Exploration of CNN Phytoplankton Classification Model

Hisham Bhatti

January 22, 2023

Abstract

Phytoplankton are a pivotal component of the planet's overall health. Machine Learning, and specifically Convolutional Neural Networks (CNNs), has become an effective technique to learn to classify species of plankton from images. Our program trains a CNN using 1000 images each for five phytoplankton taxonomic groups, achieving roughly 85% accuracy after optimization. An important concept during training is to balance overfitting and underfitting, so that the network has enough training to perform well, but not excessively repeating the same dataset at the expense of classifying new images reliably. We found that smaller batch sizes (4-8) resulted in better performance, but adjusting the original layers worsened the network compared to 32, and 64 convolutional layers. A multitude of factors (human mislabelings, sufficient data, distribution of classes) must be considered when evaluating the network in a real-world setting. But even so, this UTOPIA "five-groups" project aims to help people of all experience levels get interested and invested in machine learning, given its transformative impact on big-data in all fields.

1 Introduction

Phytoplankton are microscopic plants that are present in the whole ocean. Through photosynthesis, phytoplankton generate roughly half of the planet's total oxygen supply, while removing carbon dioxide from the ocean's surface. As such, they play a critical role in the health of the global ecosystem.

Identifying what type of phytoplankton live in certain regions gives researchers important data on the state of the planet. Phytoplankton thrive on nutrient-rich cold ocean water; rising ocean temperatures may threaten phytoplankton communities and damage marine ecosystems. Since ocean color is generally correlated with certain plankton quantities, prior research has studied the color of water from satellite data. But as climate change accelerates changes in our oceans, researchers are looking for ways to study different phytoplankton groups on large scales, and this requires a reliable method of classification.

Machine Learning (ML) is a tool used by scientists in which a computer systematically "learns" from data presented to it to make important decisions. To detect and categorize phytoplankton, we use a technique called image classification, in which a computer learns to recognize certain low-level features of images, and then combines them together to give a prediction on the class of image.

Underneath this classification algorithm lies a web of decisions called a neural network. Information is fed from the start of the network, broken down into nodes of information, passed through layer after layer using complex calculations, which ends at the output layer to provide us with the information we need (Figure 1). Initially the series of complex calculations is randomized. As we feed in hundreds or thousands of images and their human-confirmed categorizations, the network tunes its calculations in the middle layers to provide accurate conclusions on its own.

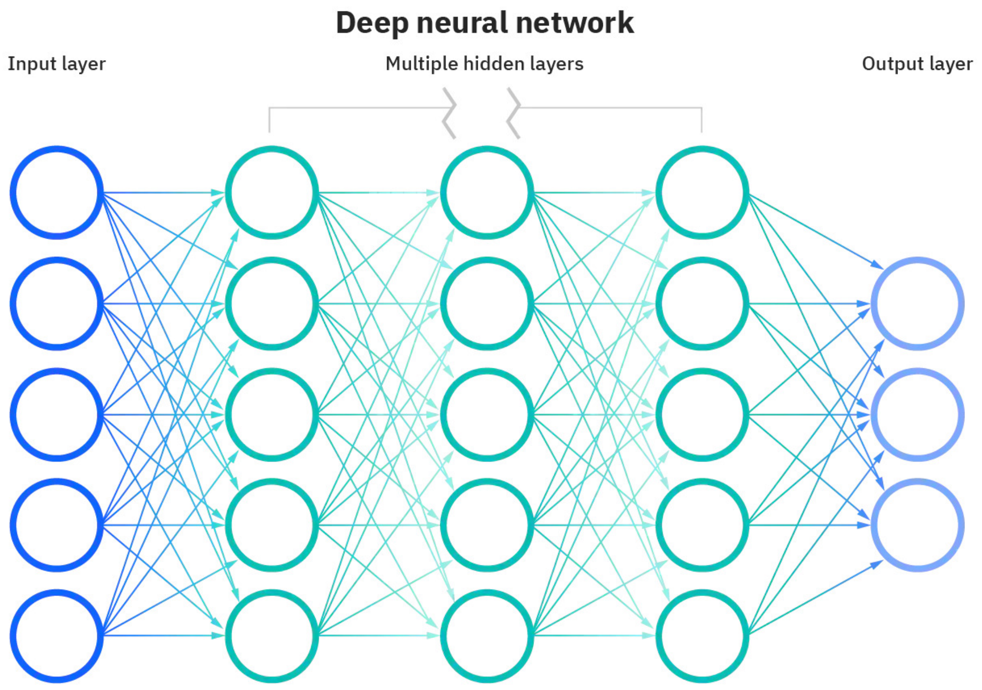


Figure 1: Example of a Simple Densely-Connected Network

The model I built is derived from the simple neural network above, called a Convolutional Neural Network (CNN). CNNs start with a multidimensional input space, which must be processed down into a single dimension of nodes. Abstractly, we encode the neural network to recognize low-level "features" of images, and then bring them together to create high-level features of a plankton.

We use the mathematical operation of a convolution to apply different filters to images that extract different characteristics. Mathematically, a convolution is a dot product between one large matrix and one smaller "filter" matrix. We slide the filter matrix through the larger matrix, multiply numbers that line up, sum them together, and put the result into a new matrix.

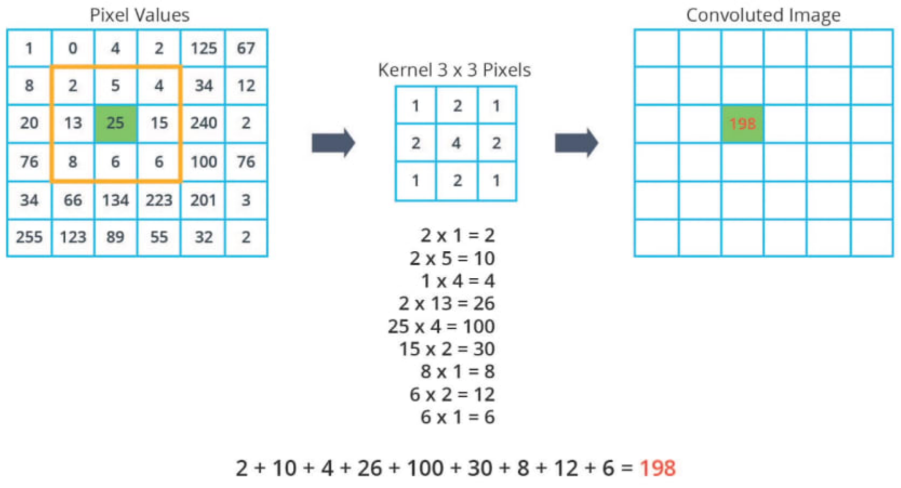


Figure 2: Convolution with a Sample Kernel

In a CNN, the network itself learns the types of filters that can best guarantee accuracy and minimize loss, and creates a series of results from different filters. These filters are then down-scaled using a pooling function, which takes the maximum element from a grid to downscale the dimensions of the filters. This process of convolution and pooling is repeated a few times, and is then flattened to be processed in a direct neural network.

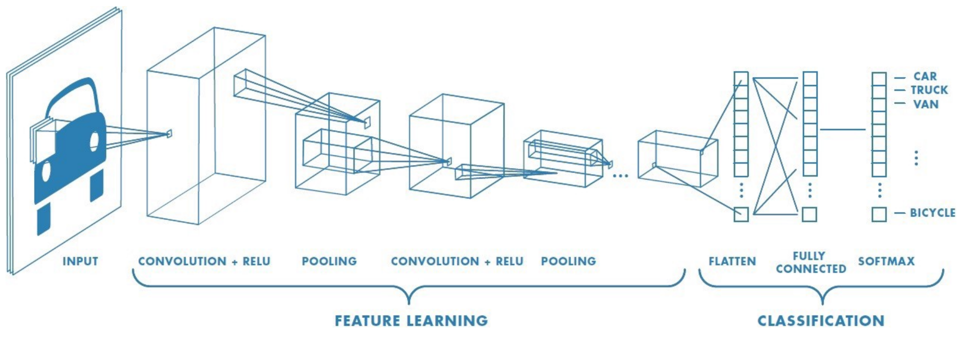
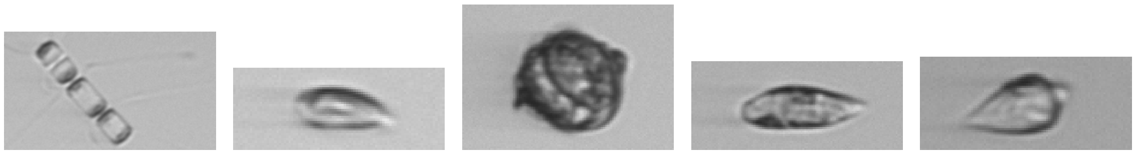


Figure 3: Convolution with a Sample Kernel

This model for phytoplankton detection uses a simple CNN constructed using Keras, an open-source library built from TensorFlow. We have 5 classes of phytoplankton: chaetoceros (chae), dinoflagellates (dino), euglenoids (eug), cryptophytes (crypt), oxytoxum (oxy), each with 1000 images of manually-labeled data (Figure 4). After optimizing certain parameters in the network, we achieved 85% accuracy on tested data.



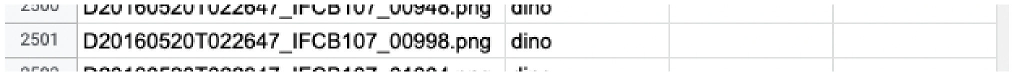
(a) Chae (b) Crypt (c) Dino (d) Eug (e) Oxy

Figure 4: Examples of our 5 Types of Phytoplankton

2 Methods/Data

We started with 1000 images each of the five types of phytoplankton. Before we could feed the images into the network, all 5000 had to be padded and processed.

Consider a random Dinoflagellate image as an example. Each image in the dataset has a specific label to identify it, and in this case our image was: D20160520T022647_IFCB107_00998. First, our program had to find the manually-labeled name of the species in the metadata. Each row in the metadata corresponds to an image in the dataset, with two columns: one with an image ID and the other with the label.

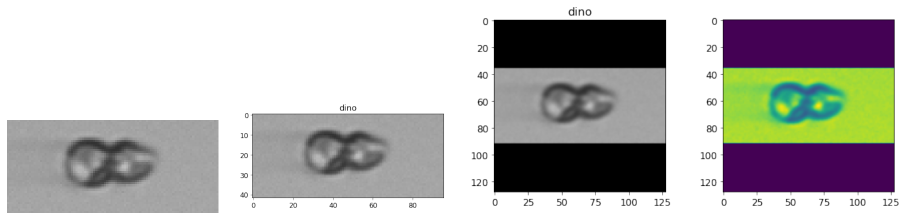


2500	D20160520T022647_IFCB107_00948.png	dino		
2501	D20160520T022647_IFCB107_00998.png	dino		
[illegible]	[illegible]	[illegible]		

Figure 5: Example Dinoflagellate in the Metadata

After finding the label as "dino," the program can begin processing the image. Each image has different pixel dimensions, so the program first aligns them bottom-right on a 2D plane. Next, the image is scaled and padded with black pixels on the sides so that all images are 128 x 128 black and white.

Note: For visualizing the data, we remove a third color dimension from the data. Images are thus viewed as a heatmap of arbitrary colors.



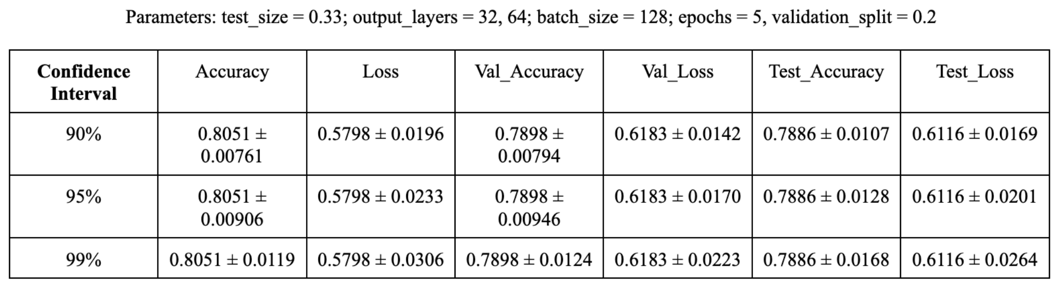
(a) Raw Image (b) Measured (c) Scaled & Padded (d) Heatmap

Figure 6: Processing of a Sample Dinoflagellate

Once the image is squared and padded, we begin defining the network. For our first run, the network has two convolution and pooling operations, with 32 and 64 layers respectively. Then, we flatten the input space and pass it off as a simple directly-connected neural network. The 5000 images were split into testing and training data with a ratio of 0.33, so that 3350 images were training and 1650 images were testing. We fed the 3350 images five times (epochs) with a batch size of 128 images each before tuning attributes of the network. 20% of the 3350 images were used as validation after each epoch, to see how the network improves over each epoch.

3 Initial Network Evaluation

After running the initial network multiple times (n = 8), we achieved confidence intervals for the following results (Figure 7).

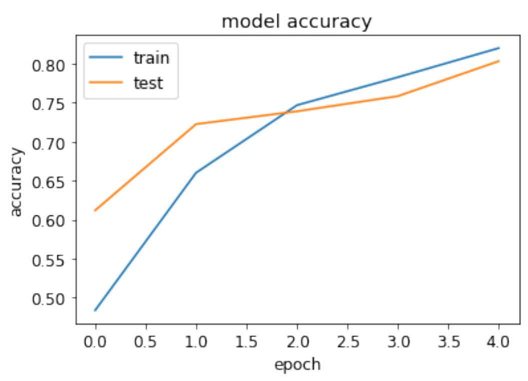


Parameters: test_size = 0.33; output_layers = 32, 64; batch_size = 128; epochs = 5, validation_split = 0.2

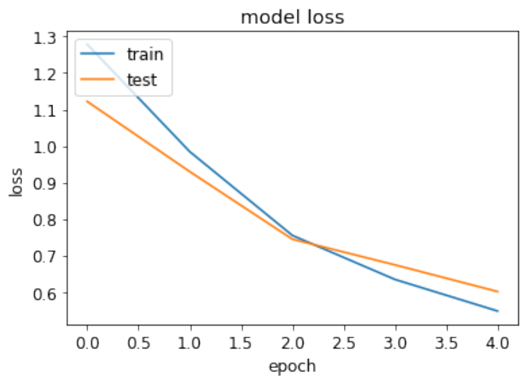
Confidence Interval	Accuracy	Loss	Val_Accuracy	Val_Loss	Test_Accuracy	Test_Loss
90%	0.8051 ± 0.00761	0.5798 ± 0.0196	0.7898 ± 0.00794	0.6183 ± 0.0142	0.7886 ± 0.0107	0.6116 ± 0.0169
95%	0.8051 ± 0.00906	0.5798 ± 0.0233	0.7898 ± 0.00946	0.6183 ± 0.0170	0.7886 ± 0.0128	0.6116 ± 0.0201
99%	0.8051 ± 0.0119	0.5798 ± 0.0306	0.7898 ± 0.0124	0.6183 ± 0.0223	0.7886 ± 0.0168	0.6116 ± 0.0264

Figure 7: Performance of CNN with default parameters (n=8)

The network, as predicted, rose in accuracy and decreased in loss for the validation sets throughout training (Figure 8). Since validation is conducted at the end of each epoch, for the first couple epochs the testing data has a greater accuracy than the training through the network, representing a network underfitting, or needing more data. Eventually, the training surpasses and follows the testing data, corresponding to a greater tuned network and perhaps even overfitting. The same principle applies to the loss function: training loss starts above testing loss but eventually dips below testing.



(a) Accuracy



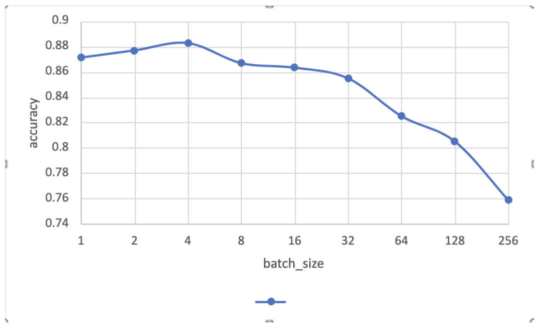
(b) Loss

Figure 8: Accuracy & Loss for the CNN per Epoch

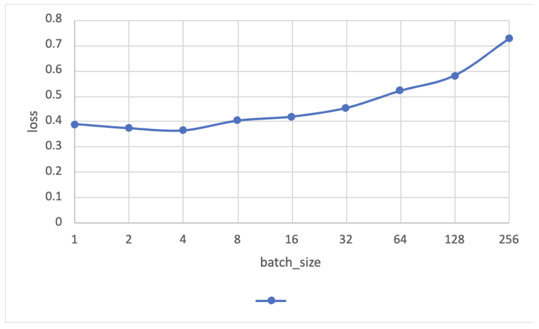
4 Exploration of Networks & Broad Outlook

With the network properly run and producing results, the next question became if we could somehow improve its performance by adjusting parameters in the program. I chose to adjust two different criteria when discussing the network's performance: properties of the data processing, and properties of the network itself.

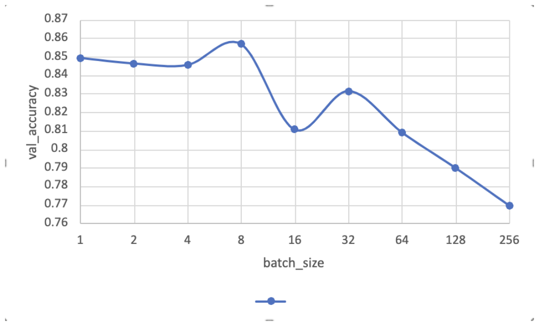
The first modification I made had to do with the batch_size. The general best practice in the realm of data science is to choose the greatest batch size possible that still sufficiently adjusts the network. Larger batch sizes improve the network's speed since the parameters of the network are tuned less often. 128 images for a size seemed arbitrary, so I ran the program with different batch sizes.



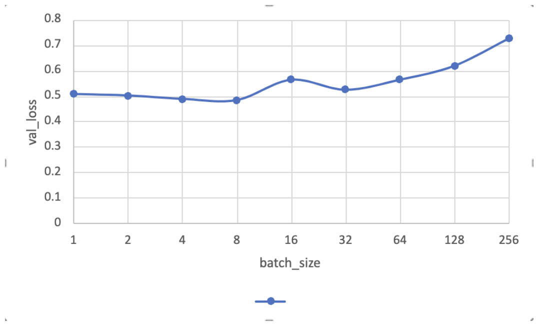
(a) Accuracy



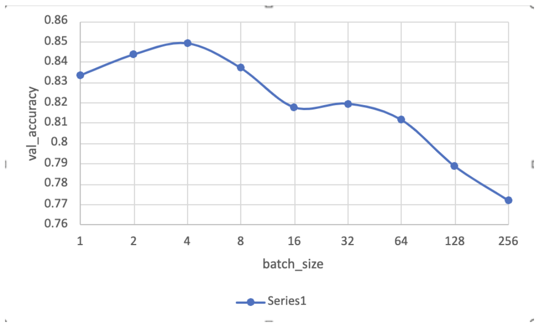
(b) Loss



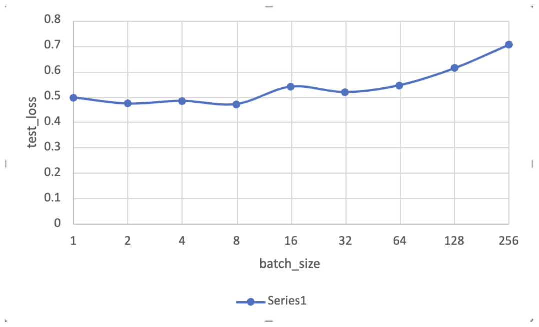
(c) Val_Accuracy



(d) Val_Loss



(e) Test_Accuracy



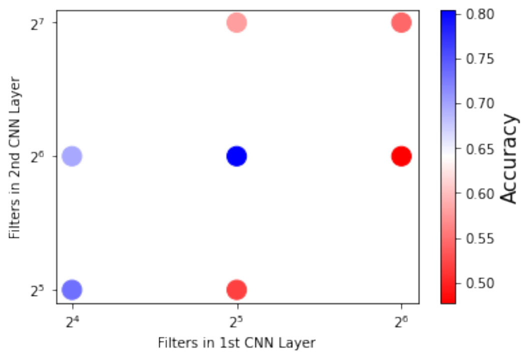
(f) Test_Loss

Figure 9: Performance of CNN vs. Batch_Size

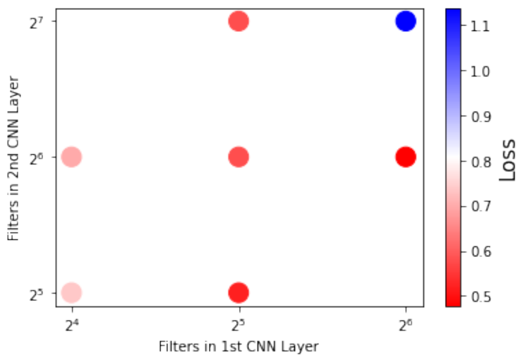
For this network it seemed that smaller batch sizes worked better than the standard batch size of 128. With the exception of batch size 16, accuracy tended to peak around 4-8 images, and then decreased slightly with smaller sizes. Loss was also minimized at roughly 4 or 8 images, but then increased dramatically as the batch size doubled.

For batch sizes smaller than the standard 128, there seemed to be a bit more overfitting. The difference between training and testing measurements widened slightly, but as both training and testing increased, smaller batch sizes seemed to help the network. Even with the batch size of 1, where the network adjusts its weights/biases after every image, overfitting didn't seem like a significant problem. On the other hand, a larger batch size than 128 clearly hurts the network on all fronts. Both the accuracy decreases and loss increases, with the testing data slightly better than the training data. Underfitting clearly seems to persist here; the network simply didn't have enough chances to adjust its parameters to fit the data. The network just can't catch up to the data, and it performs objectively worse. If this same network was to be fed millions of images, and time became an issue, I would argue that the "proper" batch size for this large dataset would be 8. It's performance is closest to the peak on all fronts, and doesn't require as much adjustment as 4, for not as much gain.

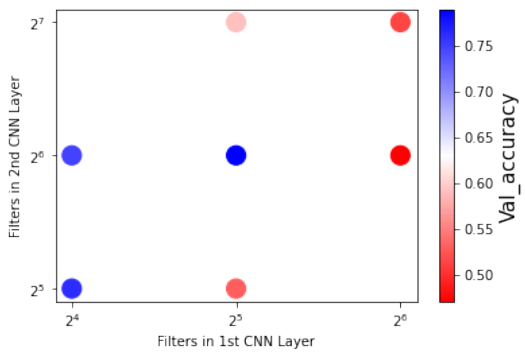
The second adjustment to the network involved altering the internal architecture of the CNN by changing the number of layers at each pooling point. After modifying both dimensions of the layers, I discovered that the optimal dimensions of layers seems to be the original, at 32/64. 32 and 64 balance the demands of over-extrapolating features of images that come with a high number of layers, as well as under-extrapolating features that come with too few layers.



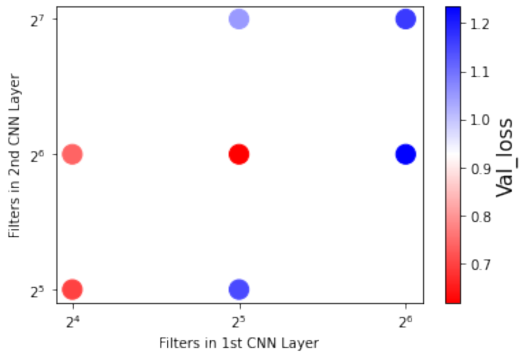
(a) Accuracy



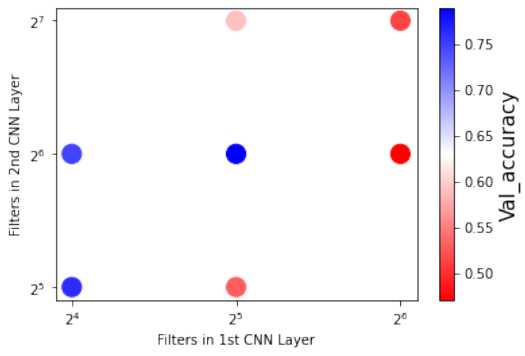
(b) Loss



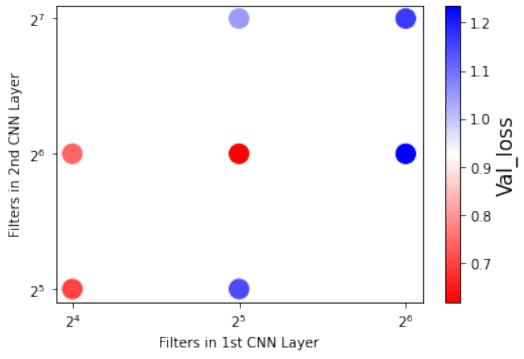
(c) Val_Accuracy



(d) Val_Loss



(e) Test_Accuracy



(f) Test_Loss

Figure 10: Performance of CNN vs. Pooling Layers

As the number of layers increased, the performance of the network worsened dramatically; both the accuracy and loss suffered. Above 32/64, it seems abstractly like the network is "overfitting" to the features identifiable in the images. More filter maps signals to the network to look for even more features in the data; perhaps the features it finds are too "low-level" to be generalized to all unknown images of phytoplankton in the testing set. The performance of testing was below training, so perhaps we overlearned the qualities present in the training set.

For layers less than 16 and 32, underfitting seems to be the issue. Though not nearly as strong as overfitting, the accuracy decreased for layers less than 16 and 32. This perhaps signals that the network is not really learning enough from the data, or the features are too "high-level" that they blur between species and cause more mistakes. Even though the accuracy decreases a bit, the increased speed of the network might be worth justifying its use. Perhaps for millions of images the slightly decreased accuracy would be worth the significant drop in training time.

Beyond the structure of the network and processing of data, there are possibly other culprits to blame for the low performance. One strong one is human error in labeling. Labeling thousands of images takes time, and is not an easy task for a human let alone a network. There are bound to be errors in mislabeling for networks of large datasets. Oftentimes the network actually classifies the data correctly, but the incorrect human label causes a mistake and decreased accuracy.

Another key idea worth mentioning is that the entire design of this project simulates ideal data. We have clear-cut categories, with large distinctions between only a few species. We have sufficient images, 1000 for each of the five species. And as mentioned before we have relatively correct labeling, as this data has been cherry-picked from different expeditions to be the clearest, more verified images taken of these phytoplankton. Though that helps in training, "perfect" data is not accurate to samples collected in the real-world.

Before creating this project with only five species of 1000 images each, my initial project began with 12 types of phytoplankton, with seemingly thousands of images to spare. But after eliminating images that weren't labeled, and throwing out blurry or corrupt pictures, we only had 75 images to work with. Clearly not enough to train a complex network.

The proportion of evenness also factors into the reliability and accessibility of the network. Having each category roughly evenly labeled with images might be helpful to the model, but is not at all accurate to the distribution of plankton in the oceans. There are over 20,000 species of phytoplankton, some of which are likely yet to be discovered (and hence even possibly labeled). The relative difference in number of each phytoplankton population poses questions about how to deal with the distribution. For example, if we were to try to label all of many different species, some uncommon plankton might have only 3 images which is not enough to train, but a really common one might have 1000, which is enough but maybe not proportionate. Would it be better for a model to possibly misread all 3 plankton for the entire species, since it doesn't affect overall accuracy much? Or is it better to try to get a strong accuracy for each *type* of plankton, even if that means the overall accuracy (# of mislabels) might be worse. Questions like these have no good answer, as it depends on what the researcher is prioritizing in their classification.

5 Impact for Oceanography & Education

With a semi-reliable image classification of phytoplankton comes possible solutions for problems facing oceanographers, students, and researchers in general; this was a large goal for the project other than just the program.

As networks and communication have connected people all across the globe over the past years, researchers have become less isolated with their own pursuits. Publishing results on the web can foster collaboration among the global workforce as a whole, catalyzing the rate of scientific advancements. One of the big goals of this project was bringing a small part of the open-science movement to oceanography, allowing plankton researchers to discuss, publish, and build off of their methods of data collection and analysis.

UTOPIA (User-friendly Tools for Oceanic Plankton Image Analysis) is a platform and movement designed for just that. After publishing this script onto their open-source Github platform, researchers now have an online prototype for how to label phytoplankton in a simple CNN. We're not at the final stage yet, where we hope to have researchers and students learn off of each other, but my simple network promises to be a first step towards that.

Another goal right from the start was providing the tools and resources to get younger, inexperienced students passionate about oceanography and machine learning in general. Machine learning has a reputation among students in undergrad for being too challenging, too inaccessible, or too removed from their field (if not in something like computer science or statistics). As such, many students interested in science or even humanities are scared to continue where machine learning is transforming their industries.

However, it's precisely for that reason, that artificial intelligence is revolutionizing the sciences and humanities, that being passionate about machine learning is so critical for research. While it may be daunting writing a complex neural network from scratch, built-in libraries like TensorFlow and Keras make writing networks quick and painless. And this program serves as a great starter-point to understand the basics behind ML.

All the code written in the program uses libraries, so the reader doesn't need to understand complex mathematics and statistics to see the big-picture. The code is also well-commented, so students can clearly understand all the processes of the program (file processing, image processing, building network, training, testing) in sequence. Image-processing is generally the standard introduction to the power of deep learning, and the phytoplankton example is simple, and interesting because of its implications for science.

This project was as much a learning experience for me and it can be for everyone. For professional oceanographers, who are encouraged to collaborate after seeing UTOPIA's open-source platform. For teachers, who can use this simple script as a great example for ML's power to transform the sciences, even at a basic level. And for students. I learned a lot about data science and its promise to transform the sciences, so can you by reading, running, and testing the program on your own.

6 References

Note: Check out the "Resources" page on the UTOPIA Github to learn more

UTOPIA:

- Github (has all the files to test the CNN yourself)
- APL
- UW eScience Institute

Machine Learning Resources/Tutorials:

- 3Blue1Brown Intro to ML
- MIT Intro to Deep Learning (2022)
- Simple MNIST Digit Recognition Model on Keras

Image Links:

- Figure 1
- Figure 2
- Figure 3