DEPARTMENT OF COMPUTER SCIENCE

On the Classification of Planktonic Foraminifera with Convolutional Neural Networks

Jacob Ramaer

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

Tuesday 1st September, 2020

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of BSc in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

JRamaer
JRamaer (Sep 1, 2020 23:02 GMT+1)

Jacob Ramaer, Tuesday 1st September, 2020

Abstract

This thesis applies current deep learning practices to the classification of small-scale, single-celled organisms named planktonic foraminifera. We show that our suggested setup outperforms the published state-of-the art in the domain [1] by 3.14% when training on a manually labelled reference dataset consisting of 34,640 foram samples across 35 separate planktonic foraminiferal species. This particular dataset is currently subject to major class imbalance, with the most abundant species consisting of nearly 1500% more samples than the least abundant species.

Over the last few years, a variety of machine learning methods have provided several significant accuracy improvements in regards to the class imbalance problem. In this paper, our focus was on designing and evaluating new image classification strategies for this specific classification task, using current state-of-the-art deep-learning methods. This included rigorous data augmentation, tests with various loss functions [2] and selective attention models [3]. We cross-validate our results to reach a final validation accuracy of 90.55%. The baseline convolutional neural networks we compare against are publicly available on endlessforams.org [1], where the best baseline network reached a top validation accuracy of 87.41%.

Contents

Acknowledgements

Firstly, I would like to thank my supervisor Dr. Tilo Burghardt for his unrelenting support and guidance throughout the course of this year, introducing me to an endlessly interesting topic which has kindled my enthusiasm for machine learning. Furthermore, I would like to thank Dr. Allison Hsiang, Dr. Daniela Schmidt and Dr. Heather Birch for their insights into the complex world of geological taxonomy.

Finally, this thesis could not have been completed without the tremendous support and encouragement from my family during the Covid-19 pandemic.

Chapter 1

Introduction

1.1 Planktonic Foraminifera

Machine learning techniques have only recently been applied to the geological domain for streamlining the tedious job of fossil species classification, specifically for planktonic foraminifera (or forams for short). These tiny, single-celled organisms are found in every corner of the Earth's oceans and are characterised by their distinctive chambered shells, which resemble chambers. There exist two broad groups of foraminifera; planktonic species dwell near the ocean's surface, in contrast to benthic species which reside in and on the sea floor.

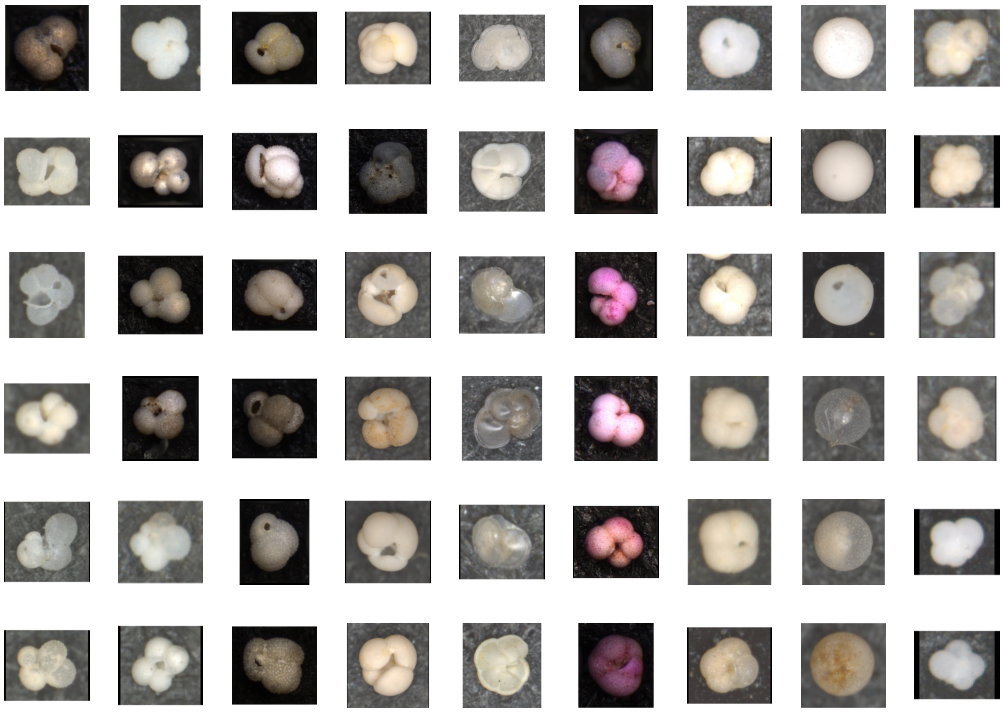


Figure 1.1: A variety of different foram species from the dataset [1], where samples in the same column are of the same species. Left to right - Beella Digitata, Globigerina Falconensis, Globigerinoides Elongatus, Globoquadrina Conglomerata, Globorotalia Menardii, Neogloboquadrina Incompta, Neogloboquadrina Pachyderma, Orbulina Universa, Tenuitella Iota

Forams are incredibly useful in understanding the makeup and history of the Earth's surface. When they die, they collect on the sea floor to form layers of carbonate sediment. Physical characteristics differ massively between species, and different species thrive under different environmental conditions. As forams mature, they grow additional chambers which can give a good indication of what the water column was like throughout their life. Because of this, their remains provide a useful snapshot of the earth at the exact point when they died. With the earliest fossils dating to 160 million years old, many different species have evolved to live in a variety of different environmental conditions. An abundance or absence of certain species in sediment samples can give a critical insight into the geological and environmental history of an area, providing invaluable data for climate researchers and industries alike.

1.2 Motivation

Foram classification is typically carried out by hand, under a microscope, by expert geologists. The reasoning behind this time-consuming method stems from the subtle differences between forams, and therefore the difficulty of accurately classifying them into their representative species. Due to a relatively low number of expert classifiers in the field, taxonomic schools regularly differ in their identifications around species boundaries. With use of a trainable machine learning model, both the speed and accuracy of the classification process could be improved dramatically. Moreover, the model has the possibility to further impact the geological community by acting as a pre-trained model for other geological classification problems.

The focus of this thesis is to design and evaluate a new supervised classification model using current state of the art deep-learning techniques suited to this particular computer vision task.

1.3 Contributions of the Thesis

In this thesis, we will investigate whether we can improve on the accuracy of a reference model through the implementation of various machine learning techniques. The contributions of this thesis can be summarised as follows:

- Literary review of several state-of-the-art CNN practices which have been developed to mitigate the negative effects of class imbalance and overfitting.
- Replication of the reference model to test and validate the results obtained in [1].
- Exploration of practices which reduce overfitting within pre-loaded models that are far removed from traditional image classification problems.
- Technical evaluation of several model configurations, with a detailed performance comparison of selected CNN practices [2, 3, 4].
- Conversion of state-of-the-art VGG16 architecture for usage as a pre-trained model within general geological taxonomy.
- Background on the foundations of machine learning and convolutional networks for the benefit of geologists interested in using and further developing the foram model.

Chapter 2

Background and Related Work

In the first part of this chapter, we will explore the theory behind machine learning and convolutional networks networks, specifically how each component of these networks works with respect to the training process. Following this, we provide a literature review of recent machine learning architectures and practices which specifically target computer vision orientated tasks. A main focus is directed towards methods that aim to lessen the negative effects of an imbalanced dataset, as well as methods that target datasets which are far removed from traditional computer vision problems. Finally, we introduce the reference model that we aim to use as a baseline for our foram model.

2.1 Machine Learning

Machine learning differs from conventional programming in that the final models created through machine learning are not explicitly programmed, but instead trained on vast amounts of data [5]. This paper will focus specifically on a branch of machine learning called supervised learning. Where a normal program is given a set of instructions along with input to then calculate an output, a supervised machine learning model is given inputs along with their corresponding outputs and uses the relationship between them to infer the instructions for itself [6]. Once the training period is complete, these inferred instructions can then be used on new data to predict new outputs. In general, the more inputs and outputs the model is exposed to within this training period, the more accurately it will perform on new data.

2.1.1 Loss functions

The way machine learning models 'infer' these instructions is by continuously searching for superior representations of the data in reference to a central loss function, which calculates how far away the model's current predictions are to the given expected output [5]. Different tasks require different loss functions. For multi-class, single label classifiers such as the reference model, it is traditional to use categorical crossentropy, defined as:

$$CE = -\sum_{i}^{C} t_i \log(s_i) \tag{2.1}$$

where t_i is the ground truth value and s_i is the CNN score for each class in C (i.e. 1 if the species is present, 0 if not) [5]. Each time the model is exposed to a set number of samples from the training data (commonly referred to as a batch), its parameters are adjusted slightly in an attempt to minimise this loss function, which effectively acts as a feedback signal.

2.1.2 Deep networks

Deep learning is a sub-field of machine learning which uses the idea of layering. In a standard deep neural network (DNN), the data flows through all layers sequentially. Each layer receives data from the layer immediately preceding it, then forms its own representation and passes that representation on to the next layer, where the first layer of the network receives the raw input data. The deeper the layer is, the higher the abstractions are that it can reach. A reference image is provided in Figure 2.1.

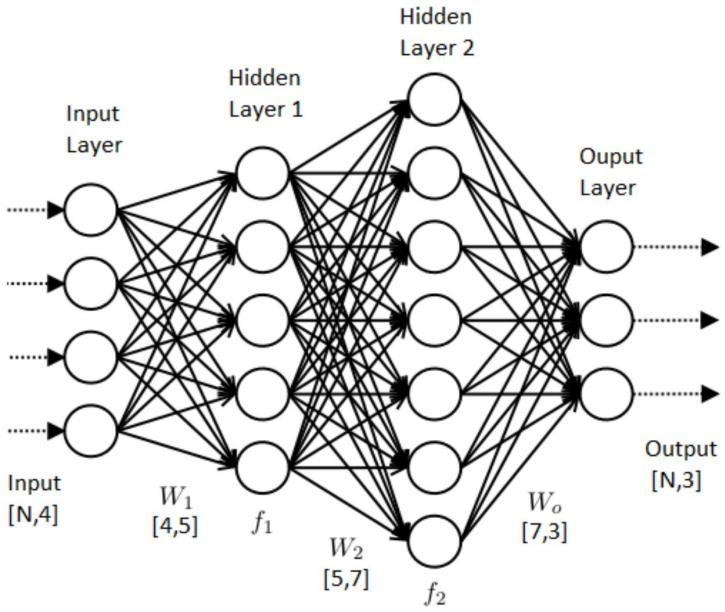


Figure 2.1: Within a deep neural network, the input data flows sequentially through each layer of the before finally reaching the final output layer. Image from [7]

A common misconception about deep neural networks is that they are modelled on how the human brain works. In reality, these networks are modelled on our knowledge of individual neurons [8]. Neurons in the brain receive signals from neighbouring neurons and only fire themselves if the sum of these signals surpasses a certain threshold value. Densely connected layers of a deep learning model consist of a large number of neurons, called nodes, which act in a similar manner.

The way nodes in a layer differ from neurons in the brain is by the inclusion of a discrete numerical weight vector $\mathbf{W}$, with each weight assigned to an input from a neuron in the preceding layer [5]. When a vector of inputs $\mathbf{X}$ are received by a node, the dot product of these inputs and their corresponding weights is calculated and a bias vector $\mathbf{B}$ is added. This result is then

fed through an activation function f, which acts as a threshold. This is demonstrated in the following equation:

$$output = f(\mathbf{W} \bullet \mathbf{X} + \mathbf{B}) \tag{2.2}$$

If the activation function is satisfied, the node sends its own signal to its connections in the next layer. Every node in a dense layer is connected to all nodes in the layers immediately before and after it. As an example, two dense layers with 1200 nodes each will share $(1200^2) = 144,000$ connections between them. Examples of common activation functions are shown in Figure 2.2.

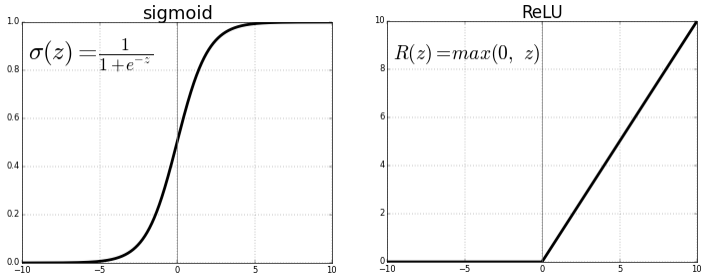


Figure 2.2: There are several activation functions which serve different purposes within a network. Sigmoid (pictured left) is commonly used in the final layer of a binary classification network to effectively squash outputs between 0 and 1, i.e. percentage certainty of the classification [5]. Softmax works in a similar way to Sigmoid for multi-class classification by ensuring all the resulting outputs sum to 1. The output of ReLU (pictured right) is directly proportional to the input as long as the result is greater than 0 [9]. Therefore, if the sum is less than 0, the node will not fire. Image from [10].

The bias values in neural networks act comparably to the y intercept in a linear equations [11]. To accurately model a slope of $y = mx + c$ you need both the gradient and the y intercept values. If we simplify machine learning to modeling a straight line, m would be the weight of the neuron and c would be the bias. There is one bias value per node and they are updated with the weights during backpropagation [5].

With use of the loss function, we can determine whether our model accuracy is improving over the training period by giving us a distance metric from the optimal result. The way networks 'learn' is by gradually changing the weights of the network to favour a lower loss value [5]. For a classification algorithm such as our foram model this can be seen as improving the accuracy of the classification, i.e. how many foram species are classified correctly.

2.1.3 Back-propagation

The back-propagation algorithm is central as to how we update the weights of the model during the training process. The output of a model is sensitive to every single weight within it, some more than others. Altering a single weight within a network will change the impact of all layers following it [5], and the relationship between the weight value and the resulting output is often not linear. Therefore the best way to approach training a network is by changing all weights at

once in small incremental steps which favour a lower loss value [12].

A given output from a layer is dependent on all outputs from layers before it. Back-propagation, through use of the chain rule, can estimate the net effect from single parameter changes by linking together the partial derivatives of all layers preceding it [12]. If we simplify a feed-forward network to the following equation:

$$f(x) = A(B(C(x))) \tag{2.3}$$

where x is the input, f(x) is the output, and A, B, C represent each layer of the network as an input/output function. We can calculate the net impact of each of these layers through finding the derivative of f(x):

$$f'(x) = f'(A) \cdot A'(B) \cdot B'(C) \cdot C'(x) \tag{2.4}$$

Furthermore, with use of the chain rule, we can work out the partial derivative with respect to a given layer, i.e. the derivative with respect to B is found as follows:

$$f'(B) = f'(A) \cdot A'(B) \tag{2.5}$$

The chain rule allows us identify how much each layer weight contributes to the output of that particular layer. If this method is used starting from the end of the network (as we can work out 'C(x) easily) the individual partial derivatives can be recursively linked together for every weight in the network, all the way back to the first layer. This is where the term 'back-propagation' is coined from. It is a form of dynamic programming, an optimisation method which aims to solve computationally expensive problems quicker through recursion and an ideal starting point. Using back-propagation, we can understand the impact of any weight on the final output of the network [5, 12], along with the direction to update it in order to reduce the loss value of the model.

2.1.4 Optimisation methods

Through the use of backpropagation, we now know the individual sensitivities of each weight in the network, i.e. their derivatives. Optimisers use this information to decide in which direction, and to what degree, to alter these weights in favour of a lower loss value. This process is called gradient descent [5].

To gain an intuition behind gradient descent, it is useful to simplify a network to two weight values. Thus the loss function can be visualised in three dimensions (shown in Figure 2.3) as a plane of peaks and valleys. We can then interpret the role of an optimiser as adjusting the weights of a network in order to reach the lowest point of this plane [5]. In standard gradient descent, individual weights are adjusted via the following equation:

$$\theta_{\mathrm{j}} = \theta_{\mathrm{j}} - \eta \cdot \frac{\delta C}{\delta \theta_{\mathrm{j}}} \tag{2.6}$$

Where θ_{j} is a given weight, η is the learning rate, and $\frac{\delta C}{\delta \theta_{\mathrm{j}}}$ is the gradient (change in loss function value δC over change in weight value $\delta \theta_{\mathrm{j}}$) computed through back-propagation [5].

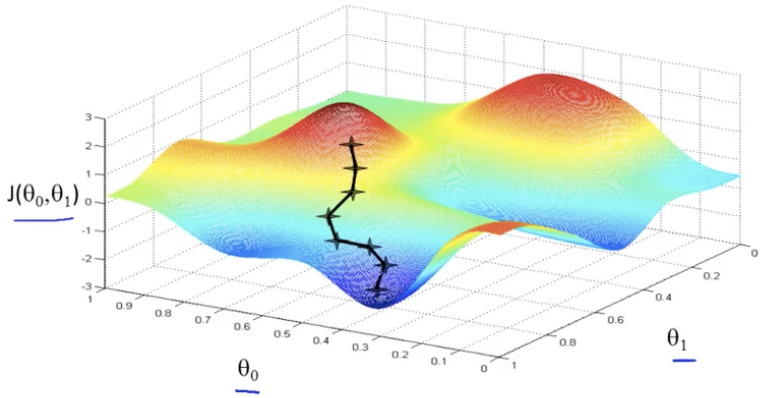


Figure 2.3: Simplifying a network to comprise of just 2 weights, θ_0 and θ_1, the loss value can be interpreted as $J(\theta_0, \theta_1)$. The job of the optimiser is to alter these weights in the right direction in order to minimise the loss value. Image from [13].

Different optimisers have their own benefits and shortcomings, and are used to solve different types of machine learning problems by adjusting the weights differently. The reference model [1] found ADAM [14] (name derived from adaptive moment estimation) to be the best suited optimiser for the foram classification problem.

2.1.5 Training and Overfitting

The model is trained through batches, which are subdivisions of the dataset. After each batch, the backpropagation algorithm is performed to gather the gradients of each weight and then uses the optimiser to decide how to adjust them to favour a lower loss value.

Before the training process begins, the data is subdivided into a training set and a validation set, typically with an 80:20 split respectively [15]. The way these two sets vary from each other lies in how they are used in the model's training process. Training sets are used as mentioned; during a training epoch, each batch is fed into the model to produce a loss value, which is then used to determine a change in the model's parameters through backpropagation and optimisation algorithms [5]. In contrast to the training set, the validation set is used immediately after each training epoch to test the model's accuracy on new, unseen data. During this validation pass the model does not alter it's parameters whatsoever. As such, the model learns nothing from the validation set, no matter how many times it is exposed to it. The reasoning behind this separation in the dataset is to compare the accuracy of the model on its training set compared to the validation set, and monitor when the model begins to over-fit. The validation data provides samples that the model hasn't trained specifically to solve, and therefore acts as a small-scale test of real-world performance.

Over-fitting occurs when the model performs considerably better at classifying training samples than validation samples [5, 4]. The training set is, by definition, a sample space of the general population of samples and therefore can never encapsulate all variations within a gen-

eral population. In terms of our Foram model, the training set consists of 22,000 samples, which is still minuscule compared to every single foram ever to exist. As previously stated, each epoch the weights of the network are updated to favour a lower training loss value. However, because this training set is a confined view of the actual population, the weights are updated to favour the classification of this restricted view of forams because this is the only data the model has been exposed to. In other words, over-fitting is a point in the learning process where the model has optimised too heavily on statistical noise unique to the training set, and thus looses its generalisability when exposed to new data [4], tested with use of the validation set.

In addition, where the model over-fits on the training data in an effort to improve the training accuracy of the model, humans over-fit on the validation data in an effort to improve the validation accuracy of the model. It is important to recognise that the validation set, like the training set, is also a sample space of the general population. Training a model for increased performance on the validation set might not mean increased performance when exposed to the general population. Therefore, the true accuracy of the model can only be assessed after being exposed to real-world data.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNN) are deep neural networks which are specially suited to image based problems. Where the dense layers mentioned above learn global patterns about the input data, convolutional layers learn localised patterns [16]. CNNs take an input image and apply a variety of filters to individual regions before combining the results to create more and more meaningful abstractions. The higher level abstractions towards the end of the network carry less visual information about the image, having substituted it for more valuable information relating to the class of the image. For instance, the first convolutional layer might be used to detect a variety of different straight lines in an image, whereas the final convolutional layers might detect the outline shape of a specific species of foram. A variety of filters are shown in Figure 2.4.

2.2.1 The Convolution Operation

Images are initially passed into a CNN as a 2D matrix of individual pixel values. If the image is of colour, as opposed to black and white, it is passed in as a 3D matrix with the 3rd dimension representing the red, green and blue channels of the image. Convolutional layer filters are 2D matrices, which have a typical size of either 3x3 or 5x5. The weights of a convolutional layer correspond to the values of its filters.

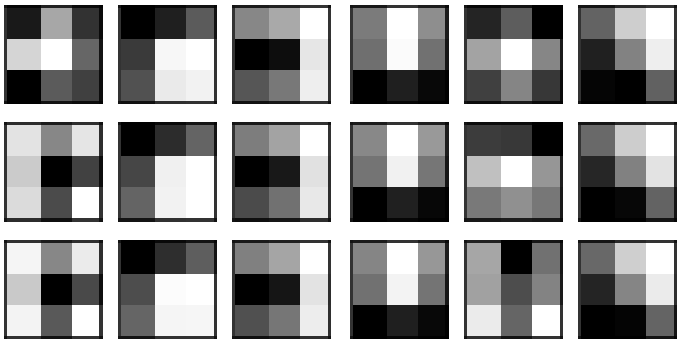


Figure 2.4: A variety of filters taken from the fifth layer of the foram model. White squares correspond to a positive, excitatory weight (i.e. 1, 2). Black squares correspond to a negative, inhibitory weight (i.e. -1, -2). These weights are changed during the training period to more accurately detect features that are directly related to the task.

Through the use of a sliding window, the algorithm passes a set number of different filters over all available regions of an input, and produces an output for each filter used. For each of these regions the convolution operation first performs element wise matrix multiplication between the filter and the region, then sums the result to produce a single output value for that region [5]. This operation is visualised in Figure 2.5. Through this method, the output image is 2 pixels shorter than the input image. By adding a padding of 1 pixel as a border around the image before each convolution, the input image dimensions are preserved. In addition, a user can specify how the sliding window passes over the input image by changing the stride parameter of the convolution layer. A stride of 1 will move along 1 space over at a time, whereas a stride of 2 will move 2 spaces.

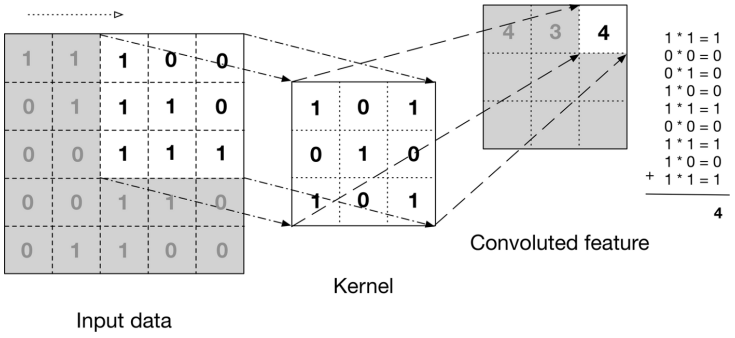


Figure 2.5: The convolution operation performs element wise matrix multiplication between the filter and a specific region of the input, then sums the resulting matrix to a single result. Image from [17]

After the initial input image is fed into the CNN, the output of every layer following will no longer be an image, but instead a feature map. This is a 3x3 matrix, with sides equal to the layer input width & height (if padding is used) and a depth equal to the amount of filters passed

over the image (commonly referred to as the channel dimension). If a depth-wise cross-section was taken of the output, we would see an individual feature map for a single filter.

2.2.2 The Max-pooling Operation

CNNs also make use of max-pooling layers, which simply output a single highest value from a specified region [5, 18]. Max-pooling layers appear directly after blocks of convolution layers, and thus receive feature maps as input. By selecting the highest value of these regional feature maps, which correspond to the feature being actively present in that area of the sample image, we can feed forward the most relevant information for later layers to work with. As with convolution, a sliding door mechanism is used to move the max-pooling operation over an image. However, as opposed to convolutional layers which typically use 3x3 windows and a stride of 1, max-pooling layers use 2x2 windows, and a stride of 2 [5, 18]. In contrast to convolution operations, max-pooling is a fixed operation which has no associated weights to train [5].

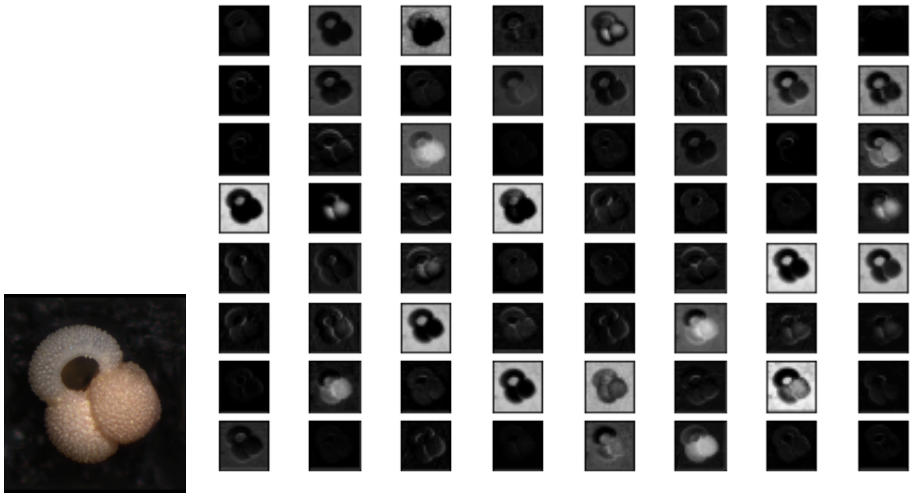


Figure 2.6: Individual feature map outputs from the second layer of the foram model. Layer 2 returns 64 feature maps. At this point in the convolutional section of the model, many feature maps closely resemble the input image.

A max-pooling layer essentially halves the size of an input, an essential operation which performs two separate roles within a given CNN. Most importantly, max-pooling layers are used to combine local features into more general, high level patterns [18]. As the convolution filter sizes remain the same throughout the network, down-sampling an image will cause neighbouring regions to merge, forming the feature hierarchy which makes higher level abstraction possible. Different layer outputs can be seen in Figures 2.6 and 2.7. This also ensures that the feature extraction is translation invariant, meaning the same feature can be recognised in any area of the image [18]. In addition to this, max-pooling layers vastly decrease the amount of parameters needed to describe an image as it passes through the network. The larger a network is, the longer it takes to train, and the more memory is needed to store it [5].

Traditionally, dense layers learn to use similarities between inputs to map to corresponding outputs. Therefore, the aim of the convolutional section of the network is to output similar

feature maps for samples of the same species. The dense layers then map similar feature maps to the correct corresponding class.

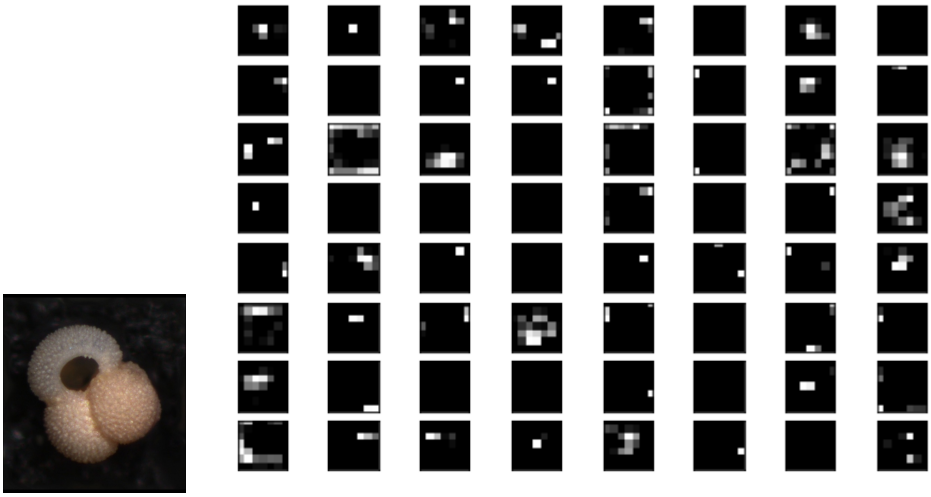


Figure 2.7: Individual feature map outputs from the 18th layer of the foram model. Layer 18 returns 512 feature maps, of which 64 can be seen in Figure (b). In contrast to the second layer, these feature maps bare no relation to the input image, having traded visual information for more useful class information.

2.3 VGG16

In 2014, the Visual Geometry Group from the University of Oxford submitted a classification model to the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [19, 20]. This challenge acts as a benchmark for CNNs by testing them on an intense classification task involving hundreds of classes and millions of images. The submitted model was called VGG [19] and used a completely new type of network architecture compared to the rest of the submissions. At the time, it was common to use larger 11x11 or 7x7 filters in convolution layers, as they are able to detect more complex patterns than 3x3 filters, such as corners and curves. In fact, 3x3 is the smallest available size to accurately detect simple up/down, left/right features within an input. the VGG team decreased the size of the filter to 3x3, which greatly reduced the amount of parameters in the model. This in turn allowed them to expand the network's depth and test a variety of different CNN configurations, from 11 layers to 19. The VGG16 model architecture is shown in Figure 2.8.

Two separate VGG models secured both first and second place in the Classification/localization ILSVRC 2014 [19], with a top-5 validation error of 6.8%. In addition, they found that VGG models with more layers gave higher validation accuracies overall. These results show that larger filters, whilst capturing more information in a single layer, are less accurate overall at image classification than the feature hierarchies formed through the layering of smaller 3x3 filters.

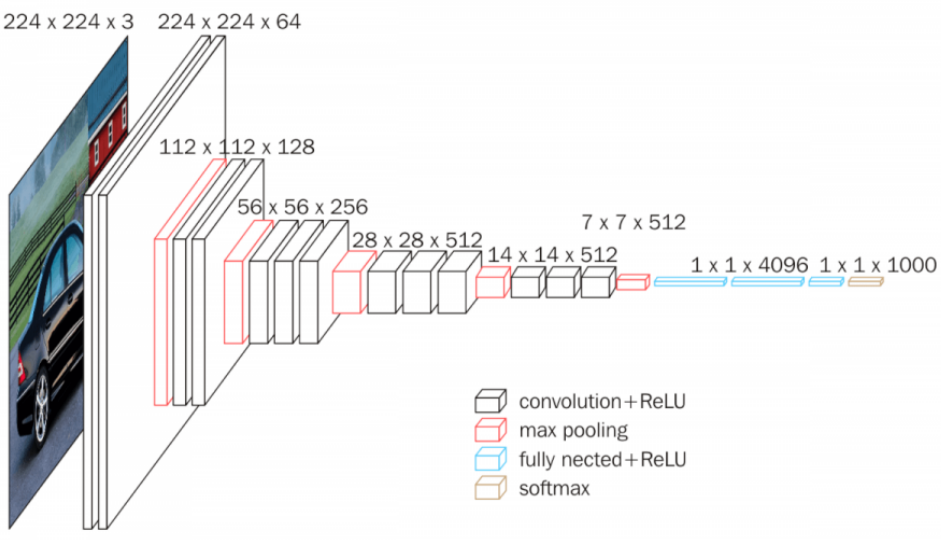


Figure 2.8: VGG16 comprises of 5 separate blocks of convolutional layers, linked together by max-pooling layers. Following this, the output is sent through a configurable number of dense layers before reaching the final output layer. Image from [21].

2.4 Transfer Learning

When a new model architecture is first introduced to the public, it's ImageNet classification accuracy is one way which it can be assessed as an accurate baseline against other models [20]. It is very common for new models to be trained extensively on the ImageNet library in order to maximise their potential accuracy and many machine learning frameworks (such as Keras) provide an option to initialise the model parameters to exactly mirror those developed over the intense ImageNet training period [5]. This process of using a pre-trained model for separate (but still related) tasks is called transfer learning [22].

Immediately before the training process, model weights are initialised randomly, which is a statistically better starting state for gradient descent than initialising all weights as one number. Traditionally, each new computer vision problem was trained independently on its own dataset [22]. The introduction of large image datasets with hundreds of classes, such as ImageNet, has allowed models to gain a basic understanding of the visual world. By initialising a new model with these parameters, researchers can take advantage of this fine-tuning for their own models instead of starting from scratch.

The main advantage of transfer learning comes from the acceleration of the training process [23]. Because the model already has a foothold on a very general computer vision task, it reaches its peak accuracy quicker than a model that is trained with randomly initialised weights. Whether the accuracy is improved overall is dependent on the size of the dataset and the relevance of the transferred model to the new task [23].

2.4.1 Layer Freezing

There are several ways to use transfer learning within a given model, which involve different levels of layer freezing. Freezing a layer of a model means its weights are not updated during the training process [5]. Through this method, when an image is fed into the model during training the weights of the frozen convolution layers remain unchanged, meaning the same image will produce the same output feature map every time. Later layers in a CNN form specific abstractions of the data relating to the task, whereas earlier layers learn more general patterns [5]. Because of this, the earlier layers of a target model are frozen to lock in the general low-level patterns learned by the transferred model. The fundamental problem of exactly where transferable models switch from general features to specific was discussed by [24]. Through testing a variety of different freezing configurations, researchers demonstrated a method to pinpoint where this switch occurs within a transferred network.

In cases where the task is completely different to the transferred model (i.e. medical imaging) transfer learning is found to produce no real performance increases [25]. The reasoning behind this is if the target task is too specialised in relation to the transfer model, they are so far removed that there are no general visual patterns shared between them.

Transfer learning applies directly to the future of the Foram classifier, as we can use the model itself for transfer learning in different areas of geological taxonomy that have not yet been introduced to machine learning practices.

2.5 Data Augmentation

A major downside of using convolutional layers is that, although they are translation invariant, they are also very sensitive to slight changes in rotation and scaling [5, 4]. A minor change to an input image through scale or rotation can result in a completely different output feature map, which results in an incorrect prediction from the following dense layers.

One way we can combat this problem is by using on-the-fly data augmentation, which works by altering the sample images through random rotations, scaling, etc. as they are being fed through the model [4]. This is in contrast to the pre-processing style of data augmentation, which alters the images before the training process begins to increase the size of the dataset [5]. With use of on-the-fly data augmentation, instead of the model training constantly on the same set of images it is now trained on a slightly new set of samples each epoch. Although this reduces the model's accuracy on the training data (because every training image is now slightly different), it also vastly reduces the effect of overfitting when exposed to new data, which can result in a higher overall validation accuracy.

As previously stated, overfitting is the direct result of the model over-learning statistical noise within the training set. The reason why the validation accuracy improves using data augmentation is because the model now generalises to include a larger class spectrum rather than over-training on a single orientation [4]. A variety of foram augmentations are shown in Figure 2.9.

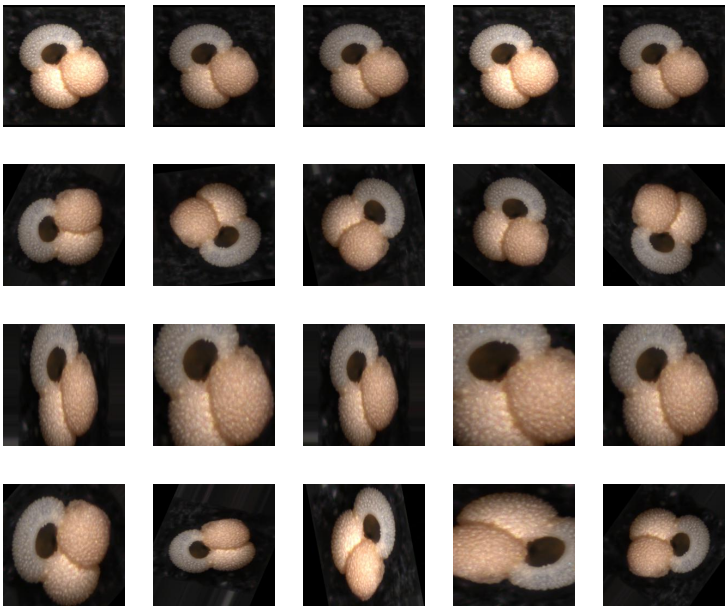


Figure 2.9: A variety of data augmentations performed on a foram sample. By row - brightness range [0.2, 1.6], rotation range 360, zoom range 0.5, all three.

2.6 Focal Loss

RetinaNet [2] attempts to lessen the issues caused by class under-representation through the introduction of a new loss function.

Class imbalance during training is the biggest obstacle impeding the speed and accuracy of single stage detectors. Examples which are easily classified dominate the gradient (i.e. which way and to what degree the weights are being adjusted) [2]. This leads to less overall influence of the harder, less available examples. However, a new loss function used in RetinaNet reduces this imbalance through prioritisation of hard examples over easy ones [2].

In statistics, outliers (hard examples) are commonly down-weighed, reflecting their difference to the mean of the class. Conversely, the loss function used by RetinaNet works by dynamically scaling the loss for a model to favour harder examples. A scaling factor is introduced to the backpropagation weighting, which decays to 0 as confidence in the correct class increases. This allows for harder classifications to have more of an effect on the re-weighting of the model, whilst still allowing easier examples to contribute to a lesser scale.

Another way to define the categorical cross-entropy loss function, as opposed to equation 2.1, is as follows:

$$CE(p, y) = -log(p_t) \tag{2.7}$$

Where

$$p_t = \begin{cases} \text{p}, & \text{if } y = 1; \\ \text{1 - p} & \text{otherwise} \end{cases} \tag{2.8}$$

y is the ground truth class, and p is the model's current probability of a correct classification for that given class. RetinaNet's focal loss function builds on this definition by introducing a modulating factor. The following equation summarises the focal loss function:

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma} log(p_t) \tag{2.9}$$

Where $-\alpha_t(1-p_t)^{\gamma}$ is the modulating factor, γ is a focusing parameter, and α is a balancing factor [2]. By observing this equation, we can see that for easily miss-classified examples, where pt is close to 0, the modulating factor stays close to 1 and the loss remains unaffected. However, for samples that are more easily classified, where pt is close to 1, the modulating factor tends towards 0 and the loss value is down-weighted. A visualisation of how different γ values affect the loss value is shown in Figure 2.10.

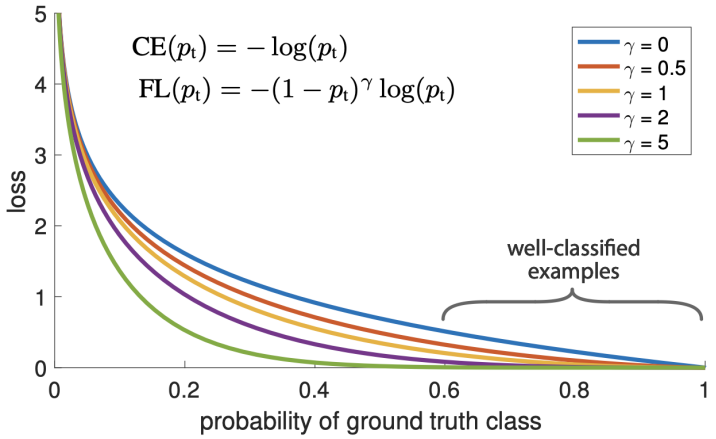


Figure 2.10: As γ is increased from 0 (which simplifies the loss function to categorical crossentropy) to 5, we see that well classified examples (where $p_t > 0.5$) contribute less to the overall loss value. Image from [2].

With use of its loss function, RetinaNet matches the current state of the art detectors when tested on the challenging COCO AP [**4**, 2]. Incorporating the focal loss function into our model will down-weight well classified examples, allowing harder examples to have a greater effect on the re-weighting of the foram model.

2.7 CBAM - Convolution Block Attention Module

Attention modules are used to direct a feed-forward CNN towards the most important areas and features within an input image [26, 3]. They have been successfully used for a variety of different machine learning tasks, from computer vision [27, 28] to natural language processing [29]. Within our foram model, important information can be seen as features which are directly related to the classification of a certain species. With use of attention modules, we can focus the model to train more heavily on these important features (i.e. the shape of the foram), whilst suppressing areas of the samples which yield less important information (i.e. image noise).

"Since convolution operations extract informative features by blending cross-channel and spatial information together, we adopt our module to emphasize meaningful features along those two principal dimensions: channel and spatial axes. To achieve this, we sequentially apply channel and spatial attention modules so that each of the branches can learn 'what' and 'where' to attend in the channel and spatial axes respectively. As a result, our module efficiently helps the information flow within the network by learning which information to emphasize or suppress." - [3].

within CBAMs, separate channel and spacial wise attention modules are implemented sequentially and then combined with the input feature map to produce a feature refined output. A diagram of the overall CBAM architecture is shown in Figure 2.11, and an outline of both sub-modules is given below.

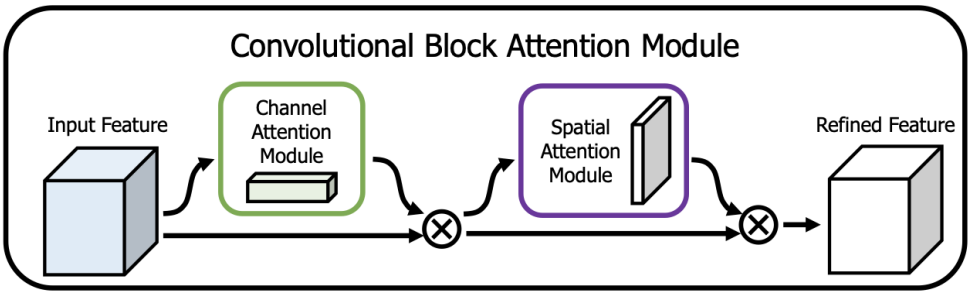


Figure 2.11: An overview of the CBAM architecture. Channel and spacial wise attention modules are implemented sequentially and then combined with the input to produce a feature refined output. Image from [3].

2.7.1 Channel Attention Modules

Channel wise attention modules focus on 'what' is important within a given image by gathering the most useful information from the channel dimension of an input. As previously stated, different channels of a feature map are a result of different filters being passed over an input. The job of this module is to find which of these filters is most valuable to the classification process and amplify them, whist suppressing filters which have little impact on the classifier [3].

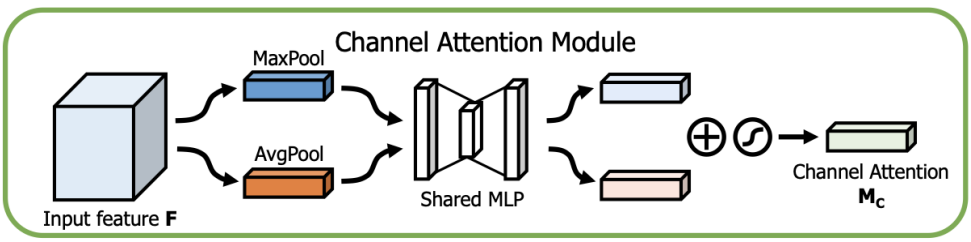


Figure 2.12: The channel attention module aggressively down-samples the spacial dimensions of the image through max and average pooling before feeding the results through a small dense network. Image from [3].

Taking an intermediate 3D feature map with dimensions (Height, Width, Channels), the channel wise attention module will first aggressively down-sample the input image through max-pooling and average-pooling to squash the spacial dimensions, resulting in two (1,1,Channel) feature maps consisting of just the average and maximum channel values [3].

Contrary to max-pooling, which condenses values to their maximum, average-pooling instead condenses values to their average. Both of these pooling methods are used as they result in different representations of the input (referred to as context descriptors), and thus gather more information for the module to work with [5].

The two pooling outputs are then separately fed through a small dense network with one hidden layer before being merged together through addition and fed through a sigmoid activation function. This hidden layer is the only part of the sub-module which is trainable, and will learn to better recognise important filters within the (1,1,Channel) feature map as the training process takes place [3]. A diagram of this sub-module is shown in Figure 2.12.

2.7.2 Spacial Attention Modules

As opposed to channel wise attention modules, spacial wise attention modules focus on 'where' important information is within a given image. Instead of finding which features are important, a spacial attention module is concerned with finding which regions of the image contain a higher amount of these features [3].

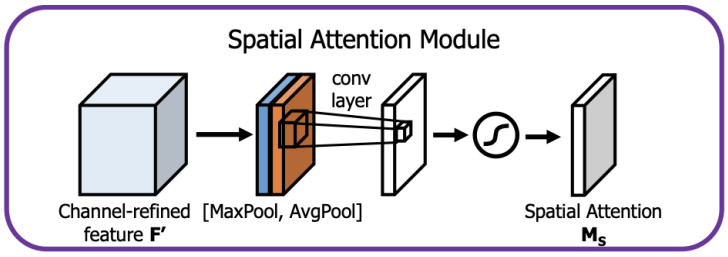


Figure 2.13: The spacial attention module aggressively down-samples the channel dimensions of the image through max and average pooling before feeding the result through a convolutional layer. Image from [3].

Max-pooling and average-pooling are now applied across the channel dimension, resulting in two feature maps with dimensions (Height, Width, 1). These are then concatenated, forming a single feature map with dimensions (Height, Width, 2). Following this, the concatenated feature map is fed through a standard convolutional layer and a sigmoid activation layer, producing an output feature map with dimensions (Height, Width, 1) [3].

As with the channel attention module, the spacial module can be trained with use of the convolution layer to better understand regions of the image which are important to classification. A diagram of this sub-module is shown in Figure 2.13.

2.7.3 Element-wise Multiplication

As previously stated, the structure of a CBAM is sequential. An input first flows through the attention sub-modules and then is combined with the original input using element wise multiplication. Through element-wise multiplication (denoted as $\bigotimes$ in Figure 2.11), both modules can broadcast their values across the other sub-module's dimensions, incorporating both the learned channel and spacial information into a final output with dimensions (Height, Width, Channels), identical to the CBAM input [3].

Due to its low amount of parameters, CBAM is a lightweight component which can easily be incorporated several times within a model at low computational cost [3]. This technique can be used in our foram model to focus its learning on features and areas of the image which are more important to the classification process.

2.8 Endless Forams

The foram model associated with this thesis builds upon the work conducted in [1]. This model used a VGG16 architecture and was trained using two separate datasets: a North Atlantic coretop collection from the Yale Peabody Museum (hereafter, YPM Coretop Collection) and the Henry A. Buckley collection from the Natural History Museum, London (hereafter, Buckley Collection) [1]. Both of these datasets were created through rigorous species identification by at least four certified taxonomists. Samples were only included in the final dataset if a 75% agreement rate was met between the human classifiers.

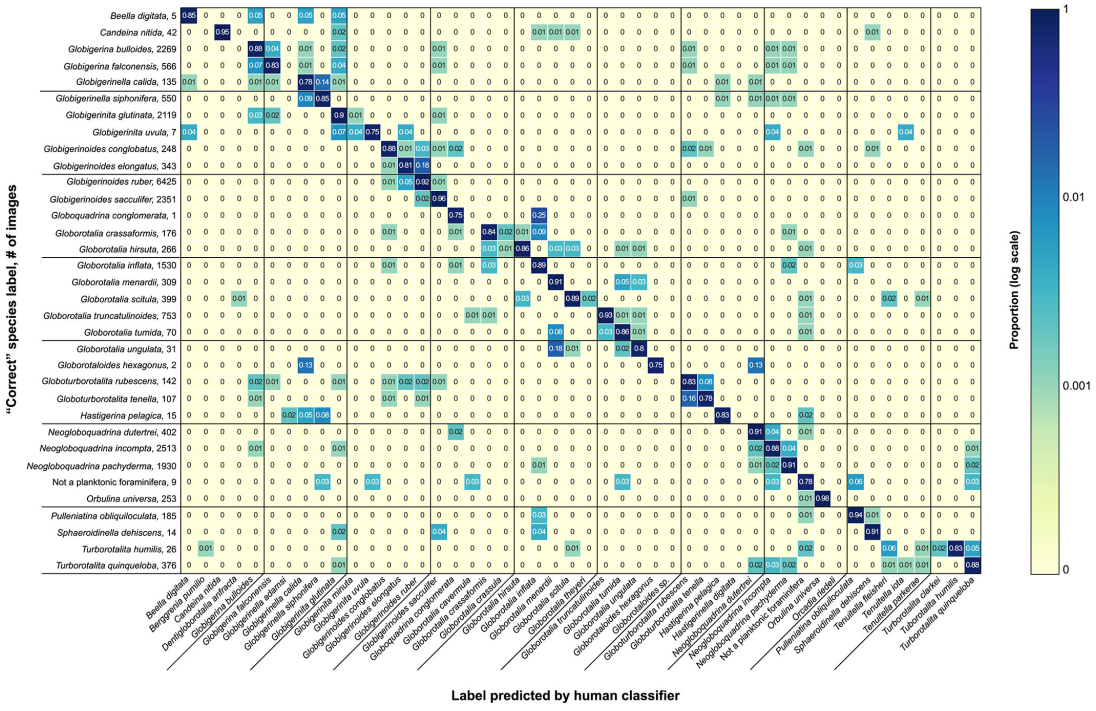


Figure 2.14: The human confusion matrix for species classification from the YPM coretop dataset. The y-axis is the correct species identifications, and the x-axis is the list of all species the samples were identified as. Image from [1]

Through testing a variety of fine tuning techniques, the model reached a maximum validation accuracy of 87% when trained with an 80% training, 20% validation split. This accuracy was achieved with the following hyper-parameters: Image size 160x160, batch size 100, 7 frozen layers, a dropout layer with a value of 0.5, a self-regulating learning rate with an adjustment factor of 0.5, and no data augmentation.

2.9 Summary

In summary, we provided an in-depth overview of the foundations of machine learning and convolutional neural networks. We then explored a variety of recent machine learning techniques aimed at improving the performance of classifiers trained on imbalanced datasets, which we now hope to incorporate into the reference model in order to improve the classification performance. Finally, we introduced the reference model architecture and configuration.

Chapter 3

Foram Classifier Implementation and Overview

The following chapter provides a full overview of the foram model implementation. As previously stated, the foram classification problem is far removed from traditional computer vision based deep-learning problems, as the images exhibited within the dataset bare little resemblance to images taken of the real world. As such, many machine learning practices that work well for other datasets might have little affect on ours. A variety of different techniques were tested in order to address the weaknesses which are specific to this unique dataset.

In the first part of this chapter, we make use of a variety of data pre-processing techniques in order to achieve the best results out of our dataset. We then explore the primitive versions of the classifier which were created to understand the fundamentals of machine learning more deeply. Following this, we implement data augmentation, focal loss and CBAM within the foram model in order to observe their individual effects on the performance of the classifier.

3.1 Data Pre-processing

3.1.1 Removing Annotations

Samples from the raw dataset used by the reference model [1] initially included annotations summarising where they were classified, the date of classification and their relative lengths. A sample from the raw dataset is included in Figure 3.1. Although annotated images are important for taxonomists, they cannot be used within the training process as they would severely affect the real world accuracy of the model.

Convolutional layers of a CNN are trained to detect features in an input image. By including the annotations within the training samples, the CNN would train not only on the features within the image, but also the features within the annotations. Convolutional layers cannot read, however they can recognise text as a pattern. If similar text is included in another sample of the same class, this is what the model will train to recognise, defeating the generalisability of the model when exposed to non-annotated images.

To counter this problem, a small OpenCV python program was created to automatically crop labels out of samples. Using the fact that the annotation backgrounds for the whole dataset

were uniformly white, the program checks each pixel vertically down the side of the image, and crops the image as soon as the pixel colour is solid white. In addition, the program also cuts the solid black boarders off of the cropped image by using the same method in the horizontal dimension. The resulting image can be seen in Figure 3.1.

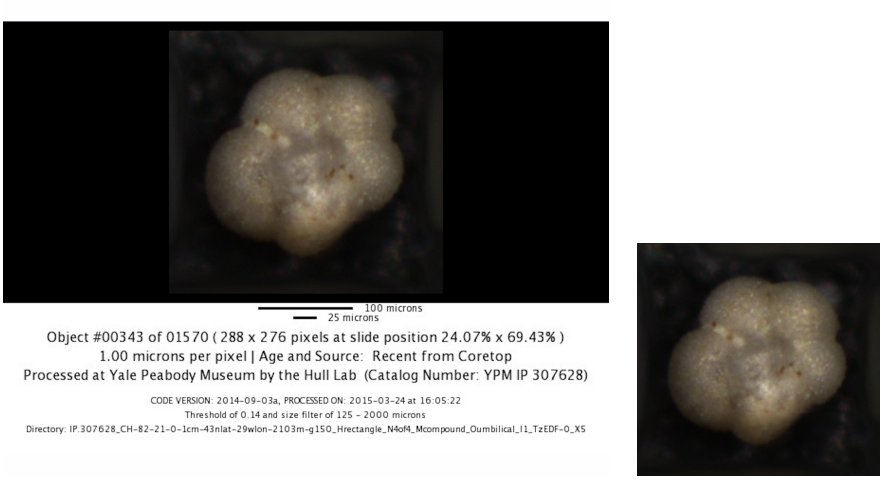


Figure 3.1: Python crop program performed on a sample from the foram dataset. The image on the left shows the annotations included by taxonomists following the classification process. The image on the right shows the result from the crop program.

3.1.2 Shuffling the dataset

In addition to the samples being annotated, it also became clear that they followed a loose ordering within the dataset. Similarly coloured forams can be seen clustering together within species folders, which is a major problem for the validity of the classifier [5].

When a dataset is subdivided into training and validation sets, both must be representative of the full dataset, meaning that they include the same proportion of all variants of a class from the full dataset. Ideally, if there are 10 pink forams within a species sample space, when the dataset is separated (with a 20:80 split) 2 pink forams should appear in the validation set and 8 should appear in the training set [5]. Without this balance, the model will either over-fit or under-fit on a certain variety of forams within the training process, which can hinder the maximum validation accuracy of the model.

To fix this problem, the dataset was shuffled with another small python program exactly once. The resulting dataset was used for training all future iterations of the foram model. The reason behind a single dataset shuffle is to keep consistency between models. If the dataset was shuffled before every training process, the results between different models would no longer be comparable as they were trained on different splits of the original dataset; an increase in validation accuracy could signify either a stronger model or a favorable shuffle.

3.2 Model Training

The largest bottleneck associated with this thesis has been the training time of the model. On average, the training period of the most complex models on Google Colab has been around 5-6 hours, even with GPU acceleration. A small code error may not be spotted until far into training process, which renders the resulting model useless. This problem is exaggerated when the final results can only be verified through cross-validation, which requires multiple training repetitions. In order to combat this, a copy of the classifier was created so two separate configurations of the same model could train in parallel using Google Colab.

3.3 The Vanilla Classifier

3.3.1 Binary

Before building a full-scale multi-class classifier, a simple binary classifier was designed to be trained on any two species in the dataset out of a possible 35. This was achieved by copying selected species into a temporary directory from a google drive repository before each training process takes place.

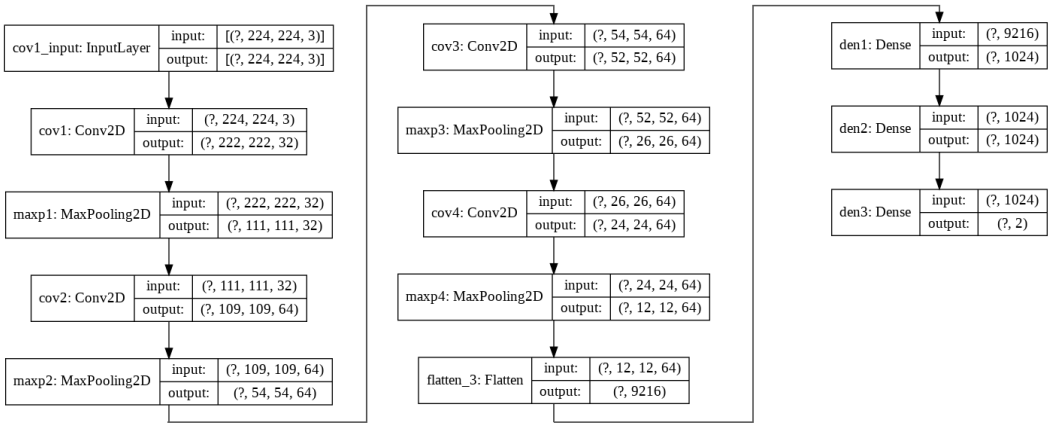


Figure 3.2: The Binary vanilla classifier consisted of alternating Convolutional and MaxPooling layers, followed by several dense layers, leading to a final sigmoid activation function in the output layer.

The structure of the binary model is sequential, with 4 layers of alternating Convolutional and MaxPooling layers. These layers are then flattened and fed through 2 dense layers before finally passing through a sigmoid activation function to give a binary classification output. As this is a binary classifier, the loss function used by this model is binary crossentropy. Following the reference model, an 80% train and 20% validation split was used during the training process. A full summary of the model architecture can be seen in Figure 3.2. After 80 epochs, the binary classifier reached a top validation accuracy of 91% (see Table 4.1) when training on the two most abundant species in the dataset, Globigerinoides Ruber (6425 samples) and Neogloboquadrina Incompta (2513 samples).

3.3.2 Categorical

After the implementation of the binary classifier, a similar model architecture was implemented as a multi-class classifier for any selection of the 35 foram species. This was realised by replacing the sigmoid activation function of the output layer to softmax and the loss function from binary crossentropy to categorical crossentropy. In addition, the optimiser was switched from SGD to Adam (adaptive movement estimation). This multi-class classifier achieved a top validation accuracy of 61% (see Table 4.1), 26% lower than the reference model [1]. In contrast, the maximum training accuracy reached by this model was 90%, which shows it suffers from a large overfitting problem. The full training process is shown in Figure 4.1a.

3.4 Reference Model Implementation

The reference model [1] associated with this paper provides a solid baseline model for our foram classifier to be built upon. A diagram of the network architecture can be seen in Figure 3.3. In order to gain a complete understanding of this model, an identical model was implemented from scratch using the same hyper-parameters. It is important to accurately recreate the reference model to iron out any inconsistencies before further advancements are made, otherwise the reasoning behind differing accuracies between models might be misinterpreted during evaluation.

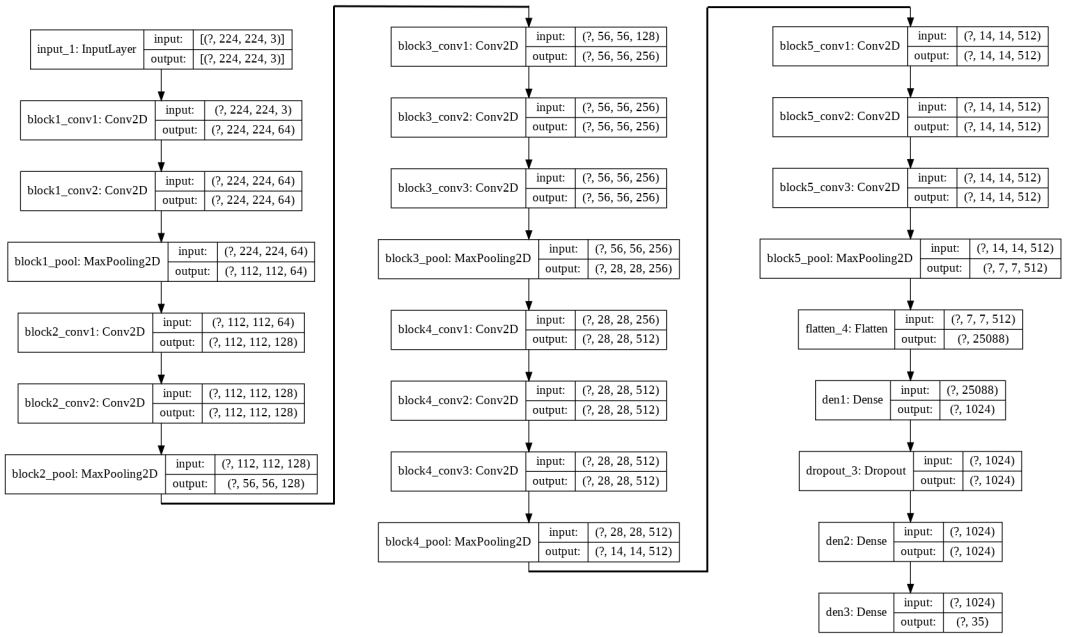


Figure 3.3: Architecture of the reference model. In contrast to the VGG16 architecture seen in 2.8, which uses 4096 dense layer units, the reference model uses 1024. In addition, a dropout layer is introduced between the dense layers, with a value of 0.5.

Through this method, the recreated model achieved a maximum validation accuracy of 0.85, similar to the reference model, which achieved 0.87. This difference in accuracy is minimal

when comparing these two models, and could stem from the initial crop and shuffle of the dataset before training.

3.5 Class Imbalance and Under-Representation

The dataset used by the reference model specifies 35 individual species classes out of a possible 40 known species. The most abundant class, Globigerinoides Ruber, boasts 5914 sample images. In contrast, the least abundant class, Globigerinella Adamsi, contains only 4 samples. This class imbalance is an obvious problem in regards to training the model. Full class sample numbers are visualised in figure 3.4.

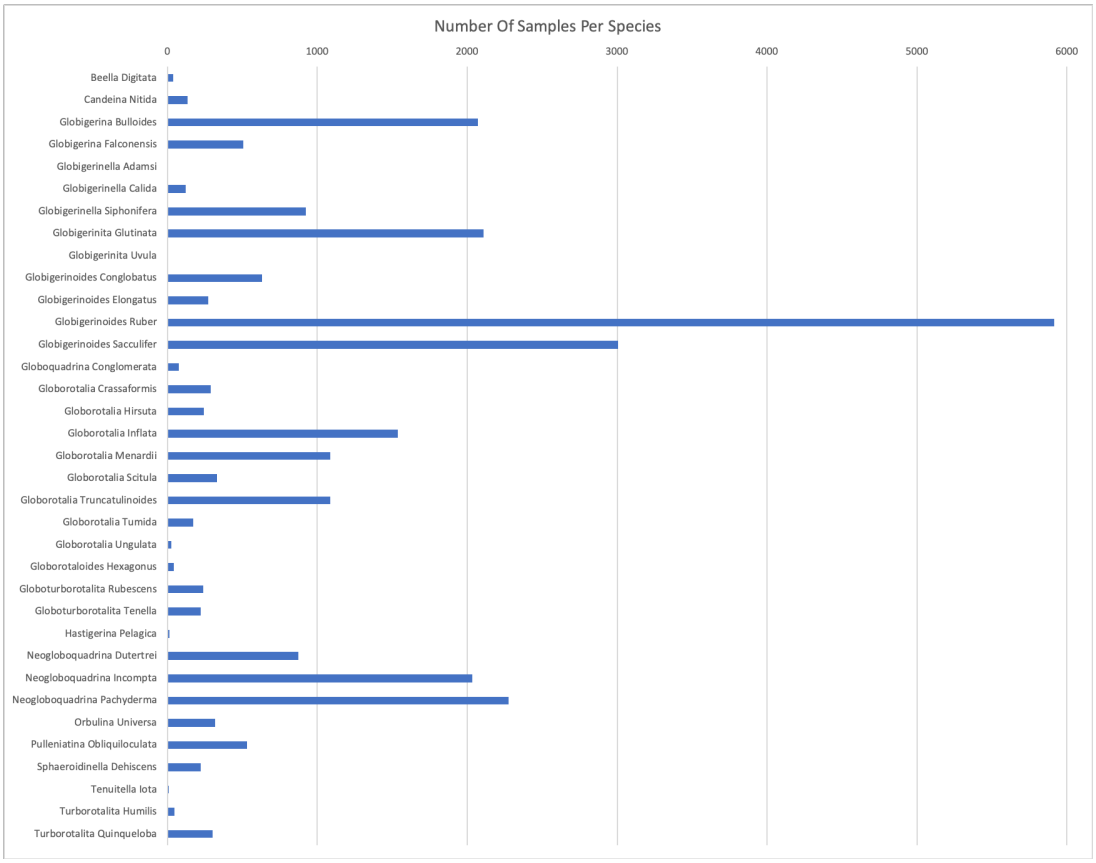


Figure 3.4: Sample numbers range from 5914 to 4, which signifies a large class imbalance. This imbalance leads to a classifier which over-trains on features related to the more abundant species, whist neglecting features associated with less abundant species [5].

Class under-representation is a key issue in machine learning. Because the model alters its parameters to favour a lower loss function value, classes which have the most samples will have a much larger impact on the reweighing of these parameters [30]. Put simply, the less data the algorithm is exposed to of a specific class, the worse it will perform when introduced to new examples of that class.

This issue is commonly overlooked when dealing with large inter-class size differences. As

an example, for a given dataset which contains 90% of species 1, 5% of species 2 and 5% of species 3, the classifier could easily classify every incoming sample as species 1 and still achieve 90% accuracy. This accuracy, although high, yields no tangible insight into the machine learning problem we are trying to solve. This is the largest problem we face when building this particular classifier. To add to this problem, the species which are often disagreed upon when classified by taxonomic experts are precisely the species which are least represented in the dataset. These "fringe" species [1] are much less common than the abundant ones, which is reflected in the number of samples.

3.6 Data Augmentation

Data augmentation improves the model's generalisability towards samples which exhibit the same variety introduced by the performed augmentations [4, 5]. It is especially useful for models with limited amounts of training data, such as our foram dataset, which has a mean class average of 792 samples per species. Although these samples are relatively uniformly orientated, some of them exhibit a wide range of both rotational and scalar differences. In addition to these positional differences, the brightness and contrast levels also differ greatly between samples. This variety between samples can best be seen in Figure 1.1.

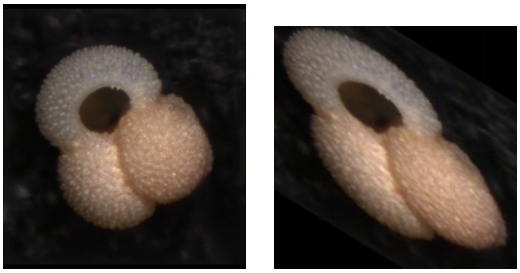


Figure 3.5: Augmenting with use of shearing affects the overall shape of an image. The left image shows an unmodified sample. The right image shows a sample subject to shear augmentation. In terms of foram classification it is critical to keep the shape of the original image, so this is not a viable augmentation.

The larger the differences between input images, the more contrast will be seen between their feature maps as outputs from convolutional layers. A model will train to recognise whatever feature map is most common for a particular class. By incorporating these differences within our augmentations, the model will be exposed to a larger variety of feature maps and can better generalise to include them.

The reference model [1] found little benefit to using data augmentations within the training process and therefore chose not to implement augmentations within their final model. Tested augmentations included: a rotation range of 20 degrees, a width and height shift of 0.2, a shear range of 0.2, a zoom range of 0.2 and horizontal flipping. As seen in Figure 1.1, the foram samples exhibit a variety of rotations larger than 20 degrees. In the new model, a rotation range of 360 was chosen to better reflect the variety of the dataset, along with a scale range of 0.2 and

a brightness range of [0.5, 1.5]. The geology department at Bristol University advised against using shearing within the augmentations, which slants the shape of the image (shown in Figure 3.5), as the shape of the foram is directly related to the species. both height and width shift had no affect on validation accuracy, as the convolution operation is translation invariant. Individual accuracies for each type of augmentation are shown in Table 4.1. By incorporating brightness into the augmentation process, the network was able to achieve a top validation accuracy of 0.9050 (see Table 4.1).

3.7 Focal Loss

The focal loss function down-weights the the influence of easily classified samples through use of a modulating factor. It was integrated into the foram model by adapting the code used by [31], replacing the categorical cross-entropy loss function of the model with the referenced custom focal loss function. A γ value of 2 was used, as proposed by the paper [2], along with several α values ranging from 0.25 to 1.

With use of classification reports adapted from scikit-learn [32], we can discern the classification accuracies of individual species. This will give us an in-depth overview of the effects of focal loss within our foram model.

3.8 CBAM

Both spacial attention and channel attention modules were integrated into the model in the form of CBAM [3], following the implementation shown in [33]. Unfortunately, this implementation does not currently support the VGG16 architecture. Therefore the module was adapted from this repository to suit our model.

In order to carry the advantages of transfer learning (shorter training periods as a result of more favorable starting weights), the foram model is generated through use of the Keras applications API, which has the option of loading model weights from ImageNet. To include CBAM within the convolutional layers of the network whilst still maintaining the ImageNet weighting, VGG16 was implemented from scratch by chaining individual layers of the pretrained model together and attaching CBAMs between them.

The ideal location of a given CBAM is within the convolutional blocks of the chosen CNN, just before the layers which down-sample the feature maps [3]. CBAM was included before the bottlenecks of the foram network, i.e. maxpool layers, to reflect this.

Two separate configurations of CBAM were tested. The first involved several CBAMs at the end of all 5 convolutional blocks. Using this configuration, after training for 20 epochs, the model reached a top validation accuracy of 23% and a training accuracy of 21%. The second involved a single CBAM in the fifth convolutional block of the model. Through the incorporation of a single CBAM module within the architecture, the top validation accuracy of the model improved to 90.9%, the highest accuracy seen so far (see Table 4.1). A cropped architecture of the current model is shown in Figure 3.6.

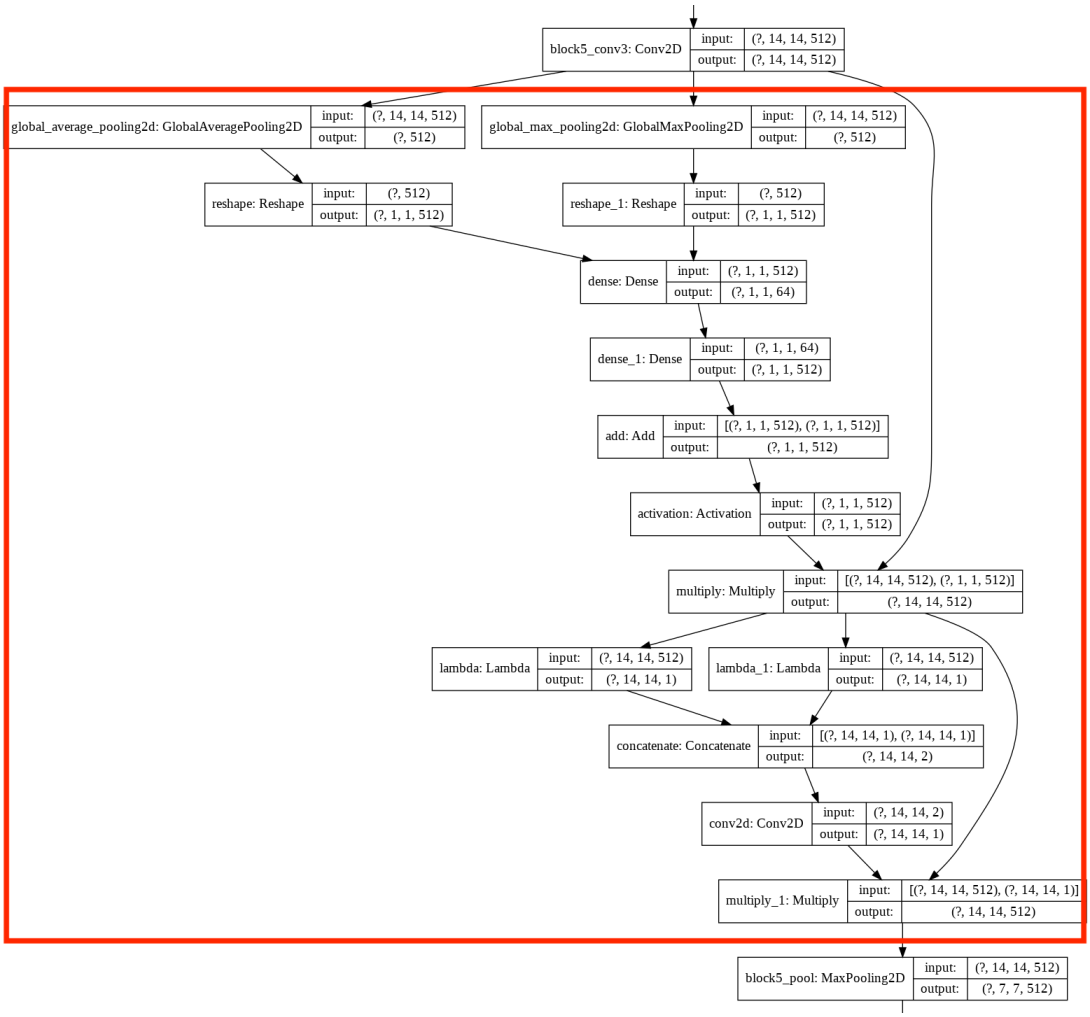


Figure 3.6: Cropped architecture of the foram model. A single CBAM (shown in red) was implemented in the fifth convolutional block, between the 'block5_conv3' and 'block5_pool' layers of the model (see Figure 3.3 for reference).

3.9 Summary

In summary, we have successfully been able to replicate and build upon the reference model [1]. The first section of this chapter outlined several pre-processing methods which ensured our dataset was presented correctly for an accurate training process. We then produced several primitive versions of the classifier in order to gain an understanding of the fundamentals of convolutional neural networks before finally replicating the same architecture and hyper-parameter configurations associated with the reference model. Following this, we explored a variety of different implementations for each of the methods discussed in the background chapter, including data augmentation, focal loss and CBAM.

Chapter 4

Results and Critical Evaluation

In this chapter we will aim to evaluate the performance of a variety of separate configurations of the foram classifier. We will determine whether the aims of the model have been met, and whether the implementation of data augmentations, focal loss and CBAM have improved the validation accuracy of the classifier.

As well as a general overview of all configurations, individual implementations will be tested and evaluated thoroughly within their own sections. A variety of ablation studies will follow this, before finally selecting the model which shows the highest validation accuracy to perform a 5-fold cross validation on.

4.1 Results Table

Table 4.1 shows the performance of all configurations of the foram model. Through this table, we can see that the reference model, shown in analysis 4, achieved a top validation accuracy of 0.87. Through the integration of data augmentation, along with a CBAM in the final convolutional layer, we have successfully increased the top validation accuracy of the foram classifier to 0.9090, shown in analysis 12 of Table 4.1

A consistent result of training all models associated with this thesis was a period early on in the training process where the validation accuracy was higher than the training accuracy. Examples of this can be seen in Figure 4.1. This raises the question - how can a model be more accurate at classifying unseen examples than samples it has trained on? Although this result may defy common sense, it can be understood by thinking more deeply about the batch size of the network and the training accuracy.

During the training period, a model learns new features which in turn increase its performance. Towards the beginning of this training process, performance increases are larger than later stages of the training process. As our network is trained in batches, the first batch of an epoch will produce a lower accuracy than batches towards the end of the epoch. Because the final training accuracy of an epoch is a weighted sum of all accuracies for each batch of the epoch, the final training accuracy of an epoch does not accurately reflect the true accuracy of the model at that point in training. As such, when the model is validated following this training epoch with the full performance gains, the validation accuracy can be higher than the training accuracy. In later stages of the training process, the increases in performance are smaller, there-

fore the final training accuracy of an epoch is closer to the true accuracy of the model at that point in training.

Analysis	Network	Loss function	Epochs trained	Max train acc	Min train loss	Max val acc	Min val loss
1	Vanilla (2 most abundant)	B	80	0.9287	0.1773	0.9174	0.1971
2	Vanilla (All vs most abundant)	B	80	0.9656	0.0969	0.9033	0.296
3	Vanilla (All classes)	C	80	0.8997	0.3261	0.6161	1.5206
4	**VGG16 Reference Model [1]**	**C**	**28**	**0.9999**	**0.0003**	**0.8741**	**0.5638**
5	VGG16	C	40	0.9905	0.0325	0.8493	0.7343
6	VGG16 + Aug(R)	C	35	0.9362	0.2049	0.9001	0.3310
7	VGG16 + Aug(B)	C	35	0.9814	0.0579	0.8558	0.7131
8	VGG16 + Aug(Z)	C	35	0.9782	0.0623	0.8524	0.7332
9	VGG16 + Aug(R, B, Z)	C	50	0.9334	0.2049	0.9050	0.0867
10	VGG16 + Aug(R, B, Z)	FL	40	0.9102	0.0607	0.8925	0.3310
11	VGG16 + CBAM	C	20	0.9775	0.0627	0.8655	0.7116
12	VGG16 + Aug(R, B, Z) + CBAM	C	50	0.9284	0.2121	**0.9090**	0.3112
13	**5-FOLD VGG16 + Aug(R, B, Z) + CBAM**	**C**	**50**	**0.9266**	**0.2169**	**0.9055**	**0.3178**

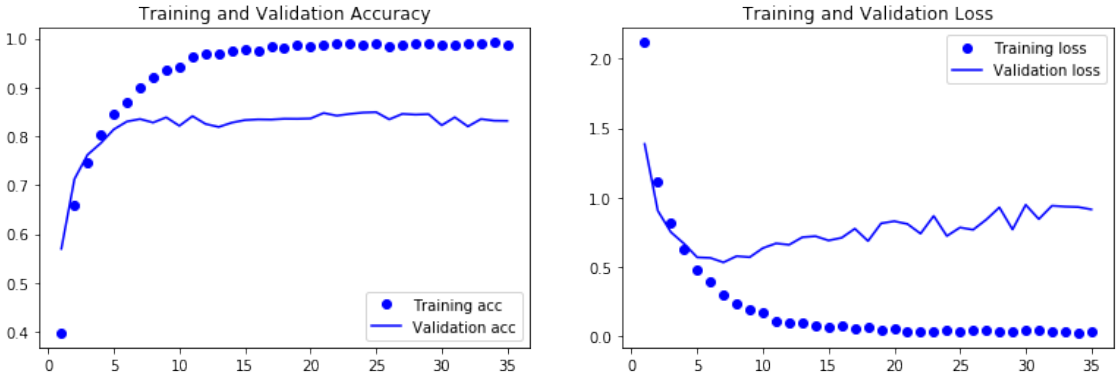
Table 4.1: Results for a variety of different foram model configurations. The horizontal divides signify a separate configuration. From top to bottom - Binary classifiers, vanilla and reference model implementations, data augmentations, focal loss, CBAM, and 5 fold cross-validation. The augmentations R, B, and Z represent rotation, brightness, and zoom augmentations.

4.2 Data Augmentation

As the analysis numbers 4-8 in Table 4.1 show, the largest increase of performance came from the implementation of data augmentation. By manipulating the sample rotation, zoom and brightness, the model's top validation accuracy increased from 0.8493 to 0.9050 (Analysis numbers 5 & 9 in Table 4.1). The individual training and validation plots for the VGG16 network with and without data augmentation are seen in Figure 4.1.

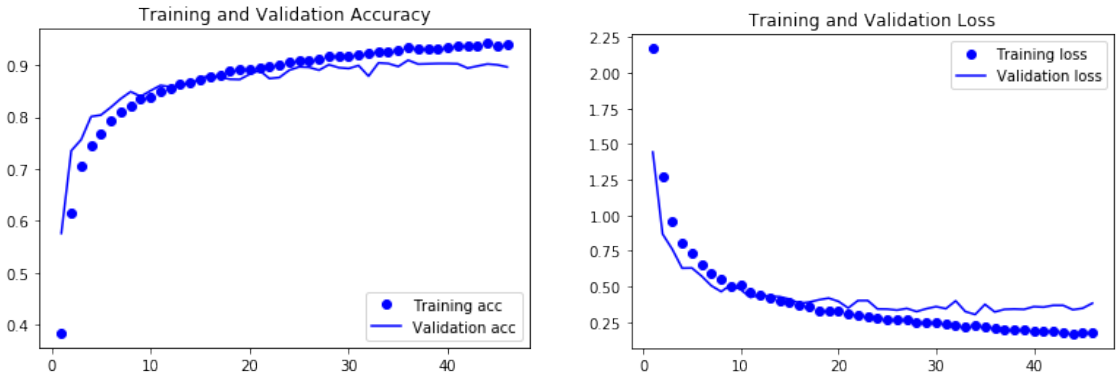
Models over-train when they begin to over-train on statistical noise within the training set, as opposed to features which are relevant to the task. In terms of the foram model, this could be interpreted as over-training on image grain/noise or a specific foram orientation. Because the augmentation model is exposed to a slightly new set of samples each epoch, it finds it more difficult to over-train on statistical noise within these samples [5]. Specifically, we see that rotation caused the largest performance increase out of the three tested augmentations (See analysis 6, Table 4.1). This signifies that the model was previously overfitting on the same orientation of foram, and therefore lacked generalisability when exposed to other rotational orientations.

The addition of data augmentation increases the length of time for the model to reach its



(a) baseline accuracy

(b) baseline loss



(c) Aug(R, B, Z) accuracy

(d) Aug(R, B, Z) loss

Figure 4.1: Accuracy and loss graphs for the baseline VGG16 model 4.1a, 4.1b vs. the VGG16 model with rotational (R), brightness (B), and scalar (Z) augmentations 4.1c, 4.1d.

peak training accuracy. In addition, the peak training accuracy is lower for the augmentation model than the baseline model, 0.9905% vs 0.9334%. This difference in training accuracy is due to the fact that the model can no longer over-train on statistical noise in the input data.

The net result of data augmentation is a reduction in overfitting, which produces a higher overall validation accuracy due to the model's increased generalisability when exposed to new data.

4.3 Focal Loss

Focal loss was shown to have an overall negative affect on the validation accuracy of the classifier for all values of α in the range 0.25 - 1. The highest validation accuracy, 0.8925, was reached by the classifier when $\alpha = 0.5$. Through use of classification reports, adapted from sklearn.metrics, we were able to discern individual accuracies for each species. A sample of these accuracies are shown in Table 4.2.

Observing this table, we find that the incorporation of focal loss increases the validation accuracy of the classifier for species with low sample numbers. This follows in accordance with

Analysis	Species	Val Sample Size	CE Val Acc.	FL Val Acc.	Val Acc. Change	Weighted Change
1	Globorotalia Hirsuta	48	0.84	0.94	+0.10	+4.8
2	Globorotalia Ungulata	5	0.75	0.8	+0.05	+0.25
3	Globoturborotalita Rubescens	47	0.83	1.00	+0.17	+8.24
4	Tenuitella Iota	9	0.00	1.00	+1.00	+9
5	Turborotalita Quinqueloba	60	0.83	0.96	+0.13	+7.8
6	Globigerinoides Ruber	1182	0.98	0.97	-0.01	-11.82
7	Globigerina Bulloides	414	0.93	0.89	-0.04	-16.56
8	Neogloboquadrina Incompta	406	0.94	0.91	-0.03	-12.18

Table 4.2: Foram model classification report results for separate loss functions. Species 1-5 reached a higher validation accuracy through focal loss, whereas species 6-8 reached a higher validation accuracy through categorical cross-entropy loss. The weighted change in the final column is calculated by multiplying the validation accuracy change by the validation sample size.

the results shown by the referenced paper [2]. When the network is exposed to less samples of a certain species, they become harder to classify. By down-weighting the influence of well classified examples, we can increase the influence of these less abundant species on the re-weighting of the network.

We can see from column 5 of Table 4.2 that focal loss results in a higher change in validation accuracy for less abundant species. For example, in the special case of Tenuitella Iota, the validation accuracy increased to 100%, where previously no samples were correctly classified. However, although the gains in validation accuracy for less abundant species are higher than the losses in validation accuracy for more abundant species, the weighted result of these changes gives an lower overall validation accuracy for the dataset as a whole. This is shown best in column 6 of Table 4.2. We can conclude that the differences seen between species in regards to class imbalance are simply too large for focal loss to have a positive overall effect on the classification accuracy of the foram model.

If the taxonomists using the classifier are more concerned with the overall accuracy of the predictions, focal loss should not be used to train the model. However, if the validation accuracy for less abundant models is viewed as more important than the overall validation accuracy of the model, focal loss can be incorporated to increase the influence of less abundant species. The ideal loss function for the model depends on what taxonomists view as the correct use case for the model.

The best solution to this class imbalance problem, as opposed to focal loss, is to simply add more samples to the dataset for less abundant species. As the sample numbers grow, the influence of the less abundant species on the re-weighting of the model will increase without the need to down-weight the influence of the already well classified examples.

4.4 CBAM

Analysis numbers 10-11 in Table 4.1 show the results of training the model with a single CBAM in the final convolutional block, using the implementation mentioned in [33].

With the incorporation of data augmentation, our network now reaches a high validation accuracy of 0.9050. However, the closer the accuracy gets to a perfect 100%, the harder it is to further increase, as only the most difficult examples remain to be classified correctly. Because of this, the differences we see in accuracy when implementing a CBAM (0.9050 vs 0.9090) could either represent a positive effect on the network, or simply a statistical fluctuation in accuracy between models. To observe the impact of the attention modules more closely, data augmentation was removed from the CBAM network (Analysis 11, Table 4.1). Through this method, we can train the CBAM integrated model against our implementation of the reference model (Analysis 5, Table 4.1), which has a measured validation accuracy of 0.8493, far lower than 0.9050. By measuring the validation accuracy of CBAM without the performance benefits of data augmentations, we can more concretely measure whether it has a positive influence on the validation accuracy of the foram model. Observing analysis 11, we can see that CBAM modules give a 1.62% increase in validation accuracy over analysis 5, a small but noticeable improvement. We can conclude that the addition of a single CBAM in the final convolutional layer results in a slightly improved validation accuracy.

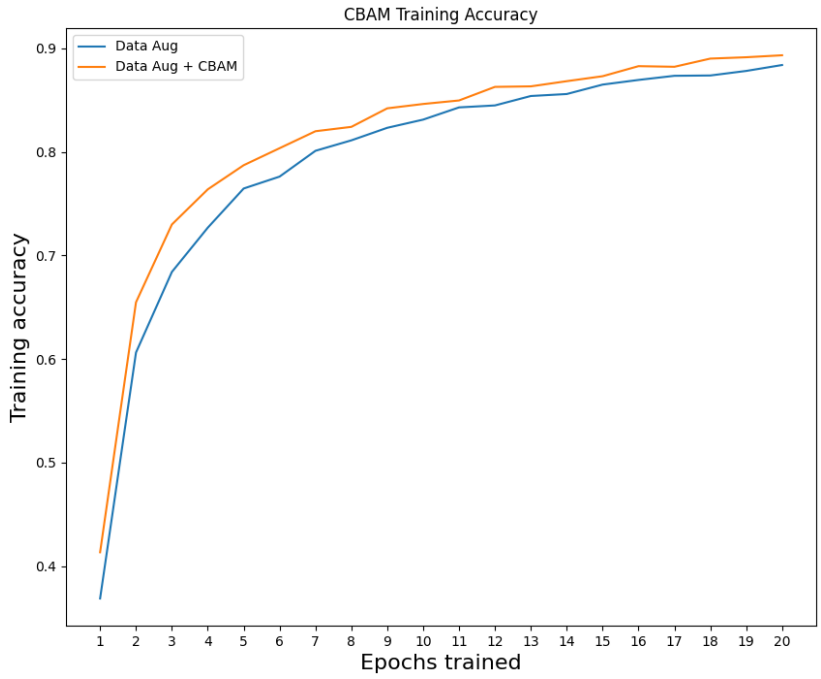


Figure 4.2: The first 20 epochs of training accuracies for CBAM integration (Analyses 9 & 12, Table 4.1). The CBAM integrated model consistently outpaces the training accuracies achieved by the data augmented model.

In addition to an increased validation accuracy, the training time for the CBAM integrated model was vastly reduced when compared to the model without CBAM implementation (shown in Figure 4.2). This can be interpreted as the model learning useful features about the classification problem earlier in the training process. We can see from Table 4.1 that the model reaches a higher top training accuracy using CBAM, which shows that the model is learning slightly more from the training data with the addition of CBAM.

The net result of CBAM integration is an increase in training accuracy, which produces a higher overall validation accuracy. This is due to the model's attention being focused on the most relevant features and locations which aid the foram classification process.

4.5 Ablation Studies

Following the implementation of data augmentation, focal loss and CBAM, a variety of different configurations of the model were tested using different hyper-parameters. As with the reference model [1], a learning rate of 0.0001, a batch size of 100 and a dropout rate of 0.5 gave the highest validation accuracy with a minimum training period. However, a trial of different levels of layer freezing for the pre-loaded ImageNet VGG16 network indicated that freezing the first two convolutional layers, as with the reference model, lead to a slightly lower validation accuracy than training the model without layer freezing. Results from the trial can be seen in Figure 4.3.

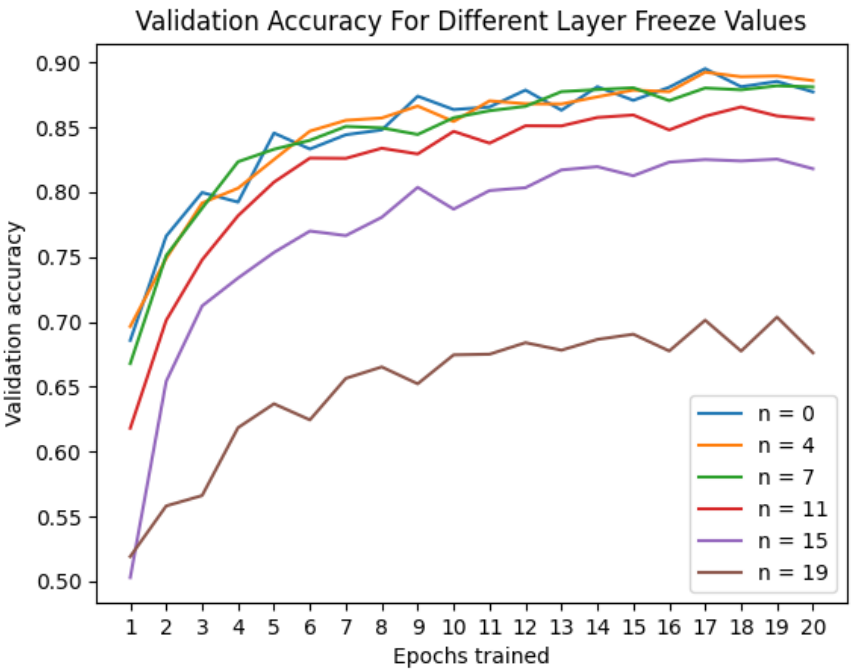


Figure 4.3: The first 20 epochs of validation accuracies achieved through freezing different levels of the network. N = 0 corresponds to no freezing, n = 4 corresponds to freezing the first convolutional block, n = 7 to freezing the first two convolutional blocks, etc. See Figure 3.3 for reference.

The reasoning behind this finding is that the foram classification problem is far removed from the ImageNet classification problem on which the transferred weights have been trained. As previously stated, earlier layers learn more general features about the input space, whereas later layers learn features specific to the classification problem. Because the samples within the foram dataset bare little resemblance to the images within the ImageNet dataset, there are less general transferable features to be shared between the two classification problems. A similar issue in medical imagery classification is discussed in [25].

This result begs the question - If the ImageNet pretrained weights are so far removed from our foram problem, why do we use transfer learning at all withing our network? During the compilation of the model, without the use of transfer learning, weights are initialised randomly. Whilst the pre-loaded weights are not ideal for our specific problem, and therefore not frozen in place, they give the model a more favorable starting position as opposed to random initialisation. This starting position allows the network to gain a general understanding of the problem in a shorter amount of time, decreasing the overall training time of the network. In terms of gradient descent, we can visualise this form of transfer learning as starting a fraction of the way down the slope of the plane (See Figure 2.3). As our foram model contains over 41 million different trainable weights, it is very unlikely that random initialisation will generate a more favorable starting position than transfer learning.

4.6 Grad-Cam Model Evaluation

A problem we encounter often within CNNs is interpreting and predicting model behaviour. Because a network is essentially a combination of millions of weights applied sequentially to an input, it is difficult to pinpoint where the network encodes class-specific features and why certain samples achieve certain results. Thus, miss-classifications of a given model can typically only be interpreted through observing the initial input image and manually assessing its differences to the rest of the correctly-classified samples.

Through the use of Grad-Cam [32], we can improve this qualitative form of miss-classification analysis by using weight gradients, similar to the back-propagation algorithm. However, instead of calculating the change of loss value with respect to the weights in the network, we instead calculate the influence of each weight on the input's predicted class. Grad-Cam generates a single heat-map for each input image, which shows what areas of the image are associated with the highest weight gradients in relation to its classification [32]. In other words, Grad-Cam highlights regions which are important to the classification process.

As a sample is fed through a CNN, the feature maps which are outputted from each layer contain less general visual information (such as straight lines), and more information relating to the class of the sample. The heat-map calculated through Grad-Cam is produced using the last visual layer of the model (i.e. before the dense layers), and therefore encodes the most relevant features about the class of a sample whilst also retaining spatial information about where these features are seen within the sample [32]. This heat-map can then be combined with the original input image to show exactly where the model is paying attention to.

Using this information provided by the superimposed heat-maps, we can interpret inconsistencies within a given dataset. For example, lets take a binary image classification problem of

dogs vs. cats. During the dataset creation, if all the pictures of dogs are taken in-front of a yellow background and the cats a blue background, the network might not train on the features of the animals but instead on the colour of the background. Even though the network might achieve a high accuracy by training on this particular dataset, it will yield a much lower performance in the real-world because the sample backgrounds are now inconsistent with the training data. Grad-Cam allows us detect these inconsistencies within a dataset by visualising exactly where the network is looking for within a sample to make a particular classification.

Several superimposed foram heat-maps can be seen in Figure 4.4. In this Figure, all samples were classified correctly apart from (e) and (f). By observing the heat-map (e), we can see that the model is directing too much attention to the background of the sample, rather than the foram itself. This suggests that our model is over-training on the background of the image for this particular class. In addition, the heat-map (f) suggests that the sample contains no features which are directly associated with the predicted class. However, as with all evaluation methods, we must be careful when making an analysis based off singular heat-maps. These samples could be outliers within a species, therefore the more samples that are tested with Grad-Cam, the more accurate our qualitative analysis will be.

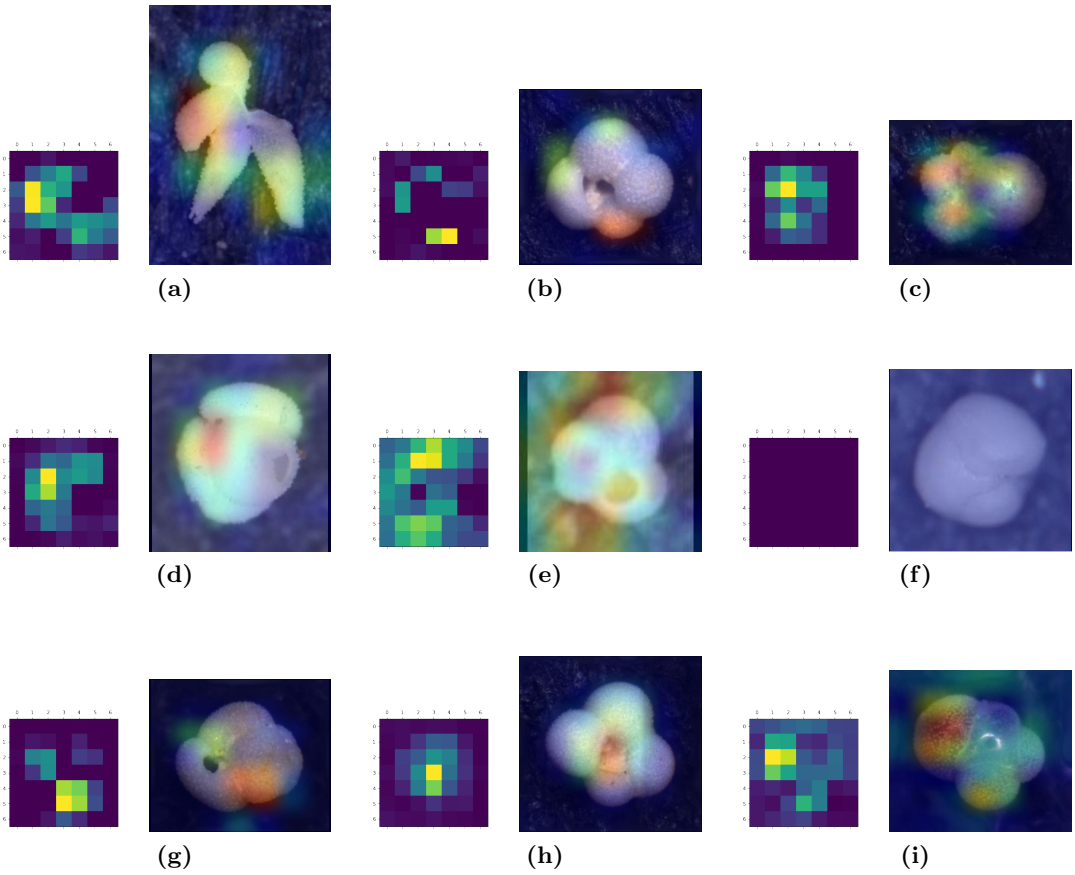


Figure 4.4: A variety of Grad-Cam heat-maps (left) for different samples along with their superimposed impressions on the input image (right). Three different samples can be seen in each row - (a) Globigerinella Adamsi, (b) Globigerina Bulloides, (c) globigerinella Calida, (d) Globigerinoides Conglobatus, (e) Globoturborotalita Tenella, (f) Globorotalia Crassaformis, (g) Globigerinoides Elongatus, (h) Globigerina Falconensis, (i) Globigerinita Glutinata.

Grad-Cam allows us to gain an insight into why particular classifications fail, rather than just if they fail. In terms of the foram model, the real benefit of Grad-Cam integration is the ability to compare areas of interest between human taxonomists and the foram classifier. If the attention location generated through a Grad-Cam heat-map matches the human attention location, it's very likely that the model is performing as intended. Conversely, discrepancy between the two could signify that the model hasn't learned the right features to look for. This comparison can be used to direct how models should be changed in order to increase overall performance.

4.7 Cross-Validation

Models which are intended to perform in the real world must be tested thoroughly before use. If the dataset used to train the model is unbalanced, the validation set might contain samples which are harder - or easier - for the model to classify in comparison to the training set and will therefore give a different validation accuracy than if the model was trained on a different split of the dataset.

For a meaningful validation accuracy result to be considered, a model must be trained and tested on all available sections of the dataset. An example of different split configurations are shown in Figure 4.5. After training on all separate splits of the available dataset, the results are then pooled together and averaged to give a weighted final result. As this weighted result is an average of all possible configurations, it carries more weight about the true performance of the model than the result from a single split.

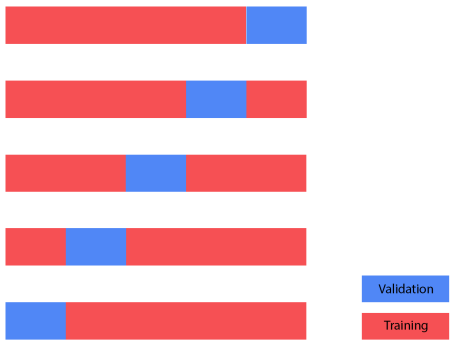


Figure 4.5: A 5-Fold split of a given dataset.

A single partition of this dataset is called a fold. The K-fold method introduces K different partitions to the model. As an 80:20 split was used to train our foram model, 5 partitions would ensure the model is trained and tested on all available splits of the dataset. An extreme version of this approach would be to train a separate model on all samples, leaving one single sample for validation. This would make K equal to the number of samples in the dataset. Through this method, a separate model would be trained on all data available and then tested against a single element.

In order to verify the results of our foram model, the results were cross validated to gain a deeper level of confidence in its predictions. The results of using 5-fold cross validation can be seen in Table 4.3.

Analysis	Max train acc	Min train loss	Max val acc	Min val loss
1	0.9284	0.2121	0.9090	0.3112
2	0.9124	0.2550	0.9030	0.3159
3	0.9430	0.1671	0.9096	0.3390
4	0.9160	0.2455	0.9007	0.3089
5	0.9334	0.2049	0.9050	0.3140
Avg.	**0.9266**	**0.2169**	**0.9055**	**0.3178**

Table 4.3: Results from 5-fold cross-validation study on the final foram model across 50 epochs.

Although analysis 5 shows a validation accuracy higher than that of our current best CBAM module, we cannot directly compare this result with any of the analysis' of Table 4.1, as the training sets and validation sets used to achieve this performance were created through using different partitions of the dataset.

Chapter 5

Conclusion

This paper has explored the classification of planktonic foraminifera through use of a CNN classifier. Through the implementation of rigorous data augmentation and selective attention modules, we have successfully improved upon the accuracy of the reference model. Cross-validating our model resulted in an average validation accuracy of 90.55%, 3.14% higher than the current reference model.

In Chapter 2, we explored the theory behind machine learning and deep convolutional networks. A literature review of recent machine learning architectures and practices followed this, with a specific focus towards methods that aim to lessen the negative effects of an imbalanced dataset, as well as methods that target datasets which are far removed from traditional computer vision problems. At the end of this chapter, we introduced the reference model which acted as a baseline for our foram model.

A practical implementation of the foram model was discussed in Chapter 3. We made use of a variety of data pre-processing techniques and explored some primitive versions of the classifier. Following this, we implemented data augmentation, focal loss and CBAM within the foram model in order to observe their individual effects on the performance of the classifier.

The results and evaluation of the model were presented in Chapter 4. Individual implementations were thoroughly tested and evaluated within their own sections. In addition, A variety of ablation studies were performed, before finally selecting the model which showed the highest validation accuracy to perform a 5-fold cross validation on.

5.1 Contributions of the Thesis

In this thesis, we have successfully improved upon the accuracy of a reference model through the implementation of various machine learning techniques. The contributions of this thesis can be summarised as follows:

- Literary review of several state-of-the-art CNN practices which have been developed to mitigate the negative effects of class imbalance and overfitting.
- Replication of the reference model to test and validate the results obtained in [1].
- Exploration of practices which reduce overfitting within pre-loaded models that are far removed from traditional image classification problems.

- Technical evaluation of several model configurations, with a detailed performance comparison of selected CNN practices [2, 3, 4].
- Conversion of state-of-the-art VGG16 architecture for usage as a pre-trained model within general geological taxonomy.
- Background on the foundations of machine learning and convolutional networks for the benefit of geologists interested in using and further developing the foram model.

Chapter 6

Future Work

Although we have significantly improved on the performance achieved by the reference model, there are still many ways our foram model can be refined. This chapter aims to address how this model should be developed in order to have a real, lasting effect on the foram classification process in the eyes of geological taxonomists.

6.1 Curriculum Learning

The ideal orientation of a foram for human classification is that which fully exposes an aperture, which is a hole used for feeding. The training samples used in the referenced dataset are all orientated as such, perfectly towards the microscope. However, although this consistency in orientation is likely to aid the accuracy of the classifier for this specific position, it also impedes the generalisability of the model when exposed to specimens which are not perfectly orientated.

Geologists will not benefit in practice from a model that requires samples to be orientated before the classification can take place. This is because the simple task of orienting the forams is more time consuming on average than the actual classification by taxonomists. The need for orientation defeats the ultimate practical purpose of the model, which is to increase the efficiency of the process as a whole. A superior model which could classify the forams from a variety of different positions could help taxonomists to completely bypass the orientation procedure and run the classifier on the samples as they happen to fall on the surface of the microscope slide.

In addition, many of the more complex foram species which are not included in the reference dataset are so closely related in shape that they often cannot be classified from a single angle. This furthers the need for a more generalisable system able to classify from several different orientations.

Forams are rarely perfectly spherical, meaning they have a set number of positions in which to lie on a flat surface. If a training dataset is developed which includes samples of all these possible positions, the model will then be able to generalise to classifying samples regardless of their orientation. What then, is the ideal method to train a classification model on a variety of different orientations of a specimen?

> "Humans and animals learn much better when the examples are not randomly presented but organized in a meaningful order which illustrates gradually more concepts, and gradually more complex ones." [34]

Curriculum learning centres around the way that humans learn as they develop from children through to fully-functioning adults. It involves a multi-stage training process which begins with emphasising fundamental concepts, which creates a model with a concrete but limited view of the classification problem as a whole. It then utilises these simpler, previously learned concepts to understand more complex samples when they are introduced later on in the training process, expanding the generalisability of the model as it progresses [34]. If we think of the classification problem as defining a set boundary between two species, curriculum learning serves to first solidify examples which are easily solvable and then to gradually expand the model's familiarity by introducing examples which are increasingly closer to this boundary. This is illustrated in Figure 6.1.

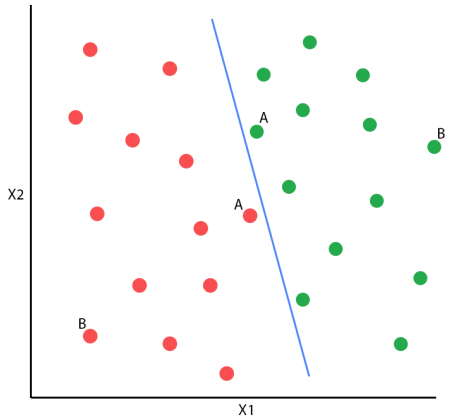


Figure 6.1: Curriculum serves to introduce gradually more complex samples to the model during the training process. This image shows an abstraction of a classification problem in two dimensions, involving two classes and a boundary set between them. The difficulty of classification for a specific sample can be interpreted as how far away from this classification boundary they lie [34]. Samples labeled 'A' will be harder for the model to classify than samples marked 'B', and should therefore be introduced later in the training process.

Curriculum learning is implemented by manually setting a weight to each of the samples. Early on in the training process, the model's loss function favours the examples which are simpler. As the training stages progress the weighting is adjusted to include more difficult examples until finally all the weights are all uniform and the model is trained on the entire set. This method is separate from focal loss as the initial weight of each sample is set by a human in accordance to its perceived difficulty. In contrast, focal loss uses the past accuracy of the model for a particular class to set the weight of a given sample of that class.

In terms of our foram model, the difficulty of each orientation of the foram can be evaluated by the taxonomists as they are added to the dataset. The current referenced dataset contains samples which are presented in their ideal orientation for classification, therefore these samples should serve as the simplified part of the overall dataset. As the training process takes place, the weighting of the more difficult orientations will increase until all samples are weighted equally.

Bibliography

[1] Allison Y. Hsiang. *Endless Forams: ¿34,000 Modern Planktonic Foraminiferal Images for Taxonomic Training and Automated Species Recognition Using Convolutional Neural Networks*. 2019.

[2] Tsung-Yi Lin. *Focal Loss for Dense Object Detection*. 2018.

[3] Sanghyun Woo et al. *CBAM: Convolutional Block Attention Module*. 2018. arXiv: 1807.06521 [cs.CV].

[4] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6 (2019), pp. 1–48.

[5] Francois Chollet. *Deep Learning with Python*. 1st. USA: Manning Publications Co., 2017. ISBN: 1617294438.

[6] Leo Breiman. "Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)". In: *Statist. Sci.* 16.3 (Aug. 2001), pp. 199–231. DOI: 10.1214/ss/1009213726. URL: https://doi.org/10.1214/ss/1009213726.

[7] Jayesh Bapu Ahire. *The Artificial Neural Networks handbook: Part 1*. 2018. URL: https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4.

[8] W.S. McCulloch and W. Pitts. *A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics*. 1943.

[9] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. 2018. arXiv: 1803.08375 [cs.NE].

[10] Sagar Sharma. *Activation Functions in Neural Networks*. 2017. URL: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[11] Farhad Malik. *Neural Networks Bias And Weights*. 2019. URL: https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da.

[12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0.

[13] Daniel Burkhardt Cerigo. *On Why Gradient Descent is Even Needed*. 2018. URL: https://medium.com/@DBCerigo/on-why-gradient-descent-is-even-needed-25160197a635.

[14] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[15] Isabelle Guyon. “A Scaling Law for the Validation-Set Training-Set Size Ratio”. In: 1997.

[16] C. Nebauer. “Evaluation of convolutional neural networks for visual recognition”. In: *IEEE Transactions on Neural Networks* 9.4 (1998), pp. 685–696.

[17] Li Yin. *A Summary of Neural Network Layers*. 2018. URL: https://medium.com/machine-learning-for-li/different-convolutional-layers-43dc146f4d0e.

[18] Dominik Scherer, Andreas Müller, and Sven Behnke. “Evaluation of pooling operations in convolutional architectures for object recognition”. In: Jan. 2010, pp. 92–101. DOI: 10.1007/978-3-642-15825-4_10.

[19] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[20] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[21] Muneeb ul Hassan. *VGG16 – Convolutional Network for Classification and Detection*. 2018. URL: https://neurohive.io/en/popular-networks/vgg16/.

[22] Fuzhen Zhuang et al. *A Comprehensive Survey on Transfer Learning*. 2019. arXiv: 1911.02685 [cs.LG].

[23] Deepak Soekhoe, Peter Putten, and Aske Plaat. “On the Impact of Data Set Size in Transfer Learning Using Deep Neural Networks”. In: Oct. 2016, pp. 50–60. DOI: 10.1007/978-3-319-46349-0_5.

[24] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3320–3328. URL: http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf.

[25] Maithra Raghu and Chiyuan Zhang. *Understanding Transfer Learning for Medical Imaging*. 2019. URL: https://ai.googleblog.com/2019/12/understanding-transfer-learning-for.html.

[26] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[27] Xinyu Yang, Majid Mirmehdi, and Tilo Burghardt. *Great Ape Detection in Challenging Jungle Camera Trap Footage via Attention-Based Spatial and Temporal Feature Blending*. 2019. arXiv: 1908.11240 [cs.CV].

[28] Long Chen et al. *SCA-CNN: Spatial and Channel-wise Attention in Convolutional Networks for Image Captioning*. 2016. arXiv: 1611.05594 [cs.CV].

[29] Andrea Galassi, Marco Lippi, and Paolo Torroni. *Attention in Natural Language Processing*. 2019. arXiv: 1902.02181 [cs.CL].

[30] Xinjian Guo et al. “On the Class Imbalance Problem”. In: *Fourth International Conference on Natural Computation, ICNC '08* Vol. 4 (Oct. 2008). DOI: 10.1109/ICNC.2008.871.

[31] Umberto Griffo. *focal-loss-keras*. 2018. URL: https://github.com/umbertogriffo/focal-loss-keras.

[32] URL: https://scikit-learn.org/stable/.

[33] ByungSoo Ko. *CBAM-keras*. 2018. URL: https://github.com/kobiso/CBAM-keras.

[34] Y. Bengio et al. "Curriculum learning". In: vol. 60. Jan. 2009, p. 6. DOI: 10.1145/1553374.1553380.

On_the_Classification_of_Planktonic_Foraminifera_2020

Final Audit Report 2020-09-01

Created:	2020-09-01
By:	Jake Ramaer (jake.ramaer@gmail.com)
Status:	Signed
Transaction ID:	CBJCHBCAABAAnaVfxya42zOHqFRix3mJ-zY-E9VuIFOs

"On_the_Classification_of_Planktonic_Foraminifera_2020" History

Document created by Jake Ramaer (jake.ramaer@gmail.com)
2020-09-01 - 9:59:56 PM GMT- IP address: 213.18.139.100

Document emailed to JRamaer (jr17327@bristol.ac.uk) for signature
2020-09-01 - 10:01:34 PM GMT

Email viewed by JRamaer (jr17327@bristol.ac.uk)
2020-09-01 - 10:01:59 PM GMT- IP address: 213.18.139.100

Document e-signed by JRamaer (jr17327@bristol.ac.uk)
Signature Date: 2020-09-01 - 10:02:45 PM GMT - Time Source: server- IP address: 213.18.139.100

Signed document emailed to JRamaer (jr17327@bristol.ac.uk) and Jake Ramaer (jake.ramaer@gmail.com)
2020-09-01 - 10:02:45 PM GMT