Capstone Project

Viswanath Ravindran

May 15th, 2018

I. Definition

Project Overview

Blue Whale tracking has been considered an exciting and dear problem for Happywhale as in link. They have quoted specifically tracking of the Humpback whale as a laborious process by scanning through several thousands of pictures posted online by whale enthusiasts/scientists/tourists/fisherman throughout the world.

It simply works by submitting photos of your marine mammal encounters and they would verify the whale species by looking for the unique marking on its fluke(whale's tail). By doing so HappyWhale tracks the movement of whales across the world.

Problem Statement

Identification of Whale population has been a challenge for a very long time. As mentioned above, Happy Whale organization gather pictures of whale encounters. The pictures that is provided for this problem specifically includes the whale fluke. The shape of the fluke with its markings are unique to the individual whales. Happy whale now would be manually checking the thousands of photographs against their existing database to id the whale in the new photographs received.

The problem on hand here is to apply computer vision and help Happy Whale organization to be able to classify the images into their respective whale ID. This is large multi class classification problem where we are treating the individual pictures to identify their class.

Metrics

The evaluation metric for this challenge is called the Mean Average Precision score at K. The k selected here is 5, The output prediction should output the 5 most possible class labels among the several available class. The choice of metric because it is defined in the kaggle problem.

Its use is different in the field of Information Retrieval and popular among various Multi Label Classification problems setting. The mathematical representation is available as in kaggle link

$$MAP@5 = \frac{1}{U}\sum_{u=1}^{U}\sum_{k=1}^{min(n,5)} P(k)$$

where `U` is the number of images, `P(k)` is the precision at cutoff `k`, and `n` is the number predictions per image.

II. Analysis

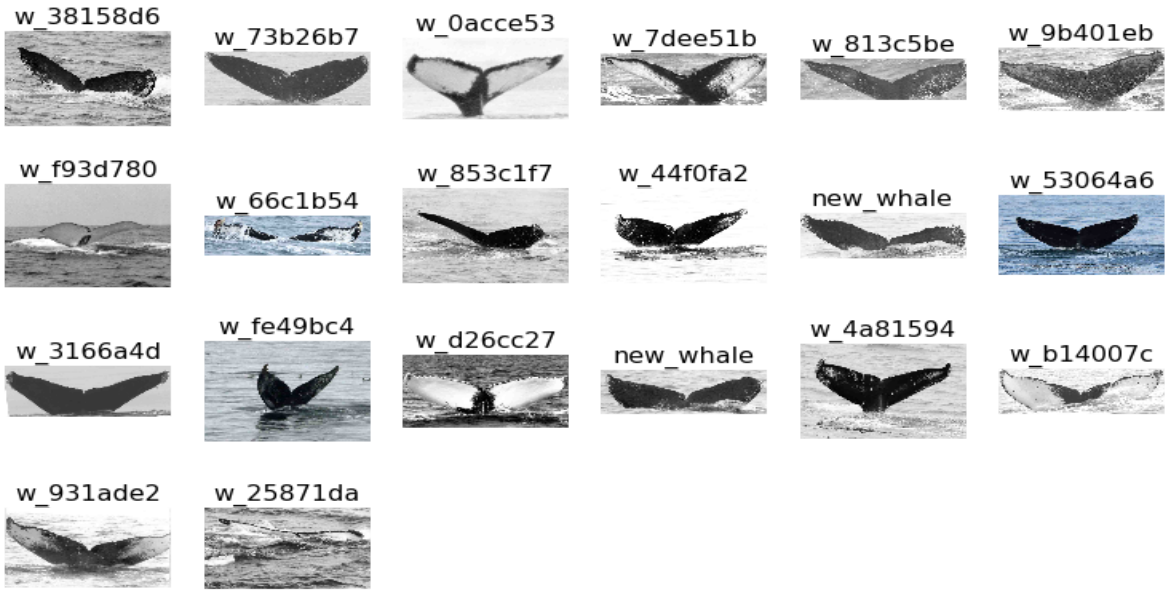
Data Exploration

The Happy Whale dataset that has been provided to us has both the train and test data set. We can have a look at the dataset from the following link

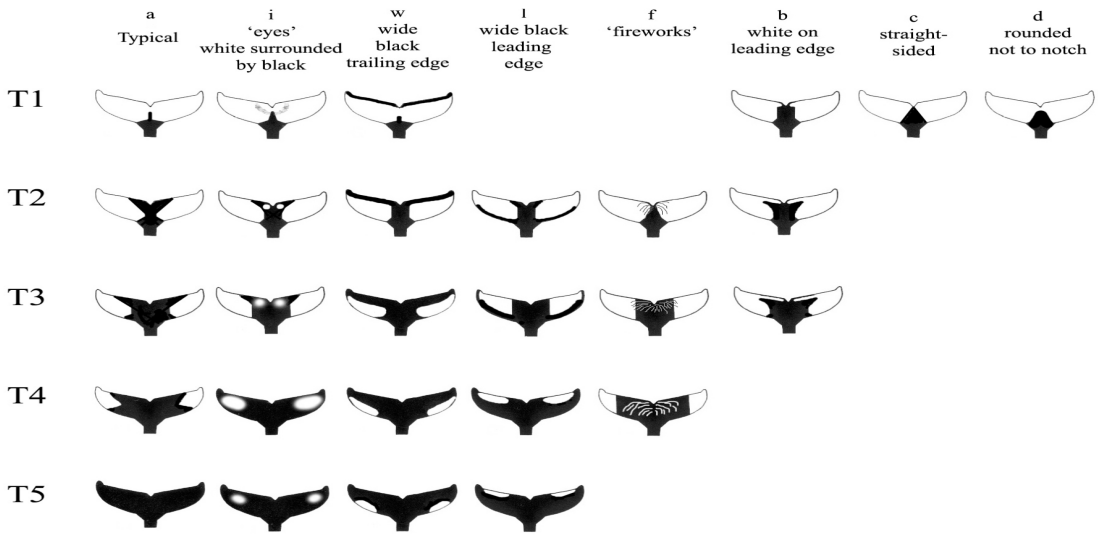
The training dataset has about 9850 images after constructing a dataframe referring individual image to its ID it looks like below:

Image	Id
../input/train/00022e1a.jpg	w_e15442c
../input/train/000466c4.jpg	w_1287fbc
../input/train/00087b01.jpg	w_da2efe0
../input/train/001296d5.jpg	w_19e5482
../input/train/0014cfdf.jpg	w_f22f3e

The test data which is the non-labelled images are about 15610 images. Some sample images from the train data with their labels are as below:



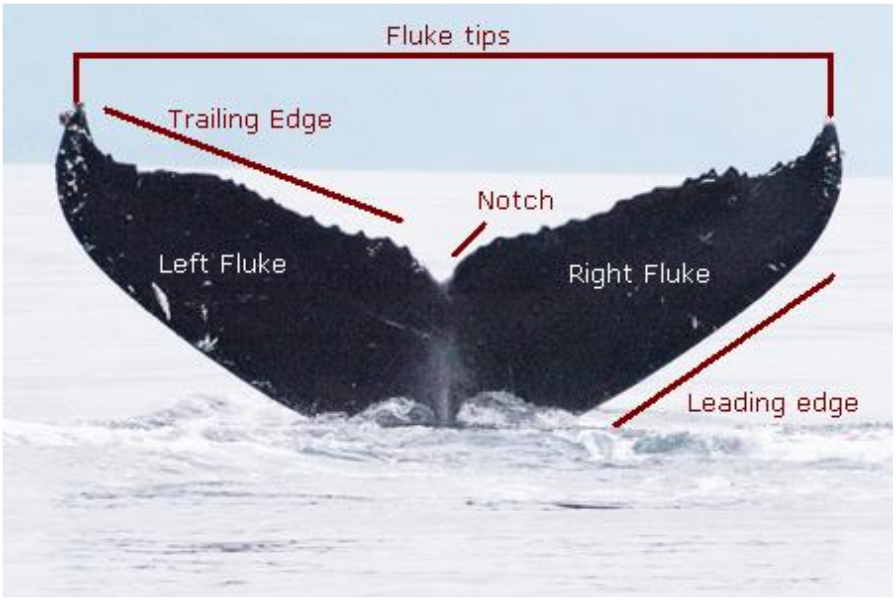
Another reference article online indicates the following information from link. There are five basic types of whale flukes: 1. Mostly white. 2. 25% white. 3. Half white, half black. 4. 25% black. 5. Mostly black. With some luck, one might only have to look into one type- this would usually only occur in mostly white or black.



Often one can eliminate three types, and look through only two types. Sometimes one has to look through three types, especially if not 100% of the fluke can be seen.

Another article online indicates the following properties above whale fluke at link

- The whales fluke has the following parts indicating its anatomy.



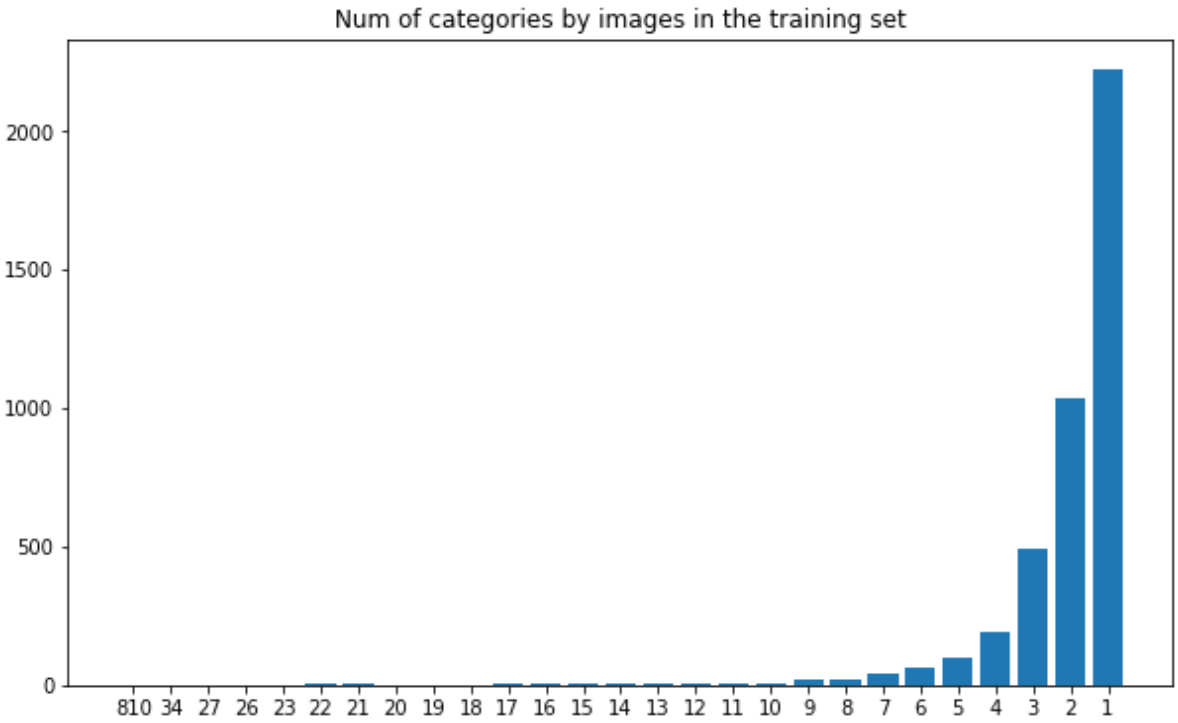
- The shape of the flukes can change to some degree depending on the angle to the surface of the water the instant the photograph was taken
- If your photo has flukes with a large area of white you may want to focus on a specific scratch or shape in the white area. If your photo has dark flukes with some white scratches, focus on finding a photo with similar scratches.
- One of the most important areas of the flukes to look at is the trailing edge. The bumps and nicks along this edge remain relatively stable throughout the whale's lifetime.
- Since the photograph of the whale are taken over a period of time there are possibilities that the fluke shape may be changed, the scratch pattern may be different now etc..
- Matching a young whale is significantly more difficult since the pigmentation on the flukes becomes more distinct as the whale gets older and then the pattern seldom changes.

To summarize the above findings the quality of picture plays an important role, pictures with a higher pixel exhibit more prominent features.

Exploratory Visualization

We will try to understand the distribution of the labels that are available in the train dataset. Some of the important features of the train data are:

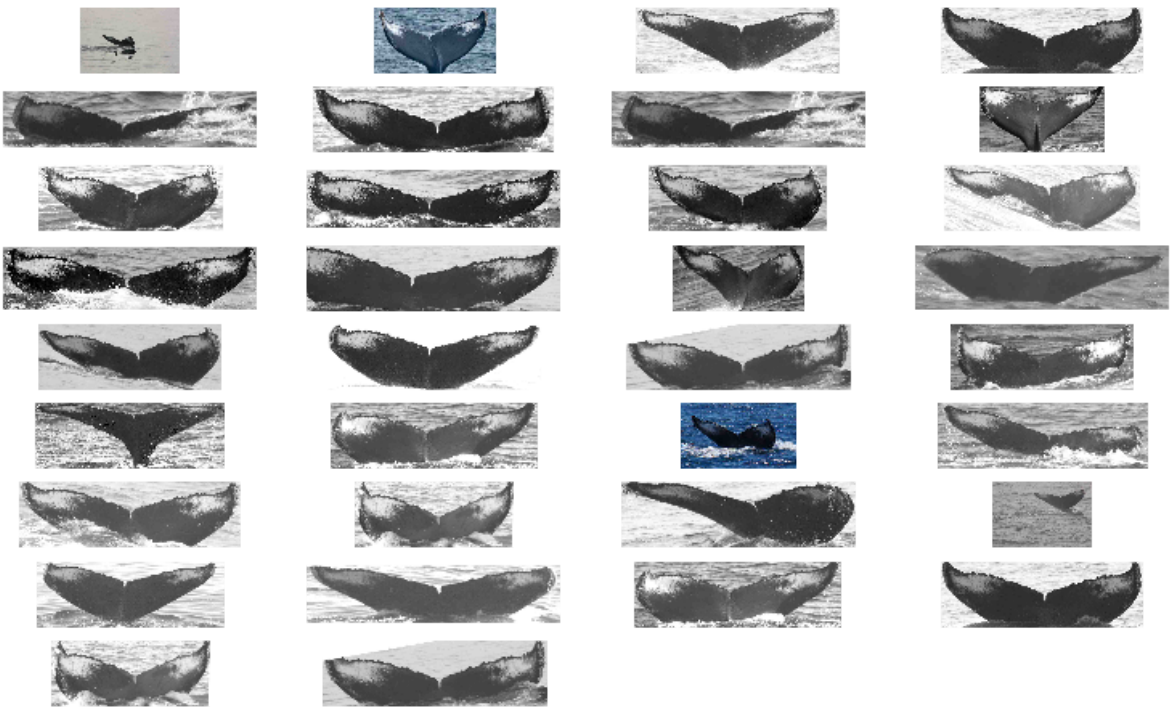
- There are a total of 4251 categories
- The number of image for the categories is very skewed. There are vast majority of classes that have only a single image in them. This makes the image problem harder.



- Interestingly the new whale category has the highest number of images however these are multiple whales which does not have a labelling in the existing database. The second whale with label w_1287fbc has about 34 images.

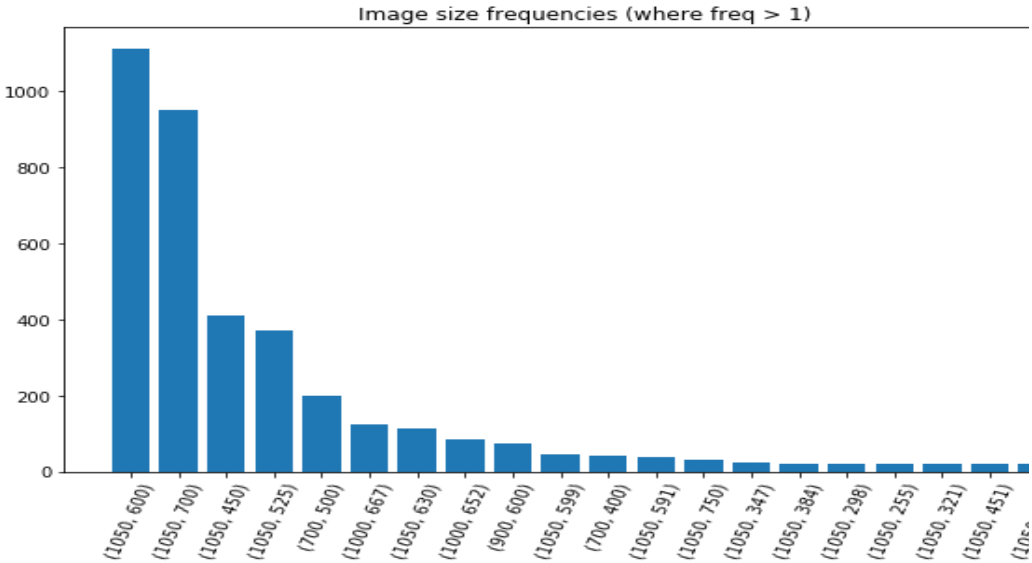
new_whale 810
w_1287fbc 34
w_98baff9 27
w_7554f44 26
w_1eafe46 23

- A sample of pictures of a particular class w_1287fbc is as below:



- Majority of the images on top are black and white which indicates we can have a basic image augmentation is to convert all images into black and white.

- The input images are of varied size and shapes, here is a plot of distribution of the different image sizes.



Algorithms and Techniques

The algorithms and techniques that are used here as mentioned below:

Neural Networks and CNN:

In general Neural Network are Universal approximates and it consists of machine learning methodology which try to mimic the human brain. Neural networks and deep learning currently has proven to provide the best solutions for many problems in image recognition, speech recognition, and natural language processing which are mundane tasks for humans.

In this problem I am specifically implementing the Convolutional Neural Network (CNN) a specialized Neural Network which is set to specifically mimic the visual cortex of a brain. There is minimal preprocessing for image based inputs while a CNN is used. CNN comprises of the following layers:

1. Convolutional layer: Unlike any traditional neural Network where the input is a vector the Convolutional layer takes the input is a multi-channeled image. In a colour image it is 3 channels (Red, Green , Blue). In this layer it convolves meaning it multiply the input image with a filter of a specific size.
2. The next layer is a pooling layer, which combines the outputs of neuron clusters at one layer into a single neuron in the next layer
3. Fully connected layers connects every neuron from above layer to a traditional network.

Image Data Generator:

Python has a fantastic Image augmentation tool which comes with Keras. I would be using that as a tool for different image augmentation before feeding the image for the CNN. The best part of the Keras Image Generator is that it handles at one image at a time rather than taking all images and it provides the images for processing just in time for the CNN Network.

The set of functions that can use within the Image Generator are:

- Sample-wise standardization
- Feature-wise standardization
- ZCA whitening
- Random rotation, shifts, shear and flips
- Dimension reordering

- Save augmented images to disk.

Transfer Learning:

Transfer Learning is a type of learning applied in machine learning where a model that is trained for a specific task can be repurposed to be used for another task. This is often helpful to apply transfer learning procedure to speed up the process of training machine learning models.

It is specifically helpful to apply transfer learning using Pre-trained models. The steps involved are:

- Select Source Model - For the purpose of our problem here I will be selecting the VGG16 pre trained model.
- Reuse Model - I will use this as a starting point for a model on the second task of interest.
- Tune Model - I will tune the model on our input-output pair data to increase the accuracy.

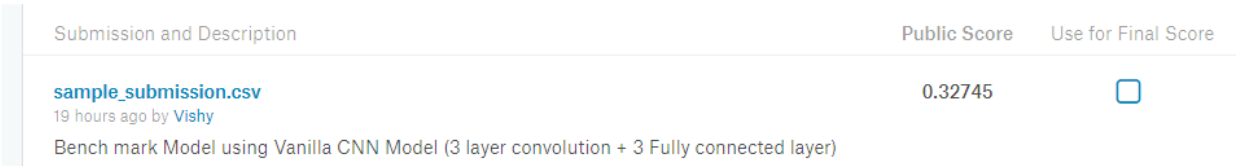
Popular Image based Pre trained Models:

- Oxford VGG Model
- Google Inception Model
- Microsoft ResNet Model

Benchmark

The Bench for a the project is a simple vanilla CNN network. The architecture of this network will be as follows:

The MAP@5 score for this simple vanilla CNN network is 0.32745



I would be focusing on building a transfer learning approach based solution to ensure that I am able to improve this score.

III. Methodology

Data Preprocessing

The python image PIL library along with the Image Data Generator function was quite resourceful in handling and creating the necessary image processing before feeding them as input. Here are the list of processing that has been followed and its justifications:

- Resizing of the images into 224*224 as size since the final version of the model includes including a pre-trained VGG16 (using imagenet weights)
- Nearly half of the pictures in the train/test data were grayscale images and I had to convert them into dumb RGB since VGG16 accepts input in the form of 224*224*3
- Using the Image Data Generator Keras function in Python after several attempts identified the best version of the arguments for the function

```
image_gen = ImageDataGenerator(featurewise_center=True, featurewise_std_normalization=True,
                               rescale=1./255, rotation_range=15,  width_shift_range=.15,
                               height_shift_range=.15, horizontal_flip=True,)
#training the image preprocessing
image_gen.fit(x_train, augment=True)
```

- There were version of a simple conversion of all images into grayscale images that was also tried which did not result in a good accuracy.

Implementation

After trying several networks & techniques the best version that I ended up is the transfer learning methodology. The final model that is implemented is the base VGG16 model and the fully connected layer with a 2 depth neural network.

Some details on the VGG16 are:

- Convolutions layers (used only 3*3 size) , Max pooling layers (used only 2*2 size) & Fully connected layers at end
- There is a total 16 layers
- The model is about 528 MB in size

The representation of VGG16 is as below:

The 16 layers that encompass are:

1. Convolution using 64 filters
2. Convolution using 64 filters + Max pooling
3. Convolution using 128 filters
4. Convolution using 128 filters + Max pooling
5. Convolution using 256 filters
6. Convolution using 256 filters
7. Convolution using 256 filters + Max pooling
8. Convolution using 512 filters
9. Convolution using 512 filters
10. Convolution using 512 filters + Max pooling

11. Convolution using 512 filters
12. Convolution using 512 filters
13. Convolution using 512 filters + Max pooling
14. Fully connected with 4096 nodes
15. Fully connected with 4096 nodes
16. Output layer with Softmax activation with 1000 nodes

Keras has the flexibility to assist in downloading the network using:

```
keras.applications.vgg16.VGG16(include_top=True, weights='imagenet',
input_tensor=None, input_shape=None, pooling=None, classes=1000)
```

Once called the model is downloaded and will reside in the local system, In my technique I would be removing the last layer of the network and replace to include a custom layer which used Softmax activation for the 4251 classes.

The full architecture of the network will be as follows:

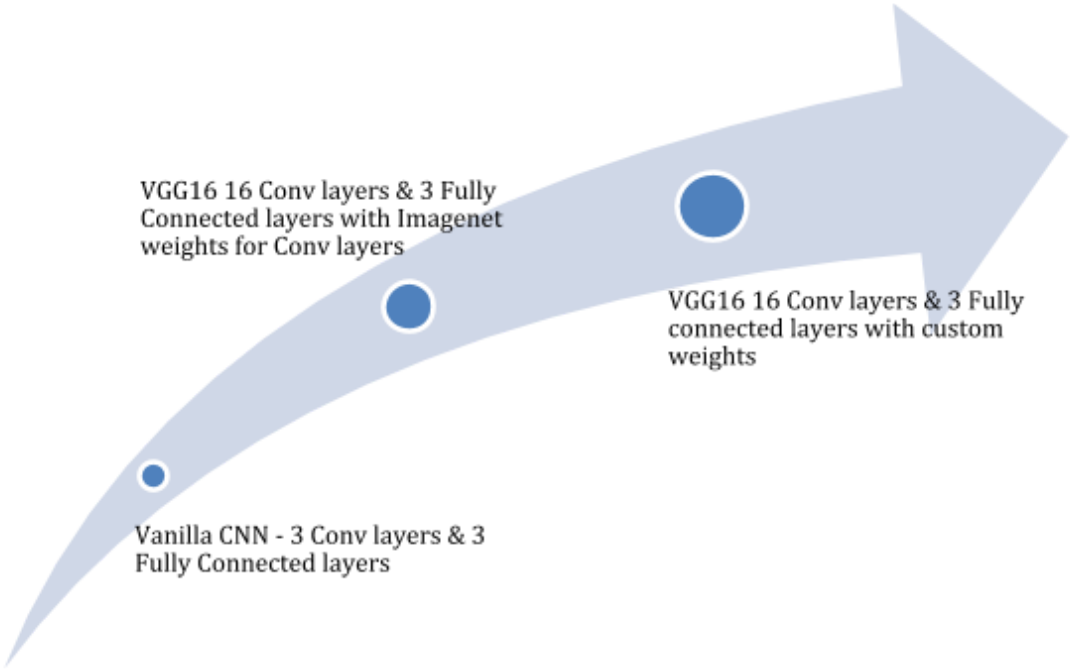
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               3211392
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 4251)              276315
=================================================================
Total params: 18,210,651
Trainable params: 18,210,651
Non-trainable params: 0
```

Some of the complications that I have faced in coding were:

- Prior to using Image Generator the task of conversion of images as a matrix input was challenging with frequent “Resource Exhaust” error messages
- Since the VGG16 is relatively big model, I could not run the algorithm with a larger batch, I was able to optimize the resource utilization and run time using a batch size of 64
- Model accuracy improved significantly after introducing regularization parameter Dropout. I have chosen a more aggressive the dropout of 0.50 leading to the best result.
- Increasing the Tuning parameter increased the cost of resource usage

Refinement & Iterations

From the benchmark Vanilla CNN model to the final below picture depicts the various iterations and refinements.



IV. Results

Model Evaluation and Validation

The Model evaluation score is based on the MAP@5 metric that was chosen for the project. The final model is able to beat the suggested best benchmark of 0.3275 that was set. The table below indicates the immediate increase in result after using a pre-trained model with its complex architecture.

Version	Implementation Architecture	Best result
Bench Mark Model		
1	A 3 layer convolutional model with 3 fully connected layers. Total params: 622,555 Trainable params: 622,555 Whale_Model1.ipynb	
	Submission 1	0.32700
	Submission 2	0.32745
Transfer Learning Approach		
2	Applying pre-trained VGG16 with unchanged imagenet weights and 3 fully connected layers. Total params: 18,969,939 Trainable params: 4,255,251 Whale_Model2.ipynb	0.36376 (Best score)

3	Applying pre-trained VGG16 with option to update weights and 3 fully connected layers + a custom label weight parameters Total params: 18,210,651 Trainable params: 18,210,651 Final_Whale_Model.ipynb	0.38369 (Best score)

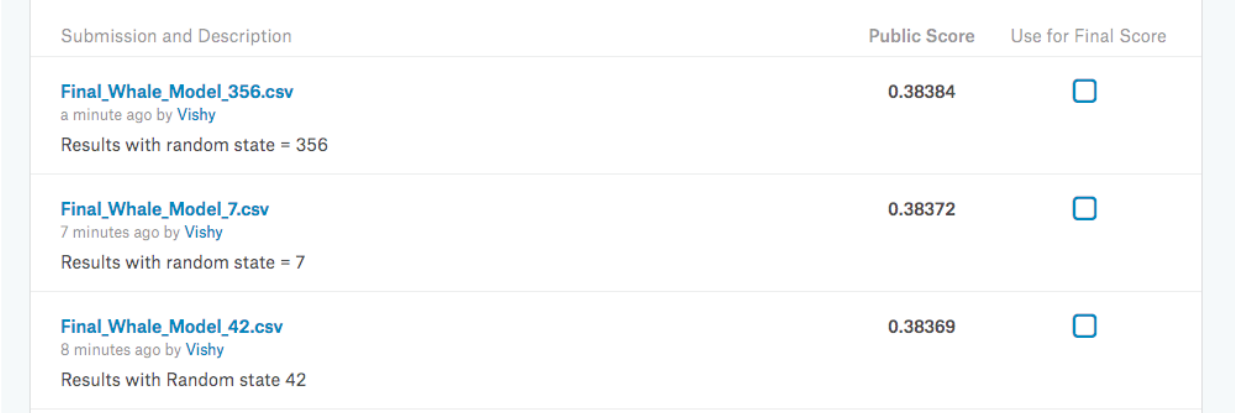
Justification

For me to show that the model was Robust enough, I would be trying to use various Train and Validation data split using different random state setting.

A important note to be mentioned is that I would not be able to replicate the MAP@5 metric calculation for the validation data that I have created from train data. Hence the default Keras Classification accuracy is considered. The best epochs model is used to then predict on the Test data and the last column in the table indicates the score in Kaggle.

S.No	Validation Data	Best Epoch Validation score (Metric - Accuracy)	Submission Score (Metric – MAP@5)
1	Random state = 42	Best Val_score = 0.0873	0.38384
2	Random state = 7	Best val_Score = 0.0744	0.38372
3	Random state = 356	Best val_Score = 0.0789	0.38369

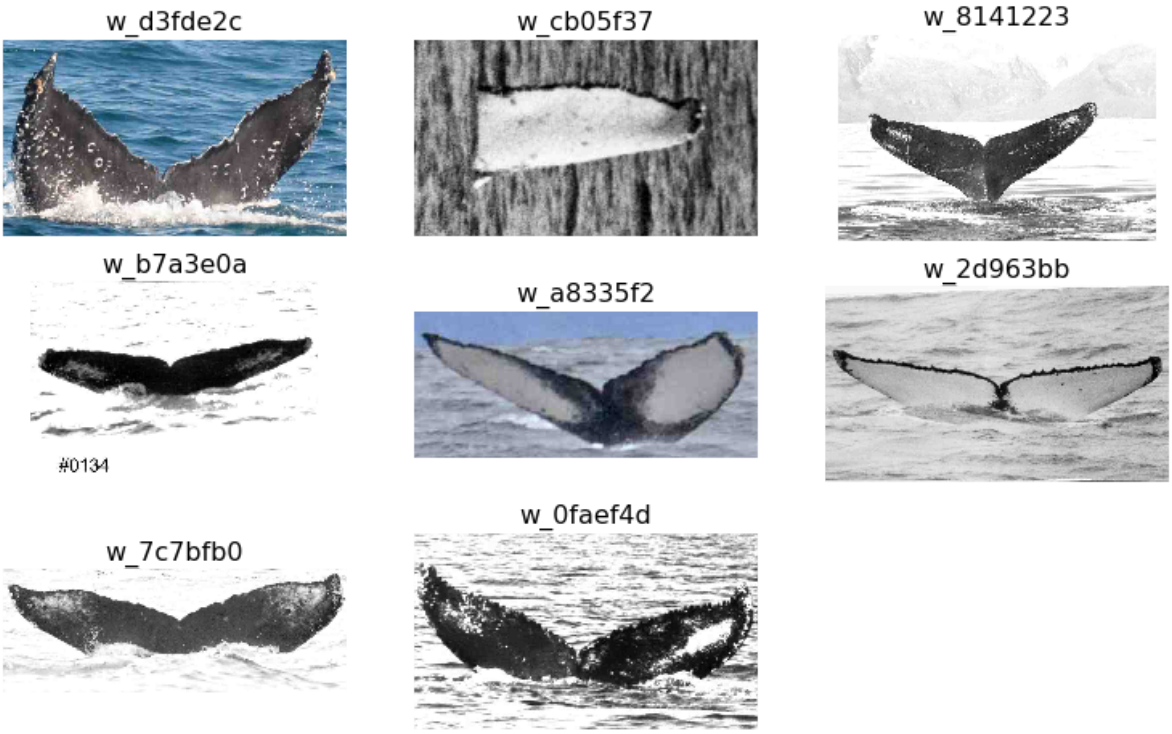
The table above depicts that the models are fairly performing quite similar. The best iteration appears to be with random_state=7



V. Conclusion

Free-Form Visualization

There were about 40 classes which has only 1 image for training and there was a sample image that I was able to observe which has only 1 tail. Using them in train or test dataset is quite tricky. As a next step I would be spending time to understand the number of classes with such images. Attached below is the example of classes with single images



Reflection

The complete solution can be summarized of applying the transfer learning architecture at the base layers and retraining the model to update the weights across all layers. Some of the important aspect that is worth noting and were challenging are:

- For the large multi class classification problems it is important to apply a deeper architecture.
- Running models with large batches will exhaust the resources hence applying steps per epoch makes it more efficient.
- Since the VGG16 was trained on over million images but was limited to 1000 classes using the existing imagenet weights does not provide a good result. This is because our model has 4251 classes against the pertained 1000 classes on a completely incomparable dataset to what I have on hand
- Since there is a very large class of New whale in the dataset it was very challenging and removing this class in training dropped the score significantly
- Due to acute class imbalance using weights helps.

Improvement

As it is reflected in the leaderboard result @ link my current submission scores at about 67th place, the 1st place submission has a score of 0.62722 against mine which is 0.38369.

Indicating the huge opportunities that are left for improvements. Some of the methodologies that can possibly improve the model are:

- Implementation of other pre-trained models apart from VGG16, or other pretrained models that is more specific to the data (whale trained model)
- Trying an ensemble version of different pre-trained models can also possible help
- Another idea that I could think of is to create a classifier to identify the black or white side of the whale fluke and the trained dedicated classifiers for both sides. This can prove to be effective considering that we have images representing both the sides.

(Note - All numbers presented in this section are true as of 12-May-2018)