Bryan Egan
Individual Report for the Final Project - DATS 6203

Introduction

Plankton are a necessary component of our ecosystem. They are small and microscopic organisms that float or drift in the ocean, providing a critical food source for larger organisms as they are the first link in the aquatic food chain. Since they are the primary food producers for ocean life, a decrease or loss of these fundamental species could detrimentally impact our ecosystem. Thus, it is not only crucial to measure and monitor plankton population with underwater cameras, but to be able to classify the high volume of images rapidly, as manual analysis would take about a year for images captured in one day. Therefore, to address this efficiency problem regarding image recognition, we will be applying deep learning techniques to build a convolutional neural network to classify images of various plankton in a timely and accurate manner.

Data Background

Our dataset was obtained from Kaggle and includes colored images of plankton captured from Oregon State University's Hatfield Marine Science Center. We were provided with a file containing folders of images for each class. Looking at the image below, the original plankton data contained a hierarchy of classes with 11 overarching species (large blue), 7 subspecies (small blue), and various remaining subspecies (red).

To lessen the size and complexity of our classes, we simply rolled up each subclass into their main, overarching class, resulting in a total of 11 classes. Next, we eliminated any classes that did not provide a sufficient number of images to train and test on, specifically, classes with less than a few hundred images. As a result, for our final data, we will be using a total of 8 classes with the corresponding number of images: *Chaetognaths* (3,443)*, Crustaceans*(4,792)*, Detritus* (2,182)*, Diatoms* (1,019), *Gelatinous Zooplankton* (8,024), *Other Invert Larvae* (1,445), *Protists* (3,808), and *Trichodesmium* (3,419).

Individual Work

Iliana and Alexa took the lead in regards to pre-processing. Since I work full-time, Alexa and Iliana were able to step up and handle most of the details regarding downloading, normalizing, and resizing the data. They also split the data into test, validate, and train sets.

I started working on the models. My biggest contribution in the beginning was in regards to the ImageDataGenerator function in keras. I found the documentation on the keras site and helped write the code. I helped figure out how to implement any of the variables within the function for testing and ran a multitude of codes pertaining to the ImageDataGenerator. I looked at normalization using both

features and samples, using sharpening, and also looking into changes in both the height and width. I wrote all the code for the ImageDataGenerator myself While I used the documentation as a reference, I did not need someone elses code to get it to work.

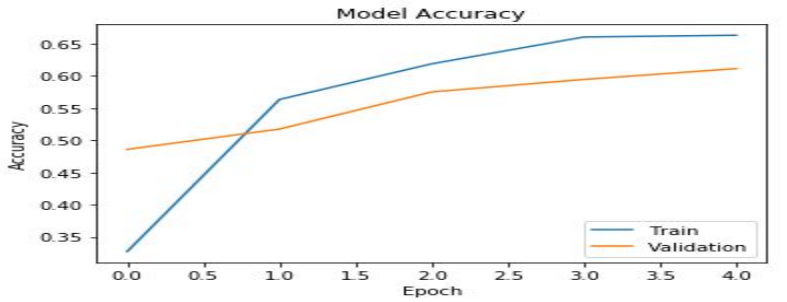
On top of also testing versions of our respective models, I also found the keras.history function so we could easily plot the accuracy and loss values. Keras stores all of the values created in model.fit into a dictionary. Which makes storing and accessing the data easy and accessible. While I did need some help from the original source, I only used them to find out how to access the values - plt.plot(history.history['loss']) - and also best practices for legend location. I wrote all the other code. Of the 15 lines of code pertaining to the plots, I wrote 11 of the lines. Using the rubrik from class, ((4-0) / (4+11)) * 100 = 26.667.

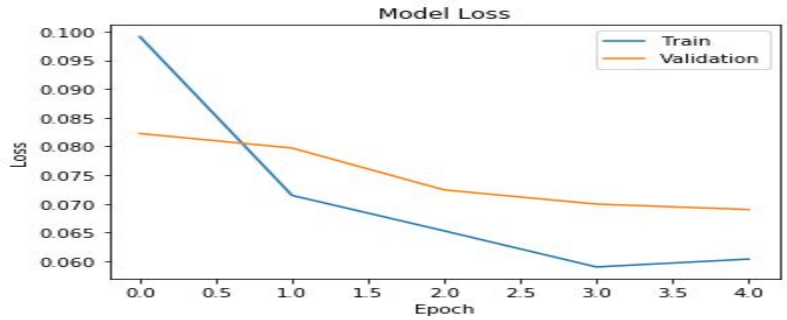
My other main portions of the project were on both the presentation and the rewrite of our respective proposal. We were initially planning on working on another dataset. That set unfortunately proved extremely difficult to use. Once we found and accessed the plankton set, I rewrote our proposal to reflect the updates to the set, our model, and everything else.

Summary of Findings

Overall the models that did the best tended to be simpler models. Based on our findings, the simple model using mean squared error and everything else remaining the same as the original model was the best overall when it comes to the validation accuracy. Interestingly, more often than not the models did a good job accounting for variation in the training models, but once you moved on, the model was far less accurate for validation or testing. What this tells me is that our model was overfitting.

I tested a version of the model using the versions of each of the parameters - best loss, best optimizer, best number of layers, best dropout, etc and put them all together. While the model did a good job of not overfitting the training data, it was surprisingly worse than what I saw with the more simple models. What this tells me is that while this model is overall better than the simple model, it is still not very good and in need of more fine tuning in the future.





Conclusion

Overall this project taught me a great deal about how to create a convolutional neural network. The biggest takeaway for me though is that sometimes, a simpler model may be best. This could mean using less parameters, or simply not throwing things into the moel just to see how they affect the outcome. I would love to try again on a similar topic to see if i could create a better model. CNNs seem to do a better job than your average model and think they are the way of the future for image recognition.

References Used

For Keras.history:
Display deep learning model training history in keras

Keras documentation:
Keras Documentation