Humpback Whale Identification Challenge

Capstone Project

Bo Liu
July 24st, 2018

I. Definition

Project Overview

After centuries of intense whaling, recovering whale populations still have a hard time adapting to warming oceans and struggle to compete every day with the industrial fishing industry for food.

To aid whale conservation efforts, scientists use photo surveillance systems to monitor ocean activity. They use the shape of whales' tails and unique markings found in footage to identify what species of whale they're analyzing and meticulously log whale pod dynamics and movements. For the past 40 years, most of this work has been done manually by individual scientists, leaving a huge trove of data untapped and underutilized.

The challenge is to build an algorithm to identifying whale species in images. I will analyze Happy Whale's database of over 25,000 images, gathered from research institutions and public contributors. And I am excited to help open rich fields of understanding for marine mammal population dynamics around the globe.

Happy Whale is the organization who provide this data and problem. It is a platform that uses image process algorithms to let anyone to submit their whale photo and have it automatically identified.

This challenge is originally form kaggle.com, more info can be find here.

Problem Statement

We have training data contains thousands of images of humpback whale flukes. Individual whales have been identified by researchers and given an Id. The problem we are going to solve is to predict the whale Id of images in the test set. What makes this such a challenge is that there are only a few examples for each of 3,000+ whale Ids. One relevant potential solution is training a model of CNN to identify those whale flukes.

Additionally, the problem can be quantified as follow, for an arbitrary image of whale flukes *x* (where *x* is the form of point series), and the corresponding whale Id *y*, find out a model *f(x)* that can map *x* to a candidate Id *y*.

The problem can be measured according to the precision at cutoff *k*.

The problem can be reproduced by checking the challenging website and public datasets of whale flukes.

Metrics

We use accuracy as the metric to evaluate the performance of our model.

$$accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y} = y)$$

wher $\dot{y}$ is the predicted label, $n_{samples}$ is the number of samples, *1(x)* is the indicator function.

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

And our final solution can be measured by the Mean Average Precision @ 5 (MAP@5) which is provided on the official website.

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

This training data contains thousands of images of humpback whale flukes. Individual whales have been identified by researchers and given an Id.

There are two folders containing training images and testing images separately, and one .csv file to map the training data to the right Id. Also a template of submission file is provided.

- train - a folder containing 9850 training images of humpback whale flukes.
- test - a folder containing 15610 test images to predict the whale Id.
- train.csv - maps the training Image to the appropriate whale Id.
- sample_submission.csv - a sample submission file in the correct format

Whales that are not predicted to have a label identified in the training data should be labeled as new_whale. Each of the images including a humpback whale fluke, So appropriately identifying those image are related to solving the problem we described before.

The dataset can be obtained from kaggle website , or you can click here.

The training dataset will be used to train our model, while the test dataset will be used to test our model and create a submission file. Our final results can be evaluated on MAP@5 by uploading our submission to the official challenge website.

For the reason that the dataset is over 700MB (9850 images) which is too large for my computer, so I sampled 2000 pictures from my project.

Training Images have various sizes and different color types, so I convert all images to gray and resize to 64x64. Figure 1 shows some of the training images.

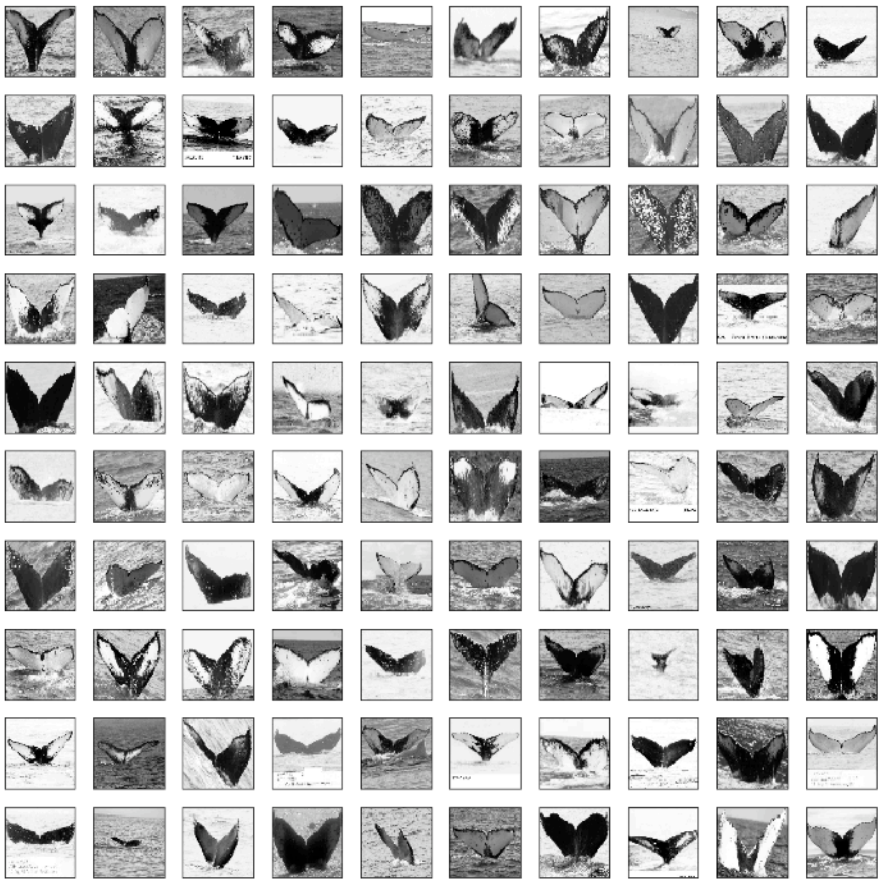


Figure 1 samples of training images

Exploratory Visualization

The competition states that it's hard because: "there are only a few examples for each of 3,000+ whale ids", there are actually 4251 categories in the training set.

There appear to be too many categories to graph count by category, so let's instead graph the number of categories by the number of images in the category which is shown on figure 2.

From the figure below we can see that, the number of samples in each category is very unbalanced.

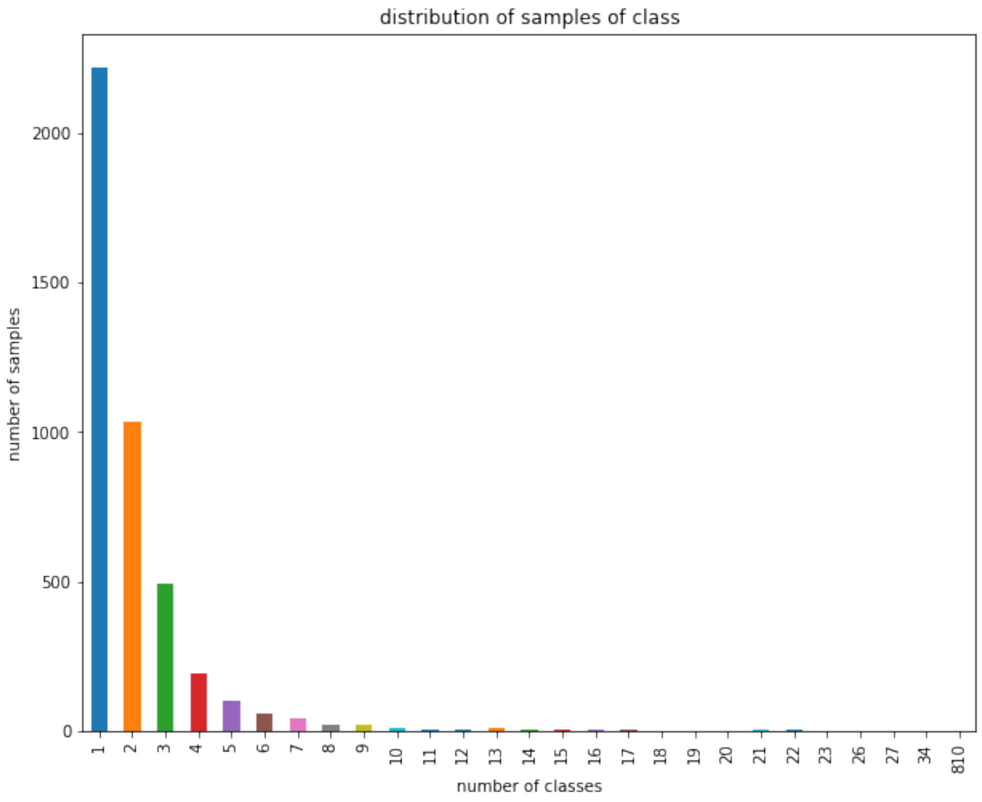


Figure 2 the number of categories by the number of images in the category

Algorithms and Techniques

A applicable solution to the problem is training a CNN model to predict the Ids.

The solution can be quantified as follow: Train a model *f(x)* using training data x_train, applying the trained model *f(x)* on the test data x_test to predict the candidate Id y.

First, we will apply a shallow CNN model with 8 layers as our benchmark model.Then, a more complexed model will be built to try to get better performance. VGG16 model is a good choice which is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. Finally, I try to using a 'combined CNN model', in which I will train shallow CNN models for each class and combined all those models as a whole to predict on testing dataset.

Benchmark

This link provided one possible solution using a traditional CNN in Keras, which I would like to be taking into account as a benchmark model.

The workflow of the benchmark model is:

importing the data -> One hot encoding on the labels -> Image augmentation -> Building and training model -> Predictions on test samples

It is a simple CNN model containing 8 layers as shown below.

Input -> conv2D@1 -> Conv2D@2 -> MaxPooling2D@3 -> Conv2D@4 -> MaxPooling2D@5

-> Dropout -> Flatten -> Dense@6 -> Dense@7 -> Output@8

Where 'Input' represent input layer, 'conv2D' represent convolutional layer, 'MaxPooling2D' represent max pooling layer, 'Dense' represent fully-connected layer, 'Output' represent output layer, '@n' indicate that this is the 'n'th layer, 'dropout' and 'flatten' represent dropout and flatten method separately, and they aren't considered as a layer. All parameters are shown below:

- batch size - 128
- epochs - 9
- conv2D@1 - number of kernels is 48, kernel size is (3, 3), strides is (1, 1), padding type is 'valid', activation function is 'relu'.
- conv2D@2 - number of kernels is 48, kernel size is (3, 3), strides is (1, 1), padding type is 'valid', activation function is 'sigmoid'.
- MaxPooling2D@3 - pool size is 3, 3), strides is (1, 1), padding type is 'valid'.
- conv2D@4 - number of kernels is 48, kernel size is (5, 5), strides is (1, 1), padding type is 'valid', activation function is 'sigmoid'.
- MaxPooling2D@5 - pool size is 3, 3), strides is (1, 1), padding type is 'valid'.
- dropout - Fraction of the input units to drop is 0.33.
- flatten - no parameters.
- Dense@6 - number of cells is 36, activation function is 'sigmoid'.
- dropout - Fraction of the input units to drop is 0.33.
- Dense@7 - number of cells is 36, activation function is 'sigmoid'.

- output@8 - number of cells is equal to the number of unique labels, activation function is ’softmax’.
- loss function - cross-entropy
- optimizer - Adadelta.
- metrics - ’accuracy’.

This model got an accuracy of xxx on training dataset, and xxx on testing dataset. After uploading our final result to the challenge website, we got a score of 0.32660.

A script of this model is provided, name ’benchmark.py’, in the submission package.

III. Methodology

Data Preprocessing

4 preprocessing steps were adopt before building the model.

- Conversing all image to gray
- Reshaping all image to (64,64)
- Image argumentation
- sample balancing

Image argumentation is used with the following settings:

- Set input mean to 0 over the dataset, feature-wise.
- Divide inputs by std of the dataset, feature-wise.
- Apply ZCA whitening and set epsilon to 1e-06
- Random rotations within 10 degree
- Random width shift within 10 percent of total width
- Random width shift within 10 percent of total width
- Random shear shift within 10 percent of total width
- shear Intensity is 0.1
- zoom range is 0.1 percent of total size

- Points outside the boundaries of the input are filled with the nearest points
- We multiply the data by the value of 1./255 to rescale the image

Accroding to the Exploratory Visualization part, the number of samples in each category is very unbalanced. So it's necessary to eliminating sample bias by weighting data. The weighting formula is:

$$w = \frac{1}{x^{\frac{3}{4}}}$$

where *x* is the number of samples of a particular class, *w* is the weight for that class.

Implementation

What we implemented in our final project is VGG16 model, as we introduced in 'Algorithms and Techniques' section, it's a well performed model proposed by K. Simonyan and A. Zisserman.

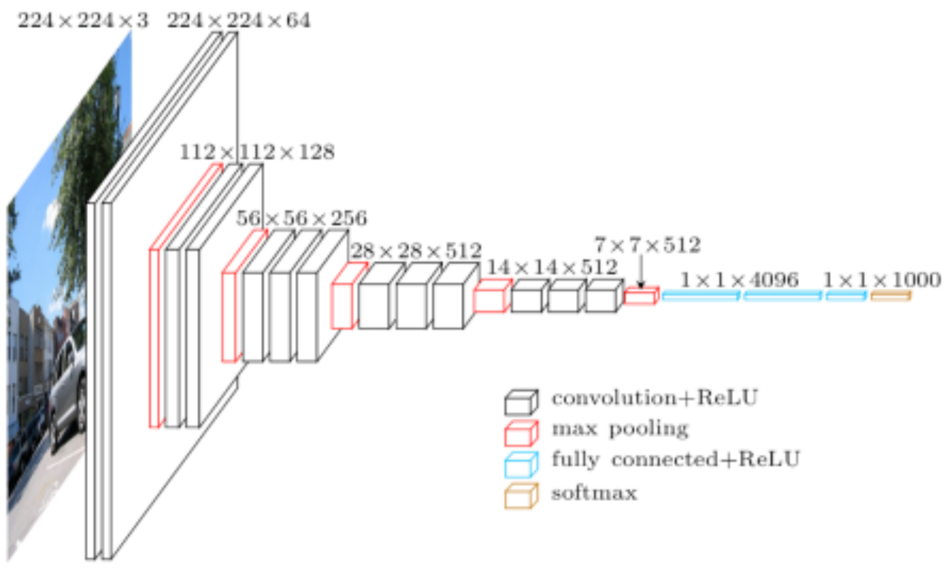
As described in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”, our model have 16 weight layers. The generic design of model is listed bellow:

- Input is passed through a stack of convolutional (conv.) layers.
- Filters in conv. layers have a very small receptive field: 3 × 3
- The convolution stride is fixed to 1 pixel; using 'same padding'
- Five max-pooling layers which follow some of the conv. layers
- Max-pooling is performed over a 2 × 2 pixel window, with stride 2.
- A stack of convolutional layers is followed by three Fully-Connected (FC) layers
- The last FC layer is set to has as many channels as the number of output class
- The final layer is the soft-max layer

The architecture of our model including 16 weight layers which organized by 6 separate blocks.

- Block 1 has two 3x3 convolution with 64 filters(aka conv3-64) layers followed by a max polling layer.
- Block 2 has two conv3-128 layers followed by a max polling layer.
- Block 3 has three conv3-256 layers followed by a max polling layer.
- Block 4 has three conv3-512 layers followed by a max polling layer.
- Block 5 has three conv3-512 layers followed by a max polling layer.
- Block 6(also named Classification block) has three fully-connected layers, the first two has 4096 neurons and the last one has as many neurons as the number of output class

The macro-architecture of VGG16 can be described in the following Fig.



Refinement

To acquire improved solutions I adjusted some parameters and utilize Fine-turning to improve our algorithm.

In the benchmark model, the epoch was set to 9. in our model we increased the epoch to 20 since more training round(in a finite range) can bring us more accuracy.

Deep learning models are by nature highly repurposable, so I have also adopted a training strategy called Fine-turning In order to improve efficiency. The optimizer and learning rate can be modified in different Fine-turning phase.

Fine-turning means using weights of a well-trained model without the top layers, and adding our customized top layers to create the final model, then fit the training data to refine the model.

More details can be found in 'IV.Results' part.

IV. Results

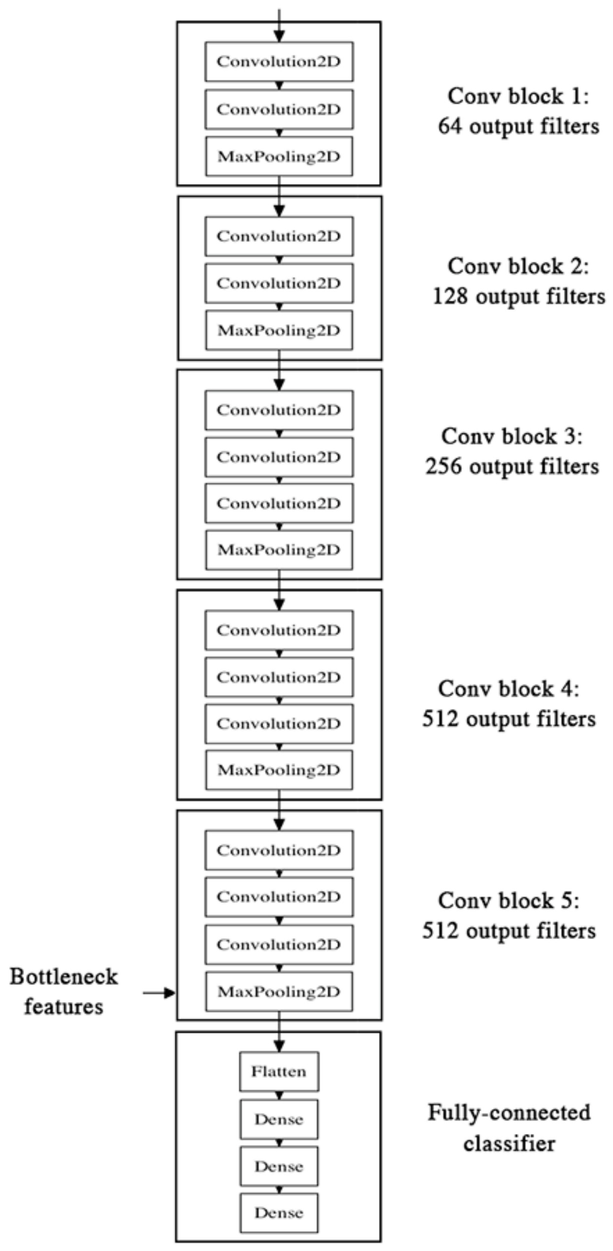
Model Evaluation and Validation

In this section, I will introduce how our model be trained and evaluated.

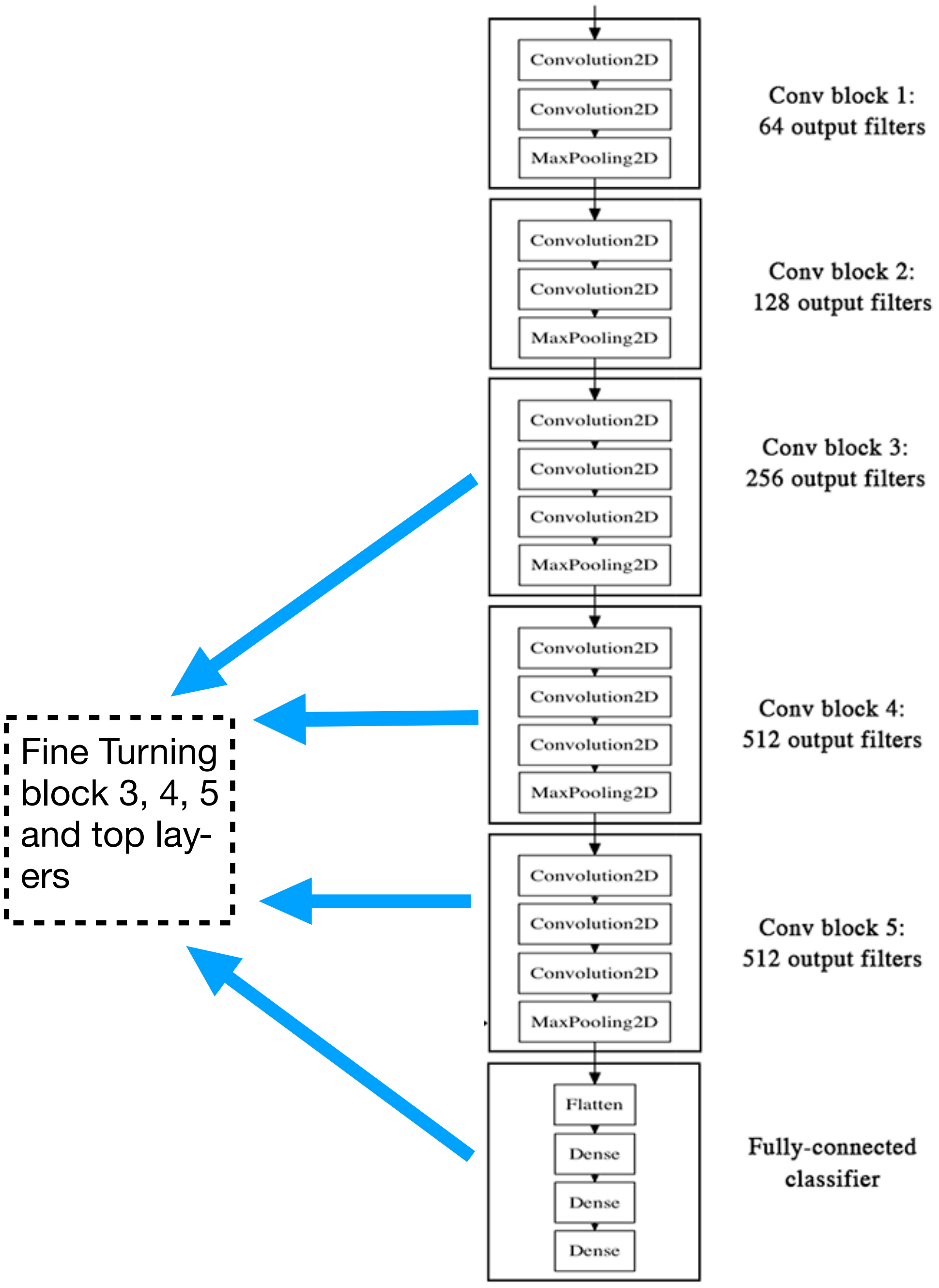
First, explain training parameters. Batch size is set to 128 which is the max value that fit n power of 2 that My machine can handle. The epoch was set to 20, this is also a big number for my computer, it will take around a whole day to finish computing.

Our training processing is divided into two stages, the top-layers-training stage, and the *Fine-truning* stage.

At stage one, we load weights of pre-trained VGG16 model without the last three Fully-Connected (FC) layers which we called 'the top layers' to get the bottleneck feature, and adding four customized layers on top of that: an average pooling layer, two Fully-Connected layer with 4096 neurons and *Relu* activate function, an output layer which has as many outputs as the data classes. All convolutional layers should be frozen at this stage and train only the top layers which were randomly initialized. *Rmsprop* optimize algorithm was picked as our optimizer which can dynamic adjust the learning rate, the loss function is categorical cross-entropy.



At stage two, the top layers are well trained and we can start fine-tuning convolutional layers from VGG16. We will freeze the bottom 2 blocks and train the remaining top layers, this is, we will freeze the first 6 layers and un-freeze the rest. We need to recompile the model with SGD optimizer, we use SGD with a low learning rate of 0.0001, cause we have to fine turning more carefully.



Justification

I have trained 3 model for comparison: benchmark model, VGG-16 model without fine-turning (initialize with random weights) , VGG-16 model with fine-turning.

The results are recorded and show as a table below:

model	traning loss	training accuracy	testing score
CNN	7.2009	0.0790	0.32660
VGG16	8.1076	0.0822	0.32786
VGG16-ft	7.9974	0.0913	0.33216

VGG-16 fine-turning model didn't get a big promotion result for the following reasons:

1. The samples is much more less in each category than the original training dataset for VGG16 model.
2. Different type of Humpback Whale image are too similar, even a small dot is the key point to separate different whales, this is much more detailed than objectives like cat and dog.
3. Limited to my computer's performance, I have only tried 10 epochs for fine-turning, the accuracy haven't stop increasing.
4. To get the final score, I have to use the whole data set for training and testing, which contains too many categories, maybe a small dataset can give the model more power.

V. Conclusion

Reflection

To predict whale Ids from humpback whale images, we gathered thousands of images of humpback whale flukes with appropriate Id and build a model try to resolve the problem.

We built a shallow CNN model with 8 layers as our benchmark model and a more complexed model VGG16 to try to get better performance. And 4 preprocessing steps were adopt before building the model.

Finally our model were tested both on training data and testing data with accuracy as the algorithm metrics, we also uploading the final result to the official competition website to get MAP@5 score.

Results show that more deeper CNN net bring about more accuracy, nevertheless Fine -turning may not be develop a good performance when the dataset have a big differences.

Improvement

There still be a big gap between our model and a real effective one, the following method may help to facilitate the algorithm.

- More aggressive data augmentation
- Getting more samples
- More aggressive dropout rate
- Using of L1 and L2 regularization in case of overfitting
- Fine-tuning more convolutional block (alongside greater regularization)