Flagellates Classification via Transfer Learning

Eric Ho
University of California - San Diego

Brian Henriquez
University of California - San Diego

Jeffrey Yeung
University of California - San Diego

Abstract

Transfer learning has become a large part of training deep neural networks due to the immense amount of data required to learn millions of weights. Transfer learning aids in this process by freezing all CNN weights except the fully-connected layers such that the learned features are also retained. By using a network trained by an expansive dataset such as the ImageNet dataset, the chance that features can apply to the new domain is high. The purpose of our project is to train both the VGG16, VGG19, and Xception networks while identifying possible ways of increasing network performance. We will focus on the effects of optimizer and learning rate variation on the learning process and whether it positively affects the final loss and accuracy of the network. Other optimization techniques include training on different classification layers. The data that will be used for this testing will be the WHOI plankton dataset provided by Professor Peter Gerstoft.

1 Introduction

With advancements in technology, plankton classification, once a tedious task manually performed by researchers, can now be greatly accelerated with machine learning techniques. The Woods Hole Oceanographic Institute (WHOI) developed an instrument capable of collecting millions of plankton images each day, and many of the collected images have been annotated and uploaded online for external references. In this project, we utilize various convolutional neural network architectures, namely VGG16, VGG19, and Xception, to explore their performances for plankton classification. Our dataset is the annotated image set provided by WHOI, specifically the Flagellates class as it has the most even distribution of subclasses, meaning that our data is not heavily skewed towards any category. In addition to training our neural networks for classification, we will also explore different parameters such as data augmentation and optimization to examine their effects on classification performance. Lastly, we will conclude the advantages and disadvantages of the various parameters and discuss any significant findings.

2 Related Works

The most recent work on plankton classification is by Eric Orenstein[3] who also used the WHOI plankton dataset. In his paper, he used three baseline classifiers: one with random forest, a CNN trained on plankton data, and a fine-tuned CNN trained on ImageNet data. The networks was trained to classify 70 classes from the WHOI dataset. Similarly, we also use the transfer learning method in classifying our dataset. They achieved 93.8% accuracy with their fine tuned model due to a large majority of the samples being in the 'mix' class, that is "a classifier could map all ROIs in the test-set to the 'mix' class and achieve 60% accuracy"[3]. The class-imbalance problem of the WHOI dataset is a major problem when classifying. Most notably, it was first mentioned in a IEEE paper by Hansang

Lee et. al.[5]. A model trained with an imbalanced dataset causes the classifier to be biased so that its performance is degraded for small-sized classes which is reflected in Orenstein's results. In contrast, our model will not be classifying all 70 classes but specifically the Flagellates species which has 12 classes. We chose that species because it was the most balanced in terms of images per class.

Recently, Kaggle hosted the "National Data Science Bowl".[1] In this competition, there are 120 classes of various species of plankton. Out of 50 million images in the dataset, 30,000 training images was "ran through an automatic process to extract regions of interest, resulting in smaller images that contain a single organism"[1]. Although our dataset does not have as many classes, we don't have as many training images which make classifying difficult. In the realm of supervised learning, more data is better in general.

3 Dataset

We used the annotated plankton dataset from Woods Hole Oceanographic Institution (WHOI) which contains more than 3.5 million images of microscopic marine plankton[6]. In this paper, we will only classify the Flagellates plankton species that consists of 12 subspecies as shown in figure 1. We accumulated images of Flagellates from the 2006 to 2014 to expand our dataset. After splitting our dataset into 70% training, 15% validation, and 15% test sets, we have 12,210 images for training, 2,572 for validation, and 2,719 for testing, totaling in 17,501 images.

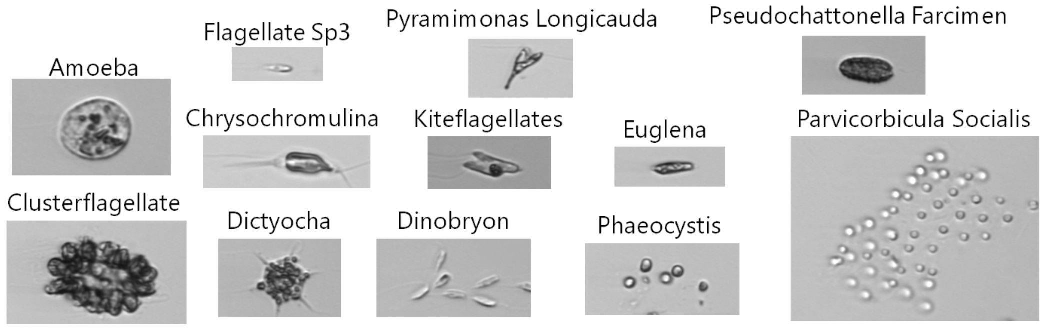


Figure 1: Twelve Flagellates subspecies for classification.

3.1 Preprocessing

Each image has variable width and height which was an issue since deep neural networks take a fixed size at the input layer. To solve this problem, we chose a set input size of 299x299 pixels and either crop or pad the images with zeros to adjust it to that size. In addition, the images are in grayscale instead of RBG which is the standard number of channels. To get three channels, we simply duplicate the grayscale channel more times to get 299x299x3 input size.

4 Methods

4.1 Image Augmentation

One of the main challenges in utilizing a machine learning model is that there is often an insufficient amount of data. Ideally, we would want to train the models with an infinite number of training images, but in practice, the number of images we have is limited. One of the most common ways to artificially increase the dataset is to manipulate and create different variations of the existing dataset to use for training; this technique is called dataset augmentation. Dataset augmentation is particularly effective in object recognition applications because images are high dimensional and contain a variation of factors.[4]

The dataset augmentation techniques we implemented for our project are horizontal flipping, vertical flipping, image padding, and image cropping. We chose horizontal and vertical flipping because our image dataset contains distinctively-looking subclasses so a simple flip is sufficient in effectively creating more image data. We also padded and cropped our images because our original dataset contained images of varying sizes, so we standardized all the images to 299x299 pixels to maintain consistency across our dataset.

4.2 Types of Models

For our plankton classification, we employed three different convolutional network architectures to compare the performances of each. The three models we used are VGG16, VGG19, and Xception. In choosing our models, we picked VGG16 and VGG19 because they are very similar, with VGG19 being slightly more complex due to its three extra weight layers. Their similarity allows us to examine whether the increased complexity and depth of VGG19 is significant with respect to our relatively distinctive dataset. Lastly, we used the Xception model because of its high performance on the ImageNet dataset, and compared its effectiveness relative to the VGG models.[11]

4.2.1 VGG16 and VGG19

The basis of the VGG models is that they were designed with a priority of increasing model depth while keeping other parameters fixed. In a VGG model, the image is passed through a stack of 3x3 convolution filters, which are used to capture the notion of left/right, top/bottom, and center. These filters are followed by three Fully-Connected layers, the first two having 4096 channels, and the third having 1000 channels for the 1000-way ImageNet classification.[11] The above describes the general architecture of the VGG models and the main difference between VGG16 and VGG19 is that there are three more convolution layers in VGG19, meaning it is a deeper and more accurate model. For our project we decided to train with both models to see if the advantage of potentially higher accuracy from VGG19 is worth the higher computational time required.

4.2.2 Xception

The central theme behind the Xception network are the depth-wise separable convolutions that make up the network. A depth-wise separable convolution is effectively a two-dimensional convolution over all the input channels separately followed by a convolution between channels. By following this scheme, the number of parameters required by the convolution is lowered. By further supposing that all color channels are completely separable (as opposed to Inception's *sufficiently* separable channels), Chollet manages to create a network that outperforms Inception on the ImageNet dataset.[2] Since this architecture is based on the Inception module, we expect it to learn larger more complex features from the plankton dataset.

For this classification, we decided to use Xception due to its performance on the ImageNet dataset. Since it exhibited the best performance of all the networks being used, we hypothesize that the Xception network will perform the best when trying to learn the plankton features. Since the Xception network has a much larger memory footprint since the entire network will be trained, the batch size will be smaller than that for the VGG networks.

4.3 Transfer Learning

Transfer learning is the process of using pre-trained network weights as a starting point for a new network. The idea behind transfer learning is the transferring of domain-specific "knowledge" between datasets in a process known as domain transferring. [4] Our networks are pre-trained on the ImageNet dataset, a dataset that incorporates 1000 classes and thousands of images for each class.

Although domain transfer is the most common usage for transfer learning, there is a topic known as distant domain transfer learning. This is closer to what will be applied here since there are no plankton in the ImageNet dataset. The idea behind this is that the network is treated as a feature selector. It is these features which are then used in order to make a classification. For VGG16 and VGG19, the weights are used a frozen feature selector which is then classified using fully-connected layers. In the case of Xception, we use the weights as an initialization so that the already present lower features allow for faster network convergence.

4.3.1 Early Stopping

Early stopping is the simplest form of regularization in deep learning. To implement early stopping, we separate a portion of our data into a validation set and proceed to evaluate the accuracy on this set after every epoch. If the accuracy of the validation set has not improved after a specified number of epochs (in our case, three), then the optimization process is ended and the current parameters are retained.

To see why early stopping acts as regularization, note that the action of stopping early during an optimization process acts as a bound on the parameter space of the model. To be precise, if we consider η to be the learning rate and ϵ to be the total number of iterations of the optimization, then the parameter space is roughly bounded by the inverse of $\eta\epsilon$.[4]

4.4 Fine Tuning

Alongside transfer learning, we fine tuned each model using a series of parameter adjustments and using the best parameter, continue to change other parameters. First, we find the best optimizer to use for training each model, specifically Adam, stochastic gradient descent, and RMSprop. After training each model with each optimizer and comparing the test accuracy, we use the optimzier with the highest test accuracy and trained the model again with various learning rates. Again, we use the highest test accuracy of a particular learning rate, and train the model with three different classification layers. We will describe each adjustments in the following sections.

4.4.1 Optimizers

In the first round of fine tuning, we chose the best optimizer for classifying our Flagellates dataset. Essentially an optimizer or optimization algorithm is needed to minimize an objective or cost function $J(\theta)$, resulting in the model learning the internal representations of the dataset, i.e. the parameters θ.[12]

Stochastic Gradient Descent In stochastic gradient descent (SGD), we seek to minimize our objective function $J(\theta)$ where by updating θ in such a way that the gradient $\nabla_\theta J(\theta)$ goes toward its opposite direction. The learning rate η tells you how far each 'step' of the gradient takes to get to the minimum.[10] SGD updates θ for each training example $x^{(i)}$ and label $y^{(i)}$:

$$\theta_{t+1} = \theta_t - \eta * \nabla_\theta J(\theta_t; x^{(i)}; y^{(i)}) \tag{1}$$

where η is the learning rate. Since SGD frequently update parameters θ, the loss fluctuates constantly. However, this method tend to converge to local minimums because the fluctuations allow the model to jump to potentially lower local minimums.[10] One caveat is that SGD usually takes longer to converge than other optimizers. In addition, it uses only one learning rate η for *all* parameter updates. This might not be the optimal as each dataset can have vastly different results depending on the chosen learning rate.

RMSprop The next optimization method we used was RMSprop which is an adaptive learning rate method developed by Geoff Hinton[7]. For RMSprop, the learning rates η is divided by a moving average of the squared magnitude of the gradient of its respective parameter θ:

$$\begin{aligned} \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{MeanSquared(\theta_t)}} * \nabla_\theta J(\theta_t) \\ MeanSquared(\theta_t) &= 0.9 * MeanSquared(\theta_{t-1}) + 0.1 * \nabla_\theta J(\theta_t) \end{aligned} \tag{2}$$

An advantage of RMSprop is that it solves the problem of a vanishing learning rate and still have the benefit of automatically adjusting learning rates.

Adam Adaptive Moment Estimation (Adam) also calculates adaptive learning rates for each θ.[9] Adam also stores the exponentially decaying average of past gradients:

$$\begin{aligned} v_t &= \beta_1 v_{t-1} + (1-\beta_1) g_t^2 \\ m_t &= \beta_2 m_{t-1} + (1-\beta_2) g_t \end{aligned} \tag{3}$$

where $\beta_{1,2}$ are hyperparameters for adjusting Adam and g_t is the gradient at iteration t. The mean m_t of the gradient is the fist moment and the variance v_t of the gradient is the second moment, hence the name of the algorithm.[10] Using these variables, the update rule becomes:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \tag{4}$$

Currently, Adam is the most popular optimization algorithm for deep learning as it performs well in practice.[9]

4.4.2 Learning Rates

After we have found our best optimizer to use for each model, we continue to fine tune by varying the learning rate η. Using the range $\eta \in [0.1, 0.01, 0.001, 0.0001]$, we applied transfer learning on each model again and saved the model and test accuracy. Choosing an optimal learning rate can be difficult because each optimizer and dataset can produce different results. A really small learning rate will cause the model to converge slowly and inefficiently. A learning rate that is too large can cause instability, hinder convergence, and even diverge.[10]

4.4.3 Various Classification Layers

Lastly, using the best optimizer and learning rate, we performed transfer learning once again with three different classification heads or layers as shown below:

Classification Layer 1	Classification Layer 2	Classification Layer 3
Average Pooling	Flatter	Flatten
Dense (512)	Dense (1024)	Dense (512)
Dense (12)	Dropout	Dropout
	Dense (512)	Batch Normalization
	Dense (12)	Dense (12)

The classification layers are important because it dictates how the model will classify a given image after all the convolutions. One can think of a CNN as a feature extractor with a classification at the end. Essentially the many convolution layers are the feature extractors as the images become more and more broken down as it propagates through the network. At the end, the classification layers will use the extracted features to classify. Training on the classification layers will help the model learn to differentiate the classes and the three different classification layers gives insight on the best method in classifying our Flagellates classes.

5 Results

The following sections details the results of running the optimizer variation on the VGG networks with a batch size of 64 and the Xception network with a batch size of 16. Note that in the long run, this difference will not cause a large problem other than inducing a slightly more stochastic accuracy/loss curve. The results fell mostly in line with what we believed would occur except for the chosen optimizer.

Before getting into the tabulated results, the following are the accuracy/loss curves for the best performing network variations:

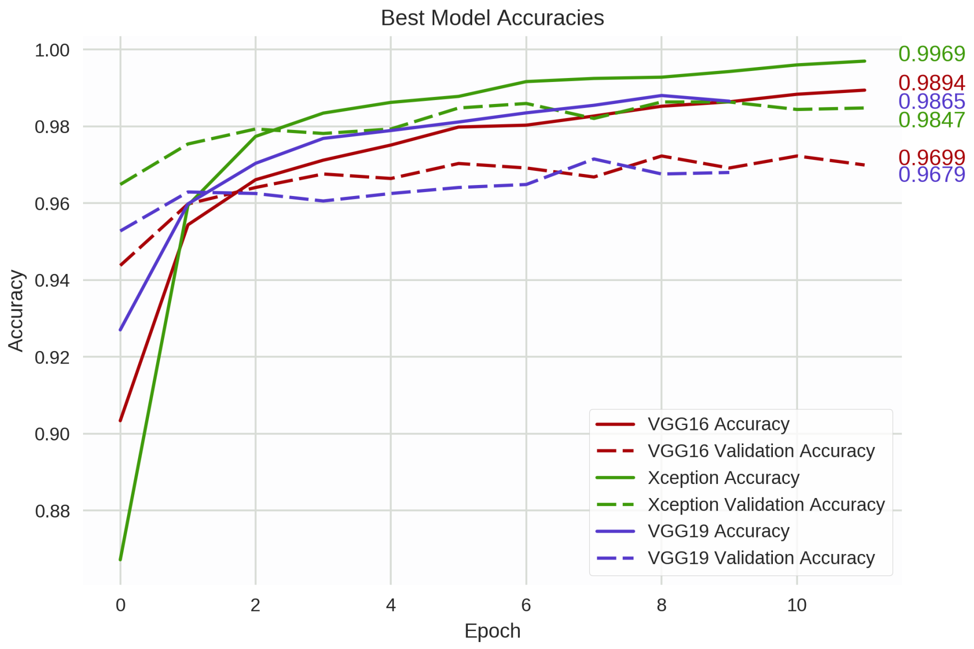


Figure 2: Best Model Accuracies

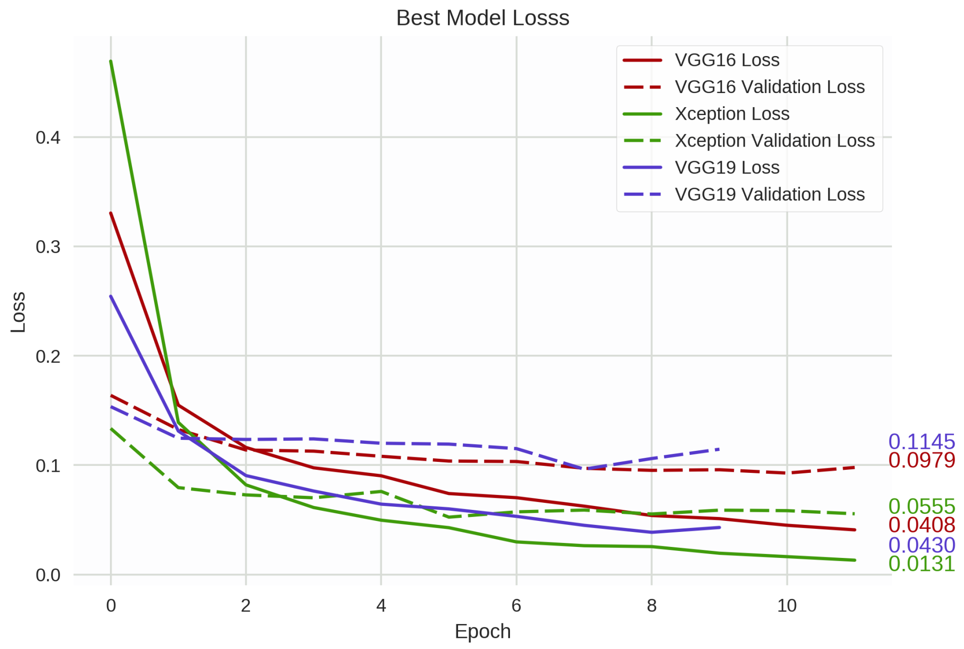


Figure 3: Best Model Losses

5.1 Optimizer

Optimizer	Adam	SGD	RMSprop
VGG16	0.97425	**0.97426**	0.97204
VGG19	**0.97572**	0.97278	0.97242
Xception	0.93159	**0.98675**	0.95476

We used the default settings for each optimizer. The best performing optimizer was found to be the SGD optimizer. This was not expected given that the Adam optimizer has had much recent publications regarding its easier convergence. [9]. This doesn't mean that Adam performs better at optimization; rather, it means that it generally requires less hyperparameter tuning to reach a decent endpoint.

Note that when considering which optimizer is the best, convergence speed is generally taken into account. As can be seen through 2 and 3, VGG19 converged slightly faster when using the Adam optimizer. This makes sense since SGD doesn't take much other than the gradient into account when performing optimization, whereas Adam has several speed up mechanics to speed up convergence.

5.2 Learning Rate

Learning rate	0.1	0.01	0.001	0.0001
VGG16	0.96800	**0.97426**	0.96947	0.95733
VGG19	0.79772	0.96175	**0.96837**	0.96800
Xception	0.97535	**0.98675**	0.98197	0.97315

The best learning rate was 0.01. Any higher learning rates were too large to converge to a good minima while smaller learning rates likely converged into suboptimal minimum.

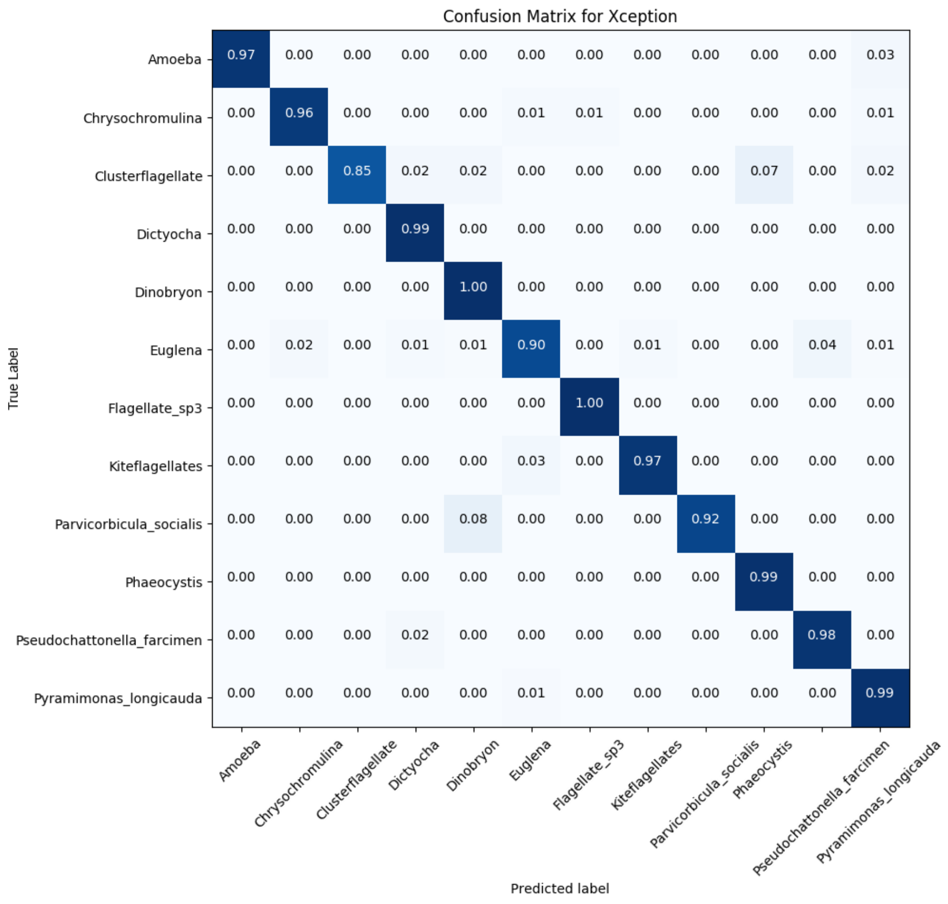
5.3 Classification Layer

Classification Layer	Type 1	Type 2	Type 3
VGG16	0.96469	0.82898	**0.97278**
VGG19	0.95550	0.44796	**0.97242**
Xception	**0.98675**	0.98161	**0.98675**

The third type of classification layer performed the best. In this case, it seems as though using a larger hidden layer count resulted in the creation of minima that the network could not get out off. Interestingly, Xception performed just as well using the type 1 and type 3 classification layers. This is surprising considering that the global average pooling layer in the first type of classification layer removes a bit of information before the fully-connected layers.

5.4 Confusion Matrix

The confusion matrix gives a good insight as to where the network may have had trouble learning larger representations. The following shows the class-based performance for the best performing Xception model:



True Label	Amoeba	Chrysochromulina	Clusterflagellate	Dictyocha	Dinobryon	Euglena	Flagellate_sp3	Kiteflagellates	Parvicorbicula_socialis	Phaeocystis	Pseudochattonella_farcimen	Pyramimonas_longicauda
Amoeba	0.97	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03
Chrysochromulina	0.00	0.96	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.01
Clusterflagellate	0.00	0.00	0.85	0.02	0.02	0.00	0.00	0.00	0.00	0.07	0.00	0.02
Dictyocha	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Dinobryon	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Euglena	0.00	0.02	0.00	0.01	0.01	0.90	0.00	0.01	0.00	0.00	0.04	0.01
Flagellate_sp3	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
Kiteflagellates	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.97	0.00	0.00	0.00	0.00
Parvicorbicula_socialis	0.00	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.92	0.00	0.00	0.00
Phaeocystis	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00
Pseudochattonella_farcimen	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00
Pyramimonas_longicauda	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.99

Figure 4: Xception Model Confusion Matrix

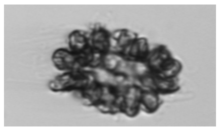


Figure 5: Clusterflagellate

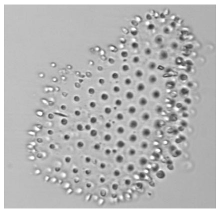


Figure 6: Parvicorbicula

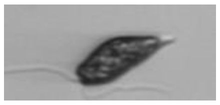


Figure 7: Euglena

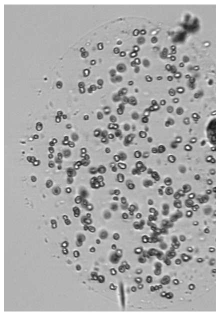


Figure 8: Phaeocystis

From 4, we can see that the neural network performed the worst on the Clusterflagellate and Euglena classes. In the first case, it most often misclassified a Clusterflagellate as a Parvicorbicula Socialis. In the second case, it most often misclassified the Euglena as a Phaeocystis. Using the above images 5, 6, 7, and 8, we can see why the network may have made misclassified these classes.

Notice that 5 and 6 share the property that they are present in clusters. Further notice that 6 is substantially larger than 5. It's possible that the resizing of the images caused the network to incorrectly view some properties of either class. Furthermore, considering how there are a few images with a single Clusterflagellate, it's possible that it viewed the single flagellate as part of a larger cluster (which Parvicorbicula are often pictured in).

Notice that 7 and 8 exhibit the same issues as the above two classes. Since Euglena are often depicted as single flagellates, it's possible that the network trained itself to recognize the appearance of individual Phaeocystis. As a result, it mistakenly classified the single Euglena as a part of a larger cluster of Phaeocystis.

6 Future Work

Looking forward, we want to attempt to classify the entire dataset with our model. As mentioned earlier, the class-imbalance problem will be a difficult problem to address. We can possibly solve this problem by limiting the maximum number of images in each class by a threshold N as mentioned in [5]. By changing N, we can fine tune our model to achieve a higher accuracy. We can further improve our model by applying some kind of normalization on the dataset before training which can help the model become more scale invariant[8].

Throughout this particular classification, it's clear that the images were not clearly sized to be similar to each other. It's possible that the network developed some sort of size dependence on the images. In order to combat this, it would have been preferable to augment the data so as to prevent the learning of object sizes (since they are all magnified different magnitudes). This could have been easily implemented through a 10-cut method when loading in the data.

7 Conclusion

The Xception architecture performed the best among the two VGG variants. This gain in performance is likely attributed to the magnitude of parameters that the Xception network had available for training. Furthermore, given that Xception is based off of the Inception module, it's not surprising that it may be better suited for larger feature recognition.

The best performing optimizer was SGD with a learning rate of 0.01. This is surprising considering Adam has been shown to perform better on most networks [9]. This can possibly be attributed to the lack of extensive hyperparameter testing in this paper's simplified parameter sweep.

Finally, from the classification layers tested, the third layer type $[Flatten \rightarrow Dense(512) \rightarrow Dropout(0.2) \rightarrow Dense(12)]$ performed the best on all three networks. It's possible that not enough iterations were carried out in order to generalize to the higher parameter count, but it's more likely due to the presence of a local minimum when optimizing with the higher hidden neuron count.

References

[1] *National Data Science Bowl*. https://www.kaggle.com/c/datasciencebowl, 2015.

[2] Francois Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*, 2017.

[3] Emily E. Peacock Eric C. Orenstein, Oscar Beijbom and Heidi M. Sosik. *WHOI-Plankton- A Large Scale Fine Grained Visual Recognition Benchmark Dataset for Plankton Classification*. https://arxiv.org/pdf/1510.00745.pdf, 2015.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[5] Minseok Park Hansang Lee and Junmo Kim. *Plankton Classification On Imbalanced Large Scale Database Via Convolutional Neural Networks With Transfer Learning*. https://ieeexplore.ieee.org/abstract/document/7533053/, 2016.

[6] Emily F. Brownlee Heidi M. Sosik, Emily E. Peacock. *Annotated Plankton Images - Data Set for Developing and Evaluating Classification Methods*. https://darchive.mblwhoilibrary.org/handle/1912/7341, 2006-2014.

[7] Geoffrey Hinton. *Overview of Mini-batch Gradient Descent* . http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[8] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. https://arxiv.org/pdf/1502.03167.pdf, 2015.

[9] Diederik P. Kingma and Jimmy Lei Ba. *Adam: A Method For Stochastic Optimization*. https://arxiv.org/pdf/1412.6980.pdf, 2015.

[10] Sebastian Ruder. *An Overview of Gradient Descent Optimization Algorithms*. https://arxiv.org/abs/1609.04747, 2017.

[11] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks For Large-Scale Image Recognition*. https://arxiv.org/pdf/1409.1556.pdf, 2015.

[12] Anish Singh Walia. *Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent*. https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f, 2017.