Classification of Plankton using Deep Neural Networks

Suman Shekhar, Vishnu Sri Ranjan DR, Smit Patel, Yasasvi Keerthi, Sravya Kanagala
Muhammad Talha Afzal

1. Introduction

Plankton are microorganisms integral to marine ecosystems. The term 'plankton' derives from the Greek for "wanderer" or "drifter," describing organisms that lack the ability to swim against the current. Due to their buoyancy, plankton often float near the surface. Some possess chlorophyll, enabling them to perform photosynthesis—converting atmospheric carbon dioxide into organic matter. This organic matter enters the marine food web and can eventually be sequestered in the deep ocean. Remarkably, the carbon sequestered by this mechanism, known as the "biological pump," can remain in the ocean's depths for millennia. Among plankton, those that contribute to carbon fixation are of particular interest due to their role in the global climate system.

Researchers prioritize studying the carbon flux associated with phytoplankton to understand their impact on climate dynamics. Such studies require accurate assessments of plankton populations, including counts, size measurements, and species identification (taxonomy). However, these tasks are meticulous and time-intensive when performed manually.

To expedite this process, oceanographers have increasingly turned to technological solutions. With the proliferation of images, there is a growing imperative to automate classification, data extraction, and analysis. Machine learning methods, especially Convolutional Neural Networks (CNNs), have been employed to achieve cutting-edge results. Our project commences with an evaluation of the VGG16 architecture—a CNN model—and its application to plankton classification. We aim to refine the model's accuracy by integrating additional layers and applying feature engineering techniques.

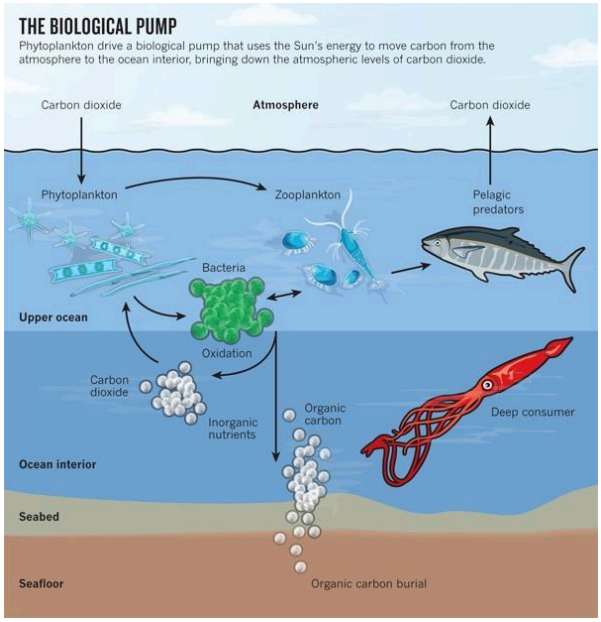


Figure 1: Marine Ecosystem

For this study, we utilized a subset of the plankton dataset from the Woods Hole Oceanographic Institution. The dataset, amassed with an in-situ automated submersible imaging flow cytometer (IFCB), encompasses over 103 species and 3.5 million images. It is meticulously organized by collection year, facilitating extensive analysis.

2. Model Architecture

The VGG16 model, short for Visual Geometry Group 16, is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group at the University of Oxford. We used this model because of its code familiarity and easy to get started.
The architecture of VGG16 is characterized by its depth, with 16 layers that have weights, including 13 convolutional layers and 3 fully connected layers. The model uses small 3x3 convolutional filters and max-pooling layers for feature extraction, followed by fully connected layers for classification. The simplicity and effectiveness of the VGG16 architecture have

made it a popular choice for image recognition tasks.

The input to the VGG16 model is a fixed-size 224 x 224 RGB image. The image is passed through a stack of convolutional layers, where the filters are used. The convolution stride is fixed to 1 pixel; the spatial padding of the convolutional layer input is such that the spatial resolution is preserved after convolution, i.e., the padding is 1 pixel for 3x3 convolutional layers.

Max-pooling is performed over a 2x2 pixel window, with stride 2. Three fully connected (FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. All hidden layers are equipped with the rectification (ReLU) to capture nonlinearity in the dataset. To make it suitable for our project we changed the number of classifiers in the last layer to 10.

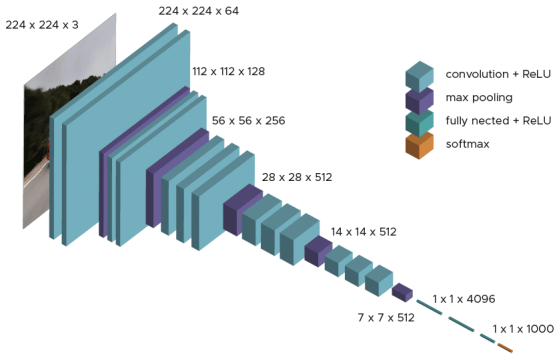


Figure 2: Visualization of VGG16

3. Software and Hardware Implementation

In our project, we used an array of software tools and hardware platforms to develop, train, and evaluate our neural network models. Our implementation was primarily executed using Python, leveraging libraries such as NumPy for efficient numerical operations and PyTorch for building and training deep learning models. These tools were essential for handling the complex computations and data structures required in our work.

To enhance the functionality of our models, we experimented with the development of several custom kernels. This involved writing specialized code to optimize certain operations, which was pivotal in achieving desired computational efficiencies and model performance.

For visual representation and analysis of our results, we employed the Matplotlib and Seaborn libraries. These tools were instrumental in creating a wide range of visualizations, from standard plots to more complex graphical representations, which facilitated a deeper understanding of the data trends and model behaviors. These visualizations were crucial for our preprocessing and data augmentation strategies, allowing us to refine our approaches based on visual feedback.

Throughout the project, meticulous version control of our Jupyter notebooks was maintained. This practice was vital for tracking progress and changes, ensuring that every modification was documented and retrievable, which proved invaluable for collaborative debugging and iterative improvements.

Our models underwent training across various environments to capitalize on different hardware configurations. This included local machines, Google Collab, the Rutgers Amarel Cluster (Rutgers high performance computing center), and Kaggle. By employing a mix of CPUs and GPUs, we optimized computational resources to handle intensive training sessions. Notably, Kaggle provided access to NVIDIA Tesla V100 GPUs and TPU VM v3-8, which significantly enhanced our processing capabilities.

Despite initial attempts to utilize the Tensor Processing Unit (TPU) for its superior processing power, we encountered several challenges that hindered its integration into our workflow. Consequently, we opted to proceed with NVIDIA's CUDA technology, which was more compatible with our existing frameworks and provided the necessary stability and performance for our extensive training requirements.

4. Preprocessing and Data Augmentation

4.1. Preprocessing

Our pre-processing phase was meticulously planned and executed, although the initial steps involved minimal intervention. Due to the nature of the IFCB plankton dataset, which comprises images from various years, we had to navigate through a significant challenge: the seasonal availability of some species, while others were consistently present. This required the comprehensive downloading and aggregation of multiple years' data. Each species' images were meticulously categorized into their respective classes within designated folders—a process that proved both time-consuming and daunting.

To streamline file management, we developed a script to automate the organization and combination of image files according to their labels. This automation was crucial in handling vast amounts of data efficiently. Additionally, the script facilitated the removal of extraneous log and database files that were not necessary for our project.

After consolidating data from various years, the dataset size expanded to over 10 GB. During a manual inspection of the images, we identified that some older images were of suboptimal quality, necessitating their removal to ensure the integrity and consistency of the data used for training.

Given the massive size of the dataset and our limited computational resources, we faced significant challenges. The dataset exhibited pronounced class imbalance, with some species represented more abundantly than others. To mitigate this, we implemented a strategy to extract only those labels with instances ranging between 300 and 500. From the myriad of class labels, we selected ten at random for inclusion in the training dataset, hindered by our computational limitations.

Our pre-processing also involved selectively retaining images in JPG and PNG formats to ensure compatibility and efficiency during model training. Besides basic organizational and cleaning steps, we rescaled the images due to their varied sizes—from as small as 40x40 pixels to as large as 400x400 pixels. We rescaled it to 225x225 as its required as an image size to feed into VGG16.

4.2. Data Augmentation

Data augmentation is a pivotal technique in enhancing machine learning model performance, especially in image recognition. This approach artificially expands the training dataset by generating new data points through various transformations that preserve the original labels.
The primary objective of applying data augmentation to the IFCB plankton dataset is to bolster the robustness of convolutional neural networks (CNNs) by introducing them to a wider array of image variations. This strategy was intended to improve model generalization, enabling better performance on new, unseen images while maintaining high accuracy and reliability. We had two experiments, the first had no augmentation while the second had augmentation.
Some of the benefits of data augmentation are follow:

Enhanced Model Generalization: By exposing the model to a broader spectrum of data variations, it is less prone to overfit specific attributes of the training images.

Expansion of Dataset Size: Augmentation effectively increases the number of training images, beneficial in contexts where data acquisition is costly or logistically challenging.

Simulation of Real-World Variations: Through transformations, models can adapt to changes in orientation, lighting, and other environmental influences that occur in natural plankton habitats.

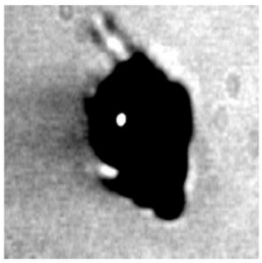
Despite the theoretical benefits, our practical experiences revealed several challenges with data augmentation on this dataset. Initially, we implemented standard augmentation techniques but observed suboptimal model performance, leading us to adjust the intensity and combination of augmentations progressively. Changes in model architecture,

such as modifying dropout rates, were also necessary to counter overfitting. Below we discuss specific techniques used and their impacts:

1. Image Size in Pixels: We standardized all images to 224x224 pixels to align with the input requirements of VGG16 and ResNet architectures. Although standardization is crucial for CNNs, which require consistent input sizes, this adjustment resulted in suboptimal model performance. The resizing process potentially compromised key features of the plankton species, which are critical for accurate classification.

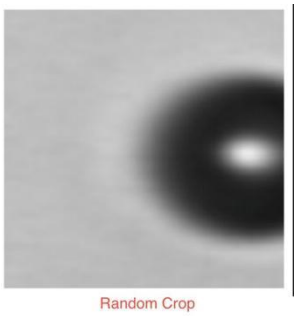
2. Size and Shape Estimates Based on Image Moments: By utilizing image moments, we extracted size and shape features crucial for classifying organisms with subtle variations in form and structure. However, this method alone was insufficient, as the model continued to perform sub optimally, struggling to accurately distinguish between similar species.

3. Normalization: We experimented with models both with and without normalization of pixel values. Normalizing the images to a scale of 0 to 1 generally enhanced model training stability and convergence speed. Nonetheless, the diverse range of contrast across images collected over various years complicated the application of a uniform mean and variance, leading to mixed results in model accuracy.



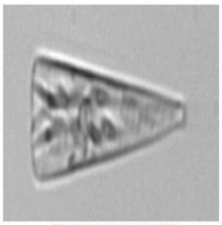
Normalization

4. Resize and Crop: Resizing and cropping are typically employed to focus the model's attention on the main subjects of the images. In our case, however, these techniques did not yield the expected benefits. Resizing blurred distinguishing features of the plankton, while cropping, especially random cropping, occasionally removed crucial parts of the specimens. Both adjustments contributed to a significant drop in model performance.

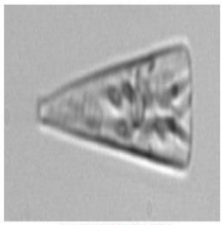


Random Crop

5. Horizontal and Vertical Flip: Flipping images horizontally or vertically was intended to introduce variability and help the model learn to recognize plankton from different orientations. This approach, however, confused the model due to the similar appearances of different species when flipped, undermining its ability to classify accurately.



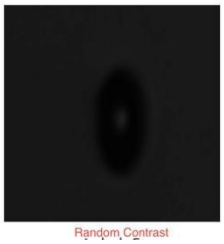
Random Horizontal Flip



Random Vertical Flip

6. Adjusting Brightness and Contrast: Adjusting the brightness and contrast of images was necessary to maintain visibility under varied lighting conditions, a common issue in practical applications. Despite this, the adjustments sometimes resulted in the loss of small, intricate details essential for the model's learning and predictive accuracy.

Random Brightness



Random Contrast

Brightness + Contrast

7. Noise Injection: Injecting noise, such as Gaussian noise, aimed to enhance the robustness of the model by mimicking real-world imperfections. Although this method is known to increase generalization, it proved detrimental in our context. The small size of plankton meant that added noise often masked critical details, decreasing model effectiveness.

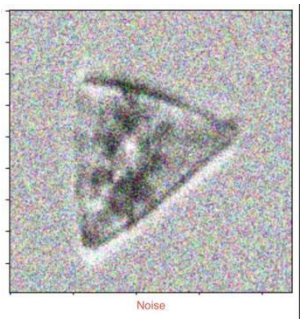
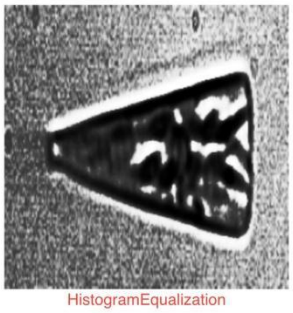


Figure 3: ADAM accuracy/Loss vs iteration

8. Random Stretch: Random stretching was used to help the model handle variations in plankton size and aspect ratios across different samples. This technique did not negatively impact the model's performance but also did not lead to noticeable improvements.

9. Histogram Equalization: We applied histogram equalization to enhance the visibility of details in images, particularly in overly bright or dark areas. While this method generally helps reveal subtle textures and structures, it did not enhance the accuracy of our model, suggesting that the current dataset and model configuration might be unsuitable for this type of processing.



HistogramEqualization

5. Validation and Training

5.1. Validation

In the validation phase, 20% of the labeled data was designated as the validation set, derived through a random division of the total dataset. This method ensures that the validation set accurately represents the overall dataset, which is essential for a reliable assessment of model performance. Despite this, the limited size of the validation set introduced variability, or 'noise', in the performance metrics, complicating the process of drawing definitive conclusions about each model's effectiveness, especially during the fine-tuning and comparison of different training strategies.

To reduce the impact of this variability and strengthen the conclusions of our study, we occasionally used an alternative validation method. We assessed some models against an external leaderboard, which helped verify our models' generalizability and consistent performance across different data sets.

5.2. Training

Our initial training approach involved using stochastic gradient descent (SGD) with a set

momentum. As the project progressed, we shifted to using the Adam optimizer, which tended to achieve better and quicker convergence. Training times ranged from 2 to 5 hours until convergence, starting with a learning rate of 0.0001.

The training-to-validation split was 80:20. Due to noticeable variations in training metrics across different sessions, we implemented seeding to stabilize results. This control was pivotal in maintaining consistency when varying data augmentations, optimizer configurations, or learning rates.

After stabilizing the training outputs, we further experimented with different learning rates and optimizers alongside various augmentations. This approach allowed us to meticulously analyze how each factor influenced the models' performance, thus identifying the most effective settings for our specific needs.

5.3. Regularization

Our model demonstrated a good level of robustness during the training phase, largely negating the need for additional regularization adjustments. The implemented dropout layers, with a probability of 0.5, effectively mitigated the risk of overfitting. This is indicative of the model's capacity to learn general patterns rather than memorizing the training data, a quality that was consistently reflected across various training iterations. The absence of overfitting suggests that our network architecture, interspersed with appropriate dropout, has been successful in preventing the learning of irrelevant patterns, or 'noise', from the data.

6. Results

Epoch	Train Loss	Train Acc	Val Loss	Val Acc	Correct	Total
1	1.6535	53.69	0.9960	82.09	614	748
2	0.8779	77.63	0.6241	84.22	630	748
3	0.6643	81.48	0.4964	87.17	652	748
4	0.5777	82.92	0.4293	87.70	656	748
5	0.5191	85.02	0.3809	89.44	669	748
6	0.4696	86.36	0.3546	90.24	675	748
7	0.4444	86.73	0.3243	91.04	681	748
8	0.4099	87.50	0.3059	91.44	684	748
9	0.4042	87.70	0.3036	90.51	677	748
10	0.3952	87.36	0.2887	90.78	679	748
11	0.3745	88.26	0.2729	92.11	689	748
12	0.3670	88.40	0.2626	92.38	691	748
13	0.3568	88.47	0.2626	91.58	685	748
14	0.3411	89.23	0.2471	92.25	690	748
15	0.3368	88.87	0.2470	92.51	692	748
16	0.3140	90.04	0.2461	92.11	689	748
17	0.3294	88.90	0.2360	92.51	692	748
18	0.3076	89.87	0.2362	91.98	688	748
19	0.3047	90.37	0.2344	91.84	687	748
20	0.2945	90.67	0.2263	92.38	691	748
21	0.2988	90.10	0.2213	92.78	694	748
22	0.2887	90.54	0.2224	92.51	692	748
23	0.2776	90.91	0.2175	92.78	694	748
24	0.2771	90.84	0.2103	92.78	694	748
25	0.2692	91.24	0.2103	92.78	694	748
26	0.2719	91.04	0.2134	92.25	690	748
27	0.2584	91.17	0.2046	93.18	697	748
28	0.2600	91.94	0.2060	92.11	689	748
29	0.2510	91.78	0.2127	92.78	694	748
30	0.2636	91.01	0.1986	93.32	698	748
31	0.2549	91.37	0.1964	93.18	697	748
32	0.2505	91.94	0.1973	92.65	693	748
33	0.2546	91.68	0.1933	93.18	697	748
34	0.2437	92.08	0.1979	93.18	697	748
35	0.2377	91.74	0.1962	93.18	697	748
36	0.2473	92.28	0.1890	93.58	700	748
37	0.2272	92.61	0.1895	93.32	698	748
38	0.2425	91.54	0.1884	93.45	699	748
39	0.2213	92.48	0.1897	92.38	691	748
40	0.2182	93.05	0.1910	93.18	697	748
41	0.2227	92.38	0.1961	93.05	696	748
42	0.2294	92.58	0.1963	92.51	692	748
43	0.2273	92.68	0.1918	92.38	691	748
44	0.2179	92.95	0.1857	93.32	698	748
45	0.2338	91.94	0.1843	93.45	699	748
46	0.2181	92.81	0.1853	93.05	696	748
47	0.2146	92.78	0.1844	92.91	695	748
48	0.1987	93.55	0.1878	92.91	695	748
49	0.2048	93.18	0.1832	93.05	696	748
50	0.2043	93.11	0.1827	93.32	698	748

We found that without any data augmentation we achieved 93.3% and the accuracy of model prediction reduced with any combination of the data augmentation we applied. Secondly, we saw an increment in accuracy of the model after adding one sequential layer to the pre-trained VGG16 model.

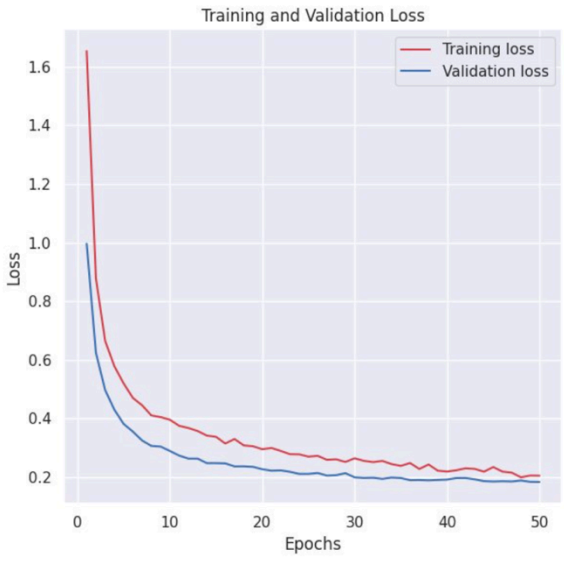


Figure 4: ADAM accuracy/Loss vs iteration

Analyzing our model's training progress, we saw a swift decline in training and validation loss initially, showing that the model learned quickly. Over time, losses plateaued, indicating the model had learned most of the patterns from the data. The validation loss closely tracked the training loss, a sign of good generalization to new data.

Figure 5: Training and Validation Accuracy

Analyzing our model's training progress, we saw a swift decline in training and validation loss initially, showing that the model learned quickly. Over time, losses plateaued, indicating the model had learned most of the patterns from the data. The validation loss closely tracked the training loss, a sign of good generalization to new data.

	precision	recall	f1-score	support
Delphineis	0.99	0.95	0.97	77
Dinophysis	0.88	0.88	0.88	76
G_delicatula_external_parasite	0.97	0.95	0.96	82
Katodinium_or_Torodinium	0.88	0.95	0.91	77
Leptocylindrus_mediterraneus	0.97	0.99	0.98	86
Licmophora	0.87	0.90	0.89	60
Strombidium_morphotype2	0.97	0.87	0.91	68
Tontonia_gracillima	0.87	0.95	0.91	80
bead	0.99	1.00	0.99	80
spore	0.95	0.85	0.90	62
accuracy			0.93	748
macro avg	0.93	0.93	0.93	748
weighted avg	0.93	0.93	0.93	748

Figure 6: Classification report for individual labels

From the above classification report, the class bead has the highest precision at 0.99 and a perfect recall of 1.00, resulting in an F1-score of 0.99, which is the highest among all classes. This indicates that the model was almost perfect in classifying the bead class, with nearly all positive predictions being correct and all actual instances of bead being identified.

On the other hand, despite Tontonia_gracillima having a high F1-score of 0.91, it has a relatively lower precision of 0.87 compared to other classes, indicating more false positives for this class than some others. The recall is quite high at 0.95, but for precision-dependent applications, this could be an issue.

7. Visualization of Model predictions

We used some visualization to understand our model more accurately. Saliency mapping is one technique where we can plot heatmap to see where the model is looking in order to predict the image.

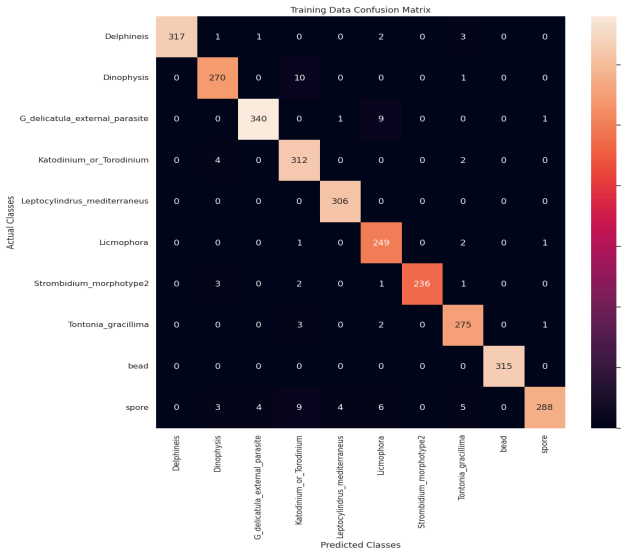


Figure 7: Confusion matrix of training data.

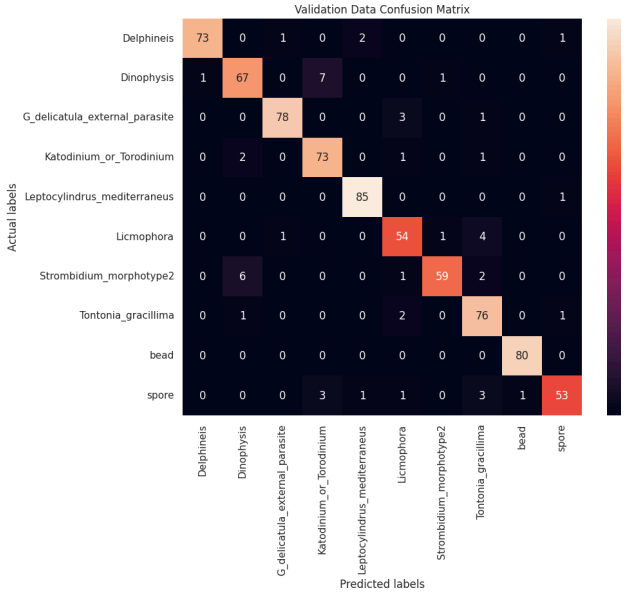


Figure 7: Confusion matrix of validation data

Confusion matrix is another way to visualize the model performance. A confusion matrix is a table often used to describe the performance of a classification model on a set of test data for which the true values are known. It allows you to visualize the accuracy of the model by comparing the actual target values with those predicted by the model.

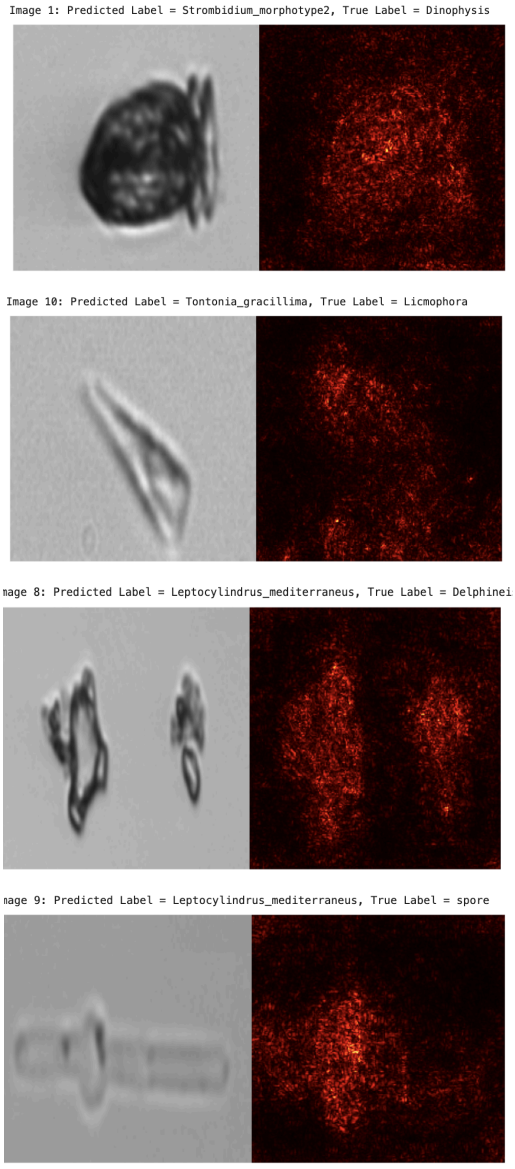


Figure 7: Saliency map of misclassified data

The figure presented above illustrates saliency maps for a selection of images from the validation dataset that were incorrectly classified by the model. Saliency maps are analytical tools that help us decipher which aspects of the data the model is paying attention to during the learning process. From the four depicted visualizations, it is evident that the model is erroneously focusing on the background noise, leading it to misidentify the images as belonging to the wrong class. A similar issue arises with the second image.

In the case of the third image, the model faces a different challenge: it depicts two organisms of the same species, and the model fails to distinguish them accurately, assigning an incorrect label. Lastly, the species in the final image is barely discernible, with most of its defining patterns lost, causing the model to associate it with an inaccurate category.

8. Conclusion

In this paper we applied transfer learning to plankton classification. Using a VGG network pre-trained on the large ImageNet dataset we then fine-tuned the last dense layer to learn features of the plankton dataset. The effectiveness of this method showcases the robust properties of the VGG network. Even though the network had been trained on different images, the many features that had been learned throughout the 16 layers of the VGG proved to be extensible to multiple classification tasks.

We also conclude that using data augmentation should be handled carefully and not blindly be applied as it could reduce the accuracy of our model. It is not necessary if it could help in improving the robustness of the model.

9. References

[1] M. Hashemi, "Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation," J. Big Data, vol. 6, no. 98, 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0263-7

[2] E. C. Orenstein, O. Beijbom, E. E. Peacock, and H. M. Sosik, "WHOI-Plankton- A Large Scale Fine Grained Visual Recognition Benchmark Dataset for Plankton Classification," Scripps Institution of Oceanography and Biology Department, Woods Hole Oceanographic Institution. [Online]. Available: https://github.com/hsosik/WHOI-Plankton

[3] Vito P. Pastore, Thomas G. Zimmerman, Sujoy K. Biswas & Simone Bianco, "Annotation-free learning of plankton for classification and anomaly detection,"

[[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in Proc. of ICLR 2015, Visual Geometry Group, Department of Engineering Science, University of Oxford, 2015. [Online]. Available: http://www.robots.ox.ac.uk/~vgg/research/very_deep/