Fish Classification

Classifying fish species using Amazons Sagemaker’s Image Classification Algorithm

Matthew Burt

Udacity Machine Learning Engineer nanodegree

22/06/2020

**1 Overview**

**1.1 Project Domain**

Image classification, in the computer vision field, is the process of classifying images into different categories based on the images content. It is a popular field that has seen massive improvements over the past few years, especially with the application of convolutional neural network (CNN) algorithms to the task. My project involves using computer vision algorithms to correctly classify pictures of fish into the correct species category. This will give users real time information of the size and bag limits of the fish they have caught.

**1.2 Project Goal**

The problem I am trying to solve is one of image classification, namely classifying images of fish into the correct species. There is currently a website that serves the purpose of giving anglers information on size and catch limits within Victorian waterways, but it requires users to either know roughly the species they have caught or click into a lot of links to get the information.

I have used SageMakers image classification algorithm, a deep convolutional neural network (ResNet) model, trained on the SageMaker platform, to classify images of fish into the correct species in order to return this information. This model was created, trained and tested on AWS Sagemaker. Once complete this was exported into ONNX format for use in my web app which I have deployed on Netlify <https://victoria-fish-species.netlify.app/>. This web application takes a user supplied image and returns a classification and a link to the correct fish on Victoria Fisheries website.

**1.3 Performance Metrics**

I used accuracy as my models performance metric, defined as:

*Accuracy = Total Correct Predictions / Total Predictions*

Accuracy is the metric that needs to be optimised. The final model will be identifying and returning one of a certain number of classes. Because the user is most concerned about receiving information about the correct fish, it is most appropriate that the model is accurately returning the correct class.

**1.4 Benchmark**

For a Benchmark Model I trained a Convolutional Neural Network in PyTorch from scratch. This was done in a Jupyter notebook instance on Sagemaker and the accuracy of this model was used for a benchmark accuracy.

Details of the benchmark model structure can be found in the appendices of this proposal.

**1.5 Dataset**

This is a supervised learning problem so it required data labelled with the corresponding classes. There are some freely available datasets of classified fish images, but none that are specific to the state of Victoria’s fish species. As part of my project I created a labelled dataset of images to represent the 10 individual fish controlled by the Victorian Fisheries Authority.

My data was scraped from Google images and is in various sizes and contexts. Because of the types of images people take of fish the images generally fall into three categories, fisherman posing with fish, fish out of water or fish swimming in the ocean.

**2 Data Exploration**

**2.1 Fish Image Dataset Overview**

The dataset is a folder of images each split into sub-folders which are named for the species of fish within each sub-folder. The images are colour images. The distribution of images are relatively balanced for each class ranging from 87 to 296 with an average of 200.5 images per class. Australian Herring were the least represented species with only 87 images and Albacore Tuna the most with 296 images.

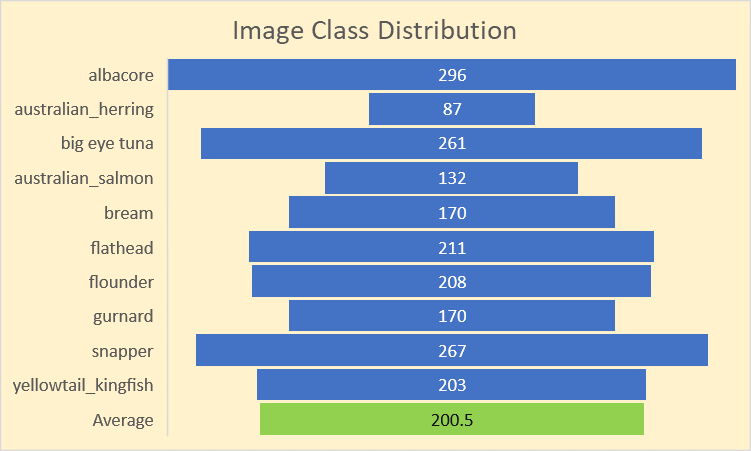


Figure Image dataset feature class distribution

Loading the images into an Amazon Sagemaker Jupyter notebook instance, I can then show a couple of example images as well as a list of class labels.

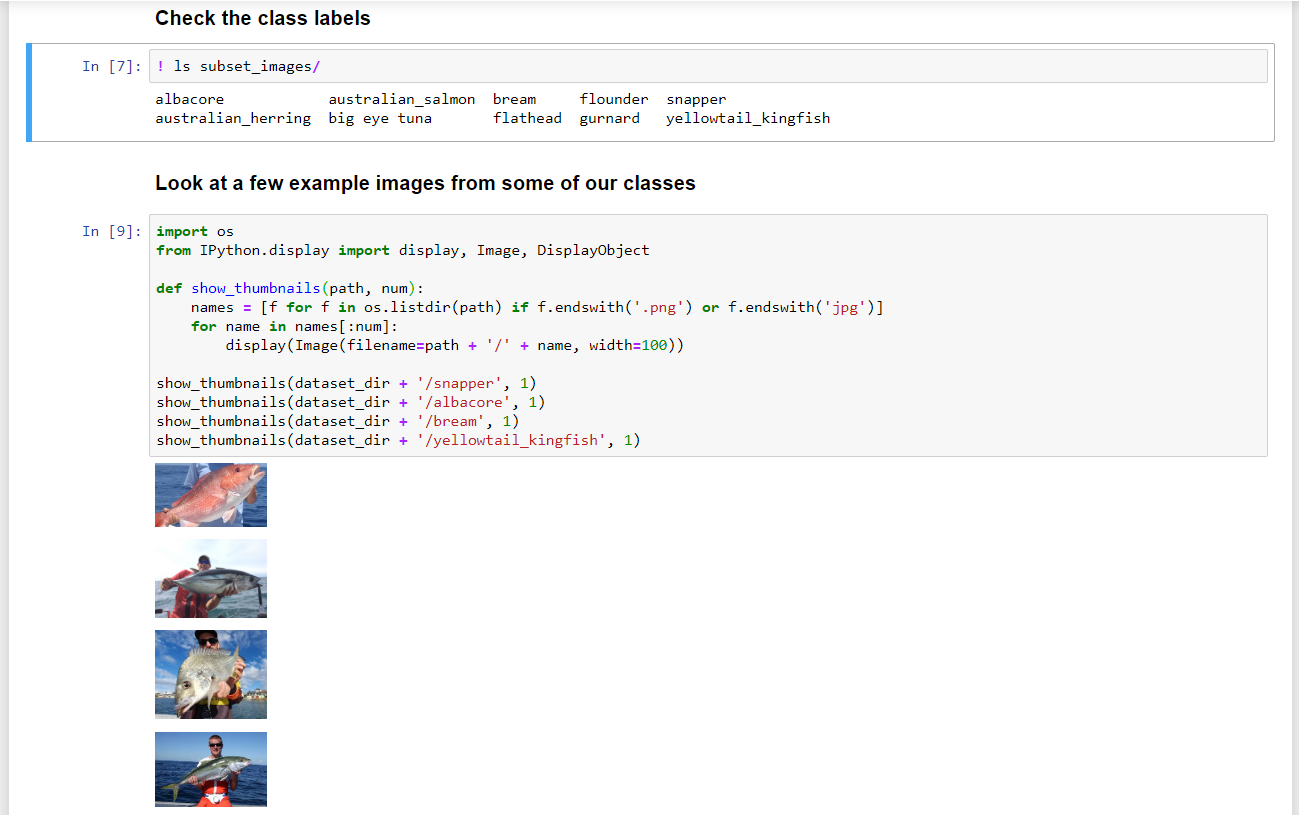


Figure Image dataset exploration

**2.2 Potential Issues**

Several issues have been identified that will need to be rectified.

**2.2.1 Dataset Size**

The dataset I have created for this project is relatively small consisting of a total of 2005 images split across 10 classes. This is not enough to ensure a good model can be trained, especially with the close visual similarity of some of the fish classes. To solve this issue, I will make use of the Sagemaker Image Classification Algorithm which enables me to use a ResNet network to train a model using transfer learning. Transfer learning utilises a pretrained neural network model trained on a large image dataset, ImageNet in this case. This allows the model to utilize features already learned in trainingthis ResNet and use these features to train only the top layers of the model to recognise any type of image classes the user wants to.

**2.2.2 Image Similarity**

Another issue was the visual similarity between separate fish classes. In particular tuna classes show very limited visual differences at all. This was not too much of an issue in this limited implementation (as there were only two tuna species included in the ten fish species I have used. But

Figure 3 Big Eye Tuna Figure 4 Albacore Tuna

**2.2.3 Image Context**

Fish images scraped from google are relatively similar contextually. They fit one of three broad contextual scenarios; fish side on lying on a flat surface (sand, boat deck, chopping board etc), fish side on held by an angler or fish swimming in the ocean. Given that this model will likely be used in one of either of the first two scenarios, I have removed images that were dissimilar to these, leaving only clear side angled shots of fish in the water.

**3 Methodology**

**3.1 Benchmark Model**

My benchmark model was a convolutional neural network trained from scratch using PyTorch on a Sagemaker Jupyter notebook instance. This algorithm has been proven to be effective in image classification tasks and should form a good baseline to compare my Sagemaker trained algorithm using transfer learning against.

**3.1.1 Data Pre-processing**

Using PyTorch, the pre-processing step for training my benchmark model was straightforward. I made use of the PyTorch ImageFolder and DataLoader functions to feed my training images into the model for training. This step also defined my batch size of 32 and image resizing dimensions of 224 x 224 pixels (to match the Sagemaker model).



Figure Data pre-processing in Jupyter notebook

**3.1.2 Neural Network Architecture**

The Convolutional Neural Network architecture I used is shown below. Originally defined in the project proposal, it has four convolutional layers followed by a fully connected linear layer. I used Cross Entropy Loss for the criterion, due to the problem being a multi class classification problem and used SGD for my optimizer.

Changes to the original model architecture detailed in my project proposal are as follows:

* Addition of a Dropout layer to reduced overfitting.
* Removal of BatchNorm2d layers two, three and four due to gradient explosion.



Figure Convolutional neural network architecture

**3.1.3 Benchmark Results**

I have decided to train the benchmark neural network model for ten epochs at a learning rate of 0.001 and a further five epochs at a reduced learning rate of 0.0001. This seems to be my optimal result, as further training of the model lead to a rapid increase of the loss. The benchmark using this neural network was an accuracy of 65% on the test images. This is the accuracy I will compare my final model to.

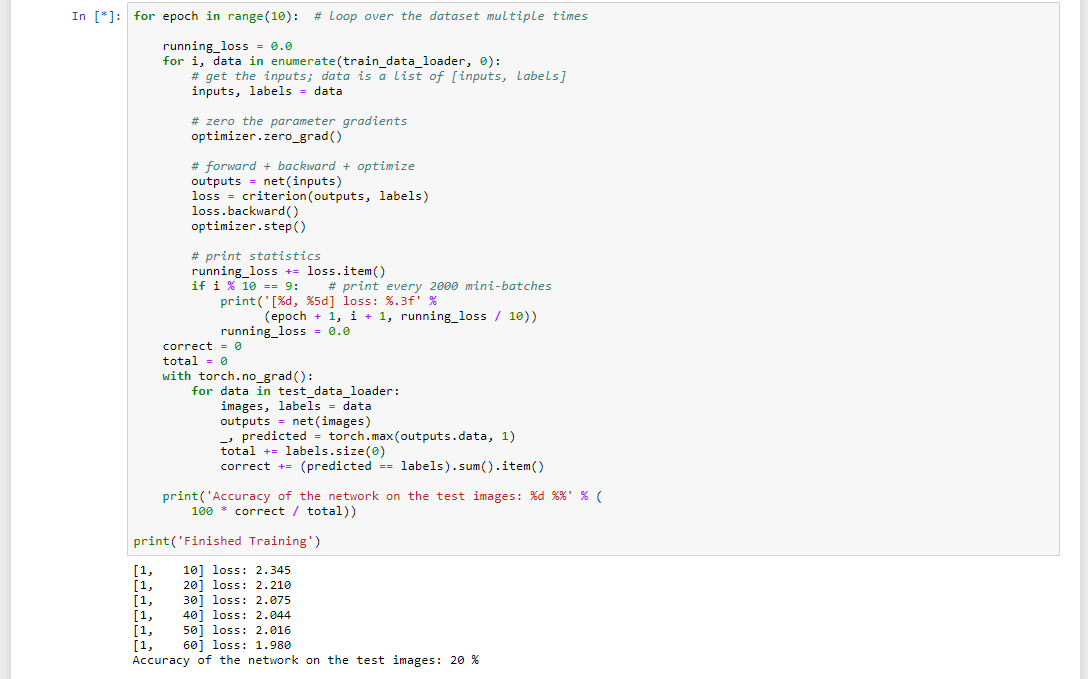


Figure Training loop in progress

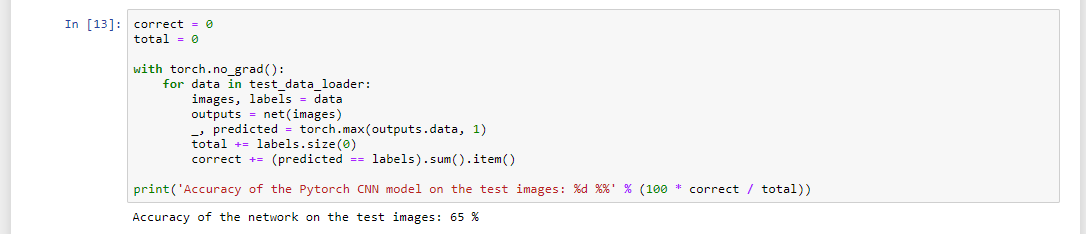


Figure Benchmark model performance on test dataset

**3.1.4 Benchmark Model Issues**

Using four batchNorm2d layers seemed to make my models loss exponentially worse during training. To resolve this, I removed BatchNorm2d layers two through to four. This left just one BatchNorm2d layer which seemed to improve the training process.

To reduce overfitting I added a dropout layer after the first convolution set to 0.2.

**3.2 Sagemaker Algorithm**

**3.2.1 Data Pre-processing**

Data pre-processing in Sagemaker is rather difficult from scratch, however with an example notebook I was able to navigate this step relatively easily.

The first step was to format my training images for use in training. The images scraped from Google vary in size and format. This needed to be addressed as the images needed to be in a standard format.

The following utility function was used to resize the images to 224 x 224 pixels and fill in any blank pixels for training the model.

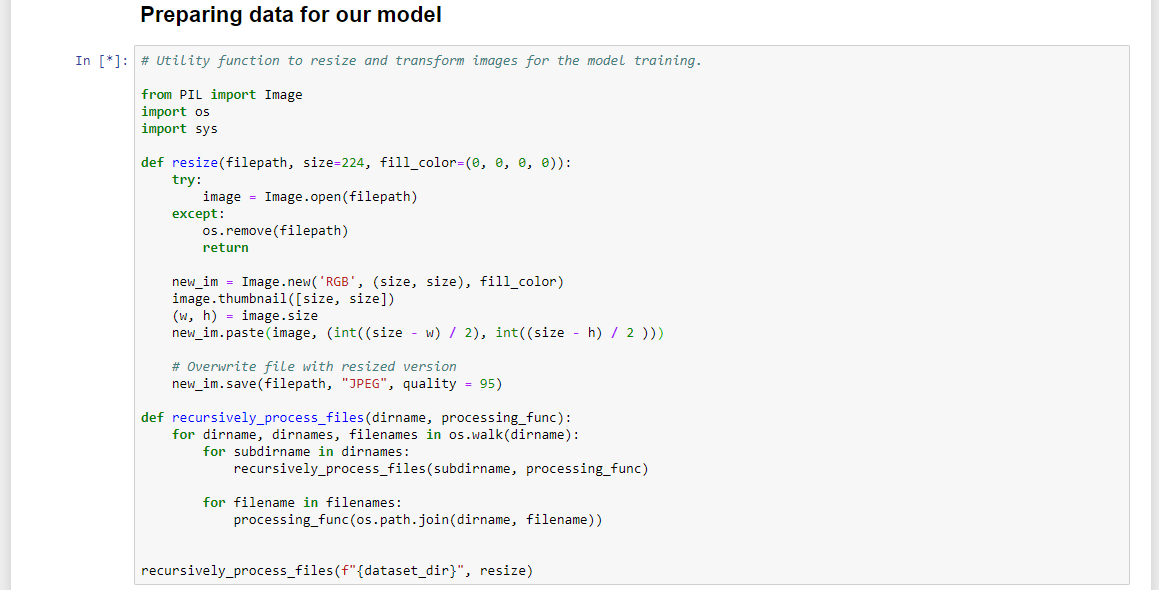


Figure Pre-processing function in Sagemaker

**3.2.1 Training Setup**

The training setup involves using a lot of Sagemaker variables, these are easily sourced online using worked examples from either Sagemaker documentation or other peoples publicly available work. A sample is below but more details can be found in the project Jupyter notebook.



Figure Sagemaker variable definition

To train an image classification model in Sagemaker, first I needed to set the models hyperparameters. Below is an example of how to do this. This is also the step which allows for the greatest amount of experimentation as you are able to tune the hyperparameters here to try to improve model performance.

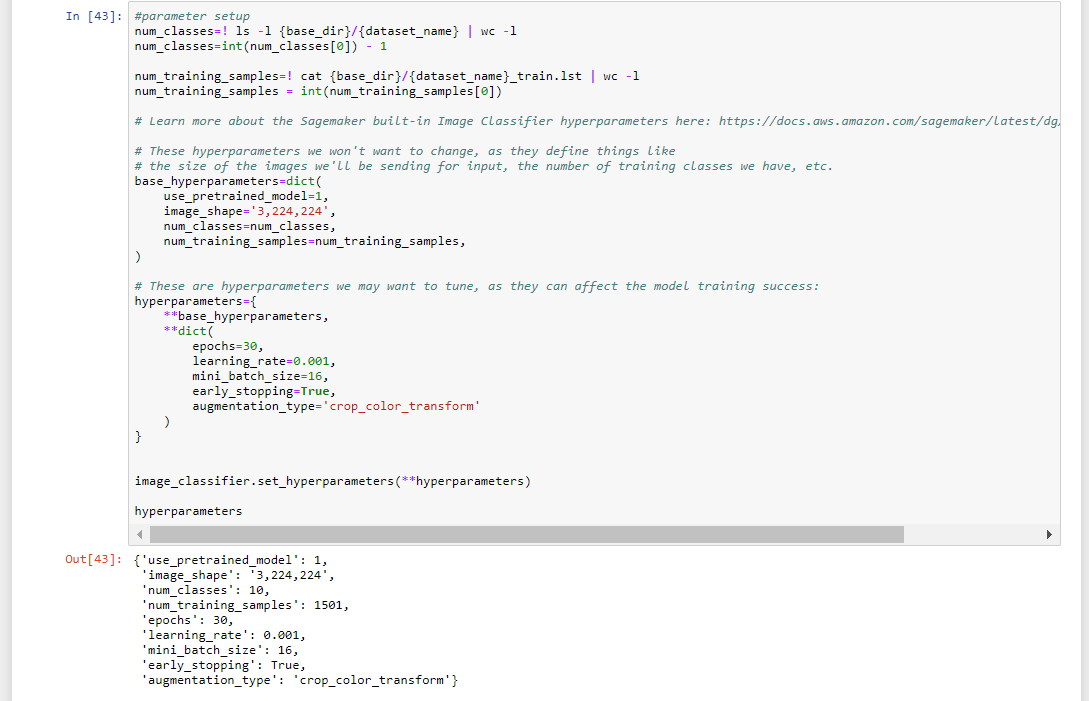


Figure Hyperparameter setup Sagemaker image classifier

Once the variables and hyperparameters were set, the following code starts a training job on Sagemaker.

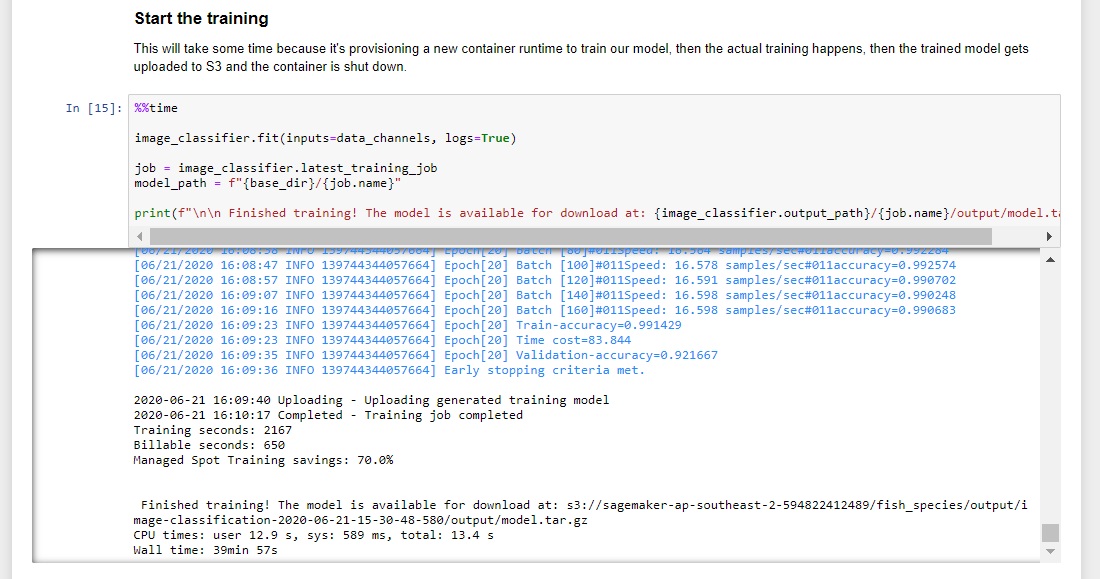


Figure Training a model in Sagemaker

**3.2.2 Model Refinement**

As part of the project, multiple model hyperparameter setups were tried in an attempt to refine the model and improve model performance on the test dataset. The performance of each of these setups is reported below.



Figure Model finetuning statistics

**3.2.3 Model Deployment and Testing**

To test the trained models accuracy, the model was deployed on Sagemaker and scored using its predictions for the test dataset. This is shown in the figures below.

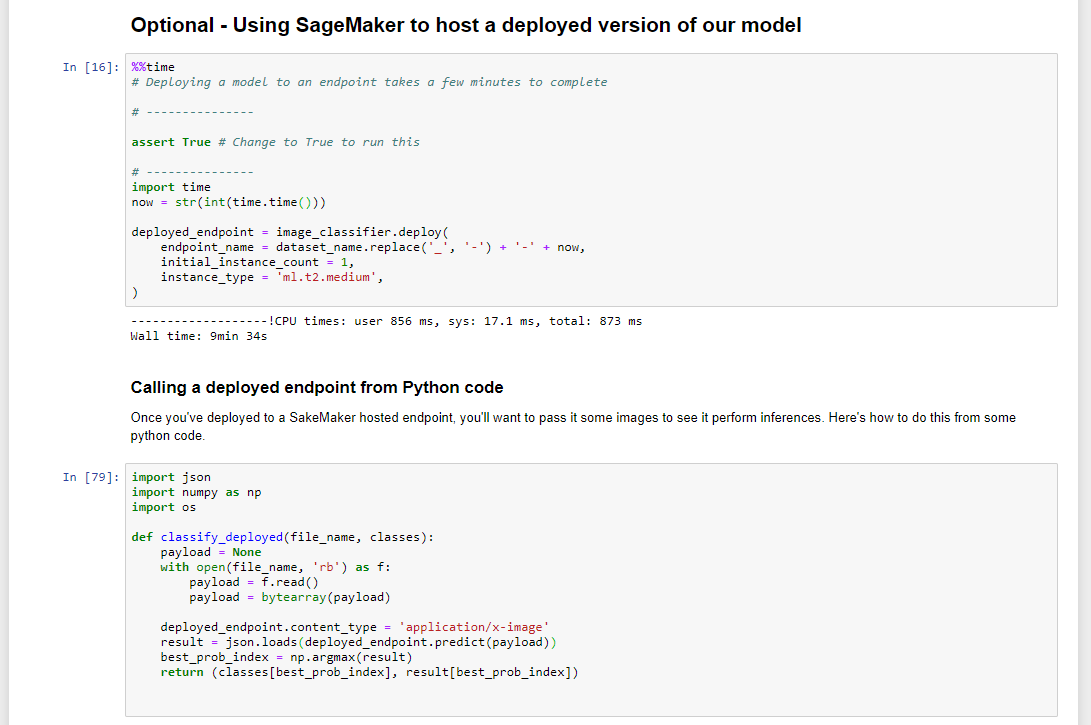


Figure Deploying a Sagemaker model and creating a prediction function to call the deployed mode



Figure 15 Testing model performance on a hold out test image dataset

**3.2.4 Final Model**

Based on my hyperparameter tuning, the my baseline Sagemaker image classifier was the best performing model with an accuracy of 94.59% on the test dataset. This accuracy was unchanged when data augmentation was used as well as when lr\_scheduler\_step was used.

The model trained with an Adam optimizer performed poorly on the test dataset with an accuracy of 51.35%, worse than my benchmark PyTorch model. Changes to my learning rate were unable to improve on the baseline model.

The model I will use in my application is the baseline model. This model has a satisfactory accuracy when tested on the test dataset. In my application, images input by users will likely be of high quality, and a prompt can be added to ensure users try to provide the best possible input image. Due to this, an accuracy of 94.59% is appropriate.

My final model has the following parameters:



Figure Final model parameters

**3.2.5 ONNX Conversion and Download**

The following code shows how I have converted the Sagemaker Mxnet model to ONNX format for download and use in my application.



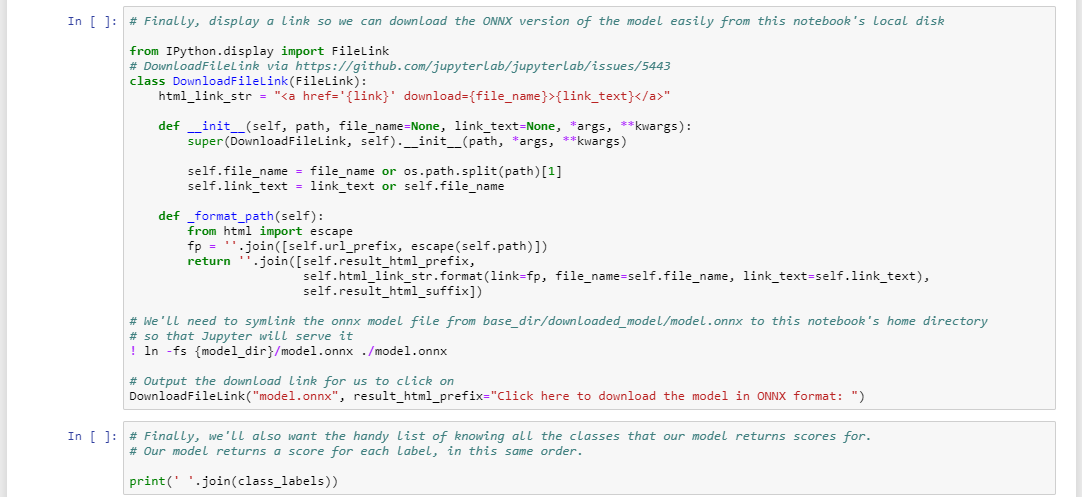


Figure MxNet model conversion into ONNX format

**4 Application**

**4.1 Application Overview**

The primary deliverable of this project is a completed web application that uses the trained model to perform inference on a user supplied image and returns information on the fish species captured.

I have decided to create an ONNX version of my trained Sagemaker model to use for this task rather than using a deployed Sagemaker endpoint to keep costs low. This does mean that the application is data intensive, requiring the user to download a 200mb model into memory, however at the initial stages of development this is acceptable as a proof of concept.

This model will be saved to Dropbox and loaded into memory each time a user visits the website.

Once the model is loaded into memory, users can input an image from a file, or using their mobile phones camera. The model then runs inference on the image, returning the predicted class and a link to the Victoria Fisheries Associations bag and size limits. A diagram of this process is supplied below as well as some screenshots of the web application.

The web application has been deployed at the following link <https://victoria-fish-species.netlify.app/>.

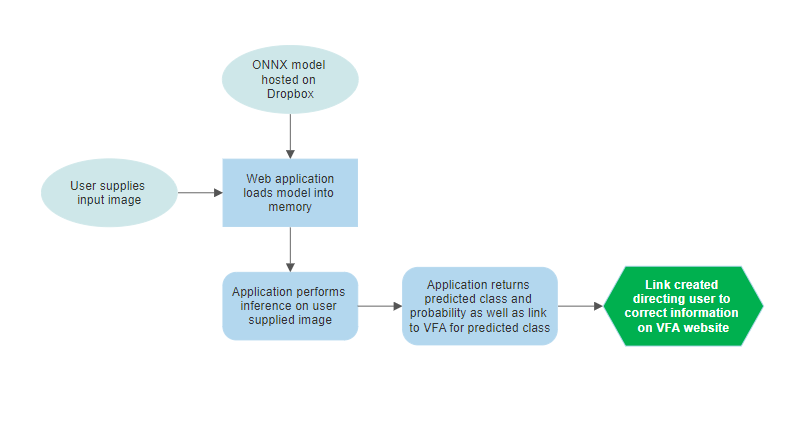


Figure 18 Web application process diagram

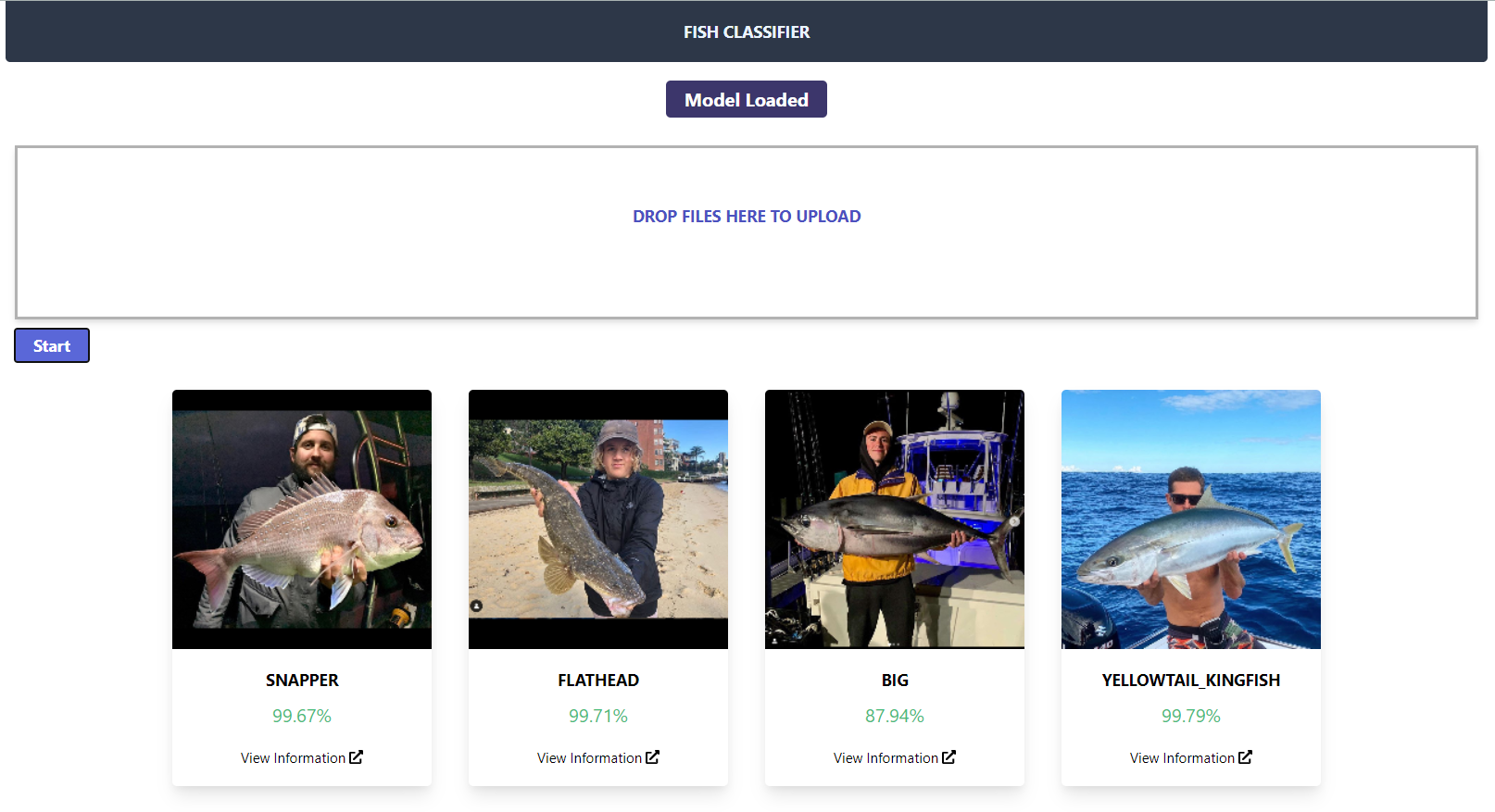


Figure 19 User input images classified using the web application

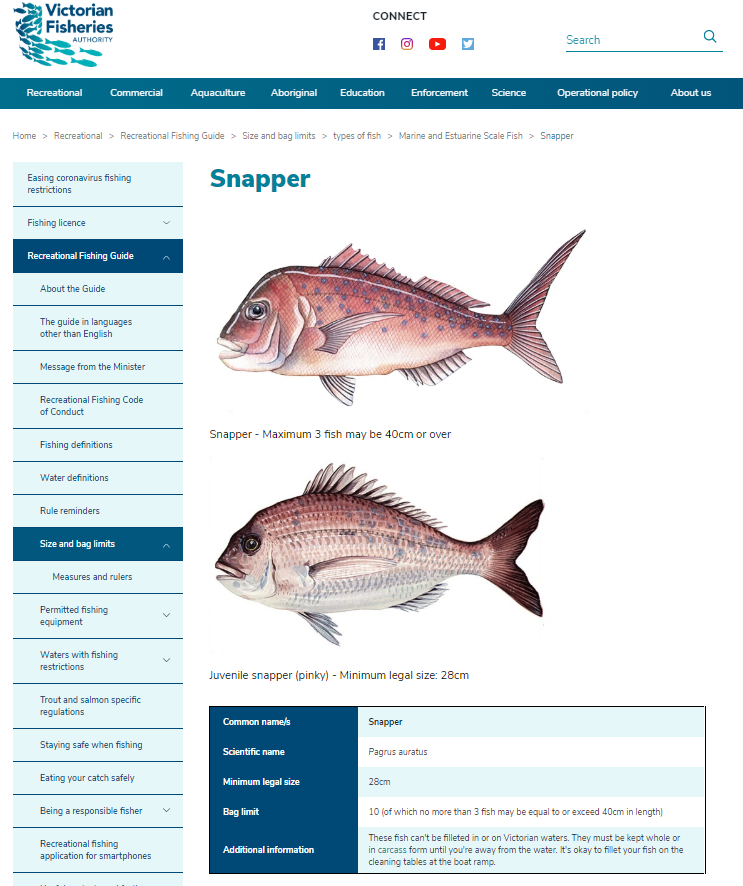


Figure 20 Sample link path to VFA information on a Snapper fish

**References**

The following resources were amazing in helping me deliver this end to end machine learning project.

<https://github.com/gabehollombe-aws/sagemaker-image-classifier-to-onnx-in-browser>,

ONNX deployment of image classifier on a web app

<https://docs.aws.amazon.com/sagemaker/latest/dg/image-classification.html>

AWS Image Classification resources.

<https://arxiv.org/abs/1512.03385> Kaiming He, et al., 2016

IEEE Conference on Computer Vision and Pattern Recognition

<https://vfa.vic.gov.au/recreational-fishing/recreational-fishing-guide/catch-limits-and-closed-seasons/types-of-fish/marine-and-estuarine-scale-fish>

VFA fish species data

<https://github.com/MorvanZhou/PyTorch-Tutorial>

Resource for PyTorch baseline model.