Classifying Marine Microorganisms using Convolutional Neural Networks

Michael Xie
Stanford University
Stanford, CA
xie@cs.stanford.edu

Gene Lewis
Stanford University
Stanford, CA
glewis17@cs.stanford.edu

Abstract

Plankton populations are a critical part of diagnosing ocean health. Plankton and other ocean microorganisms form the base of the marine food chain and join marine ecological processes[7]. An optical imaging approach would make it possible to automate the process of measuring plankton populations, and thus aid in expert diagnosis of ocean ecosystem stability. We propose a Convolutional Neural Network system for classifying plankton images. Baseline testing with a simple 3-layer ensemble Convolutional Neural Network yielded 56.8% validation accuracy and 79% training accuracy over 30k training images of 121 classes of marine microorganisms. A medium sized neural network with 7 layers yielded 95.7% training accuracy and 69% validation accuracy over 12 epochs. A fine-tuned network from the pretrained VGG 16 layer network from ISLVRC 2014 achieved 80.6% training accuracy and 64% validation accuracy over 3 epochs, but provided the best testing performance, with 1.13 log-loss on the test set with unknown labels.

1. Problem Introduction

From the Kaggle National Data Science Bowl website[5]:

> Traditional methods for measuring and monitoring plankton populations are time consuming and cannot scale to the granularity or scope necessary for large-scale studies. Improved approaches are needed. One such approach is through the use of an underwater imagery sensor. This towed, underwater camera system captures microscopic, high-resolution images over large study areas. The images can then be analyzed to assess species populations and distributions.

Our role, then, is to build an image classification system that achieves as high of an accuracy on unseen zoological data as possible with as high a confidence as possible. This is a classification problem in which we have arbitrarily sized plankton images as input and we output all class probabilities for each image. This means that, for each image, we output 121 probabilities, one for each class, which sum to 1. This notion is formalized by calculating the log-loss formula over all M classes for each given test image:

$$\text{log-loss}_{\text{example}} = -\sum_{j=1}^{M} y_j \log(p_j),$$

Taking the average log-loss over all training examples N:

$$\begin{aligned}\text{log-loss}_{\text{total}} &= \frac{1}{N}\sum_{i=1}^{N} \text{log-loss}_{\text{example}} \\ &= -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})\end{aligned}$$

This objective formalizes our intuition about the performance of our classifier; even if we achieve 100% test accuracy, we can still improve our log-loss score by having $p_{i,j_{correct}}$ be as close to 1 as possible, where i denotes the ith test image and $j_{correct}$ denotes the correct class for the given test image. This problem formulation carries the implicit constraint that we cannot be 100% certain about the class of a test image and be incorrect, as the log-loss would go to infinity; as a generalization, we see that this loss penalizes being both highly confident about a probability distribution for the classes for a test image and being wrong about the class with the highest probability.

2. Approach

To aid in our attempt to classify marine microorganisms by analyzing images, we use a Convolutional Neural Network pipeline with stochastic data processing techniques to improve performance. After outputting class probabilities at the end of this pipeline, we introduce a method for sparsifying the probability vectors in order to further minimize the log-loss. We tested our pipeline using three networks,

a 3-layer baseline network trained using a CPU, a 7-layer custom network, and finetuning the VGG 16 model from ISLVRC 2014[4].

3. Challenges

The plankton image dataset poses several challenges special to the dataset. First, the images are variably sized, grayscale, and non-natural. This poses problems in using a pretrained network to do transfer learning. Pretrained networks such as those that are trained on ImageNet [8] are trained for natural, color images. Second, the plankton can be pictured in all orientations, and we must generalize to all these orientations. Third, the images have roughly the same texture; pretrained networks such as VGG are particularly good at deducing the classification of an image from texture, but this information do not particularly visually separate the plankton classes. The plankton classes are decidedly dependent on shape; some classes are subclasses of the same species grouped by their shape only. Lastly, the presence of several "unknown" classes that semantically mean "things that don't fit into any of the other classes" are hard to train for, as they can have large variation and are not necessarily well represented in the training set. We will try to address all of these challenges in our approach.

3.1. Data Processing

The data is provided by the Hatfield Marine Science Center at Oregon University. The training data consists of 30336 images of 121 classes of marine microorganisms. The images are grayscale (black on white background), with varying dimensions. We first pre-process the data by equally padding both sides of the smaller dimension of each image to transform each image to be square. Since every image is mostly centered and cropped to fit to begin with, square padding preserves centering. We then scale the images to a uniform square dimension. Be rescaling from a square image to a square image, we do not distort the shape of the original data. This is important since the plankton are highly differentiated by shape. For our baseline results, we scaled the images to 64 by 64 grayscale images. For our 7 layer neural network, we scaled the images to 128 by 128 grayscale images. For the VGG 16 network, we scaled the images to 224 by 224, duplicating the grayscale image across color channels. The dimensions of the original images range from 30 pixels in the largest dimension to 250 pixels in the largest dimension; thus, there is an information-loss to computing time tradeoff when scaling the images to a uniform dimension smaller than the largest picture. For the baseline neural network, we store all the training examples in a $30336 \times 1 \times 64 \times 64$-dimensional matrix and the data class labels in a 30336×1-dimensional vector.

(a) Processed training example af-(b) Randomly rotated training exter square padding and resizing ample

Figure 1: Training data for the "echinoderm larva seastar brachiolaria" class

3.2. Data Augmentations

We use data augmentations to add noise to the input. These include random horizontal flips, vertical flips, and rotations. This increases the invariance of the neural network to different orientations of images. We decided against doing random shifts as the pictures are centered and cropped to fit on the whole and square padding preserves this centering. We decided against color perturbations since This is particularly justified for the plankton dataset since the plankton have no "normal" orientation and can be pictured from many different orientations. We performed horizontal flips with 50% chance, vertical flips with 50% chance, and two random rotations, with rotation angles ranging from 10°to 350°, each with 50% chance. Images chosen to be in the validation set were not augmented for use in the training set. The augmented dataset for the 7 layer neural network contained 87870 training images and 970 validation images. The augmented dataset for the VGG 16 network contained 73049 training images and 1032 images in the validation set. For the VGG 16 network only, the training images and validation images used mean pixel subtraction, using the mean pixel $[103.939, 116.779, 123.68]$ provided by Simonyan and Zisserman.

3.3. Hardware and Software

The 3-layer neural network was trained on a CPU, using the class implementation of Convolutional Neural Networks using Python and Cython. This was done on the Rye machines on Farmshare. Training the bigger networks, which had 7 and 16 layers respectively, proved to be much harder on a CPU. We used the Caffe Convolutional Neural Network library to train and specify our neural networks using a GPU [3]. We stored our augmented data and Caffe models/snapshots on Amazon S3 storage and ran spot instances on Amazon EC2. This enabled us to have enough

memory and enough stable computing power to run the networks. The Amazon EC2 instances had NVIDIA GRID K520 GPUs.

3.4. Dropout and Neuron Activation

Inverted dropout regularization was employed in the fully connected layers. Dropout introduces stochastic noise as well as limits the number of active neurons in every pass, increasing generalization to variation. Because we expect the same species of plankton to have a some amount of variation between individuals, dropout is useful for generalizing for this. We will use the ReLU activation as our default neuron activation.

4. Inducing Sparsity in Probability Vectors

In the interests of the Kaggle competition, the objective is not just to maximize the accuracy of the classifier but to maximize the log-loss of the probabilities outputted for the correct classes. In the setting of the Kaggle competition, we are presented with a test set with unknown labels and are asked to output class probabilities for each image. The log-loss is

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

This is the sum of the logs of probabilities outputted for the correct class. To minimize this, we must output 1 for each image's correct class. We can model the probability vector p outputted by the softmax layer of the neural network to be a noisy signal of the true class probability vector, which is a standard basis vector. We observe that elements of the p are frequently on the order of $1e-6$ to $1e-15$, with large probability mass on few classes. We want to aggregate as much probability mass as we can, up to a certain confidence, to the classes that we believe are correct. This is essentially inducing sparseness of the probability vectors.
We discuss 3 approaches to the probability mass-aggregation problem: Monte Carlo pruning, pairwise force simulation, and formulation as a convex optimization problem. Each represents a step in the thought process leading to the next.

4.1. Monte Carlo Pruning

The first approach is to use Monte Carlo sampling method to prune small probabilities. In this method, we take 1000 samples from the discrete distribution described by p and renormalize the probabilities based on the observed samples. Thus, classes with small probabilities will be pruned as a result of not being observed in the samples. We do this for a fixed number of iterations, according to how sure we are about the original class probabilities; more iterations results in more probability mass aggregation to the larger classes.

Algorithm 1 Monte Carlo Pruning

```
Given p, n
for i = 0 to n do
    Generate 1000 samples drawn from distribution p
    Create p̃ distribution from observed samples
    p ← p̃
end for
```

While this procedure achieves the desired result, it is computationally expensive to run Monte Carlo sampling in each iteration. Furthermore, this does not allow for additional information to be added beyond the noisy signal from the neural network itself, as we will see in the later approaches. This approach does, however, allow the possibility of changing the maximum probability class (and changing our belief about the class). We pursue an approach that incorporates outside structure into the problem, possibly enabling us to change beliefs in a non-random way based on phylogenetic information.

4.2. Phylogeny Tree

Recognizing that our data was biological in nature, we realized that a rich body of knowledge in the study of marine microorganisms could be leveraged to improve classification predictions; that is, by providing outside knowledge to our convolutional neural network, we could transform our probability vector by viewing the output of our CNN as a prior probability distribution between 121 classes, using some evidence to determine a likelihood function, and then finally computing a posterior distribution over the 121 classes. This process can be attributed to Bayes' rule, given as

$$P(X|D) = P(X)\frac{P(D|X)}{P(D)}$$

To define $\frac{P(D|X)}{P(D)}$, we turn to the use of a Computer Science structure that has a natural isomorphism to the problem of relating seemingly unrelated biological species: the graphical tree. When used to model ancestral relationships among biological organisms, the graphical tree provides a convenient structure for performing probabilistic inference with observed evidence; we can have a node potential defined between each pair of nodes that encodes our belief in how strongly those two nodes are correlated. Taken over all nodes, this structure would theoretically give us a way to calculate $P(D|X)$ by requiring us to only perform probability calculations along paths in the graph; however, this method is computationally very expensive, and infeasible for the number of images we would like to classify[6]. In addition, locating the appropriate data to construct the tree structure is difficult , and methods for estimating the appropriate probabilities are prone to severe inaccuracies.

4.3. Pairwise Gravitational Force Simulation

In this approach, we discuss how to exploit the above phylogeny tree as a structure that provides additional information when attempting to induce sparsity in our probability predictions for a given image. We first establish the notion of representing our probability prediction vector as point masses. Consider the representation of our probabilities as masses in a discrete, one-dimensional space with 121 locations, which is the number of classes. In addition to a point mass, each class can also be represented as a leaf node in the above described phylogenic tree, with each ancestral node denoting a common ancestor between a node's siblings. Classes are therefore separated in distance by the prior knowledge of the phylogenetic tree structure of the classes, where the pairwise distance $d_{ij} = d_{ji}$ between classes is the shortest path between the classes in the phylogenetic tree. We describe the force exerted by one mass p_i on another p_j as

$$F_{ij} = \frac{p_i * p_j}{d_{ij}^2}$$

The algorithm is as follows: for every class i, we consider every other class and compute the scaled force between the pair by multiplying the probability that we predict for the two given classes and scaling down by the pair-wise distance as encoded by the tree; note that the distance is always a positive integer due to it's reliance on our discrete graph structure, and so our force is always between 0 and 1 - thus, when multiplied by a mass, the force will give how much of that mass to move from the smallest node in a pair to the largest node in a pair. We encode this intuition with a pair of update equations for the probability masses of a pair of nodes:

$$max\{p_i, p_j\} = max\{p_i, p_j\} + \frac{p_i p_j}{d_{ij}^2} min\{p_i, p_j\}$$

$$min\{p_i, p_j\} = min\{p_i, p_j\} - \frac{p_i p_j}{d_{ij}^2} min\{p_i, p_j\}$$

Where we move a fraction of probability mass (encoded by the force) from the smaller mass to the larger mass. To prevent masses from changing while calculating the forces during an iteration, we define a simple algorithm for performing a simultaneous update. We first store each of the mass updates as given above. We then apply the subtractive updates first; after all quantities have been removed from their respective probability masses, all of the additive updates are performed. Thus, each of the probability masses are updated simultaneously and correctly.

This pairwise approach introduces new information, namely the notion of distance between classes, that can be used to calibrate beliefs; it also runs quickly, taking constant time per image and linear time across all images. However, incorrect phylogenic data and a poorly constructed tree has the potential to horribly re-calibrate the probability prediction vector; in addition, finding complete phylogenic data for our specific subset of plankton has proved challenging. For these reasons, we search for a method that will move probability mass independent of additional problem insight; though such a method might not change our beliefs about the most probable classification, it might strengthen our confidence in our choice in a more reasonable fashion than simply allowing the most probable estimate to have a confidence of 100%.

4.4. As a Convex Optimization Problem

Finally, we formulate the probability mass-aggregation problem as a convex optimization problem.

We would like to solve the optimization problem

$$\begin{aligned} \underset{\tilde{p}}{\text{minimize}} \quad & card(\tilde{p}) + \lambda \|\tilde{p} - p\|_2 \\ \text{subject to} \quad & \mathbf{1}^T \tilde{p} = 1 \\ & \tilde{p} \succeq 0 \end{aligned}$$

where $card(\tilde{p})$ is the cardinality of $\tilde{p}$. The regularization term is present to encourage the new probability vector $\tilde{p}$ to be similar to the initial probability vector p. The $l2$ norm is a convex quadratic term. The constraints ensure that $\tilde{p}$ is on the probability simplex, and is composed of linear equalities and inequalities, and is thus convex. However, cardinality problems are generally not convex, so we solve a convex relaxation of the cardinality problem, inspired by the *earth mover's distance* metric for distance between histograms, images, and probability distributions [9]. In this formulation, we introduce the mover matrix $S \in \mathbb{R}^{n \times n}$ and the cost matrix $C \in \mathbb{R}^{n \times n}$, $n = 121$. We constrain the mover matrix S such that $S\mathbf{1} = p$ where $\mathbf{1} \in \mathbb{R}^n$ is a vector of all ones. This means that the columns of S sum to p, the probability distribution outputted by the softmax layer of the neural network. We will solve a convex optimization problem to find S, and derive the new (sparsified) probability vector $\tilde{p}$ from $\tilde{p} = S^T\mathbf{1}$, which is the sum of the rows of S. Intuitively, we want S to define how to move the probability mass from each element of p to each element of the new distribution. For example, the first row of S defines how to distribute the mass of the first element of p to each of the $n = 121$ classes of the new distribution $\tilde{p}$. Because the first row of S adds to the first element of p, mass is conserved in the movement. As an aside, S is not a stochastic matrix as the rows do not add to 1, but rather $\mathbf{1}^T S \mathbf{1} = 1$.

The convex relaxation of the original problem is

$$\begin{aligned} \underset{\tilde{p}}{\text{minimize}} \quad & \mathbf{1}^T\tilde{p} + \lambda\|\tilde{p} - p\|_2 \\ \text{subject to} \quad & \mathbf{1}^T\tilde{p} = 1 \\ & \tilde{p} \succeq 0 \end{aligned}$$

and with the relation $\tilde{p} = S^T\mathbf{1}$ we have

$$\begin{aligned} \underset{S}{\text{minimize}} \quad & \mathbf{1}^T S^T\mathbf{1} + \lambda\|S^T\mathbf{1} - p\|_2 \\ \text{subject to} \quad & S_{ij} \geq 0,\ i,j = 1,...,n \\ & S\mathbf{1} = p \end{aligned}$$

where $\mathbf{1}^T\tilde{p} = 1$ is implicit from $S\mathbf{1} = p$. However, we see that the first part of the objective function is always constant, so the optimal value for this problem is $\tilde{p} = p$. In order to alleviate this, we introduce the cost matrix C, where

$$C_{ij} = \begin{cases} \frac{p_j}{p_i} & i \neq j \\ 0 & i = j \end{cases}$$

is the cost of moving a unit of probability mass from class j to class i. In other words, if we were to move 0.3 probability from class p_j to p_i, the cost would be $0.3\frac{p_j}{p_i}$. There is high cost of moving probability mass from a p_j with high probability and also high cost of moving probability mass to a p_i with a low probability. Thus, the cost encourages movement of probabilities to higher probability mass classes. Our objective is then to minimize the cost of moving from p to $\tilde{p}$. The final optimization problem formulation is as follows:

$$\begin{aligned} \underset{S}{\text{minimize}} \quad & \mathbf{1}^T SC\mathbf{1} + \lambda\|S^T\mathbf{1} - p\|_2 \\ \text{subject to} \quad & S_{ij} \geq 0,\ i,j = 1,...,n \\ & S\mathbf{1} = p \end{aligned}$$

The optimization problem is solved through CVX, a package for specifying and solving convex programs [2][1], which uses the SDPT3 infeasible path-following solver. With $\lambda = 0$, the problem always outputs a standard basis vector, where the class in p with the highest probability will be attributed all the probability mass. Here, we show some results for different values of λ on a random test a probability vector where $n = 10$.

We have the original vector:

$$\begin{bmatrix} 0.1525 \\ 0.1127 \\ 0.1527 \\ 0.0958 \\ 0.1346 \\ 0.0709 \\ 0.1340 \\ 0.0830 \\ 0.0036 \\ 0.0602 \end{bmatrix}$$

The following is the transformed probability vector via solving the optimization problem for $\lambda = 10$:

$$\begin{bmatrix} 0.1854 \\ 0.1236 \\ 0.1857 \\ 0.0918 \\ 0.1592 \\ 0.0322 \\ 0.1582 \\ 0.0639 \\ 0.0000 \\ 0.0000 \end{bmatrix}$$

The following is the transformed probability vector via solving the optimization problem for $\lambda = 2$: :

$$\begin{bmatrix} 0.3305 \\ 0.0000 \\ 0.3319 \\ 0.0000 \\ 0.1719 \\ 0.0000 \\ 0.1657 \\ 0.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}$$

For $\lambda = \frac{1}{std(p)}$:

$$\begin{bmatrix} 0.1536 \\ 0.1133 \\ 0.1538 \\ 0.0961 \\ 0.1355 \\ 0.0705 \\ 0.1349 \\ 0.0830 \\ 0.0000 \\ 0.0592 \end{bmatrix}$$

We choose $\lambda = \frac{1}{std(p)}$, with the intuition that extremely peaky initial probability vectors p mean that we are initially more sure of the classification, and thus we should care less about matching the old output and more with maximizing the largest class probability. When the initial probability vector is diffuse, meaning we are less sure of out output, we have a smaller standard deviation and thus care more about just keeping the probability vector the same. We see that this is the case in the above example, where there are many classes with similar probabilities. Choosing $\lambda = \frac{1}{std(p)}$ leads to a conservative sparsification policy in these cases. Lambda can also be cross-validated to ensure the right amount of sparsification occurs. One limitation of the optimization approach as presented is that the class

with the largest probability mass will always remain with the largest new probability mass, and so on for the rest of the classes. If the classifier initially gets the classification incorrect, then probability mass would be added to an incorrect class. However, it is mostly the case that the correct class is also one of the high probability mass classes, and thus it will gain mass under this approach. We see here a tradeoff between λ and the inherent accuracy of the classifier. An interesting approach to alleviate this would be to train a binary classifier over the validation set for whether certain probability vector patterns correspond with being correct; we sparsify those that we believe are already correct. By training on the validation set, we add new information to the sparsification process not seen in training. Lastly, the notion of class distances from the phylogeny tree can be encoded within the cost matrix C, which adds new information not available during training to the sparsification process.

Testing with our VGG 16 classifier increased our log-loss from 1.13 to 2.2, roughly a 2 times increase. We conclude that our classifier had not gained enough accuracy for sparsification to benefit the log-loss with the choice of $\lambda = \frac{1}{std(p)}$.

5. Results

The 3-layer Convolutional Neural Network was trained using 1000 randomly sampled images from the training set for the validation set and the remaining 29336 images as the training set. The 7 layer network was trained on 87870 augmented training examples with 970 validation examples. The VGG 16 layer network was fine-tuned from existing weights from the BVLC Model Zoo[4][3]. The test set contained 130400 images with unknown labels.

Training accuracy and Validation accuracy for three sizes of CNN

Classifier	Train acc.	Val acc.
3 layer	0.786	0.568
7 layer	0.9573	0.690
VGG 16	0.806	0.64

The VGG 16 network achieved 1.13 log-loss on the unknown label test set, through submission to the Kaggle competition. The 7 layer network was not able to generalize well, achieving a log-loss of 20.7.

5.1. Baseline Neural Network

For our baseline results, we used a simple 3-layer Convolutional Neural Network. The 3-layer network was able to fit the training data to 80% accuracy, which is surprisingly expressive. Our baseline network accepted inputs of 64 by 64 images, using a $CONV - RELU - MAXPOOL - CONV - RELU - MAXPOOL - FC - SOFTMAX$ architecture. Each convolutional layer use 32 3 by 3 filters with stride 1. The max pooling layers use 2 by 2 pooling areas with stride 2. Optimization by minibatch gradient descent used the momentum update rule with 0.9 decay parameter and a batch size of 50.

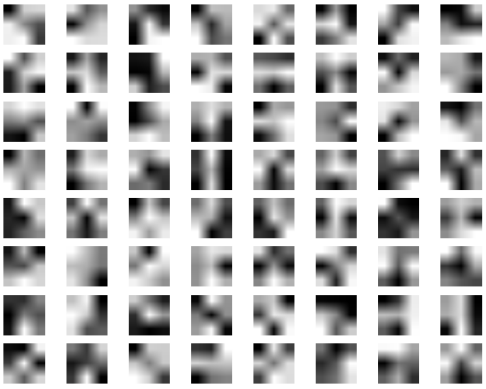

Figure 2: Filters for the first convolutional layer of the 7 layer network after 12 epochs

We trained 3 models with varying regularization parameters and learning rate annealing schedule, each with 2 epochs through the data. For a form a self-transfer learning, we initialized each of the models (except the initial) with the weights of the previous. The ensemble model over the 3 trained models attained 56.8% validation accuracy.

The baseline network was trained using a CPU, which limited the number of epochs that could reasonably be trained. The baseline fit the training data surprisingly well, but lacks generalization. This can be due both to representational power and to lack of data augmentation.

5.2. 7 Layer Neural Network

Inspired by the success of our baseline model, the 7 Layer Neural Network was an attempt to capitalize on the gains made by boosting the expressive power of our network by introducing more parameters. Outfitted with a $((CONV-RELU)\times 2-MAXPOOL)\times 2-(CONV-RELU-MAXPOOL)\times 2-FC-SOFTMAX$ architecture, the 7 Layer Neural Network was able to achieve 0.95 accuracy on the training set and 0.67 accuracy on the validation set. However, this proved to be a case of drastic overfitting, as the 7 Layer Neural Network ultimately achieved a log-loss of 20.3 over 3 submissions. We believe that this is due to the network not having the benefit of seeing a diverse range of images; as we discuss next, the filters in VGG seem to represent much higher-level concepts than

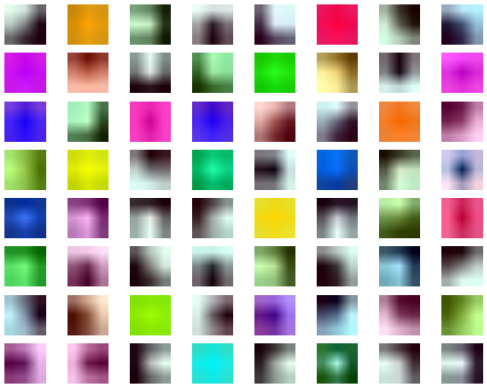

Figure 3: Filters for the first convolutional layer of the VGG 16 layer network after 3 epochs

the specific, low level ones in the 7 Layer Neural Network.

The filters learned for the 7 layer network are unconventional in that many of the filters represent curves as well as straight lines. We can explain this by thinking about the shape of plankton; they are generally closed ellipsoid in shape, with varying semi-axes. This corresponds to circles, ovals, and, with enough eccentricity, stick shapes. We can visually compare these weights to those in the VGG 16 set (right), where we see drastic differences. Ignoring the differences in color (as the VGG 16 model was trained on 3-channel images; discussed below), we can see that the VGG 16 weights primarily picked up on higher-level features, such as corners and gradients. We hypothesize that this wealth of higher-level features, in juxtaposition to the 7 Layer Network's fairly sharp, alien, plankton-specific features, are what ultimately prevented VGG 16 model from severely overfitting the plankton dataset and allowed it to achieve much better generalization results than our 7 Layer Network.

5.3. VGG 16

The VGG 16 network is finetuned from a pretrained network from Simonyan and Zisserman's ISLVRC 2014 submission, VGG 16 [4]. We utilize transfer learning from the pretrained VGG network, although the pretrained network was trained on natural, color images. We can see from the filters of the VGG 16 model that some filters learned solid colors, which is not present in the dataset. This is residual from VGG 16 having been pretrained on a color dataset and then being applied to a grayscale dataset. The color filters are not activated during finetuning and gradually become solid. Thus, this could improve the generalization of the VGG network by limiting the number of active neurons. Networks that are trained on ImageNet [8] are also known to be effective at using texture to deduce images; the plankton images all have roughly the same texture, and this poses a challenge to the VGG 16. Most of the curved edge filters learned are close to grayscale; this aligns with the plankton dataset and the usual curved shape of plankton.
Ensemble models generated from averaging the weights from 1 epoch, 2 epochs, and 3 epochs of fine-tuning resulted in a slightly lower log-loss on the unknown test set.

6. Conclusions

From the baseline results, we can see that plankton image classification problem can be handled relatively effectively with a very simple Convolutional Neural Network. When we increased the complexity of the architecture of the neural network and experimented with the stochastic data processing and training techniques, we ran into challenges with overfitting and learned the challenges of repurposing a network trained for a vastly different dataset. We got experience with training datasets with GPUs using Caffe and experimenting with CNN architectures. It is also interesting to see that training a network trained on a colored dataset learns to ignore or forget color features when fine-tuned on a grayscale dataset. Going forward, we can try running different network architectures on a very wide variety of images, and see if different combinations of ensembled models allows training on plankton to discern much higher, relevant features. Anothing interesting avenue concerns sparsifying the probability vectors; learning a classifier on the validation set to predict the lambda value or predict whether to sparsify or not, as well as constructing a rigorous phylogeny tree, could provide extra information needed to squeeze out the last few percent.

References

[1] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/graph_dcp.html`.

[2] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, Mar. 2014.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[4] A. Z. K. Simonyan. Very deep convolutional networks for large-scale image recognition, 2014.

[5] Kaggle.com. National data science bowl, 2014.

[6] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[7] e. a. M. Sieracki, M. Benfield. Optical plankton imaging and analysis systems for ocean observation, 2010.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014.

[9] L. G. Y. Robner, C. Tomasi. The earth mover's distance as a metric for image retrieval, 2000.