Humpback Whale Identification

Eddie Tseng
UC San Diego
edtseng@eng.ucsd.edu

Chun Hu
UC San Diego
chh281@eng.ucsd.edu

Ping-Chun Chiang
UC San Diego
p3chiang@eng.ucsd.edu

Abstract

We study and implement three different models for humpback whale identification based on modern convolutional neural networks. The problem of this classification is formulated as learning with unbalanced data. According to our experimental results, the convolutional neural networks are capable of capturing some context of each humpback and make accurate inferences. Finally, We present a detailed empirical analysis by visualizing the architectures, training process and prediction results.

1. Introduction

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 are 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 problem of humpback whale identification, defined as the **problem of classifying imbalanced data** [12]. In this project, we aim to analysis and implement a classifier to identify whales identity. The convolutional architecture (AlexNet) proposed by Krizhevsky *et al.* [7] has shown a great potential on object detection and classification. In this paper, we are going to exploit the convolutional architectures and learn the underlying principles of designing a learning machine for image classification. Though there have been much work in image classification [6], we would like to implement and compare the error rate of several algorithms and models such as AlexNet[7], VGG19[11] and Siamese Network[3] to find out the most suitable classification algorithm for this dataset. [9]

Figure 1: Example of whales humpbacks [2]

2. Method

2.1. Prediction

Given an input image $\mathbf{x} \subset \mathcal{X} \in \mathbb{R}^{W \times H \times 3}$, we will use a given decision function $g(\mathbf{x}; \mathbf{w}) : \mathcal{X} \rightarrow \mathcal{Y}$ to predict the label $\hat{\mathbf{y}} \subset \mathcal{Y}$ where

$$\begin{aligned} \mathcal{Y} &= \begin{bmatrix} \boldsymbol{\omega} \end{bmatrix} \\ &= \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_k \end{bmatrix} \end{aligned} \tag{1}$$

with $\omega_1, \omega_2, ..., \omega_k \in [0, 1]$. Here, we consider the number of labels find in given image $\mathbf{x}$ with $k = 4251$. For example, one can easily set $\mathbf{x}$ as label 14 if $k = 14$ has the highest value, which is 1 (after softmax). For every $\omega_i, i = 1, 2, ..., k$, it indicates the probability of

k	Image	Id
1	00022e1a.jpg	w-e15442c
2	000466c4.jpg	w-1287fbc
3	00087b01.jpg	w-da2efe0
4	001296d5.jpg	w-19e5482
5	0014cfdf.jpg	w-f22f3e3
6	0025e8c2.jpg	w-8b1ca89
7	0026a8ab.jpg	w-eaad6a8
8	0031c258.jpg	new-whale
9	0035632e.jpg	w-3d0bc7a
10	0037e7d3.jpg	w-50db782
11	00389cd7.jpg	w-2863d51
12	0042dcc4.jpg	w-6dc7db6
⋮	⋮	⋮
9849	fff04277.jpg	w-2dcbf82
9850	fffd4260.jpg	w-b9bfd4e

Table 1: The labels of each image

the i^{th} label in the given image $\mathbf{x}$. That is,

$$\begin{cases} \omega_i = 0, & \text{if } i^{th} \text{ label is not likely in } \mathbf{x} \\ \omega_i = 1, & \text{if } i^{th} \text{ label is likely in } \mathbf{x} \end{cases} \tag{2}$$

Therefore, the goal is to design an classifier which can produce an estimate given an image $\mathbf{x}$. To be more clear, we summarize the prediction in Eq. 3.

$$\begin{aligned} \hat{\mathbf{y}} &= g(\mathbf{x}; \mathbf{w}) \\ &= softmax\left[\hat{\boldsymbol{\omega}}\right] \end{aligned} \tag{3}$$

Note that $g(\mathbf{x}, \mathbf{w})$ is the output of the learning machine to input $\mathbf{x}$ when its parameters have value $\mathbf{w}$.

2.2. Learning with Original Data

Given a set of input images $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \subset \mathcal{X}$ and their corresponding Id $\{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n\} \subset \mathcal{Y}$, we want to learn a decision function $g(\mathbf{x}; \mathbf{w}) : \mathcal{X} \rightarrow \mathcal{Y}$ such that the structural risk is minimized. Now, consider a set of functions $S = \{g(\mathbf{x}, \mathbf{w}), \mathbf{w} \in \mathcal{W}\}$ implemented by a convolutional neural network of fixed architecture with the weights $\mathbf{w}$ and a structure through $S_p = \{g(\mathbf{x}, \mathbf{w}), \|\mathbf{w}\| \leq C_p\}$ with $C_1 < C_2 < ... < C_n$ [8]. The minimization of the empirical risk within

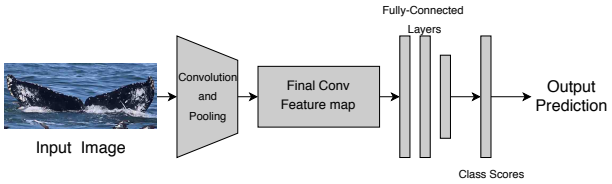

Figure 2: The illustration of Multi-label classification. With the class estimate $\hat{\mathbf{y}} = argmax(\hat{\boldsymbol{\omega}})$.

the structure S_p is

$$R_{emp}(\mathbf{w}, \beta_p) = \frac{1}{n} \sum_{i=1}^{n} L(\hat{\mathbf{y}}_i, \mathbf{y}_i) + \beta_p \|\mathbf{w}\|^2 \tag{4}$$

where β_p is the scale of introducing "weight decay" to a neural network.

Practically, we design a network to do the multi-label classification. By observing the (ω_i) tuple, it is obvious that we can decompose the problem to have a simple feed foward network classification head as we show in Fig. 2. To be more specific, the loss function of the classification head can be chosen by

$$L(\hat{\boldsymbol{\omega}}^i, \boldsymbol{\omega}^i) = -\frac{1}{k} \sum_{j=1}^{k} \left[\omega_j^i \log \hat{\omega}_j^i\right] \tag{5}$$

which is called "categorical cross entropy"[4].

After specifying the loss functions, we can then apply *stochastic gradient descent* to train a deep neural network. Note that we still need to pick a specific architecture of the deep neural network to fit into our application of humpback whale identification. In this paper, we pick **deep convolutional neural networks** including **AlexNet** [7] and **VGGNet** [11] as our classifiers.

The advantages of using **DNN-based classifiers** is that we do not need to define a domain specific model. The deep neural networks have the capability to implicitly learn from the data.

2.3. Learning with Classical Data Augmentation

After training on original data, we got 99% training accuracy. "Fantastic" we think. However, the test result Then we dive a little deeper and discover that 90% of the data belongs to one class. So what is **Imbalanced data**? Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally.[1]

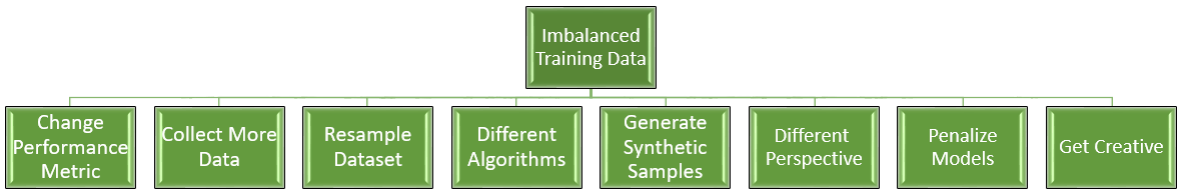

Figure 3: An illustration of fighting imbalanced data

There are many ways to solve the imbalanced data problem. For example, collecting more data, changing performance matric, resampling dataset, penalized models and even be creative.

For **collecting more data**: A larger dataset might expose a different and perhaps more balanced perspective on the classes. For **changing performance matric**: use some Confusion Matrix, Precision, Recall or F-1 score. For **resampling dataset**: Add copies of instances from the under-represented class called over-sampling. Delete instances from the over-represented class, called under-sampling. However, in humpback whale identification problem, under-sampling is not applicable. For "new whale" class, each image represents a new species of whales family. Thus, the result will not be robust if we delete some images in dataset. For **penalized models**: Penalized classification imposes an additional cost on the model for making classification mistakes on the minority class during training. These penalties can bias the model to pay more attention to the minority class. Often the handling of class penalties or weights are specialized to the learning algorithm. We will explain how we do **Class Weight Penalty** in the next section. And For **creative**: Really climb inside your problem and think about how to break it down into smaller problems that are more tractable.

2.4. Learning with Refining Class Weight

Many machine learning toolkits have ways to adjust the importance of classes. Scikit-learn, for example, has many classifiers that take an optional "class-weight" parameter that can be set higher than one. However, we decide to design it by ourselves.

$$ClassWeight = \frac{1}{x^{0.75}} \tag{6}$$

where

$$\mathbf{x}_t = valuecount\{C_t\} \tag{7}$$

$C = class$, $t = classindex$. This equation is simply a collection of classes total number from the original dataset. If the number of the class is large, new class weight will become very small. Similar, if the class's data is tiny, then new class weight will become relatively large. This method will help us get "true" accuracy and loss when we train.

2.5. Transfer Learning

The architecture of transfer learning is shown is 4 There are many state-of-the-art models, VGGNet[11], Inception[14], ResNet[13], etc.. Each of these models get great result and we liked to stand on the shoulders of giants to use the weight they have already trained. It not only save time but also get better results. For this project, we will implement **VGG19** and **InceptionResV2**.

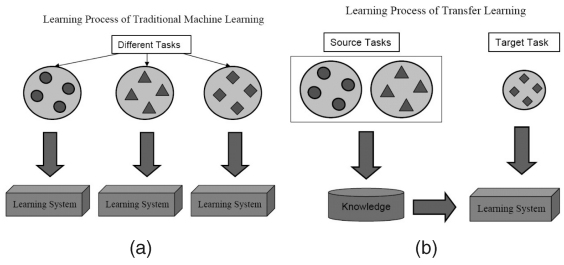

Figure 4: Different learning process between (a)traditional machine learning and (b)transfer learning[10]

Since the data is small and imbalanced, we freeze the lower ConvNet blocks as fixed feature extractor.Like 4 Take a ConvNet pretrained on ImageNet, remove the last fully-connected layers, then treat the rest of the ConvNet as a fixed feature extractor for

the new dataset. Training the new fully-connected layers (green, aka. bottleneck layers). Extract the CNN codes for all images, train a linear classifier for the new dataset. Finally, fine-tuning the ConvNet. Replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pre-trained network by continuing the back-propagation to part of the higher layers.

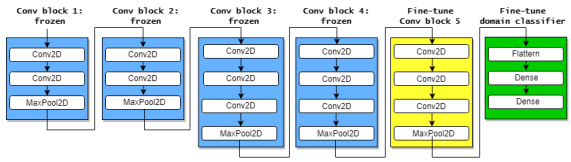

Figure 5: VGG19 ConvNet Fine-Tuning Technique for adapting to different domain

2.6. Evaluation

To evaluate the classifier on each testing image, we need to define an evaluation metric. Thanks to Kaggle's well defined evaluation metric **Mean Average Precision**[2].

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

where U is the number of images, P(k) is the precision at cutoff k, and n is the number predictions per image. For each image in the test set, we can predict up to 5 labels for the whale. Simply to say, we predict top five labels for each image. Then if the first(top) is correct, we got 5 points. If the second is the correct answer, then we got 4 points. Finally we divide by U to get MAP.

3. Experiments

3.1. Dataset

In this project, The model will be trained and evaluated on **Kaggle's Humpback Whale Identification Challenge**. In this Kaggle competition, we are given a dataset of 9,850 images containing humpback whales and are asked to build a classifier to identify which whale an image contains (or identify new whales). These images in the training set come with labels given an ID to identify the whales in the images. Every whale has its own certain ID in training set and In the training set we examine the labels, there are 4251 different whale labels among 9850 images. The problem is that there is only few images in each class. However, the distribution is not as average as our expectation.

3.1.1 Dataset Distribution

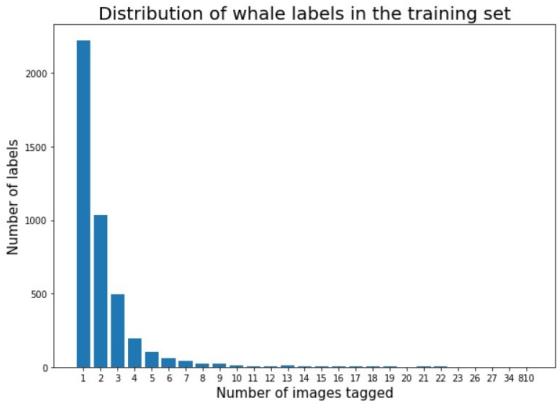

Figure 6: Distribution of whale labels

In Fig. 6 shows that there are 2220 whales that only have 1 image and 1034 whales have 2 images; and so on all the way down to the last whale containing 810 images. The class which contains 810 images is 'new whale' and has an significant percentage of full training data. Likewise, the classes contain only one image is another big problem which would influence the accuracy when implement the network. Later, We'll discuss how we go about doing the augmentation of this dataset.

3.1.2 Dataset Exploration

Now that we have identified our first issue of needing to augment our image dataset, we should examine the images themselves. If they don't share specific properties like size dimensions, color scheme, etc. then we may run into additional issue.

In Fig. 7, there's a number of things we can see from these first 25 images. First, they have different dimensions which means we will either have to make a classifier that doesn't care about image size, or we'll have to standardize the dimensions of each image for proper comparison.

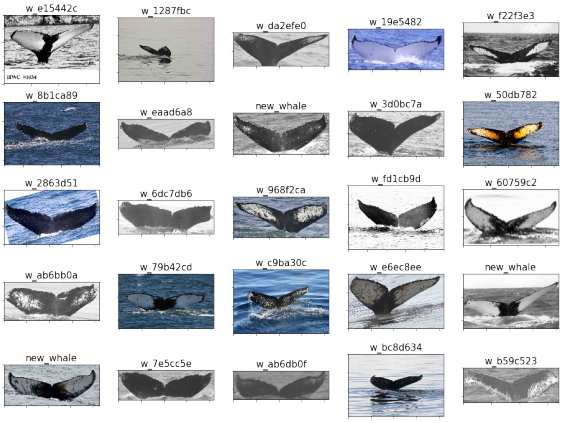

Figure 7: Whale images with label IDs

Second, some of the images are in gray scale, but others are in color which means our classifier will have to not care much about whether or not the image is in color, or we'll have to standardize all images to gray scale. It's likely a better approach to just test any given image for color, and then add the gray scaled version of it to the dataset as a separate image; this will help us in our augmentation step, since it adds more data to our sample. We can't make sure whether doing this will bias the classifier to misclassify whales in our dataset that originally don't have color if presented with a color version whale.

Lastly, you can see the first image has some words in the bottom portion of the image. The words in the label doesn't seem to correlate with the whale's label. Having images with these arbitrary words at the bottom could also further complicate things when we ignore them. This issue would truly affect the result especially since there many of these labels (not shown in the figure). However, we can't find out all this kind images, hence, we ignore this issue when we implement our project.

3.2. Platform

Since the availability of the powerful Graphics Processing Units (GPUs) will be a crucial key to train a deep convolutional neural network. We use the compute engine with one NVIDIA® GTX® **1080ti** GPU on **UC San Diego Data Science/Machine Learning Platform** (DSMLP). Note that the memory size of 1080ti GPU is 11GB, so normally it has ability to accommodate most of implementation in deep convolutional architectures. However, for our tremendous labels' data augmentation, 11 GB is still not enough. Therefore, we made each of the image size to 128 x 128. The training speed is also a critical factor to the success in deep learning. Note that the processing power of 1080ti is 11.3 TFLOPS in double precision, so it really an ideal speed up method to our expectation of training deep networks.

To build networks, we choose to use **Tensorflow** [5] and **Keras** as our library due to the following tractabilities. First, by using **TensorBoard**, we are able to visualize and track our training progress in terms of accuracy and loss. Secondly, by coding in Tensorflow and Keras, we get a chance to understand the details in customizing estimators and loss functions.

3.3. Models

We implement classical CNN in four different types of convolutional neural networks, also, use transfer learning to compare with just training on our own model's weight. As we have mentioned in the previous section, the humpback whale identification needs to do multi-label classification, so the strategy here is to train the classification model with different perspective.

Notice that the classification outputs 4251 probabilities $\hat{\omega}_1, \hat{\omega}_2, ..., \hat{\omega}_{4251}$ that indicate the presence of each Id of each image. The descriptions of four different types of convolutional neural networks are presented as following.

3.3.1 Baseline

In classification head, Baseline consists of two convolutional layers, which are all followed by max-pooling layers, and three fully-connected layers.The architecture is illustrated in Fig. 8.

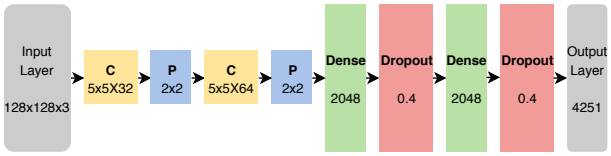

Figure 8: The architecture of Baseline. C denotes the convolutional layer, P denotes the max-pooling layer and Dense represents the fully connected layer.

3.3.2 Non-separated AlexNet

We use a non-separated AlexNet since 1080ti can almost accommodate the whole network (we add a pooling layer at the output of the fourth convolutional layer reduce the number of weights to fit the whole network in 1080). The architecture is illustrated in Fig. 9.

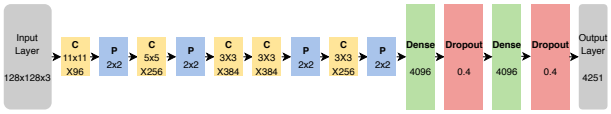

Figure 9: The architecture of non-separated AlexNet.

3.3.3 VGGNet

VGGNet is the runner-up in ILSVRC 2014 which was proposed by Karen Simonyan and Andrew Zisserman. Their main contribution is that showing the depth of the network is a critical component for good performance (keep it deep, keep it simple). In this project, we use VGG-19. The architecture is illustrated in Fig. 10.

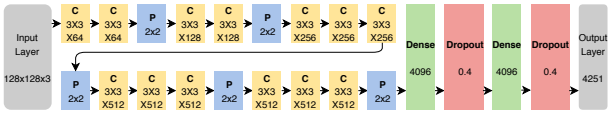

Figure 10: The architecture of VGGNet (VGG-19).

3.3.4 Transfer learning (VGG-19)

We assume through transfer learning, we can save a lot more time and get a better result. Whereas the result is quite surprising for us. From the knowledge we have, transfer learning for small dataset requires add few FC layers and output layer, set the weights for earlier layers and freeze them. All these method was done as we expected, loss was decreasing, accuracy for every epochs was increasing. But the overall accuracy is still very low even we train as much as we can (lower then Baseline when submitting the predict labels).

3.3.5 Transfer learning (InceptionResnetV2)

After the frustration, we hypothesized transfer learning on VGG is not ideal network. Therefore we turned to newer model "InceptionResnetV2" which is created by Google. Not as we expected, the result was slightly better then VGG but still not good enough. Maybe we should Focus on new network, which lead us to **Siamese Network**.

3.3.6 SiameseNet with Triplet Loss

Before introducing Siamese Network, we want to talk about **one-shot learning**[3]. One shot learning is the technique of learning representations from a single sample. As we mentioned above, our dataset is imbalanced and more than two thousand classes only have one image. Therefore, building and training a typical convolutional neural network will not work as it cannot learn the features required with the given amount of data. So, this is a one-shot learning task where you build a similarity function which compares two images and tells you if there is a match.

In **Siamese networks**, we take an input image of a whale and find out the encodings of that image, then, we take the same network without performing any updates on weights or biases and input an image of a different whale and again predict its encodings. Now, we compare these two encodings to check whether there is a similarity between the two images. These two encodings act as a latent feature representation of the images. Images with the same whale have similar features/encodings. Using this, we compare and tell if the two images have the same whale or not.

Train the network by taking an query image and comparing it with both a positive sample and a negative sample. The dissimilarity between the query image and positive image must low and the dissimilarity between the query image and the negative image must be high.

This special network comes with a special loss function called **Triplet Loss Function**.

$$L = max(d(q,p) - d(q,n) + margin, 0)$$

The formula above represents the triplet loss function using which gradients are calculated. The variable q represents the query image, p represents a positive image and n represents a negative image. Another variable called **margin**, which is a hyperparameter is added to the loss equation. Margin defines how far

away the dissimilarities should be, i.e if margin = 0.2 and d(a,p) = 0.5 then d(a,n) should at least be equal to 0.7. Margin helps us distinguish the two images better.

Therefore, by using this loss function we calculate the gradients and with the help of the gradients, we update the weights and biases of the Siamese network. The architecture is illustrated in Fig. 11.

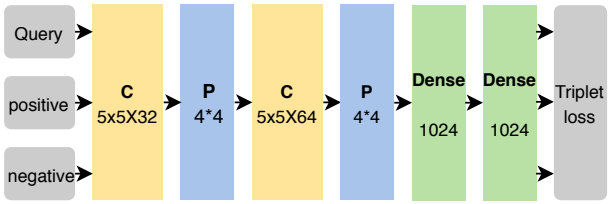

Figure 11: The architecture of Siamese Network.

3.4. Results

3.4.1 Training

With the number of training iteration increases, we compare our baseline model and Siamese model by the following perperties.

- Training accuracy
- Time to loss convergence

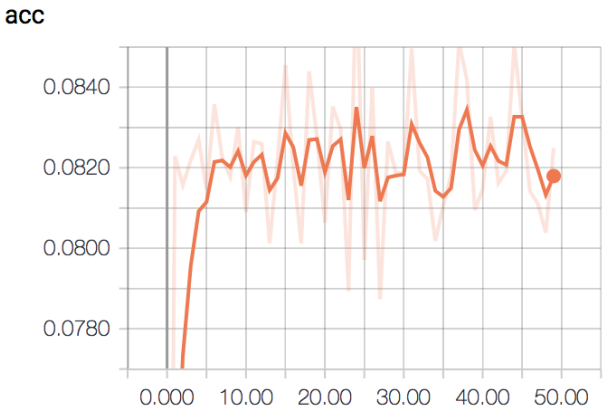

Figure 12: The training accuracy of baseline model.

As we can observe in Fig. 12, 13 and 14, there is a trade-off between accuracy and the difficulty of training a network. Since we can think of the training speed as an indicator of the difficulty to train a network.

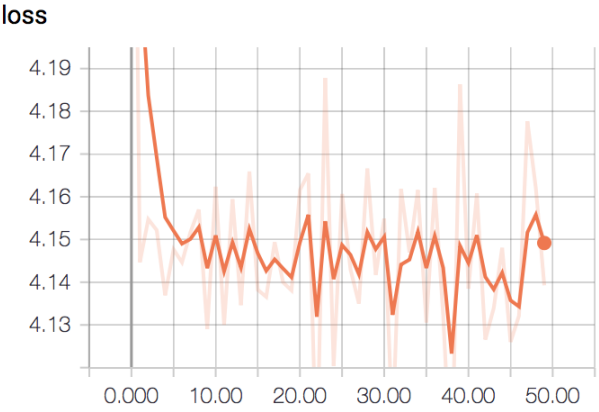

Figure 13: The training loss of baseline model.

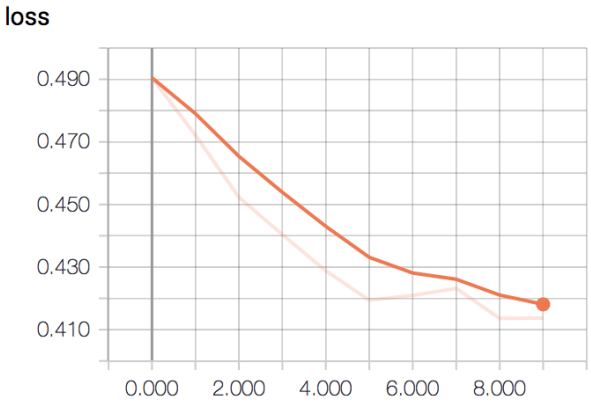

Figure 14: The training loss of Siamese model. Notice that accuracy for Siamese network is not necessary (we only compare two images in each step). We only train 10 epochs due to the limited time.

3.5. Evaluation

Due to the limitation of Kaggle's dataset, we cannot compute testing accuracy (need to submit results to get the score). To address this problem, we choose to train as much epochs and try to prevent overfitting with early stopping. In Table 2, we summarize the accuracy of our several different classification methods.

3.5.1 Prediction

According to our evaluation result, the performance of Siamese Network is not so bad. Therefore, we will discuss some results of Siamese Network in this section. Recall that the output of the classifier will produce the estimate $\hat{\mathbf{y}} = \left[\hat{\omega}\right]$. In detail, Fig. 16 shows the top 5

Model	Score on Kaggle
Baseline	0.032
Non-Separated AlexNet	0.08
VGGNet	0.16
Transfer learning(VGG19)	0.0016
Transfer learning(InceptionResV2)	0.03
SiasmeNet	**0.387**

Table 2: Score of different classification methods. Best score 0.387 get place 93 out of 354 teams.

prediction of each testing image by imposing a probability density on it.

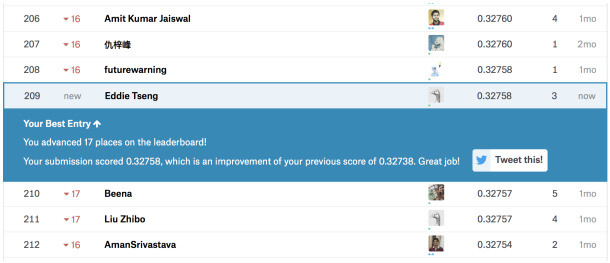

Figure 15: Score on Kaggle (Baseline)

Id	Image
w_d0b3293 w_ccbc782 w_5557280 w_e0cb9c5 w_fb2c3fb	c4822e88.jpg
w_17136dc w_414f402 w_a5ab9d1 w_2fe43c7 w_e54feba	7a81cd5d.jpg
w_4fd48e7 w_7de72c5 w_73cbacd w_5d0666e w_9d83b4d	4c7c5aaf.jpg
new_whale w_33973bf w_73b705e w_41ed8e8 w_4b87cba	dba9aa05.jpg
w_de5b417 w_5384c57 w_b9ee5ec w_733fe81 w_40323ac	5d97b3bf.jpg
new_whale w_8c605d2 w_2c68b75 w_eb0a6ed w_cbedcb0	ff179dea.jpg
w_7b035cc w_02facde w_715d152 w_21e178f new_whale	5c923031.jpg
w_71e6583 w_1eafe46 w_47d2bc6 new_whale w_93bf5f4	73e64479.jpg
w_b939998 w_4f248f3 new_whale w_0acce53 w_c1b685f	312207f4.jpg
w_b8f8e69 w_b942708 w_a18d0dc new_whale w_4114553	96ae4ce4.jpg
w_daeb296 w_8cd3036 w_8d83172 w_987a36f new_whale	b4077458.jpg
w_42f935c w_e42c624 w_3f91e04 w_373593e w_7028d77	60dd112e.jpg
w_9947243 w_73d5489 new_whale w_0e4f53c w_c4f3619	5be9b4cf.jpg
new_whale w_886257d w_187e235 w_3674103 w_eb0a6ed	9325ec9c.jpg

Figure 16: Baseline top 5 prediction

To address the difficulties in new whale class(actually it is not a class, just a lot of unknown whales in the same category), we tried as much methods as we can. However, the score is not high enough to satisfy our team. Thus, we definitely will try to get better result and score on it.

Repo: Github link

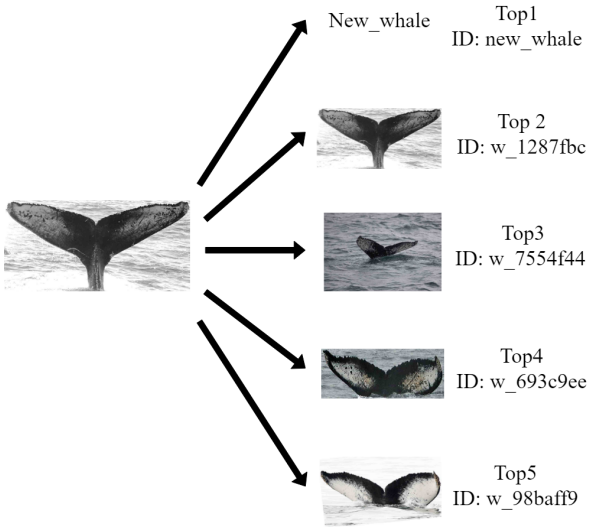

Figure 17: Example of Baseline Top-5

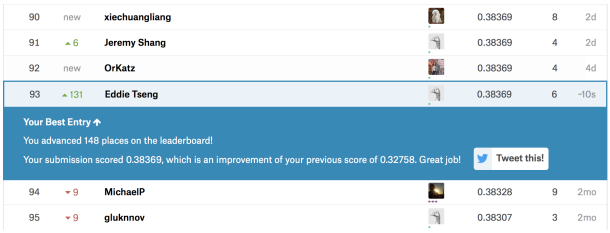

Figure 18: Score on Kaggle (Siamese Network)

Id	Image
w_c6372b8 new_whale w_dd0c2e0 new_whale w_797e546	00029b3a.jpg
w_fae2cb6 w_0b775c1 w_1772ed2 w_cb9220b w_2901dbf	0003c693.jpg
w_8b56cb1 w_a24f581 w_d78513b w_2173953 w_15f29b7	000bc353.jpg
w_a59905f w_3cf3853 w_4e68ddc w_414f402 w_cd1f1ed	0010a672.jpg
w_18eee6e w_2173953 w_2173953 w_bb2d34d w_1d53d9c	00119c3f.jpg
w_7554f44 w_569c2c7 w_883557a w_d984eb7 w_8f1aa27	001259cc.jpg
w_3d66298 new_whale w_09e0cbf w_d84e006 w_a494171	0015f9b4.jpg
w_cf24f84 w_62c3998 w_62c3998 w_d234e6b new_whale	0018c4ba.jpg
w_aa7a302 w_da0b8b5 w_ee948c6 w_6ab7b53 w_77d2aea	001bf484.jpg
w_20da4cf w_e07f3d1 w_22f0a7d w_38a3f72 w_ddf14ae	002d8d81.jpg
w_a25caa9 w_3674103 w_d984eb7 w_97da401 w_7554f44	00342aa2.jpg
w_c34afd7 w_883557a w_b588283 w_794effc w_7ed0d17	00367faf.jpg
w_9122179 w_f18df18 w_7b39dda w_bf38f05 w_8867074	003abee6.jpg
w_3e4bb3f w_540fd73 w_f5eb6c6 w_f5eb6c6 w_5919cf7	003ac3da.jpg

Figure 19: Siamese Network top 5 prediction

4. Conclusion and Difficulties

We have studied some sequential and non-sequential convolutional neural networks that encode contextual information to refine the inferences iteratively over the stages. Moreover, we have implemented six different CNN-based classifier from scratch by using both Tensorflow and Keras. This project not only gives us comprehensive understand-

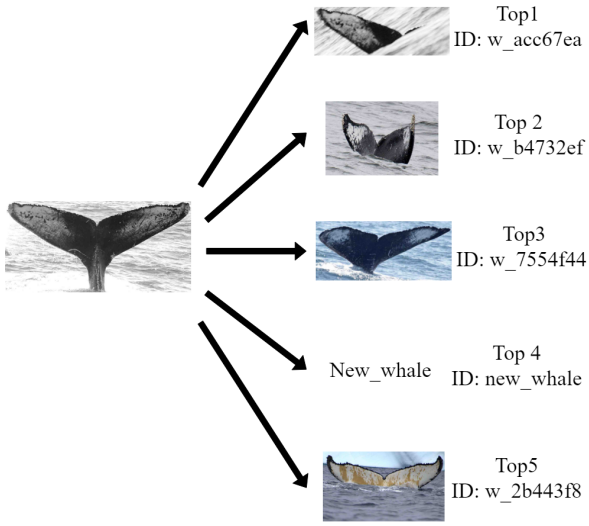

Figure 20: Example of Siamese Top-5

ing about image classification but also tells us the important of pre-processing dataset.

As for **difficulties**, being a rookie of machine learning, although Tensorflow and Keras are relatively easier for people to code then programming on Python or other programming languages. It took us **several days to get familiar with these packages**. Furthermore, we coded locally on our own **MacBook Pro**, so we use Tensorflow 1.8.0 with Keras 2.1.6 (these two version works together pretty fine). First, we made sure the code could run and then we ran on DSMLP. However, the campus platform has Tensorflow 1.4.0 with Keras 2.1.5, which has some **function mismatch**. For instance, "softmax" and "l2-normalization" function in Keras has property named 'axis' but Tensorflow named 'dim'. We asked TA, professor and ETS for help, they told us they are stuck at CUDA 8.0 for the remainder of this school year, as an update to CUDA 9 will require broad, coordinated upgrades of many other packages. Many such upgrades would introduce disruptive behavior or functionality changes. Thus, we need to change some function and debug for a while or **run the code on our own computer**(which took a long time for training).

Moreover, **imbalanced data** really frustrated us for training networks. We have tried a lot of methods to deal with it and score No.93 for now. Nonetheless, the results were not satisfied by us.

5. Future Work

Due to the limitation of time and resource, we haven't meet our goal. In the future, we will improve the performance of the prediction such as **bounding box** and **normalization** with hand and investigate novel architectures which could be potentially better tailored towards imbalanced data problems in general. Also, we will try to find out more efficient way for training our models.

References

[1] Combat Imbalanced Classes in Your Machine Learning Dataset. https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes/in-your-machine-learning-dataset/. Accessed: 2018-05-12.

[2] Humpback Whale Identification Challenge. https://www.kaggle.com/c/whale-categorization-playground. Accessed: 2018-05-02.

[3] One Shot Learning and Siamese Networks. https://sorenbouma.github.io/blog/oneshot/. Accessed: 2018-05-20.

[4] Why You Should Use Cross-Entropy Error Instead Of Classification Error Or Mean Squared Error For Neural Network Classifier Training. https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy/error-instead-of-classification-error/or-mean-squared-error-for-neural/network-classifier-training/. Accessed: 2018-05-02.

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[6] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(6):610–621, Nov 1973.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional

neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, pages 950–957, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[9] M. Pal and P. M. Mather. Support vector machines for classification in remote sensing. *International Journal of Remote Sensing*, 26(5):1007–1011, 2005.

[10] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[12] Z. Sun, Q. Song, X. Zhu, H. Sun, B. Xu, and Y. Zhou. A novel ensemble method for classifying imbalanced data. *Pattern Recognition*, 48(5):1623 – 1637, 2015.

[13] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.