Figure 4.5: Binary classifier folder structure

4.5.3 Evaluation parameters

To evaluate the performance of the trained binary CNN classifier on type 1 classes, the model is placed into inference mode and used to predict the presence of the specific class vs the other class. The ability of the model to make a correct decision on whether it encounters a sample of interest indicates that the sample has learnable and distinguishable features even with its significant intra-class spatial differences.

4.6 Investigation B: CNN Subphylum Classifier and Feature Extractor

4.6.1 Introduction

CNN feature extractors such as the one used for MorphoCluster are trained to classify between a variety of different classes. The problem with training networks in many classes is that the network tends to overfit or underfit. This causes the network not to recognize the differences between classes from the same subphylum. This is due to the large spatial differences between some of the classes whilst others exhibit very high similarities, as shown in Figure 4.3.

This investigation proposes training individual CNN models on type 2 or 3 subsets of the original dataset. This approach aims at teaching the network more fine-grained differences between similar classes.

4.6.2 Data

Investigation B makes use of the original dataset except for all the type 1 classes. The considered classes are separated into groups based on biological feature similarities. These groups include seven different subphylum groups of classes, four specialized groupings, and the overall subphylum group as explained in Section 4.2. The folder breakdown is shown in Figure 4.6.

Figure 4.6: Folder groupings

The subphylum classes are constructed using the plankton taxonomic classifications from the EcoTaxa and WoRMS online platforms. These subsets are unbalanced, which would cause the networks to overfit to the most abundant classes. This imbalance is mitigated by using a weighted sampling method.

The algorithm starts by analysing the number of samples within each class and then proceeds to take one ROI from each class until the desired batch size is met. This means the algorithm resamples ROI from less abundant classes in the subset, but due to the image processing and augmentation techniques described in the image processing section, the resampled ROI will have a different appearance. The algorithm terminates once all the ROI in the considered subset have been sampled at least once.

4.6.3 Evaluation Parameters

The performance of each subphylum trained model is compared to the general CNN architecture used in methodologies such as MorphoCluster. This is done by putting the subphylum models into inference mode and testing the ability of each algorithm to identify classes from its dedicated subphylum from the testing data. Another generalized model that is trained on all the data used to create the subphylum datasets is then also placed into inference mode and tested on each subphylum test set. The performance of each individual subphylum model is then compared to the ability of the generalized model to predict classes from its dedicated subset.

4.7 Investigation C: HDBSCAN of Ensembled Feature Extractors

4.7.1 Introduction

Modern plankton classification techniques such as MorphoCluster make use of a CNN feature extractor trained on the entire dataset and then proceed with unsupervised clustering of the visual features proposed by the CNN. The HDBSCAN algorithm aims to group samples with like features and identify outlier samples. This investigation aims to determine whether ensembling the feature extraction methodologies used in investigations A and B result in increased performance as opposed to using a single general feature extractor.

4.7.2 Data

Investigation C makes use of a stacked classification architecture that uses a stack of models trained in investigations A and B. These models are then compared to individual classifier networks trained on variations of the entire dataset. The data in this investigation is used to evaluate the performance of the stacked classifier network against that of the singular classifier network.

4.7.3 Evaluation Parameters

To evaluate the performance stacked classification network against that of the singular network, each algorithm is trained and tested on the same variation of the dataset. This evaluation is done by comparing the network under consideration's predictions against the ground truth values. Once the best performing network is selected, the more complex data is injected into the dataset, and the algorithms are trained and tested again.

4.8 Investigation D: Automated Cluster Identification Using Predicted Pseudo-Labels

4.8.1 Introduction

The MorphoCluster methodology that represents the current state-of-the-art approach to plankton taxonomy uses a combination of machine learning techniques and a marine specialist's assistance. This hybrid approach produces results potentially unattainable by the machine or human counterpart alone. The use of an expert to validate and name the clusters based on visual perception results in high accuracy but reduces the speed at which the taxonomic process can be performed tenfold. This investigation makes use of an auxiliary branch of the feature extractors studied in investigations A and B. This auxiliary branch contains the soft label prediction of the CNN feature extractor before decapitation. The way human marine experts name the clusters using visual perception is mimicked by CNN networks specifically trained on a subset of classes, be they binary or multiclass classification networks. If more than one sample is perceived to be of a certain class with a certain confidence, the cluster is then renamed to be a cluster of that specified class. This autonomous approach, although not as accurate as using human experts, increases the efficiency of the plankton taxonomic system.

4.8.2 Data

This investigation uses the entire labelled dataset without separating the data into either its binary subsets or subphylum subsets. Like Investigation C, Investigation D is an unsupervised method and therefore does not require training. Instead, the dataset is used to evaluate the performance of the automated plankton taxonomic pipeline.

4.8.3 Evaluation Parameters

Evaluating the performance of unsupervised algorithms is not as simple as measuring the performance of supervised algorithms. Where supervised methodologies utilize ground truth labels to determine the performance of the algorithm directly, unsupervised methods are usually evaluated based on some similarity or dissimilarity measure such as the distance between cluster points. The evaluation metrics used in this investigation include the number of identified clusters, the Silhouette Coefficient and Dunn's Index.

The Silhouette Coefficient or Silhouette Score is a method used to validate the goodness of a clustering algorithm. The algorithm, as shown in Equation 21, contains two variables, a and b. The a variable represents the average distance between data points within each cluster, and b represents the average distance between the clusters.

$$\boldsymbol{S(i) = \frac{b_i - a_i}{max(b_i, a_i)}} \quad (21)$$

The resultant value can range between -1 and 1, where a result of 1 means there is clear separation and dense clusters, and a score of -1 indicates the clusters are not assigned correctly. Another metric measure of fitness is known as Dunn's Index.

Dunn's Index shown in Equation 22, makes use of the minimum inter-cluster distance as a ratio with respect to the maximum intracluster distances.

$$\boldsymbol{DI = \frac{min(Inter\ cluster\ distance)}{max(Intra\ cluster\ distance)}} \quad (22)$$

A higher DI means more densely packed clusters that are well separated, meaning a higher DI value indicates a potentially better clustering algorithm.

However, the two mentioned evaluation methods are more suited to non-density-based clustering methods as they do not take noise into account. Noise is an important characteristic in density-based clustering applications, as discussed in Chapter 3.4.2. To evaluate the performance of the automated pipeline, each sample is assigned to either a corresponding

cluster or as noise. Once every sample is assigned, the algorithm starts identifying the name of the clusters and merging similar clusters. For every sample, the automated pipeline's predicted cluster label is then compared to the true label, just like when measuring the performance of supervised learning algorithms.

5 CNN Binary Classifier and Feature Extractor for Classes Exhibiting Low Intra Class Similarities

5.1 Introduction

This chapter provides a full analysis of the proposed feature extraction algorithm for classes that exhibit very low-class similarities (Type 1). The proposed network makes use of a binary classification technique, where instead of training the network on a set of different classes, the network is trained to identify the presence of a single class. The data and evaluation parameters used are discussed in Section 4.5.1. The classes considered for this investigation include Type 1 classes such as detritus. The implementation of the networks is explained in detail in the next section.

5.2 Implementation Detail

The implemented architectures are either ResNet18, Resnet36, WideResNet32 or DenseNet. Each network type is trained on the binary datasets, and their results are analysed. Once the best achieving architecture is selected for each class, the model is fine-tuned by adding a linear fully connected layer onto the network and freezing the model's original convolutional neural network part. The model is then retrained on the same dataset as before. The retraining of the model without backpropagating the convolutional part forces the network's new fully connected layers to be single layer representations of the network. These single layer representations can be seen as the features the network extracts from every sample it gets passed. A network that best classifies the test dataset will present better features in the layers before the output layer. An example of how the binary architecture is setup is shown in Figure 5.1.

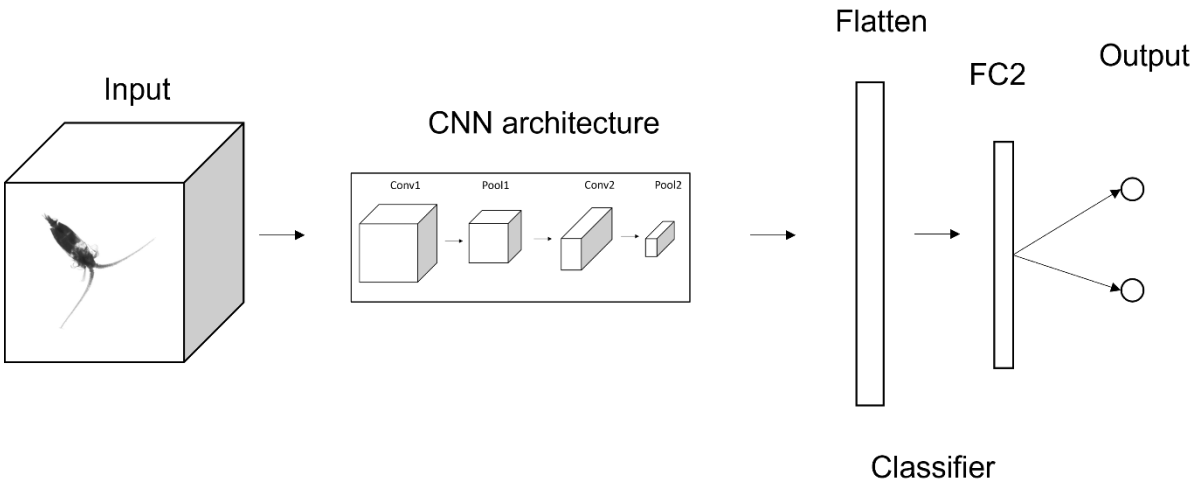

Figure 5.1: Architecture showing binary classification architecture

Once the model is retrained, the previous fully connected layer is used as the primary output in the proceeding investigations.

5.3 Results and Discussion

To validate the performance of the binary classification algorithm, various tests were conducted on a model trained to identify detritus samples. The initial test conducted determined whether including detritus samples in the training set had adverse effects on a generalized network. As explained in Chapter 4.5, most research in plankton taxonomy has found that detritus and other non-plankton samples, found in abundance in real-life datasets, negatively influence the performance of the proposed classification networks. It was found in this research that including only the detritus class in the cleaned dataset decreased the model's accuracy by 0.014 taking its testing accuracy down to 0.64. This is because the samples within the class exhibit large intra-class differences and training a model to identify many other classes including the detritus class increased the required parameters by an exponential amount.

This investigation then continued with training a binary classification model on a one vs all basis, where a classifier was trained to identify the presence of detritus only. Figures 5.2 and

5.3 show the training accuracy and validation accuracies achieved when training the algorithm. The step variable along the x-axis represents the number of epochs the training has undergone. The other class used to train the binary classifier was constructed from the original dataset from samples that are not part of the detritus class, as discussed in Section 4.5.2.

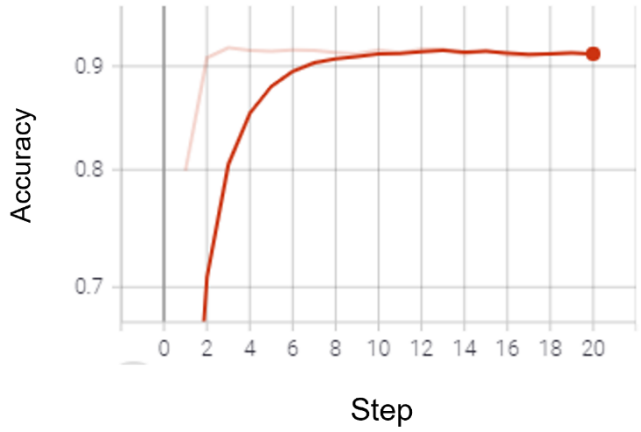

Figure 5.2: Performance of binary classification algorithm on detritus class during training

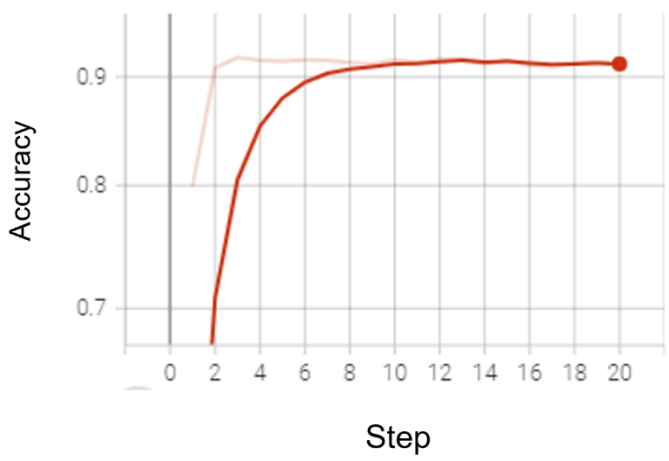

Figure 5.3: Performance of binary classification algorithm on detritus class during validation

The results shown in Figures 5.2 and 5.3 indicate that there are indeed inherent features associated with detritus samples, such that when the transformed dataset changed the focus of the model to detritus identification, it achieved a validation performance of 0.92. Figure

5.4 shows the PR curves obtained from the binary classifier. These curves trade off between true positives (versus false negatives as a proportion of the real positives) and false positives (versus true negatives as a proportion of the real negatives) and are equivalent to comparing Sensitivity (+ve Recall) and Specificity (-ve Recall). A greater area under the curve represents a higher performance.

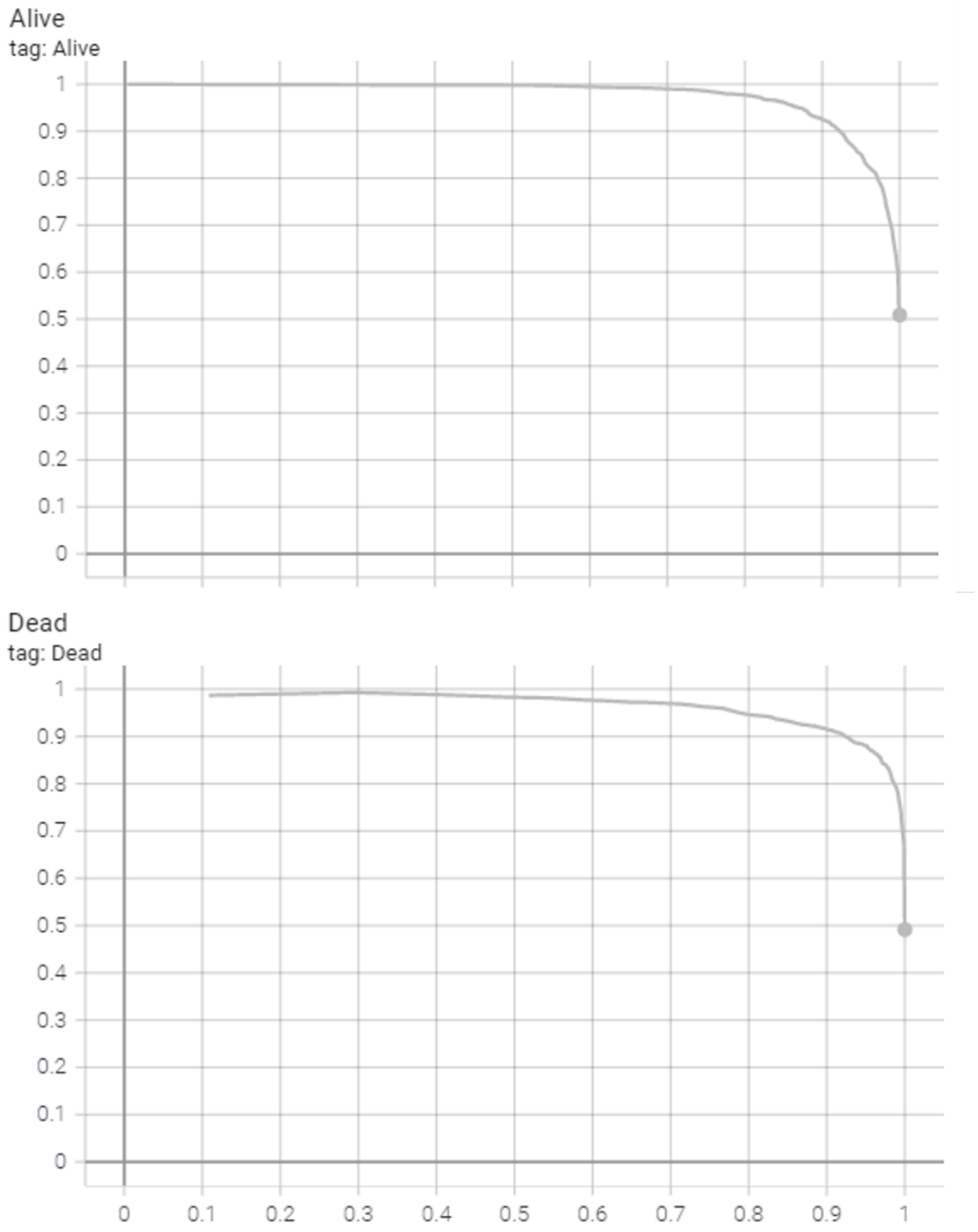

Figure 5.4: PR Curves of a model trained to identify between dead or alive samples

The PR curves shown in Figure 5.4 indicate that the model is robust to identifying between the dead and alive samples, indicating that the features extracted in the CNN are sufficient to identify the presence of detritus. Figure 5.5 shows the confusion matrix obtained when testing the matrix.

	ALIVE (TRUE)	DEAD(TRUE)
ALIVE(PRED)	4414	302
DEAD(PRED)	580	4524

Figure 5.5: Detritus classification network confusion matrix

6 CNN Subphylum Classifier and Feature Extractor

6.1 Introduction

Classes that don't belong to type 1 can be assumed to be part of type 2 or type 3 classes. These classes belong to subphylum or group samples that exhibit similar features to one another. When training feature extractors on these plankton subphylum sets, as opposed to one general CNN feature extractor, the goal is that the specifically trained algorithms tend to learn more fine-grained features between classes. This system is especially viable for plankton classification as classes are quite obviously different to samples from another subphylum.

The goal of training models based on various non-biological groupings is to show that samples removed from the models trained in the plankton studies discussed in Chapter 2 have distinguishable features and, therefore, can be learnt. The problem experienced by these studies is that training a single general algorithm to take care of such a spatially variant task results in less-than-optimal results. Further implementation details of the networks used in this investigation are described in Section 6.2.

6.2 Implementation Details

In this investigation, each of the subsets of classes considered exhibit different features. This suits some CNN architectures better than others. The deeper the network, the more complex the features extracted by the algorithm, while layers near the beginning of the network extract more rudimental features. Some of the groups created for this investigation contain only a few classes and, therefore, will be better off using shallower networks such as the ResNet18 architecture.

Similarly to Investigation A's implementation discussed in Chapter 5.2, the best achieving architecture is selected for each class, and the model is fine-tuned by adding a linear, fully connected layer onto the network and freezing the original convolutional neural network part of the model. The model is then retrained on the same dataset as before. The retraining of

the model without backpropagating the convolutional part forces the network's new fully connected layers to become single layer representations of the network. These single layer representations can be seen as the features the network extracts from every sample it gets passed. An example of how the architecture is set up is shown in Figure 6.1. The data and parameters used in this chapter are discussed in Section 4.6.

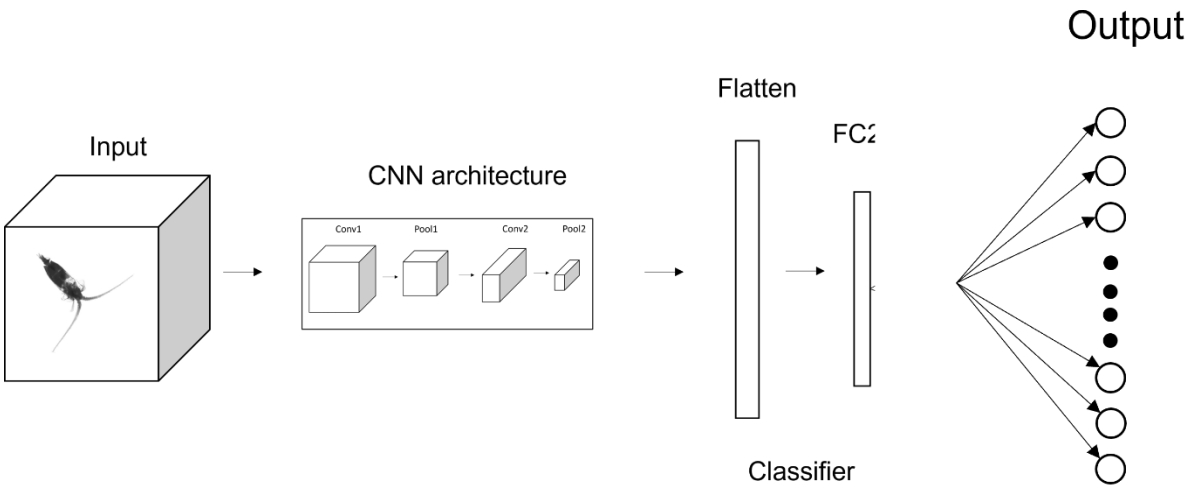

Figure 6.1: Group/subphylum architectures

6.3 Results and Discussion

The ability of the networks to distinguish between classes with minor intraclass differences can be better seen using Precision Recall (PR) Curves. These curves trade off between true positives (versus false negatives as a proportion of the real positives) and false positives (versus true negatives as a proportion of the real negatives) and are equivalent to comparing Sensitivity (+ve Recall) and Specificity (-ve Recall). A greater area under the curve represents a higher performance.

This section presents some of the more obvious situations where using the specifically designated CNN networks for certain inter-class or intra subphylum/grouping classifications outperformed the generalized CNN model proposed in another research. The generalized model used to compare the performances was trained on the cleaner dataset. Another model

trained on the entire dataset is also used to illustrate the effectiveness of breaking up a dataset into consumable subsets rather than just removing the samples.

Figures 6.2, 6.3 and 6.4 show only a few of the PR curves obtained comparing the subphylum specific models to the clean generalized classifier.

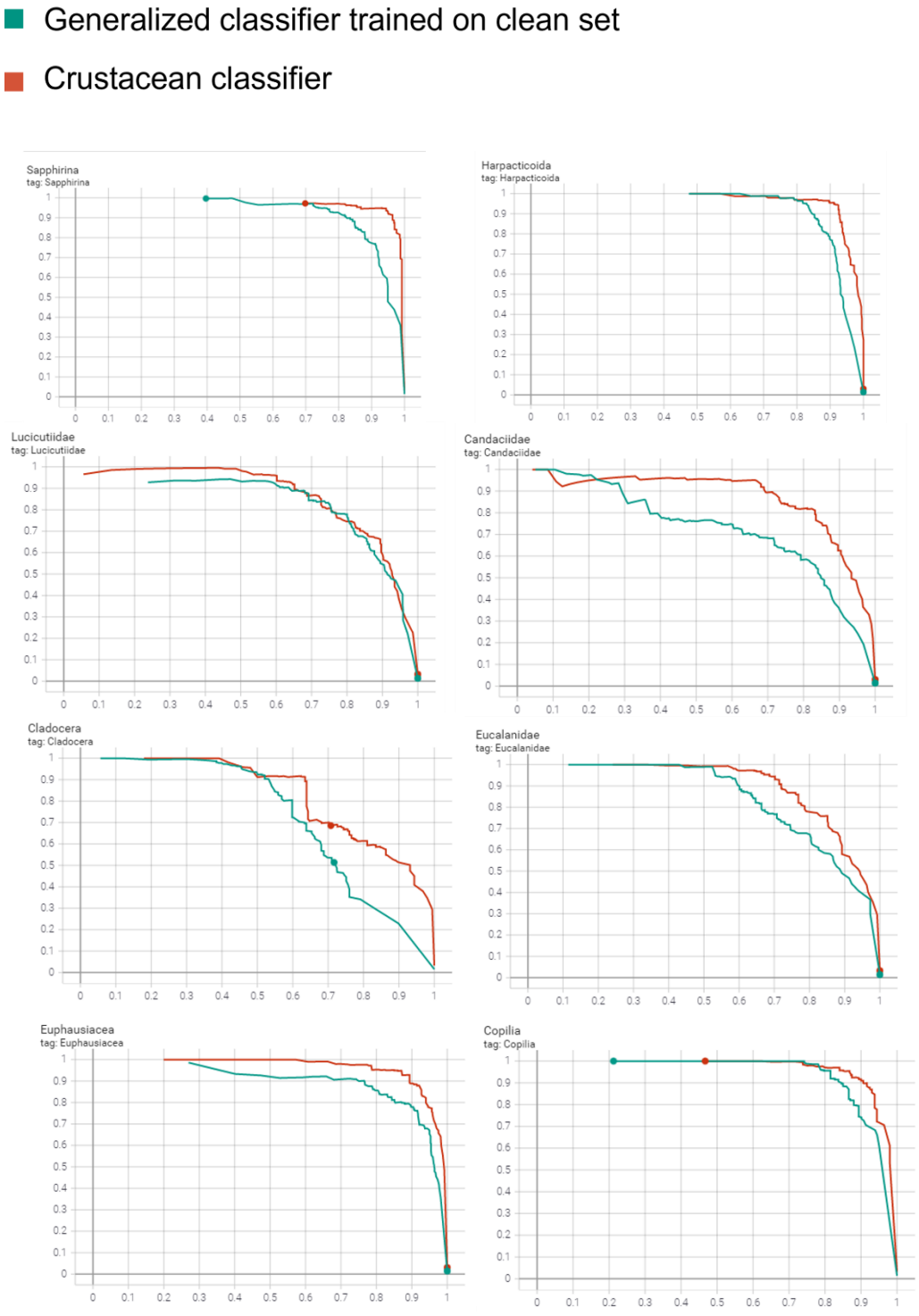

Figure 6.2: Crustacea class resultant PR curves

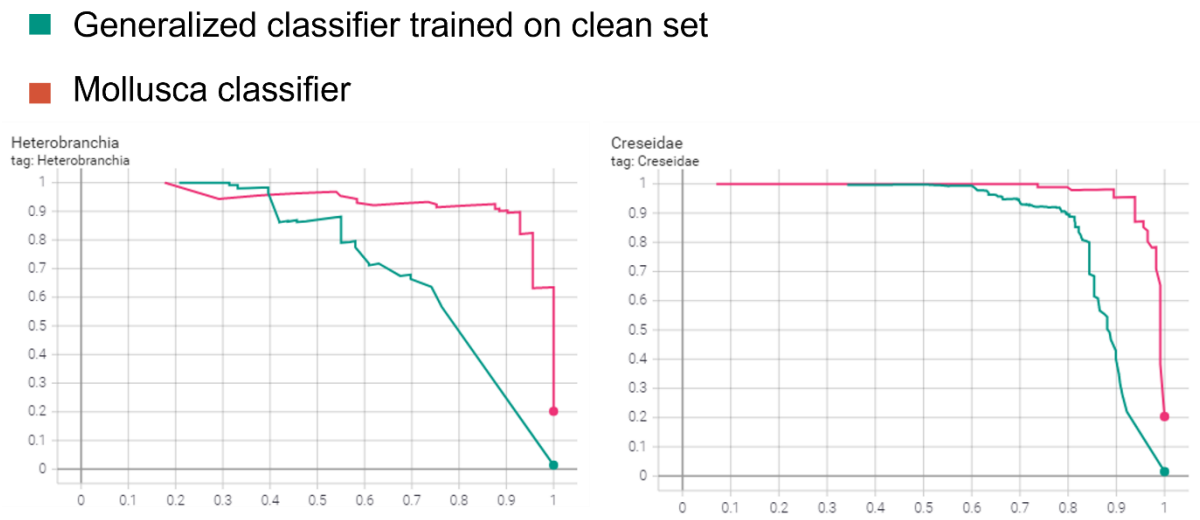

Figure 6.3: Mollusca class resultant PR curves

Figure 6.4: Various other subphylum class resultant PR curves

The results displayed in Figures 6.2, 6.3 and 6.4 indicate that the use of models trained on specific subsets of the data results in overall better recall performances than that of a single

generalized classification network. There is no case where the general model achieved a better PR curve than the relevant subphylum model. Figure 6.5 shows the PR curve results obtained from networks trained on subsets usually discarded in plankton research as they are classified as noisy samples. These PR curves are compared to the generalized classifier trained on the entire dataset.

■ Generalized classifier trained on overall set

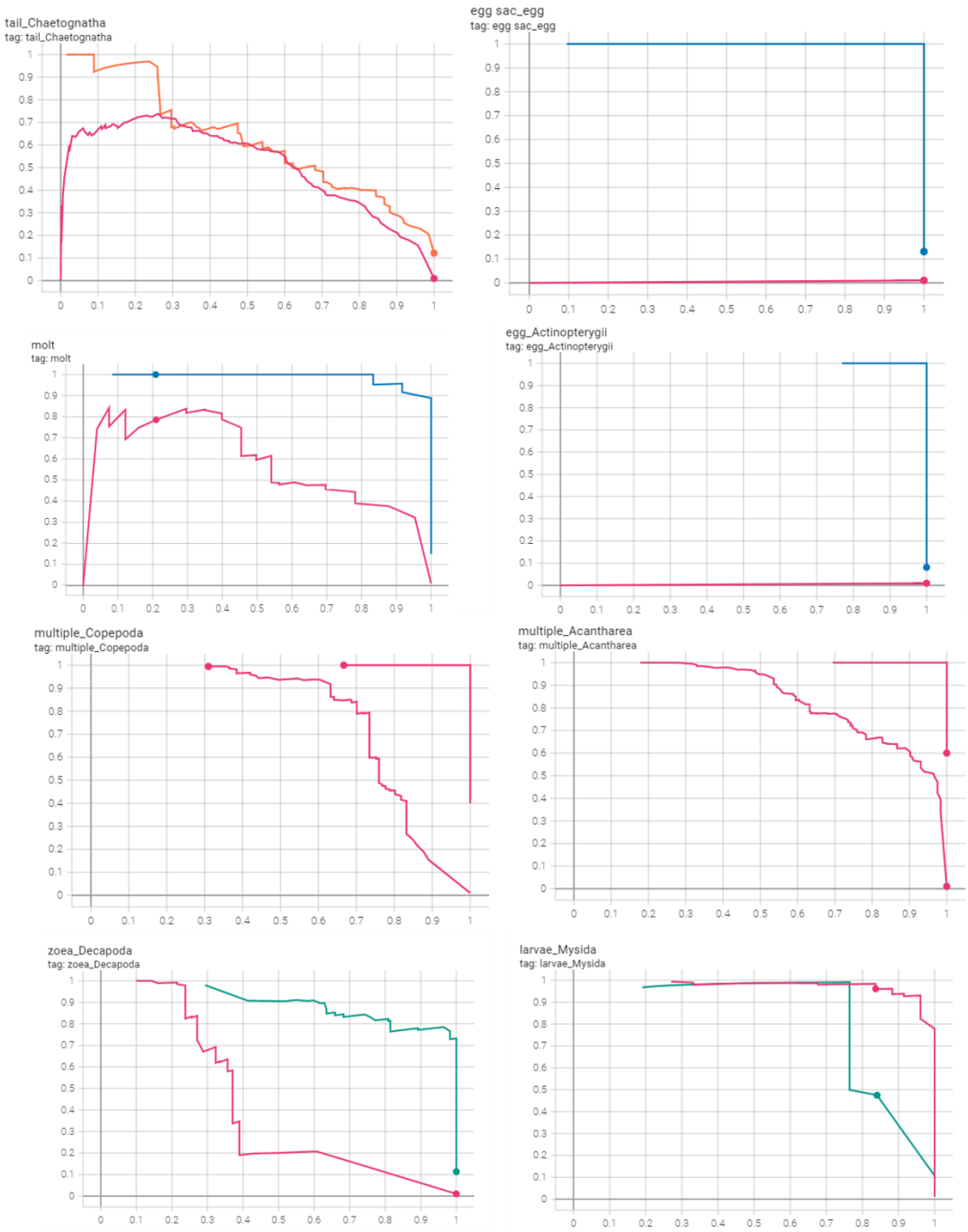

Figure 6.5: PR curves of grouping classifier compared to the generalized classifier trained on the entire set

The resultant PR curves shown in Figure 6.5 indicate that the use of subset classification networks can be used to identify classes that are usually discarded from the datasets to increase the model's performance.

Overall, the cleaned, general feature extractor correctly classified 38114 out of a total of 51930 unseen testing samples giving it a perceived accuracy of 0.73. The performance of the individual classification algorithms trained on subsets of the same data achieved an accumulative score of 43025 out of the same total 51930 testing samples resulting in a perceived accuracy of 0.829. This result also excludes the performance of these subgroup classifiers to classify samples that are not present in the cleaned, general feature extractor. However, introducing these more complex classes into a generalized model decreases the model's ability to be trained, resulting in overfitting.

Investigation 2 shows that feature extractors built to identify certain biological or physiological groupings achieve better results than a single generalized feature extractor. This is especially true when the training sets contain noisy samples such as organism body parts, eggs etc., outlined in Section 4.2 as type 2 classes.

Chapter 7 investigates the effectiveness of combining the subgroup classifiers as feature extractors for clustering purposes.

7 HDBSCAN of Ensembled Feature Extractors

7.1 Introduction

This investigation explores the ability to stack the feature extraction methods presented in the previous two investigations, A and B. Individually, the two studies prove their effectiveness in classifying their specified classes and class types. This investigation brings these studies together to analyse the performance of a novel combined technique for plankton feature extraction. The performance of the previous two studies is based on the ability of the networks to classify their dedicated classes as compared to a general network. This investigation uses all the trained networks from the previous studies and decapitates them to become feature extractors. A new classifier network is trained using these features. The new stacked classifier network is then also decapitated, and the algorithm then reduces and clusters the resultant feature vector. Further implementation details are discussed in the Section 7.2.

7.2 Implementation Details

The feature extraction and classification pipeline presented in this investigation is composed of three separate stages. The first two stages are supervised methodologies, and the final stage is unsupervised.

The first stage of the algorithm is training the relevant feature extractors as conducted in the previous two studies; this is achieved by training CNN image classifiers, using specific subsets of the original raw plankton dataset. Once these image classifiers are trained, they are fine-tuned by retraining the network after adding in an extra fully connected layer before the output layer. These trained image classifiers serve as high dimensional feature extractors for the next stage of this investigation's algorithm.

As shown in Figure 7.1, the second stage of the pipeline that needs to be trained is the stacked classifier network appended onto the outputs of all the first stage's trained networks. The

feature extraction networks take a batch of images and produce two-dimensional feature vector representations of the input images. Each networks output image feature vector is concatenated with the same picture's output from the other networks. This produces a larger feature vector with a combined understanding of the input image. The image's output combined feature vector and the associated image label is then used to train a multilayer fully connected network. The network comprises two fully connected layers, the first of which uses a leaky ReLU activation function and dropout. The second layer is connected to the output layer. In training, the loss function is calculated using cross-entropy loss. This is a loss criterion that combines SoftMax and the negative log-likelihood loss. The loss is backpropagated through only this stacked classifier network, updating only the weights of the fully connected layers. The output of the second FC layer represents the prediction scores for the various classes. The new stacked classifier network is trained using the original dataset containing samples such as detritus and other noisy artefacts.

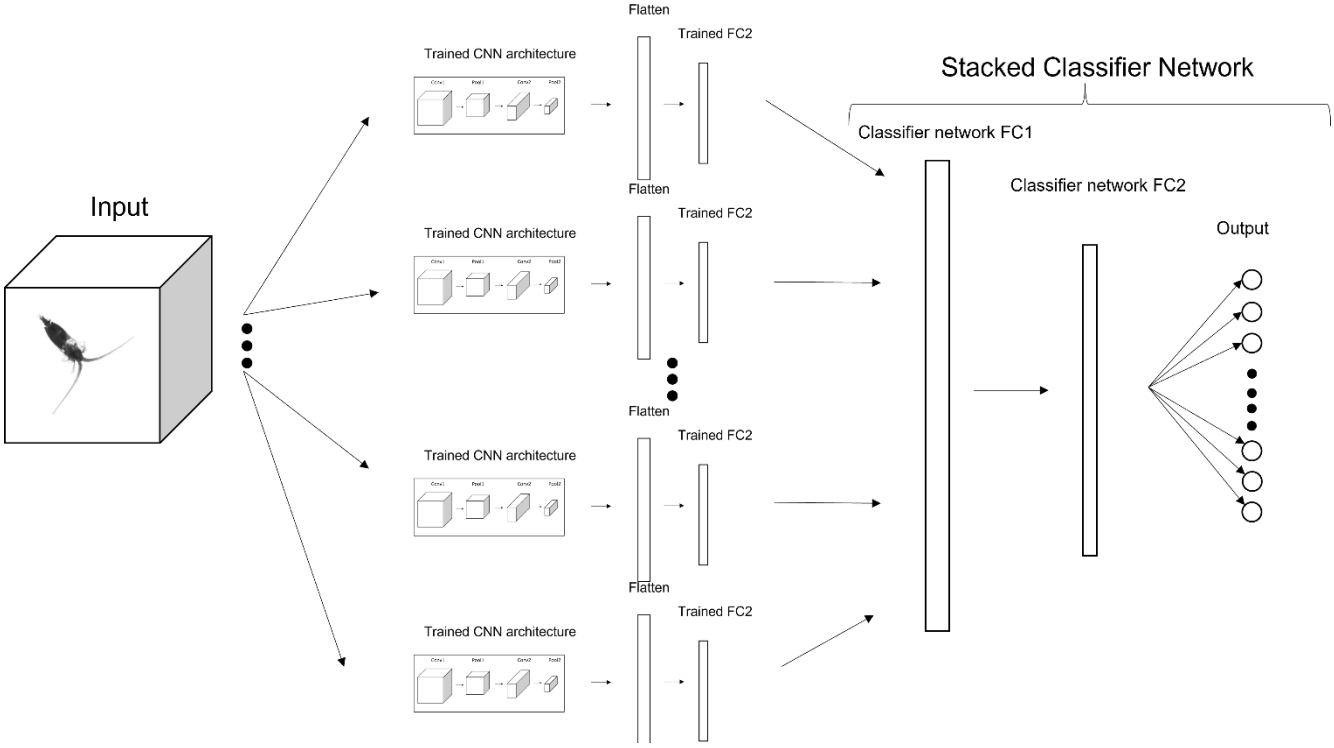

Figure 7.1: Investigation 3 architecture

The final stage of this investigation involves removing the output layer of the stacked classifier network and placing the fully connected layers in inference. This output feature vector is a lower-dimensional representation of the original input image. The extracted features are then further reduced with PCA and then clustered using HDBSCAN, as shown in Figure 7.2. The

results obtained when comparing the performance of this combinational technique versus using a single large network are presented and discussed in the next part of this chapter.

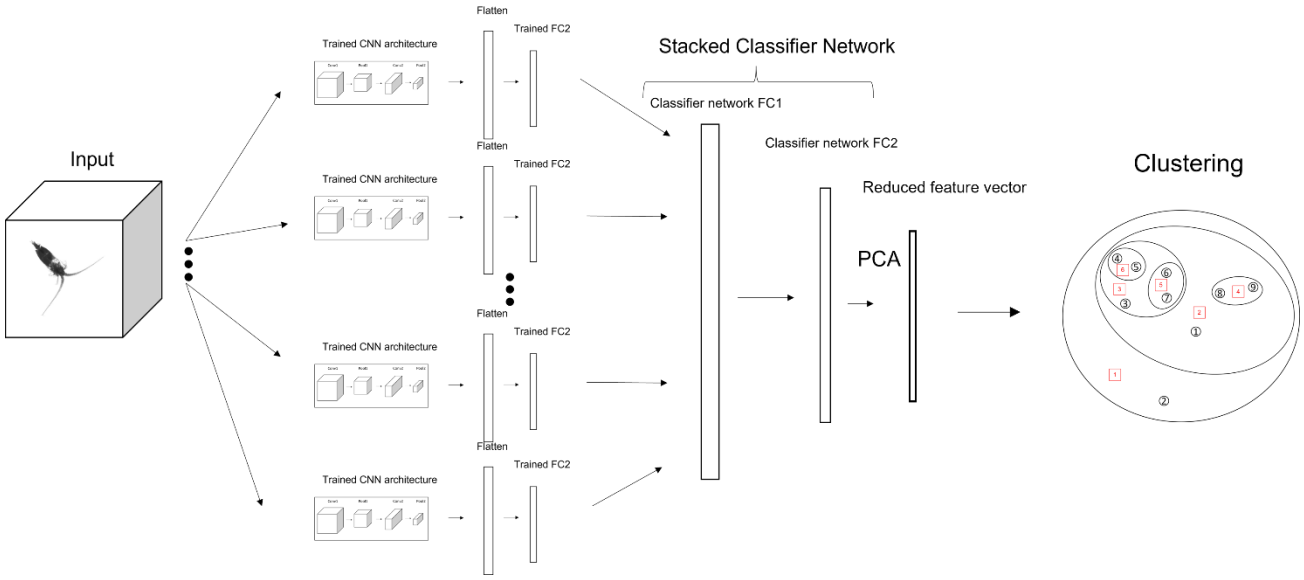

Figure 7.2: Investigation 3 semi-supervised architecture

7.3 Results and Discussion

To determine the performance of the stacked classification network, increasingly more complex datasets were used to train and test the network against the use of singular Wide-Resnet architecture. The Wide Resnet architecture was used to obtain the best score out of all the algorithms tested on the dataset.

The results for the various tests, shown in Table 2, indicate that even with more specific and cleaner datasets, the stacked classification network outperformed the singular network. The stacked classification network was trained on the subphylum dataset, which consists of all the same classes as the cleaner dataset, except that the classes are broken up into their respective subphylum, as discussed in Section 4.6.2. The stacked classification network outperformed the single classification network by 0.097 when tested on the cleaner unseen data. With the inclusion of detritus and other artefacts into the dataset, the performance of the singular network steeply declined, achieving the best accuracy score of 0.667 on unseen data. The stacked classification network, however, still managed to obtain an accuracy of 0.768.

Table 2: Performance test of a stacked classifier network against a singular network on various datasets.

Model	**Dataset**	**Top Accuracy (Train)**	**Top Accuracy (Test)**	**Classes**
Singular classifier network	Clean dataset	0.750	0.731	72
Stacked classifier network	Subphylum datasets	0.843	0.828	72
Singular classifier network	Clean dataset + detritus and noisy artefacts	0.675	0.667	88
Stacked classifier network	Subphylum datasets + detritus and noisy artefacts datasets	0.785	0.768	88
Singular classifier network	Original dataset	0.87	0.12	110
Stacked classifier network	Subphylum datasets + detritus, noisy artefacts and other datasets	0.734	0.712	110

The final performance test between the two architectures involved using the entire original dataset. This dataset is similar to what can be expected in a real-life situation and contains samples from all three type classes discussed in Section 4.2. In this experiment, the stacked classification network obtained a best test accuracy score of 0.712 while the singular classifier network's best performance was 0.12.

The high training score and low testing score obtained by the singular classifier network on the original dataset indicates that the network is overfitting. A section of the confusion matrix produced by testing the resultant single generalized feature extractor on the original dataset is shown in Appendix B. The resultant confusion matrix from the stacked classifier network is also shown in Appendix B.

As indicated in Section 4.7.3 the use of general clustering evaluation metrics is not a good indicator of the performance of density-based clustering methodologies. The preferred metric used to evaluate the density-based clustering method performances is the homogeneity score. This score compares the predicted sample label to the ground-truth label, similarly to how supervised machine learning algorithm performances are benchmarked. However, to obtain a comparable prediction, the automated cluster identification algorithm provides a way to predict the sample's class.

8 Automated Cluster Identification Using Predicted Pseudo-Labels

8.1 Introduction

This final investigation presents the implementation and results using a novel automated cluster identification algorithm for plankton taxonomy. This algorithm is built using all the techniques implemented in Investigations A, B and C. This investigation aimed to prove whether the trained model's predicted labels could be used to identify the resultant clusters. Further details about the implementation of this algorithm are presented in Section 8.2.

8.2 Implementation Details

The first three investigations proposed training individual subclass feature extractors and then combining them with a single feature extraction network which has its output feature vectors further reduced by PCA. This algorithm is then used to supply features to the clustering algorithm, which finds underlying patterns within the data and discover outliers. This investigation uses the stacked classifier network's predicted labels to name clusters that contain a certain threshold abundance of a certain class. The proposed architecture is shown in Figure 8.1.

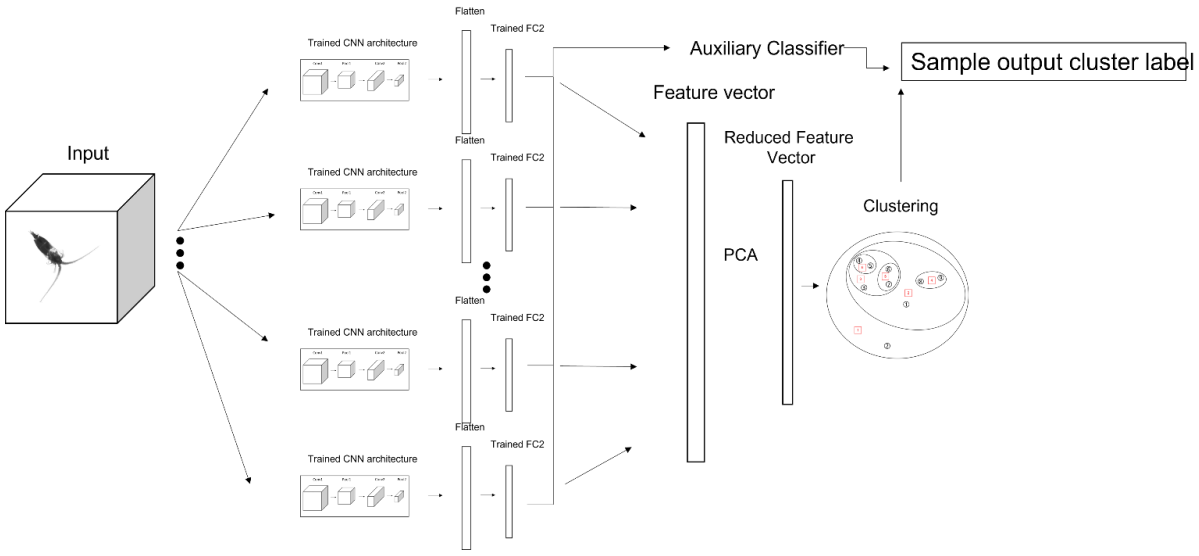

Figure 7.1: Autonomous feature extraction, clustering, and identification algorithm architecture

All the images under consideration have their features extracted and are then clustered using HDBSCAN. The features used to form the clusters are obtained by routing the output of the first FC layer in the classifier network to the clustering algorithm without removing the classifiers output layer. The HDBSCAN clustering algorithm results in a cluster number to which the features of the respective images have been deemed to form part. The stacked classifier network then uses its class prediction output layer to predict the classes of the processed samples. If a certain cluster number contains more than the threshold ratio of samples with the same predicted label, then the cluster number is renamed to the abundant class name. Clusters with the same name are then merged.

The automated clustering algorithm iteratively clusters the extracted datasets and assigns each sample either to noise or the identified cluster. The minimum cluster size parameter is initially set to a high value such as 128. This is so that the clustering algorithm finds the purest and coherent clusters. The minimum cluster size value is decreased with every iteration of the clustering algorithm. As the minimum cluster size decreases, so does the number of samples needing to be clustered. HDBSCAN also provides a membership score which indicates the confidence of the sample belonging to the assigned cluster. To identify the cluster name, the proposed algorithm identifies which sample is dominant in the cluster and assigns all the samples over a certain confidence score to the identified class.

8.3 Results and Discussion

The automated plankton identification algorithm proposed presents a novel method for extracting specific features from plankton samples and then clustering and autonomously labelling the samples. The studies discussed in Section 2.5.1 used a single CNN network as a feature extractor trained on a clean dataset with well-defined classes. These performances are summarized in Table 4. To standardize the performance test, a singular classifier network is trained on the clean dataset proposed in this research. The performance of the automated identification algorithm is compared with the singular classifier model in Table 3. The automated classification algorithm obtained a score of 0.775 when tested on the raw plankton dataset shown in Appendix A. The single classification network used in other research would overfit when trained on the same dataset and couldn't obtain scores higher than 0.12, as shown in Table 2.

Table 3: Results obtained from using automated feature extraction, clustering, and cluster identification algorithm on various datasets

Dataset	**Feature Extraction Network**	**Total Accuracy**	**Accuracy of clustered samples**
Clean dataset	Singular Classifier Network	0.752	0.810
Clean dataset	Stacked Classifier Network	0.828	0.875
Clean dataset + detritus and artefacts	Singular Classifier Network	0.685	0.702
Clean dataset + detritus and artefacts	Stacked Classifier Network	0.768	0.795
Original dataset	Stacked Classifier Network	0.724	0.775

The results indicate that the use of the clustering and the automated cluster identification algorithm increases the performance of the total dataset. The largest impact of the

automated clustering and identification approach is that samples with low confidence of a certain cluster are rejected as noise. These samples are to be hand validated by marine experts. With these noisy samples identified and removed from the testing set, the algorithm performances increase. In the case of the network implemented on the original dataset, this increases the accuracy by 0.051. Section 8.3.1 to Section 8.3.6 review the research objectives and highlight the results obtained.

Table 4: Summary of results obtained by other researchers using CNN models to classify plankton samples

Dataset	**Feature Extraction Network**	**Total Accuracy**	**Number of classes**	**Author**
Clean dataset	Singular Classifier Network	0.886	19	(Correa *et al*., 2016)
Clean dataset	Singular Classifier Network	0.934	13	(Dai *et al*., 2016)
Clean dataset	Singular Classifier Network	0.963	30	(Dai *et al*., 2017)
Clean dataset	Singular Classifier Network	0.73	121	(Li and Cui, 2016)
Clean dataset	Singular Classifier Network	0.88	20	(Lumini, Nanni and Maguolo, 2019)
Clean dataset	Singular Classifier Network	0.85	7	(Cheng *et al*., 2019a)

8.3.1 Research objective one: Investigate existing plankton taxonomy pipelines.

The literature review conducted in Chapter 2 investigated existing plankton taxonomic techniques and discussed their functionality, effectiveness, and shortfalls. The research identified that many modern machine learning techniques used for plankton classification, such as ZooplanktoNet (Dai *et al*., 2016) are only well adapted to clean datasets. This is due to the inability of purely supervised algorithms to overcome this reality gap. Clustering techniques such as the HDBSCAN clustering algorithm used in the MorphoCluster (Schröder, Kiko and Koch, 2020) implementation provide a theoretically sound way to solve the reality problem. HDBCSAN is used to augment the supervised neural network's architectures presented in Chapters 5, 6 and 7. The implementation of the HDBSCAN algorithm is discussed in Chapter 8.

8.3.2 Research objective two: Determine the underlying performance lapses in modern plankton taxonomy techniques.

Morden plankton taxonomy techniques are not considered viable for deployment on realistic datasets. As datasets become more complex, the performance of the investigated algorithms such as ZooplanktoNet (Dai *et al.*, 2016) deteriorates. Several algorithms that could increase the performance of the existing techniques mentioned in Section 8.3.1 were explored and effectively implemented.

The implemented algorithms include the binary classification algorithm discussed in Chapter 5, where a CNN is trained to classify whether a sample is detritus. The subphylum classification algorithm discussed in Chapter 6 proved to be more effective in fine-grained classification than a general all-in-one classifier. In Chapter 7 a stacked classification network was built by adding a fully connected layer on top of the pre-trained models from the first two algorithms investigated in Chapters 5 and 6.

8.3.3 Research objective three: Propose, implement, and evaluate a binary classification CNN for detritus identification

The study of using binary classification techniques for classes such as detritus is introduced in Section 4.5. The full study of the implementation and effectiveness of the techniques is discussed in detail in Chapter 5. The study conducted in Chapter 5 involved constructing a binary neural network architecture that achieved a 0.92 predictive accuracy when classifying between dead and alive samples. Detritus samples make up more than half of most real-life samples. The ability to identify and separate the dead from alive samples allows for better suited implementation of a generalized classification network. This investigation prompted a deeper investigation into whether training networks on specific subsets of data would result in classifiers that perform better than a single generalized network. This is discussed further in Section 8.3.4.

8.3.4 Research objective four: Propose, implement, and evaluate a biological group-based classification CNN for plankton identification

The implementation and results obtained from the study of using group-based classification techniques for all classes instead of one singular network was undertaken in Chapter 6. This investigation separated the larger dataset into biologically grouped subsets. It then measured the performance of individual classifiers trained on the subsets and compared the overall findings against those obtained when training a single classifier on the entire dataset.

Overall, the cleaned, general feature extractor achieved an accuracy of 0.73. The performance of the individual classification algorithms trained on subsets of the same data achieved a perceived accuracy of 0.83. This showed that feature extractors built to identify certain biological or physiological groupings achieve better results than a single generalized feature extractor. This is especially true when the training sets contain noisy samples such as organism body parts, eggs etc.

8.3.5 Research objective five: propose, implement, and evaluate the combination models trained with binary and subclass groupings for plankton identification

Chapter 7 discusses the stacked classification network, which involves training a classification algorithm that uses the outputs of the pre-trained, subclass models mentioned in Sections 8.3.3 and 8.3.4 as inputs. This study showed that the stacked classification network could extract features about plankton samples that singular classification networks could not. A test accuracy of 0.71 was achieved when trained on the uncleaned, full dataset. Whereas for the singular classification architecture, the complex uncleaned dataset forced the network to overfit, resulting in an incomparable outcome.

8.3.6 Research objective six: Propose, implement, and evaluate automated semi-supervised plankton taxonomy pipeline for real-world application

The results from the complete semi-supervised plankton taxonomic pipeline were presented in Chapter 8. The autonomous labelling of the clusters formed from the features extracted by the feature extraction algorithm resulted in a minimum increased accuracy score of 0.065. The automatic plankton taxonomy pipeline implemented outperforms algorithms used in previous studies (discussed in Chapter 2) such as ZooplanktoNet.

. Although the approach taken is similar to that used in MorphoCluster (Schröder, Kiko and Koch, 2020), the novel feature extraction and autonomous cluster labelling algorithm provides augmentations that accelerate the process by not requiring constant human assistance.

9 Conclusions and Future Work

All modern plankton classification techniques underperform when tested on real-life datasets, and this is because plankton sample sets are very biodiverse and noisy. Deep learning techniques such as MorphoCluster use a single convolutional network as a feature extractor and then cluster the resultant features using HDBSCAN. Although this method achieves state-of-the-art results, it is very human dependant and requires manual validation throughout the entire process. This research proposed a novel technique for plankton taxonomy which incorporates the methods used in the MorphoCluster algorithm augmented with enhanced feature extraction and an automated cluster identification technique.

The proposed classification pipeline was built by combining a set of models pre-trained on specific subsets of the data instead of a single network trained on one large dataset. The use of a stacked classification network increased the predictive capabilities of the feature extractor by 0.095, in some cases achieving an accuracy of 0.83 on cleaner datasets. In other cases, such as the real-life dataset, the stacked feature extractor provided a way to achieve an accuracy of 0.71 even when the singular classification could not be trained without overfitting. The autonomous clustering and cluster identification algorithm increased this performance on the raw dataset to 0.789 by rejecting unrecognized and noisy samples for manual classification.

The results achieved in this study indicate that combining models trained on specific subsets of data realized a greater recall accuracy than an individual network trained on the full dataset. Combining this feature extraction technique with a density-based clustering algorithm such as HDBSCAN further increased the recall accuracy and provides a way to identify samples that the algorithm is unsure about.

To further increase the effectiveness of the autonomous taxonomic algorithm, some recommendations for further research include incorporating the “Top 5” score of the classification algorithm into the autonomous cluster identification algorithm discussed in Chapter 8. This score represents the confidence of a network's prediction of each sample's class to the most probable top 5 classes. Instead of just using the top sample, one could form

a new cluster identity ranking system where the cluster's dominant class is built from the top 5 scores of each sample in the cluster's prediction. Another suggestion for further research is that of implementing a deep clustering network that interlocks the clustering process with the supervised feature extraction process. Every training iteration produces cluster seeds that influence both the feature extractor itself and the clustering algorithm. These techniques would further the autonomation of the entire pipeline, allowing for a self-tuning feature extraction and clustering algorithm.

10 References

Ahmad, A. and Khan, S. S. (2019) 'initKmix -- A Novel Initial Partition Generation Algorithm for Clustering Mixed Data using k-means-based Clustering'. doi: 10.13140/rg.2.2.21979.62244.

Arthur, D. and Vassilvitskii, S. (2007) 'K-means++: The advantages of careful seeding', *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 07-09-Janu, pp. 1027–1035.

Bell, J. L. and Hopcroft, R. R. (2008) 'Assessment of ZooImage as a tool for the classification of zooplankton', *Journal of Plankton Research*, 30(12), pp. 1351–1367. doi: 10.1093/PLANKT/FBN092.

Berenbaum, A. (1998) 'Experimental study of performance of minimum spanning tree algorithms'. Available at: https://scholarworks.rit.edu/theses (Accessed: 18 August 2021).

Bi, H. *et al.* (2015) 'A Semi-Automated Image Analysis Procedure for In Situ Plankton Imaging Systems', *PLOS ONE*, 10(5), p. e0127121. doi: 10.1371/JOURNAL.PONE.0127121.

Bittner, L. (1962) 'R. Bellman, Adaptive Control Processes. A Guided Tour. XVI + 255 S. Princeton, N. J., 1961. Princeton University Press. Preis geb. $ 6.50', *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 42(7–8), pp. 364–365. doi: 10.1002/ZAMM.19620420718.

Campello, R. J. G. B., Moulavi, D. and Sander, J. (2013) 'Density-Based Clustering Based on Hierarchical Density Estimates', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7819 LNAI(PART 2), pp. 160–172. doi: 10.1007/978-3-642-37456-2_14.

Chen, W. *et al.* (2019) 'All You Need is a Few Shifts: Designing Efficient Convolutional Neural Networks for Image Classification', *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, pp. 7234–7243. Available at: https://arxiv.org/abs/1903.05285v1 (Accessed: 14 September 2020).

Cheng, K. *et al.* (2019a) 'Enhanced convolutional neural network for plankton identification and enumeration', *PLOS ONE*, 14(7), p. e0219570. doi: 10.1371/JOURNAL.PONE.0219570.

Cheng, K. *et al.* (2019b) 'Enhanced convolutional neural network for plankton identification

and enumeration', *PLOS ONE*. Edited by J. Zhang, 14(7), p. e0219570. doi: 10.1371/journal.pone.0219570.

Chidester, B. *et al.* (2019) 'Rotation equivariant and invariant neural networks for microscopy image analysis', *Bioinformatics*, 35(14), pp. i530–i537. doi: 10.1093/BIOINFORMATICS/BTZ353.

Correa, I. *et al.* (2017) 'Deep learning for microalgae classification', *Proceedings - 16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017*, 2017-Decem, pp. 20–25. doi: 10.1109/ICMLA.2017.0-183.

Dai, J. *et al.* (2016) 'ZooplanktoNet: Deep convolutional network for zooplankton classification', *OCEANS 2016 - Shanghai*. doi: 10.1109/OCEANSAP.2016.7485680.

Dai, J. *et al.* (2017) 'A hybrid convolutional neural network for plankton classification', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10118 LNCS, pp. 102–114. doi: 10.1007/978-3-319-54526-4_8.

Ester, M. *et al.* (1996) 'A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise'. Available at: www.aaai.org (Accessed: 24 November 2020).

'Estuary Education Resources Catching Plankton Estuary Concept' (2012).

Falkowski, P. (2012) 'Ocean science: The power of plankton', *Nature*, 483(7387), pp. S17–S20. doi: 10.1038/483S17a.

Falkowski, P. G. *et al.* (2004) 'The evolution of modern eukaryotic phytoplankton', *Science*. Science, pp. 354–360. doi: 10.1126/science.1095964.

Fernandes, J. A. *et al.* (2009) 'Optimizing the number of classes in automated zooplankton classification', *Journal of Plankton Research*, 31(1), pp. 19–29. doi: 10.1093/PLANKT/FBN098.

File:Ensemble Boosting.svg - Wikimedia Commons (2020). Available at: https://commons.wikimedia.org/wiki/File:Ensemble_Boosting.svg (Accessed: 12 September 2020).

Gorsky, G. *et al.* (2010) 'Digital zooplankton image analysis using the ZooScan integrated system', *Journal of Plankton Research*, 32(3), pp. 285–303. doi: 10.1093/PLANKT/FBP124.

Graphical representation of data output at key stages in the HDBSCAN... | Download Scientific Diagram (2020). Available at: https://www.researchgate.net/figure/Graphical-representation-of-data-output-at-key-stages-in-the-HDBSCAN-algorithm-a_fig17_341893966 (Accessed: 15 September 2021).

Grosjean, P. *et al.* (2004) 'Enumeration, measurement, and identification of net zooplankton samples using the ZOOSCAN digital imaging system', *ICES Journal of Marine Science*, 61(4), pp. 518–525. doi: 10.1016/J.ICESJMS.2004.03.012.

Guérin, J. *et al.* (2021) 'Combining pretrained CNN feature extractors to enhance clustering of complex natural images', *Neurocomputing*, 423, pp. 551–571. doi: 10.1016/j.neucom.2020.10.068.

He, K. *et al.* (2016) *Deep residual learning for image recognition*, *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. doi: 10.1109/CVPR.2016.90.

Hu, Q. and Davis, C. (2005) 'Automatic plankton image recognition with co-occurrence matrices and Support Vector Machine', *Marine Ecology Progress Series*, 295, pp. 21–31. doi: 10.3354/meps295021.

Hua, J. *et al.* (2006) 'Noise-injected neural networks show promise for use on small-sample expression data', *BMC Bioinformatics 2006 7:1*, 7(1), pp. 1–14. doi: 10.1186/1471-2105-7-274.

Huang, G. *et al.* (2016) 'Densely Connected Convolutional Networks', *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua, pp. 2261–2269. Available at: https://arxiv.org/abs/1608.06993v5 (Accessed: 14 September 2020).

Ioffe, S. and Szegedy, C. (2015) 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', *32nd International Conference on Machine Learning, ICML 2015*, 1, pp. 448–456. Available at: https://arxiv.org/abs/1502.03167v3 (Accessed: 14 September 2021).

Jerry, W. (2019) *AlexNet: The Architecture that Challenged CNNs - Towards Data Science*. Available at: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951 (Accessed: 9 April 2020).

Jolliffe, I. T. and Cadima, J. (2016) 'Principal component analysis: a review and recent developments', *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065). doi: 10.1098/RSTA.2015.0202.

Jungan, C. *et al.* (2018) 'A k-Deviation Density Based Clustering Algorithm', *Mathematical Problems in Engineering*, 2018. doi: 10.1155/2018/3742048.

Keister, J. E. *et al.* (2012) 'Zooplankton population connections, community dynamics, and climate variability', *ICES Journal of Marine Science*, 69(3), pp. 347–350. doi: 10.1093/icesjms/fss034.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012) 'ImageNet classification with deep convolutional neural networks', *Communications of the ACM*, 60(6), pp. 84–90. doi: 10.1145/3065386.

Kwak, N. (2008) 'Principal component analysis based on L1-norm maximization', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(9), pp. 1672–1680. doi: 10.1109/TPAMI.2008.114.

Lehr, J. *et al.* (2021) 'Supervised learning vs. unsupervised learning: A comparison for optical inspection applications in quality control', *IOP Conference Series: Materials Science and Engineering*, 1140(1), p. 012049. doi: 10.1088/1757-899X/1140/1/012049.

Léonard, N. *et al.* (2015) 'rnn : Recurrent Library for Torch7'. Available at: https://github.com/Element-Research/rnn (Accessed: 19 September 2020).

Li, X. and Cui, Z. (2016) 'Deep residual networks for plankton classification', *OCEANS 2016 MTS/IEEE Monterey, OCE 2016*. doi: 10.1109/OCEANS.2016.7761223.

Loeb, N. G. *et al.* (2021) 'Satellite and Ocean Data Reveal Marked Increase in Earth's Heating Rate', *Geophysical Research Letters*. doi: 10.1029/2021GL093047.

Lumini, A., Nanni, L. and Maguolo, G. (2019) 'Deep learning for plankton and coral classification', *Applied Computing and Informatics*. doi: 10.1016/j.aci.2019.11.004.

Luo, T. *et al.* (2005) 'Active Learning to Recognize Multiple Types of Plankton', *Journal of Machine Learning Research*, 6, pp. 589–613.

Lv, X. *et al.* (2018) 'CciMST: A Clustering Algorithm Based on Minimum Spanning Tree and

Cluster Centers', *Mathematical Problems in Engineering*, 2018. doi: 10.1155/2018/8451796.

Marpaung, F. and Arnita (2020) 'Comparative of prim's and boruvka's algorithm to solve minimum spanning tree problems', *Journal of Physics: Conference Series*, 1462(1), p. 012043. doi: 10.1088/1742-6596/1462/1/012043.

Matz, M. V. *et al.* (1999) 'Fluorescent proteins from nonbioluminescent Anthozoa species', *Nature Biotechnology*, 17(10), pp. 969–973. doi: 10.1038/13657.

McKinnon, K. M. (2018) 'Flow Cytometry: An Overview', *Current protocols in immunology*, 120, p. 5.1.1. doi: 10.1002/CPIM.40.

Millar, K. *et al.* (2019) 'Using Convolutional Neural Networks for Classifying Malicious Network Traffic', *Advanced Sciences and Technologies for Security Applications*, pp. 103–126. doi: 10.1007/978-3-030-13057-2_5.

Peña, J. M., Lozano, J. A. and Larrañaga, P. (1999) 'An empirical comparison of four initialization methods for the K-Means algorithm', *Pattern Recognition Letters*, 20(10), pp. 1027–1040. doi: 10.1016/S0167-8655(99)00069-0.

Picheral, M., Colin, S. and Irisson, J.-O. (2017) 'EcoTaxa'. Available at: https://ecotaxa.obs-vlfr.fr/ (Accessed: 9 November 2021).

Ramondenc, S. *et al.* (2016) 'An initial carbon export assessment in the Mediterranean Sea based on drifting sediment traps and the Underwater Vision Profiler data sets', *Deep Sea Research Part I: Oceanographic Research Papers*, 117, pp. 107–119. doi: 10.1016/J.DSR.2016.08.015.

Salmaso, N., Naselli-Flores, L. and Padisák, J. (2015) 'Functional classifications and their application in phytoplankton ecology', *Freshwater Biology*, 60(4), pp. 603–619. doi: 10.1111/fwb.12520.

Schmale, D. G. I. *et al.* (2019) 'Perspectives on Harmful Algal Blooms (HABs) and the Cyberbiosecurity of Freshwater Systems', *Frontiers in Bioengineering and Biotechnology*, 0(JUN), p. 128. doi: 10.3389/FBIOE.2019.00128.

Schröder, S.-M., Kiko, R. and Koch, R. (2020) 'MorphoCluster: Efficient Annotation of Plankton Images by Clustering', *Sensors*, 20(11), p. 3060. doi: 10.3390/s20113060.

Shah, A. (2018) *Through The Eyes of Gabor Filter. The Gabor filter, named after Dennis… | by Anuj shah (Exploring Neurons) | Medium*, *Medium.com*. Available at: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97 (Accessed: 11 September 2021).

Sosik, H. M. and Olson, R. J. (2007) 'Automated taxonomic classification of phytoplankton sampled with imaging in-flow cytometry. Limnol. Oceanogr.: Methods 5, 2007, 204–216', *Limnology and Oceanography:*, pp. 204–216. Available at: http://www.whoi.edu/mvco (Accessed: 12 September 2021).

Szegedy, C. *et al.* (2015) *Going deeper with convolutions*, *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. doi: 10.1109/CVPR.2015.7298594.

Underwater Vision Profiler (UVP) - OceanNet (2020). Available at: https://www.ocean-net.es/catalogo/producto/underwater-vision-profiler-uvp/ (Accessed: 9 September 2020).

Weng, J. and Young, D. S. (2017) 'Some dimension reduction strategies for the analysis of survey data', *Journal of Big Data 2017 4:1*, 4(1), pp. 1–19. doi: 10.1186/S40537-017-0103-6.

Yaroslavsky, L. P. (2014) 'Fast Transforms in Image Processing: Compression, Restoration, and Resampling', *Advances in Electrical Engineering*, 2014, pp. 1–23. doi: 10.1155/2014/276241.

Zahran, M. (2021) *A hypothetical example of Multilayer Perceptron Network*, *Researchgate.Net*. Available at: https://www.researchgate.net/figure/A-hypothetical-example-of-Multilayer-Perceptron-Network_fig4_303875065 (Accessed: 13 March 2021).

Zhou, Z., Huang, G. and Wang, X. (2019) 'Ensemble convolutional neural networks for automatic fusion recognition of multi-platform radar emitters', *ETRI Journal*, 41(6), pp. 750–759. doi: 10.4218/ETRIJ.2017-0327.

ZooScan | EMBRC France (2010). Available at: https://www.embrc-france.fr/fr/prestation/zooscan (Accessed: 9 September 2020).

Appendix A: Dataset

Class	Grouping	Count	Class Type (Classes are defined in Chapter 4.2)
Abylidae	Cnidaria	89	3
Acantharea	Harosa	844	3
Acartiidae	Crustacea	1612	3
Actinopterygii	Actinopterygii	314	2
Actinula_Hydrozoa	NEC	4	0
Aetideidae	Crustacea	248	3
Aglaura	Cnidaria	114	3
Alciopidae	Annelida	74	3
Amphipoda	Overall	37	2
Annelida	Overall	214	2
Antenna_Crustacea	Parts	176	2
Appendicularia	Appendicularia	6244	2
Bivalvia_Mollusca	Mollusca	366	3
Bubble	Other	36	2
Calanoida	Overall	74617	2
Calocalanus	Crustacea	751	3
Calocalanus plumulosus	Crustacea	136	3
Calyptopsis_Euphausiacea	Absorbed in Euphausiacea	1853	2
Candaciidae	Crustacea	521	3
Cavoliniidae	NEC	1	0
Centropagidae	Crustacea	242	3
Cephalochordata	NEC	16	0
Cephalopoda	Mollusca	17	3
Chaetognatha	Chaetognatha	9402	2
Cladocera	Crustacea	238	3
Cnidaria_Hydrozoa	Young	145	2
Cnidaria_Metazoa	Young	91	2
Copepoda	Removed	406	0
Copilia	Crustacea	241	3
Corycaeidae	Crustacea	8425	3
Coscinodiscus	Harosa	29	3
Creseidae	Mollusca	403	3
Cyclopoida	Overall	98	2
Cyphonaute	Artefact	433	2
Cypris	Crustacea	41	3
Dead_Copepoda	Other	821	2
Decapoda	Overall	52	3
		67102	1
Diphyidae	Cnidaria	874	3

Doliolida	Thailiacea	3965	3
Echinodermata	Overall	173	2
Egg sac_egg	Egg	30	2
Egg_Actinopterygii	Egg	34	2
Ephyra_Scyphozoa	Cnidaria	8	3
Eucalanidae	Crustacea	397	3
Euchaetidae	Crustacea	706	3
Euphausiacea	Crustacea	3169	3
Euterpina	Absorbed Harpacticoida	19	2
fiber_detritus	Other	1709	2
Firola	NEC	2	0
Foraminifera	Harosa	2154	3
Gammaridea	NEC	10	0
Haloptilus	Crustacea	1016	3
Harosa	Overall	1091	2
Harpacticoida	Crustacea	1144	3
Harpacticoida X sp	Young	91	2
Heterobranchia	Mollusca	61	3
Heterorhabdidae	Heterorhabdidae	58	2
Hydrozoa	Overall	202	2
Hyperiidea	Amphipoda	190	2
Labidocera	NEC	4	0
larvae_Annelida	Young	126	2
larvae_Holothuroidea	Young	133	2
larvae_Mysida	Young	28	2
leg_Crustacea	Parts	653	2
like_egg	Other	1723	2
Limacinidae	Mollusca	1639	3
Lubbockiidae	NEC	5	0
Lucicutiidae	Crustacea	948	3
Mecynocera	Crustacea	301	3
Megalopa	NEC	7	0
Metanauplii_Crustacea	Young	23	2
Molt	Other	59	2
Multiple_Acantharea	Multi	432	2
Multiple_Copepoda	Multi	191	2
Multiple_other	Other	936	2
Mysida	Crustacea	269	3
Narcomedusae	NEC	5	0
Nauplii_Cirripedia	Young	25	2
Nauplii_Copepoda	Young	4668	2
Nauplii_Crustacea	Young	1616	2
Neoceratium	Harosa	169	3
Nereiphylla	Absorbed into Phyllodocidae	54	2
Noctiluca sp	Harosa	102	3

Noctiluca_Noctilucaceae	Harosa	514	3
Oithonidae	Crustacea	14116	3
Oncaeidae	Crustacea	23639	3
Ostracoda	Crustacea	7507	3
Part_Annelida	Parts	38	2
Part_Cnidaria	Parts	120	2
Part_Copepoda	Parts	1574	2
Part_Crustacea	Parts	302	2
Part_other	Other	130	2
Part_Siphonophorae	Parts	118	2
Part_Thaliacea	Parts	12	2
Phaeodaria	Cercozoa	48	2
Phaeogromida	Harosa	1081	3
Phrosinidae	Absorbed in to Hyperiidea	41	2
Phyllodocidae	Annelida	254	3
Phyllosoma	NEC	2	
Pleuromamma	Crustacea	630	3
Pluteus_Echinoidea	Echinodermata	278	3
Pluteus_Ophiuroidea	Echinodermata	146	3
Poecilostomatoida	Crustacea	174	3
Polydora	Annelida	38	3
Pontellidae	Crustacea	1138	3
Protozoea_Penaeidae	NEC	2	
Protozoea_Sergestidae	NEC	1	
Pyrocystis	Harosa	116	3
Pyrocystis noctiluca	Harosa	2170	3
Rhincalanidae	Crustacea	1058	3
Salpida	Thailiacea	865	3
Sapphirina	Crustacea	397	3
Scolecitrichidae	Crustacea	186	3
Sergestidae	Crustacea	68	3
Siphonophorae	Absorbed Hydrozoa	1105	2
Spongodiscidae	Cnidaria	269	3
Spumellaria	Polycystinea	170	2
Squillidae	Crustacea	5	3
Subeucalanidae	Crustacea	2263	3
Tail_Appendicularia	Parts	2375	2
Tail_Chaetognatha	Parts	967	2
Temoridae	Crustacea	436	3
Tomopteridae	Annelida	130	3
Trachymedusae	Absorbed Hydrozoa	28	2
Trichodesmium	Cyanobacteria	61	2
Zoea_Brachyura	Young	19	2
Zoea_Decapoda	Young	56	2

Zoea_Galatheidae	Young	25	2
	Total Number of Samples	272344	

Appendix B: Confusion Matrix Extracts

Table 5: Extract of confusion matrix obtained testing the performance of a singular classification network trained on a raw plankton dataset. (Overfit)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
194	313	151	191	37	227	22	226
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
70	0	0	3	0	53	317	34
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 6: Extract of confusion matrix obtained testing the autonomous plankton taxonomy pipeline trained on a raw plankton dataset.

0	0	0	0	0	3	27	0
0	0	6	7	0	0	0	0
272	0	0	0	0	0	0	22
0	219	0	1	0	0	0	0
0	0	276	0	0	0	0	0
0	0	0	219	0	0	0	0
0	0	0	0	263	0	0	0
0	0	0	0	0	268	30	9
1	0	0	0	0	3	213	0
9	0	0	0	0	13	0	157
13	0	0	0	0	0	0	6
0	0	0	0	0	0	0	0
0	0	0	21	0	0	0	0
0	0	0	0	2	0	0	0
0	13	0	0	0	0	0	0
0	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	4	0	0	0
0	0	0	1	0	0	0	0
0	0	0	3	0	0	0	0
0	20	0	2	0	0	0	0
0	2	0	0	0	0	0	0

Appendix C: Samples of Code

Appendix C presents some highlights of the code used to implement the unsupervised autonomous clustering algorithm.

The square pad method is an image transform used to resize images without changing the aspect ratio or trimming it.

```
class SquarePad:
    def __call__(self, image):
        w, h = image.size
        max_wh = np.max([w, h])
        hp = int((max_wh - w) / 2)
        vp = int((max_wh - h) / 2)
        padding = (hp, vp, hp, vp)
        return transforms.functional.pad(image, padding, 256, 'constant')
```

Padding and other image transforms are used to standardize the inputs to the network.

```
dataDir = 'D:/MastersData/Final Datasets/CleanedSet1'
test_dir = dataDir + '/val'
dirs = {'val': test_dir}

#data transforms for test set
datatransforms = {
    'test': transforms.Compose([
        SquarePad(),
        transforms.Resize(226),
        transforms.CenterCrop(224),
        transforms.Grayscale(num_output_channels=3),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485,0.456,0.406],
                             [0.229, 0.224, 0.225])]),
```

The make weights for balanced classes method are implemented to mitigate the unbalanced nature of the dataset. This method is used to evenly select samples from each class in the dataset and pass the to the DataLoader object.

```
def make_weights_for_balanced_classes(images, nclasses):
    count = [0] * nclasses
    for item in images:
        count[item[1]] += 1
    weight_per_class = [0.] * nclasses
    N = float(sum(count))
    for i in range(nclasses):
        weight_per_class[i] = N/float(count[i])
    weight = [0] * len(images)
    for idx, val in enumerate(images):
        weight[idx] = weight_per_class[val[1]]
    return weight
}
```

Pytorch makes use of DataLoader objects which combine the dataset and sampler and provides an iterable over the dataset.

```
imagedatasets = {x: datasets.ImageFolder(
    dirs[x], transform=datatransforms[x]) for x in [
        'train',
        'val',
        'test']}
weights = {x: torch.DoubleTensor(
    make_weights_for_balanced_classes(
        imagedatasets[x].imgs,
        len(imagedatasets[x].classes))
    ) for x in ['train', 'val', 'test']}
samplers = {x: torch.utils.data.sampler.WeightedRandomSampler(weights[x],
len(weights[x])) for x in ['train', 'val', 'test'] }
batch_size = args.batch
train_loader = torch.utils.data.DataLoader(
    imagedatasets['train'], batch_size=batch_size, shuffle=False,
    sampler=samplers['train'],  drop_last= True)
valid_loader = torch.utils.data.DataLoader(
    imagedatasets['val'], batch_size=batch_size, shuffle=False,
    sampler=samplers['val'], drop_last= True)
test_loader = torch.utils.data.DataLoader(
    imagedatasets['test'], batch_size=batch_size, shuffle=False,
    sampler=samplers['test'], drop_last= True)
dataLoaders = {'train': train_loader,
               'val': valid_loader
               'test': test_loader}
```

The custom multi-layer network takes the standard image size and a list of frozen pretrained models as input parameters and returns a prediction and set of extracted features.

```
class multilayerNet(nn.Module):
    def __init__(self, inputSize, listofmodels):
        super(multilayerNet, self).__init__()
        self.net1 = listofmodels[0]
        self.net2 = listofmodels[1]
        self.net3 = listofmodels[2]
        self.net4 = listofmodels[3]
        self.net5 = listofmodels[4]
        self.net6 = listofmodels[5]
        self.fc1 = nn.Linear(inputSize, 128)
        self.fc2 = nn.Linear(128, 98)
        self.drop = nn.Dropout(0.4)

    def forward(self, x):
        x1 = self.net1(x)
        x2 = self.net2(x)
        x3 = self.net3(x)
        x4 = self.net4(x)
        x5 = self.net5(x)
        x6 = self.net6(x)
        x = torch.cat((x1, x2, x3, x4, x5, x6), dim=1)
        x = F.leaky_relu(self.fc1(x))
        x_feat = self.drop(x)
        x_pred = self.fc2(x_feat)
        return [x_pred, x_feat]
```

A feature and label dataset are built and stored as a pandas DataFrames.

```
classes = imagedatasets['val'].class_to_idx
def get_key(val, dict):
    for key, value in dict.items():
         if val == value:
             return key


    return "key doesn't exist"
print(classes)
featuresset = []
net = fcnet
predslist = []
pathlist = []
net.eval()
with torch.no_grad():
    print("extracting features. . .")
    count = 56842
    for images, labels, paths in dataLoaders['val']:
        images = images.to(device)
        labels = labels.to(device)
        preds, outputs = net(images)
        for index, tensor in enumerate(outputs):
            outs = preds[index]
            path = paths[index]
            pred_probs = [F.softmax(outs,dim=0)]
            _, predicted = torch.max(outs, 0)
            predicted.cpu()
            predslist.append(predicted.item())
            pathlist.append(path)
            theLabel = labels[index].cpu()
            featurebuild = [theLabel.item()]
            featurebuild.extend([predicted.item()])
            featurebuild.extend([path])
            featurebuild.extend(tensor.cpu().numpy())
            featuresset.append(featurebuild)
df = pd.DataFrame(featuresset)
print('features extracted')
print(df)
pd.to_pickle(df, "./featuresFinal2710.pkl")
```

The appropriate scaling and dimension reduction is applied to the set of features to reduce the size of the feature set. These features, the scaling algorithm and the dimension reduction algorithm are saved for consistency in future usage.

```
import pandas as pd
Feats = pd.read_pickle("./featuresFinal2710.pkl")
df = pd.DataFrame(Feats)
labels = Feats[:][0]
predict = Feats[:][1]
pathway = Feats[:][2]
data_values = df.iloc[:,3:]

features_np = np.array(data_values)
scaler = MinMaxScaler()
features_np = scaler.fit_transform(features_np)
transformer = PCA(n_components=80)
featuresPCA = transformer.fit_transform(features_np)
dfFeat = pd.DataFrame(featuresPCA)
dfPCA = pd.DataFrame(list(zip(labels, predict, pathway)))
dfOut = pd.concat([dfPCA, dfFeat], axis=1)
print(dfOut)
with open('PCA_scaled_1027.pkl', 'wb') as fp:
    pickle.dump(dfOut, fp)
dump(transformer, open('transformer1027.pkl', 'wb'))
dump(scaler, open('scaler1027.pkl', 'wb'))
```

The reduced feature set is then clustered using HDBSCAN and autonomously labelled as a certain class based on their feature similarities to other samples in the same cluster. If samples are not classified or are rejected as noise, they are to be re-clustered with less stringent hyper parameters.

```
Feats = pd.read_pickle("PCA_scaled_1027.pkl")
df = pd.DataFrame(Feats)
cluster_sizes = [128]
min_sample = 1
counter = 1
assigned_cluster = []
thresh = 0.96
scaler = StandardScaler()
assignedDF = pd.DataFrame()
for size in cluster_sizes:
    labels = df[:][0].values.tolist()
    predict = df[:][1].values.tolist()
    pathway = df[:][2].values.tolist()
    data_values = df.iloc[:,3:]
    if size == cluster_sizes[0]:
        features_np = scaler.fit_transform(data_values)
    else:
        features_np = scaler.transform(data_values)

    cluster_dominants = {}
    cmodel = hdbscan.HDBSCAN(
            min_cluster_size = size, min_samples=min_sample).fit(features_np)
    clusters = Counter(cmodel.labels_)
    keyList = getList(clusters)
    doms = common_pred(predict, keyList, cmodel.labels_)
    for index, cluster in enumerate(cmodel.labels_):
        dom_in_assigned_cluster = doms.get(cmodel.labels_[index])
        label,_ = labels[index]
        preds,_ = predict[index]
        if (cmodel.probabilities_[index] >= thresh) or
            (predict[index] == dom_in_assigned_cluster):
            assigned_cluster.append(cmodel.labels_[index])
            element = list(zip(labels[index],
                               predict[index],
                               pathway[index],
                               [dom_in_assigned_cluster]))
            dfElement = pd.DataFrame(element)
            assignedDF = assignedDF.append(dfElement)
            df = df.drop(labels=index, axis=0)
        else:
            assigned_cluster.append(-1)
            df.reset_index(drop=True, inplace=True)
```

Custom method implementations are used to determine the dominant class in each cluster.

```
def most_common(List):
    counter = 0
    num = List[0]

    for i in List:
        curr_frequency = List.count(i)
        if(curr_frequency> counter):
            counter = curr_frequency
            num = i

    return num


def getList(dict):
    return dict.keys()

def common_pred(preds_list, cluster_keys, cluster_assignments):
    cluster_dominants = {}
    for key in cluster_keys:
        assigned = []
        for index, assignment in enumerate(cluster_assignments):
            if cluster_assignments[index] == key:
                assigned.append(preds_list[index])
        most_com, _ = most_common(assigned)
        cluster_dominants[key] = most_com
    return cluster_dominants
```