DEPARTMENT OF COMPUTER SCIENCE

Metric Learning Auxiliary Classifier Generative Adversarial Networks for Foraminifera Image Classification and Synthesis

Investigating The Benefits of Phasing Schemes

Dan Salter

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Friday 6th May, 2022

Abstract

In this thesis I propose the Metric Learning Auxiliary Classifier Generative Adversarial Network (MLACGAN) architecture and show it can be used to to accurately classify light microscopy photographs of foraminiferas, as well as synthesise new, plausible images of each species. I first train the model on only real images, achieving an accuracy of 91.66% and F1-score of 84.66 on unseeen test data, surpassing leading results on 416x416 pixel input images. I then show that this model completes its goals when trained with a 50/50 split of real and synthesised data, achieving a test set accuracy of 52.92 and F1-score of 38.54, while also being able to synthesise convincing foram images. I go on to propose several methods for phasing in synthesised data throughout training, in order to bridge the performance gap between these two training schemes, and show that the best of these phasing models achieved an improved test accuracy of 63.77% and F1-score of 50.47, while also synthesising plausible foram images.

- I designed the MLACGAN architecture as a marriage of ACGAN and ResNet50.
- I wrote $> 10,000$ lines of python code, including implementations for multiple CNNs and custom datasets and loss functions in PyTorch.
- I produced a catalogue of graphs and other plots in order to visualise my results.
- I trained dozens of models on BlueCrystal Phase 4, UoB's remote HPC server [19], via remotely coded shell scripts.

Dedication and Acknowledgements

I would firstly like to thank my supervisor, Tilo, for his guidance throughout this project. His mentoring helped me complete what has certainly been the most challenging project of my academic career. Thank you also to Tayfun, for your consistent willingness to provide answers whenever I asked questions. Thank you to my parents; this wouldn't have been possible without you. And finally to Luna, for her continued support, and her willingness to listen to me ramble about AI and fossils.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

Dan Salter, Friday 6th May, 2022

Contents

List of Figures

List of Tables

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Tilo Burghardt.

Supporting Technologies

- I used PyTorch and TorchVision python libraries to code all my models.
- All plots were produced using MatPlotLib and Seaborn libraries.
- My model pipeline was designed in the diagrams.net desktop application.
- My generator CNN diagram was designed in latex using a library I downloaded from this Git repo at this address: https://github.com/HarisIqbal88/PlotNeuralNet
- I trained all my models on BlueCrystal Phase 4.

Chapter 1

Executive Summary

Today, climate change is universally accepted to be one of the greatest existential threats facing humanity, but it hasn't always been viewed this way. After the discovery of global warming in 1938 it took over 50 years for the theory to be accepted as fact by governments worldwide. This sway in opinion was driven by the increasing weight of climate research, much of which was palaeoclimateological research conducted via the study of ancient climate proxies. Fossilised planktonic foraminifera, a phylum of microscopic, ocean-dwelling protists, are one such example which have taught us more about ancient ocean climates than any other resource. We can infer the ancient climate from the distributions of various foram species found in ocean sediment cores, which is why the demand for fast and accurate foram species classification is ever present. Recent work, facilitated by the collation of the publicly available Endless Forams dataset, has seen great success using CNNs to classify these light microscopy foram images into their respective species. However, the dataset is limited in size and severely imbalanced between species, so it would be in the interests of palaeoclimateologists if a method for synthesising new realistic foram images was developed. This is why, in this paper, I propose a novel generative architecture, MLACGAN, which is capable of synthesising a vast number of new foram images which resemble each of the 35 species present in the Endless Forams dataset. Additionally, this model has been shown to classify previously unseen foram examples with up to 63.77% accuracy.

MLACGAN consists of two artificial neural networks, pitted against each other in an adversarial zero-sum game, wherein the first network, the generator, seeks to synthesise images which the second network, the discriminator, fails to identify as being synthesised. To achieve this deceptive goal, the generator will learn to synthesise images which are indistinguishable from the real images in the training set. Concurrently, the generator and discriminator work together to ensure the synthesised images resemble a specific target species; the generator is doing well if the discriminator is able to accurately classify the image as belonging to its target species. A major novel contribution of this paper, and one that sets it apart from the established ACGAN architecture, is the addition of a tertiary metric learning mechanism. Here, D projects images into a high-dimensional latent space with the aim of clustering embeddings originating from the same species. Classifications of unseen data can then be drawn by projecting into the latent space then executing the K-nearest neighbours algorithm on its neighbouring embeddings.

Generative classification models are notoriously unstable to train, commonly suffering from volatile generative and classification performance. This makes sense since, in the early stages of training, the discriminator is attempting to learn to classify low quality synthesised images, which are essentially just noise. To mitigate this instability I have proposed a novel training approach, which reduces the contribution of synthesised images in D's training during early training. As the quality of synthesised images improves throughout training, their contribution is gradually phased in. Phasing in generated data in this way led to a 5.05% increase in test set accuracy. Furthermore, I have shown that this novel phasing approach aids in the synthesis of high quality foram images.

At a high level, this project aims to develop a cutting edge generative adversarial network capable of synthesising images of forams, which can be used to assist in climate research, as well as accurately classifying real foram images. These goals can be broken down into more technical sub-goals:

1. Design a novel MLACGAN architecture capable of the synthesis of convincing foram images as well as image classification via clustering in an optimised latent space.
2. Train MLACGAN as a classifier on images sampled from the Endless Forams dataset.
3. Train MLACGAN with a 50/50 blend of real and generated images.
4. Develop and experiment with novel phasing schemes for blending in the contribution of generated images over the course of many epochs.
5. Statistically analyse the classification performance of MLACGAN.
6. Qualitatively assess the images synthesised by MLACGAN.

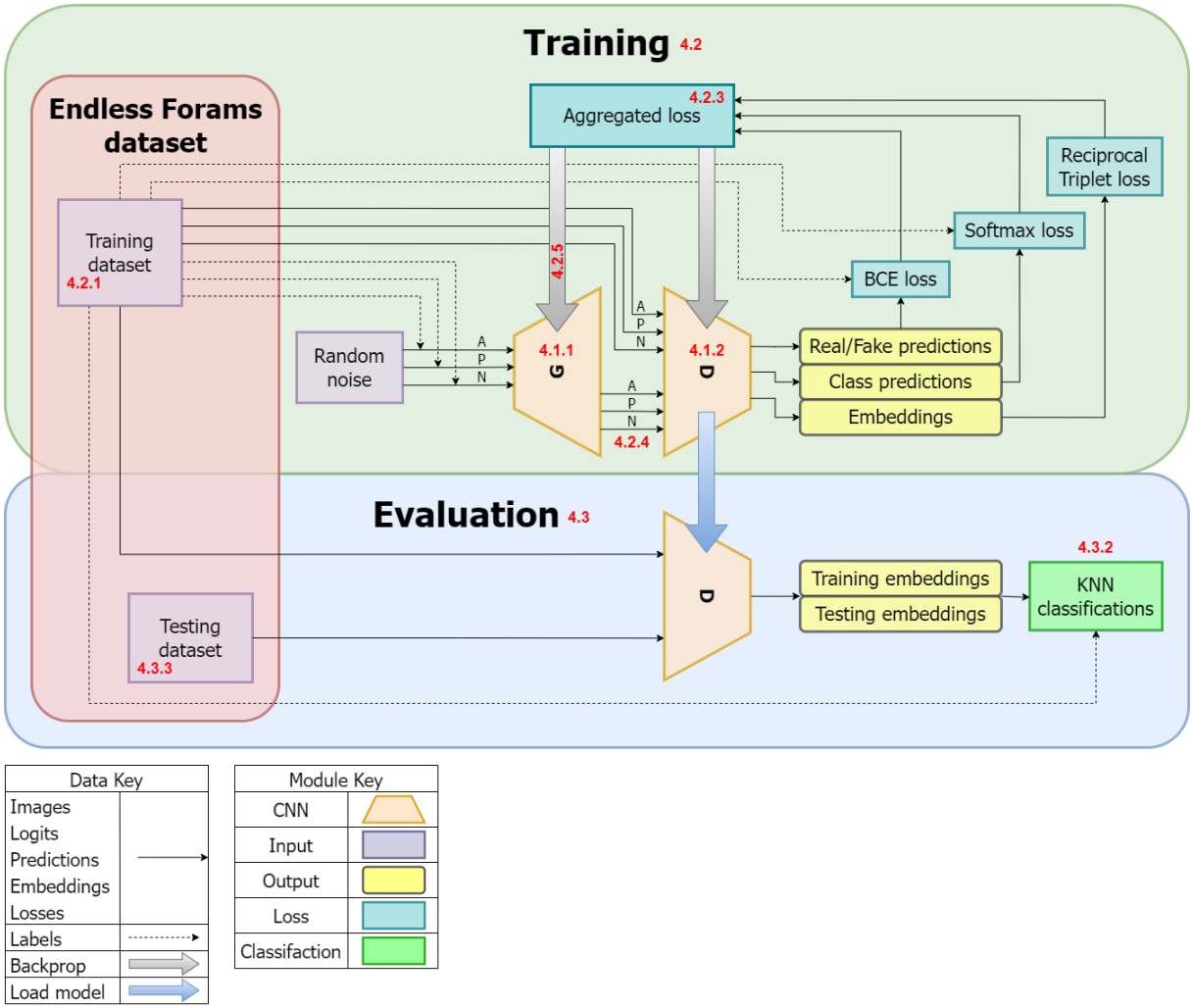

Figure 1.1: **The MLACGAN training and evaluation pipeline.** Each component of the pipeline is labelled with its corresponding section in the Project Execution chapter.

Chapter 2

Contextual Background

Human activity is driving changes in the global climate at an unprecedented rate, causing a global average temperature rise in excess of 1℃ since the pre-industrial era [22]. Not only has this dire transformation been rapid, but the rate of change continues to accelerate, with the global average rate of warming reaching 1.8℃ per decade since 1981, over double the 1881-1980 rate of 0.8℃ per decade [8]. The impacts of this systemic warming have already taken hold on the natural world. For example, coral bleaching events, caused by abnormally warm and acidic sea water, devastated global coral populations for the first time in 1998 and another four times since [67], demonstrating the resultant decline of natural habitats. The continuation of human civilisation depends on the maintained balance in natural habitats such as coral reefs, meaning their destabilisation poses an indirect existential threat to humanity. "More than 166,000 people died due to extreme temperatures between 1998-201"[68], showing that the impacts of systemic warming can also be life-threateningly direct.

Perhaps even more stark than these observations from recent times are predictions for the future impacts of climate change. It is predicted that over 410 million people will be displaced by a climate-change driven rise in sea levels of over one meter [27], with a rise of 2.5 meters possible by 2100 [60]. It has been widely speculated that this will result in the largest mass migration in human history, with the potential to "force 216 million people across six world regions to move within their countries by 2050" [11]. Furthermore, the World Health Organisation has predicted that "Between 2030 and 2050, climate change is expected to cause approximately 250,000 additional deaths per year, from malnutrition, malaria, diarrhoea and heat stress" [50], confirming the gravity of the threat to human life.

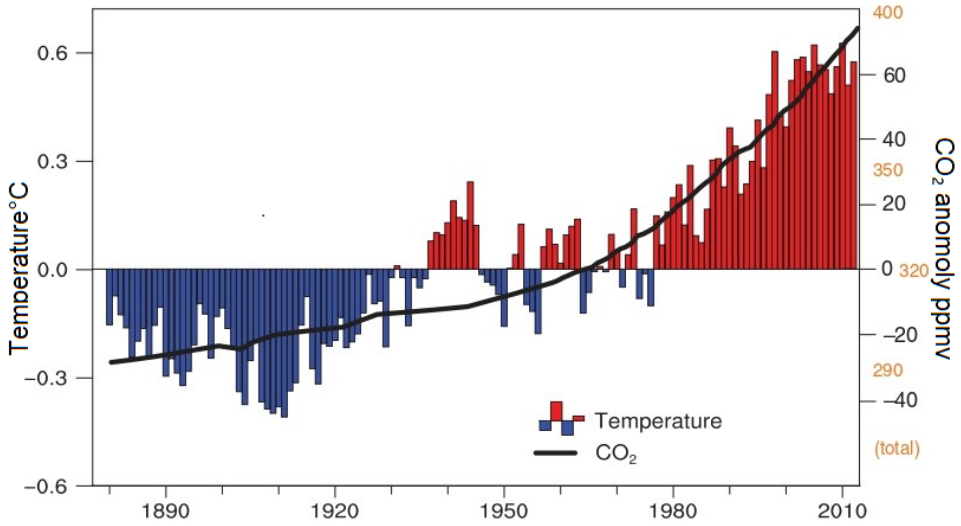

Figure 2.1: **"Estimated changes in annual global mean surface temperatures (℃, color bars) and CO_2 concentrations (thick black line) since 1880"** [65]. The temperature axis shows the annual mean temperature's deviation from the 1901-2000 average. "Carbon dioxide concentrations since 1957 are from direct measurements at Mauna Loa, Hawaii, whereas earlier estimates are derived from ice core records."

These damning statistics have been selected for their eye catching nature since, in a era when people's attentions are sought after more than ever, **shocking the public has become an essential step in motivating them to act en masse**. With an ever accelerating disaster unfolding before us, quantified predictions for the future of the global climate are paramount. In addition to motivating change at a societal level, **accurate climate predictions play an integral role in informing government policy** [52]. One shining example of a scientific discovery which led to a universal political paradigm shift is the discovery of the historical correlation between atmospheric carbon dioxide levels and global average temperatures. This realisation is accredited to Guy Stewart Callendar, who in 1938 identified a consistent rise in global temperatures over the previous 50 years and proposed the novel, but now widely accepted, theory that rising atmospheric carbon dioxide levels could be to blame for this warming [18] (see Figure 2.1). His findings were widely rejected in the scientific community, let alone in political arenas, and it was only after an undeniable body of palaeoclimatological evidence was amassed that scientists, and later politicians, were swayed to commit to a clean and carbon-free future.

Figure 2.2: **(Left)** A pair of scientists sampling a core from a coral reef [3]. **(Right)** "An archaeologist laying out two complete core samples (in 1m sections) ready for photographic recording." [6]

This anecdote should serve as a lesson on the powerful impact palaeoclimatological research can have on our perception of the world, and consequently justify its continued importance as a field of study. **A significant portion of the evidence that has swayed popular opinion has come from analysis of *climate proxies***, which are objects containing preserved physical characteristics of past climates [3]. Proxies can be preserved organisms, such as tree rings, corals and fossilised pollen, or environmental samples, such as ice cores and, most relevantly to this project, ocean sediment cores (shown in Figure 2.2). The sea floor is formed in layers by sinking organic matter and river sediment, forming a chronological record of the ocean's composition dating back millions of years. It is possible to accurately date specific layers in a core, **facilitating the reconstruction of the global climate's history from comparisons of similarly dated samples from around the world** [2].

Foraminifera (forams) are ocean-dwelling protists which can be *planktonic*, living floating in water, or *benthic*, living in or on the sea floor. They typically measure from "several milimeters to a few tens of microns" in length [1] and, most importantly for climate research, grow a protective external shell (test) composed of calcium carbonate. **Their fossilised shells are one of the most revealing**

climate proxies scientists search for in ocean sediment cores, since "species diversity, the relative numbers of planktonic and benthic species, the ratios of different shell types, and shell chemistry" are all analysed for ancient climate inference [69]. For example, warmer conditions cause organisms to proliferate, insinuating temperatures were warmer in an era from which dated cores exhibit high denities of foram shells. Additionally, since certain species prefer different conditions (see Figure 2.3), comparison of the distributions of these species can be illuminating [3]. Foraminifera have been used in this way to "map past distributions of the tropics, locate ancient shorelines, and track global ocean temperature changes during the ice ages" [69].

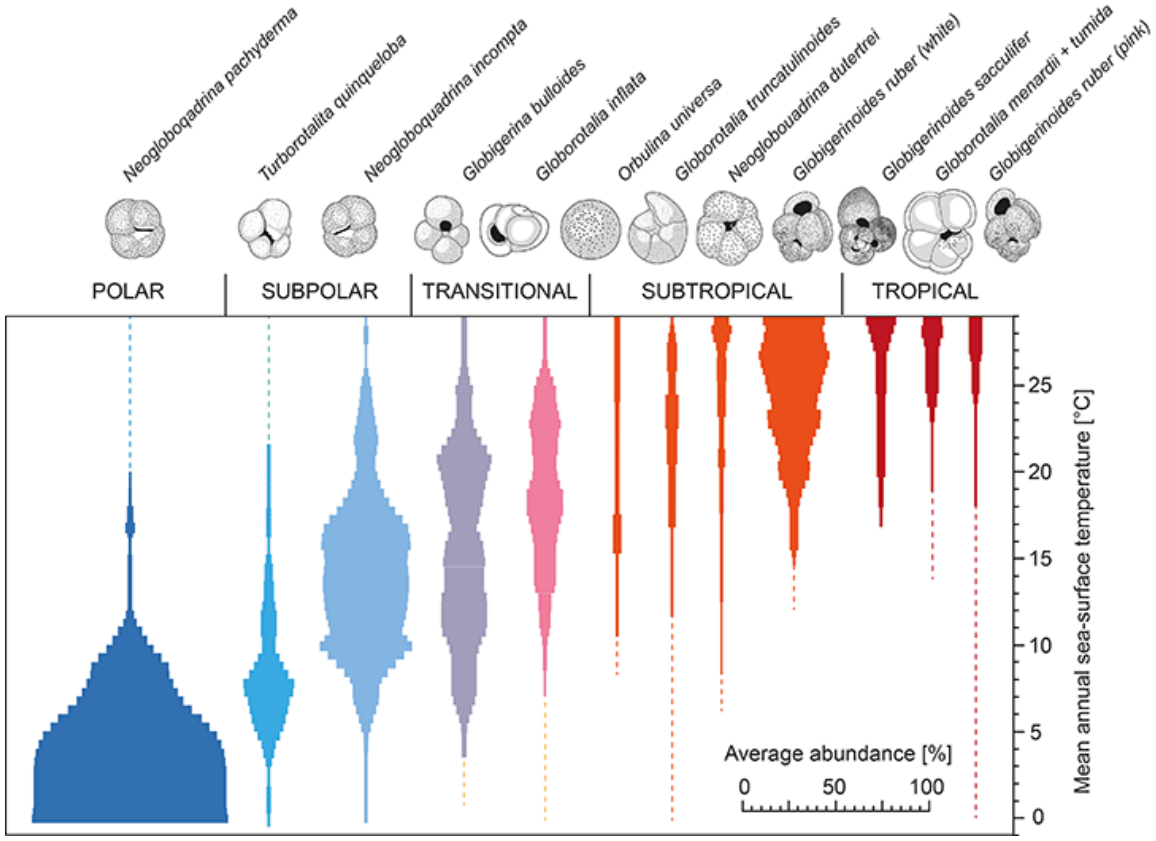

Figure 2.3: **"Abundance plots of selected modern planktic foraminiferal species in relation to sea surface temperature (SST).** The plots are based on data from surface sediments of the Atlantic Ocean (Kucera et al., 2005) and reflect the strong relationship between species abundance and SST." [59]

Traditionally, species classification of fossilised foram tests has been achieved visually by expert micropalaeontologists who assess samples one at a time via light microscopy. Classifications are decided by painstakingly cross referencing each sample with examples of each species found in taxonomic guides, such as the sketches and photographs of Dentigloborotalia anfracta shown in figure 2.4. While these guides help, they still require a wealth of hard-earned experience to apply successfully, and can often leave a lot of room for ambiguity. Unfortunately, forams exhibit notable variability in their morphology, exacerbating the issue of ambiguity and often making it hard for scientists to reach a classification consensus. Consequently, taxonomic agreement is achieved for only 75% of samples [4] This manual process is also repetitive and labour intensive, eating up precious time that micropalaeontologists could instead spend modelling population dynamics and inferring elements of the palaeocimate.

Research into the automation of foram classification from single light microscopy images has blossomed in recent years, motivated by the resource drain of manual sample classification. These works investigate training classification models via a supervised learning approach on foram images labelled with their agreed taxonomic class. To make a useful real-world tool a model must *generalise* well beyond the data it has seen during training, and achieving this with a supervised learning approach generally requires training on a large and varied dataset. Since their inception [34], convolutional neural networks (CNNs) have continually raised the bar for image classification problems, so it is no surprise that the majority of

proposed foram classification models are based on this architecture.

Early work in this field [46, 56] was limited by the quality of the data available, as the datasets used contained few species, limited image variability and were overall lacking in number of samples. In response to this progress bottleneck the *Endless Forams* dataset [38] was compiled, consisting of 34,640 images, each labelled with one of 35 classes. In the same paper detailing the creation of the dataset, Hsiang et al. presented results from training a *VGG16* CNN on the dataset, achieving a new benchmark validation accuracy of 87.41%. This model was trained to produces probabilistic class predictions via optimisation with respect to a softmax loss function (see section §3.2.1).

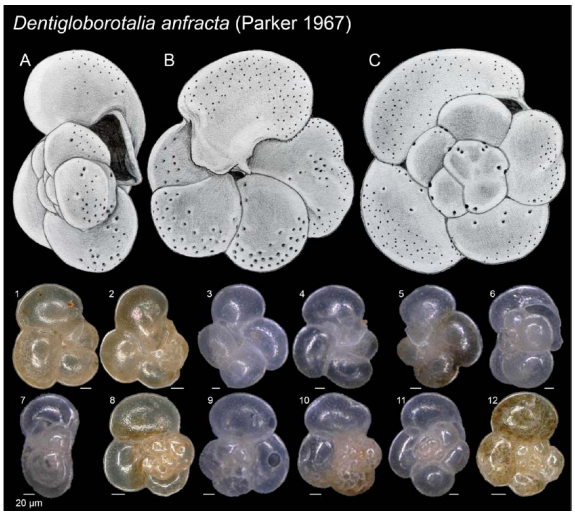

Figure 2.4: "Drawing of **A)** lateral, **B)** umbilical, and **C)** spiral view of a representative mature specimen of D. anfracta modelled on SEM images of a specimen 0.12 mm in diameter ... **1–12)** Light microscope images of specimens from plankton tows."

Karaderi et al. furthered the cutting edge of automatic planktonic foraminifera classification with their proposal of optimising a pretrained ResNet50 (see §3.1.3) image classifier with respect to a hybrid reciprocal triplet and Softmax loss function [32]. First of all, ResNet50 is a 50 layer deep CNN with added residual connections. Residual connections allow the model to contain a great number of layers without suffering the drop in training accuracy experienced by exceptionally deep non-residual models. Due to the vast number of weights that need to be optimised in a ResNet50 model, it is often advantageous to import a instance of the network which has been pretrained on the ImageNet database. Pretrained models have already learned to identify the image features necessary to perform generic image classification tasks, so can vastly accelerate the early stages of training.

Karaderi et al. sought to leverage contrastive gradients in their implementation via the introduction of *metric learning*, which "encourages augmentations (views) of the same input to have more similar representations compared to augmentations of different inputs" [58], allowing them to differentiate between morphologically similar species with finer granularity than could be achieved via purely non-contrastive learning methods. In this context, metric learning works by projecting the training images into a high dimensional latent space, before clustering them by species. Withheld test images can then be classified via a K-nearest neighbours classification of their emedding in the latent space, an approach which achieved an field-leading test set accuracy of 92%.

Another strength of models trained via metric learning is their ability to cluster entire classes not seen during training. This form of *zero-shot learning* facilitated *open set classification* of the 10 smallest Endless Forams classes. By training on the 25 largest classes then projecting the 10 smallest classes into the latent space, the model was able to predict membership to one of the withheld classes with 66.5% accuracy. Moreover, a model which performs well in zero-shot learning can expect some carryover to scenarios where there are only a few examples for certain classes, performing what is known as few-shot learning. The three least numerable Endless Foram classes contain fewer than 10 samples, so the classification of these images could certainly be deemed a few-shot learning problem. This perhaps explains how the model achieved such a high test set accuracy even though it was trained on a dataset containing so few examples of certain classes.

Metric learning is an example of *self-supervised learning*, which in essence allows models to construct their own training signals, as opposed to the supervised learning approach of simply approximating a single definitive label for each datapoint [7]. This approach affords models the freedom to learn from the relationships between datapoints, in turn allowing them to garner more information from smaller datasets. In metric learning the distance between positive and negative embedding pairs constitutes the relationship used to form learning signals. Intuitively, it makes sense that a model learning via a

comparative metric would outperform a model learning simply from labels, since the number of possible comparisons between datapoints will always far outnumber the number of labels.

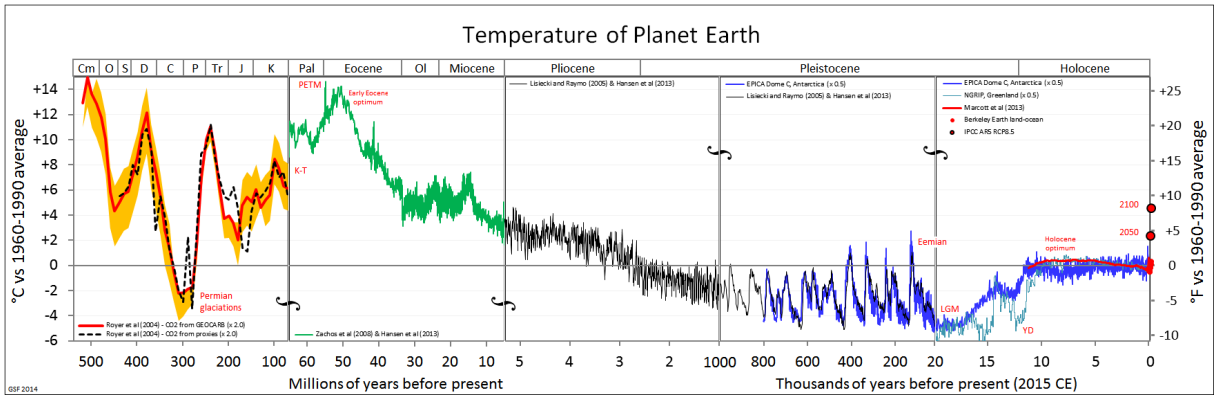

Figure 2.5: **This graph exemplifies the palaeoclimate estimates which have swayed public opinion on climate change. (Panel 1)** 540 to 65 million years ago (mya): "data is from stable oxygen isotope measurements from the shells of macroscopic marine organisms." **(Panel 2)** 65 to 5.3 mya **and (Panel 3)** 5.3 to 1 mya: Data is predicted via "global collection of oxygen isotope data from microscopic marine organisms". **(Panel 4)** 1 million to 20,000 years ago: "Temperature estimates from stable hydrogen isotope measurements from the EPICA Dome C ice core from central Antarctica" **(Panel 5)** 20,000 years ago to present (2015): Temperature estimates from oxygen isotope measurements on the north Greenland ice core, amongst other sources. [54]

Semi-supervised learning is another machine learning paradigm which allows models to learn richer interpretations from smaller training datasets. This type of learning is utilised in image synthesis models, which "build a rich prior over natural image statistics that can be leveraged by classifiers to improve predictions on datasets for which few labels exist" [49]. It is certainly the case that few labels exit for several Endless Forams classes, hence motivating an investigation into training a classifier model via an image-synthesis-driven semi-supervised learning approach. The auxiliary classifier generative adversarial network (ACGAN) proposed by Odena et al. is the perfect architecture for this use case. For each class in the training dataset, ACGAN learns an image distribution, via unsupervised representation learning [31], which approximates the distribution of real images belonging to that class. Once trained, the approximate distribution can be sampled to produce synthetic images which are visually representative of their target class. ACGAN is concurrently trained in a supervised manner to predict the class labels of a stream of interleaved real and synthetic images, with the synthetic images effectively providing "an up-sampling of under-represented classes" [47]. This marriage of unsupervised representation and supervised learning methods is what classifies this approach as semi-supervised.

As explained in detail in section 3.1.6, ACGANs consist of two opposing models: a generator and a discriminator, denoted G and D respectively. In this thesis, I propose training an ACGAN on the Endless Forams dataset, after which D can be employed as a species classifier model. Furthermore, I propose using a pre-trained instance of the ResNet50 based architecture developed by Karaderi et al. as the ACGAN's discriminator. Their model represents the cutting edge in planktonic foraminifera taxonomic classification, so any accuracy improvement granted by the addition of ACGAN's semi-supervised learning methods will boost this newly proposed model to cutting edge status.

Chapter 3

Technical Background

This chapter of the report provides a detailed explanation of the prerequisite technical concepts underlying this project. All novel contributions, which are described in more detail in the project execution chapter, combine together and build upon the architectures and algorithms laid out here. I will begin this technical overview by defining the characteristics of a number of foundational neural network architectures at a high level, before diving deeper into the practicalities of training these models.

3.1 Feedforward Artificial Neural Networks

Feedforward Artificial Neural Networks (ANNs) consist of layers of *units/neurons*, with neurons in each layer connected to the neurons in neighbouring layers by *weighted, directed connections* [42]. Each neuron receives a set of floating point input signals from the outputs of a subset of the neurons in the preceding layer. The output of the neuron, also known as its *activation*, is calculated as a weighted sum over its inputs and an additional *bias* term. The activation is then passed through some nonlinear activation function to produce its output, which is then transmitted to some subset of the neurons in the succeeding layer. In this way, information passes through the network in order from layer to layer, following an algorithm known as *forward propagation*. The calculation performed by a single neuron with n inputs can be summarised by the following equation:

$$y = f\left(b + \sum_{i=1}^{n} w_i x_i\right) \tag{3.1}$$

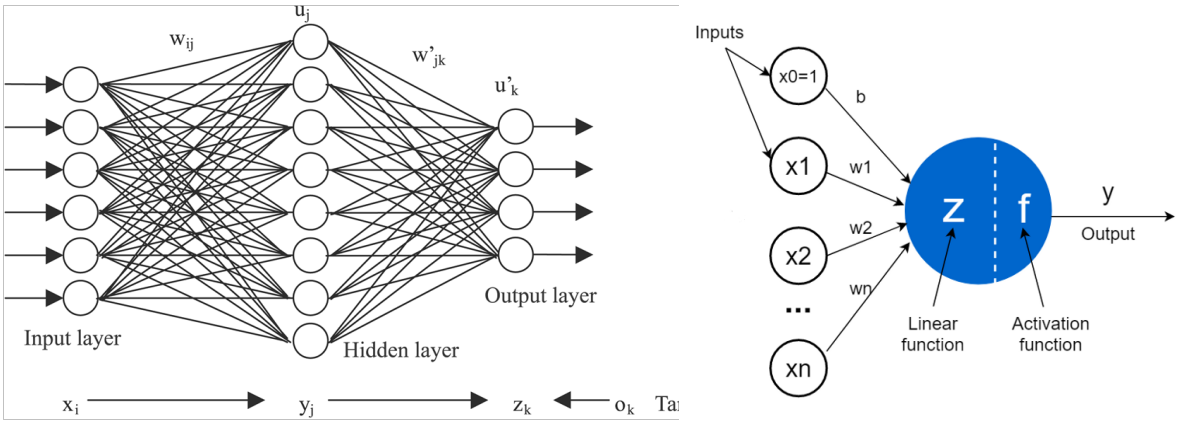

Figure 3.1: **(Left) A simple feedforward artificial neural network architecture** with one hidden layer [63]. x_i, y_j and z_k denote the outputs of the i, j and kth neurons in the input, hidden and output layers respectively and w_{ij} denotes the weight on the connection between the ith node in the input layer and the jth node in the hidden layer. **(Right)** A closer view of **the arithmetic happening in a single artificial neuron** [51], with Z representing the linear weighted sum formalised in equation 3.1.

The first layer in the network, known as the *input layer*, is fed a numerical representation of some data as input. The raw data can take many forms, including images, videos, audio files and natural language.

In the case of images, we can uniquely represent a single 224x224 pixel rgb image as a 3x224x224 matrix of integer values in the range 0-255, with each entry representing the brightness of one colour channel of a specific pixel. Similar unique numerical representations can be formed from any of the aforementioned types of data.

Once the input signal has been forward propagated through the entire network a set of output signals will be produced by the final or *output* layer. The structure of this output layer can vary widely depending on the input type and use-case of the network, with applications ranging from image classification, to image segmentation to natural language processing. *Image classification* tasks are a popular application of ANNs, in which the network estimates the probability that an image belongs to each of a number of classes. A simple example is classification of images into either 'dog' or 'cat' classes, and a more complex example is ImageNet with 1000 classes to decide between.

Any layers situated between the input and output layers are known as *hidden layers*, arising from the *black box* nature of ANNs. They are black box in the sense that we only see their input and output values, while all calculations that occur at the *hidden layers* in between are traditionally left as a mystery. Networks of this description form the largest class of neural networks, with all subsequently described neural network architectures fitting within this broad definition.

3.1.1 Fully Connected Deep Neural Networks

A fully connected deep neural network [30] has two defining features beyond those of a generic ANN. Firstly, to be considered *deep* a neural network must possess more than one hidden layer. Secondly, in a *fully connected* neural network each neuron receives an input signal from every neuron in the preceding layer and subsequently transmits an activation signal to every neuron in the succeeding layer.

Fully connected networks are powerful tools that allow for the approximation of complex non-linear functions. However, being fully connected has its drawbacks in terms of computational complexity and memory requirements. The number of weights between two neighbouring fully connected layers is quadratically related to the width of the layers, meaning that the total number of connection weights in a fully connected deep neural network can easily lie in the order of millions. Since an image input inherently requires wide layers with many neurons, the computation demands limit the potential of fully connected deep neural networks as a tool for computer vision.

3.1.2 Convolutional Neural Networks

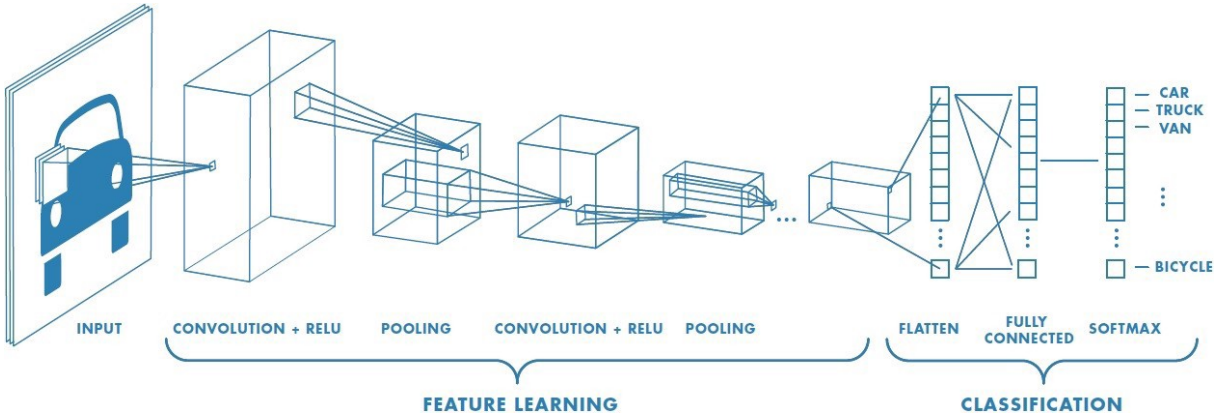

Figure 3.2: **A standard CNN architecture** designed for image classification [10].

Convolutional Neural Networks (CNNs) [35] overcome the computational limitations of fully connected deep neural networks by significantly reducing the number of weighted connections between each layer. In a *convolutional layer*, instead of receiving inputs from all neurons in the preceding layer, each neuron only receives inputs from within its *receptive field*. A neuron's receptive field is a predefined set of neurons in the previous layer, commonly being a 5x5 grid.

To illustrate this vast difference with an example, consider a network designed for processing 3x224x224 images; a single neuron in the second layer of a fully connected network will take $3 \times 224 \times 224 = 150,528$ values as inputs. Contrasted with a CNN with 5x5 receptive fields, in which a neuron in the second layer will take $5 \times 5 = 25$ values as input, it is clear to see the vast disparity in number of network parameters in each layer. Since there are fewer parameters per layer in a CNN of equal width, it allows for the design of much deeper networks capable of extracting much richer features from the input images. This potential has been confirmed empirically, with 152 layer deep ResNet-152 "taking 1st place on the ILSVRC 2015 classification task".

Having said all this, individual fully connected layers remain a powerful function approximation tool and can often be found in the final hidden layer of CNNs, where their job is to produce a classification from the features extracted by the convolutional layers.

Max Pooling

Max pooling is a *downsampling* operation which samples pools of values from an input tensor to create a smaller tensor with a similar distribution of values [70]. For each entry in the output tensor, max pooling sets the entry equal to the largest entry within a certain patch of the input tensor. Pooling layers are often applied in a CNN after a convolutional layer to reduce the dimensionality of the produced feature map. This further reduces the number of parameters to optimise, and adds a small element of translation invariance, meaning applying a small translation to the input image will no affect its classification.

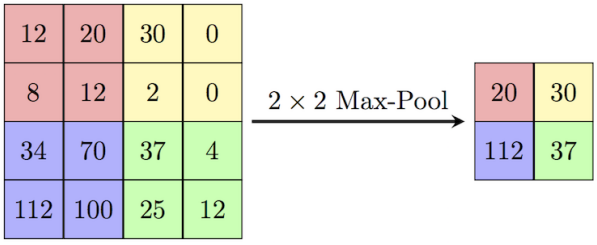

Figure 3.3: Downsampling a 4x4 matrix to a 2x2 matrix via **2x2 max pooling** [61].

3.1.3 ResNet-50

ResNet-50 is a specific CNN implementation with 50 layers [23]. It takes 224x224 resolution RGB images as input and has 1000 output nodes, facilitating image classification into 1000 distinct classes. It is possible to download the weights from a ResNet-50 model which has been *pretrained* on the ImageNet database. Beginning training with a pretrained model allows you to perform *transfer learning*, which can drastically speed up the early stages of learning since the network weights are already trained to identify basic image features, such as edges and corners.

3.1.4 Generative Adversarial Networks

Generative Adversarial Networks [21] are a class of image synthesis architectures which consist of a pair of competing ANNs which together learn to generate new images which are similar to an underlying dataset. The first network is known as the *generator*, G, and takes a random noise vector, z, as input and forward propagates it to produce a synthesised image. Increasing the relatively low dimensions of the input vector to the high dimensions of an output image is achieved by repeatedly upsampling by a factor of 2. The second network, known as the *discriminator*, D, takes an image as input and returns a single value as output representing the estimated probability that the input image is sampled from the dataset of real images.

Traditionally, during training G synthesises fake images which are then fed to D alongside an equal number of real images sampled from the dataset. The discriminator is tasked with predicting which of these images are real and which are fake. Since we know the ground truth of which were truly real and fake it is possible to identify which of the images were misclassified by D, and then use these mistakes to optimise the accuracy of the network. In this way D's classification performance will improve over time.

Clearly the performance of D is inversely proportional to G's ability to synthesise realistic looking images; the better G gets at producing fake images which are plausibly-real, the worse D will perform at correctly classifying them as fake. Therefore, if we optimise G to minimise the accuracy of D, G's ability to produce realistic images will improve over time. By training D to increase its accuracy and G to

decrease D's accuracy the two networks are pitted in direct competition with each other in what is known as a *zero-sum game*, where one model's gain is the other's loss. The idea is that, if the two models can remain close to *Nash equilibrium*, that is neither model's performance grows to dominate the other network, then they will force each other to improve over time. Eventually the generator will be synthesising images which as so convincing that they are indistinguishable from the original dataset to the human eye.

If we think of the original dataset as a set of samples taken from some distribution lying within the high-dimensional space of all possible images, then D estimates the probability that an image is a true sample from that distribution, and G approximates the underlying distribution from which the dataset is sampled. In other words, the generator learns a mapping from a latent space to a target distribution.

As we have discussed, CNNs allow for deeper networks which can perform richer image processing. Since GANs are fundamentally image processing networks it makes sense to design both G and D as CNNs, in which case the GAN is known as a *deep convolutional GAN*, or *DCGAN* [53]. It is no surprise that the images generated by DCGANs qualitatively outperform those generated by GANs when trained on the same dataset.

3.1.5 Conditional Generative Adversarial Networks

Conditional GANs (cGANs) [45] were designed to address one large shortcoming of the basic GAN architecture, which is now known as a *vanilla GAN*. Vanilla GANs have no mechanism for synthesising images from a specific class of a partitioned dataset; they can only synthesise images from the dataset as whole, ignoring any nuances of each individual class. For example, consider again a dataset of cat and dog images. If a vanilla GAN is trained on this dataset then G would learn to synthesise some plausible cat and dog images, but it would also synthesise some hybrid images from the space in between, which would appear as some blend of a cat and a dog. These images would successfully deceive D into believing they were sampled from the original cats and dogs dataset, but would fail to deceive a human, who is aware that the image should fall into one of the two distinct classes - appearing as either a cat or dog. cGAN solves this problem by conditioning G and D on class labels during training, facilitating the synthesis of images belonging to a predetermined class. Returning to our example, if we declare that the class which we wish to synthesise is, say, 'dog', then G will generate a unambiguously classifiable image of a dog.

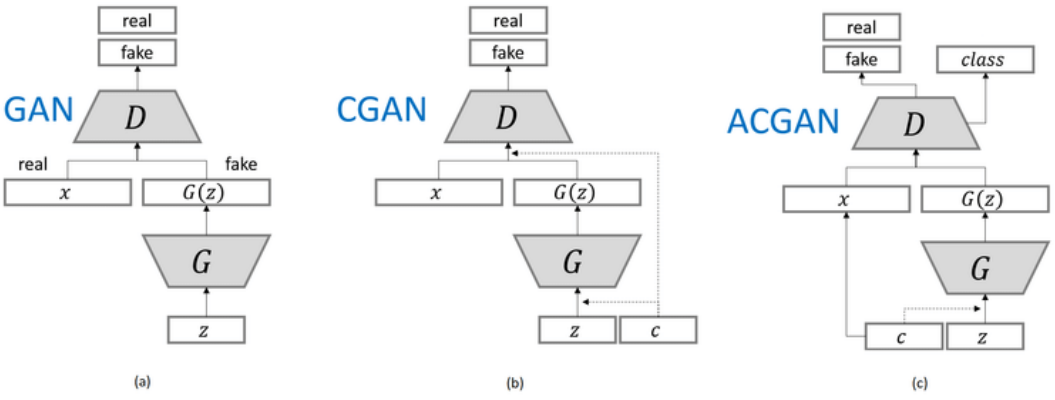

Figure 3.4: **A side-by-side comparison between GAN, CGAN and ACGAN architectures** where G and D denote the generator and discriminator respectively, x denotes a real image, c denotes a class label and z denotes a noise vector [44]. Note, the dotted arrows coming from c represent its use as a secondary, conditioning input.

3.1.6 Auxiliary Classifier Generative Adversarial Networks

In a cGAN the objective of D remains to be the accurate prediction of whether images are real or fake. *Auxiliary classifier GANs (ACGANs)* [49] take the concept of class conditioning one step further by training the discriminator as a classifier. In this architecture G receives a noise vector conditioned on a

class label and outputs a synthesised image from that class, just like in a cGAN. However, unlike cGANs, D's inputs remain unconditioned, just like in a vanilla GAN. The novel contribution of ACGAN is the addition of a secondary/auxiliary classification output to D. Now, for each image input D returns two predictions:

- The *adversarial prediction* is the estimate probability that the input image is a sample from the real image dataset.
- The *auxiliary prediction* is a set of estimates, with cardinality equal to the number of distinct classes in the dataset, such that each estimate represents the probability that the input image was either sampled from, or synthesised conditioned on a label from, the estimate's respective class.

We know the ground truth of whether each input image was real or fake, and also the ground truth of which class the image was sampled from or conditioned on, so it is possible to identify which of D's classifications were incorrect. We can use these identified mistakes to create a learning signal for D, optimising it over time to maximise both correct adversarial and auxiliary predictions. G can then be trained to minimise D's correct adversarial predictions while maximising D's correct auxiliary predictions. Over time, the images G generates will approximate the images in that class in the real image dataset. After training, G can synthesise plausible images belonging to a predetermined class, while D can accurately classify an input image into it class.

3.2 Loss Functions

So far the *optimisation* of the *performance* of these models has only been described at the highest level; the question of how a model's performance can be quantified and subsequently optimised remains. Quantifying a model's performance begins with the definition of a *loss function*, which can vary widely in design, but will work to determine a single numerical value for use as a performance metric. Optimising the performance of the model is then usually performed by some maximisation/minimisation of the loss function's output via alterations to the model's parameters.

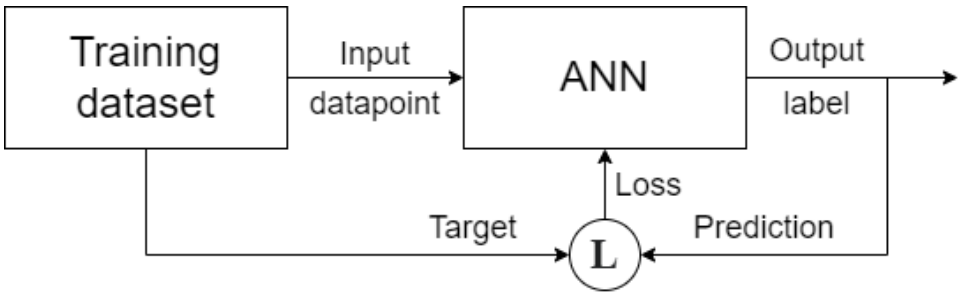

Figure 3.5: **A flow diagram depicting the supervised learning pipeline** used to train an ANN to perform a classification task, where L denotes the loss function.

3.2.1 Supervised Learning

Supervised learning is a broad class of machine learning techniques, defined by the use of a *labelled training dataset* to infer some function, in the hopes that the function generalises to unseen data. A labelled training dataset is a set of pairs, each pair consisting of a datapoint with a corresponding class label. For example, in the dogs vs cats image classification task a randomly selected pair may consist of an image of a dog with the class label 'dog'. Training a model in a supervised manner begins with sampling a pair from the training dataset. The datapoint is presented to the model as input, from which the model produces a prediction for the datapoint's corresponding class label. Learning can then take place by comparing this prediction to the true label, before updating the model's parameters so that the same misclassification is less likely to occur in future.

The choice of loss function depends on the type of output, and also the goals of the algorithm. As I will explain later it is also possible to formulate new loss functions as some average of other loss functions.

Binary Cross Entropy Loss

Binary cross entropy (BCE) loss provides a classification performance metric when performing supervised learning on a dataset with two distinct classes. For example, BCE loss is perfect for quantifying the performance of the discriminator in a vanilla GAN, where the input datapoints each belong to either the 'real' class or the 'fake' class, and all true class labels are known. This function works by comparing the following two inputs:

1. The output prediction of the model, representing the probability that the datapoint belongs to the first class.
2. The *target* prediction for the datapoint, which is determined entirely by the datapoint's true label: 1 if the datapoint belongs to the first class and 0 if the datapoint belongs to the second class.

The model's BCE loss for the datapoint is then calculated using the following equation:

$$L_{\text{BCE}}(y, p) = -(y \cdot \log(p) + (1 - y) \cdot \log(1 - p)) \tag{3.2}$$

Where $y \in \{0, 1\}$ denotes the class label and $p \in [0, 1]$ denotes the predicted probability that the datapoint belongs to class 1. Minimising this loss is equivalent to maximising classification accuracy, and as a result is the goal when optimising a binary classifier. Extending the calculation of BCE loss to a set of labelled datapoints is simply a question of averaging the BCE loss for each datapoint:

$$L_{\text{BCE}}(\boldsymbol{y}, \boldsymbol{p}) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)) \tag{3.3}$$

Where N denotes the number of datapoints in the set, $y_i \in \{0, 1\}$ denotes the class label for the ith datapoint and $p_i \in [0, 1]$ denotes the predicted probability that the ith datapoint belongs to class 1. And, indeed, averaging a loss function across a set of datapoints is possible with any loss function.

Softmax Loss

Now, let us consider the case of a classification task between $m > 2$ classes. Calculating the softmax loss of a model on a task of this nature is a two step process:

1. Apply the softmax function [20] to each of the outputs of the model to transform these predictive values into normalised class probabilities (all values non-negative and summing to 1).
2. Apply the cross entropy loss function to the set of class probabilities and respective targets.

The *softmax function* applied in step (1) is defined as follows:

$$\sigma_i(\boldsymbol{z}) = \frac{\exp(z_i)}{\sum_{j=1}^{m}\exp(z_j)} \text{ for } i = 1, \ldots, m \tag{3.4}$$

Where i denotes the index of the class probability being calculated, $\boldsymbol{z}$ denotes the vector containing the m unnormalised predictive values, z_i denotes the ith element of $\boldsymbol{z}$ and m denotes the number of classes in the dataset. We can use the softmax function to define a normalised vector of class probabilities:

$$\boldsymbol{p} = \begin{bmatrix} \sigma_1(\boldsymbol{z}) \\ \sigma_2(\boldsymbol{z}) \\ \vdots \\ \sigma_m(\boldsymbol{z}) \end{bmatrix} \tag{3.5}$$

The cross entropy loss function applied in step (2) takes the following two arguments for any image:

1. $\boldsymbol{p}$, the normalised class probabilities vector predicted for the image.
2. y, is the image's true class target.

The *cross entropy loss function* is then defined as follows:

$$L_{\text{CE}}(\boldsymbol{p}, y) = -\sum_{i=1}^{m} \mathbf{1}(y = i) \log(p_i) \tag{3.6}$$

Where $\mathbf{1}$ is the indicator function, taking value 1 when its argument is true and 0 when its argument is false. As required, this loss function quantifies the performance of a multi-class classification model, and consequently can be minimised to improve the models performance.

3.2.2 Metric Learning

Metric learning, short for 'distance metric learning', is a form of machine learning which works by optimising the embedding of datapoints into a latent space with respect to some measure of distance within the space [55]. The general idea is to update the mapping from dataset to latent space, with the aim of clustering datapoints which belong to the same class. Once this clustering is achieved it is possible to classify unseen examples by embedding them into the latent space, before observing the classes of their closest neighbouring points within the space.

Clearly, to perform this procedure some well defined measure of distance between two points, or *distance metric*, is required. *Euclidean distance* is a commonly used distance metric which, for any two vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ existing within an n-dimensional space, is defined as follows:

$$\|\boldsymbol{p} - \boldsymbol{q}\| = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} \tag{3.7}$$

Having said this, there are number of viable distance metrics, such as City-Block, Cosine, etc.[9] and, in reality, any function which satisfies the following four axioms[12] is a viable distance metric:

1. Non-negativity
2. Identity of indiscernibles
3. Symmetry
4. Subadditivity

Triplet Loss

With a distance metric selected, we must next define a loss function with which we can quantify the quality of, and subsequently optimise, the mapping from the dataset to the latent space. *Triplet loss* [26] samples three datapoints, namely an *anchor* input, *positive* input and *negative* input, from specific classes and assesses the distances between them. The anchor and positive must be sampled from the same class, while the anchor and negative must be sampled from different classes. The triplet loss is then calculated using the following equation:

$$L_{TL}(\mathcal{A}, \mathcal{P}, \mathcal{N}) = \max\left(\|f(\mathcal{A}) - f(\mathcal{P})\|^2 - \|f(\mathcal{A}) - f(\mathcal{N})\|^2 + \alpha, 0\right) \tag{3.8}$$

Where $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{N}$ are the anchor, positive and negative inputs respectively, α is the *margin* parameter and f is the embedding map. The margin is a predetermined hyperparameter which acts as a slack variable, setting the minimum acceptable difference between the positive and negative pairs' distances. Note, the distance metric used here is the *Euclidean squared distance*, which is simply the square of the standard euclidean distance. This metric is used due to its heavier weighting of greater distances, which consequently accelerates clustering. There are two ways in which triplet loss can be minimised:

- Minimisation of the distance from the anchor embedding to the positive embedding.
- Maximisation of the distance from the anchor embedding to the negative embedding.

A combination of these optimisations are undergone until the difference between these two distances is less than or equal to α, at which point the loss is zero and no more learning is undergone. Worth noting is that triplet loss optimisation is a form of *weakly supervised learning*, since the true classes of the datapoints are not used directly in the calculation of the loss. Instead of comparing classifications to a target label directly, triplet loss only asserts that $\mathcal{P}$ belongs to the same class as $\mathcal{A}$, and $\mathcal{N}$ belongs to a different class than $\mathcal{A}$, without actually using these class labels directly in the loss function.

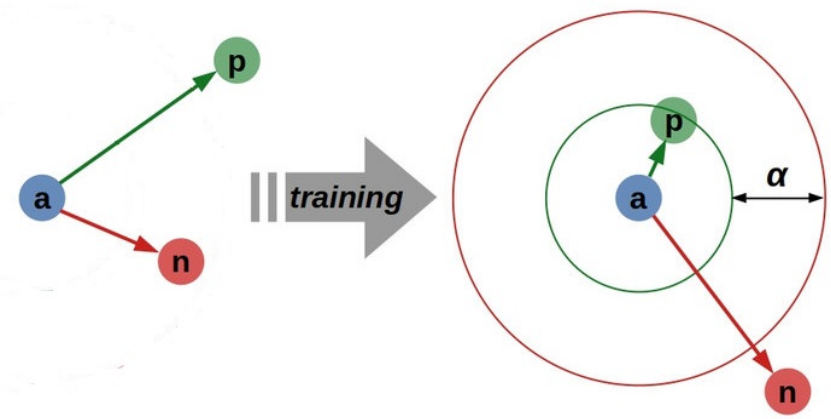

Figure 3.6: This illustration [41] depicts the 2-dimensional embeddings of anchor, positive and negative images, a, p and n respectively, such that a and p are in the same class and a and n are in different classes. Training optimises the embeddings via **triplet loss minimisation**, resulting in clustering of embeddings by class within the latent space. Note, once training is completed n is at least α further from a than p is.

Reciprocal Triplet Loss

One weakness of triplet loss is that the margin hyperparameter is a constant, so is not relative to the distances being measured. This is easiest observed via an example, so let $d_p = \|f(\mathcal{A}) - f(\mathcal{P})\|^2$ and $d_n = \|f(\mathcal{A}) - f(\mathcal{N})\|^2$ denote the distances between the anchor embedding and the positive and negative embeddings respectively. Set α arbitrarily and consider the case when d_p is particularly large. Since d_p is large, α represents a smaller proportion of this distance, and consequently it is only deemed acceptable for d_n to measure relatively marginally larger than d_p. Now considering the case when d_p is small, α is large relative to this distance, so it is acceptable for d_n to be relatively far larger than d_p. This issue of relative lengths plagues triplet loss, causing learning to happen disproportionately quickly at large distances.

Reciprocal triplet loss[40] tackles this issue by omitting the margin hyperparameter altogether. The new reciprocal triplet loss function is formulated as follows:

$$L_{\text{RTL}}(\mathcal{A}, \mathcal{P}, \mathcal{N}) = \|f(\mathcal{A}) - f(\mathcal{P})\|^2 + \frac{1}{\|f(\mathcal{A}) - f(\mathcal{N})\|^2} \tag{3.9}$$

This loss tends to zero as d_p tends to zero and d_n tends to $+\infty$, making the loss entirely agnostic of relative distances. The paper which proposed reciprocal triplet loss reported quantifiably improved classification performance over standard triplet loss, achieving an average auROC of %75.6 compared to standard triplet loss' %57.8.

Classification from a Latent Space

Minimisation of reciprocal triplet loss leads to the clustering of embeddings by class within the latent space. Although the latent space has fewer dimensions than the original data space, they can still have in the order of hundreds of dimensions, prohibiting the visualisation of the clusters which have formed. *t-distributed stochastic neighbor embedding (t-SNE)* [66] provides a window into this high-dimensional latent space by projecting the embeddings down into a two or three dimensional space which can be easily visualised (see Fig 3.7).

With the embedding clusters established from the training set, the model can now be applied to the classification of previously unseen data. This is done by embedding the unseen datapoint within the latent space with the learned mapping, before applying the *K-nearest neighbours (KNN)* [17] classification algorithm to the embedding. The KNN algorithm classifies a test sample by identifying the K training embeddings which are the least distance from the test embedding. The classification of the test sample is then decided by majority vote of the classes of these K nearest embeddings.

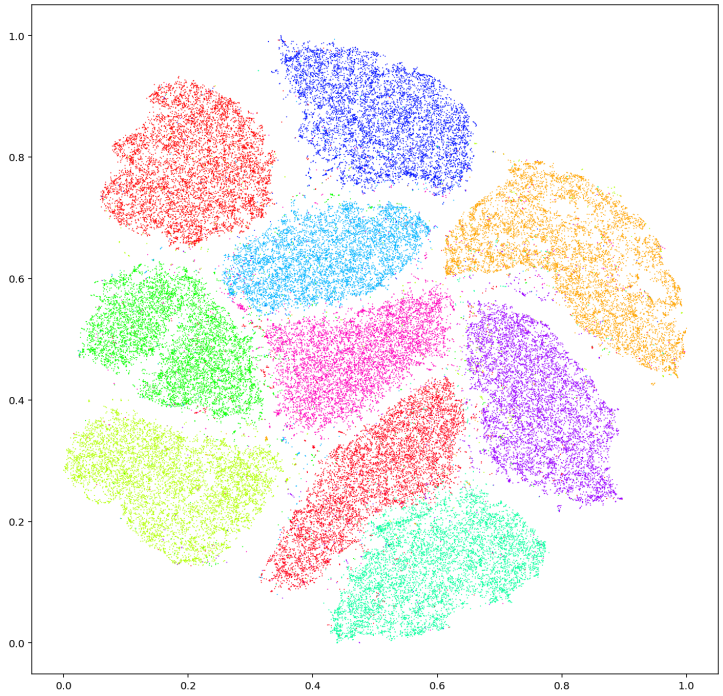

Figure 3.7: A plot of the MNIST dataset projected from a 784-dimensional latent space to a 2-dimensional space by the **t-SNE algorithm** [43].

3.3 Optimisation

Optimisation refers to the maximisation or minimisation of some real valued function, known as the *objective function*, with respect to some sampled inputs. In the context of supervised machine learning, a single optimisation step for a ML model, known as a *training epoch*, can be broken down into the following generic pipeline:

1. Sample a *batch* of training data.
2. Feed the batch to the model, thereafter executing the forward propagation algorithm.
3. Use the model's outputs to calculate the loss of the model for the current batch.
4. Systematically update the model's parameters in order to minimise the loss function.

We characterise this type of optimisation problem as a *minimisation problem* since the loss function, the problem's objective function, is minimised with respect to samples taken from a labelled dataset. This minimisation is executed iteratively, by incrementally updating the model's parameters. Therefore, the optimisation of a model is synonymous with the minimisation of the model's designated loss function via iterative updates to the model's parameters.

3.3.1 Gradient Descent Optimisation

Gradient descent optimisation describes a method for finding *local minima* solutions to minimisation problems by stepping the objective function in the direction of steepest negative gradient [13], where a local minima is a parameterisation of the objective function which is minimal in its neighbourhood but subminimal globally. In the context of ML, gradient descent is a family of algorithms which iteratively estimate the gradient of a model's loss before updating the model's parameters in the most negatively steep direction. *Online gradient descent* calculates the 'true' gradient of a loss function as the derivative of the loss across all datapoints in the training dataset. The online gradient descent update rule is formulated as follows

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \eta_n \nabla L(\boldsymbol{w}_n) \tag{3.10}$$

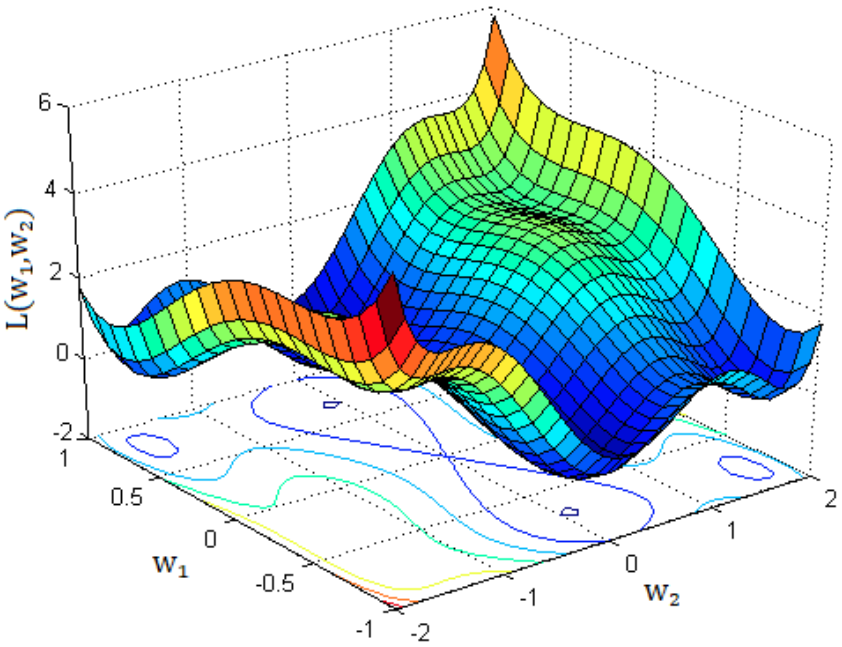

Figure 3.8: **A 3-dimensional *loss landscape***: a plot of a model's loss over its entire training set with different settings of its two parameters, w_1 and w_2 [5]. Gradient descent optimisation aims to attain the values of w_1 and w_2 at the lowest lying point of this landscape.

Where $\boldsymbol{w}_n$ denotes a vector of the model's weights after n optimisation iterations, η_n denotes the *learning rate* for the nth iteration and $\nabla L(\boldsymbol{w}_n)$ denotes the true gradient of the loss function parameterised by $\boldsymbol{w}_n$. The learning rate, η, is a hyperparameter which controls the magnitude of the weight updates; A larger alpha means larger updates. If η is too small it can result in slow learning and getting stuck in local minima, while if η is too large it can make it impossible to find any minima at all. It is therefore common to slowly reduce η over time, accelerating learning at first before slowing to allow the model to settle at a minimum; a method known as *simulated annealing*. Note also that $\eta_n \nabla L(\boldsymbol{w}_n)$ is subtracted from $\boldsymbol{w}_n$, since we wish to update w_n in the direction of steepest *negative* gradient.

Stochastic Gradient Descent

Given that ML methods require large training datasets to achieve generalisation to unseen data, calculating the loss' gradient across all these datapoints is typically infeasible. Therefore, approximating this true gradient by the gradient calculated across a *mini-batch* of randomly sampled training data is usually more computationally and time efficient. This closely related method is known as *stochastic gradient descent (SGD)* [57] , with the term stochastic arising from the stochastic sampling of the mini-batch. Formally, the true gradient is approximated as follows:

$$\nabla L(\boldsymbol{w}_n) \approx \frac{1}{m_n} \sum_{i=1}^{m_n} \nabla L_i(\boldsymbol{w}_n) \tag{3.11}$$

Where m_n denotes the size of the nth mini-batch and $L_i(\cdot)$ denotes the loss function at the ith sample of the mini-batch. SGD parameter updates are then calculated using the using this approximate gradient as follows:

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \frac{\eta_n}{m_n} \sum_{i=1}^{m_n} \nabla L_i(\boldsymbol{w}_n) \tag{3.12}$$

ADAM

Adaptive Moment Estimation (ADAM) [33] is an optimisation algorithm which builds on the general concept of SGD with the addition of adaptive learning rates for each parameter among other features.

The *adaptive gradient (Adagrad)* optimiser [15] was the progenitor of per-weight adaptive learning rates, which are calculated using the sum of the squares of the past gradients with respect to each parameter, allowing infrequently updated parameters to learn at a faster rate. However, the inverse relationship between learning rates and the running sum of gradients means the adapted learning rates eventually become infinitesimally small as the sums grow to infinity, preventing all further learning.

The *root mean squared propagation (RMSProp)* optimiser [64] replaced Adagrad's running sum with an exponentially decaying average of gradients to combat this aggressive learning rate decay. In addition to storing an exponentially decaying average of past squared gradients, ADAM stores an exponentially decaying average of past gradients.

If we imagine the current parameter setting as a ball travelling down a gradient of improved parameter settings, then this additional average of past gradients exerts forces on the ball similar to how *momentum* influences the motion of a ball in real life. These features allow ADAM to overcome some local minima and *saddle points*, giving a model optimised by ADAM a chance of achieving a better performing parameter configuration. The ADAM update equation is formulated as follows:

$$G_{t+1} = \beta_1 G_t + (1 - \beta_1)\nabla L(W_t) \tag{3.13}$$

$$\overline{G} = \frac{G_{t+1}}{1 - \beta_1^t} \tag{3.14}$$

$$A_{t+1} = \beta_2 A_t + (1 - \beta_2)(\nabla L(W_t))^2 \tag{3.15}$$

$$\overline{A} = \frac{A_{t+1}}{1 - \beta_2^t} \tag{3.16}$$

$$W_{t+1} = W_t - \eta \frac{\overline{G}}{\sqrt{\overline{A}} + \epsilon} \tag{3.17}$$

Where β_1 and β_2 are *gradient smoothing* hyperparameters and ϵ is a small hyperparameter which prevents division by zero.

3.3.2 Backpropagation

Backpropagation is an algorithm which calculates the loss function's gradient (e.g. $\nabla L(\boldsymbol{w}_n)$) with respect to the weights of the network via *reverse auto-differentiation* [14]. Intuitively, the algorithm attributes some of the loss to each weight in proportion to the amount that the weight contributed to causing the error. The attributed loss of each weight, known as its *error derivative*, is then used in one of the gradient descent methods described above to update the weight's value.

The error derivative of a weight is another name for the partial derivative of the loss with respect to the weight. Backpropagation, being a dynamic programming algorithm, breaks the problem down into subproblems by calculating each of these partial derivatives using recursive applications of the *chain rule* [28]:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x} \tag{3.18}$$

Backpropagation garners its name from this recursive propagation of the partial derivatives back through the network's layers. More specifically, consider the loss's partial derivative with respect to the weight of the connection between the kth node in layer $N-1$ and the jth node in layer N. This partial derivative can be calculated via the chain rule with only the knowledge of the partial derivatives in the layer above:

$$\frac{\partial L}{\partial w_{jk}^N} = \sum_{i \in C_j^N} \frac{\partial L}{\partial w_{ij}^{N+1}} \frac{\partial w_{ij}^{N+1}}{\partial w_{jk}^N} \tag{3.19}$$

where C_j^N denotes the set of all nodes in layer $N+1$ which have connections with the jth node in the Nth layer. Using this equation it is possible to calculate the partial derivatives of the loss with respect to all weights in the network. To increase the efficiency of these calculations backpropagation calculate

these partial derivatives starting with the final layer of the network then working backwards. This means that when calculating the error derivatives in the Nth layer, such as:

$$\frac{\partial L}{\partial w_{jk}^{N}}$$

the required set of partial derivatives from the succeeding layer:

$$\frac{\partial L}{\partial w_{ij}^{N+1}} \text{ for all } i \in C_j^N$$

are already known. It is then only a question of calculating the set of partial derivatives of the connected weights in layer $N+1$:

$$\frac{\partial w_{ij}^{N+1}}{\partial w_{jk}^{N}}$$

with respect to w_{jk}^{N}, which can be done via regular differentiation. Once error derivatives have been calculated for all weights in the network, a gradient descent step can finally be performed for all weights in the graph:

$$w_{jk}^{N} \longleftarrow w_{jk}^{N} - \alpha \frac{\partial L}{\partial w_{jk}^{N}} \tag{3.20}$$

3.4 Activation functions

Activation functions are real valued, typically nonlinear functions which are applied to the outputs of nodes in ANNs during forward propagation. They take the result of the weighted sum performed by the node as input and output a single *activation* value. Biologists discovered that animal neurons have an innate voltage threshold which their cumulative weighted inputs must surpass before the neuron produces any output voltage. ANN activation functions were initially modelled on these biological *threshold potentials* as a way to control the inputs to the neurons in the following layer.

Today, the *nonlinearity* of activation functions is their most valuable attribute. To see why, suppose we apply a linear activation function between each layer. Both the sum and product of two linear functions is itself linear, meaning that by applying linear functions to the weighted sums of neurons across many layers will result in nothing more than a simple linear relationship between the input and output of the model. Applying many nonlinear functions throughout the forward pass of a network allows the network to learn complex, nonlinear patterns, making the model a more powerful function approximator.

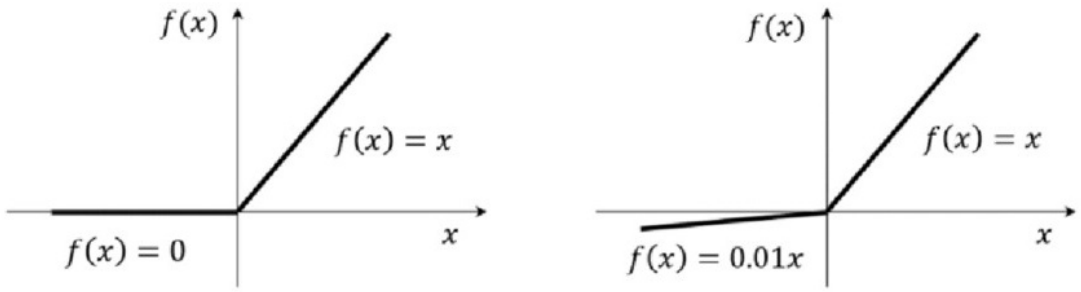

Figure 3.9: **(Left)** $f(x) = \text{ReLU}(x)$, **(Right)** $f(x) = \text{LeakyReLU}(x)$ [37].

3.4.1 ReLU

The *Rectified Linear Unit (ReLU)* activation function [48] is a nonlinear activation function defined as follows:

$$\text{ReLU}(x) = \max\big(0, x\big) \tag{3.21}$$

ReLU's simple first order derivative equation is also worth noting:

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ \text{undefined} & \text{if } x = 0 \end{cases} \tag{3.22}$$

ReLU boasts all the benefits of a nonlinear activation function discussed above, as well being computationally inexpensive to calculate, producing *sparse activation* (∼50% neurons firing) and combatting the *vanishing gradient problem* which plagues sigmoid and tanh activation functions.

3.4.2 Leaky ReLU

Unfortunately, ReLU suffers from the *dying ReLU* problem, in which neurons can learn to be active for essentially no feasible inputs. At this stage the neuron has died, and it becomes almost impossible for them to learn again since no error is backpropagated to inactive neurons. *Leaky ReLU* [39] was developed to tackle this weakness by allowing 1% of the neurons input to pass as output when the output is negative, ensuring some output is produced in all cases. Leaky ReLU and its first order derivative are defined as follows:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{if } x < 0 \end{cases} \tag{3.23}$$

$$\text{LeakyReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0.01 & \text{if } x < 0 \\ \text{undefined} & \text{if} x = 0 \end{cases} \tag{3.24}$$

3.5 Endless Forams Dataset

All experiments conducted during this project involved training and testing models on images from the *endless forams* dataset [29]. The dataset consists of 34,640 416x416 pixel light microscopy images of *planktonic foraminifera*, each expertly hand labelled as one of 35 foraminiferal species, and is partitioned into a training set of 27,737 images and a test set of 6,903 images. Automating species classification on this dataset is an especially difficult challenge for several reasons [32]:

1. There is a high degree of visual similarity between certain species.
2. There are high levels of morphological variability within many species.
3. The perspective from which the forams are viewed is not consistent between images of the same species, meaning important morphological features can be obscured or unrecognisable.

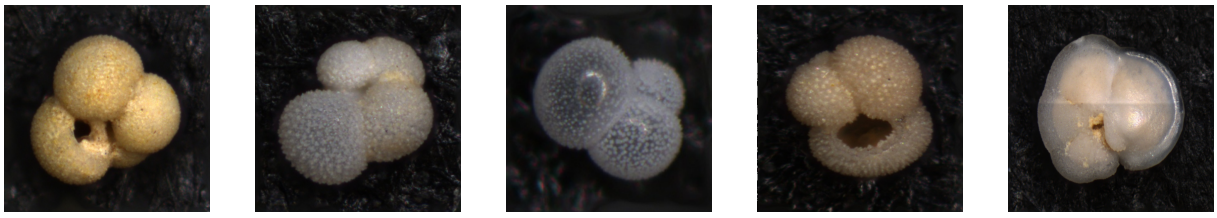

Figure 3.10: **Five images of Foraminifera belonging to different species**, sampled from the Endless Forams dataset [38].

Chapter 4

Project Execution

This chapter lays out the novel MLACGAN architecture as well as detailing the training and testing methodologies. As an alternative but equivalent objective, this section explains every stage of the system overview depicted in Figure 1.1, with each component of the pipeline numbered with the corresponding explanatory section of this chapter.

4.1 Network architecture

MLACGAN's architecture is based primarily on that of ACGAN, with both sharing their composition of an adversarial generator and discriminator pair. However, in terms of the network's architecture, MLACGAN diverges from ACGAN firstly with the replacement of a simple, untrained CNN discriminator with a significantly deeper pre-trained ResNet model, and secondly in its addition of a tertiary *embedding output layer*.

4.1.1 Generator

The MLACGAN generator, G, is identical to the generator implemented in ACGAN, and as such takes the same two inputs:

1. A random noise vector, denoted by z.
2. A conditioning class label.

The size of z is predetermined by a hyperparameter, `noise_dim`, which sets the dimensionality of the noise space from which z is sampled. Each component of z is sampled randomly from a standard Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. The class label is an index corresponding to one of the dataset classes. In the case of this project, the Endless Forams dataset has 35 classes, so class labels are an index in the set $\{0, 1, \ldots, 34\}$. G's forward pass algorithm can be summarised as follows:

1. Produce a label embedding.
2. Element-wise multiplication of noise and label vectors.
3. Deconvolve the conditioned noise.

The first operation in G's forward pass is to map the conditioning label into a latent space, producing a *label embedding* vector. The label latent space has dimensionality equal to `noise_dim`, the same dimensionality as the noise space, facilitating the second step which is their element-wise multiplication. This new, conditioned noise vector is then fed into the *deconvolutional neural network*, where it is upsampled and convolved to produce a 256x256 colour image as output. After training, this output image depicts a plausible, novel example from the conditioning class. The deconvolutional neural network's architecture is depicted in Figure 4.1.

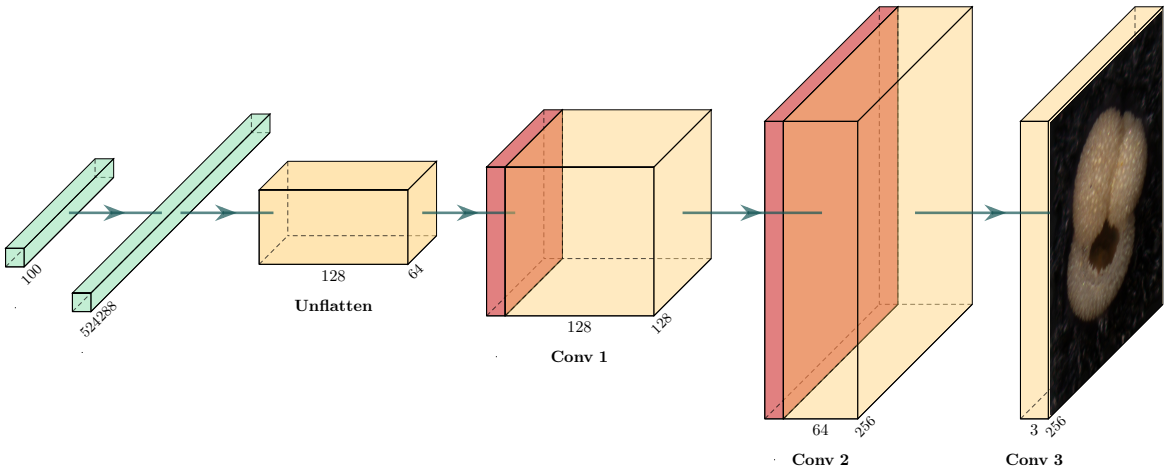

Figure 4.1: **The generator's deconvolutional neural network** takes a batch of 100-element conditioned noise vectors as input. Let B denote batch normalisation, L denote element-wise LeakyReLU function, U denote upsampling with scale factor 2 and C denote a 2D convolution with a 3x3 filter, stride 1 and padding 1. **(1)** A fully connected layer between 100 input features and 524288 output features. **(2)** Unflattening the features into a 128x64x64 tensor. **(3)** Apply B, U then C. **(4)** Apply B, L, U then C. **(5)** Apply B, L, C then a hyperbolic tangent function element-wise.

4.1.2 Discriminator

The MLACGAN discriminator, D is implemented very differently to ACGAN's discriminator. D takes a 224x224 colour image as input, then passes the image through the convolutional layers of a pre-trained ResNet50 model in order to extract deep residual features (DRFs). ResNet50 then flattens the DRFs and passes them through a fully connected layer to produce 1000 class prediction logits. D differs at this point, taking the output of the fully connected layer and passing it through three more, separate fully connected layers to produce three distinct outputs:

1. A single logit, known as the *adversarial prediction*, representing the likelihood that the input image was sampled from the training dataset, as opposed to being synthesised by G.
2. A set of 35 logits, known as the *softmax prediction*, representing the likelihood that the input image was sampled from, or synthesised conditioned on, each of the dataset classes.
3. A 100-element *image embedding*.

The image embedding is a vector representing a point within the 100-dimensional latent space. It is these embeddings that are optimised by metric learning. Given its 50 layers, ResNet50 is too complex to succinctly display in its entirety, so a summary of the modified ResNet50 implemented in D is depicted in Figure 4.2. These three output layers can alternatively be though of as belonging to three separate networks which share weights between their convolutional and residual layers.

4.2 Training

Training MLACGAN is a multi-stage process which invariably proceeds with the following steps:

1. Initialise the core training components:
   - Dataset and dataloader.
   - Generator and discriminator.
   - Adversarial and auxiliary loss.
   - Optimisers for G and D.
2. Sample a batch of triplets from the training dataloader.

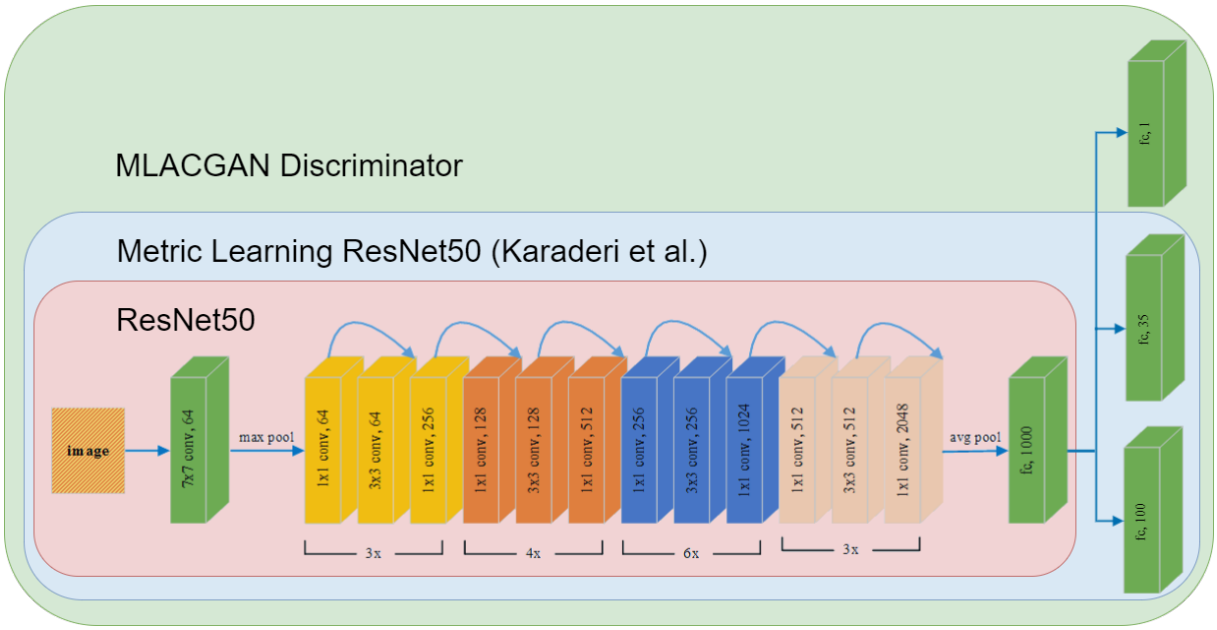

Figure 4.2: **The architecture of the discriminator, D.** The pretrained ResNet50 architecture within D is highlighted in red. The blue arrows above the convolutional layers represent residual connections, and the multiplications underneath denote the repetition of those blocks. The three output layers, from top to bottom, are the adversarial, softmax and embedding outputs. D is the model implemented by Karaderi et al. with the addition of a single adversarial logit.

3. Synthesise a batch of triplets using G.
4. Pass synthesised batch to D to calculate adversarial and softmax predictions and embeddings.
5. Calculate and backpropagate G's loss.
6. Pass both real and fake batches to D to calculate adversarial and softmax predictions and embeddings.
7. Calculate and backpropagate D's loss.

Each of these steps will be explained in detail in this section.

4.2.1 Training Dataset

Introduced in §3.5, the Endless Forams training set consists of 27,737 416x416 pixel images from 35 classes. As shown in Figure 4.3, there is a large degree of class imbalance in the training set. Imbalance leads to a bias towards accurate classification of the largest classes while classes with fewer examples are overlooked, adding an extra level of challenge to classifying the least represented classes. The semi-supervised learning produced by the generative aspect of the model was introduced partly as a method for combatting this class imbalance.

Calculating the network's reciprocal triplet loss requires access to embeddings of triplets of images, and in order for D to output triplets of embeddings we must feed it triplets of images as input. As explained in §3.2.2, these triplets must be constrained so that A and P belong to the same class, while A and N belong to different classes. To satisfy this requirement I wrote a custom dataset which, when sampled, returns a triplet of images in this format, along with the positive and negative class labels. The sampling algorithm works as follows:

- Randomly sample an image A from a randomly selected class C_A.
- Randomly sample an image P from class C_A, such that $P \neq A$.
- Randomly sample an image N from a randomly selected class C_N, such that $C_N \neq C_A$.

At this stage a handful of transformations are applied to each sampled image:

1. A random rotation in range $[-180, 180]$.
2. A random horizontal flip with probability 0.5 .
3. Resizing the image from 416x416 to 224x224 .
4. Convert the image to a tensor.
5. Normalise the values of the tensor.

The random rotation and horizontal flip are *augmentations*, implemented to increase the amount of training data and improve the trained model's generalisation to unseen data. Intuitively the choice of these augmentations makes sense, since the forams have been photographs from arbitrary perspectives, so application of either/both of these transformations will produce a plausible training example. One common transformation which has been omitted is a random affine transformation, which I assumed would hinder the model's ability to generalise to unseen data since all the images in the test set are centered within the image and are scaled roughly the same, both features that a random affine transformation would disrupt. Resizing the images to 224x224 is necessary since D requires input images with these dimensions. Converting the image to a tensor is necessary before input to D. Finally, normalising the tensor forces the tensor's values to lie in the same range as our ReLU activation functions, hence reducing the frequency of non-zero gradients during training, speeding up learning.

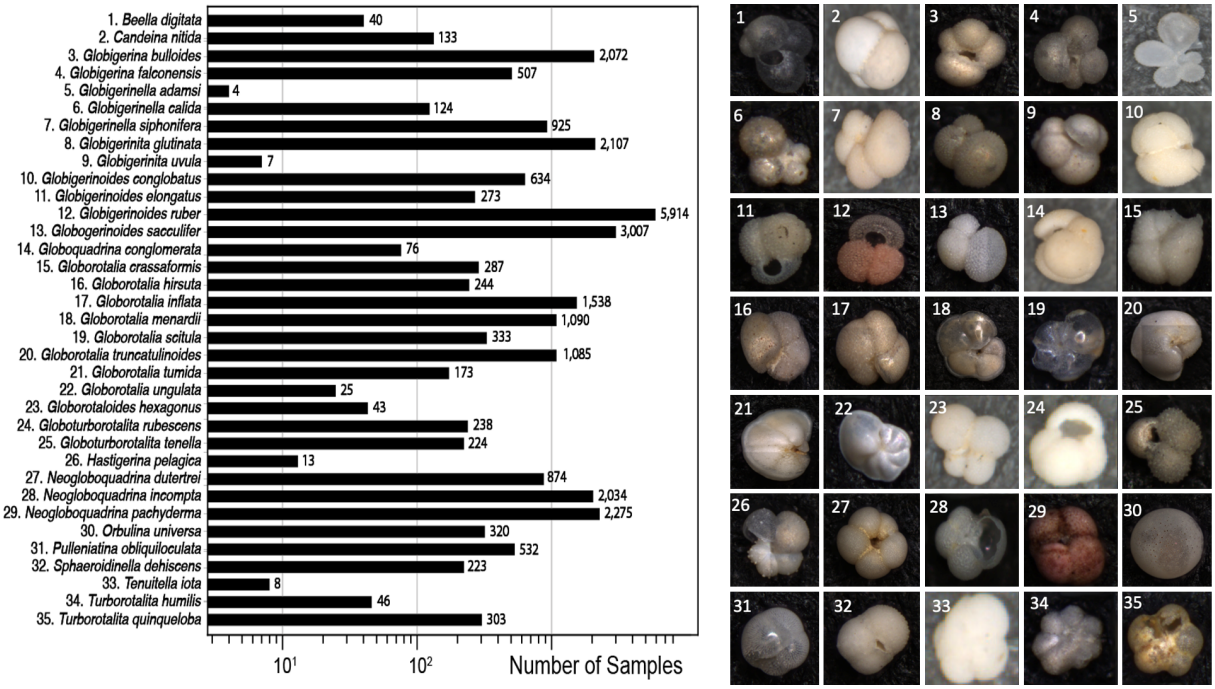

Figure 4.3: **(Left)** A chart with logarithmic scaling depicting **the number of samples in each class of the Endless Forams training set**. **(Right)** An image sampled from each of the 35 Endless Forams classes. Both produced by Karaderi et al. [32].

Finally, the dataset is wrapped in a dataloader object which, when iterated, returns *batches* of triplets. Each batch is a 4-dimensional tensor, with the size of each dimension corresponding to:

1. The number of triplets in the batch.
2. The number of colour channels (RGB).
3. Image height.
4. Image width.

Since the batch is comprised of triplets, as opposed to individual images, it essentially triples the number of images in each batch when compares to a standard dataloader. This meant that, due to memory limitations I was forced to experiment with smaller batch sizes of 32, 16 and 8. Also worth noting is, at the beginning of each *training epoch* a shuffle operation is applied to the ordering of the training dataset, meaning the anchors from which the triplets are generated are ordered differently in each epoch, which in turn reduces the chance of learning entering a detrimental cycle.

4.2.2 Initialise Generator and Discriminator

When training from scratch I initialised G with weights sampled from a normal distribution. To be specific, each weight in each convolutional layer was sampled from the zero-mean Normal distribution with standard deviation 0.02, $\mathcal{N}(0, 0.02^2)$. For each batch normalisation layer, each weight is sampled from the Normal distribution with mean 1 and standard deviation 0.02, $\mathcal{N}(0, 0.02^2)$, while each bias is set to 0.

Notably, there are a couple of options for initialising D. Firstly, the ResNet50 component of D (see Fig 4.2) can be initialised with weights imported from a ResNet50 model which has been pre-trained as an ImageNet classifier. Pre-training the ResNet50 component allows it to act as a DRF extractor, so D is initialised ready to learn classifications from these features. The outcome of this is greatly accelerated early stages of learning. Alternatively, the Metric Learning ResNet50 component of D (again see Fig 4.2) can be initialised with weights from an identical model which has been pre-trained as a classifier on the Endless Forams dataset. This method of pre-training allows an untrained D to perform at a test set classification accuracy of $> 92\%$.

At regular intervals during training, the current weights of both G and D are saved as a *model state dictionary*, which is then pickled (compressed) and saved to a file. The saved state dictionary acts as a model checkpoint, so that training can then be resumed later by unpickling (uncompressing) and loading the weights back into their respective networks.

4.2.3 Loss Functions

It is crucial to note that G and D have different loss functions, denoted L_G and L_D respectively, each of which I will refer to as *aggregate loss* functions since they are formulated as a weighted sum of several other losses. At the highest level they are both averages of an *adversarial loss* and an *auxiliary loss*:

$$\mathrm{L}_G = (1-\gamma)\mathrm{L}_{Aux} - \gamma\mathrm{L}_{Ad} \tag{4.1}$$

$$\mathrm{L}_D = (1-\gamma)\mathrm{L}_{Aux} + \gamma\mathrm{L}_{Ad} \tag{4.2}$$

Where $\gamma \in [0, 0.5]$ is a hyperparameter determining the relative contributions of L_{Aux} and L_{Ad} towards the aggregate losses for G and D (see §4.2.4).

First of all, the value of L_{Ad} is determined by D's ability to classify input images with respect to their source: either sampled from the real image dataset, or synthesised by G. More accurate image source classification means a smaller value of L_{Ad}, meaning in order to minimise L_D D must classify the source as accurately as possible. Note, however, that in order to minimise L_G G must maximise L_{Ad} by reducing the source classification accuracy of D. So, in order to 'fool' D into misclassifying a greater number of image sources, G is optimised to synthesise images which, for D, are indistinguishable from the real images in the training dataset. Formally, the adversarial loss is defined as the BCE loss (see §3.2.1) of D's adversarial source prediction:

$$\mathrm{L}_{Ad} = -(y \cdot \log(p) + (1-y) \cdot \log(1-p)) \tag{4.3}$$

Where p denotes D's estimated probability that the data was sampled from the training dataset, and y denotes the true source label, where $y = 0$ if the data is synthesised and $y = 1$ if sampled from the training dataset. Minimising this BCE loss is equivalent to maximising the log-likelihood of correct classification:

$$\mathbb{E}\left[\log P\big(S(X) = \text{real} \big| X \in \text{real}\big)\right] + \mathbb{E}\left[\log P\big(S(X) = \text{fake} \big| X \in \text{fake}\big)\right] \tag{4.4}$$

Where X denotes the training data, $S(X)$ denotes the true source of X and $P(S(X)|X)$ is D's probabilistic, adversarial source prediction. Note, this is equivalent to the binary cross entropy loss introduced in §3.2.1. L_{Aux} contributes the remainder of the aggregate losses, but is itself a form of aggregate loss, consisting of a softmax loss and a reciprocal triplet loss:

$$L_{Aux} = L_{SoftMax} + \lambda L_{RTL} \tag{4.5}$$

Where λ is a constant hyperparameter which weights the contribution of the reciprocal triplet loss. As introduced in §3.2.1, SoftMax loss is calculated from the 35 softmax prediction logits as follows:

$$L_{SoftMax} = -\log\left(\frac{\exp(z_{\text{class}})}{\sum_{j=0}^{34}\exp(z_j)}\right) \tag{4.6}$$

Where z_j denotes D's unnormalised softmax prediction logit for the jth class and z_{class} denotes the logit corresponding to its prediction regarding the image's true class. Intuitively, the value of $L_{SoftMax}$ is determined by D's ability to accurately determine the class of a given image. Since a portion of the input images D receives are synthesised by G, $L_{SoftMax}$ is also indirectly determined and by G's ability to synthesise images which can be distinguished as belonging to the class it was fed as its conditioning input. Minimisation of L_G and L_D requires the minimisation of this SoftMax loss component, meaning G will be optimised to synthesise convincing representations of individual foram species, while D learns to accurately classify images by species.

The reciprocal triplet loss component is also defined similarly to in §3.2.2, but now with the addition of 'batch hard' triplet mining [24], in which the loss is calculated with respect to the 'hardest' negative:

$$L_{RTL} = \|f(\mathcal{A}) - f(\mathcal{P})\|^2 + \frac{1}{\|f(\mathcal{A}) - f(\mathcal{N})\|^2} \tag{4.7}$$

Where $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{N}$ are the anchor, positive and negative image embeddings returned by D, such that $\mathcal{N}$ is the furthest embedding from $\mathcal{A}$ in the minibatch.

Again, minimising both L_G and L_D requires the minimisation of L_{RTL}, which is done by minimising the euclidean squared distance between $\mathcal{A}$ and $\mathcal{P}$, while simultaneously maximising the distance between $\mathcal{A}$ and $\mathcal{N}$. This results in the embeddings clustering by class within the latent space. Optimising the network via minimisation of RTL should therefore effect G in a similar way to minimising the SoftMax loss, in so much that G will learn to synthesise images which produce optimal behaviour when passed to D as input. In the context of RTL this means that G will learn to synthesise triplets of images for which the anchor and positive embeddings will be spatially close, while the distance between the anchor and negative embeddings will be large. In doing so, the synthesised images from each class will visually resemble real images sampled from the same class.

Importantly, L_G and L_D are calculated with respect to differing sets of input image batches. Specifically, L_G is calculated from D's outputs given a batch of purely synthesised triplets. On the other hand, L_D is calculated from D's outputs given a mixed batch of images, formed as the union of the same synthesised batch with a batch of real images sampled from the training dataset. The ratio of synthesised to sampled images in the mixed batch is set by the r hyperparameter (see §4.2.4). In order to reflect this varying ratio in the calculation of L_D, I weighted the contributions of losses from synthesised and sampled images with γ, the normalised construction of r. L_D can hence be reformulated to reflect this:

$$L_D = \gamma \cdot L_D^{fake} + (1-\gamma) \cdot L_D^{real} \tag{4.8}$$

Where L_D^{fake} and L_D^{real} denote L_D calculated only with respect to the synthesised and sampled images within the mixed batch respectively, while γ denotes the normalised r. Hence L_D can be expressed in terms of adversarial and auxiliary losses with respect to synthesised and sampled data:

$$L_D = \gamma(\gamma L_{Ad}^{fake} + (1-\gamma)L_{Aux}^{fake}) + (1-\gamma)(\gamma L_{Ad}^{real} + (1-\gamma)L_{Aux}^{real}) \tag{4.9}$$

4.2.4 Batch Synthesis

In order to have G synthesise triplets, with each triplet consisting of $\mathcal{A}$ and $\mathcal{P}$ from the same class and $\mathcal{N}$ from some other class, it is necessary to feed G triplets of conditioning labels in this format. Selection of a conditioning label triplet is done via the following steps:

1. The class label for $\mathcal{A}$ and $\mathbb{P}$ is selected at random as one of the 35 classes.
2. The class label for $\mathcal{N}$ is selected at random from one of the remaining 34 classes.

Also required is a triplet of random noise vectors, which we can sample from a zero-mean Normal distribution with standard deviation 1. These three noise vectors are passed to G along with the three conditioning class labels, at which point the noise is conditioned with the labels before being fed to the deconvolutional network.

r

It turns out, the size of the synthesised batches is an influential factor for the training of the model. In is customary to train an ACGAN with an equal mixture of sampled and synthesised images, meaning the synthesised batch size is usually set equal to the sampled batch size. When implementing a pre-trained model as the discriminator in an ACGAN, training on an equal split of sampled and synthesised data can cause the auxiliary classification to experience a harsh drop in its classification performance on unseen test images. This is because G is essentially producing random noise in these early stages of training, while D is being trained to classify this noise into its labelled class. Since the noise produced by G has no resemblance of the examples from that class, D is faced with an impossible task, and its learning is badly affected as a consequence.

My proposal to combat this weakness is to phase in the synthesised images over the course of many epochs via gradual incrementation of the r hyperparameter. r takes values in the range $[0, 1]$ and determines the size of the synthesised batch in relation to the size of the sampled batch. For example, given a sampled batch size of 64 and a r of 0.5, the synthesised batch will contain 32 triplets. The r is initially set to zero, before being incremented linearly over the course of a predetermined number of epochs, known as the number of *phase epochs*. Since the product of the r and batch size is not guaranteed to be an integer, the synthesised batch size is calculated by rounding the result of this product to the nearest integer.

In addition to this, r is used to calculate the value of γ, which itself weights the various subcomponents of the aggregated losses (see §4.2.3). I experimented with a few different formulae for calculating γ:

$$\gamma_1 = \frac{r}{1 - r} \tag{4.10}$$

$$\gamma_2 = \frac{r}{2} \tag{4.11}$$

$$\gamma_3 = \frac{r^2}{2} \tag{4.12}$$

These effects are best understood graphically so refer to Figure 4.4 for a visualisation:

These phasing effects are also well illustrated by a tabular example. The following table shows the values of r and the synthesised batch size, given a sampled batch size of 64, during a training regimen with 5 phase epochs and 2 subsequent fully blended epochs:

Epoch	r	γ_1	γ_2	γ_3	Sampled Batch Size	Synth. Batch Size
1	0	0	0	0	64	0
2	0.2	0.167	0.1	0.02	64	13
3	0.4	0.286	0.2	0.08	64	26
4	0.6	0.375	0.3	0.18	64	38
5	0.8	0.444	0.4	0.32	64	51
6	1	0.5	0.5	0.5	64	64
7	1	0.5	0.5	0.5	64	64

Interpolation

Recall that as a result of their respective architectures G synthesises 256x256 pixel images, while D requires 224x224 pixel images as input. There are a number ways in which the synthesised images can be transformed to the required 224x224 resolution, for example applying a central crop. Unfortunately,

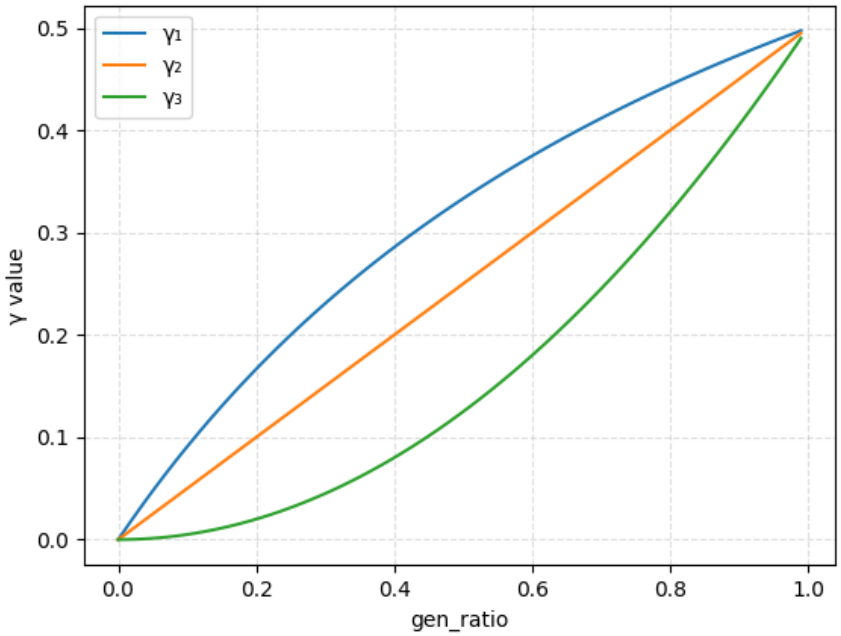

Figure 4.4: **A plot showing the relationship between r and the various formulations of γ.** γ_1 exhibits faster phasing in the early stages of training then slows down, γ_2 phases linearly and γ_3 phases more slowly at first then speeds up later in training.

many of these options will cause important elements of the image to be lost, making classification by D significantly harder. Moreover, if the transformation changes the image in some obvious way, such as the zooming effect a central crop would produce, then D will have too easy of a time distinguishing the synthesised and sampled images. This could slow down learning, and even have a detrimental impact on the images G synthesises, since G will now be trying to synthesise images that look plausible *after* the transformation has been applied. In the case of the central crop, G will learn to synthesise images which are more zoomed out than the images in the training set, since they will look similar *after* the crop has been applied.

Interpolation is the best option for resizing an image while preserving the information it contains. There are many forms of interpolation, but they all work by setting the colours of the pixels in the new image as some function of the pixels in their neighbourhood in the old image. In the end I opted for a *bicubic interpolation* which acts as a function of a neighbourhood of 16 pixels, producing a significantly smoother interpolation than 2D nearest neighbour and bilinear methods which only consider a neighbourhood of 4 pixels (see Figure 4.5). The calculation from of a larger neighbourhood is much more computationally expensive than alternative interpolation methods, which inevitably slows down learning. However, this cost is worth it since any loss of quality during the interpolation will reduce the upper limit of the image quality G can learn to reproduce.

4.2.5 Optimisers

As explained in §3.3.1, ADAM is a parameter optimisation algorithm which provides superior performance thanks to is implementation of per-parameter adaptive learning rates. I used separate ADAM optimisers for the optimisation of the weights of both G and D. ADAM has three hyperparameters which need setting before training:

1. Learning rate - parameter step size.
2. β_1 - exponential decay rate for the first moment estimates
3. β_2 - exponential decay rate for the second moment estimates

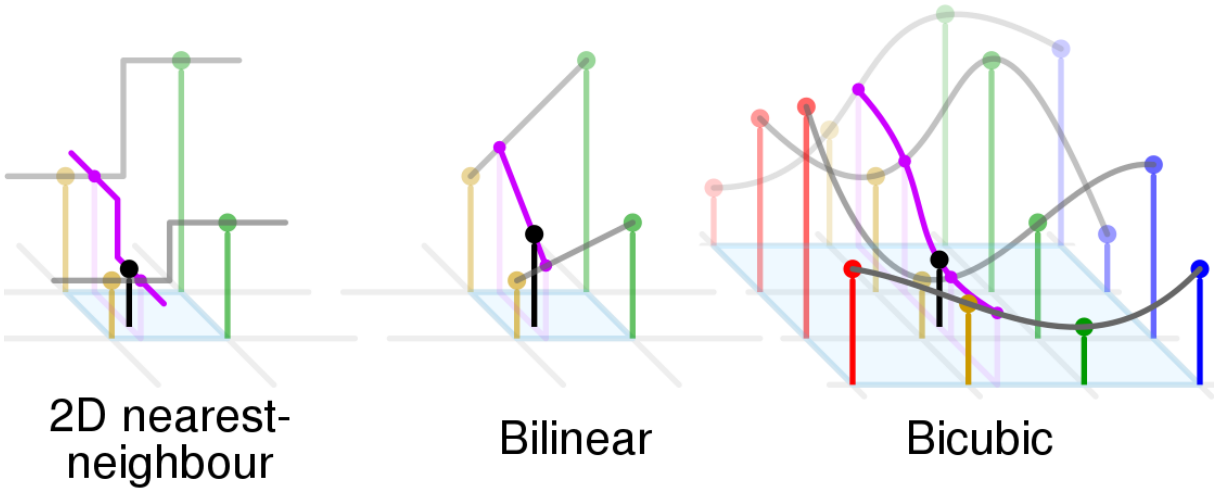

Figure 4.5: **A visual comparison of 2D nearest-neighbour, bilinear and bicubic interpolation methods.** "The black dots correspond to the point being interpolated, and the red, yellow, green and blue dots correspond to the neighbouring samples. Their heights above the ground correspond to their values." Courtesy of CMG Lee on Wikipedia Commons [36].

4. ϵ - prevents division by zero

In my implementation I set $\beta_1 = 0.5$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$ for all experiments. I chose this high value of β_2 to maximally preserve gradients, since image processing is typically a task which involves sparse gradients. ϵ was set to a tiny number to avoid division by zero while affecting gradient calculation as little as possible. For all experiments initialised with an untrained generator I set the learning rate to 0.0002, which is moderate but still slightly above the default 0.0001 to allow for faster learning in the early stages. I also experimented with smaller learning rates when learning with a pre-trained model to facilitate more precise optimisation of the model's parameters.

4.3 Evaluation

MLACGAN learns to satisfy two main objectives:

1. Synthesis of morphologically accurate images of each of the 35 foram species present in the Endless Forams dataset.
2. Species classification of light microscopy foram photographs.

4.3.1 Qualitative Generator Evaluation

The network's success in these two objectives must be evaluated independently to assess its overall performance. Objective (1) must be evaluated *qualitatively* by visually assessing the degree to which images synthesised of each species correlate with real images of the same species. A set of N synthesised images can be produced for each class by loading a trained generator, then feeding it N pairs of random noise with the target species label. This can be repeated for each of the 35 classes to acquire N synthesised images of each of the 35 species. The following questions investigate the visual attributes which contribute to the assessed quality of the synthesised image:

- Does the synthesised foram test posses the number of chambers characteristic of that species?
- Is the test's chamber arrangement characteristic of that species?
- Is the test's aperture located characteristically of that species?
- Is the test textured and coloured realistically?
- Do the synthesised images fully cover the morphological variability seen in each species?

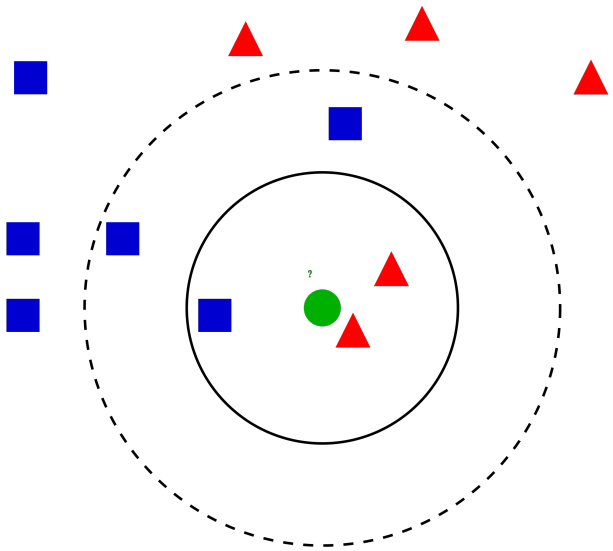

Figure 4.6: **A visual representation of K-Nearest Neighbours classification algorithm** in a 2-dimensional feature space. The blue squares and red triangles are datapoints from two distinct classes, while the green circle is the datapoint we are hoping to classify. The two radii show how the classification can change depending on the setting of K. The solid black line is the boundary defined when $K = 3$, in this scenario the classification will be a red triangle. The dotted line is the boundary when $K = 5$, here the classification is a blue square. Figure produced by Tavish on AnalyticsVidhya.com [62].

These assessments can be carried out by a non-expert cross-referencing the synthesised images with real photographed examples. However, for the most reliable qualitative assessment it is best to seek a foraminiferologist's opinion on the answers to these questions, as their years of experience can assist their perception of minute details that may be overlooked by a non-expert.

4.3.2 Quantitative Discriminator Evaluation

In contrast, objective (2) must be evaluated *quantitatively* via an array of numerical performance metrics. This involves loading a trained discriminator, then having it produce embeddings for the entirety of the training set, as well as for a disjoint testing set which was unseen during training. I then construct a 128-dimensional latent space containing all the training set embeddings along with their class labels. The test set images are then classified by placing their embeddings in the latent space, before estimating their class labels via a k-nearest neighbors classifier. The k-nearest neighbours (KNN) algorithm simply works by taking a majority vote on the labels of k closest datapoints. Specifically, the class of a test image is estimated by taking a majority vote on the classes of the k training points with embeddings closest to the embedding of the test image in question. For all evaluations I used $k = 5$. The performance of this classification can then be evaluated via accuracy metrics, such as accuracy, precision, recall and F1 score.

The accuracy produced by this method is known as the *test accuracy* and certainly provides a good surface-level insight into the performance the model as a classifier. However, to gain a deeper insight it is also customary to evaluate the model's *train accuracy*. Contrasting the test and train accuracies can provide valuable insights. For example, if the train accuracy is increasing but the test accuracy has peaked and began to stagnate or even decrease then this serves an an indication that model is beginning to overfit to the training data. Overfitting describes the scenario when the model has become too specialised at classifying the training data, and consequently fails to generalise to unseen testing data. When the beginnings of overfitting are identified via a comparison of train and test accuracies, further overfitting can be prevented by halting training early.

In order to quantify the model's training accuracy via a similar KNN approach I randomly partitioned the training set into disjoint subsets A and B, containing $\sim$99% and $\sim$1% of the training set respectively. I then using a trained D to embed all of A within a 128-dim latent space, before applying KNN classification to the embeddings of each of the elements of B. I repeated this process 10 times, each time with a new random partitioning of the training set, and averaged the outcomes to acquire a representative training

accuracy.

4.3.3 Test Dataset

The Endless Forams dataset contains a predefined test set with which to evaluate classification models trained on the Endless Forams training set. Provision of a standardised test set facilitates fair comparison of different Endless Forams classification models. The test set consists of 6,903 416x416 pixel colour images, none of which appear in the training set. The train/test split is roughly 80/20, however there is variation between classes. For example, classes with relatively few examples have disproportionately many of these examples in their test sets, which ensures the test set contains enough examples to remain representative of the morphological diversity present within that species.

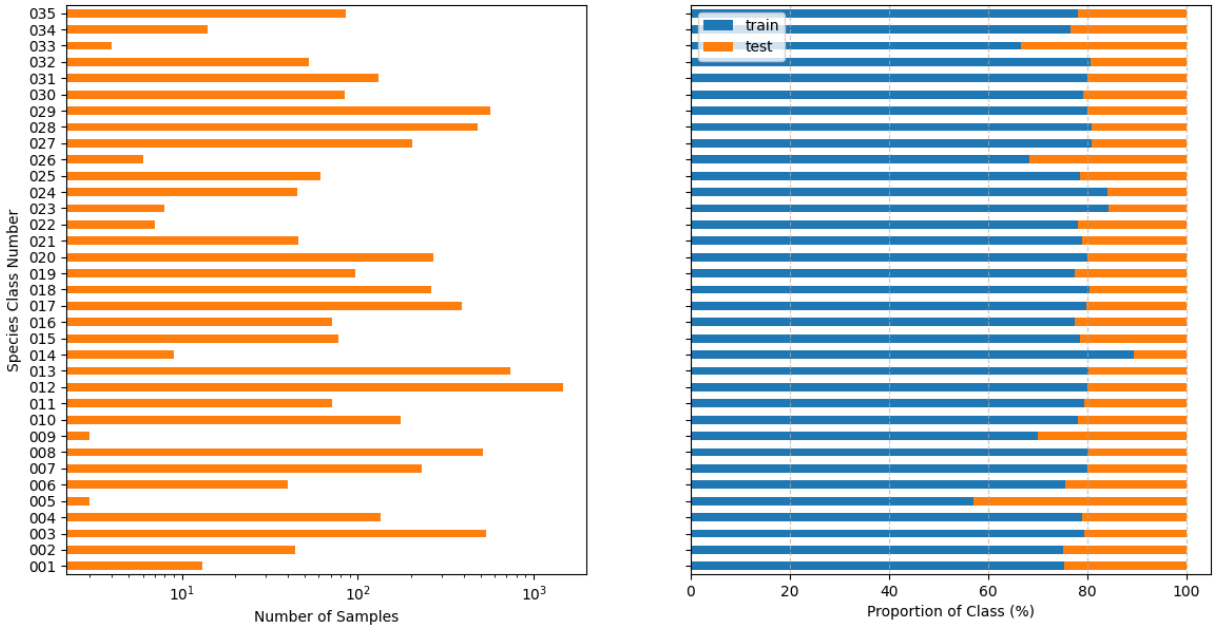

Figure 4.7: **These figures are an analysis of the distributions of images accross the classes of the Endless Forams dataset. (Left)** The number of samples in each class of the Endless Forams test set, plotted on a logarithmic scale. **(Right)** The percentage of each class of the Endless Forams dataset that consists of images in the training set and the test set. Note the correlation between classes with few examples and classes with disproportionately large test sets compared to the size of their training set (see classes 005, 009, 026 and 033).

Chapter 5

Critical Evaluation

In this chapter I provide a through quantitative and qualitative evaluation of the application of MLACGAN to the tasks of foram image classification and synthesis. I first contrast the model's performance when trained on a 50/50 blend of sampled and synthesised images against its performance when the proportion of synthesised images is gradually phased in throughout training. I then propose and evaluate several phasing schemes, which alter details of how the loss and real/fake split are interpolated during training. Lastly, I provide a summary of my quantitative and qualitative findings in the form of statistical metrics and samples synthesised by G.

5.1 Training with Only Real Images: r = 0

First of all, I attempted to reproduce the results of Karaderi et al. using a pre-trained MLACGAN discriminator. To do this I loaded their pre-trained model's weights into the portion of the discriminator consisting of the ResNet50 architecture and the fully connected layers which produce the softmax and RTL outputs, which is highlighted in blue in Figure 4.2. The remaining adversarial output layer was left initialised with untrained weights. Since I was evaluating a model which had been optimally pre-trained on the same dataset, I expected the model's performance to be optimum for the outset. However, it turned out that the accuracy of the pre-trained discriminator was ∼88% before any further training, significantly lower than the 91.6% accuracy recorded by Karaderi et al. It is possible this lower initial accuracy is due to inconsistent versioning of python or any of the libraries used in the implementation.

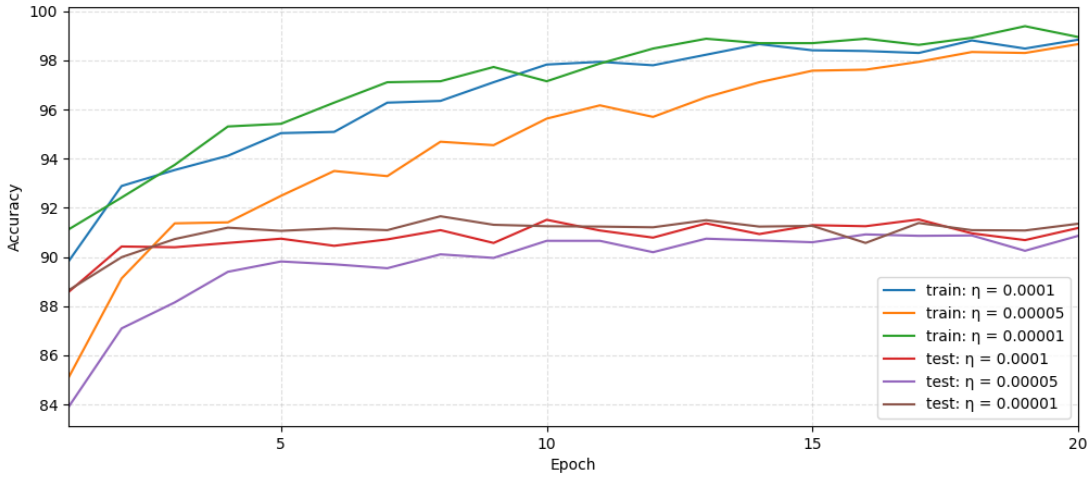

Figure 5.1: **The curves in this graph depict the evolution of the discriminator's classification accuracy on the training and testing sets over the course of 20 training epochs.** The increasing test accuracy is evidence that the model learns information from the training set that generalises to the test set during the first 5-10 epochs. The continued growth of the training accuracy, far beyond that of the test accuracy, is evidence of overfitting setting in. There is no obvious correlation between the learning rate setting and the rate of test accuracy improvement. However, as expected, the greatest test accuracy of 91.66% was achieved with the lowest learning rate of 0.00001.

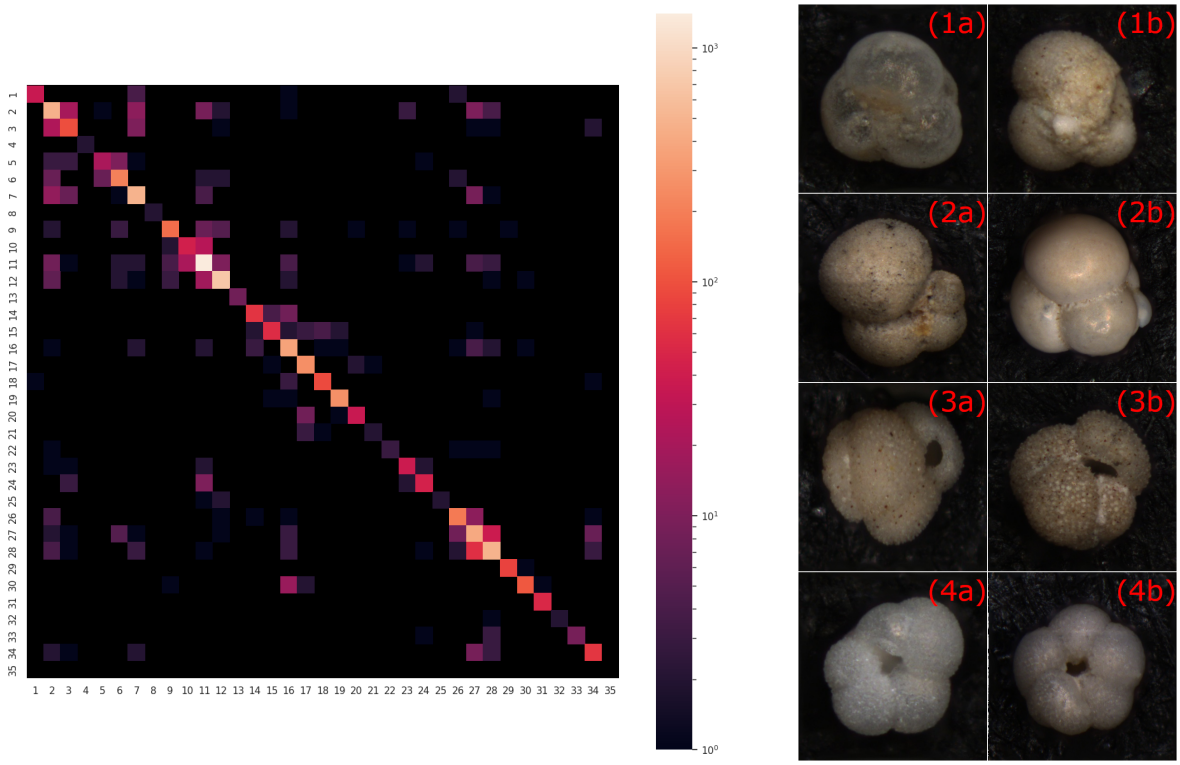

Figure 5.2: **(Left) A confusion matrix visualising, for each pair of Endless Forams classes, the frequencies with which D classified images from the first class as belonging to the second.** Black cells represent zero classifications while lighter cells represent many classifications. The cells on the diagonal represent correct classifications, with all others representing misclassifications. **(Right) Examples of the visually similar forams which were misclassified by D.** These misclassifications are sampled from the easily confused classes identified in the confusion matrix: **(1a,b)** 2 and 3, **(2a,b)** 2 and 7, **(3a,b)** 10 and 11, **(4a,b)** 27 and 28. Note (1b) and (2b) are also visually similar and belong to the easily confused classes, 3 and 7.

In order to recreate the reported results I trained this model for 20 epochs, experimenting with learning rates of 0.0001, 0.00005 and 0.00001 to see if this would have any impact on the rate of improvement or the maximum performance achieved. For this experiment, along with all others conducted in this thesis, I set $\lambda = 0.01$ as recommended by [25]. The r was set to a constant value of 0, meaning each training batch consisted entirely of real sampled images, with no images synthesised at any point. A r of 0 also meant that $\gamma = 0$, regardless of the choice of γ_1, γ_2 or γ_3, so the loss function for D was equivalent to the one implemented by Karaderi et al. :

$$L_D = L_{SoftMax} + \lambda L_{RTL} \tag{5.1}$$

This formulation of the loss function ensures that the lack of pre-training done on the adversarial output layer has no bearing on the subsequent training of the model, since the adversarial loss it produces is given zero weighting in the aggregate loss function. Figure 5.1 depicts the accuracy evolution throughout this training process, and shows that all learning rates lead to a maximum test accuracy of $> 91\%$. The smallest learning rate, $\eta = 0.00001$, led to the greatest maximum accuracy out of the three learning rates, achieving a maximum accuracy of 91.66%, marginally greater than the 91.6% accuracy reported by Karaderi et al. This result makes sense because a smaller learning rate updates the model's parameters with smaller steps, enabling a more precise identification of the optimal parameter settings. I saved a checkpoint of the specific model weights which achieved this most accurate result, enabling later loading as a pre-trained MLACGAN. From now on I will refer to this optimally pre-trained MLACGAN as *model A*.

To remain concise, I will refer to this training scheme of setting $r = 0$ as *RN*, where N denotes the number of training epochs. In this specific case I conducted all experiments with this scheme over 20 epochs, hence they follow an *R20* scheme.

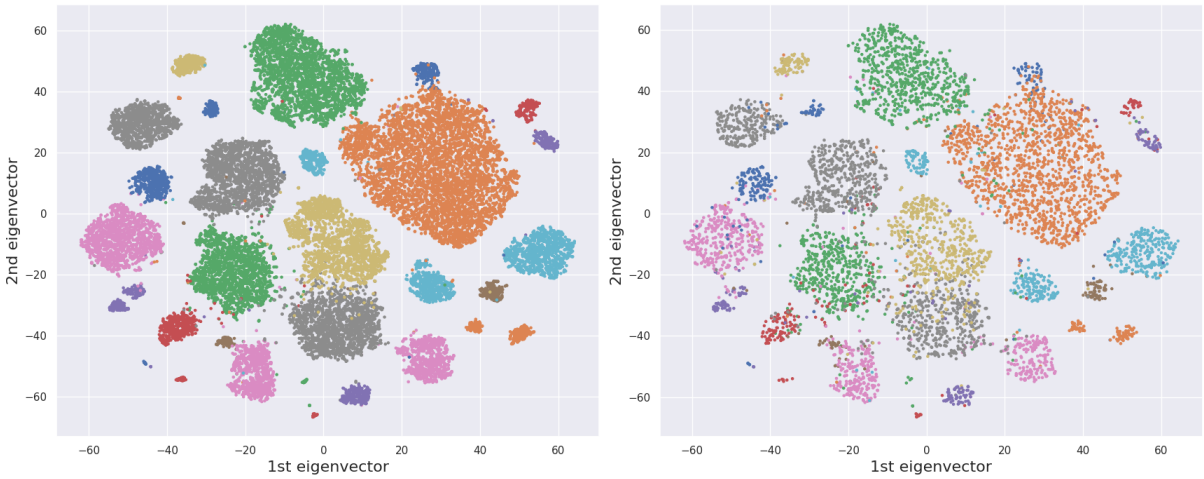

Figure 5.3: **A pair of 2-dimensional t-SNE plots which visualise the structure of the optimised latent space. (Left)** depicts the optimised training set embeddings, an estimated 97.15% of which are clustered with at least 3 out of their 5 nearest neighbours being from the same class. **(Right)** depicts the predicted embeddings of the test set. These plots were calculated by performing principal component analysis on the union of the 128-dimensional train and test embeddings. The vector space spanned by the first 50 orthogonal principal components was then projected down to 2 dimensions by applying the t-SNE algorithm for 1000 iterations.

5.2 Training with Synthesised Images: r = 1

My first experiment training MLACGAN to synthesise foram images was done with the training regimen recommended in the paper proposing ACGAN. Specifically, this meant ensuring every minibatch consisted of exactly half synthesised and half real images, with the losses for both G and D calculated with equal weighting on their adversarial and auxiliary components. The only deviation from ACGAN's loss being the auxiliary loss's reciprocal triplet loss subterm:

$$\mathrm{L}_G = \frac{1}{2}\mathrm{L}_{Aux} - \frac{1}{2}\mathrm{L}_{Ad} \tag{5.2}$$

$$= \frac{1}{2}(\mathrm{L}_{SoftMax} + \lambda \mathrm{L}_{RTL}) - \frac{1}{2}\mathrm{L}_{Ad} \tag{5.3}$$

$$\mathrm{L}_D = \frac{1}{2}\mathrm{L}_{Aux} + \frac{1}{2}\mathrm{L}_{Ad} \tag{5.4}$$

$$= \frac{1}{2}(\mathrm{L}_{SoftMax} + \lambda \mathrm{L}_{RTL}) + \frac{1}{2}\mathrm{L}_{Ad} \tag{5.5}$$

This desired behaviour, both in terms of the ratio of synthesised images in the minibatch and the formulation of the loss functions, emerges when r is set to zero. Therefore, I loaded the weights from pre-trained model A into a new MLACGAN and proceeded to train it with $r = 0$ and $\lambda = 0.01$ for 50 epochs. Figure 5.5 (Left) depicts the evolution of the model's classification accuracy on both the training and test sets. It is clear that by replacing half of the real images D was seeing with their synthesised counterparts has introduced a great deal of instability to the network. The accuracy can be seen wildly fluctuating between ∼80% and ∼40%, at times varying by almost 40% in a single epoch. This instability is expected, since the discriminator is being trained to classify equal portions of real and synthesised images from the outset, at which stage G is completely untrained and hence only synthesises random noise, as shown in Figure 5.5 (Right). D is therefore trying to learn mappings from the visual features of the noisy images to their respective class labels, and in doing so disrupts the optimised mappings from real images to class labels which it had inherited from P. The visual features it is searching for are non-existent, leading to the observed lower maximum, and overall erratic changes in, classification accuracy.

This sudden negative impact on D's ability to classify real images can be seen more clearly in Figure 5.6, where the auxiliary losses for both real and fake images are plotted alongside a line showing the

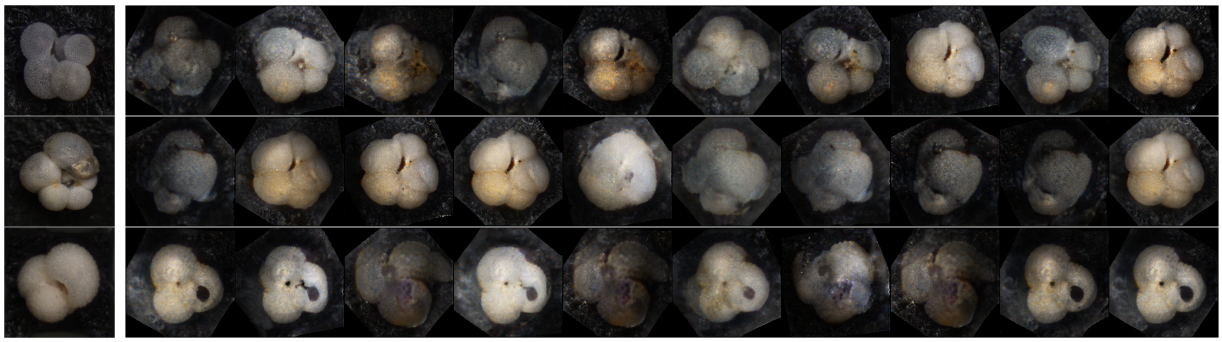

Figure 5.4: **(Left) Light microscopy photographs of forams sampled from the Endless Forams training set.** These images have been expertly labelled as belonging to species (from top to bottom) Globigerina falconensis, Globigerinella siphonifera and Globigerinoides conglobatus, class 4, 7 and 10 of the dataset respectively. **(Right) 10 randomly synthesised images from each of (also from top to bottom) classes 4, 7 and 10.** Note, for each class, the presence of synthesised samples with close visual resemblance to the real photograph sampled from the same class. Also note the repetition of near identical examples within each class, and also between classes, eg. the rightmost example from classes 7 and 10.

auxiliary loss which model A settled to. The auxiliary loss on real images spikes immediately to ∼0.6, while D's ability to classify real images is harmed by the introduction of noisy synthesised images, then steadily recovers to this baseline after 40 epochs. In contrast, the fake image auxiliary loss begins at a significantly greater value of ∼1.75, then falls to its resting loss of ∼0.25 in under one epoch. This much more rapid decrease makes sense, since G and D are working together to bring this value down, while only D is working to bring down the auxiliary loss for real images. In other words, G is being optimised to help D by synthesising images which D can easily classify. Interestingly, the loss from fake images never reaches the same minimum loss produced by real images, implying G has been unable to learn to synthesise images which totally accurately represent the species they are trying to portray. This hypothesis is further supported by the synthesised images shown in Figure 5.4, where it is clear that, while morphologically accurate images have successfully been synthesised, a large portion of the synthesised images do not accurately represent their target species.

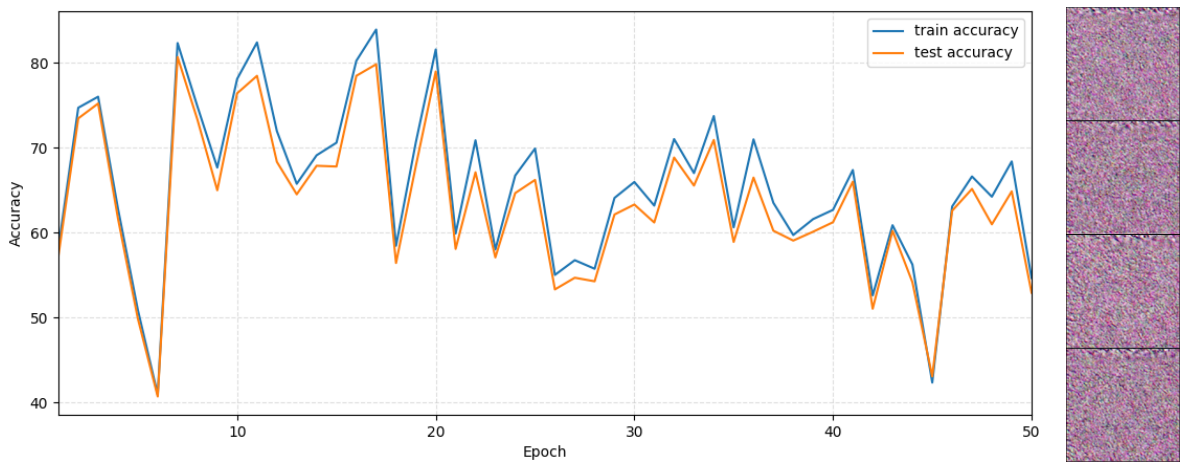

Figure 5.5: **(Left) A plot of MLACGAN train and test accuracy evolution throughout 50 epochs of training with pre-trained weights loaded from model A and r set to 1.** The instability introduced by the synthesised images is evident when this plot is compared to Figure 5.1. Also notable is the significantly lower accuracy achieved than P, falling from 91.66% to a maximum of 80.67%, as well as the continual downward trend in accuracy throughout training. **(Right)** Images synthesised by G, of species 1-4 from top to bottom, after less than one full epoch of training.

Another takeaway point from Figure 5.4 is that *mode collapse* has likely occurred, signified by the repetition of near identical images. Mode collapse is a a common issue faced when training GANs, wherein

the generator learns to map a wide range of noise vectors to the same/similar output images, and is best avoided since it limits the possible outputs of the trained generator. Mode collapse is closely associated with unstable training, which has been observed in Figure 5.5, hence the observation is unsurprising. Figure 5.7 gives further insight into a possible cause for this mode collapse, by revealing the failure of D and G's adversarial loss to converge to an equilibrium. Lack of convergence causes G to collapse to only synthesising the few image modes which may have at one point point caused D to predict the real/fake label incorrectly.

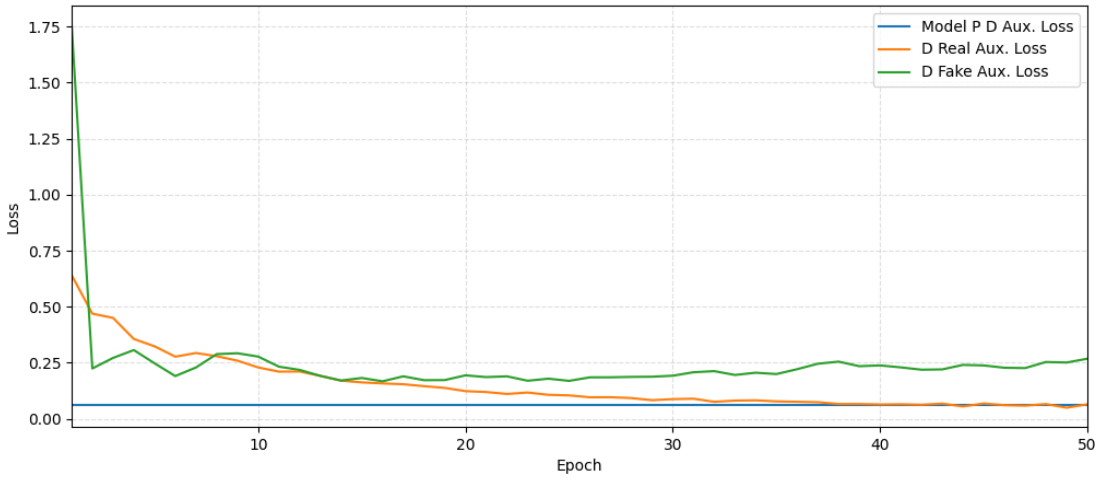

Figure 5.6: **This plot compares the evolution of auxiliary loss from D's class predictions on *real* foram images against auxiliary loss from *synthesised* image classifications.** Note the fake loss drops rapidly at first but is later surpassed by the real loss. Also plotted is the line *model A D Aux. Loss* = 0.06, showing the auxiliary loss which model A settled to. This is a benchmark which D is able to recover to after the initial impact of introducing synthesised data.

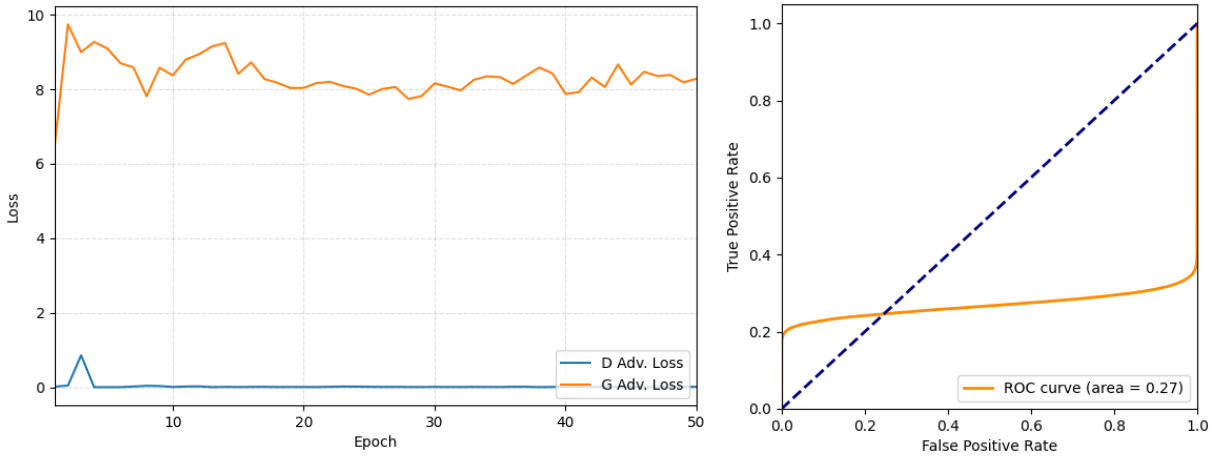

Figure 5.7: **(Left) A plot depicting the evolution of the adversarial losses calculated for D and G.** D's adversarial loss was approximately zero for the entire duration of training, while G's was almost always > 8; the desired balance between D and G's adversarial loss was never achieved. **(Right) An ROC curve showing the relationship between D's true and false positive predictions for the entire training set and an equal number of synthesised images.** In this context a 'positive' is when D predicts the source of an image as being 'real', so this graph shows D is biased towards 'real' adversarial predictions.

Looking more closely at Figure 5.7, D's adversarial loss is instantaneously at zero, since it is almost immediately able to detect the difference between G's noisy synthesised images and the clear images in the Endless Forams training set. D is initialised with its feature-extracting convolutional layers fully trained, giving it a massive advantage over G, which needs to optimise hundreds of thousands of weights before it can synthesise convincing images. For D to detect G's noisy images as fake it merely needs to learn a mapping from the image's extracted features to the adversarial classification. This 'head start' enjoyed by D makes it impossible for G to 'catch up' and synthesise images which could fool D. I believe this is a systemic issue which will arise whenever training a GAN which has a pre-trained discriminator and untrained generator.

The absence of adversarial convergence exhibited by this model is disastrous for synthesised image quality, since it removes any signal G could learn from: if D perfectly classifies every synthesised image as fake then G has nothing to learn from. This adversarial balance is known as the GANs *Nash Equilibrium* [16]. It has been posed that GANs Nash Equilibria may not exist, however optimisation of both models towards this equilibrium remains the primary adversarial training objective.

Figure 5.8 provides further convincing evidence for an inter-class mode collapse. The confusion matrix clearly shows D only predicting a small subset of the 35 total classes, which I believe is evidence that G has collapsed to only synthesising images from this select subset of classes; if G only synthesises images which resemble these dominant species, then a well trained discriminator will only predict those species. Fascinatingly, when compared to the confusion matrix of P's class predictions on real images (see Figure 5.2) a correlation in these dominant classes becomes evident: the classes which G has collapsed to are the same classes which D was most often misclassifying real images as belonging to. I believe this could be because, in G's effort to synthesise images which D can most easily classify, it has learned to always synthesise images from the classes which D predicts most often; ie. G learned to reliably synthesise these few classes that D predicts most often, because they are the classes which D will be most likely to correctly classify, *even if the image G has synthesised does not accurately resemble the images in that class.*

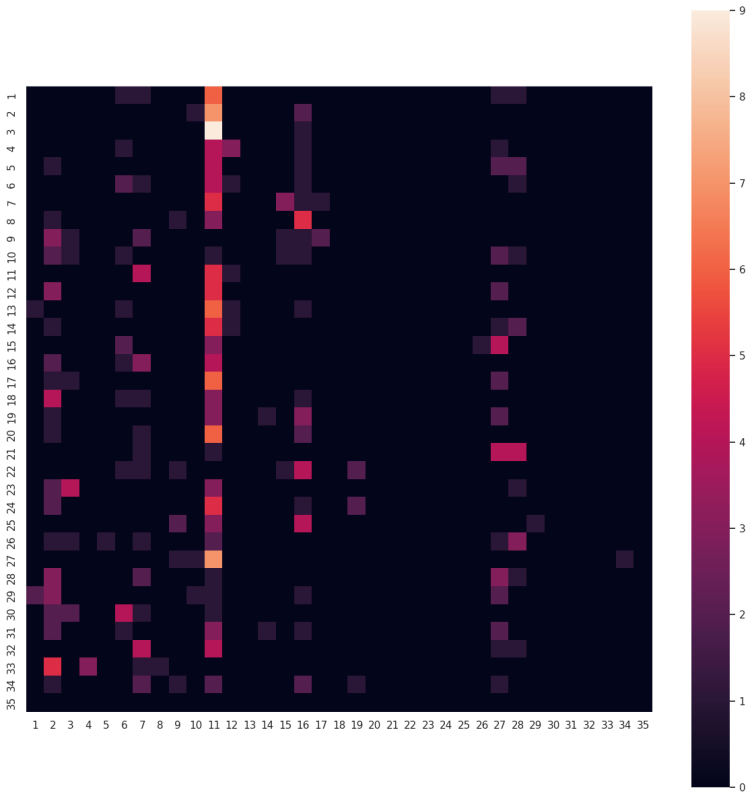

Figure 5.8: **A confusion matrix showing D's classifications, after training via G50, for a set of 350 synthesised images, containing exactly 10 images per class.** There is a clear trend of D only predicting classes from a small subset of the 35 total classes, specifically classes 2, 3, 6, 7, 16, 27 and 28.

5.3 Phasing in Synthesised Images

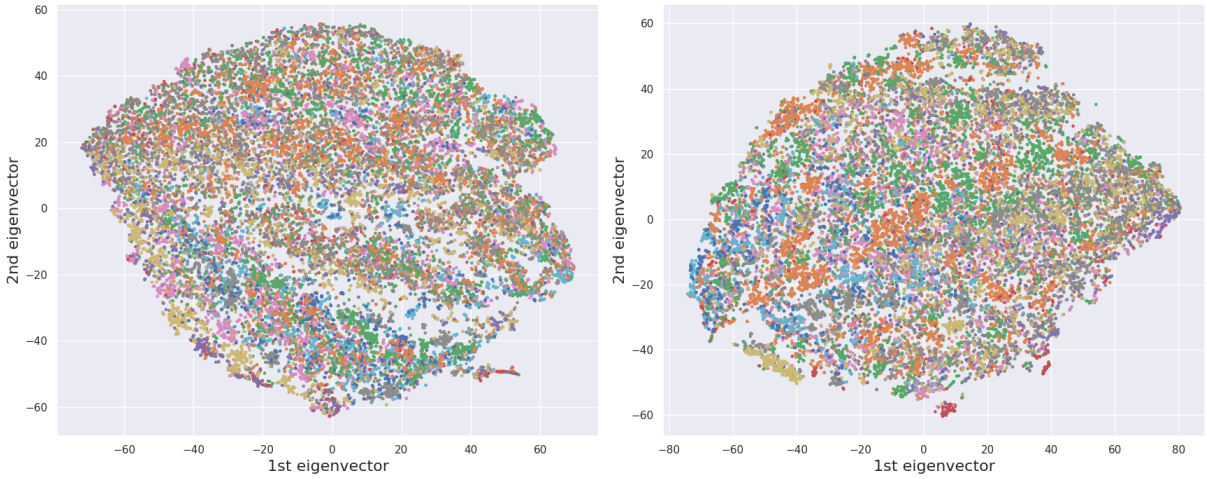

Figure 5.9: **t-SNE visualisation of all training set embeddings after 50 epochs of training with (Left) a G50 scheme and (Right) a P50 scheme.** There is more clustering visible in the Phase50 plot; P50 training has clearly helped D preserve more of the structure of the latent space.

To summarise the last section, training MLACGAN, consisting of a pre-trained discriminator and untrained generator, on a 50/50 split of real and synthesised data caused two main problems:

1. Very unstable classification performance by D.
2. D dominating G in the adversarial zero-sum game.

In an effort to overcome the first of these weaknesses I began an investigation into the effects of beginning training on minibatches of purely real images, before gradually increasing the proportion of synthesised images in each minibatch, a technique I will refer to as *phasing in synthesised images*. To achieve this affect I introduced a new hyperparameter r which, as the name suggests, determines the ratio of real to synthesised images in each minibatch. This relationship can be represented mathematically as follows:

$$F : R = r : 1 \tag{5.6}$$

where F denotes the number of fake, synthesised images and R denotes the number of real, sampled images in each minibatch. Intuitively, showing D fewer of the noisy images synthesised in the early stages, before phasing in more as the quality of the images improves, should stabilise classification performance by ensuring D's pre-optimised embedding mapping will be less affected by attempts to optimise the latent space around noisy synthesised images.

This proposed method, which I will refer to as *P50*, is essentially an interpolation between the training scheme employed by Karaderi et al. and the classical ACGAN training scheme. Therefore, I formulated a new loss function, parameterised by a new hyperparameter γ, which interpolates between the loss functions employed in these two training schemes. This new loss, capable of phasing in the contribution of the adversarial loss component, is defined in equations 4.1 and 4.2. While I opted to always interpolate linearly between a purely real batch and a 50/50 blended batch, by incrementing r linearly, I chose to experiment with three different schemes for interpolating γ, which I have defined in equations 4.10, 4.11 and 4.12.

Figure 5.10 shows a drastic fall in test set classification accuracy from ∼88% to between 20% and 30%. Unlike the behaviour exhibited by Model A in figure 5.5, this drop in accuracy is sustained for several epochs. This may seem bad at first glance, however the absence of drastic accuracy fluctuations is indicative of more stable training, one of the main goals of this implementation. Also note that by the end of 50 phased epochs the test set accuracy is sitting at around 70%, irrespective of the elected γ scheme, which is around 10% greater than the accuracy achieved after 50 epochs of training on 50/50 real/fake data.

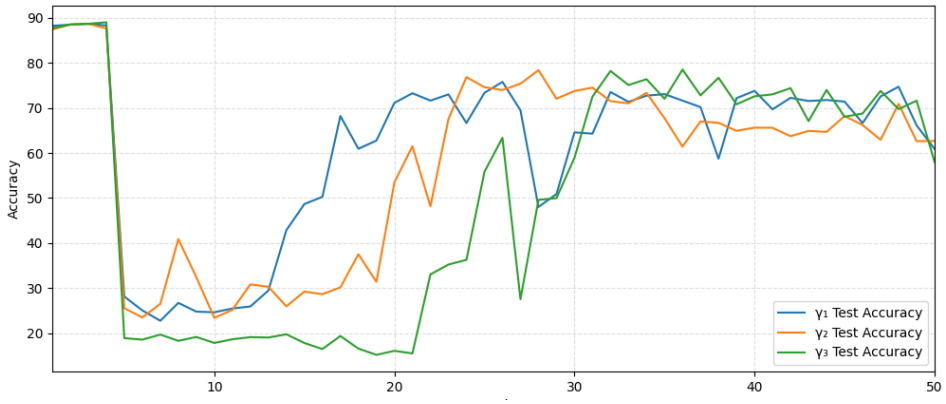

Figure 5.10: **The evolution of test set classification accuracy when synthesised images are phased in over the course of 50 epochs, with loss functions interpolated via γ_1, γ_2 and γ_3 schemes.** In all cases there is a drastic drop in accuracy when D is first shown synthesised images, with an eventual recovery to ∼70%.

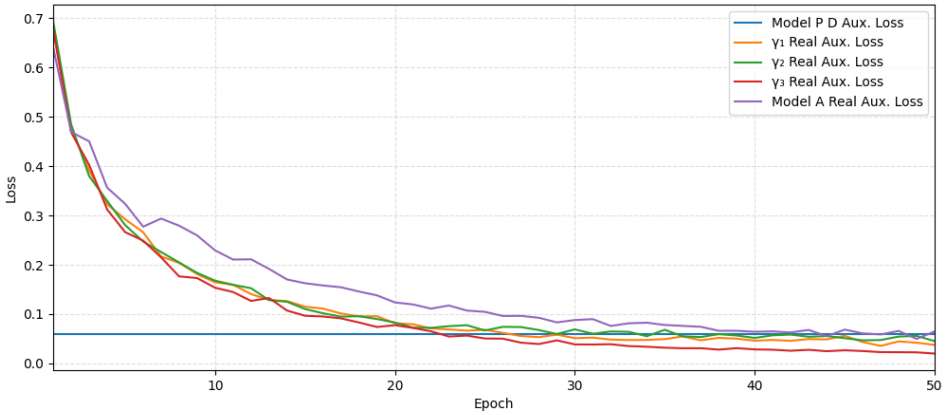

Figure 5.11: **The evolution of the auxiliary loss component calculated from real images, for models trained with the P50 scheme and γ_1, γ_2 and γ_3 each.** The real auxiliary losses for Phase models are consistently below the losses achieved by model A, and also settle to losses lower than the minimum loss achieved by model A.

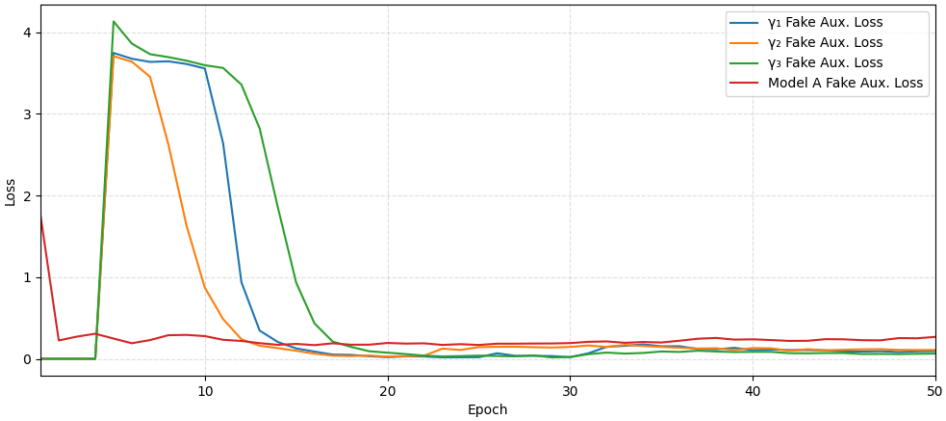

Figure 5.12: **The evolution of the fake auxiliary component for a model trained with the P50 scheme and loss functions interpolated via γ_1, γ_2 and γ_3.** Also plotted, for contrast, is the fake auxiliary loss evolution of model A trained with a classical ACGAN scheme. Note the massive spike in loss when images are first synthesised, and that all three P50 losses eventually reach a level below the model A benchmark.

Further evidence of this improvement can be seen in figures 5.11 and 5.12 where, for both real and fake images, all models trained via P50 exhibit auxiliary loss *lower* than the loss achieved when training on 50/50 real/fake data.

Figures 5.13 and 5.9 shed some light on *how* P50 training has lead to these improved accuracy and loss statistics. Firstly, the confusion matrix visualising the classification performance of a P50 model shows a decreased rate of misclassifications; There are fewer non-black cells located off of the diagonal, and many of these remaining cells show reduced colour intensity, indicating misclassifications of that nature were less frequent. Logically, in order for D to have improved its KNN classification accuracy, the latent space must be more optimally clustered, and this is indeed what we see evidence of in figure 5.9. While the left plot does show some 'miniclusters', these clusters are larger and more distinct in the right plot. This suggests that P50 training has helped D to maintain more of the structure of the latent space it inherited from model A.

The question remains however, as to whether phasing has negatively impacted G's image synthesis ability, since it is given less chance to learn adversarially in the early stages of training. Figure 5.14 shows G is indeed negatively impacted in the early stages, with its adversarial loss spiking to almost double the terminal adversarial loss of a model trained on 50/50 real/fake blend. This peak occurs around half way through training, while the prominence of synthesised data in each minibatch is still limited, before the loss eventually recovers to a similar level seen in model A. This shows quantitatively that phasing can train G to synthesise images that are just as convincing as those produced by training on a constant 50/50 blend. For qualitative evidence of this claim, see figure 5.21.

It is worth noting that, while the various γ schemes do cause the model to learn in a slightly different way, this appears to have almost no bearing on the final training outcomes. Hence, from now on I will experiment exclusively with the γ_3 scheme, which I will refer to simply as γ.

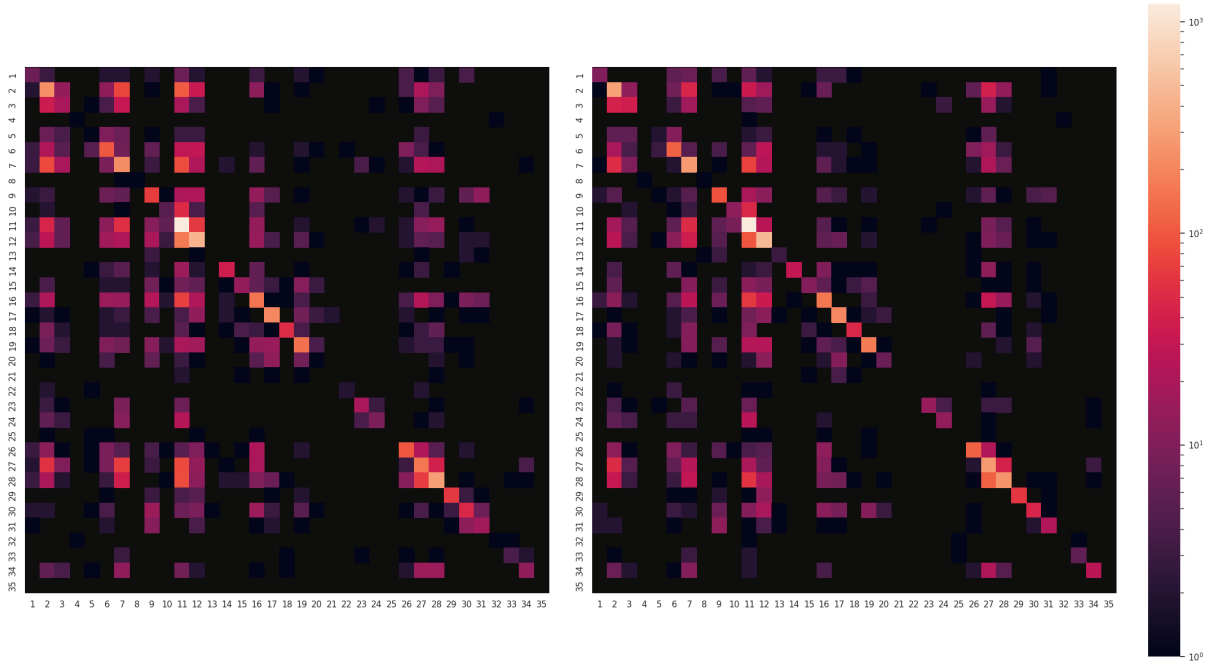

Figure 5.13: **Confusion matrices visualising the most common misclassifications made by D when trained via (Left) a G50 scheme and (Right) a P50 scheme.** P50 training has produced marginally improved classification performance, with slightly fewer visible confusions.

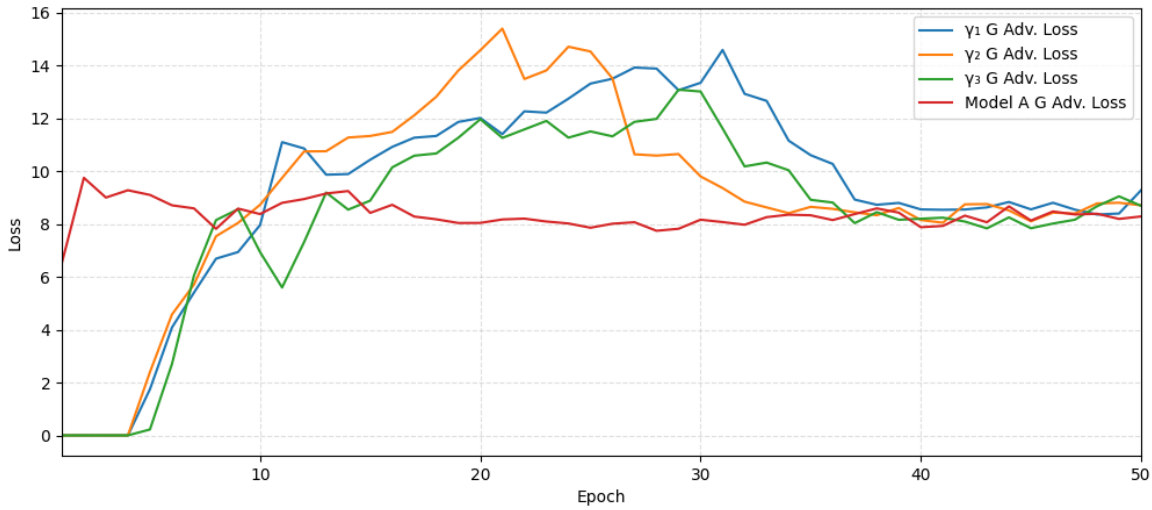

Figure 5.14: **The evolution of G's adversarial loss when trained via P50 and each of γ_1, γ_2 and γ_3.** Phasing clearly causes a spike in the mid-stages of training but all models recover to the same level level as model A.

5.4 Reformulating the Aggregate Loss Function

My initial thoughts upon analysing figure 5.15 were that training D on a 50/50 mix of real and synthesised images must have somehow taught D to recognise some extra features in the real training images, perhaps by G presenting some overlooked features of the data to D in a new light. However, after more thought I realised that this increase in softmax accuracy may actually be caused by softmax loss overpowering the reciprocal triplet loss component. This hypothesis is further supported by 5.5, which shows a marked decrease in the model's KNN classification accuracy from the benchmark 91.66% posted by model A, evidencing a decline in the model's ability to optimise the latent space by minimising RTL. I soon realised that the formulation of the loss function was indeed weighting RTL inversely proportionally with γ:

$$\mathrm{L}_D = (1-\gamma)\mathrm{L}_{SoftMax} + (1-\gamma)\lambda\mathrm{L}_{RTL} + \gamma\mathrm{L}_{Ad} \tag{5.7}$$

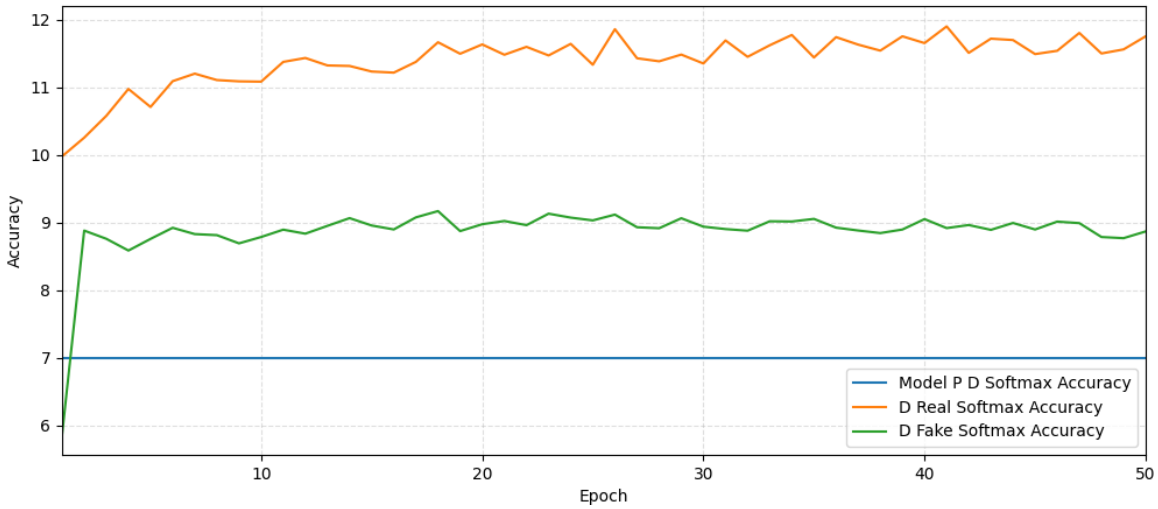

Figure 5.15: **A plot depicting the evolution of the accuracy of the discriminator's softmax predictions for the class labels of both real and fake images, when trained with r=1.** Also plotted is the line *model A D Softmax Accuracy* = 7%, indicating the softmax accuracy which model A settled to. The accuracy of D's softmax predictions on real images has surpassed this benchmark, showing that training D with r=1 has improved the quality of these predictions.

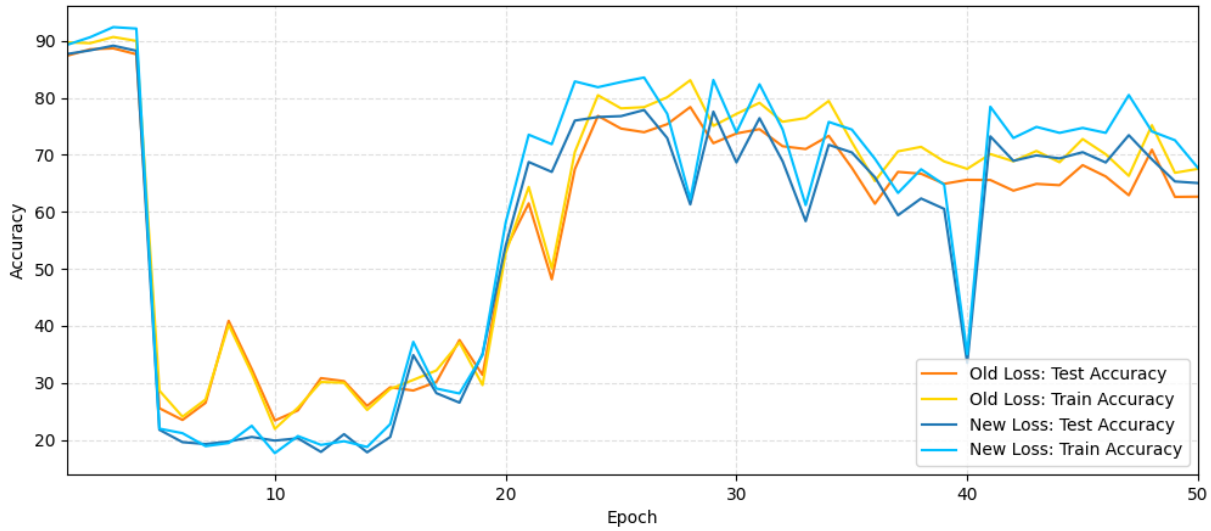

Figure 5.16: **A plot contrasting the accuracies of two models, with one optimised via the minimisation of each of the new loss function and the old loss function.** During the last 10 epochs of training the new loss model achieves a loss 5-10% greater than that of the old loss model.

Specifically, if r=0 then $\gamma = 0$, so L_{RTL} is weighted by λ as intended, while if r=1 then $\gamma = \frac{1}{2}$, regardless of gamma's formulation, so L_{RTL} is instead weighted by $\frac{1}{2}\lambda$. Indeed, as hypothesised L_{RTL} has been given less weighting in the experiment employing a 50/50 mix of synthesised and real data. This consequently interfered with the λ setting of 0.01 recommended by [25], effectively reducing the weighting parameter to 0.005 in the r=1 case.

To test whether this unintended reduction in the weighting of L_{RTL} had indeed negatively affected the KNN classification accuracy of D I devised a reformulated loss function which ensures the weighting on the RTL component is independent of γ:

$$L_D = (1-\gamma)L_{SoftMax} + \gamma L_{Ad} + \lambda L_{RTL} \tag{5.8}$$

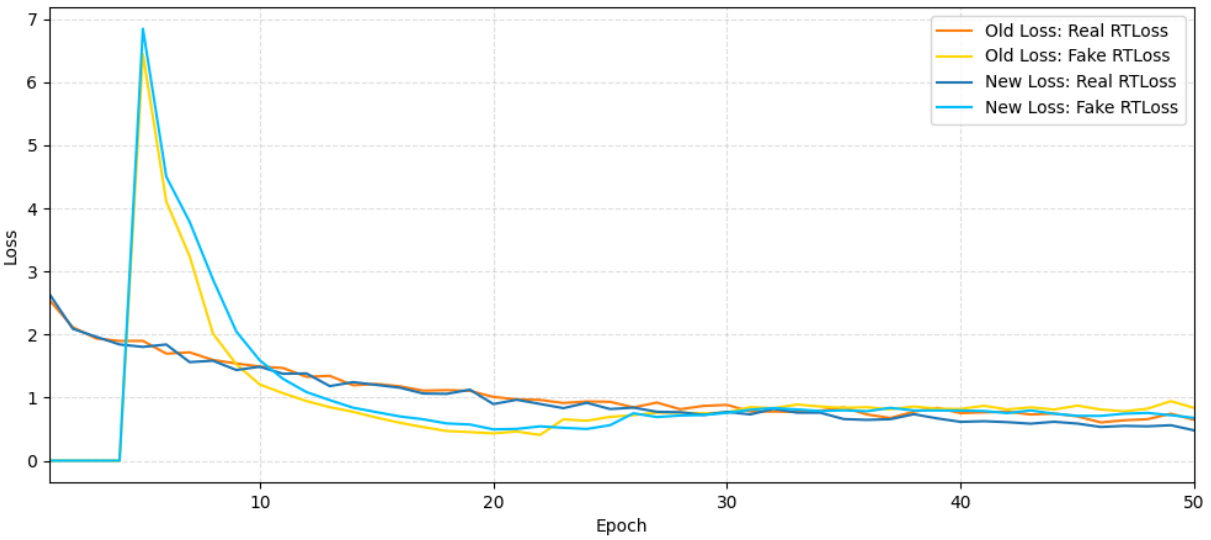

Figure 5.17: **A plot of both real and fake reciprocal triplet losses produced by two models, one with each of the new and old loss functions.** The two real RTL curves are almost identical, signalling that there is minimal impact on the real image embeddings from altering the weighting of the RTL component.

Figure 5.16 shows that a model optimised via the minimisation of this 'new' loss function can indeed achieve better test set classification accuracy. Within the first 15 epochs of training, the accuracy from the new loss function hits a lower minimum than the minimum accuracy from the old function, which makes sense since the new loss function gives the RTL component almost double the weighting during these early stages. This greater weighting causes D to put greater emphasis on fitting the few noisy images it is fed early on into its embeddings space, which in turn hinders the model's ability to generalise to unseeen test images. This is only temporary however, as the new loss accuracy then recovers to a level approximately 5% greater than that of the old loss. I attribute this to fact that RTL has truly been weighted by $\lambda = 0.01$ for the duration of training.

Figure 5.17 appears to repeat this trend of the new loss function granting worse performance initially, with a recovery to a marginally improved terminal performance. Specifically, the new loss fake RTL is higher from 5-20 epochs and subsequently improves to be lower from epoch 35 onwards. However, this evidence has led me to conclude that this new loss formulation has minimal impact on the optimisation of the embedding space. Having said this, due to the marginal performance increase I will use this loss formulation for the remainder of the experiments.

5.5 Biasing Training Towards the Generator

Complete adversarial dominance by the D has been a consistent trend present in all generative models trained so far. This dominance spells disaster for G's ability to learn to synthesise convincing foram images, so must be avoided at all costs. It is somewhat unsurprising that D dominates G completely, since D is always initialised with pre-trained weights, while G is not, giving D a significant head-start in detecting the true source of G's synthesised images. To overcome this adversarial imbalance I propose biasing G's loss function towards its adversarial component by weighting the adversarial loss component in L_G more heavily than in L_D, which, in theory, affords G a chance to improve the quality of its synthesised images faster than D can learn to detect them.

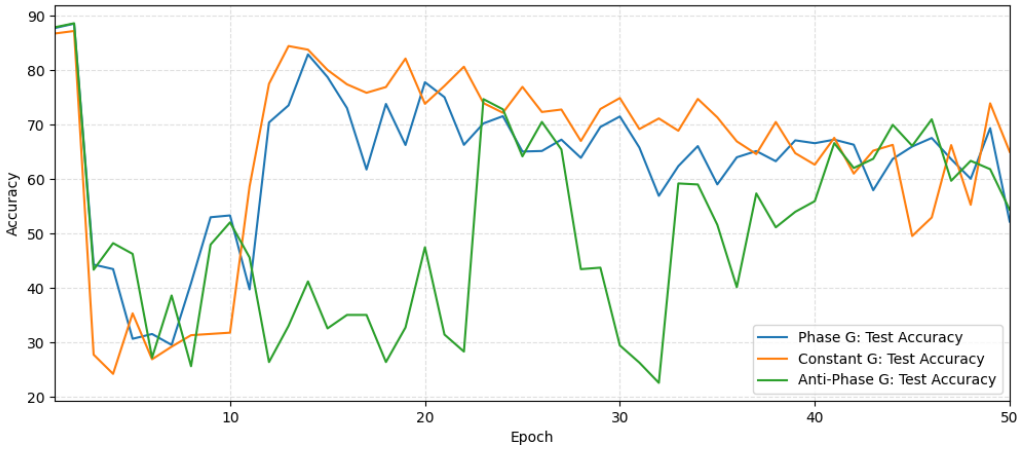

Figure 5.18: **A plot displaying the impacts of various generator phasing schemes on the evolution of test set accuracy throughout 50 phased training epochs.** From epoch 15 onwards, the Constant G scheme appears to give a 5-10% accuracy boost over the baseline Phase scheme accuracy. Anti-Phase has clearly introduced some accuracy instability to the network.

All Phase schemes proposed involve weighting D's adversarial loss with zero initially, before interpolating that value to 1 by the end of training. This means that D is learning to detect synthesised images at a slower rate during the early stages of training, which I propose to take advantage of via two new schemes for interpolating G's loss.

1. *Constant G* - G's adversarial and softmax loss components are given constant, equal weighting throughout training, formalised by the following loss function:

$$L_G = \frac{1}{2}L_{SoftMax} - \frac{1}{2}L_{Ad} + \lambda L_{RTL} \tag{5.9}$$

2. *Anti-Phase G* - The weighting of G's adversarial and softmax loss components are reversed in relation to D:

$$L_G = \gamma L_{SoftMax} - (1-\gamma)L_{Ad} + \lambda L_{RTL} \tag{5.10}$$

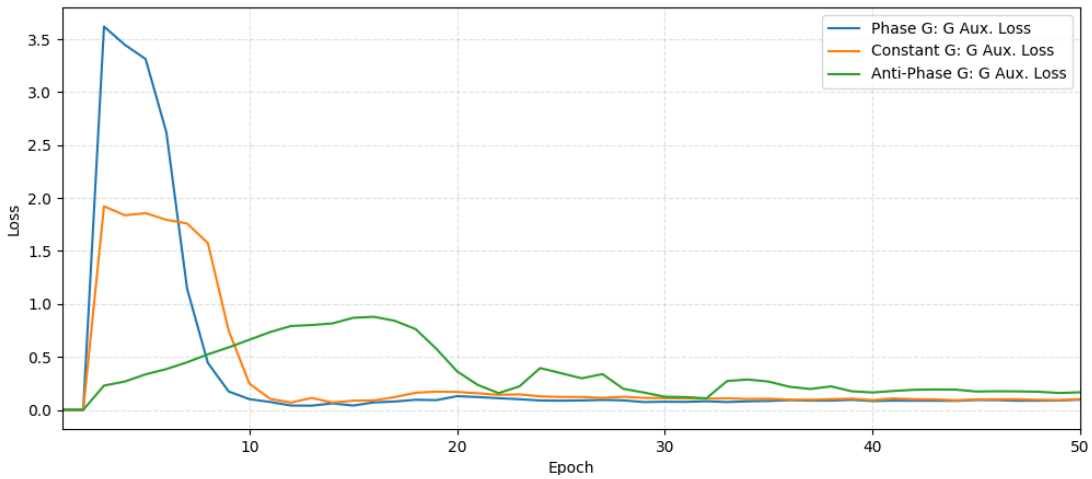

Figure 5.19: **A plot comparing the evolution of G's auxiliary loss while training via Phase, Constant and Anti-Phase generator schemes.** Compared to the baseline Phase G curve, both other G schemes lead to a delayed and mitigated auxiliary loss spike, as well as a delayed recovery to baseline loss. This behaviour is exaggerated in Anti-Phase.

Notice, both new schemes feature the independently weighted L_{RTL} component introduced in the previous section. If we train a model using a Phase scheme, with D optimised via the minimisation of the loss defined in equation 5.8, then both of these schemes satisfy the goal of weighting the adversarial loss component more heavily in G's loss than in D's. To test the impact of these new schemes, I trained models with Constant and Anti-Phase G, as well as regularly phased G as a control, with P25 and then G25 schemes, which I will refer to as P25G25. Figure 5.18 is promising evidence that models trained with a Constant G scheme can classify real foram images at least as accurately as when trained with a Phase G scheme. Unfortunately, Anti-Phase seems to have introduced training instability to the model, with the appearance of large fluctuations in accuracy. This hypothesis is supported by figure 5.19, which shows fluctuations in G's auxiliary loss, as well as higher terminal loss.

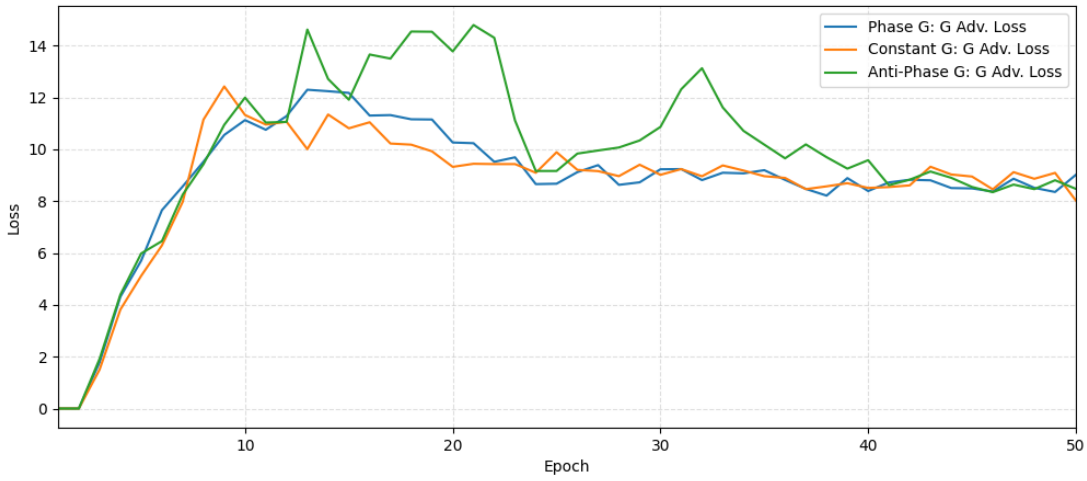

Figure 5.20: **A plot of G's adversarial loss evolution over the course of 50 phased epochs, comparing the impacts of a variety of G phasing schemes.** Phase and Constant G schemes show generally little difference, with a slight improvement by Constant G in the early-mid stages. Anti-Phase has clearly introduced loss instability to the network.

Instability introduced by Anti-Phase training is seen again in figure 5.20 with erratic fluctuations in the adversarial loss, which is a symptom of generator mode collapse. A generator suffering from mode collapse maps many points within the noise space to the same output mode, meaning any changes to the model's weights that affect the appearance of one of these modes will have a wide reaching affect, changing the appearance of all the output images in that mode, and leading to sudden shifts in loss.

Figure 5.21 compares the generative performance of models trained via P25G25 phasing and P50 phasing, with Phase, Constant and Anti-Phase G schemes. The Anti-Phase results show that this scheme has indeed induced severe mode collapse, spanning between separate classes. P50 with Phase G shows signs of mode collapse and produces markedly lower quality images than G50 images shown in figure 5.4. P25G25 Phase G has terrible quality images, while P25G25 Constant and Anti-Phase have produced far higher quality images, rivaling those from G50. Anti-Phase has, as suspected, suffered severe mode collapse spanning between classes.

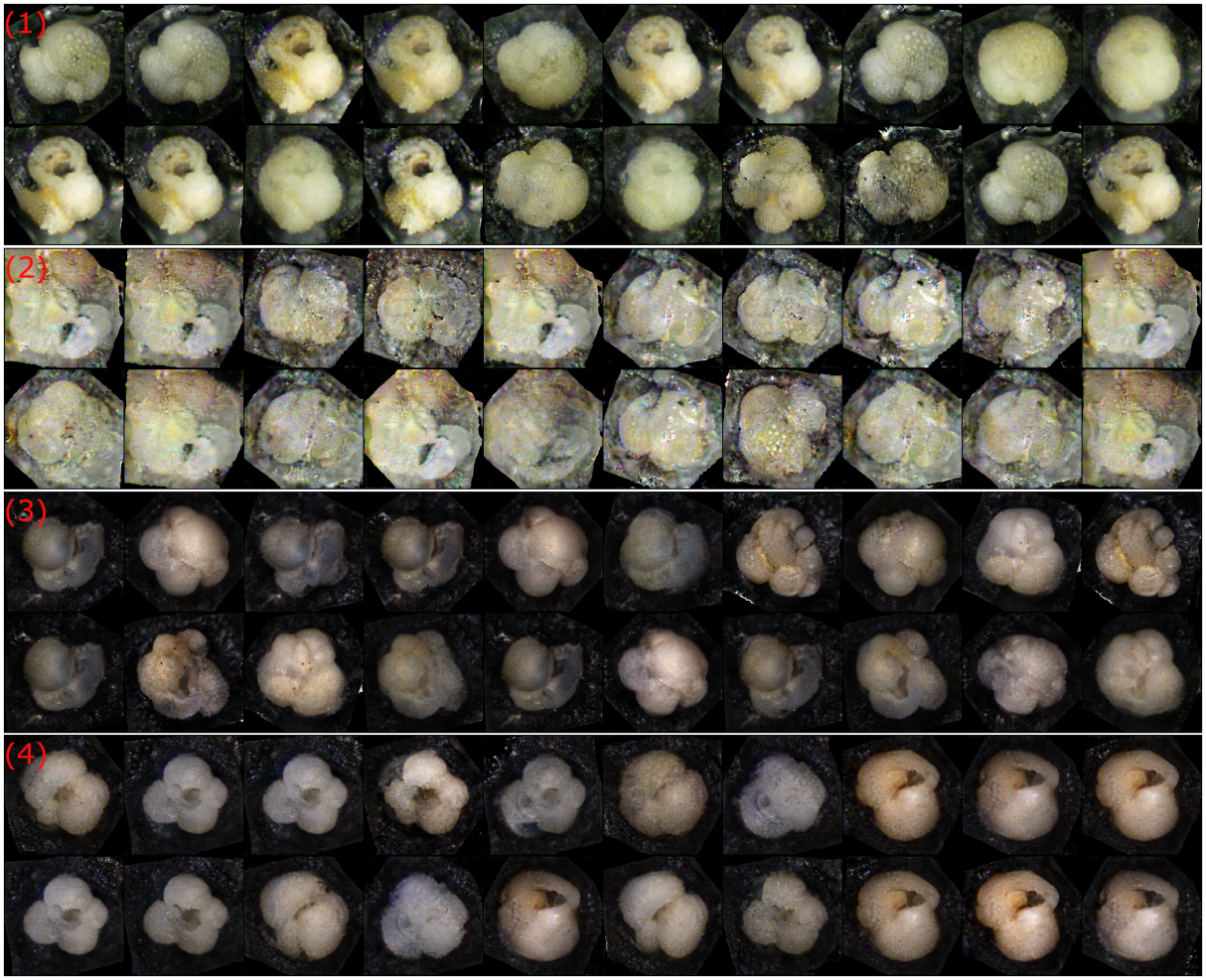

Figure 5.21: **Foram images randomly synthesised by MLACGANs trained using a variety of phasing schemes.** In each pair of rows, the top and bottom rows are conditioned on classes 6 and 7 respectively, with both synthesised by the same model. The training schemes used are **(1)** P50, Phase G **(2)** P25G25, Phase G **(3)** P25G25, Constant G **(4)** P25G25, Anti-Phase G, all of which use γ_3 and new RTL weighting.

5.6 Results summary

This final section summarises the above findings in table 5.1, which facilitates quantitative comparisons at a glance. The statistical metrics filling the right hand side of the table measure the classification performances of each model. The final row, containing results from following P25G25 scheme with the new RTL weighting and Anti-Phase generator phasing, shows the best classification performance of any generative model by some margin. This is no surprise however, as this model represents the culmination of all the best-performing phasing behaviours proposed throughout this chapter.

Unfortunately, none of the MLACGAN models trained can compete with the cutting edge CNN classification accuracy reported by [32]. It appears that optimising the embedding space around imperfect, synthesised images seriously hurts the models ability to forms clusters in the latent space. This negative impact of applying RTL to synthesised images can be visualised by contrasting figures 5.3 and 5.9. I also believe that minimising RTL with respect to triplets of synthesised images has been a large contributing factor to the mode collapse present in all experiments. The smallest distance possible between two points is zero, when the two points are equal. With this in mind, in order to fully minimise the distance between the anchor and positive images in each synthesised triplet, G *must* learn to synthesise anchor and positive pairs which are *equal*. Logically, this process will eventually lead to G only synthesising one image per class; for any conditioning class label, no matter the noise vector input, the output image will always be identical.

Another significant weakness of MLACGAN is the discriminator's dominance, which has been observed in all experiments involving synthesised images. I believe this can be attributed to two factors:

- The discriminator is pre-trained, giving it a head-start over G.
- The discriminator network is a powerful ResNet50-based architecture.

Training	**LR**	**r**	**γ**	**Pre-T D**	**RTL w.**	**G Loss**	**Acc.**	**Prec.**	**Recall**	**F1**
[32]	0.0001	0	n/a	ResNet50	n/a	n/a	91.6	88.2	78.3	81.3
R20	0.0001	0	n/a	[32]	Old	n/a	91.24	89.25	79.52	83.32
R20	0.00005	0	n/a	[32]	Old	n/a	90.99	86.97	79.28	81.78
R20	0.00001	0	n/a	[32]	Old	n/a	91.66	90.95	81.16	84.66
G50	0.0002	1	n/a	A	Old	Const.	52.92	47.02	34.86	38.54
P50	0.0002	Phase	1	A	Old	Phase	55.31	47.96	35.11	40.54
P50	0.0002	Phase	2	A	Old	Phase	45.39	34.57	24.30	26.63
P50	0.0002	Phase	3	A	Old	Phase	57.97	50.42	36.89	40.63
P50	0.0002	Phase	3	A	New	Phase	62.66	54.91	39.71	44.09
P25G25	0.0002	Phase	3	A	New	Phase	52.22	41.33	29.74	32.41
P25G25	0.0002	Phase	3	A	New	Const.	49.57	46.87	33.19	36.52
P25G25	0.0002	Phase	3	A	New	AntiPh.	63.77	63.20	46.52	50.47

Table 5.1: **This table compares a variety of classification performance metrics, measured from MLACGANs trained using each of the schemes proposed in this chapter.** Also featured are the results reported by Karaderi et al. [32]. R20 results are taken during the model's best performing epoch, with all other results measured at the end of training. All precision, recall and F1 scores were calculated with macro averaging.

Chapter 6

Conclusion

In this thesis I have proposed MLACGAN, an ACGAN derivative which I have shown can accurately classify foram images, as well as synthesise images which convincingly resemble a number of foram species. This model builds on the work of Karaderi et al. [32] by similarly embedding images in a metric space which is optimised via minimisation of reciprocal triplet loss. Classifications can then be drawn from the embeddings by performing K-Nearest Neighbours classification in the latent space. I employed a novel approach of using this metric learning technique to achieve class-conditioned image synthesis, which was executed by embedding the synthesised images in the same latent space, before optimising the embeddings distances. During my analysis, I noticed a trend that, with this approach, G is biased towards synthesising images from classes which D most frequently incorrectly predicts images as belonging to. I found an explanation for this phenomena, in that G synthesises classes which D is most likely to predict, even if G's synthesised representation is poor.

I also proposed an array of schemes for phasing in synthesised images during training. I researched the impacts of each of these schemes on the model's accuracy classifying the Endless Forams dataset, and was able to demonstrate a quantifiable improvement over the classification accuracy achieved when training an MLACGAN *without* phasing. I also showed that phasing in synthesised images works to reduce the disruption to the clustering in the latent space. My analysis uncovered that G is dominated by D in the adversarial zero-sum game, and identified this as a cause of G's poor synthesis quality. In response to this weakness, I theorised that biasing the adversarial game towards G, via alterations to the loss function, could help G overcome D's dominance. I subsequently collected both quantitative and qualitative evidence that this modification improves both classification and image synthesis performance, with the model phased in this way achieving the greatest Endless Forams test set classification accuracy of all MLACGANs, with 63.77%.

6.1 Future Work

There are certainly weaknesses with the approach investigated in this thesis, such as significantly lower classification accuracy than can be achieved with CNNs, as well as systemic mode collapse for the synthesised images. I will now present some directions in which future works could take the MLACGAN concept, which will each focus on tackling these identified issues.

I have attributed the poor classification accuracy to the negative impact of minimising reciprocal triplet loss with respect to triplets of synthesised images. In order to minimise this impact, future works could investigate *feeding D triplets which consist of a blend of real and fake images.* I believe this could help preserve the clusters in the latent space. Furthermore, this direct comparison of distances between real and fake image embeddings could help G learn to synthesise images which even more closely resemble the images in their target class, and differ from the images in other classes. For example, if RTL minimisation is applied to triplets formed from a synthesised anchor with real positive and negative samples, then G will learn to synthesise images which will be embedded close to the positive but far from the negative.

A future work which solely targets the generative objective of MLACGAN could build on this idea

of using mixed triplets with a synthesised anchor image and real positive and negative images, by **pre-training D as a classifier, ie. via R20, then fixing these real image embeddings and minimising the RTL of these mixed triplets by only training G**. This would teach G to synthesise images which map to embeddedings within the clusters formed during the R20 stage.

A large weakness of the approaches researched in this paper is the complete discriminator dominance they give rise to. I identified one of the causes as D being pre-trained while G is not. I believe this effect could be mitigated by **pre-training G on the same dataset using a standard ACGAN architecture and training scheme**, ie. start with a simple and untrained discriminator and omit the reciprocal triplet loss component from both models. This would effectively pre-train G to be able to synthesise somewhat convincing images. The effects of initialising MLACGAN with both pre-trained G and D could make an interesting future research direction. There are other common ways of reducing discriminator domination which could additionally be investigated, such as impairing its adversarial ability by randomly feeding it incorrect source labels, or adding dropout layers. These practices could also reduce the prevalence of mode collapse in G.

Bibliography

[1] Foraminifera.

[2] Evidence: Sediment cores: Amnh, Oct 2008.

[3] Paleoclimatology, 2021.

[4] Nadia Al-Sabouni, Isabel S Fenton, Richard J Telford, and Michal Kučera. Reproducibility of species recognition in modern planktonic foraminifera and its implications for analyses of community structure. *Journal of Micropalaeontology*, 37(2):519–534, 2018.

[5] Renaud Allioux. How to use deep learning on satellite imagery - playing with the loss function, Feb 2019.

[6] Wessex Archaeology. Complete seabed cores, Feb 2011.

[7] Michael Auli. Self-supervised learning: The dark matter of intelligence, May 2019.

[8] Jessica Blunden. State of the climate: Global climate report for annual 2020, Jan 2021.

[9] CJ Carey, Yuan Tang, and William de Vazelhes. 1. what is metric learning?, 2020.

[10] Juan Carrillo, Mark Crowley, Guangyuan Pan, and Liping Fu. Design of efficient deep learning models for determining road surface condition from roadside camera images and weather data. *arXiv preprint arXiv:2009.10282*, 2020.

[11] Viviane Clement, Kanta Kumari Rigaud, Alex de Sherbinin, Bryan Jones, Susana Adamo, Jacob Schewe, Nian Sadiq, and Elham Shabahat. *Groundswell Part 2: Acting on Internal Climate Migration*. World Bank, 2021.

[12] MICHAEL J. CULLINANE. Metric axioms and distance. *The Mathematical Gazette*, 95(534):414–419, 2011.

[13] Haskell B Curry. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261, 1944.

[14] Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962.

[15] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[16] Farzan Farnia and Asuman Ozdaglar. Gans may have no nash equilibria. *arXiv preprint arXiv:2002.09124*, 2020.

[17] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

[18] James Fleming. *The callendar effect: the life and work of Guy Stewart Callendar (1898-1964)*. Springer Science & Business Media, 2013.

[19] Caroline Gardiner. Advanced computing research centre, Oct 2017.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 6.2. 2.3 softmax units for multinoulli output distributions. *Deep learning*, page 180, 2016.

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[22] Ed Hawkins, Pablo Ortega, Emma Suckling, Andrew Schurer, Gabi Hegerl, Phil Jones, Manoj Joshi, Timothy J Osborn, Valérie Masson-Delmotte, Juliette Mignot, et al. Estimating changes in global temperature since the preindustrial period. *Bulletin of the American Meteorological Society*, 98(9):1841–1856, 2017.

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

[24] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[25] Tomáš Hodan, Pavel Haluza, Štepán Obdržálek, Jiri Matas, Manolis Lourakis, and Xenophon Zabulis. T-less: An rgb-d dataset for 6d pose estimation of texture-less objects. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888. IEEE, 2017.

[26] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International workshop on similarity-based pattern recognition*, pages 84–92. Springer, 2015.

[27] Benjamin P Horton, Nicole S Khan, Niamh Cahill, Janice SH Lee, Timothy A Shaw, Andra J Garner, Andrew C Kemp, Simon E Engelhart, and Stefan Rahmstorf. Estimating global mean sea-level rise and its uncertainties by 2100 and 2300 from an expert survey. *NPJ climate and atmospheric science*, 3(1):1–8, 2020.

[28] Hospital. *Analyse des infiniment petits pour l'intelligence des lignes courbes*. Nabu Press, 1696.

[29] Allison Y Hsiang, Anieke Brombacher, Marina C Rillo, Maryline J Mleneck-Vautravers, Stephen Conn, Sian Lordsmith, Anna Jentzen, Michael J Henehan, Brett Metcalfe, Isabel S Fenton, et al. Endless forams:¿ 34,000 modern planktonic foraminiferal images for taxonomic training and automated species recognition using convolutional neural networks. *Paleoceanography and Paleoclimatology*, 34(7):1157–1177, 2019.

[30] Alekse Grigorevich Ivakhnenko and Lapa Valentin Grigorevich. *Cybernetics and forecasting techniques*, volume 8. American Elsevier Publishing Company, 1967.

[31] Suraj Kamal, A Mujeeb, MH Supriya, et al. Generative adversarial learning for improved data efficiency in underwater target classification. *Engineering Science and Technology, an International Journal*, 30:101043, 2022.

[32] Tayfun Karaderi, Tilo Burghardt, Allison Y Hsiang, Jacob Ramaer, and Daniela N Schmidt. Visual microfossil identificationvia deep metric learning. *arXiv preprint arXiv:2112.09490*, 2021.

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[35] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[36] CMG lee. Comparison of 1d and 2d interpolation. 2020. File: `Comparison of 1D and 2D interpolation.svg`.

[37] Z Li, WT Nash, SP O'Brien, Y Qiu, RK Gupta, and N Birbilis. cardigan: A generative adversarial network model for design and discovery of multi principal element alloys. *Journal of Materials Science & Technology*, 2022.

[38] Pincelli M. Hull and Y. Allison Hsiang, 2019.

[39] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[40] Alessandro Masullo, Tilo Burghardt, Dima Damen, Toby Perrett, and Majid Mirmehdi. Who goes there? exploiting silhouettes and wearable signals for subject identification in multi-person environments. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0--0, 2019.

[41] Mark Alan Matties. Vector embeddings with subvector permutation invariance using a triplet enhanced autoencoder. *arXiv preprint arXiv:2011.09550*, 2020.

[42] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115--133, 1943.

[43] Kyle McDonald. T-sne embedding of mnist, Nov 2017.

[44] Ajkel Mino and Gerasimos Spanakis. Logan: Generating logos with a generative adversarial neural network conditioned on color. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 965--970. IEEE, 2018.

[45] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[46] R Mitra, TM Marchitto, Q Ge, B Zhong, B Kanakiya, MS Cook, JS Fehrenbacher, JD Ortiz, A Tripati, and E Lobaton. Automated species-level identification of planktic foraminifera using convolutional neural networks, with comparison to human performance. *Marine Micropaleontology*, 147:16--24, 2019.

[47] Eric Muccino. Improving classification accuracy with acgan (keras), Apr 2021.

[48] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[49] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*, pages 2642--2651. PMLR, 2017.

[50] World Health Organization et al. Cop24 special report: health and climate change. 2018.

[51] Rukshan Pramoditha. The concept of artificial neurons (perceptrons) in neural networks, Dec 2021.

[52] Sara C Pryor, Donald Scavia, Charles Downer, Marc Gaden, Louis Iverson, Rolf Nordstrom, Jonathan Patz, and G Phillip Robertson. Midwest. climate change impacts in the united states: The third national climate assessment. *In: Melillo, JM; Richmond, TC; Yohe, GW, eds. National Climate Assessment Report. Washington, DC: US Global Change Research Program: 418-440.*, pages 418--440, 2014.

[53] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[54] Dejan Radivojevic. A hundred years of milutin milanković's climate change theory -geological implications. *Geoloski anali Balkanskog poluostrva*, 81, 03 2021.

[55] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[56] Kamal Ranaweera, Adam P Harrison, Santo Bains, and Dileepan Joseph. Feasibility of computer-aided identification of foraminiferal tests. *Marine Micropaleontology*, 72(1-2):66--75, 2009.

[57] David Saad. *On-line learning in neural networks*. Cambridge University Press, 1999.

[58] Nikunj Saunshi, Jordan Ash, Surbhi Goel, Dipendra Misra, Cyril Zhang, Sanjeev Arora, Sham Kakade, and Akshay Krishnamurthy. Understanding contrastive learning requires incorporating inductive biases. *arXiv preprint arXiv:2202.14037*, 2022.

[59] Gerhard Schmiedl. Use of foraminifera in climate science. In *Oxford Research Encyclopedia of Climate Science*. 2019.

[60] William Sweet, Joseph Park, John Marra, Chris Zervas, and Stephen Gill. Sea level rise and nuisance flood frequency changes around the united states. 2014.

[61] Fariborz Taherkhani, Jeremy Dawson, and Nasser M Nasrabadi. Deep sparse band selection for hyperspectral face recognition. In *Hyperspectral Image Analysis*, pages 319--350. Springer, 2020.

[62] Tavish. K nearest neighbor: Knn algorithm: Knn in python & r, Mar 2018.

[63] Graham Templeton. Artificial neural networks are changing the world. what are they?, Oct 2015.

[64] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26--31, 2012.

[65] Kevin E Trenberth and John T Fasullo. An apparent hiatus in global warming? *Earth's Future*, 1(1):19--32, 2013.

[66] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[67] John Veron. Coral bleaching events, 2018.

[68] Pascaline Wallemacq, Regina Below, and Denis McClean. *Economic losses, poverty & disasters: 1998-2017*. United Nations Office for Disaster Risk Reduction, 2018.

[69] Karen Wetmore. Foram facts | anintroduction to foraminifera.

[70] Kouichi Yamaguchi, Kenji Sakamoto, Toshio Akabane, and Yoshiji Fujimoto. A neural network for speaker-independent isolated word recognition. In *ICSLP*, 1990.

Appendix A

An Example Appendix

Content which is not central to, but may enhance the dissertation can be included in one or more appendices; examples include, but are not limited to

- lengthy mathematical proofs, numerical or graphical results which are summarised in the main body,
- sample or example calculations, and
- results of user studies or questionnaires.

Note that in line with most research conferences, the marking panel is not obliged to read such appendices. The point of including them is to serve as an additional reference if and only if the marker needs it in order to check something in the main text. For example, the marker might check a program listing in an appendix if they think the description in the main dissertation is ambiguous.