DATA-698 Capstone Project, Spring 2017

Mohamed Elmoudni

Contents

Abstract

The Nature Conservancy, an environmental non-profit, is working with several Pacific Island nations and a big tuna fishing company to more easily count and identify fish caught at sea using cutting edge technology. The goal is to use deep learning to help fishermen reduce the number of protected animals like species of sharks and tunas that are accidentally caught along with the tuna. The Nature Conservancy hopes that the program could prevent overfishing and help threatened and endangered sea life recover without putting fishermen out of work. The Nature Conservancy has installed nearly a dozen boats with electronic monitoring systems that include a set of cameras, sensors, and GPS devices. The system can record most of what takes place on board to prove that the operators did nothing illegal and to back up any compliance data that they must present to officials when they deliver their catch.

Background

Nearly half of the world depends on seafood for their main source of protein. In the Western and Central Pacific, where 60% of the world's tuna is caught, illegal, unreported, and unregulated fishing practices are threatening marine ecosystems, global seafood supplies and local livelihoods. The Nature Conservancy is working with local, regional and global partners to preserve this fishery for the future.

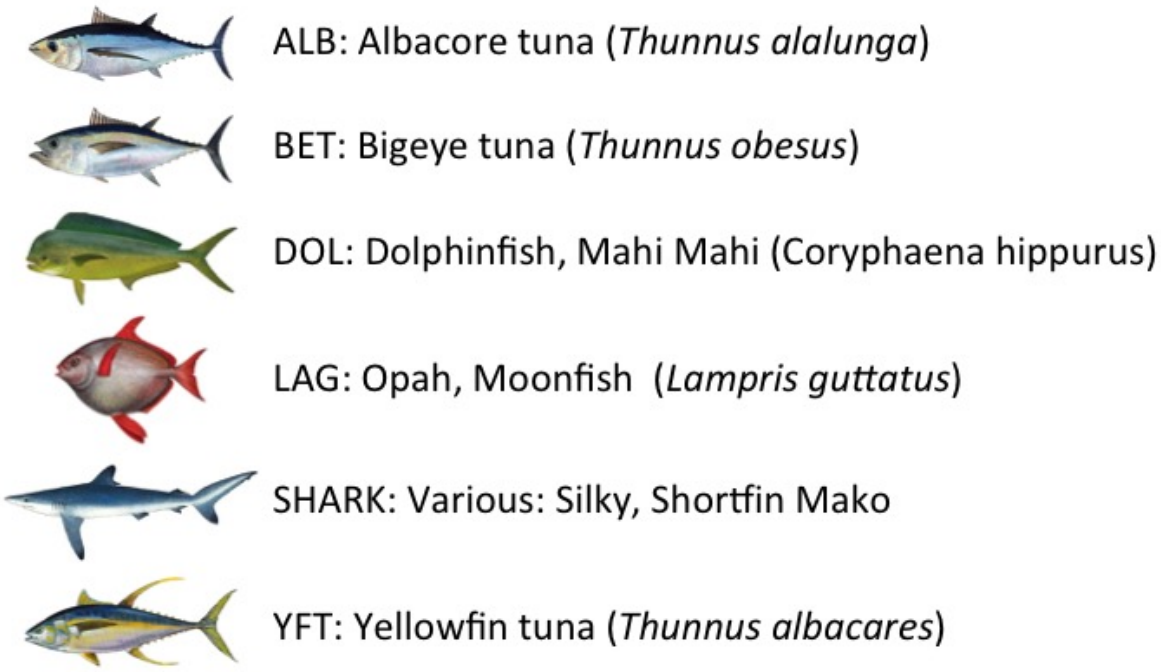
The Nature Conservancy pioneers new technology and data analytics solutions to collect baseline scientific data on fishery health, monitor fleet compliance with fishery regulations and deliver "bait to plate" traceability to seafood suppliers and consumers. In WCPO longline tuna fisheries, the Conservancy is deploying electronic monitoring systems on longline vessels to improve scientific data collection and reduce poaching.

Project Overview

In this project, we will be training a deep learning model to automatically detect and classify species of tunas, sharks and more that fishing boats catch, which will accelerate the recording review process. The approach is to have an algorithm embedded into a fully automated software tool that workers could use in their fishing operations. The algorithm will help fishermen detect which species of fish appears on a fishing boat, based on images captured from boat cameras of various angles.

Approach

In this project, we are going to train a machine learning model to predict the likelihood of fish species in each picture. Eight target categories are available in this dataset: Albacore tuna, Bigeye tuna, Yellowfin tuna, Mahi Mahi, Opah, Sharks, Other (meaning that there are fish present but not in the above categories), and No Fish (meaning that no fish is in the picture). Each image has only one fish category, except that there are sometimes very small fish in the pictures that are used as bait. We'll first create a synthetic dataset and use that to train our model. We'll then validate the model synthetic test data. We will create multiple models using grid search for epochs, batch size, and optimizer. Below is a sample of fish images (categories) we will be looking for in the fish boats:



Fish images are not to scale with one another

Data Acquisition

The dataset was compiled by The Nature Conservancy in partnership with Satlink, Archipelago Marine Research, the Pacific Community, the Solomon Islands Ministry of Fisheries and Marine Resources, the Australia Fisheries Management Authority, and the governments of New Caledonia and Palau.

The train and test datasets is downloaded locally from Kaggle website. Then, the data is uploaded into Docker container. The train data is then loaded into the /DATA/TRAIN/ folder. It will be stored accordingly:

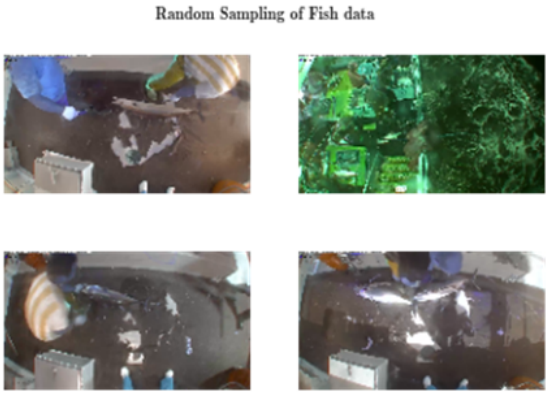
Albacore tuna: /DATA/TRAIN/ALB/
Bigeye tuna:/DATA/TRAIN/BET/
Mahi Mahi:/DATA/TRAIN/DOL/
Opah: /DATA/TRAIN/LAG/
Sharks: /DATA/TRAIN/SHARK/
Yellowfin tuna: /DATA/TRAIN/YFT/
Other fish than the above: /DATA/TRAIN/OTHER/
No fish in the picture: /DATA/TRAIN/Nof/

The test data is loaded in /DATA/TEST/

**** Please refer to the dataload.txt script in the documentation.

Data Exploration

Below is a sample pictures of the fish data that was caught in the boats. In addition, a fish photo count and type is also depicted below.



Fish Count and Type

Fish Count	Fish Type
1719	Albacore tuna
200	Bigeye tuna
117	Opah
67	Mahi Mahi
176	Sharks
734	Yellowfin tuna
299	Other fish than the above
465	No fish in the picture

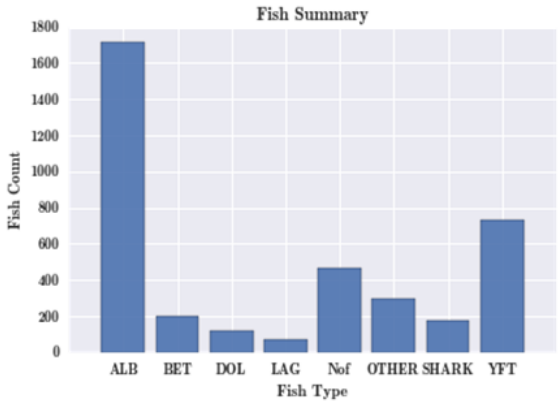


Figure 1:

Model creation

I will be creating three Convolution Neural Network (CNN) models with different topologies and different hyper parameters. In addition I will apply augmentation techniques on the best performing model.

A base model (model One) will have two convolution layers with the following major CNN components:
1- Convolutional layer with 60 feature maps of size 5x5.
2- Pooling layer taking the max over 2x2 patches.
3- Convolutional layer with 12 feature maps of size 3x3.
4- Pooling layer taking the max over 2x2 patches.
5- Dropout layer with a probability of 50%.
6- Flatten layer.
7- Fully connected layer with 128 neurons and rectifier activation.
8- Dropout layer with a probability of 50%.
9- Fully connected layer with 50 neurons and rectifier activation.
10- Dropout layer with a probability of 50%.
11- Output layer.

A second CNN model called model Two , with three convolution layers with the following major CNN components:
1- Convolutional layer with 16 feature maps of size 5x5.
2- Pooling layer taking the max over 2x2 patches.
3- Convolutional layer with 32 feature maps of size 3x3.
4- Pooling layer taking the max over 2x2 patches.
5- Convolutional layer with 64 feature maps of size 3x3.
6- Pooling layer taking the max over 2x2 patches.
7- Dropout layer with a probability of 50%.
8- Flatten layer.
9- Fully connected layer with 64 neurons and rectifier activation.
10- Dropout layer with a probability of 50%.
11- Fully connected layer with 64 neurons and rectifier activation.
12- Dropout layer with a probability of 50%.
13- Output layer.

A third CNN model called model Three, with three convolution layers with the following major CNN components:
1- Convolutional layer with 32 feature maps of size 5x5.
2- Pooling layer taking the max over 2x2 patches.
3- Convolutional layer with 64 feature maps of size 3x3.
4- Pooling layer taking the max over 2x2 patches.
5- Convolutional layer with 128 feature maps of size 3x3.
6- Pooling layer taking the max over 2x2 patches.
7- Dropout layer with a probability of 50%.
8- Flatten layer.
9- Fully connected layer with 128 neurons and rectifier activation.
10- Dropout layer with a probability of 50%.
11- Fully connected layer with 128 neurons and rectifier activation.
12- Dropout layer with a probability of 50%.
13- Output layer.

Model Selection

a. Model comparaison and tuning strategy

To improve the algorithm the best performing model, I will be tuning the below areas by Grid-Searching the hyper parameters when applicable:

1- Network Topology: I will be changing the network topology by using different number of layers and neurons.

2- Learning Rate: I will experiment with very small learning rates and large rates. I will try adding momentum term; then change the learning rate.

3- Activation Functions: I will be experimenting using the different activation function sigmoid,tanh, relu, then a softmax, linear or sigmoid on the output layer.

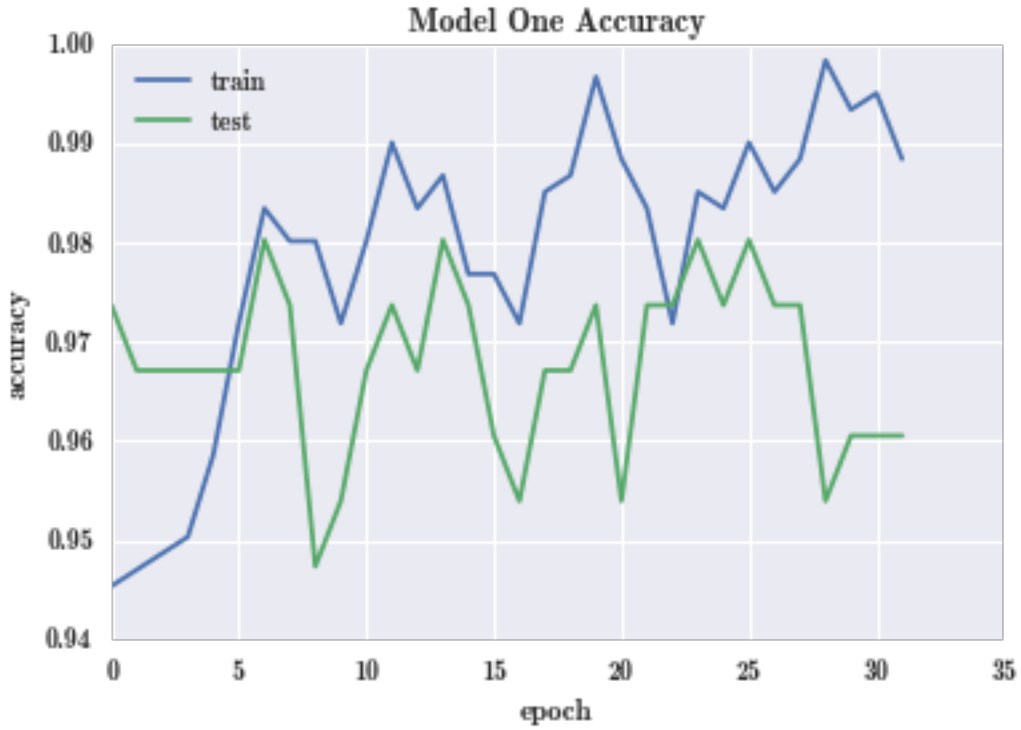
4- Batches and Epochs: The batch size defines the gradient and how often to update weights. An epoch is the entire training data exposed to the network, batch-by-batch. I will be experimenting running small batch sizes with large epoch size.

5- Regularization: regularization is a great approach to curb overfitting the training data. I will be using the dropout regularization technique. Dropout randomly skips neurons during training, forcing others in the layer to pick up the slack. I will be experimenting with dropout in the input, hidden, and output layers.

6- Optimization and Loss: There are many optimization methods that offer more parameters, more complexity and faster convergence. However, for our classification problem, we will be mainly experimenting with the followings: adam and RMSprop

Models Results

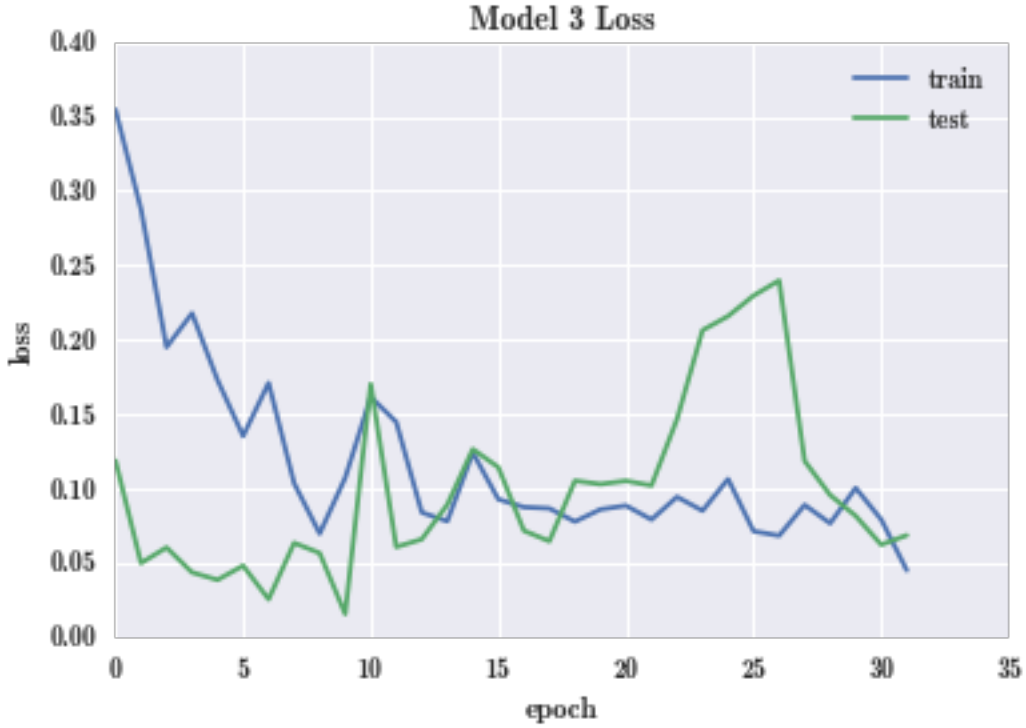
Below are the accuracy and loss for all three models. Each model has the accuracy and loss based 32 epochs.




Model Two Accuracy
train
test
accuracy
epoch
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0
5
10
15
20
25
30
35


Model Two Loss
train
test
loss
epoch
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20
25
30
35


Model 3 Accuracy
accuracy
epoch
train
test
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0
5
10
15
20
25
30
35


Model 3 Loss
loss
epoch
train
test
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
30
35

Model Comparaison

From the below model accuracy performance figure, we clearly see that model Three has outperformed the other two models. In addition it has lower loss value of 3.17% unlike model One and Two which have an error loss of 3.31% and 8.99% respectively. Model Three strength was based on robust topology of three layers and maximum nodes of 128 nodes , optimizer function RMS prop that utilizes the magnitude of recent gradients to normalize the gradients. As it always keep a moving average over the root mean squared (hence Rms) gradients, by whichit divides the current gradient, and learning rate of 0.0001.

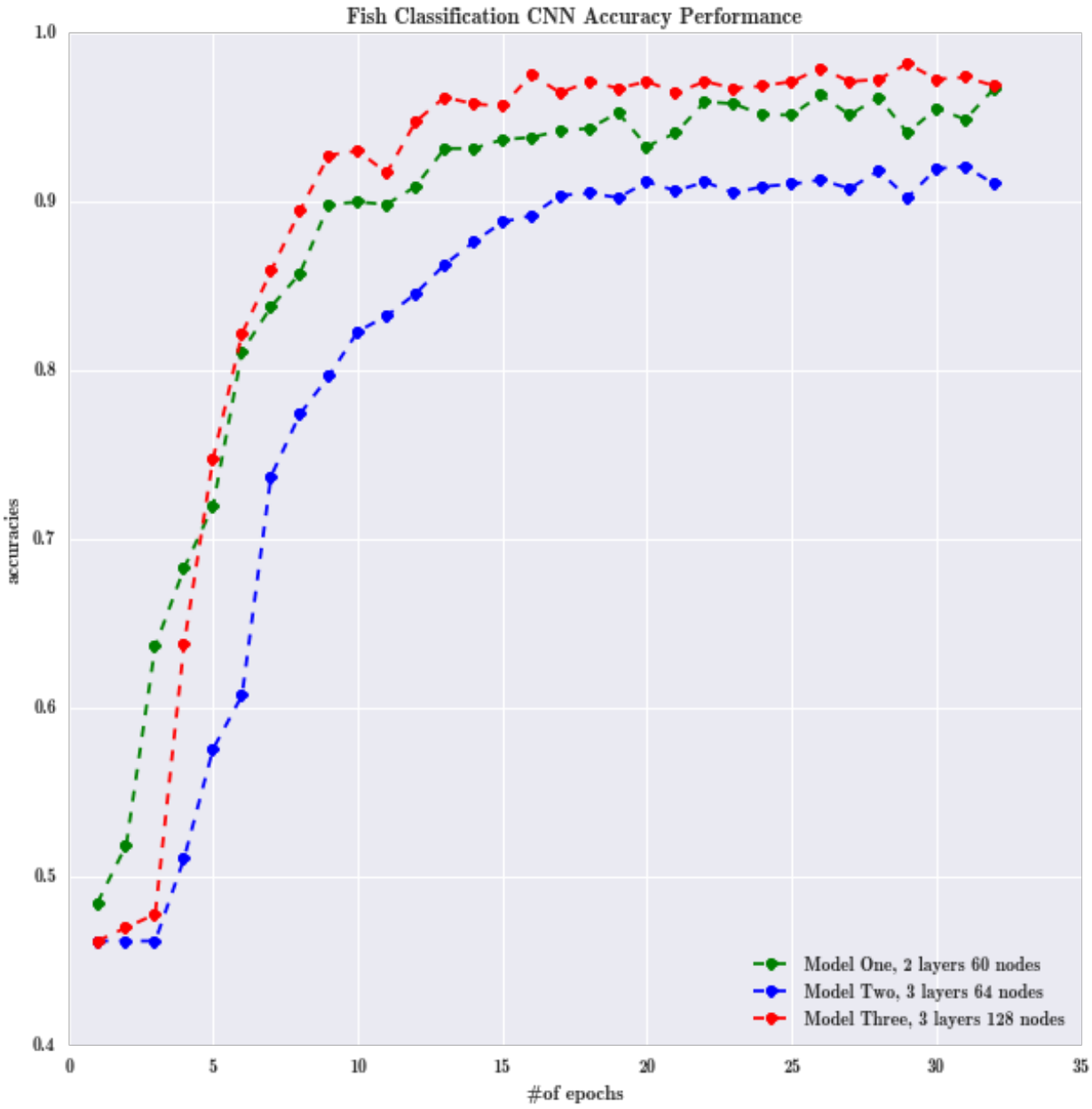


Figure 2: Model Accuracy Performance

Prediction

a. `Prediction on test/unseen data`

We can see that our model did a pretty good job at predicting data and it did so with a training set of just over 3777. If we continue to train with the larger set of 1,000,000 images, we will achieve even better results. The accuracy was above 96% with some exceptions in lower 80%..However, I noticed that our model did not perform well on images that were taken at night. The accuracy on dark image test data was below 50%, around 45%.

Table 1: Prediction on test data

image	ALB	BET	DOL	LAG	Nof	OTHER	SHARK	YFT
img_06959.jpg	0.963790	6.00e-07	0.00e+00	0.0e+00	1.16e-03	9.00e-07	0e+00	0.000000
img_00582.jpg	0.979436	4.00e-07	1.00e-07	7.0e-07	4.78e-05	1.03e-05	0e+00	0.000003
img_03760.jpg	0.928267	5.30e-06	6.00e-07	1.8e-06	2.45e-04	7.11e-04	1e-07	0.001176
img_05721.jpg	0.000000	0.00e+00	2.00e-07	0.0e+00	0.00e+00	0.00e+00	0e+00	0.990492
img_04156.jpg	0.414800	2.19e-05	1.00e-07	7.0e-07	1.55e-05	4.65e-03	1e-07	0.000538
img_06373.jpg	0.000001	0.00e+00	2.00e-07	0.0e+00	0.00e+00	0.00e+00	0e+00	0.993941
img_04024.jpg	0.827182	0.00e+00	0.00e+00	0.0e+00	0.00e+00	9.00e-07	0e+00	0.001773
img_04037.jpg	0.960498	9.00e-07	0.00e+00	2.0e-07	3.13e-05	4.59e-04	0e+00	0.000015
img_01773.jpg	0.039052	3.00e-07	0.00e+00	0.0e+00	1.00e-07	1.40e-06	0e+00	0.460410
img_05908.jpg	0.007481	6.57e-05	4.85e-05	2.1e-06	4.50e-01	7.20e-06	7e-06	0.018059

Conlusion

The project was challenging at the same time very interesting as it is Deep learning project. The use of deep learning Docker image has helped tremendously especially in packaging all the modules needed for the program compilation.

To improve the model, I focused in tuning the data and the algorithm. For the data, I used augmentation specifically normalization. I was not able to try other augmentation tasks such as the ZCA whiting and other image manipulations due to resources constraints (memory).
As for algorithms tuning, I was able to improve the model performance by using different parameters values for activation functions, learning rates, and network topology.
However, I think given adequate system resources memory and cpu/gpu's , I would have tried trained more data, used more epochs, data augmentation, and certainly deeper network topology.