Proposal of

Applied Neural Networks: Aquatic Life Identification

A project

submitted by

John Schulz

For ECE 468 Neural Networks

Final Project

Department of Electrical Engineering

Peoria, IL

4/25/18

Project Proposal: Aquatic Life Identification
Number of Group Members: 1
Student: John Schulz

Abstract

There exists little to no practical or efficient solutions for controlling invasive species in the world's oceans and local waterways. The first step in controlling a population is identification. Although, identifying invasive species can be done by training humans, it is not a viable or cost-effective approach when it comes to marine life. The research that will be conducted for this project will be a proof of concept to apply a deep neural network to identify marine life and implement it in a resource efficient manner that the core functionality can be deployable on a low cost computing device or smartphone.

Proposed System

The aim of this project is to utilize video captured from a couple of different species of aquatic life in a contained environment like a tank. Video data will be captured at 30 frames per second and will have a certain number of frames per seconded converted to a common image data type. A human must parse and sort the data into feature types because no preprocessing neural network will be used. The pictures will be scaled down in resolution, but RGB (Red Green Blue) data will remain and be utilized by the neural network for increased accuracy. The features of the system are not only to identify a couple species, but to be able to sufficiently deal with false cases, such as an empty space or multiple aquatic animals. In order to deal with outliers in the data, images that are blurry or barely recognizable will be excluded from the training, testing, and verification data. The current plan is to utilize images of the subjects positioned at any angle in accordance to the camera. The number of subjects in one image will be limited to 2 or less, with the preference on using pictures containing one subject for this project.

Model Design

In designing this artificial neural network, a Restricted Boltzmann Machine (RBM) deep feed forward network utilizing the backend support of Google's Tensorflow architecture and the networks weights are exported using the Keras Sequential model framework [7]. The neural network is written with Python 3.6 in the Thonny Integrated Development Environment (IDE). The models weights are imported into Apple's XCode IDE to be deployed on the Neural Engine 1.0 in Apple's iPhone X. The model in Keras is then exported in the .h5 modeling framework and converted to Apple's proprietary coremltools python software, which exports the new model in their ".coreml" architecture [1, 8]. An advantage of using Core ML, is that it can dynamically split the load between the CPU (Central Processing Unit), GPU (Graphical Processing Unit), and the Neural Engine 1.0 depending on how efficiently the model is running [1, 9-10]. The model is executed in real time in the FinFind application written in Swift 4.0 and the application supports all devices capable of running iOS 11.

The approach of this project is to have enough hidden layers for the network to accurately hypothesize the target species. Variables such as the number of epochs and learn rate were experimented with throughout training runs to obtain optimal decision making. The tolerances for the system's accuracy to predict the correct answer was held to a high level as would be needed for feature recognition of aquatic life. The loss function that was selected for the model was the Stochastic Gradient Descent function or SGD, and the model's optimizer utilized Keras' mean square error function. The resulting output of the model was trained using hot-key encoding [7]. Micro-batching and various batch sizes were experimented with to maximize the number of nodes that could fit onto the GPU during training. The goal of the Core ML model in FinFind is to produce the predicted feature name and a floating point hypothesis.

Prediction

The function of this neural network model is for video data of fish to be feed into the network in order to produce a hypothesis on the specie(s) of the aquatic organism in real time. A sufficient data set required for each species will likely be around 15-30 minutes of VGA resolution video, because the subjects will be held in the controlled environment of a tank. The aim is for the network to produce the correct answer at or above 80%; however, the results will probably be closer to 60%. Shape detection will be vital to the overall success of the network. Consequently, it is expected that the network will take a considerable amount of training time for live subjects like fish that are constantly changing in position ever so slightly from frame to frame.

Hypothesis

The accuracy of the neural network will depend heavily on three key points: the amount of learning data collected, the variety of environments that the subject exists in, and the amount of training time that can be obtained. The difficulty associated with reaching accuracy in the 75th percentile with less than 30 minutes of data per species will be because the subjects will not always be positioned broadside towards the camera where aquatic species are most discernable. The system should increase greatly in accuracy once the network learns physiological traits like gills size, fin positioning and shape, lateral lines, exterior color, and facial anatomy. Therefore, detailed traits will reinforce the identification process along with RGB data will aid in maximizing the deep neural networks overall prediction capabilities.

Methods

The architecture of the network will utilize a classic RBM feed forward network combined with a ReLU (rectified linear unit) activation function. The final activation layer will include a standard sigmoidal function. During the neural networks learning process, batches of data will be randomized in a recursive manner with each batch and micro-batch being mixed. An adaptive learning rate was used in the models optimizer. Training time was analyzed by viewing log files of the model's accuracy and loss function. No time or accuracy ceiling was implemented, the only limit was on the number of epochs.

Tech Details

1. Data
    a. Video Resolution Captured: VGA (640 x 480) minimum to HD (1240 x 720) maximum
    b. Still Images Resolution (Downscaled): QVGA (128 x 96) PNG
    c. Batch Size: 100 images
    d. Micro-batch Size: 50 images
    e. 70% [7 min] for training, 20% [2 min] for testing, and 10% [1 min] for validation.
2. Hardware: Specs
    a. Machine 1: Programming and Batching Computer
        i. Late 2016 MacBook Pro 15”
            1. Operating System: MacOS Sierra 10.12 & High Sierra 10.13
            2. Processor: Intel i7-6820HQ (4-cores) @ Base Clock: 2.6 GHz
            3. GPU: AMD 460
            4. Ram: 16 GB LPDDR3 2133 MHz
    b. Machine 2: Training Computer
        i. Early 2018 Alienware 13 R3
            1. Operating System: Windows 10 Pro 64-bit
            2. Processor: Intel i7-7700HQ (4-cores) @ Base Clock: 2.8 GHz
            3. GPU: Nvidia GTX 1060 (non-Max Q)
            4. Ram: 32 GB LPDDR3 2400 MHz
            5.
    c. Machine 3: Embedded Device
        i. iPhone X
            1. Operating System: iOS 11 64-bit
            2. Processor: A11 Bionic (6-cores) Modified ARMv8 @ 2.4 GHz
            3. Co-Processor: Neural Engine 1.0 (600,000 MIPS)
            4. Ram: 3 GB DDR4
    d. Imaging Device 1
        i. Cannon 70D
            1. Sensor: 20.2MP
            2. Lens: EF-S 18-55mm f/3.5-5.6 IS STM
    e. Imaging Device 2
        i. Apple iPhone X
            1. Sensor: 12MP
            2. Lens: f/1.8 or f/2.4

Timeline

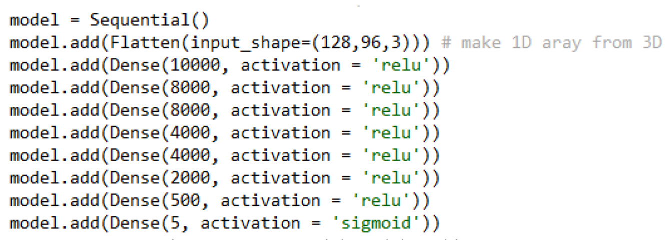
TABLE I
Project Timeline

Date	Description
2/16	Began Theorizing Aspects of the Project
3/5	Presented Project Proposal
3/26 – 4/2	Collected Video and Imagery Data of Fish
4/3 – 4/5	Sorted, Parsed, and Processed Imagery Data
4/9 – 4/10	Converted Imagery Database into Text Files
4/11	Read Database Text File to Create Batches
4/11 – 4/14	Programmed Neural Network in Keras API
4/15	Created a New Image Database in a Smaller Resolution
4/16	Began Training Machine Learning Model
4/17 – 4/20	Training and Exporting Models Using Core ML API
4/20 – 4/21	Wrote iPhone X Application: FinFind, in XCode
4/22	Finish Documentation

Architecture

The resulting architecture chosen was a deep feedforward model with decreasing layers to maximize the number of nodes that could be trained on the training computer; therefore, a tradeoff with network precision was made by switching the TensorFlow backend from Float32 to Float16 to maintain a sufficient ratio of input nodes to image pixels. Using RGB images with a resolution of 128x96, the resulting number of nodes is 36864. The highest ratio that could fit onto the GPU during training was approximately 1 input node to 3.7 pixels, thus the first hidden layer contains 10,000 nodes, as shown in figure 1. The architecture of the following layers continues the decreasing stair step profile, until the final layer which must be equal to the total number of features used in the network.

Two Keras model architectures were tested before the final implementation: Input/Output and a Sequential model, both used a ReLU function in all hidden layers and a sigmoid function in the output layer as shown in figure 1 [11]. During the testing process of the system, the Input/Output model was more memory efficient and allowed larger networks to fit onto a GPU during training than a Sequential model, but the former lacked the flexibility of being able to receive two dimensional or higher arrays and include a flattening layer. Although, an Input/Output architecture produced greater accuracy, it is not optimal when the target system is an embedded device where the goal is for video data to be directly fed into the model. Thus, a Sequential model was selected because the video data could be flattened discretely by the model on the GPU, rather than on the CPU.



```
model = Sequential()
model.add(Flatten(input_shape=(128,96,3))) # make 1D aray from 3D
model.add(Dense(10000, activation = 'relu'))
model.add(Dense(8000, activation = 'relu'))
model.add(Dense(8000, activation = 'relu'))
model.add(Dense(4000, activation = 'relu'))
model.add(Dense(4000, activation = 'relu'))
model.add(Dense(2000, activation = 'relu'))
model.add(Dense(500, activation = 'relu'))
model.add(Dense(5, activation = 'sigmoid'))
```

Figure 1. Sequential Model Architecture.

As stated under the methods section, accuracy values during training were key in tuning the system, both mean squared error and categorical crossentropy loss functions were tested; however, the mean squared error function was selected for training the model because the calculation process is more resource efficient. In the case of Keras, the mean square value is shown below in equation 1, from the publicly available source code [4].

$$\bar{\boldsymbol{x}} = \text{keras.mean}\left(\text{keras.abs}\left(\vec{\boldsymbol{y}}_{\boldsymbol{pred}} - \vec{\boldsymbol{y}}_{\boldsymbol{true}}\right)\right) \tag{1}$$

Consecutively, the variables y_{pred} and y_{true} are each single column vectors with the same number of rows as features. The variable y_{pred} stands for ‘y prediction’ representing the hypothesis; likewise, y_{true} is the known correct value that the mean squared error is calculating the prediction against. The results of this equation maintains the same column row dimension of y_{pred} and y_{true}. The Stochastic gradient descent optimizer implemented in this project utilized the default adaptive learning rate, which accelerates the SGD “in the relevant direction and dampens oscillations,” and updates the learning rate’s decay each epoch [3], [7] . Overall, the final architecture implemented as shown in figure 1, contains a Sequential model with progress monitored by a mean squared error loss function and optimized using the Stochastic gradient descent method.

Results

The database created consists of 8088 self-collected, parsed, and processed images, consisting of three unique species of aquatic life along with two other quantitative features consisting of mixed and empty images. All five features are shown in figure 2.

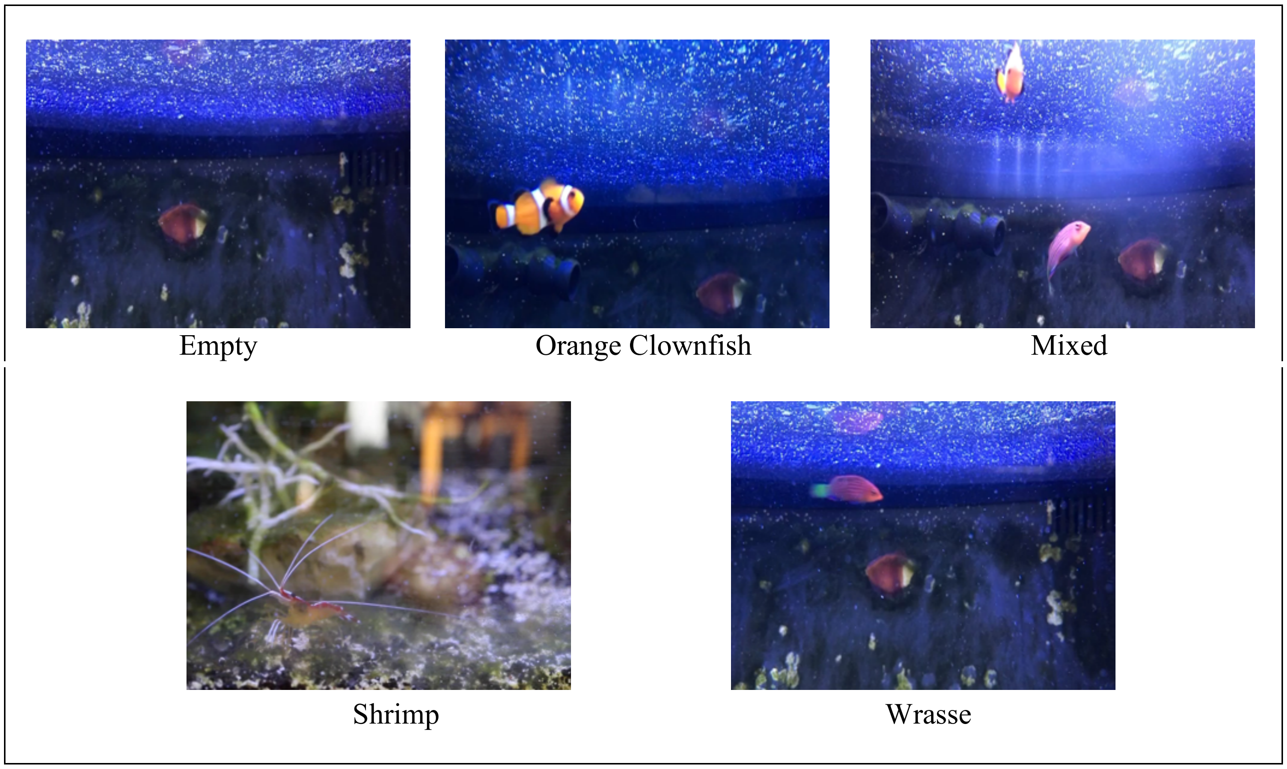


Figure 2. Sample of Each Feature in the Database

The database comes out to be approximately 24 minutes of video collected with an average of 6 frames kept per second. The 5 unique features are distributed in a 70/20/10 percent manner with 70% or 5,662 images obligated to learning, 20% or 1,618 images for testing, and 10% or 808 images saved for verification, as presented below in figure 3. Image selection when creating the 70/20/10 database was a randomized procedure and image capturing was also randomized because the subjects are in an uncontrollable environment. In conclusion, it is clear from the data that the more social species are much easier to gather data from because they do not try to hide, as shown by the Clownfish occurring more than three times that of the Wrasse.

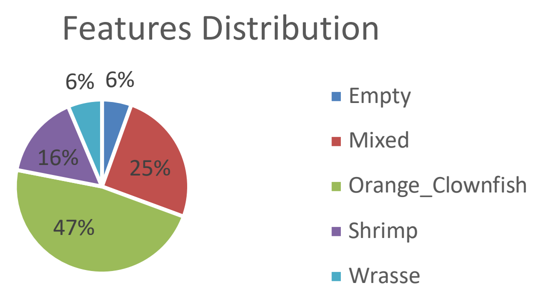


Figure 3. Distribution of Features.

The results in table II, show the mean accuracy that the model produced during its training cycles. Overall, the accuracy is within the range proposed in the hypothesis section of this paper, which considers good accuracy to be between 60 and 80%. The mean values were calculated from the log file that were saved after each epoch and the model as whole was saved through a checkpointing method every 50 epochs.

Table II
Mean Value Over 6000 Epochs

Type	Mean
Mean Training Accuracy	0.7686
Mean Training Loss	0.0660
Mean Verification Accuracy	0.7297
Mean Verification Loss	0.0760

The reason verification accuracy is less than training accuracy is because the network has never seen the verification data before and due to the fact that all of the verification values saved after each epoch were the result of a mean average taken over 10 random images. The mean accuracy and loss values in table II are approximately equal to each other, as should be expected. The collection of the data from the log files plotted in figure 4, did not contain any holes or extreme abnormalities. The trend in the top subplot below trends toward greater accuracy in both training and verification lines and the bottom subplot decreases in loss as the system learns. Thus, the model appears to have enough data to reach sufficient accuracy levels and the results from the plotted log files show that it trained effectively.

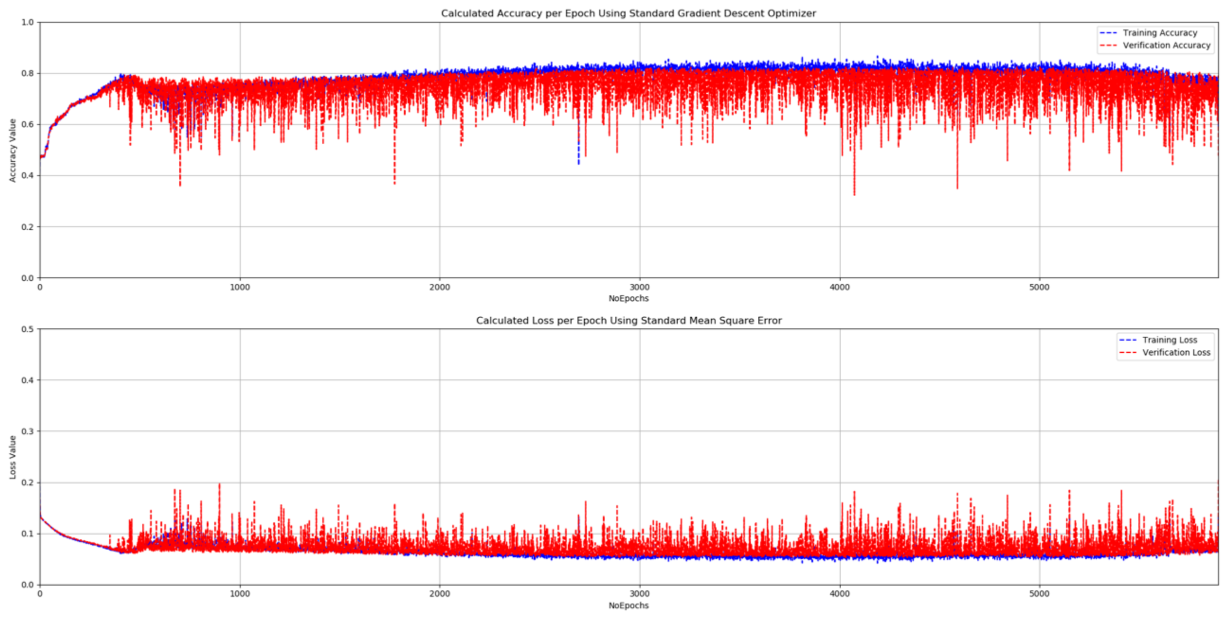


Figure 4. Model Accuracy and Loss Plot from Log Files

Implementing the model on hardware requires that the Keras model be exported entirely in the .h5 architecture, which includes the models structures, weights, names of features, floating point value used during training, type of input, and an input node scaling factor. This allows for import solutions, whether it is an embedded system or Apple's coremltools, to know that the input is an RGB image. In addition, the Keras model describes the value that the input nodes must be scaled to 1/255 for color images, whether to execute the model in Float16 or Float32, and lastly to let the importing system know that the output of the model will contain an identifier data type of a string array of the models features. Thus, once the Keras model is exported in .h5 format, then it can be imported by Apple's coremltools python library to be converted, and lastly exported as a "coreML.model" to be used in Apple's XCode IDE [8-10].

Figure 5. FinFind Application Logo.

The Core ML model is used inside the custom application FinFind designed by John Schulz; the logo is shown above in figure 5. The simple and clean UI shown in figure 6, allows for the user to view in real time through the use of the 12 megapixel camera to explore an encompassing environment and have the model display the name of the animal being viewed with the model's confidence to the right of the name. In software the video data is handled by the standards set by Apple's AVFoundation API (Application Programming Interface) that holds the frames in a pixel buffer [5-6, 10]. Once a signal by the Core ML model is given, Apple's Vision API grabs an image from the buffer and downscales it on the GPU to the models desired 128x96 RGB image resolution [2, 8-9].

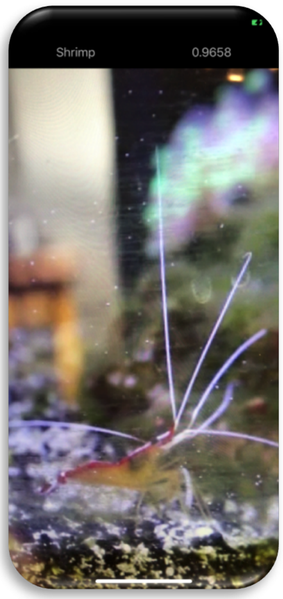


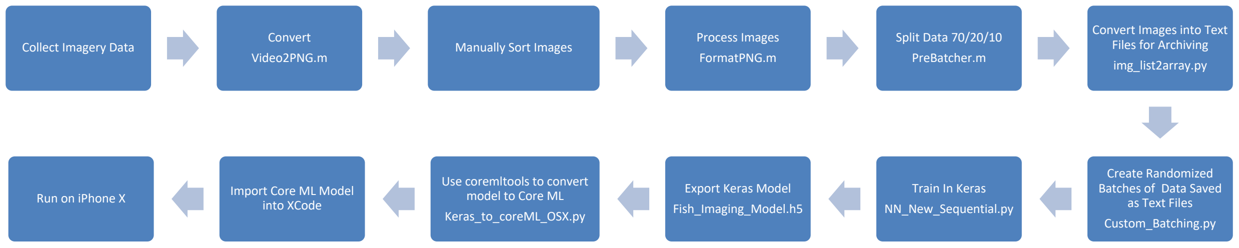
Figure 6. FinFind User Interface on the iPhone X.

Since Core ML leverages the CPU, GPU, and the neural engine; the application can make two to three predictions a seconds without any video stutter, while using less than 100 MB of RAM (Random Access Memory) and approximately 17% of the CPU across all 6 cores [1]. The application can be immediately exited and restarted without the need for the application to load, but decision making by Core ML will lag until the model is fully loaded onto the GPU again. Confidence of the model ranges from around 30% to 90% depending on the image. The application is capable of displaying a "Can't See" message when the model sends out a nil results due to Core ML not recognizing either anything changing or nothing that the model knows. Apple does not disclose any further details on what levels it takes for the Core ML framework to output a nil results. FinFind does have safety nets in place to guard against nil results from crashing the application.

The outcome of the model and its implementation on hardware has produced the desired results of being able to identify the five trained features. The data collected and used in the training of the Keras model is considered to be accurate and categorically correct. Results collected during training and in FinFind do not present any outliers or undiscernible traits on its target subjects. Consequently, the models performance is as hypothesized, hence it produces results in the ranges seen during validation testing and in the FinFind application.

Discussion

The process used to create the software and hardware implementation are shown in the flowchart below in figure 7. Four main areas exist in the creation of this project: creating the database in steps one through five, archiving and creating batches from the database using text files in steps six to seven, training the model in Keras in steps eight and nine, and lastly building an application on an embedded system for the model to run on.


Collect Imagery Data
Convert Video2PNG.m
Manually Sort Images
Process Images FormatPNG.m
Split Data 70/20/10 PreBatcher.m
Convert Images into Text Files for Archiving img_list2array.py
Create Randomized Batches of Data Saved as Text Files Custom_Batching.py
Train In Keras NN_New_Sequential.py
Export Keras Model Fish_Imaging_Model.h5
Use coremltools to convert model to Core ML Keras_to_coreML_OSX.py
Import Core ML Model into XCode
Run on iPhone X
Figure 7. Project Flowchart

Improvements to this process could be made by writing the new drivers in Apple's Vision API to handle the flattening of images from the cameras pixel buffer to utilize the increased accuracy that can be gained from using an Input/Output Keras model [8]. The user interface in FinFind could be modified to allow for the phone to change from portrait to landscape mode. This change would not affect the Core ML model's functionality, it would only require further development with Apple's UIKit API.

Another improvement that could be made is during the data collection stage, which is keeping the background as constant as possible. In other words, hand held video data of moving subjects in front of a highly detailed background is not optimal and was detrimental to the detection of the Wrasse fish. Additionally, it was difficult to capture adequate video and images of the Wrasse because it tended to stay around natural coverage. If the image database had been larger, the removal of the mixed feature would have been tested during training to see if the detection accuracy of the Wrasse improved, because the network struggled with knowing the difference between the Wrasse feature from the mixed feature.

Currently, developing machine learning applications for the iPhone has some disadvantages, the first being the model's physical size that it takes up on the SSD (Solid State Disk) memory on the device. With the introduction of TensorFlow lite, model sizes have shrunk dramatically in resource requirements making them more efficient. Although, Google does state that its TensorFlow lite supports iOS devices, Keras does not support TensorFlow lite on the backend at this time. This is an issue because Apple's coremltools require either a Caffe or Keras exported .h5 model. In the future it is likely that Apple will expand coremltools to support a wider array of model export types for developers.

The second disadvantage is that the machine learning industry as a whole, until TensorFlow lite, has not supported dropout layers for models exported to embedded systems because the nature of dropout layers is that they require random probability of nodes to be dropped out, something that conversion tools like coremltools have decided not to deal with for now. This is unfortunate because dropout layers prevent over fitting, which is an issue especially in multi-layer dense neural networks, like the one in FinFind. Over fitting can be defined when a model learns in such a way that it over memorizes to the point that an unknown or barely known input can produce a drastically worse hypothesis. Overfitting can be seen in the top subplot of figure 4 half way into the 5,000 epoch where the accuracy starts to steadily decrease until training ended at 6,000 epochs. It is important to note that the Input/Output model when tested during training with only two dropout layers in the third and sixth hidden layers using the architecture in figure 1, was able to produce confidence values by the end of 1000 epochs well above 90% accuracy.

Lastly, a serious difference in performance exists between a numpy feed Input/Output model and TFRecord feed Sequential model, which resulted in approximately a fivefold increase in performance when using the Sequential network. Even though all the training data for the Input/Output model was sitting in RAM, it was still slower than the Sequential model that queued data directly into VRAM (Video RAM) as a colored image.

Conclusion

The overall goal of the project is to be able to identify the features of aquatic life using machine learning algorithms, which was successful. Ultimately, the project required the combination of three computers, four operating systems, three computer languages, two imaging devices, and one high-end graphics card. Utilizing these pieces of hardware and software, allowed for the simple task of recording nearly thirty minutes of video to be utilized by machine learning algorithms in order to produce a model capable of classifying one pre-learned specie from another. Although, this project was aimed at being a proof of concept, the reach of artificial intelligence applied to the animal kingdom can offer countless benefits from its use. Hence, FindFind showed the capability of the model for the use in identifying aquatic species and that such a system can be deployed on an embedded device.

References

[1] Apple Inc., “coremltools.converters.keras.convert,” 2018. [Online]. Available: https://apple.github.io/coremltools/generated/coremltools.converters.keras.convert.html. Accessed: on April 19, 2018

[2] Apple Inc., “Vision,” 2018. [Online]. Available: https://developer.apple.com/documentation/vision. Accessed on: April 21, 2018.

[3] keras-team, “optimizers.py,” Feb. 20, 2018. [Online]. Available: https://github.com/keras-team/keras/blob/master/keras/optimizers.py. Accessed on: April 21, 2018.

[4] keras-team, “losses.py,” Jan. 8, 2018. [Online]. Available: https://github.com/keras-team/keras/blob/master/keras/losses.py. Accessed on: April 17, 2018.

[5] Lets Build That App, “CoreML: Real Time Camera Object Detection with Machine Learning -Swift 4,” July 14, 2017. [Online]. Available: https://www.youtube.com/watch?v=p6GA8ODlnX0&t=514s. Accessed on: April 17, 2018.

[6] E. Oliveros, “Learn Swift 3 Camera embedded in UIView AV Foundation Part 1,” Jan. 11, 2017. [Online]. Available: https://www.youtube.com/watch?v=yW-6bxk3j2g. Accessed on: April 18, 2018.

[7] Keras, “Optimizers,” 2018. [Online] Available: https://keras.io/optimizers/. Accessed on: April 22, 2018.

[8] Apple Inc., “Classifying Images with Vision and Core ML,” 2018. Available: https://developer.apple.com/documentation/vision/classifying_images_with_vision_and_core_ml. Accessed on: April 18, 2018.

[9] Apple Inc., “coremltools,” 2018. [Online]. Available: https://apple.github.io/coremltools/index.html. Accessed on: April 17, 2018.

[10] Apple Inc., “Integrating a Core ML Model into Your App,” 2018. [Online]. Available: https://developer.apple.com/documentation/coreml/integrating_a_core_ml_model_into_your_app. Accessed on: April 21, 2018.

[11] Keras, “Getting Started With the Keras Sequential Model,” 2018. [Online]. Available: https://keras.io/getting-started/sequential-model-guide/. Accessed on: April 23, 2018.