Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Oda Scheen Kiese

Towards a balanced-labeled-dataset of planktons for a better in-situ taxa identification

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl and Aya Saad
June 2020

NTNU
Norwegian University of Science and Technology

Oda Scheen Kiese

Towards a balanced-labeled-dataset of planktons for a better in-situ taxa identification

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl and Aya Saad
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

Studying the dispersion and abundance of plankton organisms in-situ is a driver to recent research activities and oceanography due to their ecological importance. With the introduction of underwater marine robots equipped with sensors and advanced cameras, in-situ identification and classification of underwater microscopic organisms are now possible. Populations of plankton are naturally of different sizes, which is reflected in plankton imagery data sets captured *in-situ*. Commonly, these data sets suffer from class imbalance, i.e. most data examples belong to a few highly represented classes while some classes are ill-represented. Class imbalance impacts the classification performance of deep learning methods like convolutional neural networks (CNNs), as the imbalance can make the classifier biased towards the highly represented classes. Classical approaches to address the issue are resampling strategies and cost-sensitive training. However these methods can lead to overfitting, the introduction of noise and elimination of valuable information. In this thesis we investigate a recent method called GAN-based oversampling, which uses the generative models Generative Adversarial Networks (GANs) to generate synthetic images of planktonic organisms in order to overcome the class imbalance problem. The generated images are used in a synthetic oversampling technique, to balance the class distribution of the data set prior to training a deep neural network. The performance of the GAN and the quality of the generated images is measured by the Fréchet Inception Distance (FID), where a lower score is associated with higher diversity and quality. The method is compared to the frequently used methods random majority undersampling and random majority oversampling, and is evaluated by the classification performance of a CNN called COAPNet. Our main evaluation metric is the F1 score. Based on the results from our experiments we can conclude that partial random oversampling is the superior method. However, GAN-based oversampling can improve classification performance as well and in some cases achieve results equivalent to random oversampling.

Sammendrag

Å studere mangfold og spredning av planktonorganismer *in-situ* er en pådriver for nyere forskningsaktiviteter og oseanografi på grunn av planktons økologiske betydning. Med introduksjonen av marine roboter utstyrt med sensorer og avanserte kameraer, er nå lokal identifisering og klassifisering av mikroskopiske organismer under vann mulig. Populasjonen til ulike planktonarter er naturlig av ulike størrelser, noe som gjenspeiles i datasett av planktonbilder som er fanget *in-situ*. Vanligvis lider disse datasettene av *class imbalance*, dvs. de fleste dataeksempler tilhører noen få høyt representerte klasser mens noen klasser er dårlig representert. *class imbalance* påvirker klassifiseringsytelsen til dype nevrale nettverk, som convolutional neural networks (CNNs), ettersom ubalansen gjør klassifiseringen partisk mot de høyt representerte klassene. Klassiske tilnærminger for å løse problemet er *resampling* strategier og kostnads-sensitiv trening. Imidlertid kan disse metodene føre til *overfitting*, innføring av støy og eliminering av verdifull informasjon. I denne oppgaven undersøker vi en nyere metode kalt GAN-basert oversampling, som bruker de generative modellene Generative Adversarial Networks (GANs), for å generere syntetiske bilder av plankton organismer for å minske effekten av *class imbalance*. De genererte bildene brukes i en syntetisk oversamplingsteknikk for å balansere klassefordelingen av datasettet før trening av et dypt nevralt nettverk (DNN). Ytelsen til GAN og kvaliteten på de genererte bildene måles med Fréchet Inception Distance (FID), der en lavere poengsum er assosiert med større mangfold og kvalitet blant de genererte bildene. Metoden sammenlignes med de ofte brukte metodene tilfeldig *undersampling* og tilfeldig *oversampling*, og evalueres ved klassifiseringsytelsen til en CNN kalt COAPNet. Vår viktigste evalueringsmetrikk er F1-poengsum. Basert på resultatene fra eksperimentene våre kan vi konkludere med at delvis tilfeldig oversampling er den overlegne metoden. Imidlertid kan GAN-basert oversampling også forbedre klassifiseringsytelsen, og i noen tilfeller oppnå resultater som tilsvarer de oppnådd av tilfeldig oversampling.

Preface

This thesis is the Master Thesis in a Master of Science degree in cybernetics and robotics, at the Norwegian University of Science and technology (NTNU). This thesis was done under the Department of Engineering Cybernetics and the AILARON project. It is a continuation from my specialization project.

Based on the work in this thesis and the specialization project, I was invited to orally present the findings at the Ocean Science Meetings 2020, which held place this February in San Diego. The presentation was called ”Towards a Balanced-Labeled-Dataset of Planktons for a Better In-Situ Taxa Identification”, and was presented under the section ”Artificial Intelligence Systems for Advancing the Study of Aquatic Ecosystems I”

I would like to thank Aya Saad and Anette Stahl, from the Department of Engineering Cybernetics, for their support and guidance as supervisors during this project and for their contribution to the preliminary abstract for the Ocean Science Meetings conference.

Table of Contents

Abstract i

Sammendrag i

Preface ii

Table of Contents iv

List of Tables vi

List of Figures viii

Abbreviations ix

1 Introduction 1

2 Theory 3
2.1 Machine Learning . . . 4
2.1.1 Convolutional Neural Networks (CNNs) . . . 6
2.1.2 Evaluation Metrics . . . 6
2.1.3 Class Imbalance Problem . . . 9
2.2 Methods to Address Class Imbalance . . . 10
2.2.1 Data-level Methods . . . 10
2.2.2 Algorithmic-level Methods . . . 16
2.2.3 State-of-the-Art Methods . . . 17
2.3 Generative Adversarial Networks (GANs) . . . 20
2.3.1 Conditional GAN . . . 21
2.3.2 Deep Convolutional GAN (DCGAN) . . . 21
2.3.3 Wasserstein GAN (WGAN) . . . 22
2.3.4 WGAN Gradient Penalty (WGAN-GP) . . . 22
2.3.5 Spectral Normalization GAN (SN-GAN) . . . 23
2.3.6 Evaluation Metrics . . . 23

3 Experiment **27**
3.1 Dataset and Classification Model 28
3.2 Methods of Addressing Imbalance Compared in This Study 29
3.3 Evaluation Metrics and Testing 31

4 Implementation **33**
4.1 Dataset 34
4.1.1 Scaling 35
4.2 COAPNet 38
4.3 GANs 39

5 Analysis **41**
5.1 Effect of Image Resolution and Preprocessing on Classification Performance 42
5.2 Class Imbalance Impact Without Mitigation Methods 43
5.3 Comparison of GAN performance for CGAN-Based Oversampling 45
5.3.1 DCGAN 45
5.3.2 SN-GAN 46
5.3.3 WGAN-GP 47
5.3.4 Choice of GAN 47
5.4 Comparison of Methods to Address Class Imbalance 49
5.4.1 Results from training data with resolution 32x32x3 49
5.4.2 Results from training data with resolution 32x32x1 51
5.4.3 Results from training data with resolution 64x64x3 52

6 Conclusion **55**

Bibliography **55**

Appendix **63**

List of Tables

3.1 Number of images for training, testing and validation) 28

4.1 Number of images associated with each class in the SilCam data set. . . . 34
4.2 Average image resolution for each class 35
4.3 Network architecture for the COAPNet. 38
4.4 Results from training different GAN architectures with different image resolution. 40
4.5 Network architecture for the COAPNet. 40

5.1 COAPNet classification results from different resizing methods, tested on the SilCam dataset with image resolution 64x64x3. 42
5.2 COAPNet classification results with different image resolutions. 42
5.3 Results without class-imbalance mitigation for image resolution 64x64x3. 43
5.4 COAPNet classification results with different image resolutions. 43
5.5 Results from training different GAN architectures with different image resolution. 45
5.6 Results from different class-imbalance mitigation methods for 32x32x3. . 49
5.7 Results for the four important classes, for image resolution 32x32x3. . . . 50
5.8 Results from different class-imbalance mitigation methods for 32x32x1. . 51
5.9 Results for the four important classes, for image resolution 32x32x1. . . . 52
5.10 Results from different class-imbalance mitigation methods for 64x64x3. . 53

6.1 Results without class-imbalance mitigation for image resolution 64x64x1. 63
6.2 Results without class-imbalance mitigation for image resolution 32x32x3. 63
6.3 Results without class-imbalance mitigation for image resolution 32x32x1. 64
6.4 Results of full undersampling for 64x64x3. 64
6.5 Results of full oversampling for 64x64x3. 65
6.6 Results of full oversampling for 32x32x3. 65
6.7 Results of 150 oversampling for 32x32x3. 66
6.8 Results of 50 oversampling for 32x32x3. 66
6.9 Results of full GAN-based oversampling for 32x32x3. 67

6.10 Results of 150 GAN-based oversampling for 32x32x3. 67
6.11 Results of 50 GAN-based oversampling for 32x32x3. 68
6.12 Results of full undersampling for 32x32x3. 68
6.13 Results of full oversampling for 32x32x1. 69
6.14 Results of 150 oversampling for 32x32x1. 69
6.15 Results of 50 oversampling for 32x32x1. 70
6.16 Results of full GAN-based oversampling for 32x32x1. 70
6.17 Results of 150 GAN-based oversampling for 32x32x1. 71
6.18 Results of 50 GAN-based oversampling for 32x32x1. 71
6.19 Results of full undersampling for 32x32x1. 72

List of Figures

2.1 Classification process of supervised machine learning. 4
2.2 Left: Concept of underfitting. Middle: Concept of well fitted model. Right: Concept of overfitting. 5
2.3 An example of a simple feed forward neural network. 5
2.4 A simple CNN architecture, comprised of just five layers. 6
2.5 Concept of the confusion matrix in the binary case. 7
2.6 Left: The process of fully oversampling a dataset. Right: the process of fully undersampling a dataset. 11
2.7 Figure illustrating how SMOTE could contribute to increase overlap between classes. The red samples are samples that could be generated using by SMOTE. 12
2.8 Relationship between generator and discriminator in GAN training process. 20
2.9 Structure of a Conditional GAN. 21
2.10 SN-GAN trained on Cifar-10 dataset generated images with corresponding FID score. 26

3.1 Class distribution of the imbalanced dataset SilCam. 28

4.1 Class distribution for all classes from the SilCam dataset 34
4.2 Shows example images for each class. 1: Bubble, 2: Copepod, 3: Faecal Pellets, 4: Diatom Chain, 5: Fish egg, 6: Oil, 7: Oily gas, 8: Other 36
4.3 Image rescaled using different rescaling techniques 36
4.4 Example images of the SilCam after scaling is applied, where each row represents a class. 37

5.1 Confusion matrices after training COAPNet with the original imbalanced dataset, for different image resolutions. 44
5.2 FID score achieved for each checkpoint during training of DCGAN with different image resolutions. 45
5.3 Images generated by the DCGAN with the corresponding FID score achieved. 46

5.4 FID score achieved for each checkpoint during training of SN-GAN with different image resolutions. 46
5.5 Images generated by the SN-GAN with the corresponding FID score achieved. 46
5.6 Results from training the WGAN-GP on images with resolution 32x32x3. 47
5.7 Example images with resolution 64x64x3, where each row represents a class from the SilCam dataset. 48
5.8 Confusion matrices for for the different oversampling methods, for image resolution 32x32x3. 50
5.9 Confusion matrices for for the different oversampling methods, for image resolution 64x64x3. 52

Abbreviations

AI	=	Artificial intelligence
ANN	=	Artificial neural network
CBO	=	Cluster-based versampling
CGAN	=	Comvolutional GAN
CNN	=	Convolutional neural network
DCGAN	=	Deep CGAN
ENN	=	Edited nearest neighbor
FID	=	Fréchet inception distance
GAN	=	Generative adversarial network
IS	=	Inception score
KNN	=	K-nearest neighbour
LMLE-kNN	=	Large Margin Local Embedding kNN
ML	=	Machine learning
MLP	=	Multi layered perceptron
NCL	=	Neighbouring cleaning rule
OSS	=	One-sided selection
ReLU	=	Rectified linear unit
SMOTE	=	Synthetic Minority Over-sampling Technique
SN-GAN	=	Spectral normalization GAN
WGAN	=	Wasserstein GAN
WGAN-GP	=	WGAN gradient penalty
AUV	=	Autonomous underwater vehicles
TFDS	=	TensorFlow Datasets

Chapter 1

Introduction

Planktonic organisms are of fundamental ecological importance and are some of the planet's most critical organisms. They are the base of the ocean food web, produce an estimated 80% of the world's oxygen and play a vital role in many biochemical cycles such as the ocean's carbon cycle (Witman, 2017). As such, scientists are increasingly studying planktonic organisms and their importance. With imaging-based technology such as underwater marine robots equipped with sensors and advanced cameras in-situ identification and classification of underwater microscopic organisms are now possible (Sieracki et al., 2010; Jaffe, 2014), and is also considered an important task (MacLeod et al., 2010). However, the classification of plankton has proven to be a difficult task (Benfield et al., 2007; Wang et al., 2017).

The population of some planktonic species are naturally larger than that of others. With this difference in population size, when images of plankton are captured *in-situ*, this difference will be reflected in the resulting data set. By labeling each image by its plankton species and associate all images with the same label to the same *class*, we get a data set with imbalanced class distributions. The data set is said to suffer from class imbalance, i.e. some classes are represented by a high number of examples while other are represented by only a few. An example of such a data set is the publicly available WHOI-Plankton dataset (Orenstein et al., 2015) which consists of hundred of thousand of images, yet 81% of all examples belong to one class. Even though Convolutional Neural Networks (CNNs) have achieved state of the art performance for image classification (Deng et al., 2009; Buda et al., 2018) , the fine-grained and imbalanced nature of in-situ planktonic data sets makes classification a difficult task.

For imbalanced data sets, the highly represented classes are referred to as majority classes while the those that are ill-represented are referred to as minority classes. The difficulties related to imbalanced classification is that classifiers tend to favour the majority classes, and have poor performance when it comes to predicting minority class examples. This is known as the class imbalance problem. The problem origins in the design of machine learning (ML) algorithms, as most of them are designed to maximize overall accuracy, without taking relative class distributions into consideration.

This is a highly researched and documented topic for classical machine learning models, and methods of dealing with imbalance are well studied (Japkowicz and Stephen, 2002). The most common and straightforward approach is to use resampling methods. These methods work on the data, rather than the machine learning model itself, in order to balance the class distribution. The most intuitive strategy is called undersampling (Dal Pozzolo et al., 2015), which simply removes examples from the majority classes until all classes inhabits the same number of examples. Oversampling (He and Garcia, 2009) is a strategy which does the opposite, instead of removing examples, minority class examples are replicated until balance is achieved. The class imbalance problem can also be tackled by working on the machine learning model. The class distributions are left unchanged, and instead these methods forces the classifier to prioritize correctly classifying the minority classes (Jo and Japkowicz, 2004). Although both groups of methods have shown to improve classification performance in some cases, they have their drawbacks. Sampling methods can lead to discarding of valuable data or overfitting, and methods working directly with the ML model can be unfeasible in some cases.

The methods previously mentioned are well studied for classical machine learning methods. With the rise of artificial neural networks (ANNs) the recent years new methods to tackle class imbalance have been invented. In this thesis we investigate the use of a framework for estimating generative models called generative adversarial networks (GANs) by (Goodfellow et al., 2014) to mitigate the class imbalance problem. GANs are deep neural network architectures which can learn to generate synthetic data similar to the data they are trained on. They consists of two neural networks: a discriminative model (D) and a generative model (G). During the training process of the GAN the G model generates synthetic data examples while the D model evaluates given examples for authenticity; i.e. the probability whether a given instance is generated or from the training data. The objective of G is to maximize the probability of D misclassifying a generated example as a real one, while D tries to maximize its accuracy. In this process the G model learns to generate data similar to the training data.

After the invention of GANs, several improvements and different versions has emerged. The first improvement was conditional GANs (Mirza and Osindero, 2014), which allows the possibility of generating synthetic data based on a desired label. Another notable version of GANs is deep convolutional GANs (DCGANs) (Radford et al., 2015), which uses CNNs for the discriminator and generator making it possible to train GANs on images.

By training a GAN on a planktonic data set, the generator will learn to generate synthetic images of plankton, similar looking to the original images. This makes is possible to use GANs for an oversampling procedure, called GAN-based oversampling. In this thesis we investigate this method and compares it to other sampling techniques.

The remainder of this thesis is organized as follows. Chapter 2 gives an overview of the theory behind ML, GANs and class imbalance mitigation methods. In chapter 3 the experimental setup is described and in chapter 4 implementation of the setup is elaborated. Then, in chapter 5 the results from the experiments are presented and finally chapter 6 concludes the thesis.

Chapter 2

Theory

In this chapter we will first go through some fundamentals of machine learning, such as convolutional neural networks and common evaluation metrics as well as introduce the *class-imbalance problem*. Then the class-imbalance problem will be elaborated along with common class imbalance mitigation methods. Lastly we will go through generative adversarial networks, and explain how they can be used to tackle the class imbalance problem.

2.1 Machine Learning

Machine learning (ML) is defined as an automated process that extracts patterns from data (Kelleher et al., 2015), and is seen as a subset of artificial intelligence (AI). Machine learning algorithms has the ability to build a mathematical model based on sample data, in order to make decisions or predictions without being explicitly programmed to do so (Koza et al., 1996). ML has become exceedingly popular in the recent years and is used in a wide variety of applications. To ML build models for classification purposes we usually use supervised machine learning. Supervised ML techniques automatically learn a model of the relationship between a set of descriptive features and a target feature based on a set om sample data. This sample data is refereed to as training data, where each example (descriptive feature) in the data set has an associated label (target feature). All examples associated with the same label, belong to the same class. Other types of machine learning include unsupervised learning, reinforcement learning and semi-supervised learning. However, in this thesis we focus exclusively on supervised machine learning and use the term machine learning and supervised machine learning interchangeably.

Fig. 2.1 shows the concept of supervised machine learning with the purpose of classification, using a data set consisting of images as example. Labeled training data is given to the machine learning algorithm. After the training process examples without a label can be given to the trained model, and it will give a prediction of what class the example belongs to. In order to evaluate the performance of the ML model, also called classifier, a

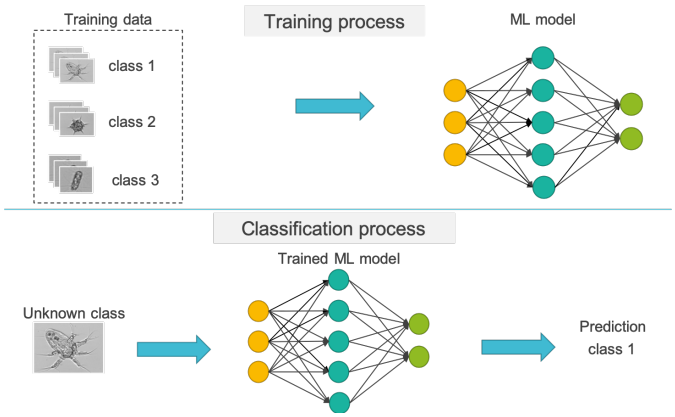


Figure 2.1: Classification process of supervised machine learning.

test data set is used. The test data set do not contain examples found in the training data set in order to provide unbiased evaluation of the model. It is important to provide a test set with exclusive data, as some ML algorithms has a tendency to *overfit*. Overfitting is a problem that occurs when the model ”memorizes” the training data and is not able to generalize. To the right in fig. 2.2 the concept of overfitting is shown. To the left in the figure we see an other concept called *underfitting*. This occurs when the model has not been able to extract some important information from the data set.

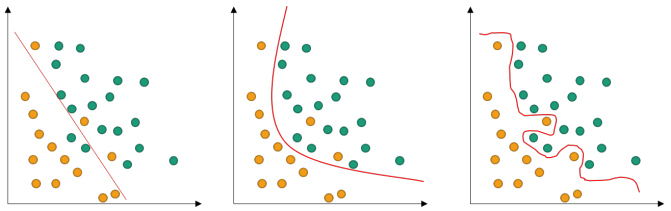


Figure 2.2: Left: Concept of underfitting. Middle: Concept of well fitted model. Right: Concept of overfitting.

Even though the concept of machine learning is an old one, it has been limited by computational power. With technological innovation the interest and research devoted to ML has increased accordingly. Especially the interest in Artificial Neural Networks (ANN), which are a set of machine learning algorithms inspired by the human brain. An example of ANN is the multi layered perceptron (MLP), however the two names ANN and MLP are often used interchangeably. From its name suggests it consists of layers, more specifically an input layer, minimum one hidden layer and an output layer. An example of a MLP network is shown in fig 2.3. Each layer consists of *neurons*, which are connected

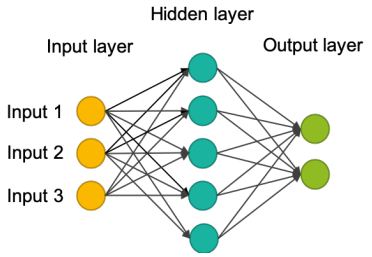


Figure 2.3: An example of a simple feed forward neural network.

to neurons in the next layer. These neurons are the building blocks of the MLP, and are simple computational units. They have weighted input signals and and produce an output using activation functions. The network architecture is represented by the weight matrix $\mathbf{W} = [w_{ij}]$, where w_{ij} denotes the weight from neuron i to j (Du and Swamy, 2013). During the training of the network, these weights are updated and in this way the network learns the features of the training data. With the rise of ANN in recent times, these models are able to far exceed the performance of previous machine learning algorithms (O'Shea and Nash, 2015)

2.1.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a subset of ANN, which are primarily used to solve tasks of image-driven pattern recognition. CNNs are very similar to ANNs and are compromised of neurons that self-optimize through learning, however, it is optimized for pattern recognition within images. The main difference between the two, is that the neurons in a CNN are organized in three dimensions, the spatial dimensionality of the input (width and the height) and the depth. CNNs are comprised of three different type of

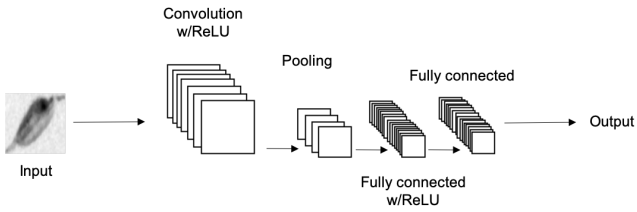


Figure 2.4: A simple CNN architecture, comprised of just five layers.

layers. These are convolutional layers, fully-connected layers and pooling layers (O'Shea and Nash, 2015). When these different layers are stacked a CNN architecture has been formed. A simplified CNN architecture for image classification is illustrated in fig. 2.4. The input layer in a CNN hold pixel values of the given image. A convolutional layer determines the output of neurons through calculation of the local input and its weight using rectified linear unit (ReLU). The pooling layer performs downsampling, reducing the number of parameters. The fully-connected layers attempts to produce class scores from its given input.

2.1.2 Evaluation Metrics

The performance of a classification algorithm can be visualized by a confusion matrix. When classifying a labeled test data set, the confusion matrix gives a summary of the prediction results with the number of correct and incorrect predictions for each class. It gives insight to the errors made by the classifier, the type of errors made and if the model is confusing classes. The confusion matrix is a special kind of table with the two dimensions *actual* and *predicted*, and the set of classes in both dimensions. Fig. 2.5 shows the confusion matrix in abstract terms in the binary case, where the columns are the actual labels and the rows are the predicted labels. From the confusion matrix one can calculate number of evaluation metrics. The *accuracy* achieved by the classifier is the total fraction of correctly labeled examples and is given by

$$Acc = \frac{\sum(\mathrm{TP} + \mathrm{TN})}{\sum(\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN})}. \tag{2.1}$$

As the accuracy only counts the number of correctly labeled examples, it can be misleading when evaluating performance. An example is the case where 95% of the data belong

		Actual	
		Positive	Negative
Predicted	Positive	True positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Figure 2.5: Concept of the confusion matrix in the binary case.

to class 1, while class 2 only consists of 5% of the data. If the classifier predicts all of the examples to be class 1 then it would achieve an accuracy of 95% even though it mislabeled every example from class 2. In order to get a better understanding of the classifiers performance one could look at precision and recall. *Precision* is the fraction of relevant examples among the retrieved examples and is given by

$$Pr = \frac{\sum \text{TP}}{\sum(\text{TP} + \text{FP})}, \tag{2.2}$$

while *recall* is the fraction of the total amount of relevant examples retrieved and is given by

$$Re = \frac{\sum \text{TP}}{\sum(\text{TP} + \text{FN})}. \tag{2.3}$$

If the precision is low this can indicate a large number of false positives, while a low recall can indicate many false negatives. As such these two metrics offer additional information about the classifiers performance. *F1* score is the harmonic mean of recall and precision and is given by

$$F1 = \frac{2}{Re^{-1} + Pr^{-1}} = 2\frac{Pr * Re}{Pr + Re}. \tag{2.4}$$

It gives the balance between precision and recall and is a widely used evaluation metric for comparing classifier performance.

In the case of a multi-class data the precision, recall and F1 score is calculated for each class. There are different ways of doing so, where the simplest is taking the arithmetic mean of the per-class scores. This is called the macro-average

$$Pr_{macro} = \frac{1}{q}\sum_{\lambda=1}^{q} Pr_{\lambda}, \quad Re_{macro} = \frac{1}{q}\sum_{\lambda=1}^{q} Re_{\lambda}, \quad F1_{macro} = \frac{1}{q}\sum_{\lambda=1}^{q} F1_{\lambda} \tag{2.5}$$

where the λ is a label and $L = \{\lambda_j : j = 1...q\}$ is the set of all labels. In this way of calculating, all classes are given the same weight. If the classes are weighted, then we get

the micro-average/weighted average

$$Pr_{micro} = \frac{\sum_{\lambda=1}^{q} Pr_\lambda n_\lambda}{N} \quad Re_{micro} = \frac{\sum_{\lambda=1}^{q} Re_\lambda n_\lambda}{N} \quad F1_{micro} = \frac{\sum_{\lambda=1}^{q} F1_\lambda n_\lambda}{N} \tag{2.6}$$

where n_λ is the number of images that exists in class λ and $\sum_{\lambda=1}^{q} n_\lambda = N$. This gives that $Re_{micro} = Pr_{micro}$.

If the data is imbalanced, that is most data belong to one or a few classes, the micro- and macro-average could give fairly different results (Van Asch, 2013). The micro-average is biased towards the most populated ones while the macro-average is biased towards the least populated ones. If the micro-average is significantly lower than the macro-average, that would mean high misclassification rate in the most populated classes. If it is significantly higher it would mean the smaller classes are poorly classified.

2.1.3 Class Imbalance Problem

In the field of machine learning the *Class Imbalance Problem* is encountered when the training data suffer class imbalance, which means you have some classes with high representation (majority classes) while others have low representation (minority classes) (Japkowicz, 2000). As data captured from the real-world often exhibit class imbalance, this problem is of high significance for many domains such as diagnosis of medical conditions (Grzymala-Busse et al., 2004; Mac Namee et al., 2002), fraud detection (Kaggle; Fawcett and Provost, 1997) and plankton classification (Wang et al., 2017; Lee et al., 2016). Most classical machine learning methods such as tree induction systems or multi-layered perceptrons are not designed to consider the relative class distribution during the training process (Jo and Japkowicz, 2004). These methods have the objective of maximizing overall accuracy, which seen from the previous section is not necessarily a good evaluation metric for classification performance in the case of imbalanced learning. The methods can easily be biased towards the majority classes as a high accuracy can be achieved by simply neglecting the minority classes. If the training data suffer class imbalance this can have significant unfavorable effect on performance, affecting both the model's capability of generalizing and convergence during training (Ling and Sheng, 2010; Japkowicz, 2000; Buda et al., 2018). The topic has been heavily researched and documented for classical machine learning methods. A systematic study of problem and mitigation methods was done by (Japkowicz and Stephen, 2002), and they concluded that the it is a relative problem that depends on; the degree of imbalance, the complexity of the training data, the size of the training set and the classifier involved. Deep Their study did not involve deep learning models. A systematic study of the impact class imbalance has on classification performance of CNNs were done by (Buda et al., 2018), and they concluded that CNNs are greatly affected, and the effect could be detrimental on classification performance.

2.2 Methods to Address Class Imbalance

There a numerous of different approaches to mitigate the impact of the class imbalance problem. The methods can be divided into two main categories; algorithmic-level methods and data-level methods (He and Garcia, 2009). Methods that work at the data-level attempts to balance the data set prior to the training process of the classifier. The methods typically consists of strategies that resamples the training data on order to achieve balanced class distributions. Algorithmic-level methods take on another approach to address the class imbalance issue. The training data and class distributions are left unchanged, and instead they consider how the classifier is handling the imbalanced data. These methods involve strategies which forces the classifier to prioritize correctly classifying the minority classes.

For presentation of the different class imbalance mitigation methods we have to establish some notations. With the training data set S with m examples (i.e. $|S| = m$), we have: $S = \{(\mathbf{x}_i, y)\}, i = 1, ..., m$ where $\mathbf{x}_i \in X = \{f_1, f_1, ..., f_n\}$ is an instance in the feature space X and $y_i \in Y = \{1, ..., C\}$ is the class label associated with $\mathbf{x}_i$ where C is the number of classes in the training data. We also define the subsets $S_{maj} \subset S$ and $S_{min} \subset S$ so that S_{maj} is the majority class instances of S and S_{min} is the minority class instances of S. This gives $S_{min} \cup S_{maj} = \{S\}$ and $S_{min} \cap S_{maj} = \{\emptyset\}$. Sets generated from sampling of S are labeled E, where E_{maj} and E_{min} represent majority and minority samples of E, respectively.

2.2.1 Data-level Methods

Random Oversampling and Undersampling

Random minority oversampling is a method that balances the dataset by adding a set E sampled from the minority class to the dataset (He and Garcia, 2009). In an unsupervised manner examples are randomly selected with replacement from S_{min}, replicated and added to S. This increases the total number of examples in S_{min} by $|E|$, and adjusts the class distribution of S accordingly. The number of examples to resample, $|E|$, can be adjusted to achieve the desired degree of class distribution balance.

Random majority undersampling is similar, but instead of appending examples to the training data this methods removes examples to achieve a more balanced class distribution (Dal Pozzolo et al., 2015). The method is based on the idea that S_{maj} consists of many redundant examples. Thus by removing examples from S_{maj} at random will not change its distribution significantly. A set E is randomly selected from instances in S_{maj} and are then removed from S so that the total number of examples left is $|S| = |S_{min}| + |S_{maj}| - |E|$. Again $|E|$ can be adjusted, and thus this is a simple method to modify the class balance of the training data S.

Fig. 2.6 shows the concept of random oversampling and -undersampling, where $|E|$ is chosen such that $|S_{min}| = |S_{maj}|$ after the resampling. Random resampling are similar methods, but each method introduces its own set of problematic consequences. The obvious drawback of undersampling is that discarding examples at random from the majority class may cause the classifier to miss key concepts that could lead to poor performance. Random oversampling has a big drawback not as obvious; the replication of minority

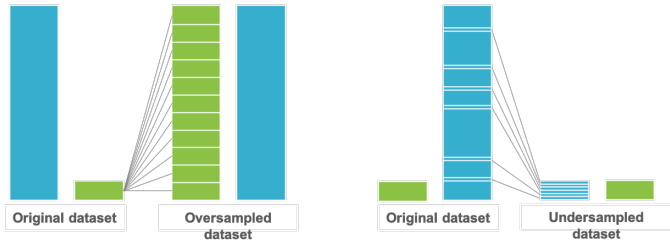


Figure 2.6: Left: The process of fully oversampling a dataset. Right: the process of fully undersampling a dataset.

examples can lead to overfitting for some machine learning algorithms. However, a systematic study of the impact of class imbalance on CNNs were done by (Buda et al., 2018), which concluded that oversampling outperformed undersampling, and that the issue of overfitting does not apply to CNNs in the case of oversampling.

Informed Undersampling

As random undersampling potentially could remove important examples, there has been several attempts at methods that overcome this deficiency by undersampling in an informed instead of random manner. Two such methods are *EasyEnsemble* and *BalanceCascade* by (Liu et al., 2008). Both strategies benefits from exploring the majority class examples, more specifically those that would otherwise be ignored by random undersampling that is $S_{maj} \cap \bar{E}$.

In contrast to random undersampling, EasyEnsemble samples several subsets E_i from the majority class S_{maj}, where $i = 1, 2, ..., T$ and T is the number of subsets and $|E_i| = |S_{min}|$. Each sample is combined with the minority class examples $E_i \cup S_{min}$ and independently used as training data for a classifier H_i, developing an ensemble learning system of multiple classifiers. All classifiers $H_1, H_2, ..., H_T$ are combined for the final decision. In this way, EasyEnsemble explores the majority class in an unsupervised manner by using random sampling with replacement.

BalanceCascade is similar to EasyEnsemble, but contrastingly takes on a supervised approach in the exploration of the majority class, by selecting which majority class examples to sample. After H_i is trained, if the classifier correctly classifies an example $x_i \in S_{maj}$, this example is seen as redundant in S_{maj} given that H_i is already trained. After the training of H_i all correctly classified majority examples are removed from S_{maj}, to avoid them from being sampled in another iteration. In this way, BalanceCascade is able to explore the majority class in a higher degree than EasyEnsemble. The final classifier of BalanceCascade also differ from EasyEnsemble, as it is cascade classifier which is the conjunction of all $\{H_i\}_{i=1,...,T}$, which means the final classifier $H(\mathbf{x})$ only gives a positive prediction if and only if all $H_i(\mathbf{x}), i = 1, ..., T$ comes to the same conclusion.

Another attempt at informed undersampling are the *NearMiss-methods* proposed by (Mani and Zhang, 2003), which uses a K-nearest neighbour (kNN) classifier for undersam-

pling. They proposed four different methods, the NearMiss-1, NearMiss-2, NearMiss-3, and the "most distant" method. The first method, NearMiss-1, selects majority examples that are close to some of the minority examples. More specifically it selects majority examples that have the smallest average distance to the three closest minority examples. The NearMiss-2 selects majority examples that are close to all minority examples, by selecting those that have the smallest distance to the three farthest minority examples. For NearMiss-3 each minority example is considered and a given number of its closest majority examples are selected. Lastly, the "most distance" method selects the majority examples who has the largest average distance to the three closest minority examples. In their results the NearMiss-2 method was the most promising, outperforming the other three methods as well as random undersampling.

Some other examples of informed examples are *One-Sided Selection*, *neighbouring cleaning rule*, *Relabel* and *Remove* which are further explained in section 2.2.1.

Synthetic Sampling

There is a set of methods that generates synthetic examples rather than simply replicating minority class examples. One such widely popular method is *Synthetic Minority Oversampling Technique* (SMOTE) by (Chawla et al., 2002), which was inspired by the fact that random oversampling with replacement doesn't necessarily improve minority class recognition. They argue that by replicating minority class examples, in some cases the decision boundary for the minority class does not change, which in return leads to the same misclassification of minority examples as before oversampling. The SMOTE method oversamples the minority class S_{min} by introducing a synthetic example for each minority example $\mathbf{x}_i \in S_{min}$ along the line-segment between all/some of the K minority class nearest neighbours. The number of generated examples per minority class example is optional. The synthetic example is generated by randomly selecting one of $\mathbf{x}_i$'s K nearest neighbours $\hat{\mathbf{x}}_i$, multiply the feature vector $\hat{\mathbf{x}}_i - \mathbf{x}_i$ with a random number between $c \in [0, 1]$, and finally, add the result to $\mathbf{x}_i$.

$$\mathbf{x}_{\text{synthetic}} = \mathbf{x}_i + c(\hat{\mathbf{x}}_i - \mathbf{x}_i) \tag{2.7}$$

(Chawla et al., 2002) argues that with their approach the decision boundary is forced

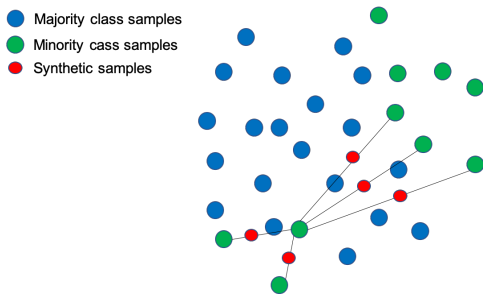


Figure 2.7: Figure illustrating how SMOTE could contribute to increase overlap between classes. The red samples are samples that could be generated using by SMOTE.

to be more general. However, the disadvantage of SMOTE is the possibility of over-generalizing by increasing the overlap between S_{maj} and S_{min}, consequently shifting the decision boundary of the minority class to a less desired state. The concept if SMOTE introducing increased class-overlap is shown in fig. 2.7.

Adaptive synthetic sampling

As the traditional SMOTE algorithm has the disadvantage of over-generalizing, there has been proposed various different adaptive sampling methods. One of them was proposed by (Han et al., 2005) and is an improvement of the original SMOTE method called *Borderline-SMOTE*. They attribute the limitation of SMOTE to the way it is indifferent to the placement of neighbouring majority class examples when generating synthetic examples of a minority example. They argue that examples far from the decision boundary between the minority and majority class have little contribution to classification, and thus introduce their improved SMOTE in which only borderline examples of the minority class are over-sampled. The borderline examples are found as follows: For each $x_i \in S_{min}$ the m nearest neighbours from the whole training set S is calculated, where $m' \in [0, m]$ is the number of majority examples among the neighbours. If $m' = m$ then x_i is considered noise. If less than half the neighbours are majority examples, $[0 \leq m' < m/2]$, x_i is considered safe but if more than half of the neighbours are majority examples, $[m/2 \leq m' < m]$, x_i is considered to be easily misclassified and put in a set called DANGER. The DANGER set contains the minority borderline examples and is a subset of the minority class DANGER $\subset S_{min}$. The examples in DANGER are denoted x'_i which gives

$$\text{DANGER} = \{x'_1, x'_2, ..., x_d\}, \quad |\text{DANGER}| = d \in [0, |S_{min}|]$$

After the DANGER set has been established, Borderline-SMOTE runs the original SMOTE algorithm on every example in the set. This generates $s * d$ synthetic examples based on the examples in DANGER, where s is an integer between 1 and K.

Another approach for adaptive synthetic sampling is *ADASYN* proposed by (He et al., 2008). Unlike Borderline-SMOTE which generates data for borderline minority examples, ADASYN looks at the level of difficulty in learning for minority examples and generates data for those that are harder to learn. This is done as follows: First the total number of synthetic examples to generate is calculated

$$G = \beta(|S_{maj}| - |S_{min}|), \tag{2.8}$$

where $\beta \in (0, 1]$ specifies the desired level of imbalance after the resampling process i.e. $\beta = 1$ will give a fully balanced dataset. Then, for each minority example $x_i \in S_{min}$ the K nearest neighbours are found and the ratio r_i is calculated by

$$r_i = \frac{\delta_i}{K}, \quad i = 1, ..., |S_{min}|, \tag{2.9}$$

where δ_i is the number of majority examples among the K nearest neighbours. r_i then is normalized according to

$$\hat{r_i} = \frac{r_i}{\sum_{i=1}^{|S_{min}|} r_i}, \tag{2.10}$$

to make $\hat{r_i}$ a density function $\sum_i \hat{r_i} = 1$. The number of synthetic examples to generate for each minority example x_i is then given by

$$g_i = \hat{r_i} * G, \tag{2.11}$$

where G is defined by equation 2.8. ADASYN works as follows: for each minority example x_i the number of synthetic examples to generate g_i is calculated. The generated examples $x_{synthetic}$ is calculated by the same equation as SMOTE uses given by eq. 2.7, where $\hat{x}_i$ is one of the K nearest neighbours of x_i chosen at random. ADASYNs generation of synthetic examples helps reduce the bias introduced by the imbalanced class distribution, and can also shift the decision boundary and make the classifier more focused on the examples that are difficult to learn.

Sampling with Data Cleaning

Sampling methods can introduce data points which contribute to class-overlap in the resampled data set. Data cleaning methods such as *Tomek links* (Tomek et al., 1976), can be an efficient way to reduce this overlap. Tomek links can be defined as a pair minimally distanced neighbours from two opposite classes. Given a pair $(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{x}_i \in S_{min}$ and $\mathbf{x}_j \in S_{maj}$ with $d((\mathbf{x}_i, \mathbf{x}_j))$ as the distance between the two examples, then the pair is considered a Tomek link if there exists no example $\mathbf{x}_k$, such that

$$d(\mathbf{x}_i, \mathbf{x}_k) < d(\mathbf{x}_i, \mathbf{x}_j) \quad \text{or} \quad d(\mathbf{x}_j, \mathbf{x}_k) < d(\mathbf{x}_i, \mathbf{x}_j)$$

If two examples form a Tomek link then both examples are near the class border or one of the examples are noise. By locating all Tomek links after synthetic oversampling one can perform data cleaning by removing all Tomek links until all minimally distanced neighbours are from the same class, and thus remove the unwanted class overlap. This way, well-defined class clusters can be defined in the training set which could help during classification.

A method which uses Tomek links to perform informed undersampling is *One-Sided Selection* (OSS) by (Kubat et al., 1997). The method identifies each example in the dataset as either *noise*, *borderline*, *redundant* or *safe*, and only samples examples categorized as safe. Borderline and noisy examples are found using Tomek links. There has also been made a integration of SMOTE and Tomek links called *SMOTE+Tomek* by (Tomek et al., 1976). A third method using data cleaning is the *neighbouring cleaning rule* (NCL) by (Laurikkala, 2001). It is similar to OSS, however it uses Wilson's edited nearest neighbor rule (ENN) (Wilson and Martinez, 2000) to identify noisy/borderline examples instead of Tomek links. ENN is a simple method that look at the three nearest neighbours of each example and removes it if the class differs from its three nearest neighbours.

(Stefanowski and Wilk, 2006) proposed two filtering techniques called *remove* and *relabel* using rough set theory (Pawlak, 1998). Both methods identify inconsistent borderline examples from the majority class. The first method removes these examples, while the other relabels them as minority class examples as a data cleaning technique.

Cluster-Based Sampling

In an imbalanced data set, the classes can also suffer from within-class imbalance. As such, with random oversampling this imbalance is not considered. *Cluster-Based Oversampling* (CBO) by (Jo and Japkowicz, 2004) is a strategy for reducing not only between-class imbalance but also the within-class imbalance, by clustering the classes and perform random oversampling cluster by cluster. CBO performs class clustering by the K-means algorithm which works as follows: k examples are chosen at random to represent each of the k clusters. The input vector for each example represents the mean of each cluster. For every example in the class, the its distance to each of the k cluster means are calculated and the example is put in the closest cluster. The cluster who received the example has its mean updated by averaging the input vectors of its corresponding examples. After each class is clustered the oversampling starts. In the majority class every cluster is randomly oversampled expect from the largest cluster, to get the same number in every cluster as the largest one given by $maxclasssize$. In the minority, class each cluster is randomly oversampled until every minority cluster contains $maxclasssize/Nsmallclass$ examples, where $Nsmallclass$ is the number of clusters in the minority class. As cluster based oversampling is able to identify rare examples en re-sample them individually the method is able to reduce both within- and between-class imbalance which can increase classification performance (Jo and Japkowicz, 2004).

Integration of Sampling and Boosting

There has been many studies of integrating sampling strategies with ensemble learning, also called boosting algorithms. Boosting is a general method which attempts to *boost* the accuracy of any given learning algorithm. The first boosting algorithm that became widely popular was AdaBoost by (Freund and Schapire, 1995). Adaboost is en ensemble of t weak classifiers h_t and the output of the final classifier is given by:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{2.12}$$

where α_t is the weight given to h_t by AdaBoost which is based on the error-rate e_t of h_t:

$$\alpha_t = \frac{1}{2} \ln(\frac{1 - e_t}{e_t}) \tag{2.13}$$

BalanceCascade and EasyEnsemble discussed in section 2.2.1 which combines their sampling strategies with the boosting algorithm AdaBoost. Another example using with AdaBoost is SMOTEBoost by (Chawla et al., 2003) in which SMOTE is integrated with the boosting algorithm. Two other examples using AdaBoost is DataBoost-IM by (Guo and Viktor, 2004) which combines its sampling of difficult-to-learn examples with the boosting technique and RUSBoost (Seiffert et al., 2009) which is based on SMOTEBoost, but has made modifications to the SMOTE algorithm. Boosting has proven to increase performance in some cases, and good be a good addition to synthetic sampling strategies.

2.2.2 Algorithmic-level Methods

While data-level methods attempt to balance the class distribution of the data set prior to training the classifier, the is another set of methods that work at the algorithmic-level. They leave the class distribution unchanged and targets the class imbalance problem by altering the classifier directly. The algorithmic-level methods will not be elaborated in the same extent as the data-level methods, as they will not be used in experiments in this thesis. However, as algorithmic-level methods have proven for some imbalanced learning domains to be a viable alternative to sampling methods, the key concepts will be introduced.

Cost-Sensitive Learning

Most machine learning algorithms assume all misclassification errors cost equally and has the objective of minimizing the total costs (Jo and Japkowicz, 2004). This is at the core of the class imbalance problem, as the total cost of misclassified minority class examples can be fairly low even though the misclassification rate is high. *Cost-sensitive methods* tries to mitigate this issue by taking the misclassification costs for the different classes into consideration by changing the classifiers cost matrix. The goal is to assign different costs to examples from different classes in a way such that the classifier will not become biased towards the majority class. The resulting cost matrix then describes the misclassification cost of any specific example (Elkan, 2001).

During the training process of a classifier, it is penalized by assigning the wrong class to an example. The cost matrix C contains the cost values (penalties) for misclassifying an example, where the value $C(i, j)$ is based on the examples label i and the classifiers prediction j. We have that $i, j \in Y = [1, \ldots, q]$, where q is the total number of labels in the data set. A typical cost matrix is shown in eq. (2.14), the cost associated with predicting class j when the example belongs to class i. (Sheng and Ling, 2006).

$$\begin{bmatrix} C(1,1) & C(1,2) & \cdots & C(1,q) \\ C(2,1) & C(2,2) & \cdots & C(2,q) \\ \vdots & \vdots & \ddots & \vdots \\ C(q,1) & C(q,2) & \cdots & C(q,q) \end{bmatrix} \tag{2.14}$$

When all costs are equal we have a uniform cost matrix which is typically used by machine learning methods before cost-sensitive methods are applied.

$$\forall i, j : C(i, j) = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases} \tag{2.15}$$

Cost-sensitive methods also assigns zero cost for correctly classified examples, $C(i, j) = 0$ if and only if $i = j$. However if we have that $i \in q_{min}$ and $j \in q_{maj}$ where $\{q_{min} \cup q_{maj}\}$ is the total set of labels and $\{q_{min} \cap q_{maj}\} = \emptyset$, the costs are assigned such that $C(j, i) > C(i, j)$. This means that the cost of misclassifying a minority example as a majority example has greater cost than misclassifying a majority example as a minority example. Cost-sensitive methods objective is to minimize the overall cost, and this is

typically done by considering the Bayes conditional risk (Sheng and Ling, 2006) which is defined as

$$R(i|\mathbf{x}) = \sum_j p(j|\mathbf{x})C(i,j) \tag{2.16}$$

where $p(j|\mathbf{x})$ is the Bayesian a posteriori probability. This is the predicted probability of $\mathbf{x}$ belonging to class j and is given by

$$p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)}{p(\mathbf{x})}p(j) \tag{2.17}$$

Cost-sensitive learning can be implemented in a numerous of different ways. With respect to neural networks is could be implemented by letting misclassified examples with a high cost contribute more to the weight updates. Another method is to change the object of the training process from minimizing the standard loss function to minimizing the total misclassification cost (Kukar et al., 1998).

Thresholding

Threshold adjusting of simply *thresholding* takes on another approach. Instead of changing the cost matrix, the strategy is to change the decision threshold of the classifier. In most machine learning algorithms, when the classifier makes a prediction its output is simply the label that had the highest probability. If the prediction is based on probability estimates given by eq. (2.17), which is the case for most machine learning, then the threshold for making a prediction can be shifted. Thresholding simply finds the best probability as the new threshold T, and this new threshold is then applied to the process of predicting class labels for test examples (Sheng and Ling, 2006). For a test example $\mathbf{x}$ with predicted probability $p(i|\mathbf{x})$ will be predicted as positive if $p(i|\mathbf{x}) \geq T$ and will be predicted as negative if $p(i|\mathbf{x}) \leq T$.

Without thresholding the theoretical threshold for optimal decision making is (Elkan, 2001):

$$T = \frac{C(1,0)}{C(1,0) + C(0,1)}. \tag{2.18}$$

However, thresholding finds the best threshold in the following manner: For a given threshold, the the total misclassification cost M_C for a set can be calculated. This is a function of the threshold $M_C = f(T)$, and its curve can be obtained after calculating M_C for every possible threshold. In order to find the best value of T, it is in reality only necessary to calculate misclassification costs for each possible probability estimates on the training examples. With this curve, the chosen value of T is the one that minimizes M_C.

2.2.3 State-of-the-Art Methods

The data-level and algorithmic-level methods discussed in the previous sections are for the mostly fairly old and only well documented and researched for classical machine learning methods. There has not been nearly as much research on the impact of class imbalance mitigation methods on Deep Neural Networks or more specifically CNNs (Buda et al., 2018). In this section we will look at some new state-of-the-art methods that focus on neural networks.

GAN-Based Methods

Generative Adversarial Networks (GANs) are elaborated in section 2.3, however here is a short overview in order to explain GAN based class mitigation methods. Generative Adversarial Networks (GANs) is a recent innovation in machine learning proposed by (Goodfellow et al., 2014), and has gained much attention since its innovation in 2014. GANs is a framework for generative models that consists of two neural networks: a discriminative model (D) and a generative model (G). During the training process of the GAN the G model generates synthetic data examples while the D model evaluates given examples for authenticity; i.e. the probability whether a given instance is generated or from the training data. The objective of G is to maximize the probability of D misclassifying a generated example as a real one, while D tries to maximize its accuracy. During this process the G model learns to generate synthetic examples that are similar to examples from the training data.

After a GAN has been trained on a data set it is potentially able to generate synthetic data that resembles the data for which it was trained on, which makes it possible to use GANs for the purpose of oversampling. *GAN Based Oversampling* are synthetic sampling methods, that oversamples the training data by generating synthetic minority examples. GAN Based oversampling is similar to the previously described method SMOTE. As SMOTE generates synthetic points along the line-segment between two examples, the data generation is done in a simple manner that does not ensure meaningful new points to be generated. In the use of GANs, the generator learns the data distribution of the training data, and uses this to generate new examples. Results show that GAN-based oversampling can synthesize helpful samples for the minority classes to assist the training the CNNs (Wang et al., 2019). The technique has also shown to have great performance, outperforming SMOTE and other class imbalance methods (Lee and Park, 2019; Douzas and Bacao, 2018; Lei et al., 2019; Mullick et al., 2019).

CGAN-Plankton by (Wang et al., 2017) is a method for solving the class-imbalance problem for fine-grained classification tasks on images. In their method they used the WHOI dataset (Orenstein et al., 2015) which consists of thousands of *in-situ* plankton images. In their method they have the generative model (G) and discriminative model (D) from a GAN as well as a classification model (C). The G and D model consists of fully convolutional layers. The GAN is trained on the minority examples from the training data, and during training G becomes able to generate similar synthetic images while D learns effective discriminative parameters. Based on this idea the parameters of D are extracted and C is implemented with the same weights, which forces the classifier to focus on the minority classes in which the GAN was trained on. Their results showed that C is a powerful classifier, and when used for the classification task of the whole data set (both minority and majority classes), it outperformed popular CNN methods on overall performance. They also tested their generator for GAN-based oversampling, and showed that all CNNs used in their experiment achieved better performance when the training data was oversampled using their G model. In conclusion they showed that CGAN-Plankton creates a strong classifier as well as a sufficient generator for oversampling for imbalanced fine-grained classification tasks. (Radford et al., 2015), which created an improvement of the original GAN (further elaborated in section 2.3.2) also concluded that the trained discriminator from a GAN can show competitive performance for image classification

tasks.

Large Margin Local Embedding (LMLE)-kNN is a method proposed by (Huang et al., 2016), which aims to mitigate the class imbalance issue by learning deep feature embeddings. Their solution is a form of resampling and classification done by a cluster-wise kNN search with a local margin decision. The method perform the sampling by what they call quintuplet sampling with triple-header loss. This way of sampling ensures locality across clusters as well as discrimination between classes. Both within-class and within-cluster margins are enforces, which effectively reduces class imbalance in the local data neighbourhood. Their results showed that LMLE-kNN greatly improved classification performance for CNNs compared to classical methods for class imbalance mitigation.

2.3 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) is a framework for estimating generative models that were invented by (Goodfellow et al., 2014), and have become increasingly popular. The framework consists of two networks; a discriminative (D) net and a generative (G) net. These two nets are trained simultaneously and play a minimax game during the training process. The G net generates data while the D net estimates the probability that a given data point comes from the generative net or the training data. The two nets play a minimax game in the following manner; the D nets training procedure is to assign the correct label to the given data point while the G nets training procedure is to maximize the probability of the D model making a mistake. The relationship between D and G is during the training process is shown in fig 2.8. As seen from the figure, the discriminator is given original data from the training data and generated data from the generator, and then assigns label. The decision made by the discriminator is fed back to the generator. (Goodfellow et al.,

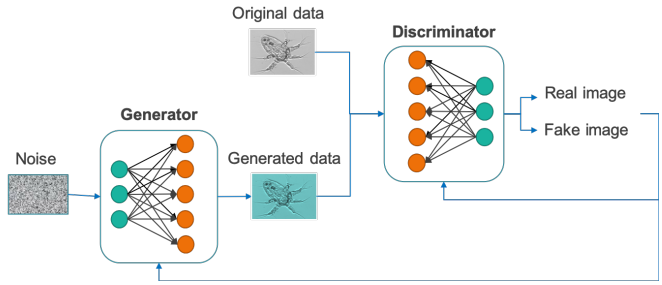


Figure 2.8: Relationship between generator and discriminator in GAN training process.

2014) proposes multilayered perceptrons for both the G and D net. In that case the whole system can be trained with backpropagation. Suppose the generator's data distribution p_g over the data $\mathbf{x}$ we define a prior noise distribution $p_{\mathbf{z}}(\mathbf{z})$ and represents the mapping to the data space $G(\mathbf{z};\theta_g)$. The generator and the discriminator are given by $G(\mathbf{z};\theta_g)$ and $D(\mathbf{x};\theta_d)$. D is trained to maximize the probability of correctly assigning correct labels $log(D(\mathbf{x}))$ while G is trained to minimize $log(1 - D(G(\mathbf{z}))$. This results in the minimax game with the following value function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[log(1 - D(G(\mathbf{z}))] \quad (2.19)$$

The minimizing of the value function $V(D,G)$ amount to minimizing the Jensen-Shannon distance between the distributions p_{data} and p_g (Fuglede and Topsoe, 2004). The minimax game results in the generators capability to generate synthetic data that is similar to the training data.

2.3.1 Conditional GAN

Short time after the introduction of GANs (Mirza and Osindero, 2014) introduced Conditional GANs. While GANs are trained unsupervised meaning the training data do not have class labels, while the conditional GANs are trained in a supervised manner where class labels are a part of the training process. The conditional version of GANs can be constructed by feeding the data we wish to condition on or the label data, y, to both the discriminator and the generator. They showed that the generator then is able to generate synthetic data conditioned by class label. The value function for the discriminator and generator will for a conditional GAN be:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x}|\mathbf{y}))] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[log(1 - D(G(\mathbf{z}|\mathbf{y}))] \quad (2.20)$$

The structure of a simple Conditional GAN is showed in fig. 2.9. With the introduction of

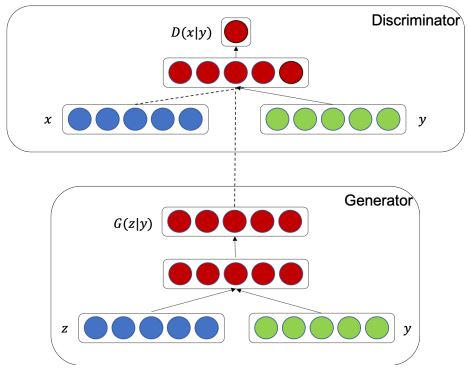


Figure 2.9: Structure of a Conditional GAN.

conditional GANs the possibility of generating synthetic labeled data is now possible.

2.3.2 Deep Convolutional GAN (DCGAN)

Convolutional GANs are GANs that uses Convolutional Neural Networks (CNNs) for the G and D model, in order to train the a GAN on a imagery dataset. The class of Deep Convolutional Generative Adversarial Networks (DCGAN) were proposed by (Radford et al., 2015), which is a class that contains a set of constrains on the architectural topology of Convolutional GANs, to make them stable during training in most settings. They introduced a set of guidelines for the CNNs G and D consists of;

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).

- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Their results showed that the networks learned good representations of images for supervised learning.

2.3.3 Wasserstein GAN (WGAN)

(Martin Arjovsky and Bottou, 2017) argues that the traditional GAN given by eg. 2.19 can lead to training difficulties, as the divergences they minimize are potentially not continuous with respect to G's parameters. They introduced the Wasserstein GAN (WGAN), and argue that WGANs cure the main training problem of GANs, by using the Wasserstein-1 distance, $W(p, q)$ instead of the Jensen-Shannon distance. The Wasserstein-1 distance can under mild assumptions be continuous everywhere and differentiable almost everywhere. The WGAN value function is given by:

$$\min_G \max_{D \in \mathcal{D}} V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[D(G(\mathbf{z}))] \tag{2.21}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions and the distributions p_{data} and $p_{\mathbf{z}}$ are the same as in eq. 2.19. This amounts under an optimal discriminator to minimizing $W(p_{data}, p_{\mathbf{z}})$. In order to ensure the Lipschitz constraint on D, (Martin Arjovsky and Bottou, 2017) propose clipping of the weights of D to lie within a compact space. They argue that their value function makes optimization of G easier and that the value function correlates with sample quality which traditional GANs value function do not.

2.3.4 WGAN Gradient Penalty (WGAN-GP)

An improved version of the Wasserstein GAN (Martin Arjovsky and Bottou, 2017) was proposed by (Gulrajani et al., 2017) called WGAN Gradient Penalty (WGAN-GP). Even though WGAN was an improvement towards stable training of GANs, it can still fail to converge or generate poor samples. They argue that WGAN-GP does not suffer from the same problems as the WGAN framework and show stable training of varied GAN architectures and high quality image generation. (Gulrajani et al., 2017) argue that the weight clipping done in the WGAN framework is the reason for the issues the framework suffer. They introduced an alternative way to ensure the Lipschitz constraint, which they call gradient penalty. They directly constrain the gradient norm of the discriminators, D, output with respect to its input. They add a penalty to the gradient norm for random samples $\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}$, which gives the new objective:

$$L = \underbrace{\mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[D(G(\mathbf{z}))] - \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[D(\mathbf{x})]}_{\text{Original Discriminator loss}} - \underbrace{\lambda \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}}[(||\Delta_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})||_2)^2]}_{\text{The Gradient Penalty}} \tag{2.22}$$

(Gulrajani et al., 2017) demonstrated that their model, WGAN-GP, outperformed WGAN and a more stable algorithm for training GANs.

2.3.5 Spectral Normalization GAN (SN-GAN)

The Spectral Normalization GAN (SN-GAN) is is a model proposed by (Miyato et al., 2018) that uses their method called spectral normalization. This method is made with the purpose of stabilizing the training of discriminator networks. Their work showed that in some cases spectral normalization can improve the quality of the generated images better than gradient penalty. They observed that a high learning rate can make the performance of WGAN-GP unstable, while with the spectral normalization remains more stable and does not easily destabilize. They also observed that SN-GAN has a lower computational cost than WGAN-GP. To explain the spectral normalization we first need to look at the architecture of the discriminator. We consider a discriminator D made made of a neural network with the input $\mathbf{x}$:

$$f(\mathbf{x}, \theta) = W^{L+1} a_L(W^L(a_{L-1}(W^{L-1}(...a_1(W^1\mathbf{x}))))), \tag{2.23}$$

where the learning parameters are given by $\theta := W^1, ..., W^L, W^{L+1}$, $W^l \in \mathbb{R}^{d_l x d_{l-1}}$, $W^{L+1} \in \mathbb{R}^{1 x d_L}$ and a_l is an element-wise non-linear activation function. Then the output of the discriminator will be given by

$$D(\mathbf{x}, \theta) = \mathcal{A}(\{(\mathbf{x}, \theta)), \tag{2.24}$$

where $\mathcal{A}$ is an activation function, that corresponds to the divergence of distance measure chosen. The spectral normalization proposed by (Miyato et al., 2018) controls the Lipshitz constant of the D function f by constraining the spectral norm of each layer: $g : \mathbf{h}_{\text{in}} \rightarrow \mathbf{h}_{\text{out}}$. The Lipshitz norm denoted $||g||_{\text{Lip}}$ is equal to $\sup_{\mathbf{h}} \sigma(\Delta g(\mathbf{h}))$, where $\sigma(A)$ is the spectral norm of matrix A:

$$\sigma(A) := \max_{\mathbf{h}:\mathbf{h}\neq 0} \frac{||A\mathbf{h}||_2}{||\mathbf{h}||_2} = \max_{||\mathbf{h}||_2 \leq 1:} ||A\mathbf{h}||_2. \tag{2.25}$$

The SN-GANs spectral normalization normalizes the weight matrix Ws norm so it satisfies the Lipshitz constraint $\sigma(W) = 1$:

$$\bar{W}_{SN}(W) := \frac{W}{\sigma(W)} \tag{2.26}$$

If every W^l is normalized using eq. 2.26 then $||f||_{\text{Lip}}$ is bounded from above by 1 and the Lipshitz constraint is enforced. (Miyato et al., 2018) achieves generated examples that are more diverse using GAN-SN than other weight normalization methods.

2.3.6 Evaluation Metrics

When using generative models the choice of evaluation metric for the model's performance is an important aspect. In the case of GANs one could look at how well the generator fools the discriminator, however this is not a good indicator of the quality or diversity of the generated images. As GANs do not have an objective function, there is no objective objective way of comparing different models or the generated output. As a result finding a proper way to evaluate the performance of GANs proved to be a difficult task (Theis et al.,

2015). The first evaluation metric that was widely used and correlated well with human judgement was the *Inception Score* (IS) proposed by (Salimans et al., 2016). As the IS did not take into consideration how well the generated images compare to the real ones, the *Fréchet Inception Distance* (FID) score was introduced by (Heusel et al., 2017), which proved to capture this similarity better. The FID score then became the standard evaluation metric for the performance of GANs.

Inception Score (IS)

The Inception Score (IS) by (Salimans et al., 2016) is a score computed over a set of generated images by a GAN. During the computation the diversity of the images are considered as well as the quality of each image. If the images are diverse and each image represent something meaningful, this will result in a high score. This means the higher the score, the better the generator is at generating distinct high-quality images. The lowest score possible is zero, while the highest theoretical score is infinite.

In order to compute the IS the conditional label distribution $p(y|\mathbf{x})$ for each generated image is extracted using the Inception Model by (Szegedy et al., 2016). For images meaningful objects, this distribution should have low entropy. As the generator is expected to generate diverse images the marginal distribution $\int p(y|\mathbf{x} = G(z))dz$ should have high entropy. The combination of these two metrics are the basis for the inception score; the more these two entropies differ the higher the inception score. This is achieved by using the Kullback–Leibler divergence (Van Erven and Harremos, 2014), also called relative entropy, given by

$$D_{\mathrm{KL}}(P||Q) = \int pln\frac{p}{q}d\mu, \tag{2.27}$$

with the conventions that $0ln(0/q) = 0$ and $pln(p/0) = \inf$. The inception score is then given by

$$\mathrm{IS}(G) = \exp(\mathbb{E}_{\mathbf{x}} D_{\mathrm{KL}}(p(y|\mathbf{x})||p(y))). \tag{2.28}$$

The inception score was widely used, and for example the makers of SN-GAN (Miyato et al., 2018) used it in order to compare their method to others. However there are limitations to using IS as the evaluation metric for GANs. For one, if the generator learns to generate something that is meaningful but is not present as a class in the training data, this will result is low IS even though the image is of high quality. Secondly, the IS does not capture inter-class diversity, so the generator can generate one image per class repeatedly and still achieve a high IS. A third drawback is that if the generator simply learns to replicate the training data, the IS will also be high.

Fréchet Inception Distance (FID) Score

Due to the drawbacks of the inception score, a new method for evaluating images generated by GANs called Fréchet Inception Distance (FID) Score was proposed by (Heusel et al., 2017). They argue that the FID score is more consistent and captures the similarity of real and generated images better than the IS. The FID score is based on goal of training a generative model that produces data that matches the training data. With this goal in mind each distance between the the probability of generating model data $p(.)$ and probability of

observing real data $p_\omega(.)$ can be used as an evaluation metric for GANs. The FID score is an improvement of the IS, by adding this element of comparing the generated data to the training data.

The FID is computed by considering the Fréchet distance Fréchet (1957) between two multivariate Gaussian distributions, representing the real and the generated data. We have one Gaussian obtained from $p(.)$ with mean and covariance $(\mathbf{m}, \mathbf{C})$, and the other obtained from $p_\omega(.)$ with mean and covariance $(\mathbf{m}_\omega, \mathbf{C}_\omega)$. With the Fréchet distance as $d(.,.)$ the FID is given by:

$$d^2((\mathbf{m}, \mathbf{C}), (\mathbf{m}_\omega, \mathbf{C}_\omega)) = ||\mathbf{m} - \mathbf{m}_\omega||_2^2 + \text{Tr}(\mathbf{C} + \mathbf{C}_\omega - 2(\mathbf{C}\mathbf{C}_\omega)^{1/2}) \tag{2.29}$$

Low FID values means the two distributions $p(.)$ and $p_\omega(.)$ are similar, and therefore means high diversity and image quality among the generated images (Heusel et al., 2017). After the invention of the FID, it replaced the IS as the standard as the GAN evaluation metric. (Lucic et al., 2018) concluded that the FID was a reasonable evaluation metric for GANs due to its robustness, and used it for their comprehensive comparison of state-of-the-art GANS.

Fig. 2.10 shows some images generated by a SN-GAN using the open-sourced implementation by (Lucic et al., 2018) available at `https://github.com/google/compare_gan`. The SN-GAN is trained on the CIFAR-10 Krizhevsky et al. (2009) dataset. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The images corresponding FID score is shown in the figure, and we can see that there is a clear correlation between FID and image quality.

Figure 2.10: SN-GAN trained on Cifar-10 dataset generated images with corresponding FID score.

Chapter 3

Experiment

This master thesis is a continuation of the specialization project with the same name. The project is part of a larger project called AILARON in which several people are working at. The objective of this thesis is described at the AILARON web-page `https://www.ntnu.edu/web/ailaron/msc-projects` as: ”Existing planktonic datasets suffer from class imbalance. Trained deep learning architectures tend to favor classification of objects to higher weight classes. This can mislead the segmentation and classification procedure. In this project, Oda is studying recent approaches to solve the class-imbalance and evaluate their performance over datasets of planktonic organisms.” The AILARON project has collected a set of images referenced as the SilCam data set, as well as developed a network architecture called COAPNet. Both the dataset and the classifier will be used in the experiments in this thesis.

3.1 Dataset and Classification Model

SilCam is a data set which consists of RGB images of microscopic planktonic organisms of different sizes. It consists of 7941 images and 8 classes. Its class distribution and the number of images per class is shown in fig. 3.1. As seen from the figure the data set has a form of linear imbalance. From the class labels we can see that the data set consists of different species of planktonic organisms as well as some other classes such as *oil* and *bubble*. In this experiment we are most interested in the classes that represent species, which is: *diatom_chain*, *copepod*, *faecal_pellets* and *fish_egg*. We can also see that the majority of the data set consists of images that belong to the class *bubble* and *other*. For the purpose of this thesis the class *bubble* will be referenced as the majority class and images associated with this class belongs to the majority set S_{maj}. It then naturally follows that the remaining seven classes are referenced as minority classes and their images belong to the minority set S_{min}. More information and implementation of the data set is elaborated in section 4.1.

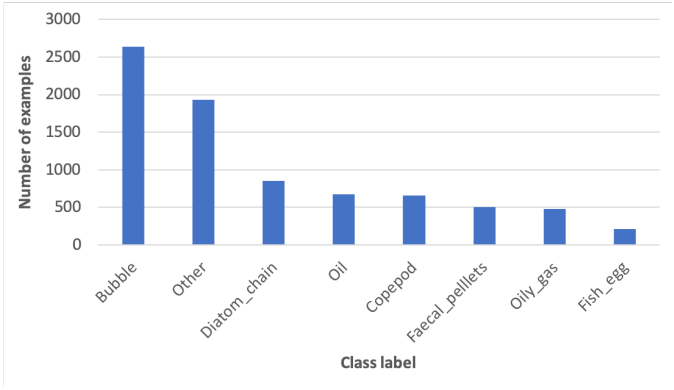


Figure 3.1: Class distribution of the imbalanced dataset SilCam.

For the purpose of comparing different class mitigation methods, the data set is split into the subsets train, test and validation which contains 70%, 15% and 15% of the total set. The number of images in each set is shown in table 3.1. The split is done in such a way that the class distribution in each set is the same as in the original data set.

Table 3.1: Number of images for training, testing and validation)

TRAIN	TEST	VALIDATION	TOTAL
5556	1199	1186	7941

All experiments are tested using the COAPNet, which is a network architecture developed by the AILARON team. It has been specialized and optimized specifically for the SilCam data set. Information about the implementation of the COAPNet is elaborated in section 4.2.

3.2 Methods of Addressing Imbalance Compared in This Study

In this thesis we compare GAN-based oversampling with random minority oversampling and random majority undersampling. Full oversampling is tested as well as minority oversampling with 50 images and 150 images which gives the tests:

1. Full Random Majority Undersampling
2. Full Random minority oversampling
3. 150 Random minority oversampling
4. 50 Random minority oversampling
5. Full GAN-based oversampling
6. 150 GAN-based oversampling
7. 50 GAN-based oversampling

For this thesis the classifier in use it the COAPNet. As this is a network under development it was preferable to investigate the impact of data-level methods compared to algorithmic-level methods. Ensemble methods require training of multiple classifiers. For deep neural networks such as the COAPNet considerable time is needed for training, which makes the training of multiple classifiers both impractical and in some cases even infeasible. LMLE-Knn is a method with promising results, however the method can not be integrated with the COAPNet. as elaborated in section 2.2.3 GAN-based oversampling has shown promising results, often outperforming SMOTE. In the project thesis this master thesis is a continuation of, GAN-based oversampling was tested on the WHOI data set Orenstein et al. (2015) and showed promising results. However the method was not compared to other methods. As this data set suffers from within-class imbalance as well as between-class imbalance, cluster-based sampling should be investigated. However, due to the Covid-19 situation, experiments had to be pushed back and cluster-based sampling experiments did not fit in the timeline.

Random Resampling Method

For the sampling methods, only the training data is oversampled, while the test and validation data is left untouched. In the training data there are 8 classes. For the oversampling methods the majority set S_{maj} consists of the largest class while the minority set S_{min} consists of the 7 smallest of them, where $S_{min,i}$ represents the set of each class, $i = 1, ..., 7$ is the class label. E_i is the set sampled from $S_{min,i}$ and we have that $n_i = |E_i|$. For the method Full oversampling we have that

$$n_i = |S_{maj}| - |S_{min,i}|, \quad i = 1, ..., 7 \tag{3.1}$$

to get a fully balanced dataset. For 150 and 50 oversampling we have that $n_i = 150$ and $n_i = 50$ respectively. The new oversampled dataset is then the sumset

$$S = S_{maj} + \sum_{i=1}^{7} S_{min,i} + E_i, \quad |E_i| = n_i \tag{3.2}$$

For the undersampling method the minority set S_{min} consisted of images from the smallest of the 8 classes. The remaining 7 classes made up the majority set S_{maj}. For the undersampling approach the number of images to sample for each class $S_{maj,i}$ was $n_i = |E_i| = |S_{min}|$. The new undersampled data set is then the sumset $S = S_{min} + \sum_{i=1}^{7} E_i$.

For both sampling methods the test set is not changed to match artificially balanced training sets. The reason for this is that the score achieved by the classifier when tested on the same test set is more comparable.

GAN-Based Sampling

For the GAN-based oversampling, three different GAN architectures were considered; DCGAN, WGAN-GP and SN-GAN. Implementation of the three are elaborated in section 4.3. The architectures were trained on the 7 smallest classes in training data from the SilCam data set. Different image resolutions for the data set were tested; 64x64x3, 64x64x1, 32x32x3, 32x32x1. For all of the architectures when trained on the SilCam dataset with image resolution of 64x64x3 out of memory error occurred, which means no GAN were able to finish training with this resolution. During the training the GAN saves checkpoints and the corresponding score achieved by that checkpoint. For each checkpoint the GAN generates three synthetic data sets and calculates the FID score for each of them. The GAN that achieved the overall highest score were chosen as the GAN to use in the GAN-Based oversampling methods. For this model, the checkpoint that scored the highest were used for image generation. The dataset is oversampled the same way as in random minority oversampling given by eq. 3.2, the only difference being E_i is the generated data and it is not a subset of S_{min}.

3.3 Evaluation Metrics and Testing

When it comes to evaluating classifier performance in the context of multiclass classification with CNNs, the metric most widely used is overall accuracy (Buda et al., 2018). However, as discussed in section 2.1.2, accuracy can be misleading, particularly in the context of imbalanced data sets. (Wang et al., 2017) did experiments of training GANs on images of planktonic organisms. They argue that the F1 score (given by eq. 2.4) is the best metric for evaluation performance of CNNs as it gives the balance between recall and precision. In our thesis we will look at the F1 score as well as precision and accuracy. These evaluation metrics are elaborated in section 2.1.2.

For the evaluation of the GANs, the FID score will be used. The FID score is discussed in section 2.3.6. During the training the GANs, the GAN that achieves the highest FID score will be considered the one that has the ability to generate images that will increase classification performance the most.

Chapter 4

Implementation

The majority of the work done in this thesis has been implementation related. As such, the project has been limited by the available hardware. For testing two GPUs of ASUS RTX2080Ti Turbo with available RAM of size 64 GB have been possible to access. However, multiple people related to the AILARON project have access to the GPUs, which makes them occupied most of the time. With more computational power available, it would have been possible to run more tests than what was done. In this chapter we will go through the implementation related to the data set, the classifier used and the generative adversarial networks tested.

4.1 Dataset

The dataset used in this theses is called the SilCam dataset, and is collected by the AILARON Project which is lead by Anette Stahl (AILARON). The images are captured in-situ by an autonomous underwater vehicles (AUV). The dataset consists of almost 8000 images of microscopic marine plankton and has 8 classes. The classes and the number of images per class are shown in table 4.1, and the class distribution is shown in fig. 4.1. As seen from

Table 4.1: Number of images associated with each class in the SilCam data set.

LABEL	NUMBER OF IMAGES
BUBBLE	2636
OTHER	1931
DIATOM_CHAIN	850
OIL	671
COPEPOD	657
FAECAL_PELLETS	504
OILY_GAS	479
FISH_EGG	213
TOTAL	**7948**

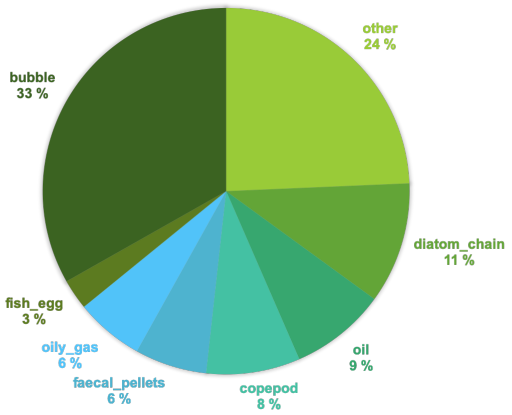


Figure 4.1: Class distribution for all classes from the SilCam dataset

both the table and the figure the dataset suffer from class imbalance, that is the number of images in each class differ. The class *bubble* consists of 2636 images which is equivalent to 33% of the dataset while the class *fish_egg* only makes up 3% of the dataset with its 213 images. The images are very diverse when it comes to resolution and quality across classes

Table 4.2: Average image resolution for each class

LABEL	AVERAGE RESOLUTION IN PIXELS	STANDARD DEVIATION
COPEPOD	(101, 83)	± (39, 37)
FAECAL_PELLETS	(32, 37)	± (21, 19)
OTHER	(38, 38)	± (42, 40)
FISH_EGG	(84, 91)	± (37, 33)
BUBBLE	(26, 26)	± (10, 10)
DIATOM_CHAIN	(49, 49)	± (43, 44)
OIL	(36, 35)	± (23, 23)
OILY_GAS	(28, 28)	± (15, 16)

as well as inter-class. Table 4.2 shows the average image resolution for each class and as the standard deviation. The average image size across the whole dataset is $(50, 48)$ pixels, which means these are images that does not contain a high level of information. Some example images from each class are shown in fig. 4.2, where the difference in resolution and image quality is shown clearly.

4.1.1 Scaling

In order to use the dataset for our purpose, all images are required to have the same resolution. Three different scaling techniques were considered; bilinear interpolation, bicubic interpolation and the reflect method.

Bilinear Interpolation is a method that considers the weighted average of four neighbouring pixel values when calculating the new value for the interpolated point (x, y) (Tezduyar et al., 1992). The computation works as follows; we want to know the value of the unknown function f at the point of interest (x, y). The value of f is known at four points $Q_{11} = (x_1, y_1)$, $Q_{12} = (x_1, y_2)$, $Q_{21} = (x_2, y_1)$, and $Q_{22} = (x_2, y_2)$. $f(x, y)$ is then given by:

$$f(x,y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}.$$

Bicubic Interpolation uses 16 neighbouring pixels for calculation of the interpolated point. The method uses a bicubic polynomial surface (Tezduyar et al., 1992). The general function is given by:

$$I_{xy} = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j$$

Reflect is a method that uses the reflection of the vector mirrored as padding. For example, if a vector $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ were to be padded by four values to the right, the results using the reflect method would be $\begin{bmatrix} 1 & 2 & 3 & 2 & 1 & 2 \end{bmatrix}$

Figure 4.2: Shows example images for each class. 1: Bubble, 2: Copepod, 3: Faecal Pellets, 4: Diatom Chain, 5: Fish egg, 6: Oil, 7: Oily gas, 8: Other

Fig. 4.3 shows the result after using the three different rescaling techniques. The original image to the left is originally 166x120 pixels. The image was rescaled to 194x194 pixels. All three techniques were applied to the whole dataset, and classification perfor-

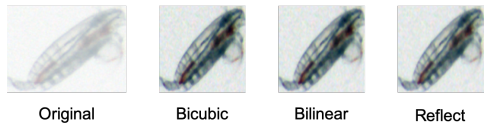


Figure 4.3: Image rescaled using different rescaling techniques

mance considered. The results are shown in table 5.1. As seen from the table the choice

of scaling technique has an impact on precision. The method that gave the best results were bicubic interpolation, and this method was therefore implemented as the rescaling method. Some example images are shown in fig. 4.4 after using bicubic interpolation for 64x64x3 pixels scaling. Each row contains images from the following classes: 1: bubble, 2: copepod, 3: diatom_chain, 4: faecal_pellets, 5: fish_egg, 6: oil, 7: oily_gas and 8: other

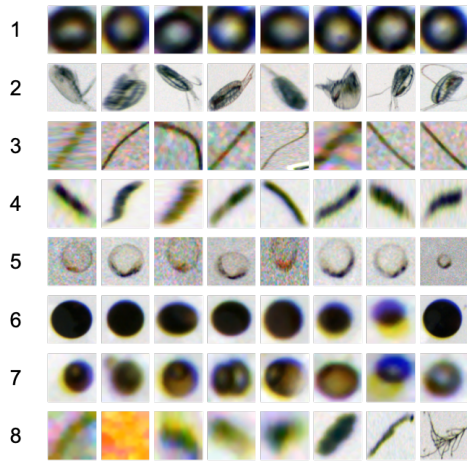


Figure 4.4: Example images of the SilCam after scaling is applied, where each row represents a class.

4.2 COAPNet

For the image classification task the small deep learning architecture COAPNet proposed by (Saad et al., 2019) were used. This is a deep neural network architecture that is specialised and fine-tuned for the SilCam dataset. Their results have showed that the COAPNet outperforms other state of the art neural networks such as VGGNet, ResNet, AlexNet, ZooplanktoNet and GoogleNet in the task of classifying and identifying in-situ plankton imagery. The architecture of the COAPNet is as follows; there are five convolutional layers with ReLU as activation, that are intertwined with five max-pooling layers for dimentionality reduction. Following these layers are three fully connected layers. This can be summarized in the table 4.3. COAPNet uses a learning rate of 0.001, optimizer Adaptive Moment Estimation and categorical crossentropy as loss function. The implementation of

Table 4.3: Network architecture for the COAPNet.

Layer 1	Convolution layer with 64 filters, each 3x3x3
	Max pooling layer
Layer 2	Convolution layer with 128 filters, each 3x3x3
	Max pooling layer
Layer 3	Convolution layer with 256 filters, each 3x3x3
	Max pooling layer
Layer 4	Convolution layer with 512 filters, each 3x3x3
	Max pooling layer
Layer 5	Fully-connected 512 node neural network
Layer 6	Fully-connected 256 node neural network
Layer 7	Fully-connected 256 node neural network
Layer 8	Fully-connected softmax with outputs to make the final prediction

the network is available at `https://github.com/emlynjdavies/PySilCam/`. In order to use the implementation for the purpose of this thesis modifications were necessary. It does not support functionality for specifying test and training data. As it is crucial for this experiment for the test data to be kept consistent across different methods, this functionality had to be implemented. Functionality for support of different image resolution were also implemented as well as a more absolute and informative result report. The network architecture however were not changed from their implementation.

4.3 GANs

Since the invention of Generative adversarial networks (GANs) there has been very high research activity on the field which have led to a numerous of different algorithms and network architectures. A comprehensive empirical study was done by (Lucic et al., 2018) where state-of-the-art GANs and evaluation measures were compared. Their results showed that most models can reach similar FID scores as long as high enough computational budget is provided. Their model implementations and setup is open-sourced at `goo.gl/G8kf5J`.

In this thesis this open-source implementation of GANs were used. However modifications had to be made for it to serve the purpose of this thesis. First of all the the setup for the implementation is outdated. Some of the required installations are old versions that are no longer available, and newer versions are not compatible with the rest of the setup. This led to a lot of testing of different installation versions in order to find a combination that was compatible. The implementation also had some bugs that had to be fixed in order to run the standard configurations. Secondly the implementation is made for comparison of GAN performance, and not for image generation. While images are generated during the training process of the GAN, these images are of random labels and discarded during training. As a result the functionality for image retrieval from trained models had to be implemented and integrated. As the experiment requires labeled generated images further modifications were made, to ensure that retrieved images were consistent with the label in demand.

The implementation by Lucic et al. (2018) is made using TensorFlow (Abadi et al., 2015), and all datasets that are used are included in the public list of TensorFlow Datasets (TFDS). In order to use the implementation with datasets that are not in the TFDS list, the dataset has to be modified and added to this library. This only has to be done locally, the dataset does not have to appear on the public list of datasets. As a result the SilCam dataset was made into a TFDS. Compatibility for the SilCam dataset was also integrated in the implementation.

For this experiment three different GANs were evaluated; the DCGAN (Radford et al., 2015), the SN-GAN (Miyato et al., 2018) and the WGAN-GP (Gulrajani et al., 2017) which are further explained in section 2.3.2, 2.3.5 and 2.3.4. The architectures were implemented with the parameters shown in table 4.5. None of these architectures were implemented as conditional GANs by Lucic et al. (2018), so modifications were made for the training process to be conditional in order to retrieve labeled images from the trained model. No modifications were made to the network architecture. The implementation were made compatible to run on GPUs, and all experiments related to the GANs were run on the available GPU. However, even with the GPUs available, the training time for the GANs, as seen in table 4.4, is fairly long. Evaluation is not included in the training time, so overall time needed to train and evaluate the GANs is around doubled from what is seen in the table. Image generation and retrieval from a trained model also took several hours. This makes tuning of hyperparameters an extremely time consuming task, which proved to not be possible in this thesis.

All experiments and GAN configurations are run with a batch size of 64. (Lucic et al., 2018) stated that one of their biggest findings were the increase in performance as batch size increase. With this finding it would be natural to test higher batch sizes in out experi-

Table 4.4: Results from training different GAN architectures with different image resolution.

GAN	Data set resolution	Training time
DCGAN	64x64x1	90 min
DCGAN	32x32x3	58 min
WGAN-GP	32x32x3	387 min
SN-GAN	32x32x3	243 min
SN-GAN	32x32x1	180 min

ments, however the batch size is limited by the hardware, as an increase batch size lead to out of memory error.

Table 4.5: Network architecture for the COAPNet.

	DCGAN	WGAN-GP	SN-GAN
Batch size	64	64	64
lambda	1	10	1
Training steps	100000	40000	40000
Discriminator iterations	1	5	1
Learning rate	0.0002	0.0001	0.0002

Chapter 5

Analysis

In this chapter we will present the results from the experiments. First we will present results from different image resolutions and preprocessing techniques. Then we will continue to discuss the imbalance found in the data set. Afterwards the results from the different GANs will be presented and for the last section the different class mitigation methods will be compared.

Table 5.1: COAPNet classification results from different resizing methods, tested on the SilCam dataset with image resolution 64x64x3.

METHOD	ACCURACY	PRECISION	F1-SCORE
BICUBIC	92.4937	92.4807	92.4735
BILINEAR	91.4929	91.5344	91.4947
REFLECT	92.4937	92.5719	92.4688

Table 5.2: COAPNet classification results with different image resolutions.

IMAGE SIZE	ACCURACY	PRECISION	MICRO F1	TRAINING TIME
64X64X3	92.4937	92.4807	92.4735	370 MIN
64X64X1	88.3236	88.2389	88.2190	200 MIN
32X32X3	90.8257	90.8121	90.7901	60 MIN
32X32X1	89.1576	89.1724	89.0341	55 MIN

5.1 Effect of Image Resolution and Preprocessing on Classification Performance

Results from the three different rescaling methods tested on the SilCam dataset are shown in fig 5.1. For all three methods the images were resized to 64x64x3 pixels. From the results we can see that bilinear interpolation was the methods that had the poorest results, and bicubic interpolation and the reflect method achieved almost identical scores. As we are interested in the F1 score, the bicubic method was chosen for the rest of the experiments.

Results from different image resolutions are shown in figure 5.2. We can see from the results that color images achieved the best classifier performance, where 64x64x3 had a higher score than 32x32x3. However, it is interesting to see that it is the other way around for the grayscale images: resolution size 32x32 gave better performance than resolution size 64x64.

Table 5.3: Results without class-imbalance mitigation for image resolution 64x64x3.

	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
CLASS 0	**1.0000**	**1.0000**	**1.0000**	33
CLASS 1	**0.9596**	**0.9500**	**0.9548**	100
CLASS 2	**0.8485**	**0.8750**	**0.8615**	128
CLASS 3	0.8759	0.8729	0.8744	291
CLASS 4	**0.8400**	**0.8290**	**0.8344**	76
CLASS 5	0.9801	0.9924	0.9862	396
CLASS 6	0.9105	0.8356	0.8714	73
CLASS 7	0.9608	0.9608	0.9608	102
ACCURACY			0.9250	1199
MACRO AVG	0.9219	0.9145	0.9179	1199
WEIGHTED AVG	0.9248	0.9249	0.9247	1199

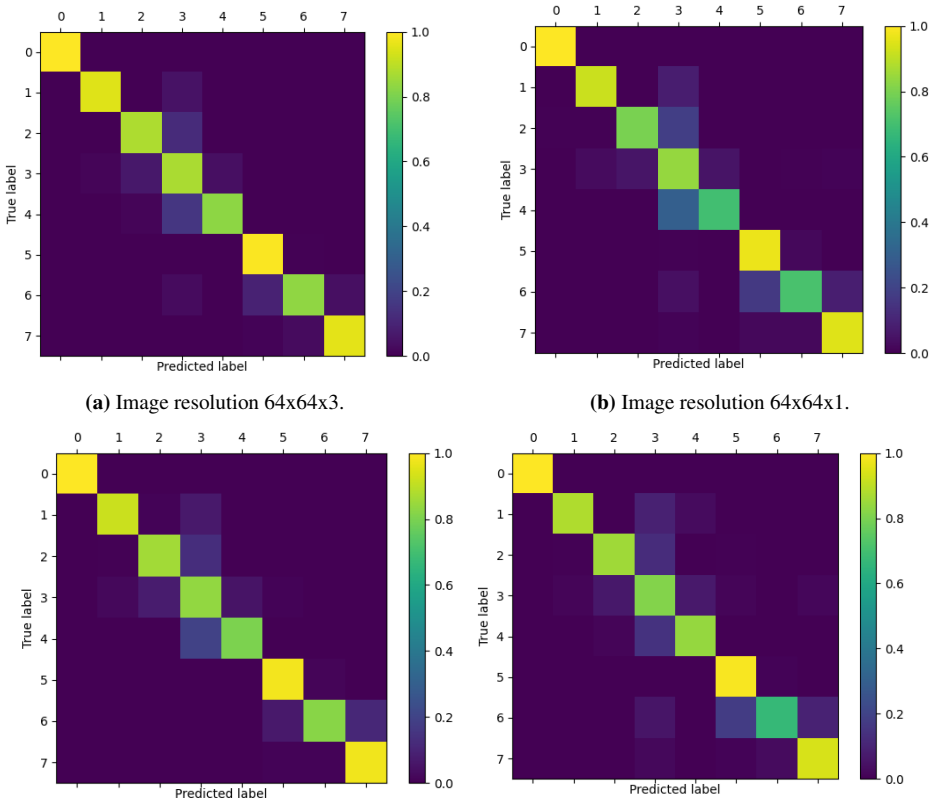
Table 5.4: COAPNet classification results with different image resolutions.

IMAGE SIZE	MACRO F1	MICRO F1
64X64X3	91.79	92.47
64X64X1	86.70	88.22
32X32X3	90.01	90.79
32X32X1	87.58	89.03

5.2 Class Imbalance Impact Without Mitigation Methods

Fig. 5.1 shows the confusion matrices for each of the four different image resolution sizes where the horizontal axis represents the predicted class label and the vertical axis represents the true label. As mentioned, the classes that are the most important are fish_egg, copepod, diatom_chain and faecal_pellets which corresponds to class 0, 1, 2 and 4 respectively. Table 5.3 shows more detailed output results from the classifier tested on the 64x64x3 image resolution. Equivalent tables for the other resolutions are shown in tables 6.1, 6.2, 6.3 in the appendix. We can see from the tables that the smallest class, class 0, is the one that achieves the highest results. However, we see that the classifier has trouble classifying one of the other small classes, class 4. From the confusion matrices we can see that class 4 examples are often misclassified as class 3. This is also true for class 2.

Table 5.4 shows the micro- and macro F1 scores for each of the four tests. As mentioned in section 2.1.2 micro-average is biased towards the most populated classes while the macro-average is biased towards the least populated ones. From the scores we see that there is no highly significant difference in the micro- and macro F1 scores. However, it is true for each test that the micro-F1 score is higher than the macro-F1 score, which implies the smaller classes are poorly classified compared to the bigger ones. This corresponds with what we see from the table 5.3.



(a) Image resolution 64x64x3.

(b) Image resolution 64x64x1.

(c) Image resolution 32x32x3.

(d) Image resolution 32x32x1.

Figure 5.1: Confusion matrices after training COAPNet with the original imbalanced dataset, for different image resolutions.

From the fig. 5.1 and table 5.4 the results in we see that in general the classification performance is poorer for the grayscale data set. We see more examples misclassified as label 3, and more example from class 6 being misclassified. What we see in the grayscale data sets is an increase in class overlap. From the confusion matrices we also see that the most common misclassification error is wrongly predicting an example of belonging to class 3. This suggests a high level of within-class imbalance in this class. This corresponds to the fact that class 3 is the class named *other*, which is a shared class for all images that do not specifically belong to one of the othre 7 classes.

Table 5.5: Results from training different GAN architectures with different image resolution.

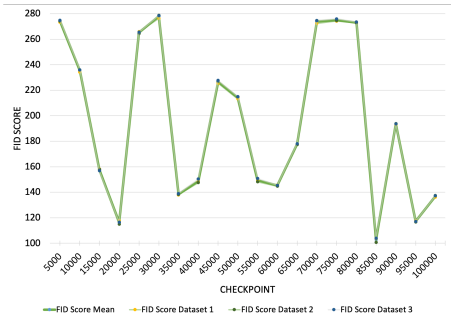
GAN	Data set resolution	Lowest Mean FID	Training time
DCGAN	64x64x1	102.82	90 min
DCGAN	32x32x3	92.06	58 min
WGAN-GP	32x32x3	58.76	387 min
SN-GAN	32x32x3	30.85	243 min
SN-GAN	32x32x1	25.99	180 min

5.3 Comparison of GAN performance for CGAN-Based Oversampling

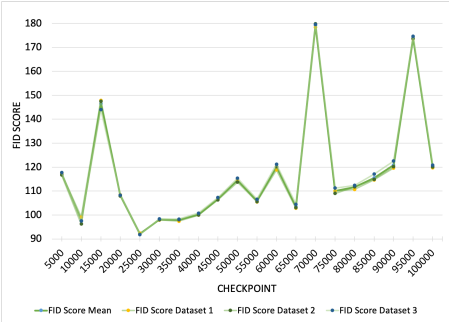
Attempts of training a GAN with resolution 64x64x3, however, this proved not possible with the hardware and computational power available. This was unfortunate as the 64x64x3 dataset were the one that achieved the highest classification score.

5.3.1 DCGAN

The DCGAN was first trained the 64x64x1 data set. After the training, the checkpoint which achieved the highest FID score was checkpoint 850000 with the FID score 102,82. The FID scores achieved for each checkpoint are shown in fig. 5.2a and images generated by that checkpoint are shown in fig. 5.3a. A FID score above 100 is extremely high, and as seen from the generated images, they are not of high quality. The DCGAN was also trained on the 32x32x3 data set. FID scores are shown in fig. 5.2b and generated images shown in fig. 5.3b. The lowest FID score accomplished was of 92,06, which is slightly better than for the 64x64x1 data set. However, this is still a high score, and also here we can see that the generated images are not of high quality. As the DCGAN achieved such high FID scores, it was not tested on the 32x32x1 dataset.

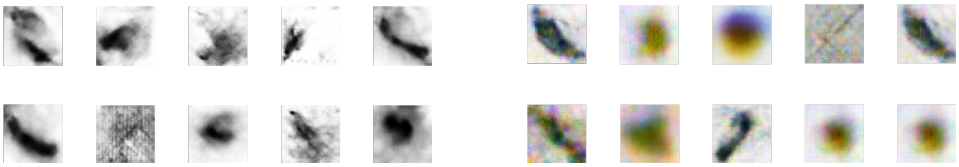


(a) Image resolution 64x64x1.



(b) Image resolution 32x32x3.

Figure 5.2: FID score achieved for each checkpoint during training of DCGAN with different image resolutions.



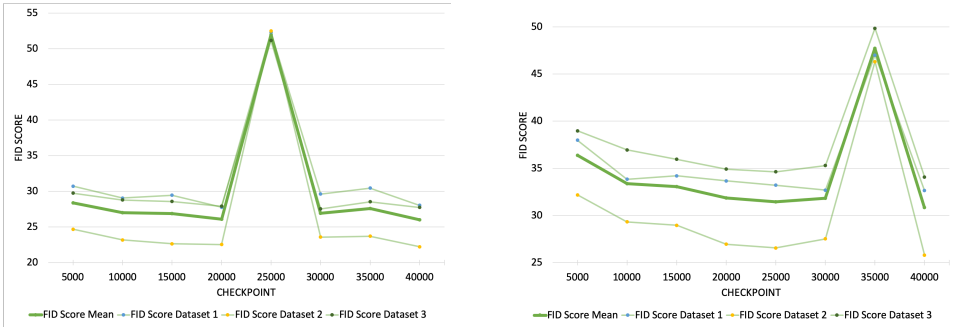
(a) Image resolution 64x64x1, FID = 102,82.

(b) Image resolution 32x32x3, FID = 92,06.

Figure 5.3: Images generated by the DCGAN with the corresponding FID score achieved.

5.3.2 SN-GAN

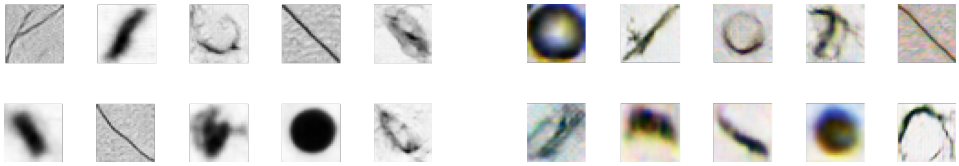
The next GAN architecture tested was the SN-GAN. The implementation of SN-GANs by (Lucic et al., 2018) is specialized for images with 32 pixel resolution, and do not support images of a higher resolution. As a result, the SN-GAN was trained on the data sets with resolution 32x32x3 and 32x32x1. The FID scores achieved for each checkpoint for both tests are shown in fig5.4. The lowest FID achieved for 32x32x3 was 25,99 and for 32x32x1 it was 25,99. These are acceptable low FID score, and the SN-GAN clearly outperformed the DCGAN. Example images from both tests are shown in fig 5.5. We can clearly see that the image quality is improved compared to the images produced by the DCGAN.



(a) Image resolution 64x64x1.

(b) Image resolution 32x32x3.

Figure 5.4: FID score achieved for each checkpoint during training of SN-GAN with different image resolutions.



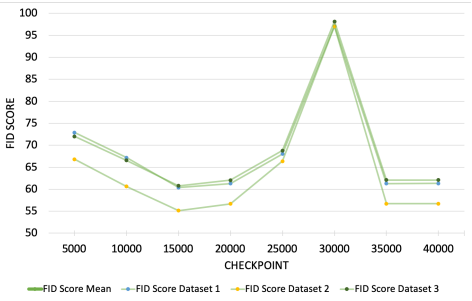
(a) Image resolution 32x32x1, FID = 25,99.

(b) Image resolution 32x32x3, FID = 30,85.

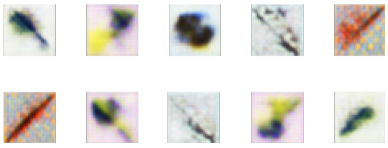
Figure 5.5: Images generated by the SN-GAN with the corresponding FID score achieved.

5.3.3 WGAN-GP

The last GAN architecture tested was the WGAN-GP. This GAN architecture was only tested on the image resolution 32x32x3. The FID score achieved by each checkpoint is shown in fig. 5.6a and images generated by the checkpoint which accomplished the highest FID score is shown in fig. 5.6b. As the images only achieved a FID score of 58,76, which is significantly higher than the SN-GAN achieved for the same data set, it was assumed that the WGAN-GP would not achieve a lowed FID score on images with resolution 32x32x1 than the SN-GAN. We can clearly see that the images produced by the WGAN-GP is of lower quality than SN-GAN generated images. It was also assumed that the FID score achieved



(a) FID score achieved for each checkpoint.

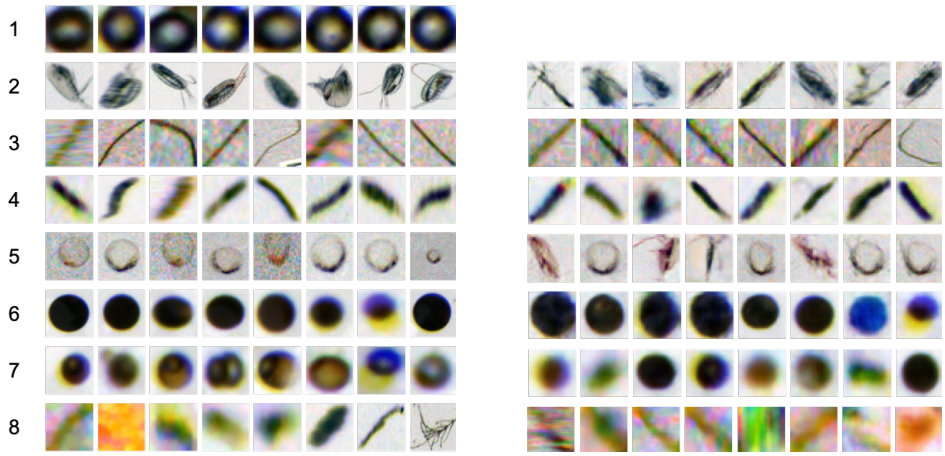


(b) Generated images, FID = 58,76.

Figure 5.6: Results from training the WGAN-GP on images with resolution 32x32x3.

5.3.4 Choice of GAN

Table 5.5 summarizes the results from each GAN test. We can see that the SN-GAN outperformed the other GAN architectures. Due to hardware limitations, no GAN were able to be trained on the 64x64x3 images. As the highest scored accomplished be the COAP-Net after the 64x64x3 images were 32x32x3 and 32x32x1, we concluded that the results from the SN-GAN were sufficient for GAN-based oversampling testing. Fig. 5.7b shows example images for each class generated by the SN-GAN, while fig 5.7a shows example images for each class from the SilCam training data set. Each row contains images from the following classes: 1: bubble, 2: copepod, 3: diatom_chain, 4: faecal_pellets, 5: fish_egg, 6: oil, 7: oily_gas and 8: other. As the SN-GAN is not trained on the largest class in the SilCam dataset, it has one row less than the SilCam images. We can see from the figure that the generated images looks very similar to the training set.



(a) SilCam images. **(b)** Generated images.

Figure 5.7: Example images with resolution 64x64x3, where each row represents a class from the SilCam dataset.

Table 5.6: Results from different class-imbalance mitigation methods for 32x32x3.

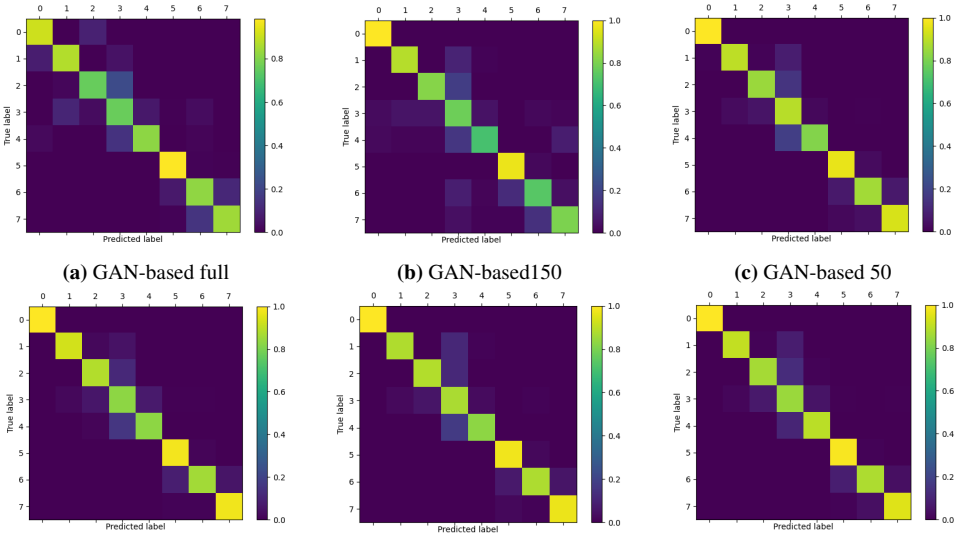
METHOD	ACCURACY	PRECISION	F1-SCORE
RANDOM 50 OVERSAMPLING	91.9933	92.0414	91.9876
RANDOM 150 OVERSAMPLING	91.9099	91.9250	91.9070
SN-GAN 50 OVERSAMPLING	91.4095	91.5358	91.4388
RANDOM FULL OVERSAMPLING	91.3261	91.4008	91.3250
ORIGINAL	90.8257	90.8121	90.7901
SN-GAN 150 OVERSAMPLING	90.3253	90.3543	90.3085
RANDOM FULL UNDERSAMPLING	86.3219	87.9693	86.2835
SN-GAN FULL OVERSAMPLING	86.1551	86.6990	86.2024

5.4 Comparison of Methods to Address Class Imbalance

5.4.1 Results from training data with resolution 32x32x3

The performance of difference methods for addressing class imbalance on training data with image resolution 32x32x3 are shown in table 5.6. Regarding the overall performance of different methods, random 50 oversampling emerged as the best method. We can see from the table that only four methods increased the overall evaluation performance: all three random oversampling methods and the GAN-Based 50 oversampling method. It is interesting to see all random oversampling methods increasing evaluation performance, while GAN-Based oversampling increased performance for 50-oversampling, while decreased performance for 150- and full-oversampling. In figure 5.8 the confusion matrices for each of the oversampling methods are shown. We can see from the matrices that the misclassification increases with the number of oversampled images for both random oversampling and GAN-based oversampling.

Table 5.7a and 5.7b shows the results for the four classes in which we are the most interested, for the SN-GAN 50 oversampling method and the random 50 oversampling method. As we can see, the difference in average F1 score for the four classes are almost identical, especially for the micro-averaged F1 score. We see that the results for class 0 and 1 are better for 50 random oversampling, while the results for class 2 and 4 are better for 50 GAN-based oversampling. Table 5.7c shows an equivalent table for the 150 random oversampling method. As we can see from the results, 150 random oversampling actually achieved a higher micro-F1 score than the 50 oversampling method for the four classes.



(a) GAN-based full **(b)** GAN-based150 **(c)** GAN-based 50

(d) Random full **(e)** Random 150 **(f)** Random 50

Figure 5.8: Confusion matrices for for the different oversampling methods, for image resolution 32x32x3.

Table 5.7: Results for the four important classes, for image resolution 32x32x3.

CLASS	PR	RE	F1
0	0.9706	1.0000	0.9851
1	0.9184	0.9000	0.9091
2	0.8720	0.8516	0.8617
4	0.8986	0.8158	0.8552
MACRO	0.9149	0.8919	0.9028
MICRO	0.9014	0.8724	0.8864

(a) 50 GAN-Based oversampled

CLASS	PR	RE	F1
0	1.0000	1.0000	1.0000
1	0.9479	0.9100	0.9286
2	0.8409	0.8672	0.8539
4	0.8095	0.8947	0.8500
MACRO	0.8996	0.9180	0.9081
MICRO	0.8811	0.8991	0.8895

(b) 50 Random oversampled

CLASS	PR	RE	F1
0	1.0000	1.0000	1.0000
1	0.9263	0.8800	0.9026
2	0.8760	0.8828	0.8794
4	0.8630	0.8290	0.8456
MACRO	0.9163	0.8980	0.9069
MICRO	0.9001	0.9913	0.8905

(c) 150 Random oversampled

Table 5.8: Results from different class-imbalance mitigation methods for 32x32x1.

METHOD	ACCURACY	PRECISION	F1-SCORE
RANDOM FULL OVERSAMPLING	89.6580	89.5735	89.5890
RANDOM 50 OVERSAMPLING	89.4078	89.4495	89.3652
ORIGINAL	89.1576	89.1724	89.0341
RANDOM 150 OVERSAMPLING	88.9908	89.0048	88.9796
SN-GAN 50 OVERSAMPLING	88.4070	88.6306	88.4387
SN-GAN 150 OVERSAMPLING	85.8215	86.0544	85.8346
RANDOM FULL UNDERSAMPLING	82.8190	85.6297	82.8445
SN-GAN FULL OVERSAMPLING	81.5680	82.2457	81.6112

5.4.2 Results from training data with resolution 32x32x1

Although overall classification performance for image resolution 32x32x1 were lower than for 32x32x3, the same oversampling tests were done, and the results are shown in table 5.8. As we can see from the results, also here the random oversampling methods were superior to the others, and undersampling shows poor performance. However, the impact from the oversampling were not as effective as for 32x32x3, and for some cases it had a negative effect. Here we can see that the GAN-based oversampling worsened the overall classification performance. Table 5.9 shows the results for the four important classes for the random oversampling technique and the GAN-based oversampling technique with the best overall performance. As we can see from the results, the random oversampling method is superior to the GAN-based oversampling. Even though the FID score of the 32x32x1 images were lower than the 32x32x3 images, the GAN-based oversampling for 32x32x1 did not produce good results. The classification performance of the grayscale images is in general poorer than the performance of the RGB images, which suggests that the colors are important for distinguishing between the classes. Even though the GAN achieved a low FID score for the grayscale images, it is safe to assume that the generated images introduced increased class overlap in the oversampled dataset, which in return resulted in poorer classification performance. An even lower FID score might might reduce the class overlap in the generated images, and help during the classification process.

Table 5.9: Results for the four important classes, for image resolution 32x32x1.

CLASS	PR	RE	F1
0	1.0000	1.0000	1.0000
1	0.9663	0.8600	0.9101
2	0.8667	0.8125	0.8387
4	0.7826	0.7105	0.7448
MACRO	0.9039	0.8458	0.8734
MICRO	0.8903	0.8220	0.8545

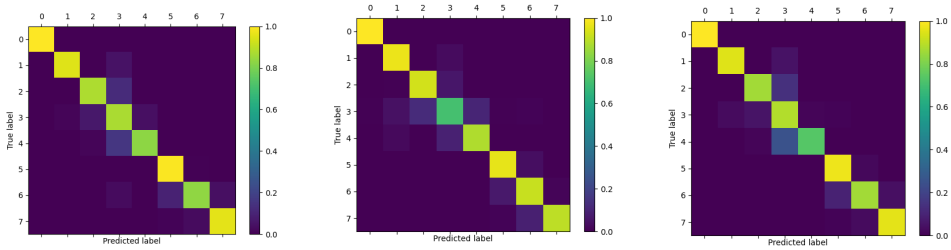
(a) 50 GAN-Based oversampled

CLASS	PR	RE	F1
0	1.0000	1.0000	1.0000
1	0.9307	0.9400	0.9353
2	0.8626	0.8828	0.8726
4	0.7692	0.7895	0.7792
MACRO	0.8906	0.9031	0.8968
MICRO	0.8752	0.8902	0.8826

(b) 50 Random oversampled

5.4.3 Results from training data with resolution 64x64x3

For the training data with image resolution 64x64x3, random full oversampling and random undersampling were tested. Due to time limit and the long training time required for the 64x64x3 data set, no further tests were done. Testing on 64x64x3 were also not prioritized as no it was not possible to perform GAN-based oversampling due to the hardware limitations, and it was therefore not possible to use this dataset for compering GAN-based oversampling to other methods. The results from full oversampling and undersampling are shown in table 5.10, and the confusion matrices are shown in fig. 5.9. For the image resolution 64x64x3 we see that both full undersampling and full oversampling has poor performance. Classification reports such as table 5.3 for all tests are found in the appendix.



(a) No resampling **(b)** Full undersampling **(c)** Full oversampling

Figure 5.9: Confusion matrices for for the different oversampling methods, for image resolution 64x64x3.

Table 5.10: Results from different class-imbalance mitigation methods for 64x64x3.

METHOD	ACCURACY	PRECISION	F1-SCORE	TRAINING TIME
ORIGINAL	92.4937	92.4807	92.4735	370 MIN
RANDOM OVERSAMPLING	91.6597	91.7357	91.6134	550 MIN
RANDOM UNDERSAMPLING	88.4904	89.7180	88.4780	73 MIN

Chapter 6

Conclusion

In this thesis we have studied the impact of class imbalance on classification performance of the data set SilCam, an in-situ imagery planktonic data set, and investigated the effectiveness of different sampling methods for addressing the issue. Most notable, we have investigated the use of generative adversarial networks (GANs) for synthetic image generation in a GAN-based sampling technique and compared the method to the traditional sampling strategies random minority oversampling and random majority undersampling. Our findings show that partial random oversampling outperforms the other methods with respect to F1-score. However, in some cases, GAN-based partial oversampling achieves equivalent results to random oversampling. With these findings we can conclude that GAN-based oversampling could in some cases be a good alternative for mitigating class imbalance.

In our experiments the generated images by the GAN achieved a fairly high FID score of 30,85. It is likely that a higher F1 score would result in better classification performance by the classifier, as the FID score represents the generated images quality and diversity. This could be achieved by tuning hyperparameters or an increase in batch size during training of the GAN. Performance could also increase by testing different GAN architectures.

Another of our findings is the impact of image resolution on classification performance. Grayscale images resulted in general in poor performance, and the class mitigation methods did not have as good impact on the classification task as for the RGB images. This is most likely due to an increased class overlap for the grayscale images.

For future work, methods which considers within-class imbalance such as cluster-based oversampling should be investigated in order to compare performance. The GAN-based oversampling method should also be improved by reducing the FID score. The method could also be integrated with data cleaning techniques. Computational power available would also need to be higher than that of this thesis, in order to get full the potential of GANs.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: `https://www.tensorflow.org/`. software available from tensorflow.org.

AILARON, . Autonomous imaging and learning ai robot identifying plankton taxa in-situ. URL: `https://www.ntnu.edu/web/ailaron`.

Benfield, M.C., Grosjean, P., Culverhouse, P.F., Irigoien, X., Sieracki, M.E., Lopez-Urrutia, A., Dam, H.G., Hu, Q., Davis, C.S., Hansen, A., et al., 2007. Rapid: research on automated plankton identification. Oceanography 20, 172–187.

Buda, M., Maki, A., Mazurowski, M.A., 2018. A systematic study of the class imbalance problem in convolutional neural networks. Neural Networks 106, 249–259.

Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P., 2002. Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research 16, 321–357.

Chawla, N.V., Lazarevic, A., Hall, L.O., Bowyer, K.W., 2003. Smoteboost: Improving prediction of the minority class in boosting, in: European conference on principles of data mining and knowledge discovery, Springer. pp. 107–119.

Dal Pozzolo, A., Caelen, O., Bontempi, G., 2015. When is undersampling effective in unbalanced classification tasks?, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer. pp. 200–215.

Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L., 2009. ImageNet: A Large-Scale Hierarchical Image Database, in: CVPR09.

Douzas, G., Bacao, F., 2018. Effective data generation for imbalanced learning using conditional generative adversarial networks. Expert Systems with applications 91, 464–471.

Du, K.L., Swamy, M.N., 2013. Neural networks and statistical learning. Springer Science & Business Media.

Elkan, C., 2001. The foundations of cost-sensitive learning, in: International joint conference on artificial intelligence, Lawrence Erlbaum Associates Ltd. pp. 973–978.

Fawcett, T., Provost, F., 1997. Adaptive fraud detection. Data mining and knowledge discovery 1, 291–316.

Fréchet, M., 1957. Sur la distance de deux lois de probabilité. COMPTES RENDUS HEBDOMADAIRES DES SEANCES DE L ACADEMIE DES SCIENCES 244, 689–692.

Freund, Y., Schapire, R.E., 1995. A desicion-theoretic generalization of on-line learning and an application to boosting, in: European conference on computational learning theory, Springer. pp. 23–37.

Fuglede, B., Topsoe, F., 2004. Jensen-shannon divergence and hilbert space embedding, in: International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings., pp. 31–.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets, in: Advances in neural information processing systems, pp. 2672–2680.

Grzymala-Busse, J.W., Goodwin, L.K., Grzymala-Busse, W.J., Zheng, X., 2004. An approach to imbalanced data sets based on changing rule strength, in: Rough-neural computing. Springer, pp. 543–553.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C., 2017. Improved training of wasserstein gans, in: Advances in neural information processing systems, pp. 5767–5777.

Guo, H., Viktor, H.L., 2004. Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. ACM Sigkdd Explorations Newsletter 6, 30–39.

Han, H., Wang, W.Y., Mao, B.H., 2005. Borderline-smote: a new over-sampling method in imbalanced data sets learning, in: International conference on intelligent computing, Springer. pp. 878–887.

He, H., Bai, Y., Garcia, E.A., Li, S., 2008. Adasyn: Adaptive synthetic sampling approach for imbalanced learning, in: 2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence), IEEE. pp. 1322–1328.

He, H., Garcia, E.A., 2009. Learning from imbalanced data. IEEE Transactions on knowledge and data engineering 21, 1263–1284.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S., 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium, in: Advances in Neural Information Processing Systems, pp. 6626–6637.

Huang, C., Li, Y., Change Loy, C., Tang, X., 2016. Learning deep representation for imbalanced classification, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 5375–5384.

Jaffe, J.S., 2014. Underwater optical imaging: the past, the present, and the prospects. IEEE Journal of Oceanic Engineering 40, 683–700.

Japkowicz, N., 2000. The class imbalance problem: Significance and strategies, in: Proc. of the Int'l Conf. on Artificial Intelligence.

Japkowicz, N., Stephen, S., 2002. The class imbalance problem: A systematic study. Intelligent data analysis 6, 429–449.

Jo, T., Japkowicz, N., 2004. Class imbalances versus small disjuncts. ACM Sigkdd Explorations Newsletter 6, 40–49.

Kaggle, . Credit card fraud detection dataset, kaggle. `https://www.kaggle.com/mlg-ulb/creditcardfraud`.

Kelleher, J.D., Mac Namee, B., D'arcy, A., 2015. Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies. MIT press.

Koza, J.R., Bennett, F.H., Andre, D., Keane, M.A., 1996. Automated design of both the topology and sizing of analog electrical circuits using genetic programming, in: Artificial Intelligence in Design'96. Springer, pp. 151–170.

Krizhevsky, A., Hinton, G., et al., 2009. Learning multiple layers of features from tiny images. Technical Report. Citeseer.

Kubat, M., Matwin, S., et al., 1997. Addressing the curse of imbalanced training sets: one-sided selection, in: Icml, Nashville, USA. pp. 179–186.

Kukar, M., Kononenko, I., et al., 1998. Cost-sensitive learning with neural networks., in: ECAI, pp. 445–449.

Laurikkala, J., 2001. Improving identification of difficult small classes by balancing class distribution, in: Conference on Artificial Intelligence in Medicine in Europe, Springer. pp. 63–66.

Lee, H., Park, M., Kim, J., 2016. Plankton classification on imbalanced large scale database via convolutional neural networks with transfer learning, in: 2016 IEEE international conference on image processing (ICIP), IEEE. pp. 3713–3717.

Lee, J., Park, K., 2019. Gan-based imbalanced data intrusion detection system. Personal and Ubiquitous Computing , 1–8.

Lei, K., Xie, Y., Zhong, S., Dai, J., Yang, M., Shen, Y., 2019. Generative adversarial fusion network for class imbalance credit scoring. Neural Computing and Applications , 1–12.

Ling, C.X., Sheng, V.S., 2010. Class Imbalance Problem. Springer US, Boston, MA. pp. 171–171. URL: https://doi.org/10.1007/978-0-387-30164-8_110, doi:10.1007/978-0-387-30164-8_110.

Liu, X.Y., Wu, J., Zhou, Z.H., 2008. Exploratory undersampling for class-imbalance learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39, 539–550.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., Bousquet, O., 2018. Are gans created equal? a large-scale study, in: Advances in neural information processing systems, pp. 700–709.

Mac Namee, B., Cunningham, P., Byrne, S., Corrigan, O.I., 2002. The problem of bias in training data in regression problems in medical decision support. Artificial intelligence in medicine 24, 51–70.

MacLeod, N., Benfield, M., Culverhouse, P., 2010. Time to automate identification. Nature 467, 154.

Mani, I., Zhang, I., 2003. knn approach to unbalanced data distributions: a case study involving information extraction, in: Proceedings of workshop on learning from imbalanced datasets.

Martin Arjovsky, S., Bottou, L., 2017. Wasserstein generative adversarial networks, in: Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia.

Mirza, M., Osindero, S., 2014. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 .

Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y., 2018. Spectral normalization for generative adversarial networks. arXiv preprint arXiv:1802.05957 .

Mullick, S.S., Datta, S., Das, S., 2019. Generative adversarial minority oversampling, in: Proceedings of the IEEE International Conference on Computer Vision, pp. 1695–1704.

Orenstein, E.C., Beijbom, O., Peacock, E.E., Sosik, H.M., 2015. Whoi-plankton-a large scale fine grained visual recognition benchmark dataset for plankton classification. arXiv preprint arXiv:1510.00745 .

O'Shea, K., Nash, R., 2015. An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 .

Pawlak, Z., 1998. Rough set theory and its applications to data analysis. Cybernetics & Systems 29, 661–688.

Radford, A., Metz, L., Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 .

Saad, A., Davies, E., Stahl, A., 2019. Recent advances in visual sensing and machine learning techniques for in-situ plankton-taxa classification.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X., 2016. Improved techniques for training gans, in: Advances in neural information processing systems, pp. 2234–2242.

Seiffert, C., Khoshgoftaar, T.M., Van Hulse, J., Napolitano, A., 2009. Rusboost: A hybrid approach to alleviating class imbalance. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 40, 185–197.

Sheng, V.S., Ling, C.X., 2006. Thresholding for making classifiers cost-sensitive, in: AAAI, pp. 476–481.

Sieracki, M.E., Benfield, M., Hanson, A., Davis, C., Pilskaln, C.H., Checkley, D., Sosik, H.M., Ashjian, C., Culverhouse, P., Cowen, R., et al., 2010. Optical plankton imaging and analysis systems for ocean observation. Proceedings of ocean Obs 9, 21–25.

Stefanowski, J., Wilk, S., 2006. Rough sets for handling imbalanced data: combining filtering and rule-based classifiers. Fundamenta Informaticae 72, 379–391.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2016. Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826.

Tezduyar, T.E., Mittal, S., Ray, S., Shih, R., 1992. Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements. Computer Methods in Applied Mechanics and Engineering 95, 221–242.

Theis, L., Oord, A.v.d., Bethge, M., 2015. A note on the evaluation of generative models. arXiv preprint arXiv:1511.01844 .

Tomek, I., et al., 1976. Two modifications of cnn. .

Van Asch, V., 2013. Macro-and micro-averaged evaluation measures [[basic draft]]. Belgium: CLiPS 49.

Van Erven, T., Harremos, P., 2014. Rényi divergence and kullback-leibler divergence. IEEE Transactions on Information Theory 60, 3797–3820.

Wang, C., Yu, Z., Zheng, H., Wang, N., Zheng, B., 2017. Cgan-plankton: towards large-scale imbalanced class generation and fine-grained classification, in: 2017 IEEE International Conference on Image Processing (ICIP), IEEE. pp. 855–859.

Wang, Q., Zhou, X., Wang, C., Liu, Z., Huang, J., Zhou, Y., Li, C., Zhuang, H., Cheng, J.Z., 2019. Wgan-based synthetic minority over-sampling technique: Improving semantic fine-grained classification for lung nodules in ct images. IEEE Access 7, 18450–18463.

Wilson, D.R., Martinez, T.R., 2000. Reduction techniques for instance-based learning algorithms. Machine learning 38, 257–286.

Witman, S., 2017. World's biggest oxygen producers living in swirling ocean waters. Eos, 98. Https://doi.org/10.1029/2017EO081067. Published on 13 September 2017.

Appendix

Classification results

Table 6.1: Results without class-imbalance mitigation for image resolution 64x64x1.

	Precision	Recall	F1-Score	Validation Images
Class 0	0.9705	1.0000	0.9850	33
Class 1	0.9020	0.9200	0.9109	100
Class 2	0.8644	0.7969	0.8293	128
Class 3	0.8007	0.8419	0.8208	291
Class 4	0.7794	0.6974	0.7361	76
Class 5	0.9625	0.9722	0.9673	396
Class 6	0.8000	0.7123	0.7536	73
Class 7	0.9151	0.9510	0.9327	102
Accuracy			0.8832	1199
Macro avg	0.8743	0.8615	0.8670	1199
weighted avg	0.8824	0.8832	0.8822	1199

Table 6.2: Results without class-imbalance mitigation for image resolution 32x32x3.

	Precision	Recall	F1-Score	Validation Images
Class 0	1.0000	1.0000	1.0000	33
Class 1	0.9293	0.9200	0.9246	100
Class 2	0.8271	0.8594	0.8429	128
Class 3	0.8622	0.8385	0.8502	291
Class 4	0.8026	0.8026	0.8026	76
Class 5	0.9774	0.9823	0.9799	396
Class 6	0.8955	0.8219	0.8571	73
Class 7	0.9091	0.9804	0.9434	102
Accuracy			0.9083	1199
Macro avg	0.9004	0.9006	0.9001	1199
weighted avg	0.9081	0.9083	0.9079	1199

Table 6.3: Results without class-imbalance mitigation for image resolution 32x32x1.

	Precision	Recall	F1-Score	Validation Images
Class 0	0.9706	1.0000	0.9851	33
Class 1	0.9462	0.8800	0.9119	100
Class 2	0.8462	0.8594	0.8527	128
Class 3	0.8495	0.8144	0.8316	291
Class 4	0.7356	0.8421	0.7853	76
Class 5	0.9538	0.9899	0.9715	396
Class 6	0.8597	0.6712	0.7539	73
Class 7	0.8889	0.9412	0.9143	102
Accuracy			0.8916	1199
Macro avg	0.8813	0.8748	0.8758	1199
weighted avg	0.8917	0.8916	0.8903	1199

Table 6.4: Results of full undersampling for 64x64x3.

class	Precision	Recall	F1-Score	Validation Images
0	0.91667	1.00000	0.95652	33
1	0.85841	0.97000	0.91080	100
2	0.76774	0.92969	0.84099	128
3	0.91928	0.70447	0.79767	291
4	0.69072	0.88158	0.77457	76
5	0.97943	0.96212	0.97070	396
6	0.72043	0.91781	0.80723	73
7	0.98925	0.90196	0.94359	102
Accuracy			0.88490	1199
macro avg	0.85524	0.90845	0.87526	1199
weighted avg	0.89718	0.88490	0.88478	1199

Table 6.5: Results of full oversampling for 64x64x3.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.97059	1.00000	0.98507	33
1	0.92233	0.95000	0.93596	100
2	0.87302	0.85938	0.86614	128
3	0.85714	0.88660	0.87162	291
4	0.93333	0.73684	0.82353	76
5	0.97229	0.97475	0.97352	396
6	0.82895	0.86301	0.84564	73
7	0.96078	0.96078	0.96078	102
ACCURACY			0.91660	1199
MACRO AVG	0.91480	0.90392	0.90778	1199
WEIGHTED AVG	0.91736	0.91660	0.91613	1199

Table 6.6: Results of full oversampling for 32x32x3.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	1.00000	1.00000	1.00000	33
1	0.93069	0.94000	0.93532	100
2	0.86260	0.88281	0.87259	128
3	0.86022	0.82474	0.84211	291
4	0.76923	0.78947	0.77922	76
5	0.96030	0.97727	0.96871	396
6	0.80597	0.73973	0.77143	73
7	0.87850	0.92157	0.89952	102
ACCURACY			0.89658	1199
MACRO AVG	0.88344	0.88445	0.88361	1199
WEIGHTED AVG	0.89574	0.89658	0.89589	1199

Table 6.7: Results of 150 oversampling for 32x32x3.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	1.00000	1.00000	1.00000	33
1	0.92632	0.88000	0.90256	100
2	0.87597	0.88281	0.87938	128
3	0.86689	0.87285	0.86986	291
4	0.86301	0.82895	0.84564	76
5	0.97980	0.97980	0.97980	396
6	0.84211	0.87671	0.85906	73
7	0.95192	0.97059	0.96117	102
ACCURACY			0.91910	1199
MACRO AVG	0.91325	0.91146	0.91218	1199
WEIGHTED AVG	0.91925	0.91910	0.91907	1199

Table 6.8: Results of 50 oversampling for 32x32x3.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	1.00000	1.00000	1.00000	33
1	0.94792	0.91000	0.92857	100
2	0.84091	0.86719	0.85385	128
3	0.88530	0.84880	0.86667	291
4	0.80952	0.89474	0.85000	76
5	0.97756	0.98990	0.98369	396
6	0.88889	0.87671	0.88276	73
7	0.95098	0.95098	0.95098	102
ACCURACY			0.91993	1199
MACRO AVG	0.91263	0.91729	0.91456	1199
WEIGHTED AVG	0.92041	0.91993	0.91988	1199

Table 6.9: Results of full GAN-based oversampling for 32x32x3.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.73171	0.90909	0.81081	33
1	0.73109	0.87000	0.79452	100
2	0.88991	0.75781	0.81857	128
3	0.83083	0.75945	0.79354	291
4	0.77500	0.81579	0.79487	76
5	0.97990	0.98485	0.98237	396
6	0.67416	0.82192	0.74074	73
7	0.88660	0.84314	0.86432	102
ACCURACY			0.86155	1199
MACRO AVG	0.81240	0.84526	0.82497	1199
WEIGHTED AVG	0.86699	0.86155	0.86202	1199

Table 6.10: Results of 150 GAN-based oversampling for 32x32x3.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.94118	0.96970	0.95522	33
1	0.92708	0.89000	0.90816	100
2	0.82707	0.85938	0.84291	128
3	0.86007	0.86598	0.86301	291
4	0.86111	0.81579	0.83784	76
5	0.97970	0.97475	0.97722	396
6	0.83582	0.76712	0.80000	73
7	0.87273	0.94118	0.90566	102
ACCURACY			0.90325	1199
MACRO AVG	0.88809	0.88549	0.88625	1199
WEIGHTED AVG	0.90354	0.90325	0.90308	1199

Table 6.11: Results of 50 GAN-based oversampling for 32x32x3.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.97059	1.00000	0.98507	33
1	0.91837	0.90000	0.90909	100
2	0.87200	0.85156	0.86166	128
3	0.86379	0.89347	0.87838	291
4	0.89855	0.81579	0.85517	76
5	0.98205	0.96717	0.97455	396
6	0.78750	0.86301	0.82353	73
7	0.94118	0.94118	0.94118	102
ACCURACY			0.91410	1199
MACRO AVG	0.90425	0.90402	0.90358	1199
WEIGHTED AVG	0.91536	0.91410	0.91439	1199

Table 6.12: Results of full undersampling for 32x32x3.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.89189	1.00000	0.94286	33
1	0.81356	0.96000	0.88073	100
2	0.78667	0.92188	0.84892	128
3	0.90431	0.64948	0.75600	291
4	0.68367	0.88158	0.77011	76
5	0.97895	0.93939	0.95876	396
6	0.60952	0.87671	0.71910	73
7	0.94118	0.94118	0.94118	102
ACCURACY			0.86322	1199
MACRO AVG	0.82622	0.89628	0.85221	1199
WEIGHTED AVG	0.87969	0.86322	0.86283	1199

Table 6.13: Results of full oversampling for 32x32x1.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	1.00000	1.00000	1.00000	33
1	0.93069	0.94000	0.93532	100
2	0.86260	0.88281	0.87259	128
3	0.86022	0.82474	0.84211	291
4	0.76923	0.78947	0.77922	76
5	0.96030	0.97727	0.96871	396
6	0.80597	0.73973	0.77143	73
7	0.87850	0.92157	0.89952	102
ACCURACY			0.89658	1199
MACRO AVG	0.88344	0.88445	0.88361	1199
WEIGHTED AVG	0.89574	0.89658	0.89589	1199

Table 6.14: Results of 150 oversampling for 32x32x1.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.94286	1.00000	0.97059	33
1	0.91919	0.91000	0.91457	100
2	0.83871	0.81250	0.82540	128
3	0.81605	0.83849	0.82712	291
4	0.84507	0.78947	0.81633	76
5	0.96742	0.97475	0.97107	396
6	0.77027	0.78082	0.77551	73
7	0.93878	0.90196	0.92000	102
ACCURACY			0.88991	1199
MACRO AVG	0.87979	0.87600	0.87757	1199
WEIGHTED AVG	0.89005	0.88991	0.88980	1199

Table 6.15: Results of 50 oversampling for 32x32x1.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.97059	1.00000	0.98507	33
1	0.89524	0.94000	0.91707	100
2	0.83077	0.84375	0.83721	128
3	0.86194	0.79381	0.82648	291
4	0.77647	0.86842	0.81988	76
5	0.96774	0.98485	0.97622	396
6	0.75000	0.78082	0.76510	73
7	0.94898	0.91176	0.93000	102
ACCURACY			0.89408	1199
MACRO AVG	0.87522	0.89043	0.88213	1199
WEIGHTED AVG	0.89450	0.89408	0.89365	1199

Table 6.16: Results of full GAN-based oversampling for 32x32x1.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.61111	1.00000	0.75862	33
1	0.76271	0.90000	0.82569	100
2	0.81061	0.83594	0.82308	128
3	0.79600	0.68385	0.73567	291
4	0.52381	0.57895	0.55000	76
5	0.97933	0.95707	0.96807	396
6	0.70423	0.68493	0.69444	73
7	0.73786	0.74510	0.74146 102	
ACCURACY			0.81568	1199
MACRO AVG	0.74071	0.79823	0.76213	1199
WEIGHTED AVG	0.82246	0.81568	0.81611	1199

Table 6.17: Results of 150 GAN-based oversampling for 32x32x1.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.73333	1.00000	0.84615	33
1	0.84762	0.89000	0.86829	100
2	0.86777	0.82031	0.84337	128
3	0.80212	0.78007	0.79094	291
4	0.76056	0.71053	0.73469	76
5	0.97222	0.97222	0.97222	396
6	0.65060	0.73973	0.69231	73
7	0.86316	0.80392	0.83249	102
ACCURACY			0.85822	1199
MACRO AVG	0.81217	0.83960	0.82256	1199
WEIGHTED AVG	0.86054	0.85822	0.85835	1199

Table 6.18: Results of 50 GAN-based oversampling for 32x32x1.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	1.00000	1.00000	1.00000	33
1	0.96629	0.86000	0.91005	100
2	0.86667	0.81250	0.83871	128
3	0.80000	0.86598	0.83168	291
4	0.78261	0.71053	0.74483	76
5	0.96701	0.96212	0.96456	396
6	0.72368	0.75342	0.73826	73
7	0.92233	0.93137	0.92683	102
ACCURACY			0.88407	1199
MACRO AVG	0.87857	0.86199	0.86936	1199
WEIGHTED AVG	0.88631	0.88407	0.88439	1199

Table 6.19: Results of full undersampling for 32x32x1.

CLASS	PRECISION	RECALL	F1-SCORE	VALIDATION IMAGES
0	0.97059	1.00000	0.98507	33
1	0.79508	0.97000	0.87387	100
2	0.78289	0.92969	0.85000	128
3	0.88360	0.57388	0.69583	291
4	0.57983	0.90789	0.70769	76
5	0.96579	0.92677	0.94588	396
6	0.50893	0.78082	0.61622	73
7	0.92308	0.82353	0.87047	102
ACCURACY			0.82819	1199
MACRO AVG	0.80122	0.86407	0.81813	1199
WEIGHTED AVG	0.85630	0.82819	0.82844	1199

NTNU

Norwegian University of
Science and Technology