



Department of Computer Science

**BSCCS Final Year Project Report
2020-2021**

20CS001

Computer vision and Web-app for Chinese White Dolphin

(Volume __ of __)

Student Name : WONG, Tsz Chun

Student No. : 55744759

Programme Code : BSCEGU2

Supervisor : Dr CHAN, Antoni Bert

1st Reader : Dr MA, Kede

**2nd Reader : Dr CHAN, Wing Kwong
Ricky**

For Official Use Only

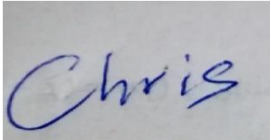
Student Final Year Project Declaration

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

Computer vision and Web-app for Chinese White Dolphin

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

Student Name: WONG, Tsz Chun

Signature: 

Student ID: 55744759

Date: July 12, 2021

Extended Abstract

In Hong Kong, there are many precious marine mammals such as the Chinese White Dolphin and Finless Porpoise. However, in recent years, more and more human activities may threaten the living of these marine mammals. In this project, a web application will be developed to collect the data of Chinese White Dolphins in Hong Kong. In this way, through images, we know what is happening with them in Hong Kong and what challenges they are facing under various environmental or man-made factors.

In this project, Chinese White dolphins will be recognized in images to help solve the above problem. As all dolphins' images are added into the system and stored in the database, they will be annotated. A computer vision model can thus be trained. Then the computer vision algorithms will identify a new dolphin image by using the previously collected images in the database. Now, if people upload any dolphin image to the web application, it can be recognized, and we know what is happening with these images.

Acknowledgment

I would like to thank my supervisor Dr. CHAN, Antoni Bert. When I am lost, he keeps giving me directions on how to do this final year project. Moreover, he has given a lot of helpful suggestions on doing the recognition. For example, I can start by implementing face recognition to get some ideas.

Secondly, I would like to thank Dr. Lee Chung Sing Victor for guiding me in this final year project. Sometimes, when I have no motivation or make no progress, Dr. Lee push me further so that I can be productive and hand in the work on time.

Thirdly, I would like to thank Dr. Brain KOT for giving me Chinese-White dolphin data. Without these data, I cannot even start doing the project, let alone building a dolphin identifier.

Last but not least, I would like to thank the people who are associated with my project. With their help, this project can be ended smoothly.

With sincere regards,

Chris Wong

Table of Contents

1. Introduction.....	7
1.1 Photo-identification.....	7
1.2 Limitations	7
1.3 Aim Of This Project	7
1.4 Project Scope.....	7
2. Literature Review.....	9
2.1 Traditional Convolutional Neural Network And Softmax Function-Based Approach	9
2.2 Modern Convolutional Neural Network And Triplet Loss Function-Based Approach	9
2.2.1 Triplet Loss Function	10
2.3 Ways To Higher The Accuracy And Performance	10
2.3.1 Domain Knowledge.....	10
2.3.2 K-Nearest Neighbors Algorithm	11
2.3.3 Support Vector Machine Algorithm	11
2.4 Integration Of The Above Techniques	11
3. Proposed Design, Solution, And System	12
3.1 Train A Fin Detector.....	12
3.2 Extracting Dolphins' Fins	12
3.3 Train The FaceNet Model	12
3.3.1 Data Argumentation	12
3.3.2 Triplet Loss Function	12
3.3.3 How To Select Triplets.....	13
3.4 Classification And Testing	13
3.5 API for integration	13
4. Detailed Methodology And Implementation	14
4.1 Detecting The Fin Part Of The Chinese White Dolphin	14
4.1.1 Set Up The TensorFlow Environment.....	14
4.1.2 Preparing The Dataset	14
4.1.3 Preprocessing	15
4.1.4 Train The Model.....	17
4.1.5 Evaluate The Model	17
4.1.6 Export The Model	18
4.2 Extracting The Fin Part Of The Chinese White Dolphin	18
4.2.1 Import Libraries.....	18
4.2.2 Load The Model And Label Map	19
4.2.3 Getting All Detection Boxes For A Single Image	19
4.2.4 Resize And Save The Fin Part.....	20
4.3 Prepare The Dataset For The FaceNet Model.....	21
4.3.1 Data augmentation.....	21
4.4 Fine-tuning The Triplet Loss Function FaceNet Model.....	22
4.5 Train The FaceNet Model Using Triplet Loss Function	23
4.6 Generating Pairs.....	23
4.7 Evaluate The Trained FaceNet Model.....	25
4.8 Train A Classifier	26
4.9 Evaluate The Classifier	27
4.10 Predict A New Input Image	28
5. Testing Procedure And Results.....	29
5.1 Testing Procedure Of The Fin Detector	29
5.1.1 The Evaluation Result Of The Fin Detector.....	29
5.2 Testing Procedure Of The Trained FaceNet Model	29

5.2.1 The Evaluation Result Of The Trained FaceNet Model.....	30
5.3 Testing Procedure Of The Trained Classifier.....	31
5.3.1 The Evaluation Result Of The Trained Classifier for ratio 50% to 50%	31
5.3.2 The Evaluation Result Of The Trained Classifier for ratio 90% to 10%	33
6 Conclusion	35
6.1 Critical review.....	35
6.2 Summary of achievements.....	35
6.3 Suggestions for extensions to the project.....	35
7. Monthly Logs.....	37
8. Reference	39

1. Introduction

1.1 Photo-identification

In Hong Kong, there are many precious marine mammals such as the Chinese White Dolphin and Finless Porpoise. However, in recent years, more and more human activities may threaten the living of these marine mammals. Therefore, ecological scientists take photos of marine mammals and use the photo-identification (photo-id) technique to identify them based on their unique fins [1]. In this way, they gain a lot of information such as the number and distribution. As a result, they know what is happening with the marine mammals in Hong Kong and their challenges under various environmental or man-made factors.

1.2 Limitations

However, after taking photos, the ecological scientists also have to manually classify them based on their fin to the correct catalog. The whole process will be very time-consuming. If the amount of unclassified images is large, it takes more labor resources as well. Moreover, people sometimes make mistakes. They may carelessly mismatch the dolphin into the wrong catalog. Then the final results of investigating the distribution of the marine mammals are also affected. Furthermore, the photo-identification technique requires specific domain knowledge. Only ecological scientists can identify the difference between two Chinese-White dolphins' photos, that they are the same or not. The costs of doing this job become very high. Thus, in this project, marine mammals will be recognized automatically in images using deep learning to help solve the above problem. In this way, the time, labor resources, and money can be reduced to fasten the investigation and do other meaningful things.

1.3 Aim Of This Project

In this project, the objective is to recognize the marine mammals in images to help solve the above problem. It will be divided into three parts. Firstly, the famous deep learning framework Tensorflow will be used to train a computer-vision model to see if it can identify the Chinese-White dolphins accurately. Then, different machine learning techniques will be used for higher accuracy. More importantly, domain knowledge known by the ecological scientists and the identification technique used in other fields such as face-recognition will also be discussed to have a higher identification accuracy.

1.4 Project Scope

The project starts with doing research and reading literature. The tasks needed will be analyzed. All the milestones and timelines will be estimated and determine the final project scope. The main part of this project is to use classified good quality marine mammals' images to train a

computer vision model. It is expected that the computer vision algorithm's accuracy will be higher than 90%.

Since this is a group project, the main part will be integrated with other students' projects to be a complete system. Thus, other technical components like database, web application, and server are needed to be handled as well to integrate the system successfully. Testing will also be conducted as part of the project scope to fix any bug that occurred.

2. Literature Review

This section will discuss the traditional approach and modern approach to doing the recognition these days. More importantly, different machine learning techniques and domain knowledge will be reviewed to increase accuracy.

2.1 Traditional Convolutional Neural Network And Softmax Function-Based Approach

In this day and age, deep learning is now all the rage. Many applications like autonomous cars, facial recognition have used deep learning technology. There are many deep learning architectures, and the convolution neural network is one of them. A convolution neural network can extract the features from the raw image, and then a classifier can classify the extracted features [2]. It is an artificial neural network. It consists of an input layer, many hidden layers, and an output layer. The softmax layer will be as the output layer. It will output every last node in a range of 0 to 1 as probability, and their sum of probabilities will be up to 1. Unlike other image processing algorithms, it is excellent that it seldom needs to do the preprocessing of images. When the input layer is initiated with images, it will then be passed through the hidden layers. After passing through each hidden layer, the most significant data characteristics will be extracted out. Finally, it is fully connected between the input layer and the output layer.

Therefore, after the layers are built up, we just need to feed the neural network with input images. Then, it will automatically do the feature extraction and provide the output results [3]. However, the traditional convolutional neural network has some drawbacks in doing the recognition. Firstly, it requires a lot of data samples as input to build up a depth model. In this way, the model has a decent accuracy for the output. For example, Google's FaceNet used more than two hundred million faces containing eight million people identities to train the model [4]. In this project, the sample data set of Chinese-White Dolphin is only a few. Using such a data-driven approach, the accuracy of the trained model will be an issue. Moreover, if I have some new identities that need to recognize, I have to add thousands of the new identity's images, change the neural network and retrain the model again. It will be very time-consuming and inefficient.

2.2 Modern Convolutional Neural Network And Triplet Loss Function-Based Approach

In this approach, a different convolutional neural network than the above one will be built up. The main difference from the above method is that instead of the softmax function, the triplet loss function is designed and used only to learn from two examples, the positive and negative. However, the above approach is learning from multiple instances. We can see that lesser training

set data are needed to understand a similarity function that quantifies how different the given images are. Thus, when we have some new identities that need to be recognized, we now do not need to add thousands of images but just a few sample images. The loss function's details are as follows [4].

2.2.1 Triplet Loss Function

Triplet loss is a loss function for the machine learning algorithm, especially for minor difference problems such as human faces. It can also be applied to the dolphin recognition problem because it is hard for normal people to identify the differences between Chinese-White dolphins.

Moreover, every dolphin is unique. The number of classes for every single dolphin is enormous. I need to compare two dolphins and say whether they are from the same dolphin or not. In this case, triplet loss architecture helps a lot, since now every unique dolphin needs to be compared with the same dolphin and a different dolphin like in figure 1. The model will be trained efficiently that the same dolphin needs to have a high degree of similarity and a very low similarity with the different one. In the end, the loss function will generate a single vector called embedding for analyzing the similarity between the images [5].



Figure 2.2.1.1 shows the learning of the model to train every single dolphin.

2.3 Ways To Higher The Accuracy And Performance

We will use the modern deep convolutional neural network and triplet loss function-based approach to do recognition as it is more efficient and requires less data input. Striving for the best, different machine learning techniques will be tested to see which one can have the best performance of the approach. Furthermore, domain knowledge will be applied to higher the accuracy of the model.

2.3.1 Domain Knowledge

As the ecological scientists know the most efficient way to identify the Chinese-White dolphins, the domain knowledge can be learned from them and applied. By reading their papers, I found that the most efficient way to determine the difference between them is to look at the unique features of the dolphin's back and dorsal fins, such as nicks, scars, and distinctive color patterns [6]. Identify each dolphin according to the features, which could provide information on individual home ranges and abundance. Therefore, there is no need to feed the whole dolphin's body image into the model anymore. Now the dolphin's fin part is only needed. It significantly

reduces the data set size and the training time of the model as the machine learning algorithm can be designed specifically to extract the feature of their fins and calculate the euclidean distance between them to see the difference [7].

2.3.2 K-Nearest Neighbors Algorithm

The k-nearest neighbors (KNN) algorithm is a supervised machine learning algorithm that can solve both classification and regression problems. In this project, we are solving a classification problem. Thus, KNN can be very useful in identifying similar Chinese-White dolphins. The main idea of the KNN algorithm is that it finds the distances between the target and the samples in the data set. Then, we choose a K value. The target will be assigned to the class of the nearest neighbor. The correct K value can be selected by trying several values that perform the best.

There are some advantages of the KNN algorithm. Firstly, it is pretty easy to implement. We do not need to tune any parameters or make any assumptions.

Moreover, it can be used in many ways like in classification, regression, and searching.

However, it also has a disadvantage. When the number of identities that need to identify increases, the algorithm will be slower. [8]

2.3.3 Support Vector Machine Algorithm

Similar to the KNN algorithm, the Support vector machine (SVM) algorithm also can be used for regression and classification tasks. It aims to build a model that can assign the new examples into one of the two categories. Now, it becomes a binary linear classifier. When the model is built up, the instances now are as points in space. A hyperplane in an N-dimensional space which N is the number of features that distinctly classifies the data points, is built up. They are divided by a wide gap that has separated these examples into categories. New samples will be predicted to fall into which category. The advantage of using this algorithm is that it may give a better performance than the KNN algorithm. However, it is pretty hard to implement and takes time to train the SVM model. When we have a new example, we need to train the model again [9].

2.4 Integration Of The Above Techniques

To sum up, in this project, both the modern and traditional approaches will be used. The traditional one will be used to detect and extract the fin part of the dolphin, as mentioned in the domain knowledge part. The most efficient way to identify Chinese-White dolphins is to look at their fins. These fin images will be gathered and labeled to be our input data. Then, the modern approach will be applied to do the recognition. Embedding will be generated from the model to analyze the similarity between the images. Finally, the SVM algorithm will be tested to see its performance in classifying the new examples into the correct category. The details of the solution will be discussed in the next section.[10]

3. Proposed Design, Solution, And System

As mentioned, recognizing Chinese-White dolphins will be divided into five parts: training a fin detector, extracting dolphins' fins, training the FaceNet model, classification and testing, API for integration.

3.1 Train A Fin Detector

We will use the Tensorflow object detection API to train a fin detector that can accurately detect the fin part of a dolphin. The pre-trained model Fast Region-based Convolutional Network model will be used to train our dataset. Hundreds of dolphin images will be used as input to train the model. A ratio of 90 to 10 will be used between the training set and the testing set. For the training dataset, there are in total 390 dolphin images. For the testing dataset, there are in total 43 dolphin images.

3.2 Extracting Dolphins' Fins

The above-trained model will be used to detect the Chinese-White dolphins' fins. A program written in Python will call the trained model, detect the fin part with the bounding box and extract out all of them. All the fin images will be saved with the corresponding class.

3.3 Train The FaceNet Model

The pre-trained FaceNet model will be used to train the Chinese-White dolphins' fins. We will divide it into a ratio of nearly 9:1 between the training and validation dataset.

The training dataset consists of 11 classes and 148 images. The validation dataset consists of 3 classes and 16 images.

3.3.1 Data Argumentation

As the dataset size is quite small, we will do data argumentation to increase the number of images. Random cropping and random horizontal flipping will be performed. In this way, we can have more data to train the model to increase the accuracy.

3.3.2 Triplet Loss Function

The pre-trained Facenet model will be trained by the embeddings of the dolphins' fins according to the similarity between these images. Two same dolphin-ID images must have very short distance embedding. Two different dolphin-ID images must have very large distance embedding. The network will be trained according to the mentioned triplet-loss function.

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]$$

Figure 3.2.1 shows the Mathematical Equation of Triplet Loss Function.

From figure 3.2.1, we can see the Mathematical Equation of Triplet Loss Function. It can be described using a Euclidean distance function. The letter i denotes the i th input. Letter a denotes Anchor image, p means Positive image, n indicates Negative image. $F(x)$ takes x as an input and returns a 128-dimensional vector W as the model has 128 numbers in the embedding vectors. It aims to minimize the first term such that the distance between the anchor and the positive image's distance is very small. In the second term, the anchor and the negative image can be maximized such that the distance between them is enormous. Moreover, we want to avoid the situation that $f(\text{Image}) = 0$. Therefore, we will add a constant α and to make the difference of the negative larger [10].

3.3.3 How To Select Triplets

If we choose an anchor and its negative randomly, it may be too easy to identify the difference by the algorithm, and the model will not learn much from this. Therefore, we need to find an appropriate triplet to train the model. The examples that are hard to differentiate are suitable for the model to learn more. The model will be ensured that the two very similar images are very different. They have a large enough difference of embedding.

3.4 Classification And Testing

Last but essential, when the model is trained by the triplet function, the embeddings with the corresponding class will be generated. Now, we can do the classification and test our trained model. As mentioned, the SVM algorithm can be used to do the classification. Suppose we now have a new Chinese-White dolphin's fin image needed to be identified. Firstly, the new image's embeddings will be generated out. Then, we calculate the distance between the new image and every class in the model to see which class has the closest distance with the new image. Finally, this new image will be associated with the corresponding class. This is the main idea of how to do the classification. The SVM algorithm will be tested out in this project to see if its performance can accurately identify the new dolphin image belonging to which class category. This will be discussed in detail in the implementation session.

3.5 API for integration

Since this is a group project, we need to integrate all the things into one system. Once the classifier and the models are ready, an application programming interface will be written for the web application system to call. Only the prediction result of the input dolphin image will be shown. In this way, other groupmates can easily get the result just by calling the API without knowing the details of the implementation.

4. Detailed Methodology And Implementation

Programming language Python is used for implementation. For the fin detector, Tensorflow object detection API will be called to train the model. Fast Region-based Convolutional Network SoftMax loss function-based pre-trained model will be used to train our own detector. For the dolphin identification, pre-trained model Inception ResNet v1 will be used to train our own model using the triplet loss function. After that, it will be used to train the classifier using the support vector machine algorithm. The details are as follows.

4.1 Detecting The Fin Part Of The Chinese White Dolphin

First of all, following the above 'Preliminary Design, Solution, And System' section, the fin parts of the images will be detected and extracted out. Tensorflow object detection API will be called to do the detection. The pre-trained model used will be `faster_rcnn_resnet50_v1_640x640`.

4.1.1 Set Up The TensorFlow Environment

Firstly, Anaconda is installed. It will be the platform to install other packages. Secondly, Python 3.6 is installed. Thirdly, TensorFlow version 1.15 is installed. Fourthly, install the TensorFlow object detection API. Fifthly, since the API uses Protobufs to configure model and training parameters, we need to install the Protobufs libraries.

4.1.2 Preparing The Dataset

After setting up the environment, we need to label our dataset. We first need to install the `labelImg` tool. We will use this tool to annotate the images. Once the images are annotated, a set of new *.xml files will be generated. Then, we will need to divide our dataset. Since only part of the dataset will be used for training, the rest will be used for evaluating the model. The ratio between the training and testing set will be 90% to 10%, respectively. For the training dataset, there are in total 390 dolphin images. For the testing dataset, there are in total 43 dolphin images.

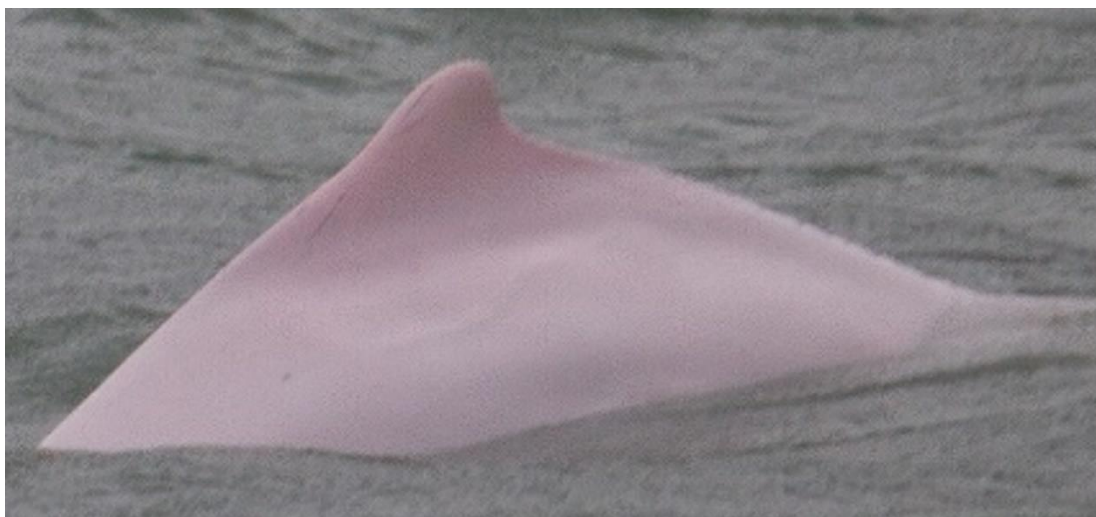


Figure 4.1.2.1 shows one of the Chinese-White Dolphin images in the training dataset.



Figure 4.1.2.2 shows one of the Chinese-White Dolphin images in the training dataset after labeling the fin part.

4.1.3 Preprocessing

As TensorFlow requires a label map, the class has to be labeled with an integer value. Then the label map will be used for training and detection purposes. The label_map.pbtxt is configured as below:

```
item {  
  id: 1  
  name: 'fin'  
}
```

Figure 4.1.3.1 shows the format of the label map.

Next, we will need to generate the TensorFlow records. We need to convert the annotations into TFRecord format, which means *.xml needs to be converted to *.record. After converting both the training data and testing data into TFRecord, we can fine-tune the pre-trained model.

Changes will be made in the pipeline.config file. The details are as follows.

Firstly, the number of different label classes will be set to 1 because we only need to detect the fin class.

```

model {
  faster_rcnn {
    num_classes: 1
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 160
        max_dimension: 160
        pad_to_max_dimension: true
      }
    }
  }
}

```

Figure 4.1.3.2 shows the configure file for fine-tuning the model.

Secondly, the batch size in the training configuration session will be set to 8 as a higher value requires more memory to train.

```

train_config: {
  batch_size: 8
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 25000
}

```

Figure 4.1.3.3 shows the configure file for fine-tuning the model.

Thirdly, we need to link the fine-tune checkpoint to the checkpoint of the pre-trained model. Set the fine-tune checkpoint type to be 'detection' because we are training a detection model.

```

fine_tune_checkpoint_version: V2
fine_tune_checkpoint: "pre-trained-models/faster_rcnn_resnet50_v1_640x640_coco17_tpu-8/checkpoint/ckpt-0"
fine_tune_checkpoint_type: "detection"
data_augmentation_options {
  random_horizontal_flip {
  }
}

```

Figure 4.1.3.4 shows the configure file for fine-tuning the model.

Finally, we need to set the path to link to our testing set and training set. And also, link to the label map path to let the model read the class label.

```

train_input_reader: {
  label_map_path: "annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
}

```

Figure 4.1.3.5 shows the configure file for fine-tuning the model.


```
eval_input_reader: {
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/test.record"
  }
}
```

Figure 4.1.3.6 shows the configure file for fine-tuning the model.

4.1.4 Train The Model

In order to train the model, we need to run the `model_main_tf2.py` script from the object detection API file by using the command:

```
python model_main_tf2.py --model_dir=models/ my_faster_rcnn_resnet50_v1 --  
pipeline_config_path=models/ my_faster_rcnn_resnet50_v1/pipeline.config.
```

Then, the model will start to be trained. Once the model's total loss is at least two or ideally one and lower, we will have a decent detection accuracy. However, a very low total loss should be avoided because it results in overfitting the dataset. It means that when we have a new image outside the dataset, the model will detect it badly. We can monitor the total loss by using the TensorBoard.

4.1.5 Evaluate The Model

After training the model, we can now evaluate the model's precision. In the above preparing the dataset section, we divided the dataset into 9:1 for training and evaluation. This section will try to use the testing images to see how well the model performs in detecting fins. By running the command:

```
python model_main_tf2.py --model_dir=models/my_faster_rcnn_resnet50_v1 --  
pipeline_config_path=models/my_faster_rcnn_resnet50_v1/pipeline.config --  
checkpoint_dir=models/my_faster_rcnn_resnet50_v1
```

The evaluation process uses the model's checkpoint file to evaluate how well the model detects the fin part in the testing dataset. The results will be stored in the form of tf event files. Then we can use the TensorBoard to look at the Detection Boxes Precision/mAP, classification loss values in the tf event files by using the command:

```
tensorboard --logdir=models/my_faster_rcnn_resnet50_v1.
```

The evaluation result of the model will be discussed in the next section, 'Testing procedure and results'.

4.1.6 Export The Model

After training and evaluating the model, we can export the model and then use it for fin detection later. We can use the command:

```
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path .\models\
my_faster_rcnn_resnet50_v1\pipeline.config --trained_checkpoint_dir .\models
my_faster_rcnn_resnet50_v1\ --output_directory .\exported-models\my_model
```

to export the model into the my_model directory. A saved_model.pb file will be generated. It will be very useful to extract the fin part of the dolphin later in the next section.

4.2 Extracting The Fin Part Of The Chinese White Dolphin

When the fin detector is ready, we can now extract the fin part of every Chinese White Dolphin image to be the input data of training another model to identify them. All the fin images will be labeled with the corresponding class. The detailed implementation is as follows.

4.2.1 Import Libraries

▼ Imports

```
[ ] import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
import time

from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
from IPython.display import display
```

Figure 4.2.1.1 shows the needed libraries and modules like TensorFlow, NumPy.

Import the object detection module.

```
▶ from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
```

Figure 4.2.1.2 shows the Object detection modules.

4.2.2 Load The Model And Label Map

Loader

```
[ ] def load_model(model_name):  
  
    PATH_TO_SAVED_MODEL = "C:/Users/tszch/Documents/TensorFlow/workspacce/training_demo/exported-models/my_model/saved_model"  
  
    print('Loading model...', end='')  
    start_time = time.time()  
  
    # Load saved model and build the detection function  
    detect_fn = tf.saved_model.load(PATH_TO_SAVED_MODEL)  
  
    end_time = time.time()  
    elapsed_time = end_time - start_time  
    print('Done! Took {} seconds'.format(elapsed_time))  
  
    return detect_fn
```

Figure 4.2.2.1 shows the Python code for loading the trained model.

Loading label map

Label maps map indices to category names, so that when our convolution network predicts 5, we know that this corresponds to `airplane`. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine

```
[ ] # List of the strings that is used to add correct label for each box.  
PATH_TO_LABELS = 'C:/Users/tszch/Documents/TensorFlow/workspacce/training_demo/annotations/label_map.pbtxt'  
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
```

Figure 4.2.2.2 shows the Python code for loading the label map.

4.2.3 Getting All Detection Boxes For A Single Image

```
def run_inference_for_single_image(model, image):  
    image = np.asarray(image)  
    # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.  
    input_tensor = tf.convert_to_tensor(image)  
    # The model expects a batch of images, so add an axis with `tf.newaxis`.  
    input_tensor = input_tensor[tf.newaxis,...]  
  
    # Run inference  
    model_fn = model.signatures['serving_default']  
    output_dict = model_fn(input_tensor)  
  
    # All outputs are batches tensors.  
    # Convert to numpy arrays, and take index [0] to remove the batch dimension.  
    # We're only interested in the first num_detections.  
    num_detections = int(output_dict.pop('num_detections'))  
    output_dict = {key:value[0, :num_detections].numpy()  
                   for key,value in output_dict.items()}  
    output_dict['num_detections'] = num_detections
```

```

# detection_classes should be ints.
output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

# Handle models with masks:
if 'detection_masks' in output_dict:
    # Reframe the the bbox mask to the image size.
    detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
        output_dict['detection_masks'], output_dict['detection_boxes'],
        image.shape[0], image.shape[1])
    detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                       tf.uint8)
    output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

return output_dict

```

Figure 4.2.3.1 shows the Python code that will get all the detection boxes of the fin part when we input a single dolphin image.

4.2.4 Resize And Save The Fin Part

```

def show_inference(model, image_path, required_size=(160,160), save_fins=True):
    # the array based representation of the image will be used later in order to prepare the
    # result image with boxes and labels on it.
    image_np = np.array(Image.open(image_path))
    # Actual detection.
    output_dict = run_inference_for_single_image(model, image_np)

    image = Image.open(image_path)
    image = image.convert('RGB')
    pixels = asarray(image)
    ymin, xmin, ymax, xmax = output_dict['detection_boxes'][0]
    ymin, xmin, ymax, xmax = abs(ymin), abs(xmin), abs(ymax), abs(xmax)
    im_width, im_height = image.size
    im_width, im_height = abs(im_width), abs(im_height)
    (left, right, top, bottom) = (xmin * im_width, xmax * im_width,
                                ymin * im_height, ymax * im_height)
    fin = pixels[int(top):int(bottom), int(left):int(right)]
    image = Image.fromarray(fin)
    image = image.resize(required_size)

    if (save_fins):
        path = os.path.split(os.path.abspath(image_path))[0]
        file_name = os.path.split(os.path.abspath(image_path))[1]
        dolphin_name = os.path.basename(os.path.normpath(Path(path)))
        project_folder=Path(path).parent.parent
        print(dolphin_name)
        target_folder=os.path.join(project_folder, 'fins_mini_dataset2', dolphin_name)
        if not os.path.exists(target_folder):
            os.makedirs(target_folder)
        target_fin_file_path=os.path.join(target_folder, file_name)
        print(target_fin_file_path)
        image.save(target_fin_file_path)
    fin_array = asarray(image)
    return fin_array

```

Figure 4.2.4.1 shows the Python code for resizing and saving the fin part.

The above Python code shows that when we get all the detection boxes of the single image by using the function in section 4.2.3, we will only use the first detection box to get the fin part of the image. Then it will be resized to 160x160 image size because later, we will use it as input

data to train the FaceNet model. After that, it will be saved in the file named fins_mini_dataset2 with the corresponding class name.



Figure 4.2.4.2 shows the fin part that was extracted out and resized to 160x160

4.3 Prepare The Dataset For The FaceNet Model

Since we have extracted the fin part of the dolphin images in the above section, we now need to divide it into the training dataset and validation dataset. At first, there are 14 classes and in total 164 dolphin images. We will divide it into a ratio of nearly 9:1 between the training and validation dataset.

The training dataset consists of 11 classes and 148 images. The validation dataset consists of 3 classes and 16 images. However, the distribution of images per class is imbalanced. For example, class 8 actually has only 4 images, but class 1 has 20 images. This will affect the classification result. Some individuals will hard to be identified due to the lack of data.

4.3.1 Data augmentation

Random cropping and random horizontal flipping will be performed to increase the number of dolphin images. With more data, the model's accuracy can be improved.



Figure 4.3.1.1 shows the original extracted fin part



Figure 4.3.1.2 shows the vertically flipped extracted fin part

4.4 Fine-tuning The Triplet Loss Function FaceNet Model

This section will start fine-tuning the FaceNet model as all the input data by extracting the fin part from the whole dolphin is ready. Since we will train the model using the triplet loss function, we will edit the parse_argument function in the train_tripletloss.py.

```
def parse_arguments(argv):
    parser = argparse.ArgumentParser()

    parser.add_argument('--logs_base_dir', type=str,
                        help='Directory where to write event logs.', default='C:/Users/tszsch/Desktop/fyp/facenet/logs/facenet')
    parser.add_argument('--models_base_dir', type=str,
                        help='Directory where to write trained models and checkpoints.', default='C:/Users/tszsch/Desktop/fyp/facenet/models/facenet')
    parser.add_argument('--gpu_memory_fraction', type=float,
                        help='Upper bound on the amount of GPU memory that will be used by the process.', default=1.0)
    parser.add_argument('--pretrained_model', type=str,
                        help='Load a pretrained model before training starts.')
    parser.add_argument('--data_dir', type=str,
                        help='Path to the data directory containing aligned face patches.',
                        #default='~/datasets/casia/casia_maxpy_mtcnnalign_182_160')
                        default='C:/Users/tszsch/Desktop/fyp/facenet/src/data2')
    parser.add_argument('--model_def', type=str,
                        help='Model definition. Points to a module containing the definition of the inference graph.', default='models.inception_resnet_v1')
    parser.add_argument('--max_nrof_epochs', type=int,
                        help='Number of epochs to run.', default=500)
    parser.add_argument('--batch_size', type=int,
                        help='Number of images to process in a batch.', default=90)
    parser.add_argument('--image_size', type=int,
                        help='Image size (height, width) in pixels.', default=160)
    parser.add_argument('--people_per_batch', type=int,
                        help='Number of people per batch.', default=3)
    parser.add_argument('--images_per_person', type=int,
                        help='Number of images per person.', default=2)
```

Figure 4.4.1 shows the parse_arguments function.

Firstly, we will set the path for the log file to store the event log. Secondly, we will set the path to store the trained FaceNet model and checkpoints. Thirdly, we will train the inception_resnet_v1 as the classifier to link the path to the ‘—model_def’ argument. Fourthly, we will set the number of epochs to run to be 500. Fifthly, we will assign the number of images to process in a batch to be 90. Sixthly, the image size must be 160x160 because the pre-trained model only accepts this size. Seventhly, we will set the number of dolphins to be trained per batch to be three and the number of images per dolphin to be two because we have a small dataset.

```
parser.add_argument('--epoch_size', type=int,
                    help='Number of batches per epoch.', default=500)
parser.add_argument('--alpha', type=float,
                    help='Positive to negative triplet distance margin.', default=0.2)
parser.add_argument('--embedding_size', type=int,
                    help='Dimensionality of the embedding.', default=128)
parser.add_argument('--random_crop',
                    help='Performs random cropping of training images. If false, the center image_size pixels from the training images are used. ' +
                    'If the size of the images in the data directory is equal to image_size no cropping is performed', action='store_true')
parser.add_argument('--random_flip',
                    help='Performs random horizontal flipping of training images.', action='store_true')
parser.add_argument('--keep_probability', type=float,
                    help='Keep probability of dropout for the fully connected layer(s).', default=1.0)
parser.add_argument('--weight_decay', type=float,
                    help='L2 weight regularization.', default=0.0)
parser.add_argument('--optimizer', type=str, choices=['ADAGRAD', 'ADADELTA', 'ADAM', 'RMSPROP', 'MOM'],
                    help='The optimization algorithm to use', default='ADAGRAD')
parser.add_argument('--learning_rate', type=float,
                    help='Initial learning rate. If set to a negative value a learning rate ' +
                    'schedule can be specified in the file "learning_rate_schedule.txt"', default=0.1)
parser.add_argument('--learning_rate_decay_epochs', type=int,
                    help='Number of epochs between learning rate decay.', default=100)
parser.add_argument('--learning_rate_decay_factor', type=float,
                    help='Learning rate decay factor.', default=1.0)
parser.add_argument('--moving_average_decay', type=float,
                    help='Exponential decay for tracking of training parameters.', default=0.9999)
parser.add_argument('--seed', type=int,
```

Figure 4.4.2 shows the `parse_arguments` function.

Moreover, we will set the number of batches per epoch to be 500. Furthermore, the dimensionality of the embedding is set to be 128. Random cropping and random horizontal flipping technique will be used. The keep probability of dropout for the fully connected layer is set to be 1. The L2 weight regularization is set to be 0. ADAGRAD will be used as the optimization algorithm. The learning rate is set to be 0.1. The number of epochs between learning rate decay is set to be 100. The learning rate decay factor is set to be 1.0. The exponential decay for tracking of training parameters is set to be 0.99.

```
parser.add_argument('--seed', type=int,
                    help='Random seed.', default=666)
parser.add_argument('--learning_rate_schedule_file', type=str,
                    help='File containing the learning rate schedule that is used when learning_rate is set to -1.',
                    default='C:/Users/tszch/Desktop/fyp/facenet/src/data2/learning_rate_schedule.txt')
```

Figure 4.4.3 shows the `parse_arguments` function

The random seed is set to be 666. We also need to set the path for the learning rate schedule file. Inside the schedule file, we will map an epoch number to a learning rate. When the epoch number are 0, 60, 80 and 91, the learning rate are 0.05, 0.005, 0.0005 and -1 respectively.

4.5 Train The FaceNet Model Using Triplet Loss Function

Now, after entering all the arguments, we can start running the `train_tripletloss.py` in the FaceNet package.

It will then create and name a file with the date-time format YYYY-MM-DD-HHMM in the log directory and model directory to record the log and model checkpoint. The model will get the input training dataset path from the argument and start the training using the inception_resnet pre-trained model. We can use the TensorBoard to monitor the training process. For example, we can observe if the loss is too low to avoid overfitting.

4.6 Generating Pairs

To validate the model, we do need the `pairs.txt`. It contains 36*30 number of matched pairs and 36*30 number of mismatched pairs. The meaning of matched pair is that the two different images with the same class. Mismatched pair means that the two different images with different classes.

```

def _generate_matches_pairs(self):
    """
    Generate all matches pairs
    """
    for name in os.listdir(self.data_dir):
        if name == ".DS_Store" or name[-3:] == 'txt':
            continue
        a = []
        for file in os.listdir(self.data_dir + name):
            if file == ".DS_Store":
                continue
            a.append(file)
        with open(self.pairs_filepath, "a") as f:
            temp = random.choice(a).split("_") # This line may vary depending on how your images are named.
            w = temp[0]
            l = random.choice(a).split("_")[1].lstrip("0").rstrip(self.img_ext)
            r = random.choice(a).split("_")[1].lstrip("0").rstrip(self.img_ext)
            f.write(w + "\t" + l + "\t" + r + "\n")

```

Figure 4.6.1 shows the generating match pairs function

The above Python code help generate the matched pairs by randomly selecting two images in the same class file. The class name, the position of the two images in the file will then be written in the text file. It follows the pattern 'class1, the position of the image in class 1 file, the position of another image in class 1 file' written in every row.

```

def _generate_mismatches_pairs(self):
    """
    Generate all mismatches pairs
    """
    for i, name in enumerate(os.listdir(self.data_dir)):
        if name == ".DS_Store" or name[-3:] == 'txt':
            continue
        remaining = os.listdir(self.data_dir)
        del remaining[i]
        remaining_remove_txt = remaining[:]
        for item in remaining:
            if item[-3:] == 'txt':
                remaining_remove_txt.remove(item)
        remaining = remaining_remove_txt
        other_dir = random.choice(remaining)
        with open(self.pairs_filepath, "a") as f:
            file1 = random.choice(os.listdir(self.data_dir + name))
            file2 = random.choice(os.listdir(self.data_dir + other_dir))
            f.write(name + "\t" + file1.split("_")[1].lstrip("0").rstrip(self.img_ext) \
                    + "\t" + other_dir + "\t" + file2.split("_")[1].lstrip("0").rstrip(self.img_ext) + "\n")

```

Figure 4.6.2 shows the generating mismatch pairs function

The above Python code help generate the mismatched pairs by randomly selecting two images in different class files. The class name, the position of the selected image in the files will then be written in the text file. It follows the pattern 'class1, the position of the image in class 1 file, class2, the position of the image in class 2 file' written in every row.


```

def generate(self):
    # The repeate times. You can edit this number by yourself
    folder_number = self.get_folder_numbers()
    # This step will generate the hearder for pair_list.txt, which contains
    # the number of classes and the repeate times of generate the pair
    if not os.path.exists(self.pairs_filepath):
        with open(self.pairs_filepath, "a") as f:
            f.write(str(self.repeat_times) + "\t" + str(folder_number) + "\n")
    for i in range(self.repeat_times):
        self._generate_matches_pairs()
        self._generate_mismatches_pairs()

```

Figure 4.6.3 shows the repeating function

The above Python code is to generate the number of matched pairs and mismatched pairs. We can set the repeat time. The exact number for matched pairs or mismatched pairs will be the repeat time times the number of classes. In our case, the exact number is the number of classes 36 times the number of repeat time 30, which is 1080. So, the sum of matched pairs plus mismatched pairs will be 1080 times two which is 2160.

4.7 Evaluate The Trained FaceNet Model

The FaceNet paper provides the following method to evaluate the model. Firstly, we will calculate the Euclidean distance d over all the pairs. Then, we will set a threshold t . When the distance d is less than or equal to t , the pairs are predicted as matched. Otherwise, it is predicted as mismatched. For each matched pair, if it is predicted as matched, the variable TP will be added by 1. If it is mispredicted, the variable FN will be added by 1. For each mismatched pair, if it is predicted correctly as mismatched, the variable TN will be added by 1. Otherwise, the variable FP will be added by 1. Now, we can use those variables to evaluate the trained model. The true positive rate can be calculated as:

$$TPR = TP / (TP + FN)$$

The false positive rate can be calculated as:

$$FPR = FP / (FP + TN)$$

The accuracy can be calculated as:

$$ACC = (TP + TN) / (TP + TN + FP + FN)$$

Besides the accuracy rate, the validation rate can also be used to evaluate the model. We will calculate the true accepts (TA) first. True accepts are the number of fin pairs correctly classified as the same dolphin using the threshold t . Then we also need to calculate the false accepts (FA). False accepts are the number of fin pairs incorrectly classified as the same dolphin using the threshold t .

The validation rate can be calculated as:

$$VAL = TA / \text{Number of pairs with the same identities}$$

The false accept rate can be calculated as:

$$FAR = FA / \text{Number of pairs with the different identities}$$

The above critical calculation is implemented in the `validate_on_lfw` Python file and included in the FaceNet package file. Therefore, we can enter the parsed argument to evaluate the model.

```
def parse_arguments(argv):
    parser = argparse.ArgumentParser()

    parser.add_argument('lfw_dir', type=str,
                        help='Path to the data directory containing aligned LFW face patches.', default='C:/Users/tszsch/Desktop/fyp/facenet/src/data2')
    parser.add_argument('--lfw_batch_size', type=int,
                        help='Number of images to process in a batch in the LFW test set.', default=16)
    parser.add_argument('model', type=str,
                        help='Could be either a directory containing the meta_file and ckpt_file or a model protobuf (.pb) file',
                        default='C:/Users/tszsch/Desktop/fyp/facenet/src/20210409-092741')
    parser.add_argument('--image_size', type=int,
                        help='Image size (height, width) in pixels.', default=160)
    parser.add_argument('--lfw_pairs', type=str,
                        help='The file containing the pairs to use for validation.', default='lfw/pairs.txt')
    parser.add_argument('--lfw_nrof_folds', type=int,
                        help='Number of folds to use for cross validation. Mainly used for testing.', default=10)
    parser.add_argument('--distance_metric', type=int,
                        help='Distance metric 0:euclidian, 1:cosine similarity.', default=0)
    parser.add_argument('--use_flipped_images',
                        help='Concatenates embeddings for the image and its horizontally flipped counterpart.', action='store_true')
    parser.add_argument('--subtract_mean',
                        help='Subtract feature mean before calculating distance.', action='store_true')
    parser.add_argument('--use_fixed_image_standardization',
                        help='Performs fixed standardization of images.', action='store_true')
    return parser.parse_args(argv)
```

Figure 4.7.1 shows the argument function for validating the model

Firstly, we need to provide the path for the testing dataset. Secondly, we need to define the number of images to process in a batch in the LFW test set to be 16. Thirdly, we also need to provide the path for the trained model. We will set the image size to be 160 pixels. We provide the path that links to the pairs to use for validation, as discussed in section 4.8. 10 folds will be used for cross-validation. Euclidian will be selected as the distance metric. Image and its horizontally flipped counterpart will be concatenated. Feature Mean will be subtracted before calculating distance. Fixed standardization of images will be performed. Also, we need to remember the validation dataset's image file name needed to be set in '0X_000i' format where the variable X is the class name and the variable i is the number of the position in that class. Then we can start running the program `validate_on_lfw.py`. The evaluation result of the two models trained in triplet loss and softmax function will be discussed in the next section, 'Testing procedure and results'.

4.8 Train A Classifier

Now, we have the trained models which are trained in the triplet loss function. We will use the model to train our classifier on our dataset. Firstly, we divide the dataset into training and testing datasets into a ratio of around 50% to 50%. The training dataset contains 10 classes as well as the testing dataset. It means that 50% of the images in each class are used for training and the other 50% are used for testing. The training dataset contains in total 78 images. The testing dataset contains in total 73 images. Then, we load the pre-trained model for feature extraction. Next, we will calculate those images' embeddings in the dataset. The calculated embeddings

will then be used to train the classifier by using the support vector machine algorithm. After training, the classification model will be saved as a python pickle.

Since the FaceNet package also provides the implementation of training a classifier using the support vector machine, we can edit the parse argument and run the classifier.py.

```
def parse_arguments(argv):
    parser = argparse.ArgumentParser()

    parser.add_argument('mode', type=str, choices=['TRAIN', 'CLASSIFY'],
                        help='Indicates if a new classifier should be trained or a classification ' +
                        'model should be used for classification', default='TRAIN')
    parser.add_argument('data_dir', type=str,
                        help='Path to the data directory containing aligned LFW face patches.', default='C:/Users/tszch/Desktop/fyp/facenet/src/data2')
    parser.add_argument('model', type=str,
                        help='Could be either a directory containing the meta_file and ckpt_file or a model protobuf (.pb) file',
                        default='C:/Users/tszch/Desktop/fyp/facenet/src/20210409-092741')
    parser.add_argument('classifier_filename',
                        help='Classifier model file name as a pickle (.pkl) file. ' +
                        'For training this is the output and for classification this is an input.',
                        default='C:/Users/tszch/Desktop/fyp/facenet/src/output/result2.pkl')
    parser.add_argument('--use_split_dataset',
                        help='Indicates that the dataset specified by data_dir should be split into a training and test set. ' +
                        'Otherwise a separate test set can be specified using the test_data_dir option.', action='store_true')
    parser.add_argument('--test_data_dir', type=str,
                        help='Path to the test data directory containing aligned images used for testing.')
    parser.add_argument('--batch_size', type=int,
                        help='Number of images to process in a batch.', default=15)
    parser.add_argument('--image_size', type=int,
                        help='Image size (height, width) in pixels.', default=160)
    parser.add_argument('--seed', type=int,
                        help='Random seed.', default=666)
    parser.add_argument('--min_nrof_images_per_class', type=int,
                        help='Only include classes with at least this number of images in the dataset', default=16)
    parser.add_argument('--nrof_train_images_per_class', type=int,
                        help='Use this number of images from each class for training and the rest for testing', default=1)
```

Figure 4.7.2 shows the argument function for training the classifier

Firstly, we need to state the mode as ‘TRAIN’. Secondly, we need to provide the path to the training dataset. Thirdly, we need to provide the path to the trained model. Fourthly, we need to provide the classifier model file name, which should be a pickle (.pkl) file. We will set the number of images to process in a batch to 15 as well. The image size in pixels is set to be 160. After running the classifier.py, we will get the following output:

```
Number of classes: 10
Number of images: 79
Loading feature extraction model
Model directory: C:/Users/tszch/Desktop/fyp/facenet/src/TripletModel
Metagraph file: model-20210711-005357.meta
Checkpoint file: model-20210711-005357.ckpt-4000

Calculating features for images
Training classifier
Saved classifier model to file "C:/Users/tszch/Desktop/fyp/facenet/src/output/result.pkl"
```

Figure 4.7.3 shows the output after successfully running the classifier.py

4.9 Evaluate The Classifier

After training the classifier using the support vector machine algorithm, we will use embeddings from the testing dataset to test the classifier. We just need to change the mode from ‘TRAIN’ to ‘CLASSIFY’ in the above argument function shown in figure 4.7.2. Then we can get the output shown below:

```

Number of classes: 10
Number of images: 73
Loading feature extraction model
Model directory: C:/Users/tszch/Desktop/fyp/facenet/src/TripletModel
Metagraph file: model-20210711-005357.meta
Checkpoint file: model-20210711-005357.ckpt-4000

```

Figure 4.9.1 shows the output after running the classifier.py

The results of the trained classifier will be discussed in the section ‘Testing procedure and results’.

4.10 Predict A New Input Image

Now we have a trained model and a trained classifier. We can use them to predict the class of a new input image. As the FaceNet package has already provided the predict.py, we can simply run the program to get the prediction. We will need to enter the parse arguments.

```

def parse_arguments(argv):
    parser = argparse.ArgumentParser()
    parser.add_argument('image_files', type=str, nargs='+', help='Path(s) of the image(s)',
default='C:/Users/tszch/Desktop/fyp/facenet/src/Test/01/01_0002.png')
    parser.add_argument('model', type=str,
        help='Could be either a directory containing the meta_file and ckpt_file or a model protobuf (.pb) file',
default='C:/Users/tszch/Desktop/fyp/facenet/src/20210601-154710/')
    parser.add_argument('classifier_filename',
        help='Classifier model file name as a pickle (.pkl) file. ' +
        'For training this is the output and for classification this is an input.' ,
default='C:/Users/tszch/Desktop/fyp/facenet/src/output/result2.pkl')
    parser.add_argument('--image_size', type=int,
        help='Image size (height, width) in pixels.', default=160)
    parser.add_argument('--seed', type=int,
        help='Random seed.', default=666)
    parser.add_argument('--margin', type=int,
        help='Margin for the crop around the bounding box (height, width) in pixels.', default=44)
    parser.add_argument('--gpu_memory_fraction', type=float,
        help='Upper bound on the amount of GPU memory that will be used by the process.', default=1.0)
    return parser.parse_args(argv)

```

Figure 4.10.1 shows the argument function for prediction

Firstly, we will need to provide the path of the image we want to predict. Secondly, we need to include the trained model’s path. Thirdly, we need to give the path for the classifier model. For example, now we want to predict the image 01_0002.png. We can run the program to get the output.

```

Loaded classifier model from file "C:/Users/tszch/Desktop/fyp/facenet/src/output/result.pkl"

people in image C:/Users/tszch/Desktop/fyp/facenet/src/Test/01/01_2.jpg :
01: 0.520

```

Figure 4.10.2 shows the output after running the predict.py successfully

From the above figure, we can see that by using the trained classifier, we can correctly predict the image’s class. In other words, we can correctly identify the dolphin. This will also be presented as the API because by calling this, the result can be output directly. Once the web-application system receives a new dolphin image, it will be called and do the prediction.

5. Testing Procedure And Results

This section will talk about the testing procedure and results of the fin identification by using the above methodology and implementation.

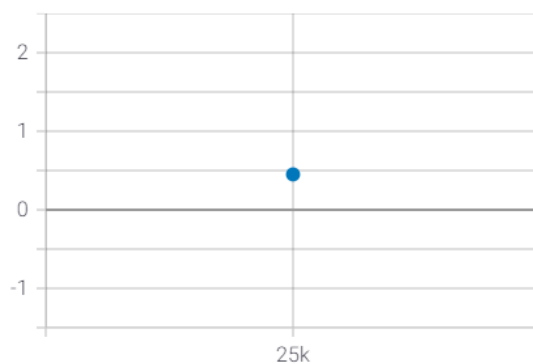
5.1 Testing Procedure Of The Fin Detector

Since the fin detector model may be over-trained or trained in a very short time, we need to observe the total loss value. It is good to allow the model to reach a total loss of at least 2 or ideally 1 and lower. However, the total loss cannot be too low. Otherwise, it will be result in overfitting.

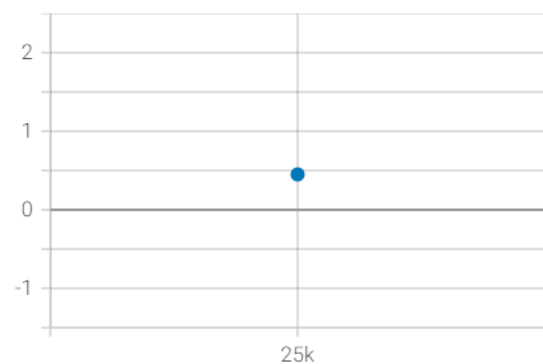
5.1.1 The Evaluation Result Of The Fin Detector

By using the pre-trained model Fast RCNN model, we have trained our fin detector. Then we can use the TensorBoard to look at the Detection Boxes Precision/mAP, classification loss values in the tf event files. The result is as follows.

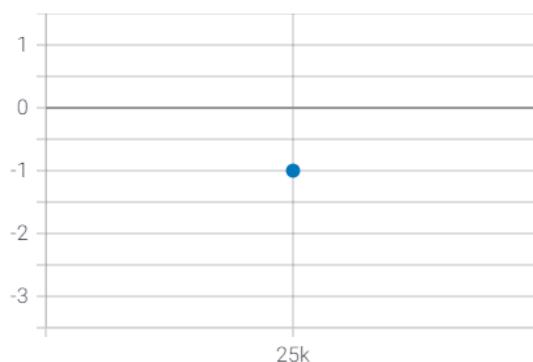
DetectionBoxes_Precision/mAP
tag: DetectionBoxes_Precision/mAP



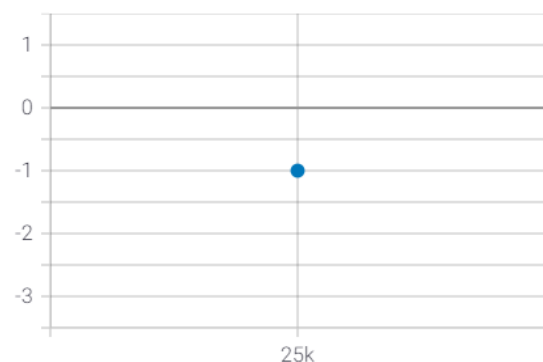
DetectionBoxes_Precision/mAP (large)
tag: DetectionBoxes_Precision/mAP (large)



DetectionBoxes_Precision/mAP (medium)
tag: DetectionBoxes_Precision/mAP (medium)



DetectionBoxes_Precision/mAP (small)
tag: DetectionBoxes_Precision/mAP (small)



The detection boxes precision/mAP is around 45%.

5.2 Testing Procedure Of The Trained FaceNet Model

As mentioned, the model may result in overfitting or underfitting. Therefore, we need to supervise the total loss carefully while training. The total loss is ideally under 1 but cannot be

too low. We can also set different period of time to train multiple models. After the training, we can then evaluate the model through accuracy, validation rate, area under curve and equal error rate values to see if it is overfitting or underfitting. In this way, we can get the optimal one.

5.2.1 The Evaluation Result Of The Trained FaceNet Model

As discussed, by running the `validate_on_lfw.py`, we can get the accuracy and validation rate of the model. Firstly, we have trained the model only by 2 hours. The result is as follows.

```
Accuracy: 0.58889+-0.02833
Validation rate: 0.13333+-0.02722 @ FAR=0.00000
Area Under Curve (AUC): 0.423
Equal Error Rate (EER): 0.567
```

Figure 5.2.1.1 shows the result of the model trained by 2 hours

We can see that the accuracy is not very good. It is around 59%. The validation rate is even lower only around 13%. Moreover, from the area under curve, we know that the model is underfitting. The AUC is only around 42%. It means that we can train the model in a longer time. The equal error rate is quite high. It is like 57%.

Secondly, we have trained another model by 12 hours. The result is as follows.

```
Accuracy: 0.70000+-0.02357
Validation rate: 0.21111+-0.04157 @ FAR=0.00000
Area Under Curve (AUC): 0.712
Equal Error Rate (EER): 0.350
```

Figure 5.2.1.2 shows the result of the model trained by 12 hours

This time, we trained the model in a longer time. We can see that the result is getting better. The accuracy is **70%** which is pretty decent and much better than the above model. It means that the model predicts quite well among all the pairs overall. However, the validation is only about 21%. When it comes to predicting the pairs of the same dolphin, the model cannot predict correctly. It causes a low validation rate. Nevertheless, it is still better than the above. When it comes to the AUC, it is around 71% which is way better than the above's 42%. It means that the performance of the classification model is getting better. However, the model is still underfitting. Since, when AUC is near 1, the higher the authenticity of the detection method. Therefore, we still can train the model in a longer time. When the AUC is near 1, we can stop training because otherwise it may result in overfitting. Lastly, we can see that the error rate is improved. The above 56% has dropped to 35%.

5.3 Testing Procedure Of The Trained Classifier

The optimal model will be used to train the classifier. As we have discussed in the above section, there are 14 classes in total. However only 10 classes will be used for training as well as for testing. Because the other 4 classes have too few images like only 3-4 images per class. The quality of the images is not good too. Therefore, 10 classes are selected for this part. The main component needed to be tested is the impact on dividing the ratio of the images in training and testing datasets. Firstly, we will divide the ratio into 50% to 50% between training and testing. That means 50% of the images in every class will be used for training and the other 50% will be used for testing. Secondly, we will divide the ratio of training and testing into 90% to 10%. Then we will compare the results and take the ratio that gives the best result.

5.3.1 The Evaluation Result Of The Trained Classifier for ratio 50% to 50%

As we have discussed, we divided the dataset into the training set and testing set into 50% to 50% first. The testing set contains 10 classes and 73 images. The training set contains 10 classes and 78 images. We have evaluated the trained classifier. The result is as follows.

```
Testing classifier
Loaded classifier model from file "C:/Users/tszch/Desktop/fyp/facenet/src/output/result.pkl"
 0  01: 0.666
 1  01: 0.617
 2  01: 0.680
 3  01: 0.639
 4  01: 0.666
 5  01: 0.567
 6  01: 0.722
 7  01: 0.686
 8  01: 0.588
 9  01: 0.588
10  02: 0.310
11  02: 0.491
12  02: 0.524
13  02: 0.573
14  02: 0.481
15  03: 0.515
16  03: 0.591
17  03: 0.515
18  03: 0.603
19  03: 0.623
20  03: 0.579
```

```
21 03: 0.520
22 04: 0.490
23 04: 0.582
24 04: 0.475
25 04: 0.510
26 04: 0.490
27 06: 0.627
28 06: 0.600
29 06: 0.611
30 06: 0.628
31 06: 0.600
32 06: 0.579
33 06: 0.584
34 06: 0.672
35 07: 0.594
36 07: 0.618
37 07: 0.544
38 07: 0.553
39 07: 0.587
40 07: 0.540
```

```
41 07: 0.527
42 07: 0.593
43 07: 0.603
44 02: 0.367
45 02: 0.312
46 09: 0.639
47 09: 0.729
48 09: 0.553
49 09: 0.751
50 09: 0.602
51 09: 0.722
52 09: 0.640
53 09: 0.595
54 09: 0.649
55 09: 0.794
56 09: 0.729
57 09: 0.713
58 09: 0.523
59 11: 0.441
60 11: 0.460
```

```
61 11: 0.494
62 11: 0.333
63 12: 0.655
64 12: 0.610
65 12: 0.625
66 12: 0.616
67 12: 0.554
68 12: 0.559
69 12: 0.441
70 12: 0.655
71 12: 0.663
72 12: 0.658
Accuracy: 0.973
```

Figure 5.3.1.1 shows the classification result

The leftmost column is the image number. The middle column is the predicted class name, and the rightmost column is the classification probability. For example, the second last row “72 12: 0.658” means that the image index number 72 is predicted as class 12 because class 12 has the highest classification probability around 65%. The last row shows the accuracy of the classifier is **97%** which is an excellent result. From the above result, only image index number 44 and 45 are predicted wrongly. They belong to class 08. However, they are wrongly predicted as class 02.



Figure 5.3.1.2 shows the image index 44 which belongs to class 08



Figure 5.3.1.3 shows the image index 12 which belongs to class 02

From the above figures, we can see that the images of class 08 and class 02 are pretty similar. They are both in color pink and have some dots in the fin. Therefore, it is difficult for the classifier to predict correctly. The other main issue is that for the class 08, there are only in total 4 images. The dataset is too less for the model to differentiate the difference between class 08 and class 02.

5.3.2 The Evaluation Result Of The Trained Classifier for ratio 90% to 10%

In this part, we divided the dataset into the training set and testing set into 50% to 50% first. The training set contains 10 classes and 131 images. The testing set contains 10 classes and 21 images. We have evaluated the trained classifier. The result is as follows.

```
Testing classifier
Loaded classifier model from file "C:/Users/tszch/Desktop/fyp/facenet/src/output/result.pkl"
 0 01: 0.822
 1 01: 0.730
 2 02: 0.542
 3 03: 0.699
 4 03: 0.660
 5 04: 0.672
 6 06: 0.722
 7 06: 0.695
 8 07: 0.655
 9 07: 0.751
10 08: 0.504
11 09: 0.718
12 09: 0.778
13 09: 0.692
14 09: 0.652
15 09: 0.727
16 09: 0.841
17 09: 0.797
18 11: 0.548
19 12: 0.831
20 12: 0.756
Accuracy: 1.000
```

Figure 5.3.2.1 shows the classification result in a ratio 90%:10%

From the above figure, we can see that the accuracy of dividing the training set and testing set in a ratio of 90%:10% is **100%**. Every testing image is predicted correctly to its belonging class. Image index number 10 is predicted correctly to 08 class not the wrong but similar class 02. It means that with more training data, we can predict a new input image much more accurately. To conclude, a 90% to 10% between training and testing ratio should be taken especially when the data images are too less so that we can have more training examples.

6 Conclusion

6.1 Critical review

At the beginning of this project, the process of building a fin detector was doing quite well. It is because we have a sufficient dataset for training the model. Moreover, we can just call the TensorFlow object detection API to finish the job. Then we are able to detect the fin part of the dolphin.

However, when it comes to identify the dolphin, the progress was very slow. It is because firstly, the data is very rare. It is hard to capture the complete fin part of the dolphin as the dolphin move in the water. After capturing, the ecologist also has to label an ID with it to identify it, making sure it does not collide with another one. The labeling and capturing process is time-consuming and not an easy job. This causes we only have a few data to train and test the model. By keep communicating with Dr.Brian, more and more data is gained to higher the accuracy of the model.

Secondly, it takes more time to figure out the direction. Since at first, I did not have related experience in doing identification, I have no idea to do the project. Fortunately, communicating with my supervisor, he has given me some very useful paper like dolphin-identification, FaceNet papers. I have more and more ideas and clear direction in doing it.

To conclude, this is a very good learning experience in learning how to overcome existing problems actively, how to communicate with others and have the spirit in seeking knowledge.

6.2 Summary of achievements

Firstly, I have successfully built up a fin detector which can be accurately detect the fin part of the dolphins. Secondly, the fin part can be successfully extracted out and saved with the corresponding class file. Thirdly, a fin classifier with decent accuracy has been successfully build up in identifying dolphins. Finally, an API is built up for the integration of the whole system.

6.3 Suggestions for extensions to the project

There are several ways we can improve the performance of our fin classifier.

Firstly, we do need more data. We have only 14 classes and in total 164 dolphin images for doing identification. Getting hundreds or even thousands of dolphins' images is essential for higher the accuracy of the model to get better classification result. Moreover, the distribution of images per class is imbalanced. For example, class 8 actually has only 4 images, but class 1 has 20 images. This will affect the classification result. Some individuals will hard to be identified due to the lack of data.

Secondly, after getting more data, we can fine-tune the FaceNet model to extract the fin features better and transform them into embeddings.

Thirdly, besides using the support vector machine algorithm to train the classifier, K-NN algorithm can be tested to see if it can get better performance.

7. Monthly Logs

September:

1. Have done online researches on looking into the basic understanding of what is machine learning, deep learning, the difference between face recognition and face verification. What machine learning techniques or algorithm can be used in this project.
2. Looking into previous Cityu students' final year project.
3. Trying to call Tensorflow object-detection API.

October:

1. Have read different academic papers including "Individual common dolphin identification via metric embedding learning" and "FaceNet: A Unified Embedding for Face Recognition and Clustering".
2. Have tried and successfully detected Chinese-white-dolphins and their fins by calling Tensorflow object-detection API.
3. Trying to implement face recognition by using Facenet model on Tensorflow.
4. Writing project plan.

November:

1. Trying to implement Face recognition by using different machine learning techniques including KNN and SVM algorithm and the facenet model.
2. Trying to use the face recognition technology on doing the dolphin recognition.
3. Writing interim report I.

December:

1. Trying to extract the dolphins' fins for the data set of the fin recognition.

January:

1. Trying to use some software like Darwin, Finbase to extract the dolphins' fins for the data set of the fin recognition.

February:

1. Trying to generate the embedding of the dolphins' fins.

March:

1. Successfully generated the embeddings of the dolphins' fins.
2. Trying to use K-NN algorithm to predict the fins without fine-tuning the FaceNet model.

April:

1. Fine-tuning the FaceNet model and train the model with triplet-loss function by using my own Chinese-White Dolphin dataset.

2. Training the support vector machine to be the classifier to classify the Chinese-White Dolphin-ID.
3. Looking into the softmax function for doing a comparison with the triplet-loss function.

May:

1. Writing interim report II.
2. Evaluating the trained FaceNet model.
3. Training the Support Vector Machine using the trained model.
4. Trying to predict a new CWD image using the trained SVM.

June:

1. Writing interim report II
2. Getting more dolphin data from Brian
3. Testing the accuracy after inputting more data
4. Reviewing interim report's comments from supervisor

July:

1. Writing final report
2. Final testing of the implementation
3. Preparing for the presentation
4. Preparing the demonstration

8. Reference

- [1] S. Bouma, M. D. M. Pawley, K. Hupman and A. Gilman, "Individual Common Dolphin Identification Via Metric Embedding Learning," 2018 International Conference on Image and Vision Computing New Zealand (IVCNZ), Auckland, New Zealand, 2018, pp. 1-6, doi: 10.1109/IVCNZ.2018.8634778.
- [2] H. Khalajzadeh, M. Mansouri, M. Teshnehlab (2014) Face Recognition Using Convolutional Neural Network and Simple Logistic Classifier. In: Snášel V., Krömer P., Köppen M., Schaefer G. (eds) Soft Computing in Industrial Applications. Advances in Intelligent Systems and Computing, vol 223. Springer, Cham. https://doi.org/10.1007/978-3-319-00930-8_18
- [3] Wang, Jie & Li, Zihao. (2018). Research on Face Recognition Based on CNN. IOP Conference Series: Earth and Environmental Science. 170. 032110. 10.1088/1755-1315/170/3/032110.
- [4] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, 3 2015, pp. 815–823.
- [5] Moindrot, O. (2018, March 19). Triplet Loss and Online Triplet Mining in TensorFlow. Retrieved October 25, 2020, from <https://omoindrot.github.io/triplet-loss>
- [6] Hupman, K. (2019, March 28). Know your dolphin by the fin, says NIWA scientist. Retrieved October 25, 2020, from <https://niwa.co.nz/news/know-your-dolphin-by-the-fin-says-niwa-scientist>
- [7] Ahdid, R., Safi, S., & Manaut, B. (2017). Euclidean and geodesic distance between a facial feature points in two-dimensional face recognition system. *Int. Arab J. Inf. Technol.*, 14, 565-571.
- [8] Harrison, O. (2019, July 14). Machine Learning Basics with the K-Nearest Neighbors Algorithm. Retrieved November 08, 2020, from <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>

[9] Gandhi, R. (2018, July 05). Support Vector Machine - Introduction to Machine Learning Algorithms. Retrieved November 08, 2020, from <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>

[10] Reddy, S. (2020, July 09). The intuition of Triplet Loss. Retrieved November 08, 2020, from <https://medium.com/analytics-vidhya/triplet-loss-b9da35be21b8>

[11] Craeymeersch, E. (2019, September 19). One Shot learning, Siamese networks and Triplet Loss with Keras. Retrieved November 08, 2020, from <https://medium.com/@crimy/one-shot-learning-siamese-networks-and-triplet-loss-with-keras-2885ed022352>