**Humpback Whale Identification Challenge**

Yingkai Jiang

Yichao Li

Yunshu Zhang

1. **Introduction**

This project comes from a Kaggle competition of the same name, aiming to identify a whale by the picture of its fluke (see figure 1). This competition began 3 months ago and will be close 3 months later. Here are more details.

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.



Figure 1: An example of fluke of a whale

In this competition, we are challenged to build an image processing algorithm to identify whale species in the image.

1. **Data description**

The training data contains 9850 images of humpback whale flukes. Individual whales have been identified by researchers and given an Id. The challenge 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.

The testing data contains 15,610 images of un-labelled humpback whale flukes.

All of the data is provided by Happy Whale[1].

Since it is very difficult to recognize the exact whale, the submission is evaluated according to the mean average precision @5, which is defined as:

where is the binary value recording whether the i-th cutoff of the k-th test image predicts right.

1. **Exploratory data analysis and data pre-processing**
2. Distribution of categories

First, let’s check the distribution of labels in training set:

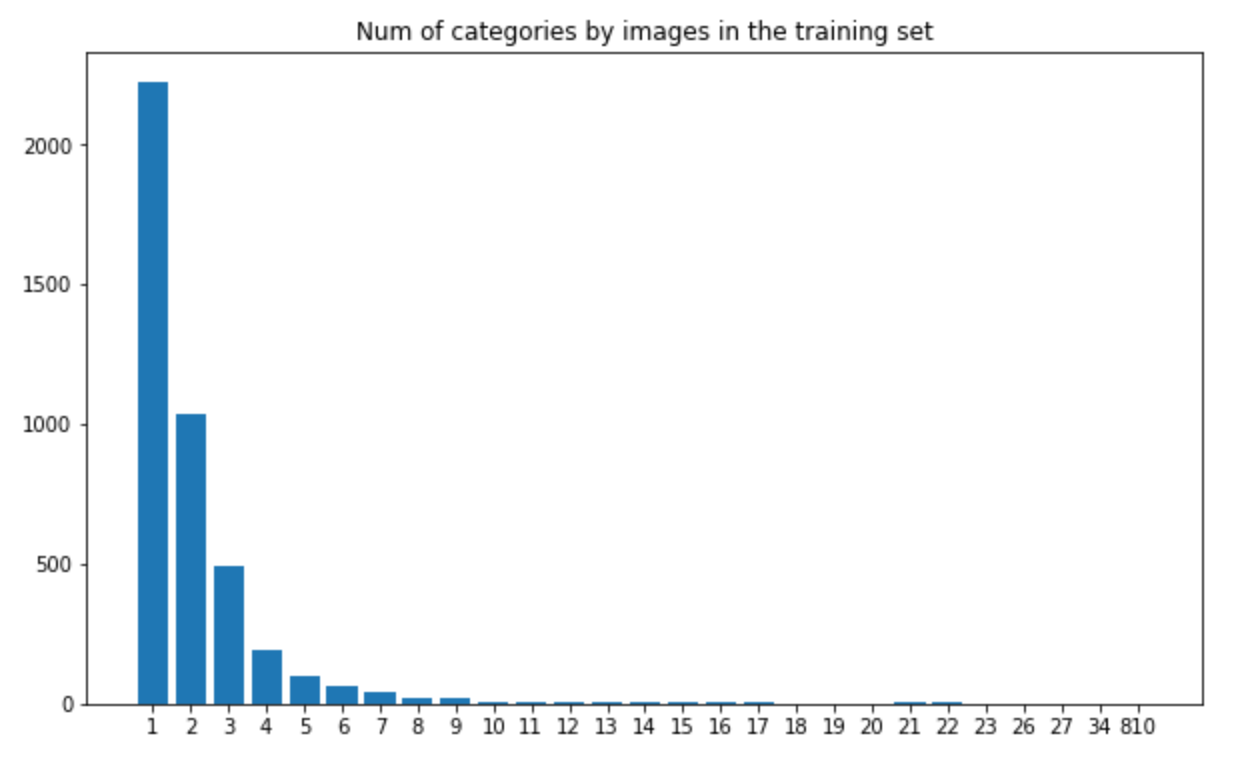


Figure 2: Number of categories by images in the training set

The above figure illustrates that the distribution of all 4251 categories are highly unbalanced. More than 50% classes have only one sample, which makes this problem very difficult.

Next, we can reverse the table to see number of images of top-10 classes:

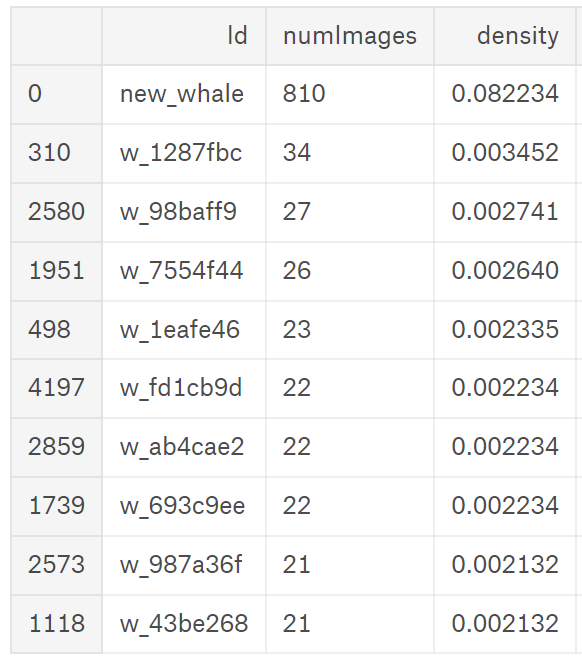


Figure 3: Number of images of top-10 classes

As we can see, 8% whales are in the category of *new\_whale*, which contains much more images than other classes. This phenomenon leads to a strategy: we can always guess the test image as *new\_whale*, and remove all the *new\_whale* in training set. The reason is that guessing image as *new\_whale* is very likely to be correct if test set and training set have similar distribution of images. Our experiment also shows the effectiveness of this strategy. Although it is a little tricky and violates the research goal in some sense, the strategy indeed improves our result.

1. Standardization of images

Let’s see some samples in the training set:



Figure 4: training samples

We can easily find our first question: the images have different sizes. We calculate the image size frequencies (we omit the sizes whose frequency equals to one):

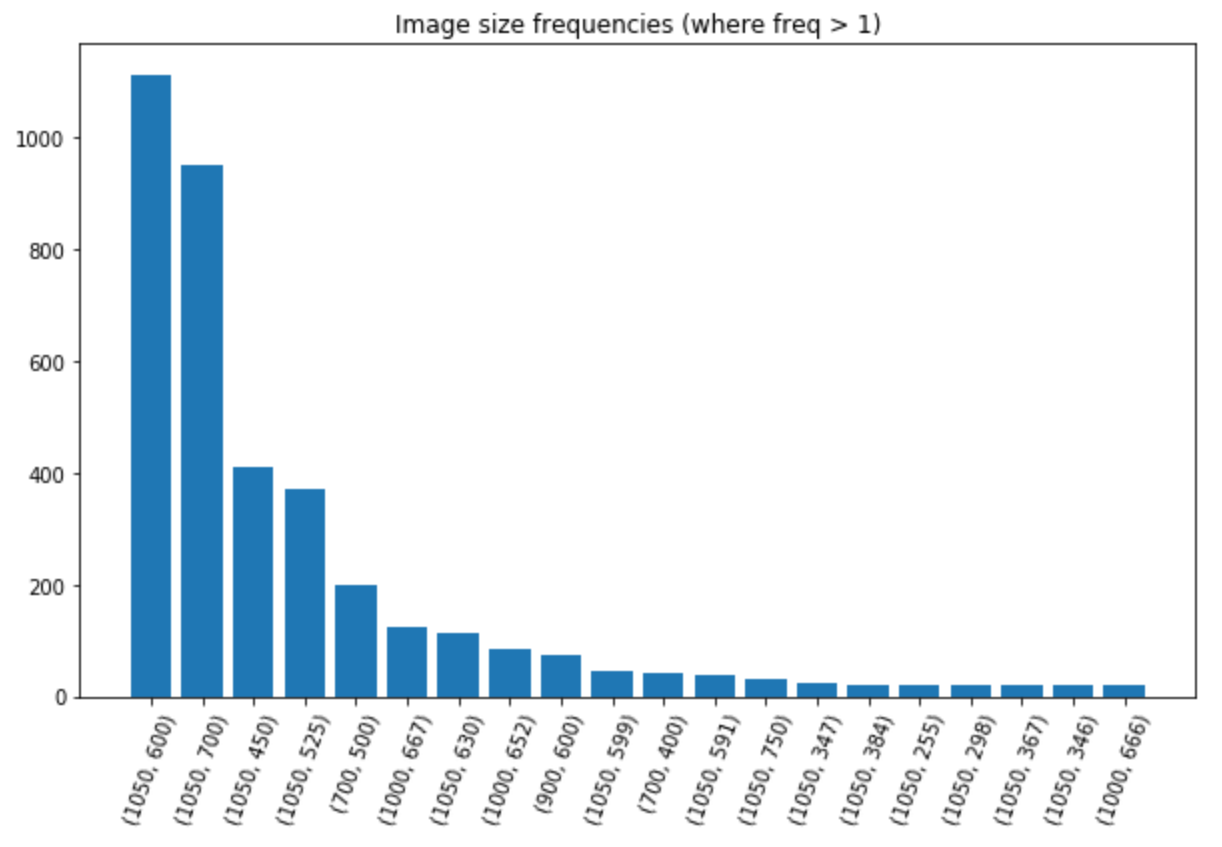


Figure 5: Image size frequencies (where frequency > 1)

It is impossible to import the data into CNN directly since the number of neurons is fixed. As a result, we use the *resize* function in OpenCV in python to change the image sizes to 32\*32. The algorithm of this function is bilinear interpolation. We also tried 64\*64, but the result does not improve a lot. In fact, due to the computational limitation, the result even gets worse. We will write it in detail in part 5. To sum up, we choose the size as 32\*32.

The second question is also easy to find: some images are black and white, while others are colorful. It also means their sizes are different. The proportion of greyscale images is 48.8%. We can change RGB images to greyscale images using a simple formula:

The bad thing is that we lose lots of information during this resizing process.

After all, the images are in same size now.

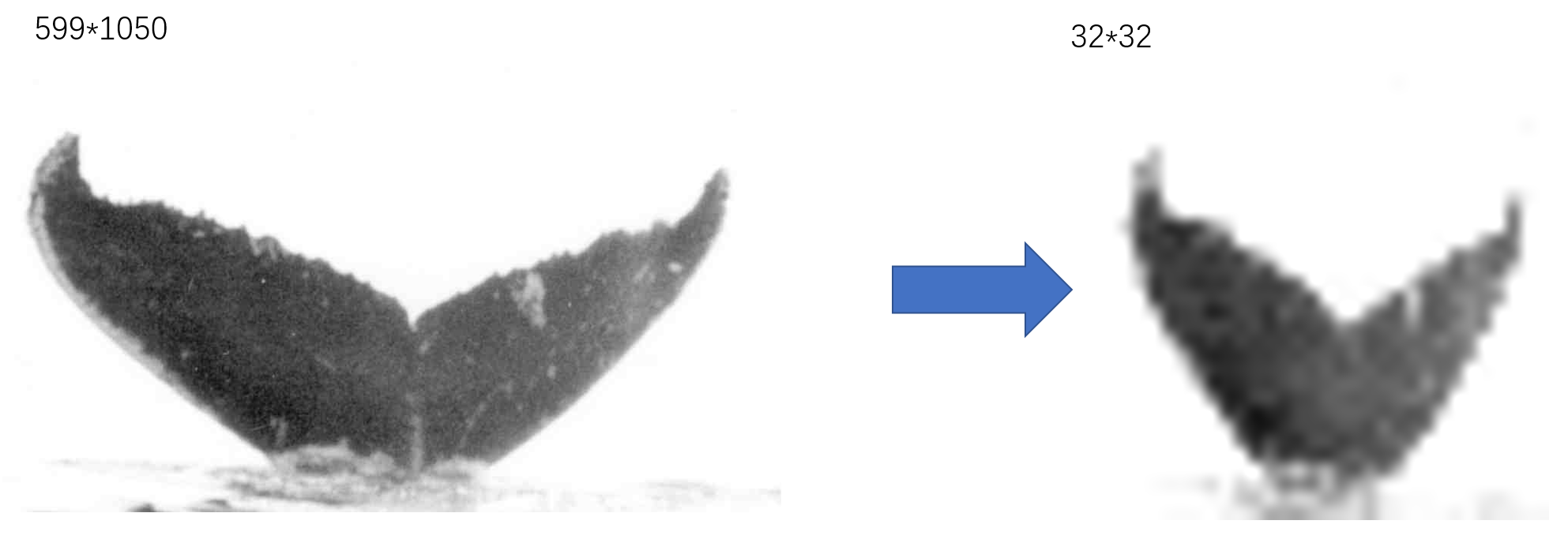


Figure 6: Standardization of images

1. Image augmentation

As we have shown, many classes have only one or two samples, which makes it difficult to train machine learning models. For example, it is impossible to build a validation set. What’s more, the extremely small sample size will lead to serve overfitting problem. For example, if a class A whale has a fluke towards down, while other whales with their flukes up. Then we can figure out class A whale using the direction information, which is actually not the characteristic of other class A whales. Thus, it is necessary to do data augmentation.

Fortunately, we can directly use data augmentation function *ImageDataGenerator* in *keras*. Here we include some of the augmentation methods:

1. Rescale: We rescale the data range into 0-1 to simplify our further calculation. Consider that the maximum of greyscale data is 255, we choose the rescale factor as 1/255.
2. Rotation: We randomly rotate some angle between -40 degrees and 40 degrees.
3. Shift: We randomly shift the width and height of the image, with the fraction equals to 15%.
4. Flip: We allow randomly flip of the image horizontally.
5. Zoom: We randomly zoom the pictures in some given range.

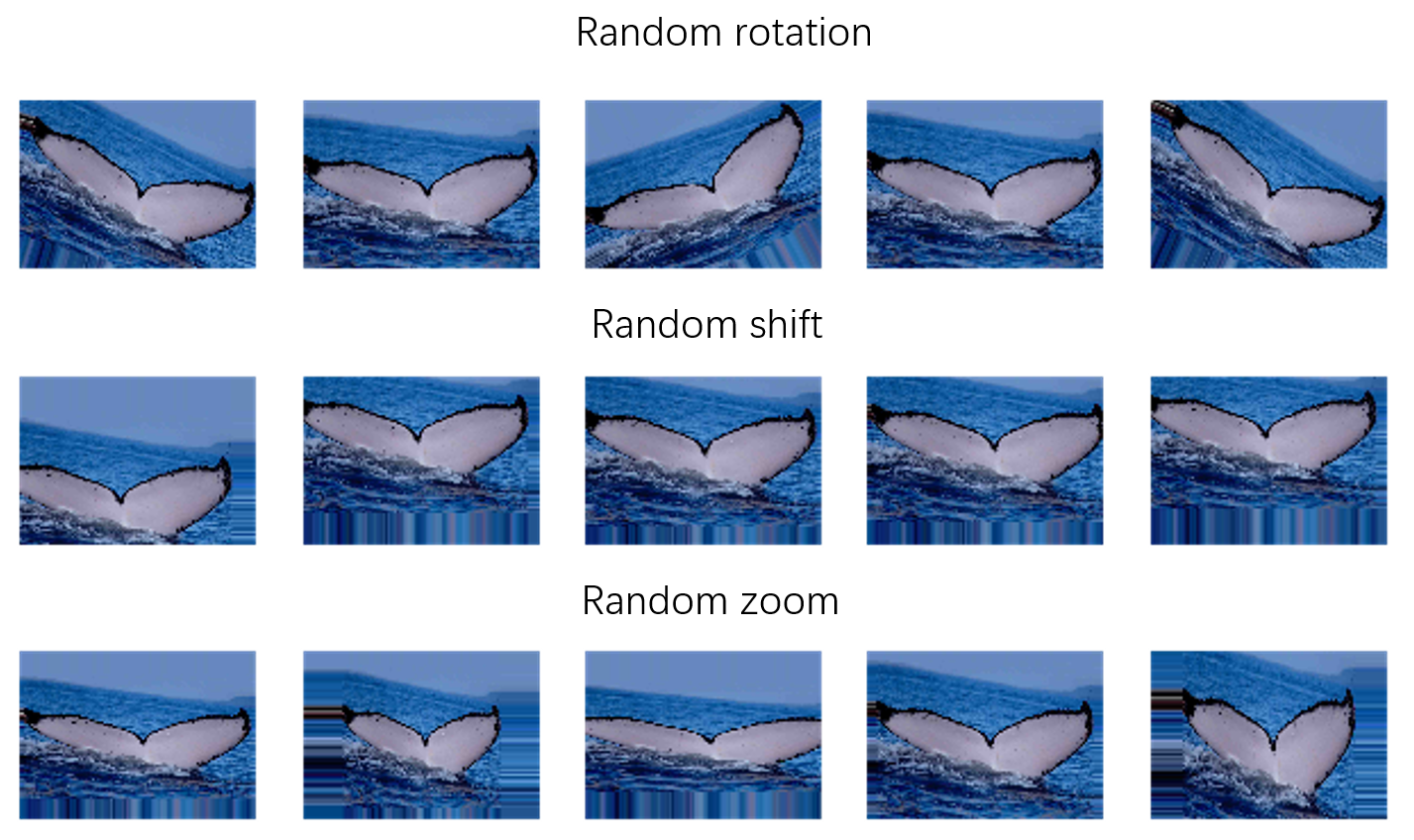


Figure 7: Data augmentation process

Each time we fetch a batch of data, we apply data augmentation function on them. As a result, we can increase the sample size to make our machine learning algorithm possible. Here are a batch of images after augmentation:

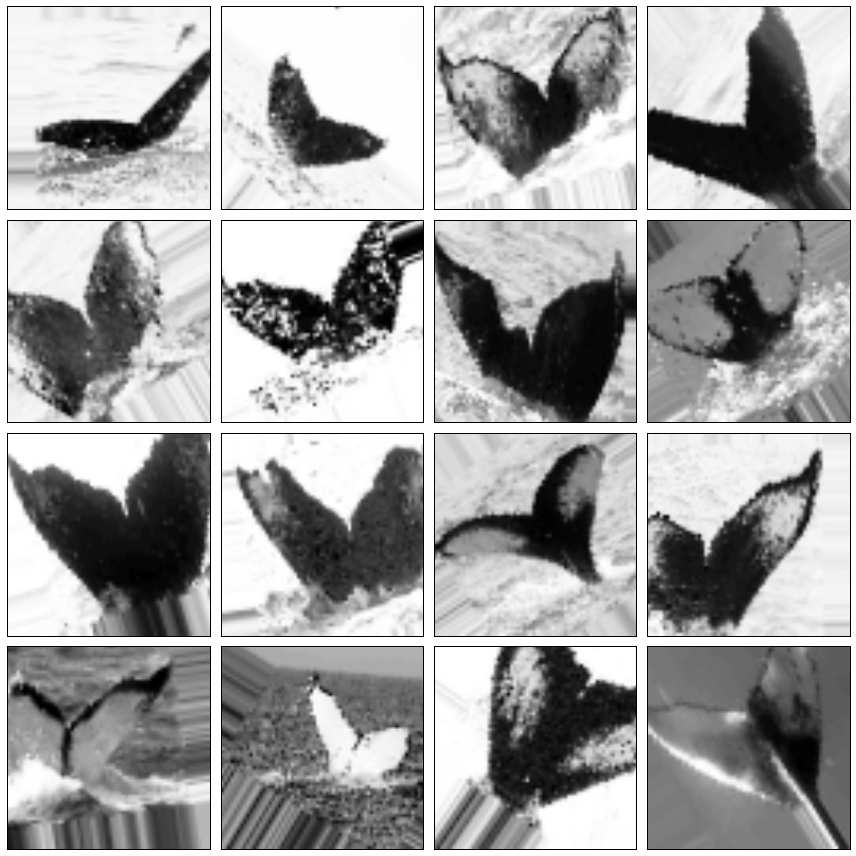


Figure 8: Images after augmentation

1. **Machine learning model**

Since convolutional neural network (CNN) has great power in computer vision, we use a 10-layer CNN as our machine learning model. The following figure shows the framework of our model.

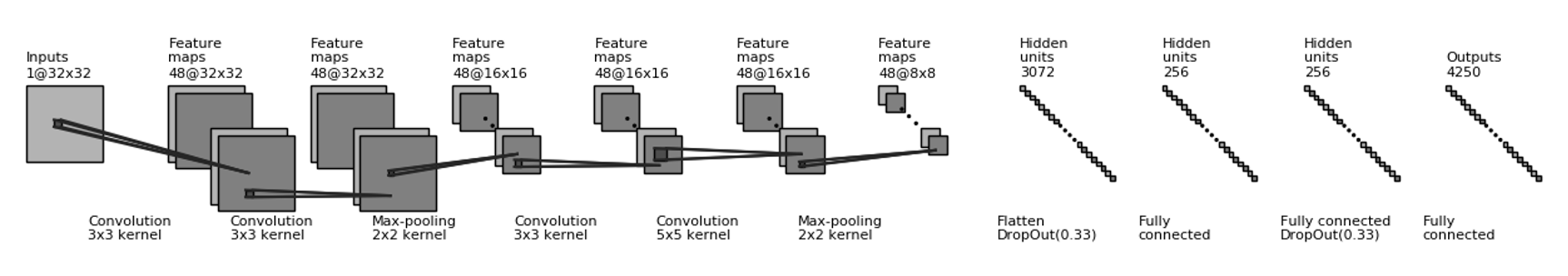


Figure 9: Model architecture

We firstly use two convolutional layers, each has 48 convolutional 3\*3 kernels. Then we add a max-pooling kernel with size equals to 2\*2. After that, we repeat a similar procedure, changing the size of second convolutional kernel to 5\*5. Till now, the dimension changes from 32\*32 to 8\*8 due to our max-pooling layers.

Then, we flatten all the square data into 8\*8\*48=3072 units. Since there are so many units, we add a dropout procedure before we flatten the data in case the number of parameters explodes. We set the drop out rate equals to 0.33.

After we flatten the data, we add three fully connected layers with 256, 256, 4250 units. We also add a dropout procedure after the first fully connected layer to decrease the number of parameters. In the end, we use softmax procedure to get probabilities on each category.

For active function, we use ReLu function for most cases and Softmax function at the end. Vast experiments tell us ReLu function has good performance, and Softmax can transform data into output probabilities.





We choose batch size as 128 and epochs as 40. What’s more, we use AdaDelta algorithm to optimize our parameter. When doing convolution and max-pooling, we use a padding with zero around the square data.

For loss function, we firstly use categorical cross-entropy: , where is the binary variable which denotes the category of -th training image, is the softmax output of model.

In the categorical cross-entropy loss function, each category is not equal weighted. Those higher frequency category in training set influence the loss function more. Hence, we use *weighted categorical cross-entropy*. If , where is the frequency of category , the weighted categorical crossentropy is most fair for each category. In practice, we set .

1. **Results**

Since we don’t have specific answer for test dataset, we will mainly show some prediction results of training samples. For test dataset, we can only upload our code to get an overall test score for the whole dataset. We will show this information in the following results.

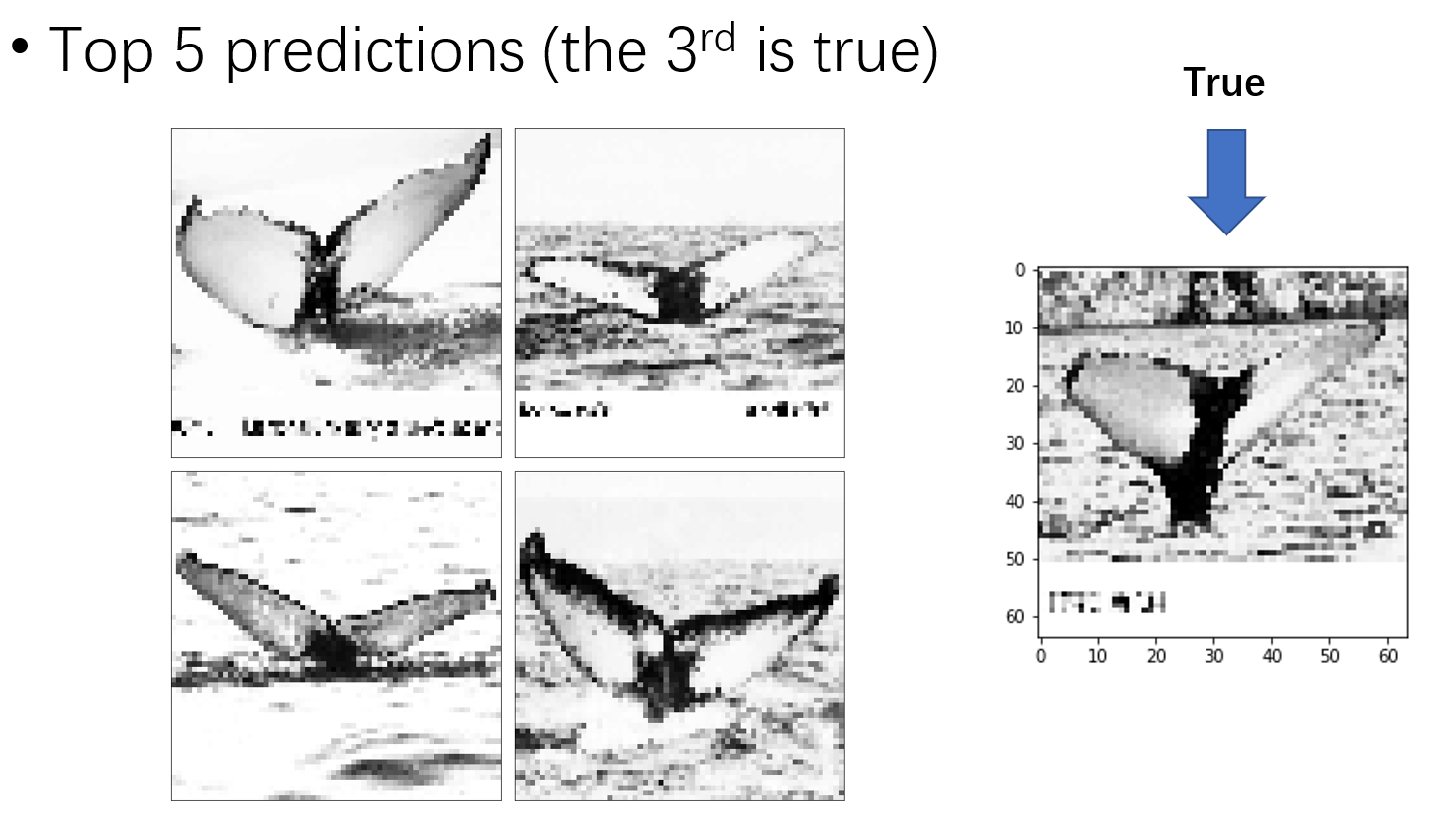


Figure 10: Prediction result of a training sample

We apply our CNN model on a training sample, and we find a picture for each predicted category for representation. The third prediction result matches with the true answer, which is not bad in consideration of our MAP@5 criterion. We also notice that the five predictions look very similar, verifying the difficulty of our problem.

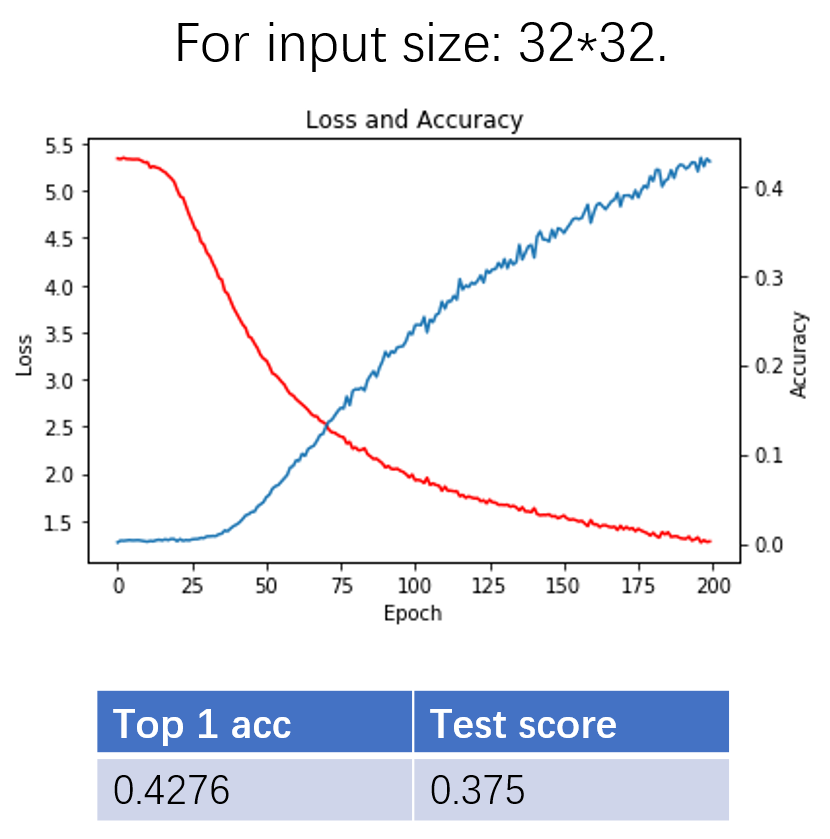


Figure 11: Prediction result for input size 32\*32

Figure 11 shows our prediction result for input size 32\*32. As our training process goes, the loss function decreases and accuracy increases. After 200 epochs, we have our loss to be 1.3 and top 1 accuracy, which records whether the true category is the first prediction, to be 0.4276. This is a good performance on whole training set, since the baseline is 1/4250. As for test set, the test score is 0.375, which is rank 79 on the leaderboard.

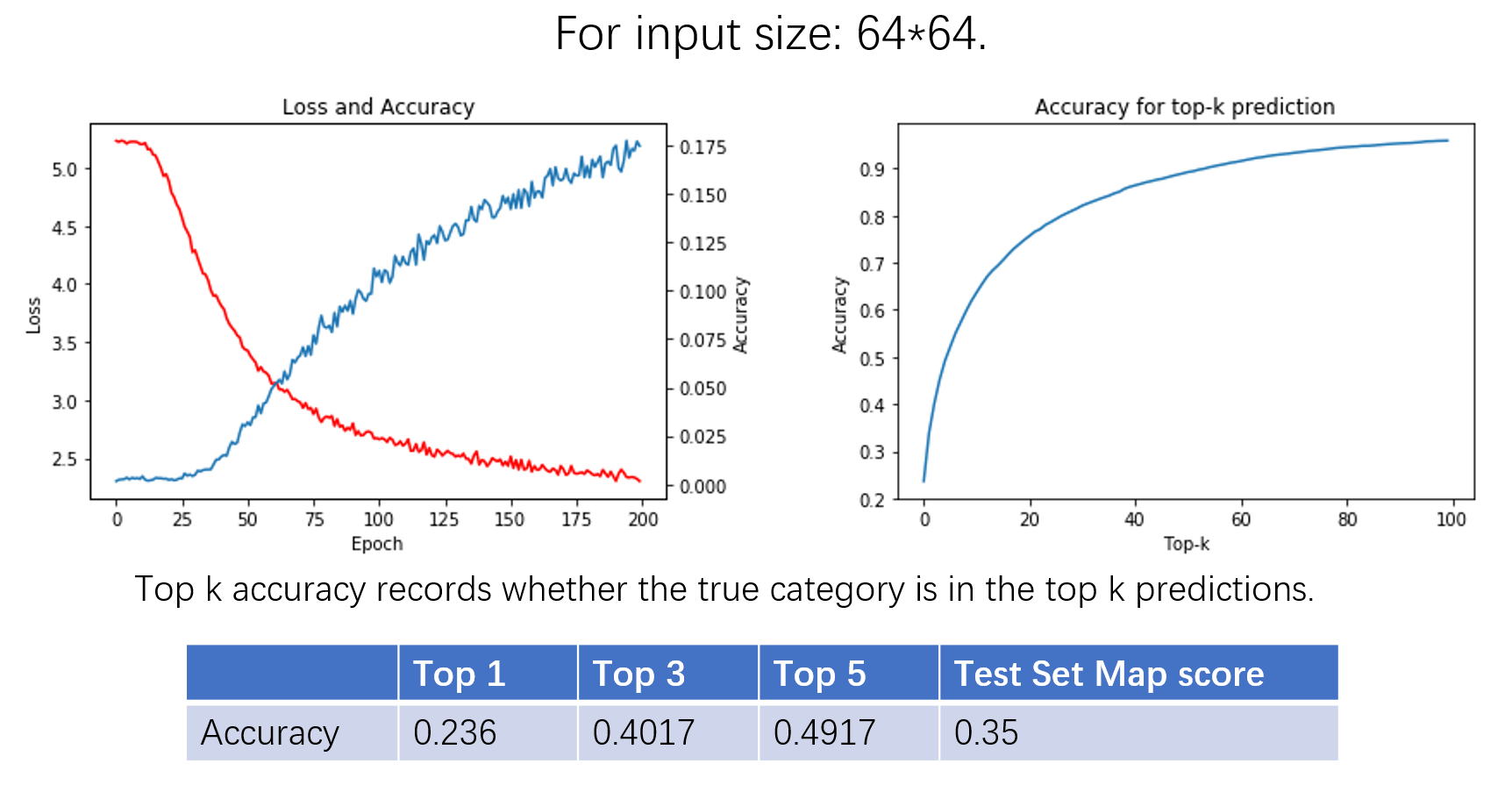


Figure 12: Prediction result for input size 64\*64

Compared to the previous result of 32\*32, the accuracy drops significantly. This does not make sense because larger image should have more information than smaller image. We guess the reason is that the model does not converge after our training. From the plot, it seems that the loss function can continually become smaller. If we have more computation resource, we may let the 64\*64 result exceed the previous one.

Besides top 1 accuracy, we also calculate top k accuracy for k from 1 to 100. As we can see, the top 5 accuracy has achieved about 0.5. For k equals to 100, the top k accuracy exceeds 0.9 and is close to 1. In consideration that the total number of categories is 4250, the result is satisfying.

1. **Discussion**

There are large amount of “new\_whale” in both training and test set. We use a “cheating” strategy when working on this problem, which is omitting all the *new\_whales*. However, *new\_whale* don’t need to be “new” whale. It is possible to be an existed whale but not distinguished.

We have ever tried other machine leaning frameworks. However, we failed to try “ResNet50” because of the limitation of computation ability. We may continue to work on other models if we have enough computation resources.

We didn’t use cross validation, because in the training set, there are very few images for large part of categories. Although using data augmentation method, we can apply cross validation in our model. However, we think this will hardly improve our result.

1. **Reference**

[1] <https://happywhale.com/home>

[2] <https://www.kaggle.com/c/whale-categorization-playground#description>

[3]<https://www.kaggle.com/lextoumbourou/humpback-whale-id-data-and-aug-exploration>

[4] https://www.kaggle.com/gimunu/data-augmentation-with-keras-into-cnn/notebook

[5] https://keras.io/preprocessing/image/