Simulating Octopus Color Camouflage with Genetic Neural Networks on a Raspberry Pi

Chaihyun (Catherine) Lee
University of Chicago
Chicago, IL
clee0505@uchicago.edu

Lucy Newman
University of Chicago
Chicago, IL
newmanlucy@uchicago.edu

ABSTRACT

In this paper, we use evolutionary genetic algorithms to investigate the possibility that color camouflage may have developed in the color-blind octopus over the course of evolution. The dataset is underwater coral reef images. We first investigate the feasibility of this colorization task with a convolutional autoencoder trained using backpropagation, and then used feedforward networks to explore minimum network architecture necessary for a model to learn colorization in order to simulate the simplicity of the octopus' neural circuitry. For testing the evolutionary genetic algorithm as a potential training method for colorization, we constructed a convolutional autoencoder with seven layers that were first trained to reproduce grayscale images and retrained the last layer on the colorization task using the genetic algorithm. We tried several variations of evolutionary algorithms for training this final layer, settling on an "elitist" algorithm that selected best network among the networks that were derived from a single parent network. At each generation, we had seven parent networks, and we would randomly mutate the weights and biases of the parents to create a total population of 65 networks. We would then choose the best of the children of each parent, or the parent itself, to survive to the next generation. To better simulate the potential limitations octopus may face due to their simple neural circuitry, we make some memory and latency optimizations with regards to network and batch width in order to allow real-time inference on a Movidius Neural Compute Stick connected to a Raspberry Pi. We were able to run inference on the Raspberry Pi alone, but because of limitations of the Movidus SDK, we have not been able to successfully run our models on the Neural Compute Stick.

CCS CONCEPTS

• **Software and its engineering** → **Abstraction, modeling and modularity**; *Software verification and validation*; • **Hardware** → Communication hardware, interfaces and storage.

KEYWORDS

Camouflage, genetic evolutionary algorithm, modeling, colorization, octopus, convolutional autoencoder, feedforward network

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Deep Learning Systems, 2018
© 2018 Association for Computing Machinery.
ACM ISBN 123-4567-24-567/18/12…$15.00
https://doi.org/10.475/123_4

ACM Reference Format:
Chaihyun (Catherine) Lee and Lucy Newman. 2018. Simulating Octopus Color Camouflage with Genetic Neural Networks on a Raspberry Pi. In *Proceedings of Dec, 2018 (Deep Learning Systems).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Octopus, of the cephalopod class, can camouflage in color despite not having cones in their eyes which allow many species of animals to see in color. There are several hypotheses as to how octopi camouflage in color despite not having cones, including that they can "see" color through their skin [6] or through slits in their pupils [7]. We present evidence for an alternative hypothesis: we suggest that octopi cannot, in fact, see color, but have rather evolved an ability to become the color of their surrounding through natural selection, wherein individuals that are able to guess the correct color in the presence of a predator are better able to survive and pass on this ability to offspring. We attempt to explore this hypothesis by modeling the process of natural selection using a genetic algorithm. We first train a convolutional autoencoder on a set of black-and-white images. We then feed the resulting network's latent representation into a genetic network that attempts to assign color, using the true colored pictures as labels.

2 METHODS

The first portion of this project was to train various machine learning architectures using backpropagation. This was done to both verify learning colorization task of underwater pictures is feasible and obtain a baseline performance of these classic models. The underwater image dataset used for this research was provided by Professor Rick Stevens. As there have been successful cases of colorization tasks using convolutional autoencoders [8], we first explored the boundaries and difficulties of training colorization of these images with a convolutional autoencoder. Then, we implemented simple three-layer and five-layer feedforward networks to explore what might be the simplest model architecture that allows the network to successfully learn the colorization task with the expense of high performance as the octopus' neural circuitry is rather simple. Next, we tested whether genetic evolutionary algorithms can be used to train various networks to successfully perform colorization of these images. To thoroughly investigate the feasibility of learning colorization using genetic evolutionary algorithm, we tested multiple different model architectures as well as several variations of genetic algorithms.

The second part of this project was to run inferencing on a Raspberry Pi. While the "training" of evolution takes place over millennia, the octopus does "inferencing" in real-time, in fractions

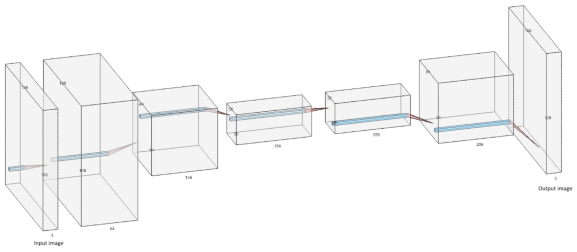

Figure 1: Architecture of convolutional autoencoder.
The convolutional autoencoder architecture with least number of filters used for each convolution layer that successfully performed colorization. The figure is not depicting max pooling and upsampling layers.

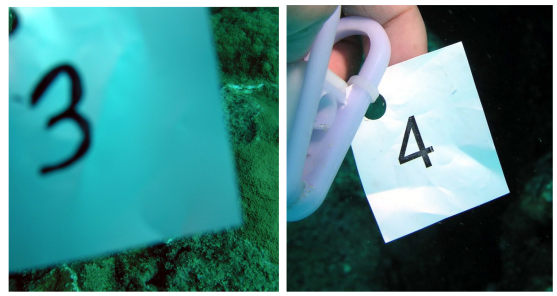

Figure 2: Example of color distortion in an underwater image.
This figure demonstrates color distortion in an underwater image. Both images are underwater pictures of a white piece of paper. However, the white paper on the left image appears to have a blue tint just like how the colors of real underwater scenes appear affected.

of a second, in the face of a predator. Furthermore, the octopus likely has relatively little "memory," so the limitations that a Raspberry Pi would impose seemed appropriate.

We first tried running the model we trained on our laptops or GPUs directly on the Pi, then on Movidius Neural Compute Stick, a processor designed specifically for real-time computer vision tasks, equipped with 4GB of RAM, in contrast to the Pi's 1GB.

Finally, we tested how our model performs on images that are visually very different from underwater images the network was trained on. To do so, we read in image frames from the web cam on a laptop, resized them and converted them to black and white, and then used the evolutionary model to infer color.

3 CONVOLUTIONAL AUTOENCODER

We implemented a convolutional autoencoder to explore the difficulties of the colorization task of underwater images. The convolutional autoencoder consisted of an encoder portion (three cycles of convolutional and max pooling layers) and a decoder portion (three cycles of upsampling layers using nearest neighbor method and convolutional layers, followed by an additional convolutional layer at the end). Once we found a model architecture that showed good colorization performance, we conducted a parameter sweep for the number of convolution filters to search for the smallest model with minimum influence on the quality of colorization [Fig.1].

3.1 Modifying underwater image dataset

The convolutional autoencoder was trained on the colorization task by taking the grayscale images as inputs and colored version of the images as the output. To our slight surprise, the model had difficulties learning the task.

The major barrier in training colorization was that the majority of the images in our underwater dataset suffered from color distortion [5]. Because of the amount of water between the camera and lens and the object, most of the images in the dataset had an overly blue tint [Fig.2] that encouraged the model to just predict all images to be of some blue shade. To avoid this problem, we built a new dataset by selecting images that had minimum color distortion while maintaining a mix of images with various colors and patterns.

3.2 Successful colorization using convolutional autoencoder

Training the convolutional autoencoder using images from this new dataset significantly improved its performance. Although it had difficulties correctly inferring for corals with rarely seen colors, it successfully predicted majority of the colors correctly and even had the capability to fix mild color distortion observed in the label images [Fig.3]. Compared to other models that we implemented for this project, convolutional network showed best performance in terms of both accuracy and number of iteration necessary for training [Fig.4].

4 FULLY CONNECTED FEEDFORWARD NETWORKS

After verifying that colorization task of underwater images is feasible through the convolutional autoencoder, we continued to explore the boundaries of the task by using three-layer and five-layer fully connected feedforward networks in order to investigate how simple a network can be to learn to recreate images and colorize [Fig.5]. This step of the project was largely motivated by our knowledge that the octopus' neural circuitry is rather simple and the fact that training a simple network architecture would be much easier when we train these models using evolutionary genetic algorithm as the number of parameters would decrease.

The three-layer and five-layer feedforward networks were first tested on recreating the input grayscale images without colorizing them. While both models were able to recreate approximations of the grayscale images after many iterations of training, the outputs from these models were not as accurate and clean compared to the outputs from the convolutional autoencoders. As expected, both models had a much tougher time learning to colorize the images. However, after a couple hundred thousand iterations of training, the five-layer feedforward network was able to produce images

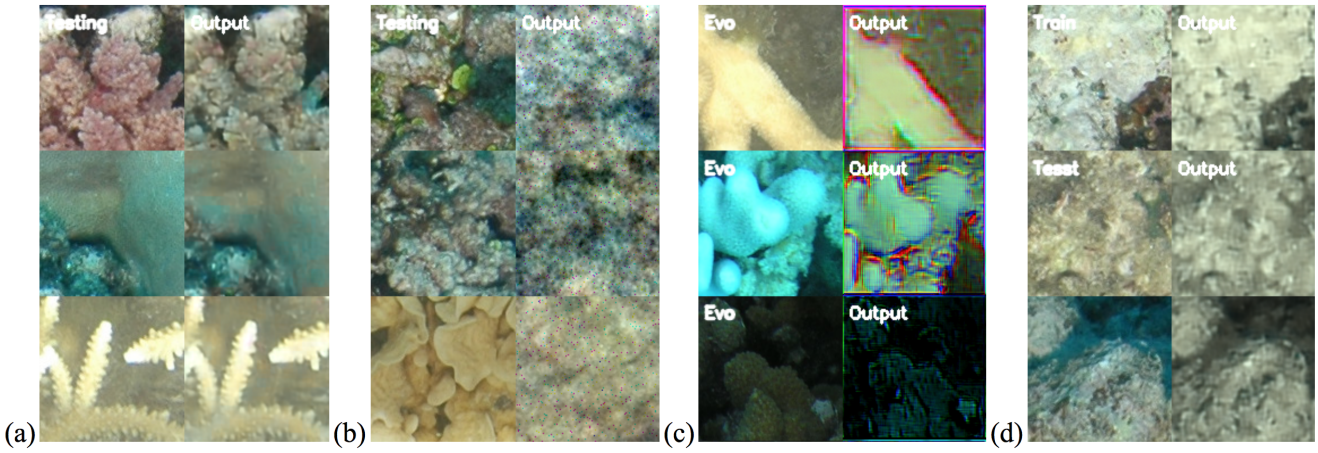

Figure 3: Colorization task outputs from various model architectures and training methods.
Left columns show the ground truth label images and right columns show corresponding output images from a network. (a) Right column contains output from a convolutional autoencoder in which all of its layers were trained for colorization using backpropagation. The colorization of this network was best among all network configurations and training methods. It accurately labeled each object in the images with appropriate colors, besides objects with very rare colors such as pink as shown in the topmost panel of this figure. (b) This is an output from five layer fully connected feedforward network. This network was able to accurately predict the general hue for each elements of the image, but the output images did not accurately capture the contours of each object and contained several pixels that were saturated which couldn't be trained. (c) Right columns show output images from models that were trained using a genetic algorithm. The network was able to produce several colors that often matched up with the colors in the label images. However, the variety of colors it was able to generate seemed to be more limited compared to both convolutional autoencoder and feedforward networks that were trained fully for colorization using backpropagation. (d) This shows outputs from a network that had the same architecture as the genetic model (trained on the second to the last layer output from a convolutional autoencoder trained on reproducing grayscale images) but was trained using backpropagation. We can clearly see that the model is great at recreating all the contours but is unable to generate fine tuned colors compared to the convolutional autoencoder that was trained fully on colorization.

Table 1: Training results for various model architectures and training methods.
This table shows a roughly calculated median loss at plateau for each type of model architecture and training method as well as the minimum loss value observed at any model training session. It is notable that the feedforward network outputs appear to be able to generate more varied colors when appropriate compared to either of the model that only the last layer was trained for colorization [Fig.3]. It certainly appears that the huge loss may be coming from the lack of ability to recreate the fine contours than learning colorization for the feedforward network, while the loss values for the network that was trained using latent representation are relatively small because it already knows how to recreate the contour of the image and can minimize loss by simply mapping darker shaded regions to darker colors and lighter shaded regions to some bright colors even without generating various color hues.

	Convolutional autoencoder	Feedforward network	Genetically trained model	Backpropagation trained model
Median loss at plateau	150	4500	6300	450
Min loss observed	60	1500	1800 (loss not observed at plateau)	150

that were roughly in the same tints as the label images, but not at the accuracy of the outputs from the convolutional autoencoders even with much longer training period [Fig.3,4].

5 GENETIC EVOLUTIONARY ALGORITHM

As our main part of the project, we trained various models using a genetic evolutionary algorithm on the colorization task in order to explore the hypothesis that octopus' ability to camouflage in color despite their monochromatic vision evolved through evolution. As each generation of networks is generated from random manipulations on weights and biases rather than calculated optimization towards a lower loss, a genetic algorithm typically takes much more number of training epochs compared to training a convolutional network using backpropagation. However, the run-time cost of

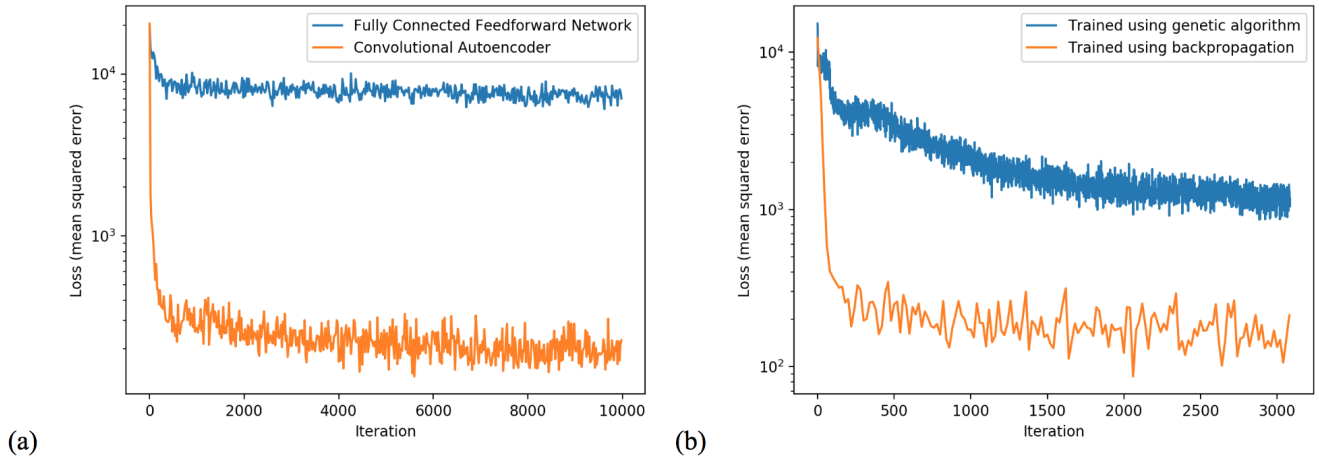

Figure 4: Learning trends for various model architectures and training methods.
(a) This plot shows the loss values (mean squared error between ground truth images and network outputs) of fully connected feedforward network and convolutional autoencoder over iterations. Both models were trained using backpropagation. Typically the loss value for feedforward network's seemed to plateau a little below 4000, although there were a couple runs in which the model had a loss value near 1000. Although it does a pretty good job on estimating correct colors, the relatively huge loss value could be due to the fact that this model could not learn to recreate fine contours of the elements in the images and had several pixels that saturated and did not learn even with training for many iterations [Fig.3]. In comparison, the convolutional autoencoder learned the task much faster and performed it with much higher accuracy (with longer training, the loss goes below 100). (b) This plot shows the loss values when training the last layer of a convolutional autoencoder that was trained to reproduce grayscale images to perform colorization using genetic algorithm and backpropagation. Because these models were trained on the latent representation from an already trained autoencoder, these models showed better colorization performance with significantly less number of iteration compared to the models shown in (a). As expected, the loss of genetically trained model takes much longer than the one trained using backpropagation, although it seems like the genetic algorithm model is continuing to learn while the backpropagation one converged after a couple hundred iterations.

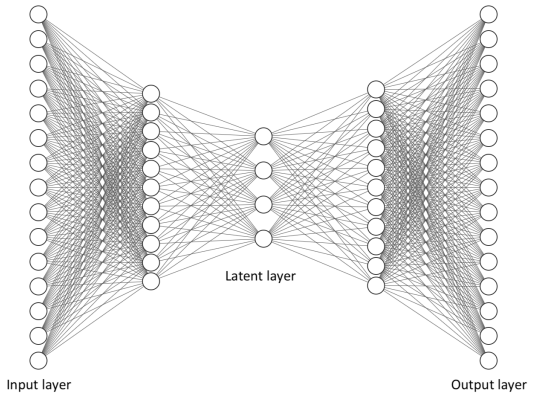

Figure 5: Schematic of a fully-connected five layer feedforward network
This figure illustrates the structure for the five layer feedforward network. The first and last layers each had 125 neurons. The second and fourth layers had 100 neurons. The latent layer (the innermost third layer) had 75 neurons. The overall architecture is simulating the general structure of an autoencoder which is consisted of an encoding part and a decoding part.

each epoch when using evolutionary algorithm is generally much less compared to training the models using backpropagation which includes calculating the stochastic gradient.

5.1 Exploring genetic evolutionary algorithm

A genetic evolutionary algorithm is an alternative method of training machine learning architectures that emulates the process of evolution. It starts by randomly generating a set number of networks, measuring the fitness of these networks based on some fitting function appropriate for a given task, and selecting a portion of the best networks to survive to the next iteration as the "parent" networks [Fig.6]. A new set number of networks are generated by "mutating" the parent networks, i.e. introducing randomized manipulations to the weights and biases of the parent networks. Training iterations when using genetic evolutionary algorithm include a cycle of measuring fitness of all models, selecting some portion of the models to survive, and repopulating back to the starting number of models by introducing mutations to the selected models. For our project, the fitness of the networks was determined by the mean squared distance between their outputs and the labels.

As the methods used for selecting which networks to survive and the amount of random mutations introduced per iteration heavily influence the success of training a model using genetic evolutionary algorithm, we explored several algorithms in order to thoroughly

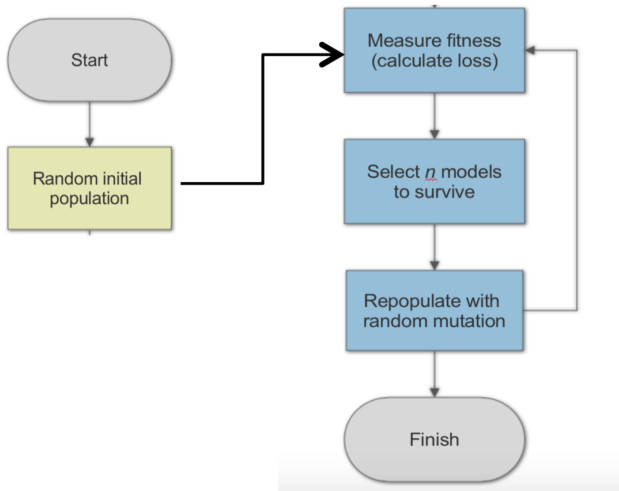

Figure 6: Genetic evolutionary algorithm.
Flow of model training using a genetic evolutionary algorithm.

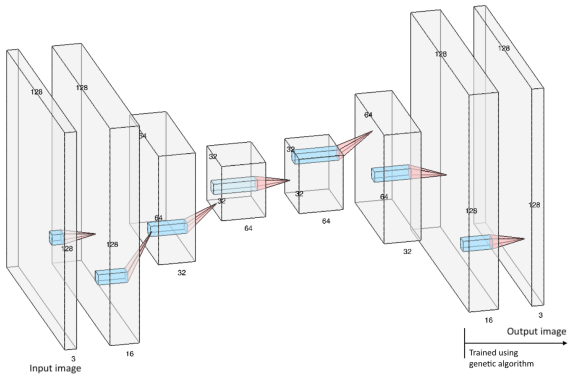

Figure 7: Architecture of the genetically trained model.
The convolutional autoencoder with the same architecture as shown above was first trained on reproducing grayscale images. Then, the last layer was replaced with numerous randomly generated networks that were trained using a genetic algorithm on the colorization task.

analyze the feasibility of using genetic algorithm as a method of training colorization of underwater images.

For selecting survivor networks, we first tried naively selecting a set number of networks with the least mean squared distance each iteration which brought problems that are analogous to "inbreeding." Soon after couple several iterations, we realized that all 65 networks in the population originated from one original parent model, severely limiting the variety in the pool of models that we are working with at a very early stage. While this would be a great feature if we ended up with a very promising initial configuration that could quickly get us to successful colorization with few random mutations, it is more likely to hurt the genetic algorithm process by providing lack of variety in the overall pool. As a result, we implemented a different method of selecting survivor networks in which we compared each parent network with only the networks that were generated from it and picked one best network among them to survive [3]. This selection method allowed the network to show plateau in the loss curve much more slowly compared to the naive selection method.

Observing that changing the selection method was not sufficient to propel the models from getting stuck and not being able to find a model with lower loss value, we added two more features to our genetic algorithm. The first addition was a tool to dynamically change mutation rate (proportion of weights and biases that are getting altered at each iteration) and mutation strength (how much the weights and biases would be shifted by), another important parameters for genetic algorithm. With an assumption that initial configurations are likely quite far away from successfully colorizing the images, we set the initial mutation rate and strength to be quite high [1]. Afterwards, we decremented the rate and strength as the loss value of the best model decreased to ensure that we are not taking too big of a step from a relatively good model. We further decreased the mutation rate and strength when we observed no change in the loss of the best model over many iterations regardless of the current loss value.

Another addition to the genetic algorithm was to introduce “migration” during early iterations. We introduced a set number of newly generated models and allowed the best model among these to survive to the next iteration in order to allow opportunities for finding a better starting configuration until the loss of the best model reached under a certain threshold. This was largely an effort to compensate the resource limitations of how many networks we can simulate at a given training session.

5.2 Training colorization using genetic evolutionary algorithm

Training a model that can successfully colorize was extremely challenging, but we were able to train models to colorize with a decent performance. These models took the output of the second to the last layer of a convolutional autoencoder that was trained to reproduce the grayscale images (but not on the colorization) as inputs and had colored images as the labels [Fig.7]. While these models' performance was not comparable to the colorization of the convolutional autoencoders trained using backpropagation, but they were still able to accurately label the images with several of the most common colors [Fig.3,4].

We also constructed a very simple network with two convolutional layers (one with 16 filters and another with 3 filters) that took grayscale images as inputs and trained it on colorization using the genetic algorithm. While we were able to find networks that accurately colorize individual images [Fig.8], it was difficult to train

[1] We believe that starting with lower mutation rate and strength would help, but we used high rate and strength to quickly observe evolution of these networks with fewer iterations due to the time constraints.

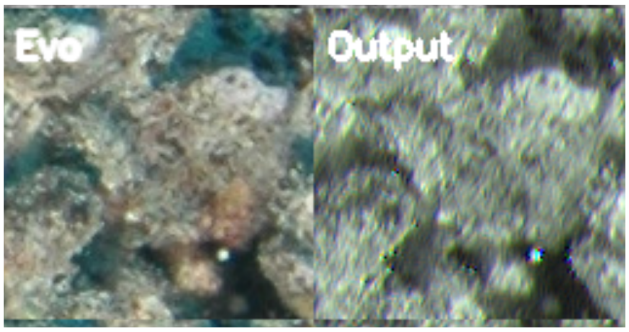

Figure 8: Output of genetic model trained on one image.
This image shows an example output of a two-layer convolutional model that was trained on grayscale images using a genetic algorithm. We were able to train it to color the image with reasonable tint, but it was difficult to test this on multiple images or for a longer duration as this required a lot of computation power.

a network that can successfully inference or can handle multiple images.

6 RUNNING ON SMALL HARDWARE

In order to visualize the results of our models, and to emulate the simple "machinery" of an octopus, we tried running the model on a Raspberry Pi. We first tried running a naive inferencing implementation on the Raspberry Pi 3 B+. This crashed due to a memory error, since the Pi 3 B+ has only 1GB of RAM, and computer vision tasks can be very memory-hungry if not done with care.

We then made some simple modifications to the genetic model in order to reduce memory consumption. Initially, we just loaded up the saved genetic model which included all 65 networks that were trained in parallel. We found it was sufficient and much more efficient to choose the best training model to be used for inference, and only run that one network while testing. Additionally, we initially ran test inference in batches of 16, the same batch size as used in training. We decided to reconfigure the trained network to process one image at a time in order to be more efficient for real-time image processing. This decreased the memory usage to a maximum of 900MB so that, with these modifications, we were able to run inference for the evolutionary model on the Pi. However, on the Pi, inference took around 3.1 to 3.3 seconds, and we wanted to see results in real time.

In order to speed up compute time, we attempted to run inference on a Movidius Neural Compute Stick (NCS) attached to the Pi. The NCS is a processor designed for real-time computer vision, and we believed that it would work well for our task. Models can be run on the NCS through the Movidus NCSDK or Movidius API, and there are three steps with which to do so. First, it is necessary to save a Tensorflow network using the Tensorflow `saver.save()` method. Second, the NCSDK compile function converts the Tensorflow network into a graph that can be interpreted by the Movidius API during inference. And third, from the NCS, the NCSDK or API runs inference on new inputs using the graph that

We encountered two issues in the compiling step, only the first of which we were able to overcome. First, the network must be compiled by the NCSDK, not the API alone, and the NCSDK can only be installed on a computer or VM running Ubuntu 16.04. Ubuntu 18.04, the operating system of the laptop we were using, is not compatible. However, we were able to circumvent this issue by installing the SDK on a virtual machine running Ubuntu 16.04.

The second issue was more difficult. When compiling the network, we got the error, “Stage Details Not Supported: ResizeNearestNeighbor”. The Movidius SDK does not support the Tensorflow function `tf.image.resize_nearest_neighbor` that we were using to process images. We have not yet succesfully overcome this issue to be able to run our model on the NCS.

Our takeaway from our experience with the Movidus Neural Compute Stick is that the software supporting it is not yet mature. Given the tight limitation in both operating system and Tensorflow/Caffe support, the NCS can be difficult to work with for novel tasks [1, 2].

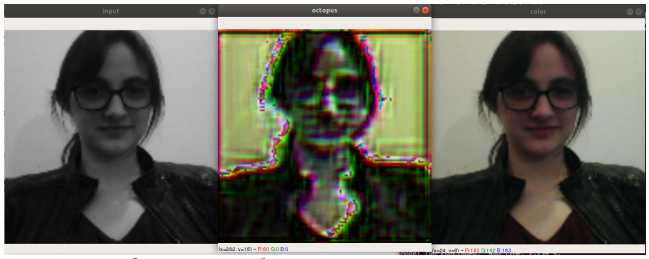

Figure 9: Inference on above-water images.
The left image is input, the center is the evolutionary model’s output, and the right is the correct coloring of the image.

7 VISUALIZING OCTOPUS CAMOUFLAGE IN THE REAL WORLD

When a cuttlefish, another animal in the cephalopod class, is put in a setting with color patterns very dissimilar from the ocean, such as a checkerboard, its camouflage becomes confused and inaccurate [4]. We wanted to do a similar experiment on our simulation, testing how it responds to images fundamentally different from those on which it was trained. In order to test this, we set up a live camera stream capture from a laptop and ran inference on these images. Each inference took approximately 2.0-2.5 seconds on the laptop.

We found that the inferences on non-ocean images had a different type of color distortion compared to the underwater images. In particular, the model struggled with sharp edges and very well-lit areas. This may make sense given the training data, since the coral images the network was trained on had few images with sharp edges and was mostly not well-lit. [Fig.9 provides an example of the model’s output for an above-water image

8 DISCUSSIONS

8.1 Weight comparison between training methods

In order to compare the nature of the models that are trained using different training methods (genetic algorithm and using backpropagation), we looked into the weights of the convolutional filters that were trained for the colorization task. All of these models used

Table 2: Average distance between weights.
Mean squared distance between the convolution filters extracted from the last convolutional layer from multiple models trained on colorization task were calculated. Key: GA = genetically trained model; BP = model trained using backpropagation; 1 and 2 indicate model obtained from a different training session.

	GA2	BP1	BP2
GA1	3.03	1.52	1.52
GA2	–	1.48	1.48
BP1	–	–	0.01

for this comparison were trained on top of the same base network that was trained on reproducing grayscale images. The filters from the models that were trained using backpropagation were very similar to each other with a mean squared distance of roughly 0.01 [Table. 2]. On the other hand, the filters obtained from the genetic model were not only quite different from the backpropagation models, but also significantly different among themselves [Table. 2]. Furthermore, the filter kernels from backpropagation models appeared more organized and structured than those from the genetic model which appeared quite random [Fig. 10].

One potential reason for such trend may be that each genetic model may be taking a significantly different route to finding the best network configuration and these models have not yet reached the most optimal state yet due to the limited number of training iterations. It would be interesting to observe if further training these genetic models and a decrease in colorization task loss correspond to the decrease in the distance between the filters of the genetic models to those from the backpropagation model. If the decrease in loss shows high correlation with a decrease in the filter distances, it may suggest that the genetic evolution is converging towards finding the same solution as the backpropagation models did. However, if those do not correlate or the distance between those filters do not get smaller, it may suggest that the genetic models may be solving the task in a different manner than the backpropagation model which may provide a room for an interesting comparison between the colorization behaviors between the models that were trained using different training method besides just simply looking into their loss values and using it to critique their colorization performance.

8.2 Potential improvements in the training method

A potential point of improvement in training that we did not have time to experiment was to use a different type of loss term other than mean squared error between the ground truth image and the output images as our primary goal is to observe if the model can perform colorization and reproduce the image suitable as a "disguise". In other words, the octopus does not need to camouflage to match the exact contour and fine detail of the scene underneath it. Instead, it just has to be able to blend into a various spectrum of colors and generate some pattern that matches a portion of the scene to successfully camouflage. With respect to this goal, mean square error might not be the most effective metric for training. For instance, we can observe that the loss of the network that was trained on the latent representation using backpropagation which was able to recreate the contour in fine detail had much smaller loss value compared to the feedforward network despite its relative lack of variety in the colors it is generating [Table. 1, Fig.3b, Fig.3d]. Hence, using another metric that encourages the model to be able to generate just the main contour patterns and colors may be much more realistic and effective for observing successful "camouflage" in our models.

8.3 Inherent limitations and challenges in modeling

In these experiments, we were able to train several networks to colorize with some level of accuracy using a genetic evolutionary algorithm, though the performance of these networks are still far below what we observe from octopuses or traditionally trained convolutional autoencoders. However, we observe several key limitations in training our networks as compared to the octopus.

First, we note that the cephalopod class has been around for 300 million years and may live up to about five years before reproducing, so they have had approximately 60 million generations to refine their colorization abilities. In comparison, our evolutionary models were trained for at most around 14,000 generations, several orders of magnitude less than the octopus.

Second, due to memory and computational limitations, we had to limit the number of models at each evolutionary generation to maximum 65. In contrast, there are millions of organisms of octopus per generation, providing a much more diverse pool from which to choose the fittest. We would like to note that given more resources, it would not be hard to run our current algorithm at a much larger scale. The evolutionary training step is easily parallelizable, since each parent and its children networks do not interact with other networks in the population, requiring no shared resources between different families of networks.

Third, the images we were training on were likely lower quality and provide less information than the coral scenery observed by octopi. While we tried to hand-pick images with a less blue tint, this limited the size of our training set. Further, our images were cropped small, limiting the amount of scenic information compared to what an octopus could see. Such cropped images may look almost identical as black and white images, while their color labels are different.

Finally, there are limitations to our evolutionary algorithm as compared to the natural process of evolution. We did not consider the process of crossing over, in which organisms exchange traits with others. Each of our surviving models is the descendant of exactly one initial random attempt, rather than a "best of all worlds" of an exponential number of ancestors. Additionally, we did not know how much variance to provide in the random mutation for each child. We erred greatly on the side of providing a high amount of variance, so that we could quickly see a change, but once the algorithm improves, a smaller amount of variance would allow it to hone closer to an accurate value. The octopus likely had a much smaller variance at each generation and was able to see these results

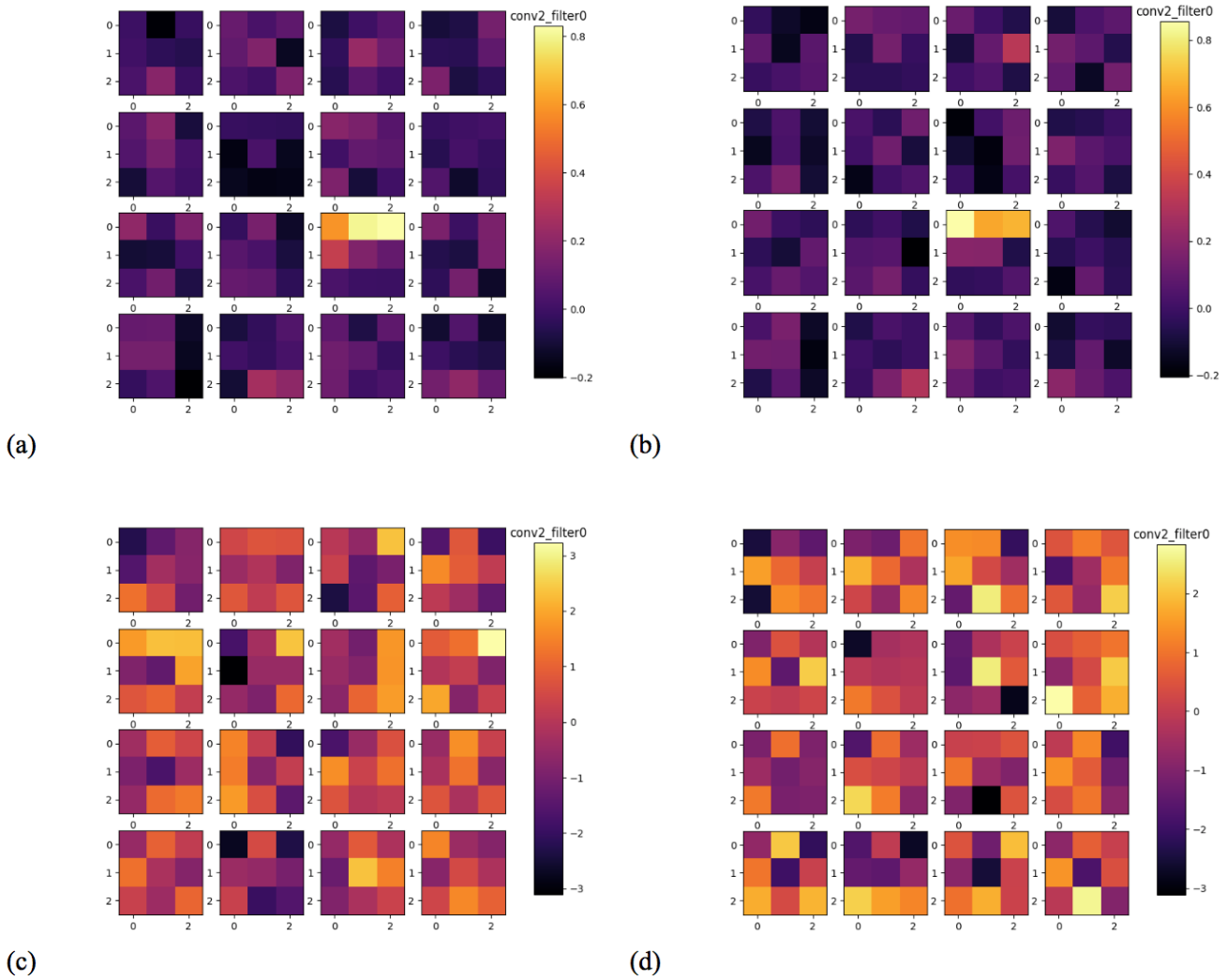

Figure 10: One of the three convolutional filters extracted from the last layer of models trained on the colorization task. The last layer of the models that were trained on the colorization task was a convolutional layer with three filters, each with a shape of 3x3x16. (a) and (b) represent filters from two different models that were both trained using backpropagation. We can observe that the patterns of these filters we observe are very similar to each other as supported visually in the figure and analytically [Table.2]. (c) and (d) are filters extracted from the last layer of two different models that were trained using a genetic evolutionary algorithm. Unlike the filters obtained from the models that were trained using backpropagation, these filters appear less organized and random.

because they had so many more generations that we were able to train for.

9 CONCLUSIONS

Through this project, we explored whether the octopus' ability to camouflage in color despite their black and white vision could have developed through evolution by modeling octopus' monochromatic visual system and camouflage in color. We hope to suggest that the possibility of training colorization using genetic algorithm appear plausible given that the longest training time using the genetic algorithm was several hours on CPU with a pool of 65 networks, though whether we can train a network with better colorization performance is still an open question as the models we were able to train showed limited colorization abilities.

ACKNOWLEDGMENTS

Many thanks to Professors Rick Stevens and Ian Foster of The University of Chicago who guided us through this project.

Catherine's role in this project was building a new dataset, developing neural network architectures (convolutional autoencoders, feed-forward networks, and evolutionary networks), exploring variations of these network architectures to find the simplest network

architecture that could perform colorization. She also explored the effects of various genetic algorithms and compared the genetically trained network to an analogous network trained using backpropagation. On the side, she simplified the architecture of the genetic network to decrease the memory usage and computation requirements to assist the hardware portion of the project. She wrote sections 3 through 5 of this report. Lucy worked with the Raspberry Pi and Movidius NCS to run the algorithms on a smaller hardware, analyzed the memory usage and computation time when the models are ran on different hardwares, and experimented with non-ocean camera images. She also implemented the framework necessary for streaming using images from camera input and displaying the network output on screen for real-time demo. She wrote sections 6 and 7. We worked together on the introduction and conclusion, methods, discussion, background, and abstract.

REFERENCES

[1] Edward Borasky. 2017. If you want to learn state-of-the-art computer vision algorithms and put them to work, this is the device you need. *Amazon.com* (Nov. 2017). https://www.amazon.com/gp/customer-reviews/R2TI1KS49PGCS6/ref=cm_cr_dp_d_rvw_ttl?ie=UTF8&ASIN=B076751BN8

[2] Amazon Customer. 2018. Examples out of date, documentation poor, burned too much time. *Amazon.com* (Oct. 2018). https://www.amazon.com

[3] Abdelhakim Gharib, Jamal Benhra, and Mohsine Chaouqi. 2015. A Performance Comparison of PSO and GA Applied to TSP. *International Journal of Computer Applications* 130, 15 (Nov. 2015). https://doi.org/10.23883/ijrter.2018.4270.s3bvz

[4] Simon Miner Lydia M.MÃďthger, Alexandra Barbosa and Roger T. Hanlon. 2009. Color blindness and contrast perception in cuttlefish (Sepia officinalis) determined by a visual sensorimotor assay. *Vision Research* 46, 11 (2009). https://doi.org/10.1016/j.visres.2005.09.035

[5] Justin Marshall. 2017. Vision and lack of vision in the ocean. *Current Biology* 27, 11 (June 2017). https://doi.org/10.1016/j.cub.2017.03.012

[6] Lydia Mathger, Nadav Shashar, and Roger Hanlon. 2009. Do cephalopods communicate using polarized light reflections from their skin? *Journal of Experimental Biology* 212, 14 (April 2009), 2133–2140. https://doi.org/10.1242/jeb.020800

[7] Alexandar Stubbs and Christopher Stubbs. 2016. Spectral discrimination in color blind animals via chromatic aberration and pupil shape. *PNAS* 113, 29 (July 2016). https://doi.org/10.1073/pnas.1524578113

[8] Richard Zhang, Phillip Isola, and Alexei Efros. 2016. Colorful Image Colorization. *arXiv* (March 2016). https://doi.org/10.1007/978-3-319-46487-9_40