Plankton Species Identification via Convolutional Neural Networks

Michael DAngelo
Stanford University
mdangelo@stanford.edu

Josh Tennefoss
Stanford University
joshx@stanford.edu

Abstract

In this paper we demonstrate a method to identify photoplankton species from gray-scale images. The images have been pre-processed by Kaggle as part of The National Data Science Bowl (NDSB) [3] to produce scenes containing a single photoplankton species. We implemented and tested several different architectures of multilayer convolutional neural networks on AWS using K40 GPUs and Caffe's open source neural networks libraries. Our best performance architecture is a 29 layer network that achieves a testing accuracy of 26.1% in predicting the correct class out of 121 possible different classes.

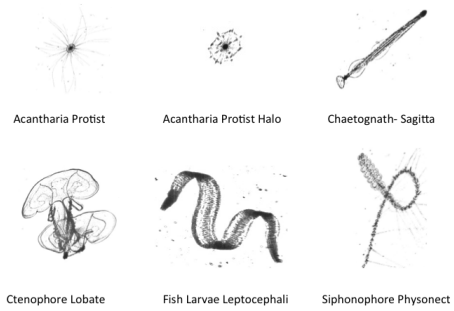

Figure 1. Sample images from NDSB dataset.

1. Introduction

Plankton are small, prevalent sea creatures that float throughout our oceans, gathering sunlight and feeding larger animals. There are 1000's of uniquely identified species of photoplankton, each of which serves a unique purpose in their environment. Understanding each species movements, diversity, population density, and growth patterns are crucial for gathering insight into the life cycles of our oceans. However, since there cane be hundreds of thousands of Plankton per square meter, manually identifying species is a futile task. Machine vision algorithms for automatic identification of plankton species in underwater images will allow Plankton to be tracked at the scale necessary for accurate documentation.

2. Competition

Kaggle, in combination with the Hatfield Marine Science Center at Oregon State University, has established this competition to encourage participants to build efficient plankton species type classifiers.

2.1. Dataset

All data from this paper was prepared and distributed through Kaggle. Kaggle is an online data science competition platform on which different groups can post their datasets and prediction challenges that sit on those datasets. To incentive people to work on the problems they often provide prizes for the top performing models. The challenge that provided the data for plankton species identification is called the National Data Science Bowl (NDSB) and offers $175,000 in prizes for the best performing algorithms.

According to the NDSB: "For this competition, Hatfield scientists have prepared a large collection of labeled images, approximately 30k of which are provided as a training set. Each raw image was run through an automatic process to extract regions of interest, resulting in smaller images that contain a single organism/entity." [2] Examples of some of these images may be seen in figure 1.

The NDSB also provides an unlabeled test set composed of approximately 150K images. All images in the NDSB training and test datasets are grayscale, ranging in dimensional area from from 861 pixels^2 to 163590 pixels^2. Figure 2 presents a histogram of image area over the training set.

As can be seen from the histogram, most images are below 5000 pixels ^{2}. We tried to consider the useful are of the images when designing our network architecture. Before training the networks we subtracted the mean image of the entire training set from each training example (and, if applicable, the mean image from the pre-trained network). Figure 3 represents the mean image when all images in the

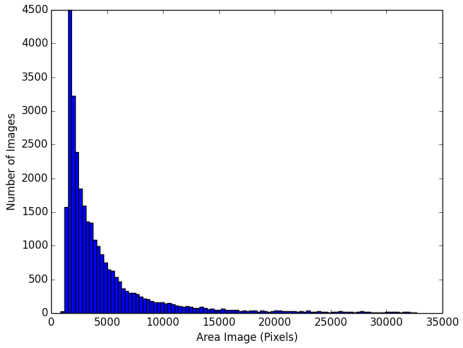

Figure 2. Distribution of area of images in the NDSB dataset.

Figure 3. Mean image from the training dataset at 96x96 pixels.

training set have been resized via stretching with a bicubic sampling function to 96x96 pixels.

The NDSB training set provided by Kaggle was split into training and validation sets for our network. After experimenting with several different division strategies, we chose to randomly select 30% of the images from each class in the training set to be considered our validation set. We chose this strategy because it seemed to provide good validation accuracy in comparison to other divisions, and we found ample support in existing literature.

In addition to the raw images that were provided to us by the NDSB we created additional training sets of data by cropping, flipping, stretching, warping, and changing the contrast of the images. This was done prior to rescaling the images to a consistent size for training. This will be discussed in detail below in the data augmentation segment of this paper.

2.2. Evaluation Criteria

There are 121 uniquely identified species of photoplankton in the dataset provided for the competition. Kaggle asks us to give predictions for each image in the test set in the form of per class probabilities. For each example, competitors must submit a 130,400 by 121 CSV file with probabilities for each example that sum to 1. They will then evaluate our scores by the following log-loss function:

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

N represents the number of images in the test set, M represents the number of class labels, y_{ij} is 1 if observation i is in class j and 0 otherwise, and p_{ij} is the probability that image i is class j. Kaggle composed additional constraints by limiting probability scores to be between 10^{-15} and $1 - 10^{-15}$.

Our best performing network had a logloss score of 1.34, which corresponds predicting the correct class with 26.1% probability. We will discuss this below in greater detail.

3. Background

Convolutional neural networks have been gaining traction in the last decade due to increase data availability, computing power, and software for relatively quick training. In particular they have been successful in image classification tasks, including the popular ImageNet challenge. We are lucky to have had a class offered at Stanford that taught of the ins and outs of these networks without having to begin learning about them from the literature. Though the implementations in this paper leverage what we learned in CS231N, keeping up with the current literature would certainly help us improve these models.

4. Infrastructure

In this section, we will discuss the infrastructure that we established in order to train and validate networks.

4.1. AWS

We used two AWS EC2 GPU instances as our primary computing resource. These instances were built with Ubuntu 14.04 and Cuda 6.5. Using high end K40 GPUs, we were able to drastically decrease network training and prediction time. For an 18 layer network we built with 12% prediction accuracy, the network typically took on the order of 51 seconds per epoch, versus approximately 300 seconds in CPU mode, constituting a 5X speedup.

4.2. Convolutional Neural Network Framework

Caffe is a convolutional neural networks framework that was born out of UC Berkley. They have created relatively simple interfaces for adjusting network architecture and have options that allow GPU implementation [1]. We wrote wrappers around Caffe that set up our network using Bash

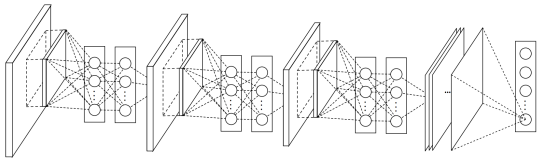

Figure 4. Network in Network Topology

Topology	Layers	Validation Accuracy	Test Accuracy
ImageNet	27	68.3%	24.5%
NIN	31	66.1%	26.1%

Figure 5. Network Performance Statistics

scripts and python. Additionally, we used ipython notebook to visualize our networks layers and perform computations where visual inspection was needed, as our AWS instances had no gui packages installed.

5. Architecture

After experimenting with several different network architectures, we settled on a Network in Network model based on the work of Min Lin REF. This network is composed of 31 layers, representing several convolutions with a large fully connected network at the output.

The architecture of the NIN model may be visualized in figure 4. Notice the repeated sets of convolutional layers and the final fully connected layer.

The network architecture can be textually visualized as follows:
INPUT $\rightarrow$ [[CONV $\rightarrow$ RELU]*3 $\rightarrow$ POOL]*3 $\rightarrow$ DROP $\rightarrow$ [[CONV $\rightarrow$ RELU]*3 $\rightarrow$ POOL] $\rightarrow$ SOFTMAX

Table 6 represents the parameters associated with each layer.

6. Looking Deeper into the Net

Some people consider neural networks to be black boxes. This might be partially true since it is difficult to understand reason certain weights are increased during training. But it is possible and insightful to analyze the features of a trained networks.

6.1. Filter Visualizations

In class we talked about the benefit of visualizing network weights to improve our insight into what the network is doing. The next page includes a number of sample weight visualizations from a single image that was run through the network.

Name	Output	Kernel	Stride
conv1	96	11	4
cccp1	96	1	1
cccp2	96	1	1
pool0	(Max)	3	2
conv2	256	5	1
cccp3	256	1	1
cccp4	256	1	1
pool2	(Max)	3	2
conv3	384	3	1
cccp5	384	1	1
cccp6	384	1	1
pool3	(Max)	3	2
drop	dropout_ratio: 0.5		
conv4-1024	1024	3	1
cccp7-1024	1024	1	1
cccp8-121	121	1	1
pool4-1	(Average)	6	1
loss-1	Softmax Loss		

Figure 6. Network in Network Topology

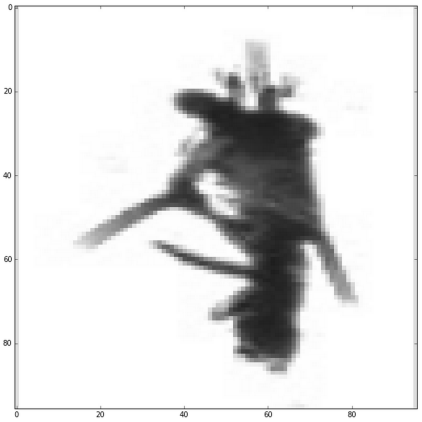

Figure 7. Example image for analysis. Class: Stomatopod.

6.2. Failure Analysis

Our network's best case performance was 26%. The authors hypothesize that there are two main reasons for relative low performance of our network. 1) Many of the classes the network predicted over in the training set were visually similar. There was no color information, and 2) many classes had very few training examples (in some cases as few as 9), while other classes had over 2000 images. A histogram of examples by class size in the training and validation sets may be seen below as figure 12. (In addition, it is very possible that we made some prediction mistakes regarding mean image subtraction that we did not have time to fix.)

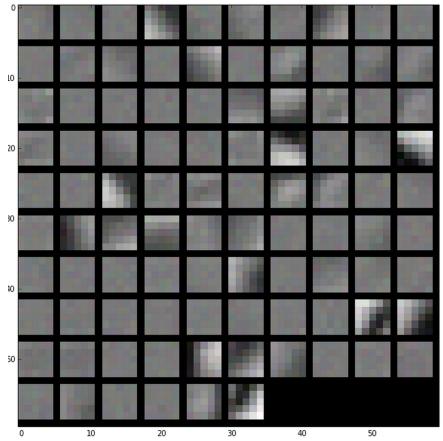

Figure 8. First layer filter visualizations of the image.

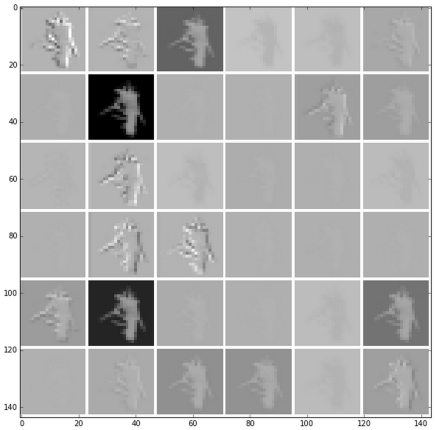

Figure 9. First layer output of the single image.

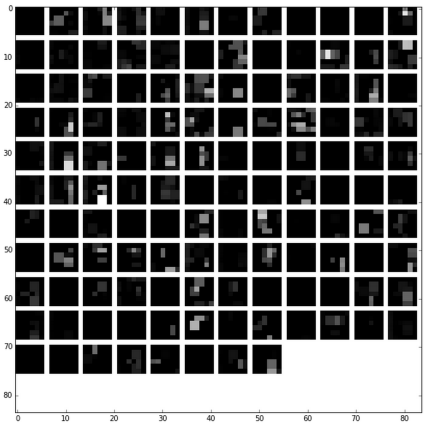

Figure 10. Final fully-connected layer output.

Because there were so few examples of some classes in our training set, the features relating to classes with few examples are likely to not accurately represent what they are intended to predict. We see this in classes like hydromedusae_haliscera_small_sideview which yield no classifications in our test set and which is only subtlety different than hydromedusae_haliscera with 229 examples, and the 3099 other examples divided into the 21 hydromedusae subclasses. An example of the similarity between hydromedusae_haliscera_small_sideview and hydromedusae_haliscera can be seen below.

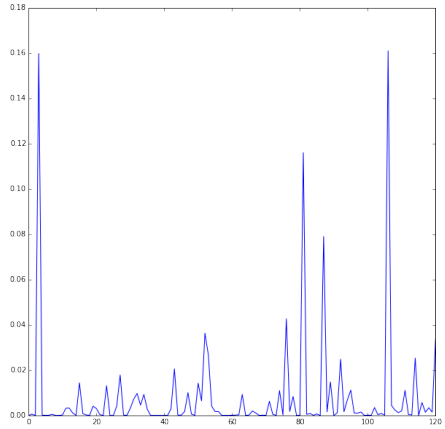

Figure 11. Final probability output from network. Notice that the wrong class had the highest score, since the correct class is 106, but the network predicted 112. See Failure Analysis section for further discussion.

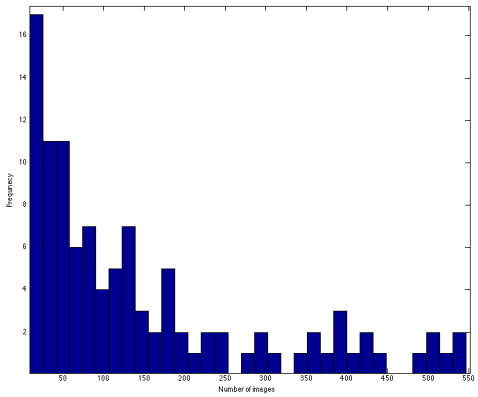

Figure 12. Histogram of number of examples per class. Note histogram was clipped. Some classes contain up to 2000 examples.

If we examine the results of a prediction of the image in 7, as presented in 11, we can see that class 3 and class 81 also scored very highly and nearly resulted in a missclassification. Examples of class 3 and class 81 may be viewed below as figures 15 and 16.

Clearly these images are visually similar. In training our

Figure 13. Haliscera Side-view

Figure 14. Haliscera

Figure 15. Example image for analysis. Class: Amphipod.

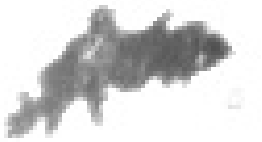

Figure 16. Example image for analysis. Class: Polychaete.

network was unable to find sufficiently unique features to distinguish many classes.

7. Reducing Overfitting

We employed many commonly used techniques in our network to reduce overfitting. These techniques include using dropout, building ensemble models of different networks trained with different training and validation classes, using data augmentation to perspective warp, contrast stretch, and distort images, and employing transfer learning.

7.1. Data Augmentation

Several different forms of data augmentation were employed, both internal and external to Caffe. We wrote filters for edge detection, adaptive contrast normalization, perspective warp, rotations, and blurring. In addition, Caffe provided an interface for random crops and flips. These filters did not seem to have significant affect on our validation accuracy, nor did they significantly improve our test accuracy. Our best case network without data augmentation yielded 23.9% correct class prediction. With data augmentation the performance rose to 26.1%. Some examples of different data augmentation we performed may be viewed below as figure 17

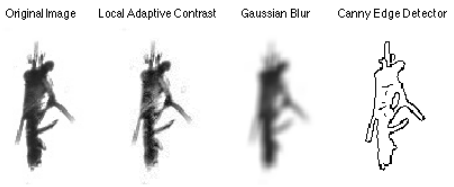

Figure 17. Example Data Augmentation

Figure 18. Mean image from network-in-network pretrained weights.

Data augmentation was performed on each image, and the augmented images were added to the training set to serve as more data examples.

7.2. Transfer Learning

We experimented with transfer learning with several different networks as part of Caffe's Model Zoo. Our best performance topologies include networks built for the 2012 ImageNet challenge ILSVRC2012 and the Network in Network model built by Min Lin [4]. The network in network model contains 31 layers and was designed for the imagenet challenge with 1000 classes in the final layer. We adapted this network by replacing the last layer with a 121 class softmax probability layer and trained the network for 10,000 iterations. We used the mean image as seen in figure 18 as opposed to the mean image from our dataset in figure 3. Further, we decreased the learning rate when training as to not overtrain. The layer weights presented in this paper were taken from this network.

Figure 19 shows the training loss over the first 1000 iterations of training the NIN network with a modified softmax probability layer.

As can be seen from the figure, the network very quickly

Figure 19. Loss over first 1000 iterations of training

learns final layer weights and converges. The final loss after 50,000 iterations was approximately 1.

8. Discussion

This discussion section focuses on a few things we learned while doing this project.

The most significant of these experiences was learning the basics of Caffe, including installation and running simple networks. Just being able to take a model from the model zoo, replace the last fully connected layer, and retrain with new data is incredibly powerful for many task. In fact, one of the authors will be using Caffe in this manner for his work in computer vision.

By working through this project, we learned that there can be many limiting factors that inhibit high network accuracies. These are addressed in the following section. The most notable of these is the amount of time it takes to train and test a network. We saw first hand that training a convnet is not like training a traditional ML classifier like an SVM or KNN, it can take much longer and is as much an art as it is a science.

Along this same vein, after looking at our classmates posters and listening to the lectures on convets, it is clear that the technology is in its infancy. In other words, we learned: if you have an idea that you haven't seen in the literature: its worth trying it out. Who knows, maybe it's a crazy idea like dropout that ends up working wonders.

9. Challenges

Much of our time working on this project was taken up by a few recurring challenges.

First, getting Caffe up and running on an EC2 instance, with the proper GPU libraries was not a trivial task. During this process we noticed that the documentation for Caffe is incomplete, and it is still a young library.

Once the library was working on the examples we moved to training our own models. Using the mean image in the network proved to be a difficulty. After a lot of experimentation we were unable to train our own networking using a mean image that we generated. As a temporary work around we subtracted the mean image in our data augmentation script. However, in the end, we moved to using pretrained models to take advantage of transfer learning, and we did not encounter the same mean image problems with those networks.

Once we had a trained network, we moved to predicting image classes on the test set to send into Kaggle. Using the normal Python predictions scripts, we found that is was quite slow to predict on the whole test dataset. We were required to predict on the whole set in order to submit to Kaggle. As a results we had to run the prediction script for about 24 hours each time we wanted to submit to Kaggle. In addition, we believe that we might have been having problems with the prediction script since some of our best test accuracy was about 26.1% but our validation accuracy was 68% at that time. In general we found that it was difficult to debug the predictions.

In general the two most time-consuming challenges that we faced was time limitations and limited Caffe documentation.

There are so many things to experiment with in neural networks that we felt we did not have time to test all of them. Since these were the first convolution networks we had ever trained we were interested in trying a large range of tests, but we found that we didn't have the time to try all of these since a significant amount of our time was spent getting the code running and debugging problems.

We found that the Caffe documentation was not comprehensive enough to be much help with debugging or implementing some techniques that we wanted to try. We both plan to use convolutional nets in the future, and are hopeful that the documentation for Caffe will continue to improve.

10. Acknowledgements

We would like to thank the CS231N teaching team for putting together a great course. We hope the next years courses go as well as this one.

References

[1] Caffe description. `http://caffe.berkeleyvision.org/`. Accessed: 2015-02-14.

[2] Nation Data Science Bowl data description. `https://www.kaggle.com/c/datasciencebowl/data`. Accessed: 2015-02-14.

[3] Nation Data Science Bowl kaggle description. `https://www.kaggle.com/c/datasciencebowl`. Accessed: 2015-02-14.

[4] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.