Project Title: Humpback Whale identification

Bolin Chen, Siqi Wu, Yanming Lai

Github: https://github.com/bobbyfine/Humpback-Whale-identification-

1. Introduction and Problem Statement

In the past 40 years, scientists who studied on whales had to manually match whale individuals with the photos they had gotten in order to monitor the size, health of humpback whale groups. However, this was relatively inefficient for the process of building up a thoroughly global database. Therefore, it is inevitable to develop an advanced technique to identify whale individuals with their tale photos. The effective solution to this issue will be training convolutional neural networks with multiple layers to let models categorize photos. We actually not only trained neural networks but also compared different models that were applicable to identify images. With a limited amount of training data, we were able to identify some whales with the test whale tail images.

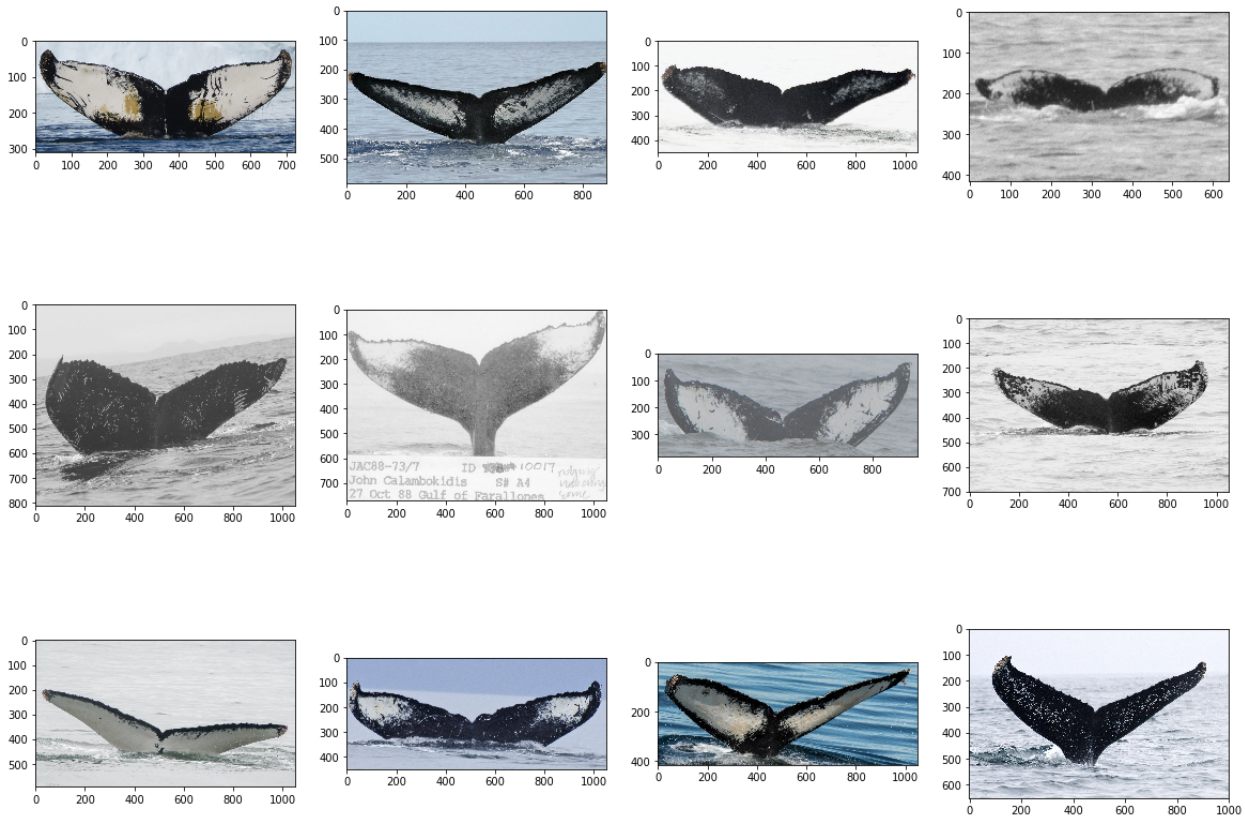
2. Related Work:

Some groups on kaggle leaderboard of whale identification competition shared their thinking process and what they actually implemented to gain a relatively high identification accuracy. Two out of the top five groups applied Siamese Neural Network which is a class of neural network architectures that contain two or more identical subnetworks. We built up a simple Siamese Neural Network to compete against our major model.

Siamese Neural Networks: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

3. Data Sets

Data sets are composed of images of whale tails. Images are different in color. Some are gray scaled and some are in colors. Images also come with different sizes. Even though they are all in triangular shapes, they are not in the same ratios. A few images are compressed which lead to hard recognition. Moreover, the shapes of the whale tails are irregular, since some tails are side view whale tails. Below is an example of 12 training images.



Data: https://www.kaggle.com/c/humpback-whale-identification/data

However, there is one more thing we need to care about. The labels corresponding to some whale tail images are “new_whale”. About 40% of the images have this label. These images are omitted from the training dataset because they do not help models identify which whale it is.

```
In [5]: train['Id'].value_counts()[:4]
```

```
Out[5]: new_whale    9664
        w_23a388d      73
        w_9b5109b      65
        w_9c506f6      62
        Name: Id, dtype: int64
```

In order to train the model efficiently, we only select the top 28 whales that have more than 30 corresponding images after we get rid of the “new_whale” labeled images. Below is the frequency of different labels. With a limited amount of computational power, we have to remove some low-priority images.

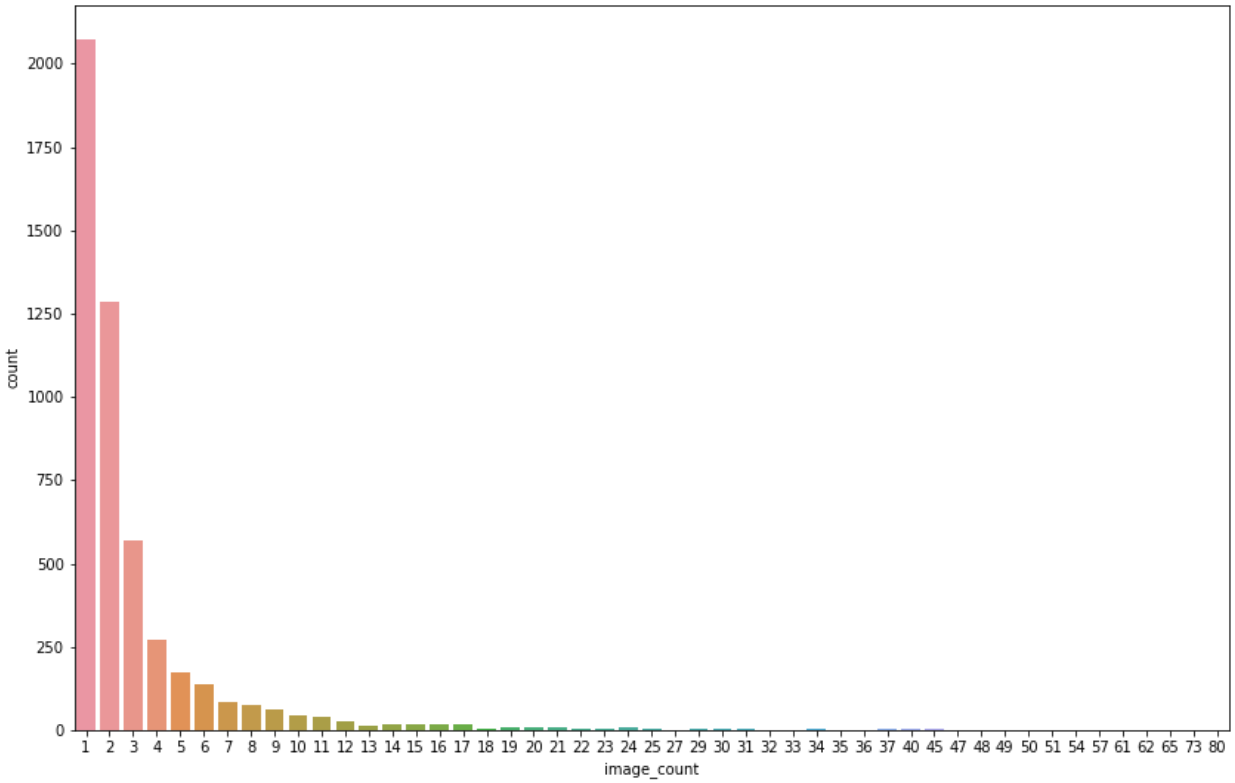


Image source: https://www.kaggle.com/kretes/eda-distributions-images-and-no-duplicates

4. Description of Technical Approach

The core of our project contains three main sections: data preprocessing, convolutional neural network models, and model ensembling.

1. Data preprocessing
   The images in the dataset do not have a uniform size so we pad each images to have equal length and width, then we resize every image to 128 by 128 pixels. As shown below, we use the "get_pad_width" function to get the proper pad size which is then passed to the "np.pad" function. The "cv2.resize" function completes the task by resizing the image to a desired size.

```
img = cv2.imread('./' + dataset + '/' + image_path )

pad_width = get_pad_width(img, max(img.shape))
padded = np.pad(img, pad_width=pad_width, mode='constant', constant_values=0)

resized = cv2.resize(padded, (desired_size,)*2).astype('uint8')

return resized
```

2. Building convolutional neural network models (CNN)
   Since whale identification is a classic image classification problem, we chose CNN over other classification methods from the very beginning. Through a series of experiments,

which will be explained in section 5, we built four CNN models that produce fairly nice results. Here is a snapshot of one of our models.

```
model_base_1 = nn.Sequential (
    nn.Conv2d(3,32, kernel_size = 3),
    nn.LeakyReLU(inplace = True),
    nn.BatchNorm2d(32),
    nn.MaxPool2d(kernel_size=2, stride = 2),

    nn.Conv2d(32,32, kernel_size = 3),
    nn.LeakyReLU(inplace = True),
    nn.BatchNorm2d(32),
    nn.MaxPool2d(kernel_size=2, stride = 2),

    nn.Conv2d(32,64, kernel_size = 3),
    nn.LeakyReLU(inplace = True),
    nn.BatchNorm2d(64),
    nn.MaxPool2d(kernel_size=2, stride = 2),

    nn.Conv2d(64,128, kernel_size = 3),
    nn.LeakyReLU(inplace = True),
    nn.BatchNorm2d(128),
    nn.MaxPool2d(kernel_size=4, stride = 4),

    nn.Dropout(0.20),

    Flatten(),
    nn.Linear(1152,128),
    nn.ReLU(inplace = True),
    nn.Dropout(0.3),
    nn.Linear(128,132)
)
```

This model has 4 Conv2d layers each followed by a pooling layers and 2 fully connected layers at the end to produce classification results. For activation function, we used LeakyReLU to punish incorrect classification. We also used the “dropout” function to prevent overfitting. The other models are similar with this one except that they have different combinations of layer count, dropout rate, kernel_size and channel count.

3. Model Ensembling

With four CNN models, our last step is to ensemble them to further improve accuracy. As shown in the image below, we generate a set of predictions for each model and simulate a majority voting. In another word, we find the most popular label prediction among four models for each image and return that label as final classification.

```
num_correct = 0
num_samples = 0
model_1.eval() # Put the model in test mode (the opposite of model.train(), essentially)
model_2.eval()
model_3.eval()
model_4.eval()
for x, y in validation_generator:
    x_var = Variable(x.type(gpu_dtype), volatile=True)

    scores_1 = model_1(x_var)
    scores_2 = model_2(x_var)
    scores_3 = model_3(x_var)
    scores_4 = model_4(x_var)

    _, preds_1 = scores_1.data.cpu().max(1)
    _, preds_2 = scores_2.data.cpu().max(1)
    _, preds_3 = scores_3.data.cpu().max(1)
    _, preds_4 = scores_4.data.cpu().max(1)

    preds = preds_1
    for i in range(list(preds_1.shape)[0]):
      preds[i] = np.argmax(np.bincount([preds_1[i],preds_2[i],preds_3[i],preds_4[i]]))

    num_correct += (preds == y).sum()
    num_samples += preds.size(0)
acc = float(num_correct) / num_samples
print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 * acc))
```

4. Siamese Neural Network (experimental, not included in the final submission)
   Base on the Siamese Neural Network paper, we implemented a simple Siamese NN model. In this model, we replaced the ReLU activation function with a sigmoid activation function. According to the paper, such type of neural network should work well when the data set contains a lot of classes with very few data points. However, our model did not perform well on our current data set. Detailed results can be found in section 5.

5. Software

A. For the code itself, we used:

a. Jupyter Notebook, a web-based interactive computational environment for creating Jupyter notebooks documents. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

b. Colab, is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

c. Python, an interpreted, high-level, general-purpose programming language.

d. PyTorch, an open-source machine learning library for Python, based on Torch. It is primarily developed by Facebook's artificial-intelligence research group.

PyTorch provides two high-level features in this project:

- Tensor computation (like NumPy) with strong GPU acceleration
- Deep neural networks built on a tape-based autodiff system

e. Scipy, is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In this project, we primarily use Numpy as dapa process.

E. Code from other people that we used:

e. As we discussed above, the data set contains more than 20000 images with all different colors and shapes. We need to preprocess the data. We get the idea from a repository in Kaggle that shows how to reshape the image into same shape and same resolution without losing the Information. (https://www.kaggle.com/xhlulu/exploration-and-preprocessing-for-keras-224x224)

6. Experiments and Evaluation

In the process of prepare dataset and divide the dataset into train and evaluation, we noticed that the dataset is very biased in terms of diversity and quality. For example, as we discussed above, in total of 25361 images, the label "new_whale" has frequency 9664 times. This label means the corresponding image has no label which means cannot be classified.

```
In [4]: label_df['Id'].describe()

Out[4]: count         25361
        unique         5005
        top       new_whale
        freq           9664
        Name: Id, dtype: object
```

So in our project, we decide to delete all data which labeled "new_whale" and use top 28 labels considering the computing resources we can use. Since we only have limited sources in Google Colab, if we load all images with labels, the virtual machine will crash due to RAM limitation.

The method we use to evaluate our model is *cross-validation*. We divide the processed data into training set with 1000 and validation set with 235. We use "check_accuracy" function from assignment 3 to test the accuracy of our model.

```
[ ]  label_df = label_df[label_df['Id'] != "new_whale"]
     count = dict()
     for index, row in label_df.iterrows():
       if row['Id'] not in count.keys():
         count[row['Id']] = 1
       else:
         count[row['Id']] += 1


     good_ids = []
     for id, c in count.items():
       if c >= 31:
         good_ids.append(id)
```

Result:

Model	Epochs	Max Pooling	Random Dropping	Result
3-Layers Convolutional Neural Networks	25	Yes	No	Got 95 / 235 correct (40.43)
3-Layers Convolutional Neural Networks	100	Yes	No	Got 112 / 235 correct (47.66)
Simple Simulation of Siamese Neural Networks	80	Yes	Yes	Got 94 / 235 correct (40.00)
Ensemble of four CNN models	50 (each model)	Yes	Yes	Got 152/235 correct(64.88)

7. Discussion and Conclusion

As we were working through this project, we had a clear feeling that the whale identification problem is very different from the homework practices. Specifically, the enormous amount of class labels and the lack of training images for majority of the labels made us realize that it requires knowledge and skills beyond the scope of this course to fully solve the problem.

However, we were not discouraged from the obstacles we faced. One the one hand, we had the insight that real life issues are full of uncertainties and irregularities that are missing in our regular homework. To get prepared for solving these more challenging problems, we learned many useful techniques relevant to image classification such as the advanced siamese neural network method. On the other hand, we were glad that we can apply the knowledge we gained in this course to practice. After simplifying the original problem, we were able to achieve a satisfying result with basic cnn models.

The major limitation of current approach is that the accuracy decrease significantly as we increase the number of classes to classify into. As a comparison, with 28 classes of 1235 images, our best model had a accuracy of nearly 65%. But the accuracy dropped to less than 50% when we increase the dataset to 132 classes of 3207 images. As a continuation of this project, we will try to use data augmentation to create more useful training data.