

**Contents:**

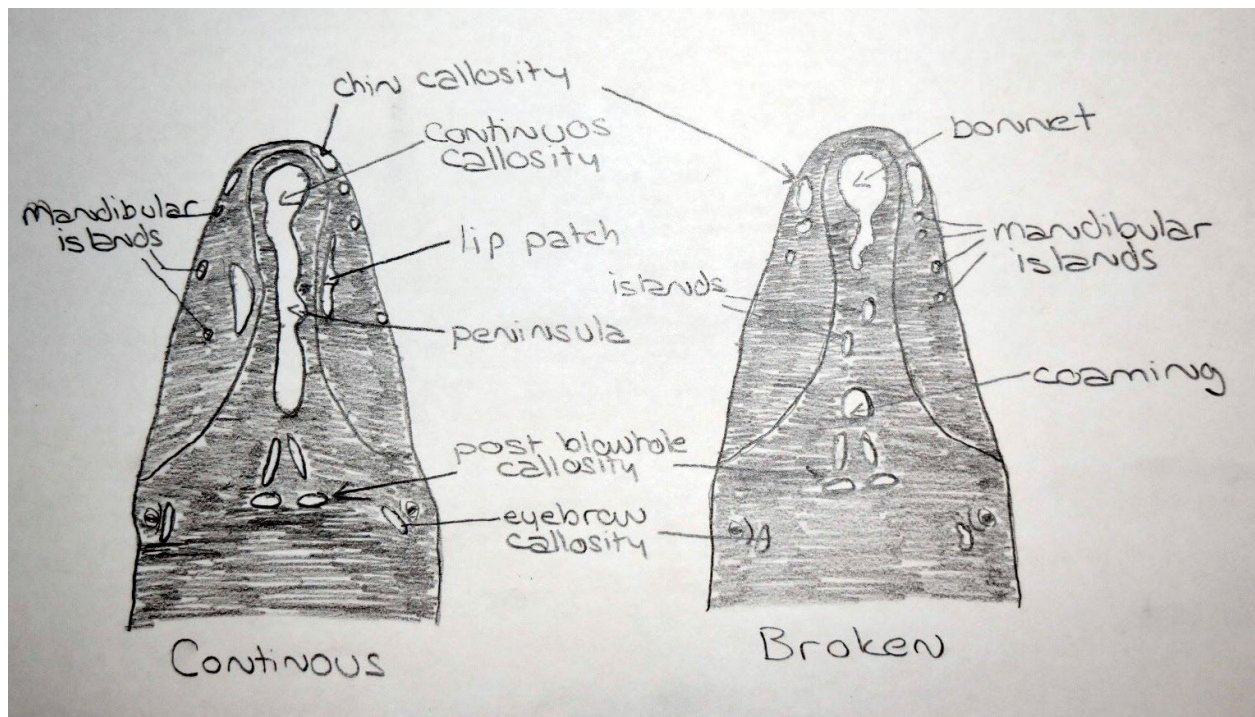
- 1. Introduction**
- 2. Background**
- 3. Data Preparation**
- 4. CNN model**
- 5. Kaggle Submission**
- 6. Conclusion And Future Improvements**
- 7. References**

## **1. Introduction**

In this assignment, I need to try to recognize the right whales by their images which were taken during different aerial surveys and participate in Kaggle competitions. The dataset of right whales was distributed on the Kaggle website by National Oceanic and Atmospheric Administration. Because of the activities of people, more and more animals die in our world. Some species of animals simply disappear from the face of the earth, and some are already on the verge of extinction. Right whales are also on the verge of extinction, and there are less than 500 of them left on our land. In order to monitor the health of each individual and protect them from extinction, and to facilitate this for scientists, it was decided to try to use Artificial Intelligence for this purpose.

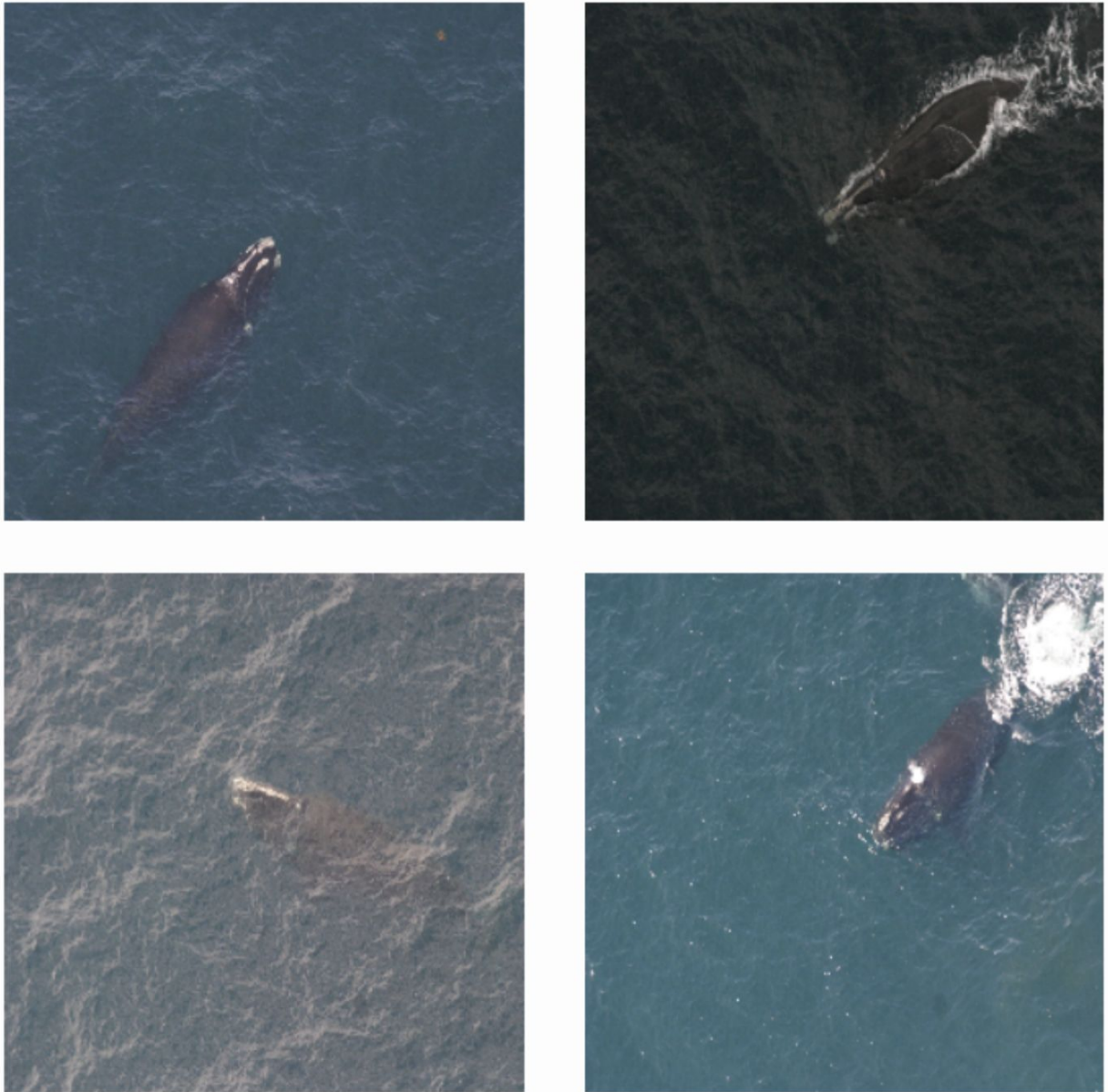
## **2. Background**

As I mentioned, the Right Whale is an endangered species that has decreased a hundred times over 100 years. In order to ensure their safety and preserve this species, scientists constantly put a lot of effort into this and track them across the ocean to understand their behavior and monitor their health. This process, of course, is carried out manually and this process is quite laborious and resource-intensive. This process begins with the fact that scientists photograph these whales, then process them, and, finally, the image data is compared with all the whales in the catalog developed by the researchers. Comparison takes place over the head as each right whale has its own unique pattern on the head(Image 1). The aim of the competition was to develop an artificial intelligence algorithm to automate the process of identifying whales from images, which will greatly facilitate the work of scientists and, in some cases, will avoid errors.



**Image 1. Patterns On Right Whale Head**

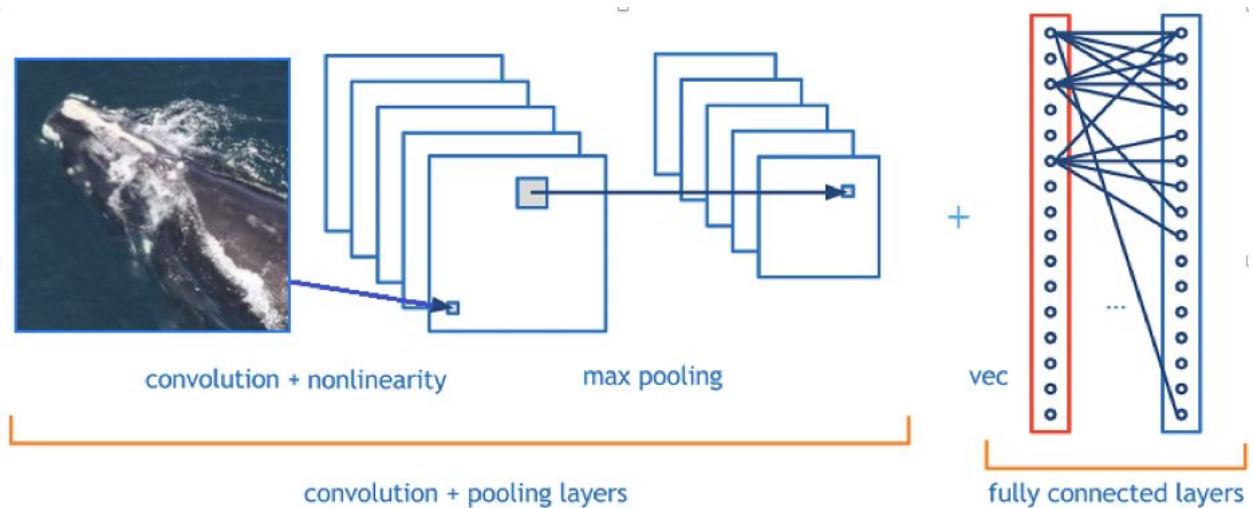
Images of whales were shot from a height and the quality of photographs always varies, due to the fact that they were shot at different years, different times of day, with different weather conditions and of course with different camera equipment. On Image 2 we can see sample pictures of whales from our dataset. As you can see all pictures are taken on different conditions, and it is very hard to identify them by just looking at these pictures.



**Image 2. Sample pictures of whales from dataset**

Recognizing the whales from these images is a very difficult task and artificial intelligence can help us in recognizing these whales. Convolutional Neural Networks is already well used in recognizing people's faces, in recognizing different types of images. I will try to use Convolutional Neural Networks in whale recognition. The main benefit of using convolutional neural networks is that they can focus on one specific part of the image because of dividing an image into several parts. The convolutional neural network consists of several layers, each

layer responsible for different tasks. First, in the input we have an image, in our case its whale image, then we have several convolutional layers and pooling layers that go in sequence. Pooling layers decrease the number of features and decrease dimensionality which allows our network to focus on smaller parts. All these convolutional layers and pooling layers at the end connects to a fully connected neural network with hidden layers and at the end of the network output layer with classifier.



**Image 3. Convolutional Neural Networks basic architecture**

### **3. Data Preparation**

The database consists of more than 4 thousand labeled images of 447 unique whales. The resolution of the raw images is very large and the biggest part of the images was taken by water. These water basically on all images creates a lot of noise, because the water splashes near the whale. (Image 4)





**Image 4. Sample image of whale**

Let's dive into code. Mainly I used raw python and basic python libraries for doing some data analysis and preprocessing. (Image 5) I am using the 'os' library for reading and manipulating files on my hard disk from python code.

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import numpy as np
import json
import math
import sys
from PIL import Image
```

**Image 5. Importing python libraries**

For the reading dataset, I used the pandas library. The dataset basically consists of two columns, the first column is an indexing column with Image names and the second and the main column is the 'whaleID'.(Image 6)

```
train = pd.read_csv('noaa-right-whale-recognition/train.csv', index_col='Image')
train.head()
```

whaleID	
Image	
w_7812.jpg	whale_48813
w_4598.jpg	whale_09913
w_3828.jpg	whale_45062
w_8734.jpg	whale_74162
w_3251.jpg	whale_99558

**Image 6. Reading the dataset by using Pandas library**

If we print a small description of the dataset we will see that the dataset consists of 4544 entities and there are 447 unique whales. (Image 7)

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4544 entries, w_7812.jpg to w_9468.jpg
Data columns (total 1 columns):
whaleID    4544 non-null object
dtypes: object(1)
memory usage: 71.0+ KB
```

```
train.describe()
```

whaleID	
count	4544
unique	447
top	whale_95370
freq	47

## Image 7. Small description of the dataset

The distribution of images on a whale varies greatly. There are 38 whales that have more than twenty photos. (Image 8) Other whales have less than twenty images and even some whales have only one image. This is not good data distribution for making neural networks, because it can cause overtraining and the model will be based on 38 whales. For example if we get cats and dogs classifier, and train it with the data in which dogs images would be 80 percent of the data, our classifier could overtrain and start to classify cats as dogs.

```
train["whaleID"].value_counts()
```

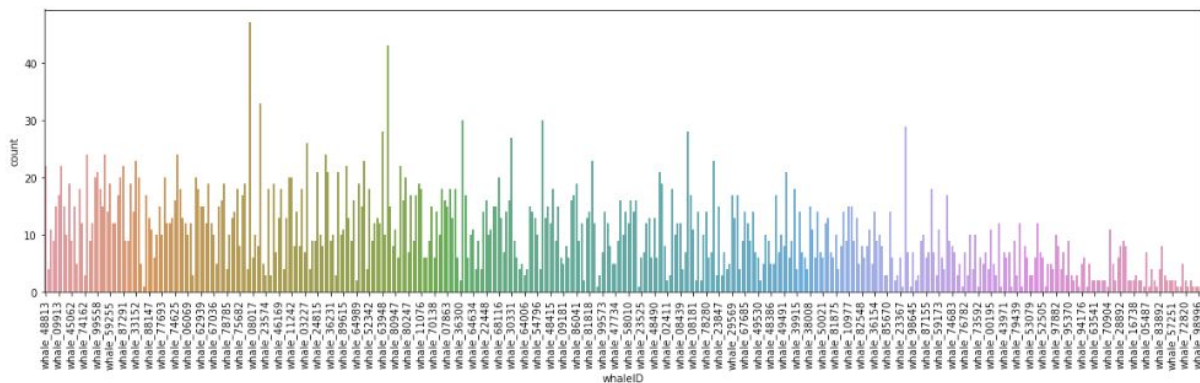
```
whale_95370    47
whale_38681    43
whale_28892    33
whale_90957    30
whale_36851    30
whale_24458    29
whale_85464    28
whale_51195    28
whale_65586    27
whale_52749    26
whale_89615    24
whale_78280    24
whale_08017    24
whale_34656    24
whale_79823    23
whale_68116    23
whale_73666    23
whale_95091    23
whale_87291    22
whale_48966    22
whale_70138    22
whale_48813    22
whale_26288    22
whale_61461    21
whale_87604    21
whale_03227    21
whale_74232    21
whale_66353    21
whale_17604    21
whale_11076    20
..
```

## Image 8. Quantity of images per whale



Let's print out the graphical illustration of the images per whale. We can see clearly that the images per whale vary hugely. There are almost 24 whales with only one photo! (Image 9) That's why we cannot perform just straight splitting to the training and validation. If put such whales into validation sets only, it will result that the classifier will not be able to predict this whale. As a Beginner, I made a big mistake by just randomly splitting the dataset inside the model.

```
#Importing seaborn for making visualizations
#Plotting countplot for graphical visualization of images per whale
import seaborn as sns
plt.figure(figsize=(20, 5))
plots = sns.countplot(train["whaleID"])
plots = plt.xticks(range(0, 447, 5), rotation=90)
```



**Image 9. Histogram of the images per whale**

The resolution of the given images are very huge, some of them more than 2000\*2500. (Image 10)

```
#Reading sample image
#Make sure that the direcorey of the file is right

img = cv2.imread('imgs/w_19.jpg')
plt.imshow(img)
```

<matplotlib.image.AxesImage at 0x1403e3470>



**Image 10. Sample image with its resolution before cropping.**

We don't need such resolution because of several reasons, first of all as I already mentioned water makes a lot of noise and besides that, the ROI (region of interest) is the head of the whales. Moreover, the higher the resolution of the image, the more computing power is needed. That's why I wanted to create my own head localizer by using CNN, but for my luck, I found opensource JSON with the head locations. For cropping images, first of all I created an array with unique whale ids, then I created folders for each unique whale. (Image 11)

```
#Creating list with unique whale ids
whaleIDs = list(train['whaleID'].unique())

#Creating folders with whale IDs
for w in whaleIDs:
    os.makedirs('imgs/'+w)
```

**Image 11. Creating an array with unique whale id and creating folders**

I copied all whale images to new folders, and now all whale images are separated by id. (Image 12)

```

#Formatting images into different directories for futere operations
#Copying each whale image into its own directory
for image in train.index:
    folder = train.loc[image, 'whaleID']
    old = 'imgs/{}'.format(image)
    new = 'imgs/{}/{}'.format(folder, image)
    try:
        os.rename(old, new)
    except:
        print('{} - {}'.format(image, folder))

```

**Image 12. Copying images into the folders by whale id**

Now we cut out an area with the whale face, resize the images to 256\*256. These steps made passport photos of whales. (Image 13)

```

#Reading JSON file with head annotations
#Cropping images, in which there would be only heads of whales
#Make sure that the direcorey of the file is right

def ensure_dir(fn):
    d = os.path.dirname(fn)
    if not os.path.exists(d):
        os.makedirs(d)

# The output image's width and height
mw, mh = 256, 256
# Try to expand any rectangular crops into squares
PAD_TO_BOX = True

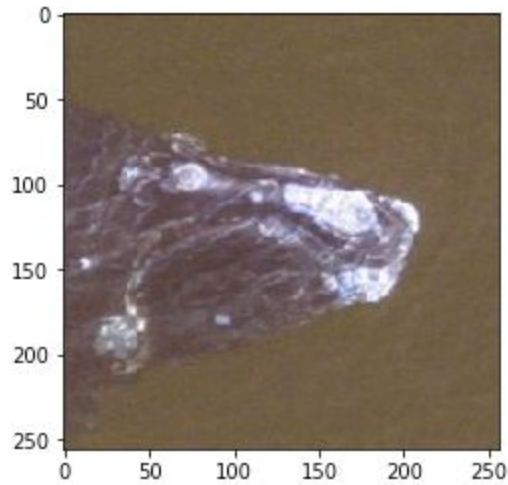
# Load the JSON annotations
for json_fn in sys.argv[1:]:
    #make sure you have this file and fix the path to it
    annotations = json.loads(open('whale_faces_vinh.json').read())
    for data in annotations:
        fn = data['filename']
        # Fix any filenames that start with '../imgs/'
        fn = fn[3:] if fn.startswith('.') else fn
        print('Processing {}'.format(fn))
        if data['annotations']:
            # We're only interested in the Head annotations
            points = [x for x in data['annotations'] if x['class'] == 'Head']
            if not points:
                continue
            points = points[0]
            print('Processing {}'.format(fn))
            ###

```

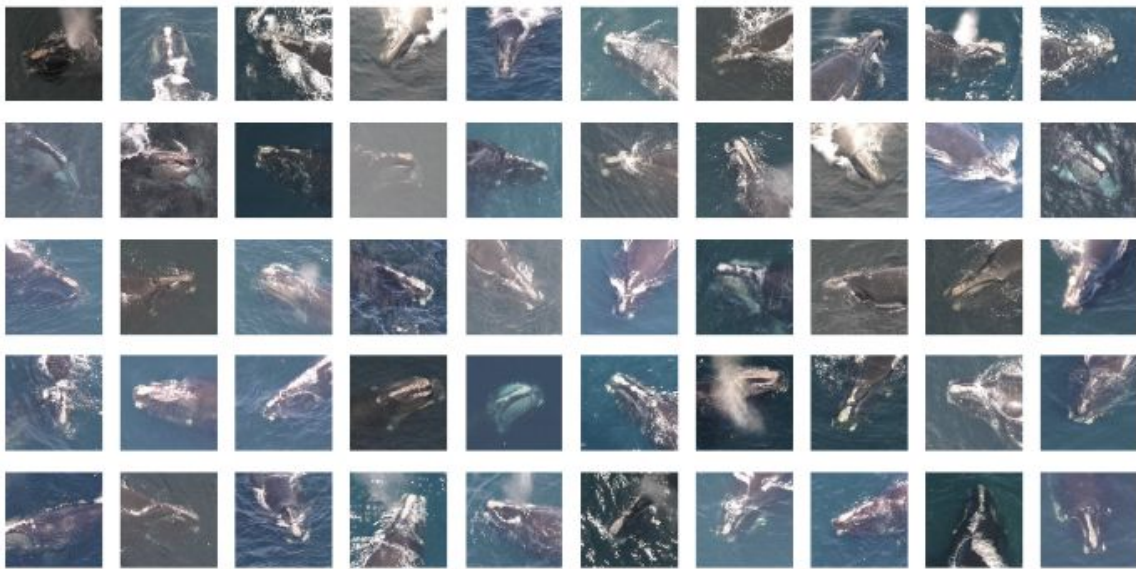
**Image 13. Piece of the cropping code**

```
#Reading sample image after cropping  
#Make sure that the direcorey of the file is right  
img = cv2.imread('heads/whale_97440/w_8600.jpg')  
plt.imshow(img)|
```

<matplotlib.image.AxesImage at 0x12b990470>



**Image 14. Sample cropped image of the whale with resolution**



**Image 15. Other samples of cropped images**

## 4. CNN model

Now we have almost prepared dataset, we need to make a few changes. First we need to convert this dataset into training data. We fixed the issue of different size and made a passport photos, now we need to create training data, for doing this we need to read images and convert them into grayscale. Converting images to grayscale will help us to reduce the size and dimensionality, and especially in this task color of the images is not differentiating fact.

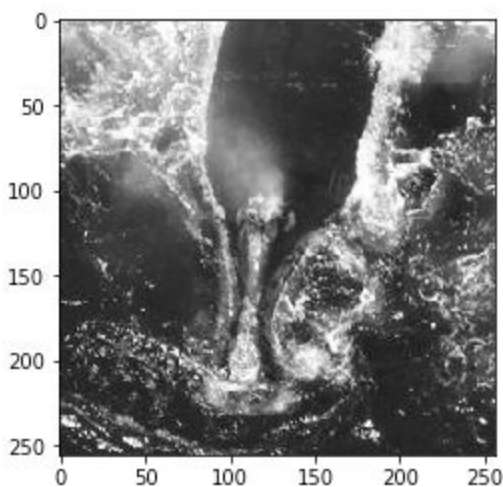
```
#Creating datadirectory variable
#Make sure that the direcorey of the file is right
DataDir = 'heads'

#Creating training dataset
#Reading images in grayscale
training_data = []
for a in whaleIDs:
    path = os.path.join(DataDir,a)
    class_num = whaleIDs.index(a)
    for photo in os.listdir(path):
        photo_array = cv2.imread(os.path.join(path,photo), cv2.IMREAD_GRAYSCALE)
        training_data.append([photo_array,class_num])
```

**Image 16. Creating training data and converting images into grayscale**

```
#Printing sample image from training data
plt.imshow(photo_array, cmap= 'gray')

<matplotlib.image.AxesImage at 0x13df3ca58>
```



**Image 17. Printing sample image from training data**



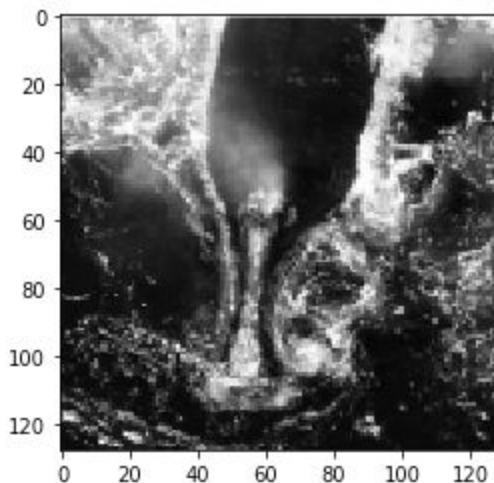
256 by 256 was the perfect size for our images to make predictions. But unfortunately, when I created a model and tried to train it, I realized that it needs much more calculation power. Each epoch on my PC took 4 hours and after 2 epochs my PC overheated. Then I tried to use the Google collab which took 1 hour per epoch. Then I decided to resize the images to 128 by 128 and recreated the training dataset with resized images. (Image 19-20) Resizing images reduced significantly the amount of time of training the model.

```
#Creating training dataset
#Reading images in grayscale
training_data = []
for a in whaleIDs:
    path = os.path.join(DataDir,a)
    class_num = whaleIDs.index(a)
    for photo in os.listdir(path):
        photo_array = cv2.imread(os.path.join(path,photo), cv2.IMREAD_GRAYSCALE)
        photo_array = cv2.resize(photo_array,(128,128))
        training_data.append([photo_array,class_num])
```

**Image 19. Creating training data with resizing the images to 128 by 128**

```
#Printing sample image from training data
plt.imshow(photo_array, cmap= 'gray')
```

<matplotlib.image.AxesImage at 0x127d6bf60>



**Image 20. Sample image after resizing to 128 by 128**

As we can see the quality of the image is still good and we can clearly see the head of the whale. Now we need to randomly shuffle our data. Because if the

data is not random, our classifier will start misclassifying the images. We will use a Random library to shuffle our data. (Image 21-22)

```
import random
random.shuffle(training_data)
```

### Image 21. Shuffling the data

```
for sample in training_data[:10]:
    print(sample[1])
```

```
273
101
376
207
319
391
143
217
308
146
```

### Image 22. Printing labels of the shuffled data

Now we have randomized data, which we should separate for images(features) and labels. I just created a loop for saving our features and labels separately. (Image 23)

```
X = []
y = []
for features, label in training_data:
    X.append(features)
    y.append(label)
```

### Image 23. Separating and saving features and labels

We need to convert our features and labels into a NumPy array for feeding it to our model. Features has a shape of (128,128), it needs to be reshaped into

(-1,128,128,1) shape. 128 means the width and height of our images, the last one means that our images are in grayscale. (Image 24)

```
#Converting our features into numpy array  
X = np.array(X)
```

```
#Checking shape of X  
X[0].shape
```

```
(128, 128)
```

```
#Reshaping our features array  
X = np.array(X).reshape(-1,128,128,1)
```

```
#Checking shape of X after reshaping  
X.shape
```

```
(4543, 128, 128, 1)
```

### Image 24. Converting features into NumPy array. Reshaping our array

Our data needs to be normalized before feeding it to our model. The easiest way of normalizing image data is just dividing it to 255.0. (Image 25)

```
X = X/255.0
```

### Image 25. Normalizing the data

My model is a Sequential model, based on AlexNet. In the first layer, we have a convolutional layer, with 3 by 3 window and with the “Relu” activation function. I also added batch normalization, which gives us more learning rates and speeds up the training process, plus it helps to reduce the sensitivity to initial weights. After that we have a pooling layer with 2 by 2 pooling size. (Image 26)

```
model = Sequential()  
  
model.add(Conv2D(64, (3, 3), activation = 'relu', padding='same', input_shape = (128,128,1)))  
model.add(BatchNormalization())  
model.add(MaxPool2D(2,2))
```

## Image 26. First layer of the model

The model is consists of four convolutional layers. All the layers have different values for pooling size and the size of the convolutional windows. All layers have the same activation function and batch normalization. (Image 27)

```
model.add(Conv2D(128, (7, 7), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(2,2))

model.add(Conv2D(192, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(2,2))

model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(1,1))
```

## Image 27. Convolutional layers of the model

Our data is in 2D shape and before feeding it to a fully connected layer, we need to flatten the data by using Flatten() function. Now we can add fully connected layers, but unfortunately to me, I could add only one layer, because I had problems with computational powers. Basically originally I wanted to add three layers of fully connected neural network layers, in which two of them were supposed to have “Relu” or “LeakyRelu” activation functions, and the last layer supposed to have “Softmax” activation function, with different densities. But if I add more than one layer Google collab stops working because of lack of the RAM. That’s why I have only one fully connected layer with batch normalization and the “Softmax” activation function. The density of the layer is 447. (Image 28)

```
# Flatten the data
model.add(Flatten())

#Adding fully connected neural network
model.add(Dense(447))
model.add(BatchNormalization())
model.add(Activation('softmax'))
```

## Image 28. Flattening the data. Adding a fully connected layer.

For loss function, I used Sparse categorical cross-entropy, first of all, to save a computational memory and because there are 447 categories. I tried to use two different optimizers, stochastic gradient descent, and Adam optimization. I choose Adam because it showed a little bit better accuracy. Now we need just fit the model and train it with our data. The batch size is 32, it is an optimal batch size. The difference in batch sizes is that bigger batch size results faster training but converge would be lower, in smaller batch size it will train slower but converge would be faster. The next attribute is the number of epochs, in ideal, they should be around 40-50, but because each epoch consumed a lot of time I decided just to train with 20 epochs. I split 30 percent of the data to the validation set inside a fit function.

```
#Defining optimizer, loss and accuracy metrics
model.compile(loss= tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer='adam',
              metrics=[ 'accuracy' ])

# Training the model
history = model.fit(X, y, batch_size=32, epochs=20, validation_split=0.3)
```

## Image 29. Defining model parameters and fitting the data.

```
Epoch 15/20
100/100 [=====] - 362s 4s/step - loss: 5.4302 - accuracy: 0.9571 - val_loss: 6.0933 - val_accuracy: 0.0409
Epoch 16/20
100/100 [=====] - 365s 4s/step - loss: 5.4093 - accuracy: 0.9646 - val_loss: 6.0939 - val_accuracy: 0.0402
Epoch 17/20
100/100 [=====] - 362s 4s/step - loss: 5.3764 - accuracy: 0.9696 - val_loss: 6.0933 - val_accuracy: 0.0424
Epoch 18/20
100/100 [=====] - 374s 4s/step - loss: 5.3569 - accuracy: 0.9687 - val_loss: 6.0938 - val_accuracy: 0.0446
Epoch 19/20
100/100 [=====] - 369s 4s/step - loss: 5.3413 - accuracy: 0.9737 - val_loss: 6.0937 - val_accuracy: 0.0394
Epoch 20/20
100/100 [=====] - 378s 4s/step - loss: 5.3237 - accuracy: 0.9743 - val_loss: 6.0922 - val_accuracy: 0.0402
```

## Image 30. Training the model

### 5. Kaggle Submission

For making predictions we need to read “sample\_submission.csv” file and read all images and columns and save them into variables for future use. We need



to read images for predictions, resize them and convert them into NumPy array, then we need to normalize the dataset by dividing it to 255.0. (Image 31)

```
#Reading submission dataset
sample_df = pd.read_csv('noaa-right-whale-recognition/sample_submission.csv')

#Saving column names
cols = sample_df.columns

#Saving image names
fnames = sample_df[['Image']].values

#Reading the images and creating data for predictions
#Converting images into grayscale
#Resizing images to 128 by 128

testing_data = []
path2 = os.path.join('imgs')
for a in fnames:
    for photoes in os.listdir(path2):
        if photoes == a:
            photo_arrayes = cv2.imread(os.path.join(path2,photoes), cv2.IMREAD_GRAYSCALE)
            photo_arrayes = cv2.resize(photo_arrayes,(128,128))
            testing_data.append(photo_arrayes)

#Converting the data to NumPy array
#Normalizing the data
testing_data = np.array(testing_data).reshape(-1,128, 128, 1)
testing_data = testing_data / 255.0
```

### Image 31. Creating dataset for making predictions

Now we can make predictions and save our predictions into CSV file for Kaggle submission.

```
#Feeding the data for making predictions
prediction = model.predict(testing_data)
```

```
# Creating new dataframe for kaggle submission
values = np.hstack([fnames, prediction])
submission_df = pd.DataFrame(values, columns= cols)
submission_df.head()
```

	Image	whale_00195	whale_00442	whale_02411	whale_02608	whale_02839	whale_03103	whale_03227	whale_03623	whale_03728	...
0	w_1947.jpg	0.00589136	0.00235737	0.00488318	0.00292185	0.00202118	0.00485863	0.00179226	0.00134175	0.00209616	...
1	w_11096.jpg	0.00193331	0.00238502	0.00182447	0.00585489	0.00221039	0.00156203	0.00102742	0.0012095	0.00212263	...
2	w_10973.jpg	0.00280997	0.00087733	0.00152864	0.00336862	0.00292212	0.0032394	0.003043	0.00288873	0.00328712	...
3	w_10442.jpg	0.00133291	0.00175103	0.00161679	0.00114142	0.00118953	0.00179361	0.00226315	0.000593626	0.00240328	...
4	w_10606.jpg	0.00319403	0.0033369	0.00443455	0.00211453	0.00482272	0.00146776	0.00139901	0.00153946	0.00376231	...

5 rows x 448 columns

```
#Saving our data into csv file without indexing
submission_df.to_csv('submission.csv', index=False)
```

**Image 32. Making predictions and creating a new data frame with predictions**






Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
submission.csv	2 hours ago	0 seconds	5 seconds	6.46389
Complete				
<a href="#">Jump to your position on the leaderboard</a> ▼				

**Image 33. Submitting predictions to Kaggle competition**

1 submissions for <a href="#">Davron Karimov</a>		Sort by <a href="#">Most recent</a> ▼	
All	Successful	Selected	
Submission and Description	Private Score	Public Score	Use for Final Score
<a href="#">submission.csv</a> 2 hours ago by <a href="#">Davron Karimov</a>	6.43921	6.46389	<input type="checkbox"/>

**Image 34. My submission scores**

After submitting to Kaggle, unfortunately, I didn't appear in the leaderboard. My position would be 225 in the leaderboard.

221	ISILAB		6.40241	15	4y
222	Carsonar		6.40278	1	5y
223	jwjohnson314		6.41746	21	4y
224	klakhor		6.41827	2	4y
225	Sahar		6.53615	4	5y

**Image 35. Leaderboard**

## 6. Conclusion And Future Improvements

The biggest weakness of my work is the lack of hardware computational power. My project had a lack of cross-validation and I couldn't train more complex networks because it consumes unreasonable amounts of time. I couldn't be able to go through all my basic hypothesis and changing parameters of the network, because every tiny change consumed a lot of time.

By using CNN algorithms, we can create head aligners for rotating images in one direction, and it means that the classifier will no longer need to study functions that are invariant to extreme translation and rotation.

I made a mistake by using a local validation set. The dataset should be split separately from our model. Also as I mentioned already, one more challenge of this task was in the distribution of the images per whale, it limits our model significantly. I found several ways of dealing with it, one of them using some augmenting methods. Images will go through filters that are designed to indicate the variance between the images in the set, and then use the filter output as additional training data. These filters uses smoothing, sharpening, and different transformations. Or there is an idea based on FaceNet [2].

Trying to use different methods and approaches, adding more fully connected layers will also improve the model. Transfer learning could also offer

better model solutions for our classification problem. All this hypothesis needs more time and GPU power. With this challenge, I learned a lot when was trying to solve so many problems. Also with this pandemic, I couldn't team up with other students to perform better solutions.

## 7. References

1. Kaggle.com. 2020. *Right Whale Recognition* | Kaggle. [online] Available at: <<https://www.kaggle.com/c/noaa-right-whale-recognition>> [Accessed 6 May 2020].
2. Schroff, F., Kalenichenko, D. and Philbin, J., 2020. *Facenet: A Unified Embedding For Face Recognition And Clustering*.
3. Masters, D. and Luschi, C., 2014. *Revisiting Small Batch Training For Deep Neural Networks*. [online] arXiv.org. Available at: <<https://arxiv.org/abs/1804.07612>> [Accessed 6 May 2020].
4. Simonyan, K. and Zisserman, A., 2015. *Very Deep Convolutional Networks For Large-Scale Image Recognition*. [online] arXiv.org. Available at: <<https://arxiv.org/abs/1409.1556>> [Accessed 6 May 2020].
5. Géron, A., 2019. *Hands-On Machine Learning With Scikit-Learn, Keras, And Tensorflow*. O'Reilly.
6. McKinney, W., 2018. *Python For Data Analysis*. O'Reilly.
7. Szegedy, C., Vanhoucke, V. and Rabinovich, A., 2014. *Going Deeper With Convolutions*. [online] arXiv.org. Available at: <<https://arxiv.org/abs/1409.4842>> [Accessed 6 May 2020].
8. Deeplearningbook.org. 2020. *Deep Learning*. [online] Available at: <<http://www.deeplearningbook.org/>> [Accessed 6 May 2020].
9. GitHub. 2020. *Edward0im/Alexnet-Using-Keras-2X-Tensorflow*. [online] Available at: <[https://github.com/edward0im/Alexnet-using-keras-2x-Tensorflow/blob/master/alexnet\\_train\\_mnist\\_example.py](https://github.com/edward0im/Alexnet-using-keras-2x-Tensorflow/blob/master/alexnet_train_mnist_example.py)> [Accessed 6 May 2020].
10. Kaggle.com. 2020. *Right Whale Recognition Discussions* | Kaggle. [online] Available at:

**<<https://www.kaggle.com/c/noaa-right-whale-recognition/discussion/17421>> [Accessed 6 May 2020].**

- 11. Open Data Science (ODS.ai). 2020. *Open Data Science*. [online]  
Available at: <<https://ods.ai/>> [Accessed 6 May 2020].**