Andrew Huston
4/22/23

Underwater Object Detection Overhaul: Infrastructure and Generalizability

Introduction

The University of Michigan Robosub Team is a student project team on campus. We develop an autonomous submarine to take part in an international competition that takes place every summer. The submarine has to do a variety of tasks to earn points, and the team that earns the most points wins. These tasks include navigating through a gate, hitting buoys with certain images on them, firing torpedoes, and other nautical themed challenges.

In order to be successful at these tasks, our system uses a camera and a machine learning model to detect different objects under the water that we may need to navigate to or interact with. In real time, the submarine can see the items that are around it, and our code will select which actions to take accordingly. A few examples of the detection from the submarine's perspective can be seen in Figure 1. If this machine learning model does not accurately detect the images, this invalidates all of the work that team members have done all year, since the system does not have the data it needs to make informed decisions. Unfortunately, the team experienced some issues related to this at our competition last year. The model was not performing well, and to make matters worse, we did not have the correct infrastructure in place to be able to easily train a new one. In order to mitigate these problems in the future, this project aims to fix our process of training a model and make it perform better in different environments.

Figure 1: Machine learning detection of various images needed for competition tasks

The first problem that needed to be addressed was our machine learning training pipeline. The previous process for training was very convoluted and wasn't practical for training new models on the spot in a pinch. Not only was it very difficult for beginners to follow along, but it also still took longer than ideal if you did know what you were doing. Since the training of a model uses a lot of computing power, we train our models on a university resource called the Great Lakes Computing Cluster. Many of the existing workflow's steps required manually uploading things to this remote computer, which took lots of time and could be easily automated. Also, depending on the state that the computer was in, the user had to manually complete a different set of tasks, making it hard to know exactly which steps they should do. This could be automatically detected and addressed accordingly. I determined this process could be made a lot simpler and faster, so it was on the list of things for me to do.

The next problem I wanted to address was the process we use when collecting new data to train our model. This case was a little different from the last. Instead of having a set process that was dysfunctional, there was really no formal process for collecting and processing data at all. Whoever collected it just did whatever they happened to think of with the data. This led to many issues, including not being able to find the original raw data and an inconsistent naming scheme. Most importantly, this led to not having a test set of images that could accurately evaluate the performance of the model. When training a machine learning model on images, it is important to make sure that the model works well on a test set. If you were to use the images the model was trained on, it would have already seen the correct answer while training and would provide a biased prediction. By setting aside roughly 10% of images to be part of this separate test set, it would ensure we have the resources needed to tell if our model is performing well. I knew a few scripts to implement this formatting and partitioning of images would make a good addition to this project.

The idea of being able to test our models carries over to the next goal for the project. When we are trying to train the best model possible, we need a method to compare two of them and choose which is better. This method did not exist in the past, so to tell if the model was working, we just passed a few images into it and checked to see if the predictions looked about right. This is not feasible going forward and hard data is the best way to compare two models. There are many metrics that grade the effectiveness of a model's predictions on a test set, but we

just need code we could run easily to get the numbers. Whether it involved writing code ourselves or seeing if any resources were available to use, the ability to better evaluate our machine learning models was a must for this project.

The final part of the project was to use the other changes I had implemented to try to increase the generalizability of our machine learning model. The pool that we test in on campus belongs to a hydrodynamics lab, and really isn't meant for swimming or any vision based tasks. Due to this, the water is very green, which is a problem when we train our model. An example of one of these green images can be seen in Figure 2. Since the only images the model has seen are green, it does not know how to respond when we put the sub in an environment that is not this color. This happened at our last competition, where the model that had worked so well at home no longer was performing well. To mitigate this in the future, we want to make our model more generalizable, so it will work well in many environments. One approach I decided to investigate is color correction. If we were to alter the colors before the model was trained, perhaps it could work better when not used in green environments. As the final part of my project, I set out to experiment with color correction in order to improve our model's generalizability.

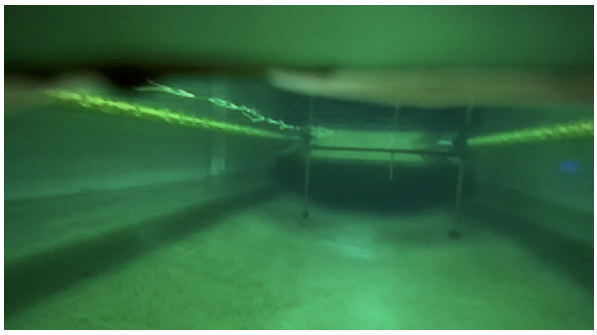


Figure 2: A green image taken while testing on campus

Overall, addressing these four issues will have a major impact on the success of the Robosub team at this year's competition. Our machine learning infrastructure will be much easier for any member of the team to interact with and the model will have the potential to perform much better.

Objectives

Based on the motivations laid out above, the objectives for this capstone project were to:

- Overhaul the model training pipeline
- Overhaul the data collection pipeline
- Update documentation for using these pipelines
- Determine method for model evaluation
- Experiment with color correction to improve model generalizability

Methods

Machine Learning Training Pipeline

The first step to understanding what needed to be changed in our machine learning training workflow was to understand the current method and what the pain points were. To do this, I utilized the method of workflow mapping. I made a diagram of all the steps that were necessary throughout this process, which illustrated where simplicity could be increased. This workflow diagram can be seen in Figure 3.

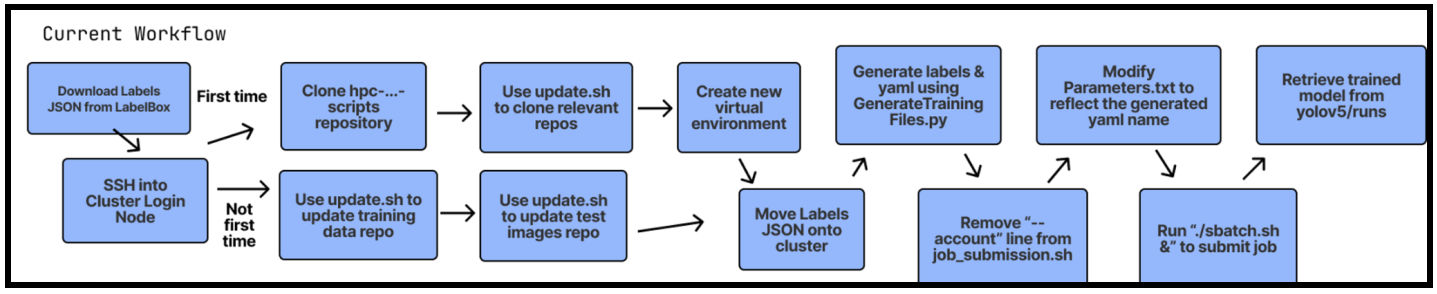


Figure 3: A workflow mapping of the original model training process

One thing I immediately knew I wanted to address was the split in the path that begins on the second step. It is more work for the user if they need to know about the state of the system and perform different tasks accordingly, so I thought this was something I could easily automate. This would squash the workflow down into a single path that would be most approachable to members of the team that do not work with machine learning as much. Another pain point I noticed was the uploading of files necessary to generate the labels that go along with the images. Not only is it extra work to generate those labels, but since we do this training on a remote computer, it requires effort just to get the necessary files there. There was one other step that I knew it would be easy to eliminate for simplicity. This was the step labeled “Remove ‘--account’

line from job_submission.sh". This line in the file had started to cause us errors with the remote computer, and we had found deleting it made things much smoother. I knew I could easily permanently delete it to remove the need for this step. After identifying these changes, this left me with an ideal workflow diagram which can be seen in Figure 4.

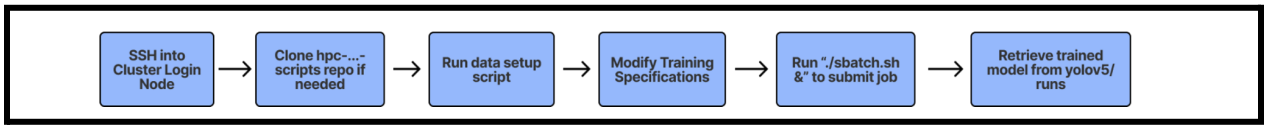


Figure 4: The ideal workflow mapping for our model training pipeline

The first step was to create the new data setup script. This script detected what files were already present and did the remaining actions needed to prepare the environment. It checked if all of the training and test data images were there, and fetched them if they weren't. It also made sure that the code repository we use to do the actual training of the model was on the machine. Finally, it made sure that there was a virtual environment set up that the training could activate when it wanted to install new python libraries. This could all be done with a few keystrokes, without having to make decisions and do things manually.

Next I eliminated the need for continuously generating image labels during the pretraining process. Since the items in the images do not move after the first time we label them, I realized there was no reason at all for us to be dynamically generating them when we wanted to train. I changed our data storage method to also store the labels, alongside their corresponding images. This works out perfectly, since the labels are static and dont need to be touched by us again. Then, we can simply remove the steps of uploading necessary files and running commands to generate labels, making our process quicker and simpler.

Last came the change of removing the problematic line that was causing us errors. The first thing I did was check in with the person who originally wrote that line to make sure it wasn't more important than I thought. He confirmed that he did not really know why he put it there, so I was good to remove it if everything continued to work well. I made this change to the master copy of our code to finalize the change, and did a run through of this new pipeline. It worked smoothly, and I had successfully reduced the workflow down to the ideal one.

Data Collection Pipeline

The way I approached the data collection pipeline changes was a little different, primarily due to the fact that there was no formal process that already existed. I did not have the ability to look at what we were currently doing and see how it could improve. Instead, I had to come up with the process from scratch, making sure to include all of the most crucial steps. After sitting down and thinking of all the necessary outputs we needed for this process, I came up with the workflow diagram from Figure 5.

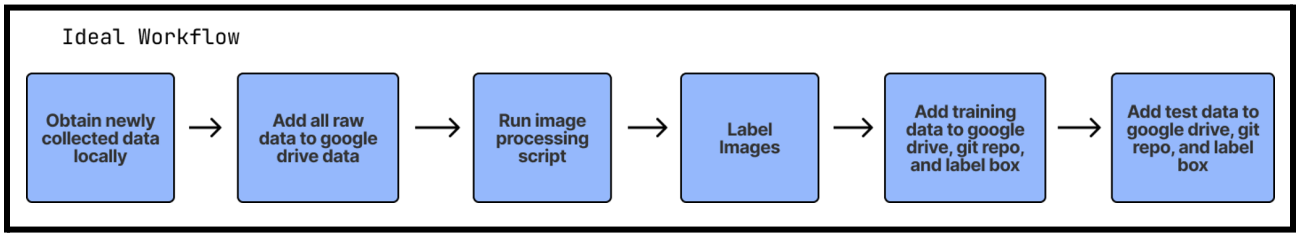


Figure 5: An ideal workflow for data collection and processing

This process begins from the point where the user has many images they have just taken from a testing session in the water. The reason that it does not begin before this point is because there are a variety of ways that the data is physically captured on our team. Sometimes the images are taken from the sub while it is underwater, other times we have a video and need to extract out the individual frames for training. Other times we use a gopro in a pool to take images, or even sometimes just use a phone camera to take pictures out of the water. It is easier to leave it up to the users to obtain a bunch of raw data, and then instruct them where to go from there. The first step is to upload all of that data to our google drive. The reasoning for this is to ensure we always have a backup of the images so all progress will not be lost if something accidentally happens throughout the process.

Next, I created an image processing script. This makes sure that the images are all the proper size and are the right file type. They all need to be 672 by 376 pixels in size and be a PNG for the training to successfully work. This image processing script also makes sure the image matches the naming scheme that our system uses. This puts the date collected, followed by the size, followed by an identification number as the title of our image, and is necessary for the labels to be generated later in the process.

Once the images are preprocessed, they are uploaded to a website called LabelBox for labeling. Members of teams draw bounding boxes around each of the objects in the images, which will help to generate the individual label file that will match the picture later. Once all of the images are labeled, a large JSON file is downloaded from the website that has all of the information about the bounding boxes that were just drawn.

Finally, the JSON file is passed into a script called GenerateTrainingFiles.py. This script performs many crucial jobs for the data collection process. First, it matches up each of the images with the LabelBox data and creates an individual label file that will forever be associated with the image. Next, it takes 10% of these image label pairs and sets them aside to be part of the test set. As mentioned previously, it is very important to have a separate test set in order to evaluate the amount of understanding the model really has about the objects it is detecting. Incorporating this into our official data collection process will ensure we diligently add to our test set whenever we collect new data. The script leaves the user with two folders, one with test data and one with training data. From there, all they need to do is upload those files so they are ready to be pulled onto the remote computer for training.

Model Evaluation Method

Next I needed to establish a method to evaluate the performance of a model, primarily so we could compare two and see which was better. While I thought I may have to research how to actually record some of these metrics myself, this part of the project actually turned out to be one of the easiest. I began by navigating to the website for the code that we use to train our models. I knew there were plenty of other files in their code base that we did not use, so I thought maybe one of them would be useful for me. After clicking around a few pages, I found that running the following line of code should do exactly what I was looking for:

python3 val.py --data $DATA --task test

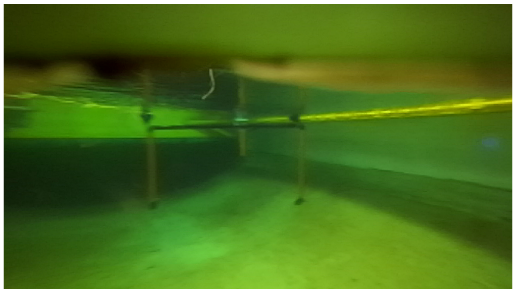
Val.py is a very useful program that can do many tasks, including testing how a model performs on different sets of data like a test set, a validation set, or even individual images. By telling it that the desired task is test, it knows to only run the model on those images designated as part of the test set. The $DATA variable would be a file path to a file that provides the

program a few more pieces of information it needed to know. Mainly, it would tell the program where it could find the specified test images, as well as where the model had been saved. The output of this command is very illustrative of the model's performance. Many graphs and metrics are generated. This could be done for two different models and their performance on the same images could be directly compared, which is exactly what we were looking for.

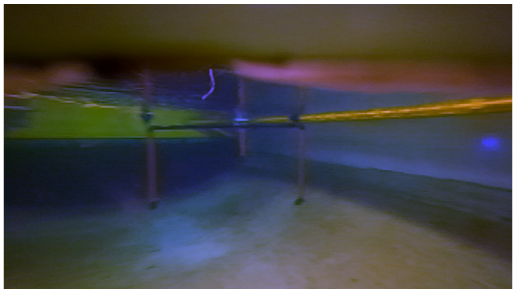
Color Correction

Finally came the task of making the model more generalizable. Since the team believed that the green color of our training images is what was primarily causing our bad performance at competition, it only made sense to begin experimentation with color correction techniques. I did a bit of research and found a good one to start out with for this project. It was called a gray world algorithm and would be pretty easy to implement. It took the value of each color channel and moved it closer to a neutral gray. This would help to normalize some of the more over-represented colors, which is exactly what I was interested in doing. The results of applying the algorithm to a green image can be seen in Figure 6.

While I believed that the gray world algorithm had some promise, my advisor had warned me to not expect too great of results from this alone. It is a very basic and naive approach, and relies on some color assumptions about the image that may be broken by our green environment. We decided to move forward with this approach, as at the very least, this would serve as a proof of concept that the other changes I had made to the machine learning infrastructure would allow for easy implementation, training, and evaluation.



No Color Correction



Gray World Algorithm Applied

Figure 6: Application of the gray world algorithm

Results

Training Time Reduction

The results of the new model training workflow were very clear. I performed 10 trials of setting up the remote computer for training, each time from scratch, with both the old and redone setup methods. The purpose was to see how much the changed infrastructure had brought down the time. The average preparation time previously had been about 6 minutes and 23 seconds. After the changes I made, it could easily be done in about 1 minute and 39 seconds. Additionally, most of this time was now spent waiting for all of the images to be downloaded onto this remote computer instead of having to work as the user. The data can be seen illustrated in Figure 7.

It is also important to note that the pre-overhaul data is done by someone who already knows how to work with the system in the old way. If it were a member of the team who had never worked with the machine learning components before, it certainly would have been much longer. One of the improvements from the overhaul is simplicity and reduced burden on the user. This is much harder to quantify with hard data, but since there are less decisions for the user to make, it is simple to just follow along with an established guide. Overall, the changes have really made the training preparation quick and concise.

	Average Training Prep Time
Pre-Overhaul	6 minutes, 23 seconds
Post-Overhaul	1 minute, 39 seconds

Figure 7: Table of model training preparation times

Color Correction Results

A model trained on our original images can be compared to the model trained on the images that had the gray world algorithm applied. Some of the evaluation metrics can be seen in Figure 8. These figures plot the precision of the model for different objects corresponding to the confidence level that the model determines for that prediction. Ideally, the precision should be high everywhere, but especially when the model submits its prediction with a high confidence. Each line represents the results for one specific class of objects the model is trained to detect, and the dark blue line represents all of the different object types combined. We can see that there is

not much of a significant difference between the performance of the two models. There are a few points where performance may have increased a bit, such as where the green line's precision no longer dips down around a 0.8 confidence. Despite the few increases, the improvements do not seem to justify the overhead of applying the correction to all images and investigation of different methods should continue in the future.

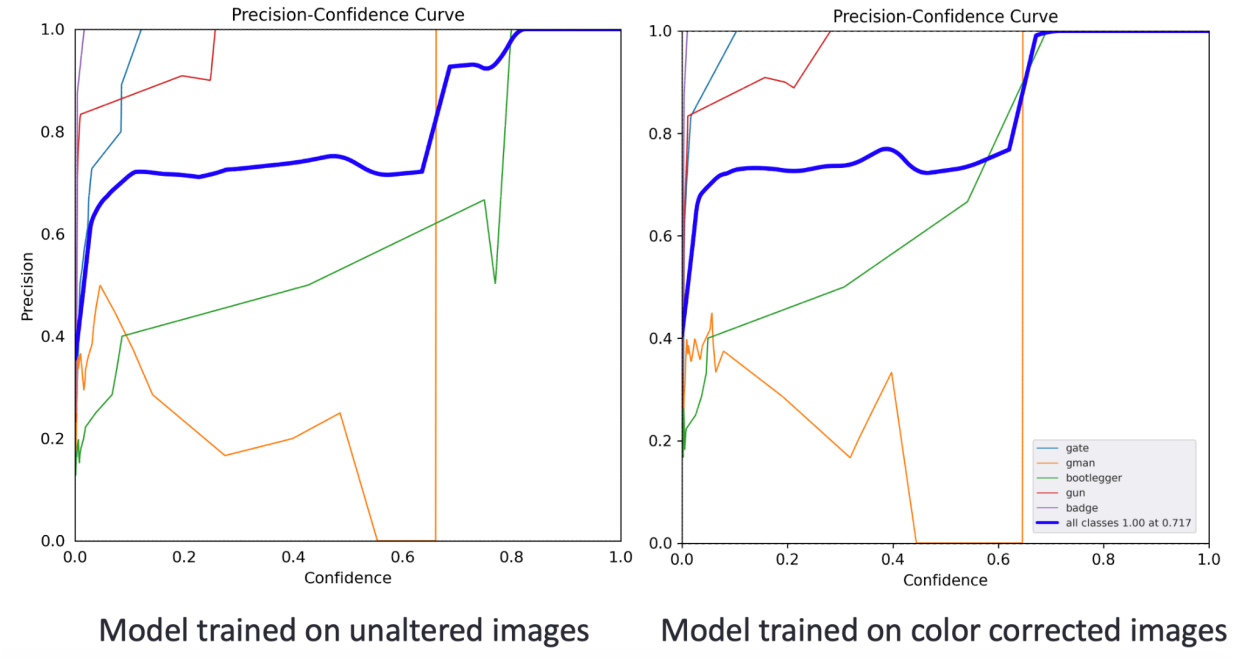


Figure 8: Evaluation metrics for the original and color corrected models

This experiment was still a large success. It showed the changes made to the machine learning infrastructure have made it much easier to iteratively train models and measure improvement. Not only did the overhauled workflows make the process of training these models very simple, but the evaluation script that produced the figures was also a great help. Without this evaluation method, we would have no way of knowing that the color correction technique had minimal benefits. A final observation is the jaggedness of the generated figures. There are many sharp points instead of smooth curves one might expect to see. This must be because of the limited test set size, as it only included around 50 images, which is pretty small. Continued use of the new data collection pipeline will ensure we add sufficient images to our test set in the future to get smoother, more illustrative curves.

Conclusion

This project was a great experience, being able to merge the project team I am passionate about with a project that I have the ability to lead and make decisions on. I learned a lot during the process, like how to do effective research. I used this when finding a method for model evaluation and finding a color correction algorithm. I also learned that planning might be the most important step of a project like this. If I had planned a little more in depth about what the model training pipeline should look like before I started implementing, I definitely would have saved some time. I used this knowledge when implementing the data collection pipeline, and that went much smoother.

All objectives that I set out to do were completed successfully. The model training pipeline is much easier to use now and takes much less time. The data collection pipeline is now well established and will ensure we have sufficient images in our test set. I have updated the documentation on how to use each of these processes. There is now an easy way to get data on how well a model performs on our test set as well. Finally, there is a good foundation on how to apply color correction methods when training models in an attempt to increase generalizability. While the method I tried did not work very well, the system is in place to try something else easily. The next step is to research more complex and cutting edge methods of color correction, such as histogram normalization, in an effort to find one that truly does increase the model's generalizability.

Overall, the changes made to the machine learning training pipeline and data collection pipeline give the Robosub team the flexibility to train new models quickly in the presence of new data. Additionally, the framework established can be used to find methods of making the model even more generalizable. Overall, this project has greatly increased the team's potential for success in the yearly competition.