Parallel Massive Dataset Cleaning

Jianmeng Yu

4th Year Project Report

Computer Science

School of Informatics

University of Edinburgh

2018

Abstract

This project adopts the decision algorithm[1] developed by Pugh on a massive parallel scale. This aims to remove a large amount of False Positive fish detections in the Fish4Knowledge (F4K) dataset[2], without losing too many True Positives.

According to Qiqi Yu's estimated runtime[3], the cleaning process will take more than 1000 days to complete on a 40-core machine. Simply running the process on a parallel scale will not be sufficient, optimization of the code is also essential for making the processing more feasible.

This document describes the detail of various approaches to reduce unnecessary work during pre-processing and improve the cleaning algorithm. In this process, evaluation of efficiency for different implementations of machine learning techniques is used to reduce computational time cost.

After translating and optimizing the decision algorithm, the project distributed the task to 200 4-core machines and finishes the decision algorithm in 10 days. However, due to a mistake in previous work the cleaning does not achieve the expected accuracy, only 42% of the bad detections were removed instead of 90%, at the lost of 7% of good fishes. The cleaning process reduces the total data size by 27.8%, which saves about 600 GB of disk space.

A more detailed roadmap this project is provided in Chapter 1.

Acknowledgements

I would like to thank my project supervisor, Prof. Fisher, for his constant, patient support throughout the year. Without his expert knowledge in the field, it would be impossible for me to navigate through all of the data source and prior work of the Fish4Knowledge project.

I would also like to thank Mr Matthew Pugh for spending time answering my questions on the project and precious advice on the implementation of his algorithms.

I must also extend gratitude to my friends, and my family back in China, for all their help and encouragement during my study.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jianmeng Yu*)

Table of Contents

Chapter 1

Introduction

The main goal of this project is to produce a cleaned subset of a 1.6 TB dataset for future research purposes. And the main challenge of the project is to re-engineer the framework used to make it more scalable, which allows the project to finish the 800,000-hour task within a reasonable amount of time.

1.1 Fish4Knowledge Project (F4K)

The Fish4Knowledge (F4K) project, funded by EU's Seventh Framework Programme (FP7), studied ecological issues by analysing raw videos and extracting information from it, so researchers could use the data for studies without much programming skills.

The project acquired video data collected by Taiwan Ocean Research Institute. They set up 9 cameras in different coral reef areas in Taiwan such as Nanwan National Park (NPP-3), Lanyu, and Houbi Lake (HoBiHu). The project collected 5 years of recording, about 524,000 10-minute video clips with a total size of 91 TB. Approximately 1.4 billion fish detections found in the videos. We call this the F4K Original Data Set (FDS).

Attempting to reduce the dataset, the F4K project developed and applied a species recognition algorithm. This algorithm extracts all detections as 100x100 RGB images and corresponding description files, reducing the dataset to approximately 839 million detections, having a combined size of 1.6 TB This dataset is called Reduced FDS (RDS). A more detailed composition of these files is described in Chapter 3.

1.2 Project Motivation

In 2015, Pugh[1] developed a cleaning algorithm for RDS based on Huang's thesis[4], which would approximately remove 90% of the False Positives (objects that are not fish, recognized as fish), while only losing about 8% of True Positives (true fish detections).

However, due to lack of time and resource, the framework is not implemented and the cleaning was not applied to the full dataset. In 2016, Yu[3] added voting constraints on the cleaning algorithm, to both increase accuracy and reduce runtime. After evaluation, it's estimated this constraint did reduce the time cost of the algorithm for about 10%, at cost of 5% of accuracy.

Even after the reduction by Yu, it is evaluated the cleaning algorithm would still take 25,000 hours on a 40-core machine[3]. This means simply put the task on parallel would not be sufficient, the project need to distribute the task to more machines. For this purpose, the project uses the lab machines provided by the University of Edinburgh for cleaning. Even with about 800 cores available, the algorithm would still take more than 1,000 hours to finish, this the code should be also optimized for the project to be more scalable.

1.3 Project Outcome

The project manages to reduce the total runtime by 50% after translating the pipeline classifier developed by Pugh[1], and removing unnecessary operations in it. It is predicted that this classifier reduces the dataset by about 28%, with the following statistics:

	Predicted Fish	Predicted Non-Fish
True Fish	92.528%	7.472%
Non Fish	57.980%	42.020%

However, this does not meet the expectation from Pugh's thesis, whereas 90% of the False Positive is removed. This fault is caused by an extraction mistake in preprocessing stage, which causes the Convolutional Neural Network (CNN) over-fits on the training dataset. Re-training the CNN is not used due to the limit of time and resource.

1.4 Contribution

During this project, the parallel task distribution programme is based on a public GitHub repository "mpi-master-slave" created by user "luca-s"[5], some changes were made to the work queue and protocol for thrashing prevention and crash recovery, the pseudo-code of this framework is provided in Appendix A.

A data extraction pipeline was created in Python to partition, extract, and parse the raw SQL dump file to Comma Separated Values stored in plain text files, this removes the need of maintaining a SQL server and speeds up the extraction.

The pipeline classifier is also re-written in Python, and the unfinished part of the original pipeline is implemented. Due to the removal of the SQL server in the pipeline, the extraction and visualization MATLAB scripts created by Pugh are re-written into Python functions. Some metric algorithms were translated from MATLAB for loops to Numpy/Scipy operations to increase efficiency.

F4K project's feature extraction MATLAB code developed by Huang[4] is not translated and is called within Python using PyMatlab library. Some minor changes like replacing the edge extraction algorithm used and error handling were added to some of the unstable functions.

For validation of Pugh's classifier performance, a separate set of videos were ground-truthed. After testing the classifiers on this dataset, it's discovered that Pugh's classifiers over-fits on the training dataset. Also by inspecting the training code, it is found that the training set was heavily biased. A new complementary ground truth dataset was created for training new classifiers.

The classification step was originally going to use Pugh's trained Support Vector Machines (SVM) and Convolutional Neural Network (CNN). Due to the problem above, the SVM parameters used are re-calibrated for higher accuracy on the unseen dataset. The Python's `sklearn.svm.SVC` was used to achieve the same result instead of translation. The CNN implemented with Lua torch are rewritten and called with Lutorpy library. A fatal mistake was found in the CNN design, after evaluating the result with voting strategy, it is discovered the CNN trained were almost useless. Details about the fault and the re-training attempts were included in Section 5.3.

Reconstruction of the videos was not applied due to the low accuracy, a binary decision vector was generated instead.

1.5 Document Structure

Chapter 2 discusses the previous work and designs details used in the project.

Chapter 3 described the details of the data sources, storage and preprocessing used in the cleaning algorithm.

Chapter 4 describes the first stages of the cleaning: early detection removal, feature extraction, preprocessing for classification in the next stage.

Chapter 5 discusses the final classifiers used in the cleaning, with evaluation of the results and comparison between different algorithms.

Chapter 6 talks about the task distribution system used in this project and some of the difficulties and solutions.

Chapter 7 contains the conclusions and possible future work needed for the project.

Chapter 2

Background

2.1 Big Data and Distributed Computing

Big data is one of the hottest trending topics recently, where the amount of the data generated is not possible to be manually analysed. Different to the popular text stream analysis, this project is more focused on image processing of a large collection. There are already some image processing libraries with work distribution framework. For example: Hadoop Image Processing Interface[6], an Apache Spark based 4Quant[7] and other tool-kits for distributed parallelization.

Due to the limit of the project scale, the project could not use dedicated servers for cleaning the dataset. Instead, this project used the student lab machines provided by The University of Edinburgh. These machines have the Distributed Informatics Computing Environment (DICE) desktop installed, and using Andrew File System (AFS) for storage, this provides immense convenience on the project's need of fast and distributed I/O. Approximately 200-300 student lab DICE machines, a 1 TB and 256 GB disk space on AFS were used for this project.

The DICE machines used in the project do not have a shared memory. This means the project will also need tools for distribution of the task before being parallelized locally. Since a shared file system is already provided, the more standard and portable Message Passing Interface (MPI) is used for distribution of the task. More specifically, the MPI4PY[8], an MPI library designed for Python is used for this project. Details of the task distribution design used are described in Chapter 6.

2.2 Classification Schema

The reduction procedure of F4K project removed some of the False Positives from FDS. However, there are still a lot of False Positives in the RDS. To resolve this issue, a classification schema is created to identify the detections.

In the previous work of Pugh[1], ten different detection classes were used to ground truth the dataset, which is later used to train different classifiers used in the cleaning.

These 10 classes can be divided into 3 main categories (with examples in Fig 2.1):

- **I Not A Fish** - These detections are marked for removal in future.
  - **1 Compression Artefact** - During the process of recording video, some bits were dropped during transmission of the compressed video. These detections usually have rigid square shapes.
  - **2 Illumination Artefact** - Changes of brightness recognized as fish, they are usually refraction caused by turbid water, or light reflecting plankton.
  - **3 Background Vegetation** - Some of the videos are captured with dynamic backgrounds, where the swaying plants are recognized as fish.
  - **4 Others** - Everything else, this includes large floating matter, empty contours created by faults in previous algorithms.
  - **5 Unknown** - Due to issues like lighting, blurry and stretched video frames, it's uncertain the detection is fish or not.
- **II A Fish** - These frames are useful for future researchers.
  - **6 Good Boundary** - With clear ocean as background, these fish have good boundaries, and are useful for future species recognition.
  - **7 Partial Fish** - Mostly good detection boundary, but part of the fish is cut-off for various reasons:
    - **i** Fishes cut by frame boundaries.
    - **ii** Fishes are covered by vegetation or other fishes.
    - **iii** The fish is too big and cropped by the 100x100 boundary.
  - **8 Bad Boundary** - The fish is clearly captured, but the boundary extracted is erratic and useless for research.
- **III A Fish, but not useful** - These frames detections are true fish, but misleading information may be extracted. It's unsure these frames should be kept or not.

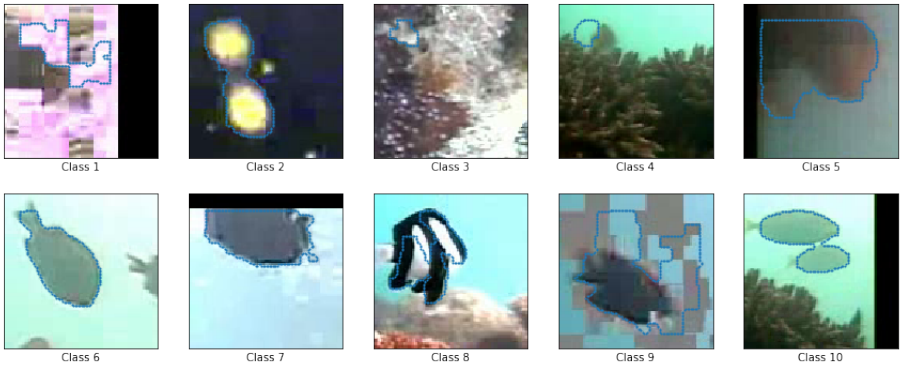

Figure 2.1: Example Detections From Each Class

9 **Other Errors** - like compression artefact are found in the image.

10 **Multiple Fish** - with shared contour.

However, this classification schema was not good enough for evaluating the accuracy of the classifiers due to the similarity between the classes, this limitation and the possible improvements are described in Section 5.1.

2.3 Pipeline Classifier

The main part of the project is to translate and apply the pipeline classifier, designed by Pugh[1]. However, under limitations, the pipeline itself could not be applied directly on a parallel scale in this project.

The first limitation is the SQL database, which stores the track and contour information of the image. However, the SQL database is too large and is estimated to be slow for the project. To make the extraction more sensible, a Python script was used to partition the raw `.SQL` file, this removed the need for a SQL database server. More details of this modification are in Section 3.2.

In Pugh's thesis[1], a Frame Edge Indicator Function (FEIF) is used to directly reduce the number of frames that need to be classified. After improvements and some new additions on dataset reduction, pre-processing cleans out about 20% of the detections. More details on the reduction are in Chapter 3.

Before sending the data into the classifiers, preprocessing is needed to give a more sensible result. The project uses feature extraction code from both Huang's[4] and

Pugh's[1] work, normalizing and transforming them with Principal Component Analysis (PCA). After extracting and reducing the features, they are fed into 3 Convolutional Neural Networks (CNN) and 10-class Support Vector Machine (SVM) to obtain the predicted probability of each class from each classifier. Where a voting strategy was used to combine the results from the classifiers into the final decision array.

Fig 2.2 shows the steps used in the original pipeline classifier designed by Pugh. Note that in this project, the SVM was changed to a single Multi-class SVM and the Top-N Decision was replaced by a voting strategy. See Chapter 5 for more details.

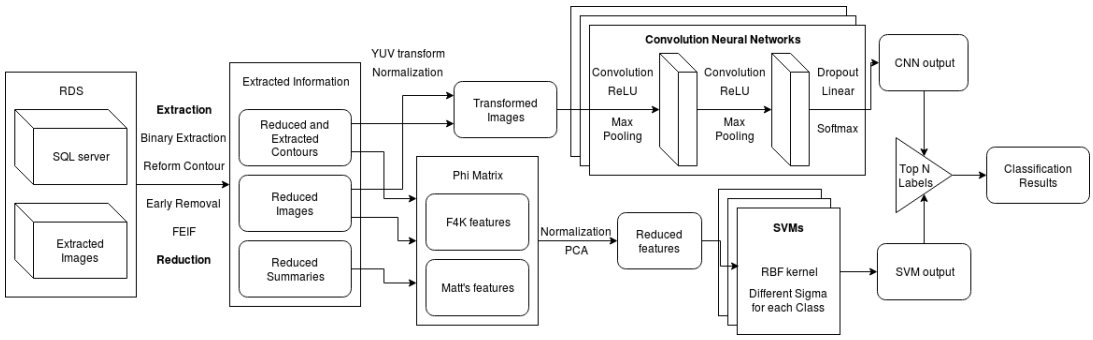

Figure 2.2: Pipeline Classifier designed by Pugh, modified

2.4 Yu's Voting Constraints

In 2016, Yu[3] tried to add a voting constraint to the Pipeline Classifier. Yu evaluated her 3 voting methods on the results obtained by Pugh and tested it against the obtained result of the Top-N method. Yu's evaluation discovered that this method would reduce the total runtime by 10%. However, this process decreases the True Positive Rate (TPR) by 5% and is not considered useful for the project.

After further evaluation of both Pugh and Yu's work, it is discovered that Pugh's final classifier over-fits on the training dataset due to deep net coverage, and mistakes in preprocessing steps. Details of this problem are discussed in Section 3.3 and Section 4.6.

To fully utilise the results from the over-fitted classifiers, a voting approach similar to Yu's work was tested and applied. The final result will be obtained by voting of 4 classifiers, instead of using the Top-N function. Detail of the voting strategy is included in Chapter 5.4.

Chapter 3

Data Source

The species recognition of the F4K project provided three types of output files:

- Extracted 100x100 RGB images, compiled into `.avi` video file.
- Corresponding summary of the video, recording detection id and bounding box sizes. Stored in Comma Separated Values format, as `.txt` file.
- A `.sql` dump file of 500GB, from the database used for species extraction.

In the species recognition process, a `video_id` is generated for every 400,000 videos, this consists of a 32 byte hash of video, a `#`, and the filming date in `YYYYMMDDhhmm` format. Where each of the `video_id` have a corresponding `.avi` and `.txt` file.

3.1 Extracted Images

The species recognition in F4K project extracted every detection with `w` and `h` both smaller than 90. This process is illustrated in Figure 3.1, where a 100x100 area is selected with top left corner coordinates `(w-10)` and `(h-10)` and cropped from the image. If the some of the selected areas is out of the frame, it will be filled with black pixels.

During this process, the contour and the bounding box of the fish were also calculated. The bounding box consists of 4 values `x,y,w,h`, where `x` and `y` are the coordinate of the top left corner, `w` and `h` are the width and height of the bounding box. Those cropped images are then stored in file `summary_(video_id).avi`, with detection id and `w` and `h` stored in corresponding `frame_info_(video_id).txt`.

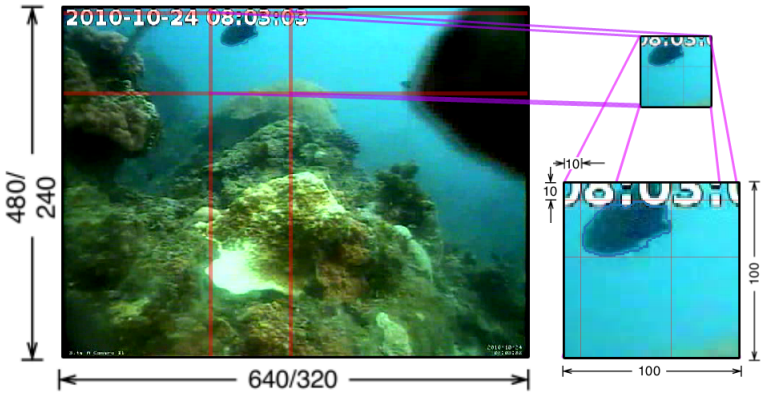

Figure 3.1: Process of Extracting Image

There are a total of 396,901 of such videos, consist of 839,465,846 frames, with a total size of 1.14 TB. Figure 3.2 shows a example of these videos. Note that the corresponding frame is in "`detection_id, width, height, x&y`" format, the "`x&y`" field always have a value of 10 for all extracted detections.

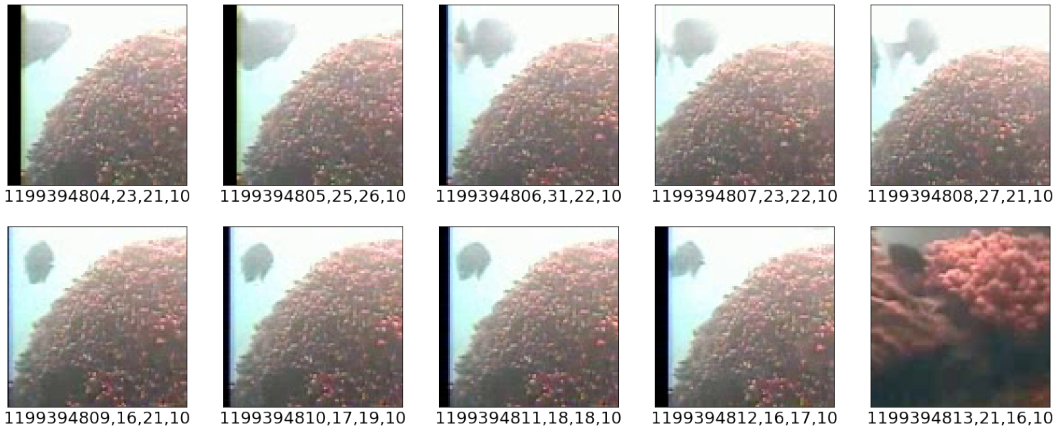

Figure 3.2: First 10 frame of a "`.avi`" file, with corresponding entry in "`.txt`" file.

Since the summary text file does not contain information about the bounding box of the detection, relevant data extraction from the `.sql` dump file is needed.

3.2 SQL dump file

The `.sql` dump file comes from the SQL workflow used in the F4K project, which means not the only details of fish detection are stored, other irrelevant components like user logs are also stored inside.

In this project, only the “Fish Detection” and “Camera” table is needed for the cleaning, hence extraction of the relevant information might be needed before the cleaning. For example, below is the schema of the “Fish Detection” table.

```
CREATE TABLE `fish_detection` (
`detection_id` int(11) NOT NULL AUTO_INCREMENT,
`fish_id` int(11) NOT NULL,
`video_id` char(45) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
`frame_id` mediumint(9) NOT NULL DEFAULT '0',
`timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
`bb_cc` blob NOT NULL,
`detection_certainty` float DEFAULT NULL,
`tracking_certainty` float DEFAULT NULL,
`component_id` smallint(6) NOT NULL )
```

This `.sql` dump file is stored as plain text files that could be loaded with a SQL server. Under limitations of disk space and access speed, loading such large SQL database dump file into a server and performing 400,000 queries is unnecessary and very time-consuming, hence making it the slowest part of the cleaning. A Python script with standard stream pipeline is used to parse and partition the SQL dump file into directly usable files instead.

3.2.1 Standard Stream Based Extraction Script

In this project, each record needed for the cleaning is independent (given they have different video_id). If each detection is stored in a corresponding file with its video_id, the information extraction will be much faster without the need to seek through all the records of other videos.

The project uses a standard stream pipeline because each detection is only needed once during the extraction, and the dump file is too large to be loaded into the RAM.

With simple Python functions such as `split()`, the records in the dump file of form:

```
INSERT INTO `fish_detection` VALUES (1),(2),(3),(4)...(N)
```

These are parsed into directly usable lists of values, separated by a `newline` character. This extraction procedure reduces the time cost for loading the dataset reduced to an almost negligible amount.

The output of this script contains 326 GB of `.csv` files, each one corresponds to the associated `video_id`. Some example lines of parsed `.csv` files is shown in the table below, the `bb_cc` field is removed as it containing binary data.

detection_id	fish_id	video_id	frame_id	timestamp	bb_cc	det_cert	trk_cert	component_id
1199394804	100550034	'0000e3df18698b393e9b3b8703138d6a#201011231100'	870	'2013-07-10 12:42:59'	BLOB	-1	-1	135
1199394805	100550034	'0000e3df18698b393e9b3b8703138d6a#201011231100'	871	'2013-07-10 12:42:59'	BLOB	-1	-1	135
1199394806	100550034	'0000e3df18698b393e9b3b8703138d6a#201011231100'	872	'2013-07-10 12:42:59'	BLOB	-1	-1	135
1199394807	100550034	'0000e3df18698b393e9b3b8703138d6a#201011231100'	873	'2013-07-10 12:42:59'	BLOB	-1	-1	135
1199394808	100550034	'0000e3df18698b393e9b3b8703138d6a#201011231100'	874	'2013-07-10 12:42:59'	BLOB	-1	-1	135
1199394809	100550062	'0000e3df18698b393e9b3b8703138d6a#201011231100'	890	'2013-07-10 12:42:59'	BLOB	-1	-1	135
1199394810	100550062	'0000e3df18698b393e9b3b8703138d6a#201011231100'	891	'2013-07-10 12:42:59'	BLOB	-1	-1	135
1199394811	100550062	'0000e3df18698b393e9b3b8703138d6a#201011231100'	892	'2013-07-10 12:42:59'	BLOB	-1	-1	135
1199394812	100550062	'0000e3df18698b393e9b3b8703138d6a#201011231100'	893	'2013-07-10 12:42:59'	BLOB	-1	-1	135
1199394813	100550129	'0000e3df18698b393e9b3b8703138d6a#201011231100'	1095	'2013-07-10 12:42:59'	BLOB	-1	-1	135

3.2.2 Translation of Binary Data

With the above extraction, another problem arises. In the schema mentioned in section 3.2, there is a column called `bb_cc`, which means "Bounding Box Chain Code". This column containing the chain code is used to store the fish boundary data in a more compact format.

Since the binary file is stored as a text file, a different encoding is used so it won't cause a parsing fault during loading. For example, the `null` character consists of 8 0-bits, are stored as two bytes in ASCII format of `''\0''`. A cleaning function is implemented to enumerate through the raw bit array to translate them back to original values.

After comparing binary values of `bb_cc` and the corresponding detection image, it was found that the binary data is in the following format:

- **First 42 (11,11,10,10) bits** - `x,y,w,h` coordinates of the bounding box.
- **Next 11 bits** - The x-coordinate of the first contour point.
- **Following 3 bits** - Padding of zeros added to the end of chain code below.
- **All other bits** - Chain Code of the contour, 3 bits each. Pointing towards next contour point from previous one.

Raw Binary Data				00000000111000001101000000010001000001000100000001101011000000000000000000000000010100010010010001001001001 00110100110110101001001001001001000111001001001001001011010111011011011011011011011011111111111000111000		
X	Y	W	H	First X	Padding	Translated Chain Code
7	52	17	17	13	3	00000000121122222223233244444434444446656666666667777707

Figures 3.3, 3.4, 3.5, and the above table shows the steps of transforming binary blob data into contours of the detection. Note that the Y-axis is reversed due to the data format used in the species extraction algorithm.

The chain-code start at the red point (First-X,Y) in Fig 3.3. After applying the direction instructions provided in Fig 3.4, the path draws the contour on the image as in Fig 3.5.

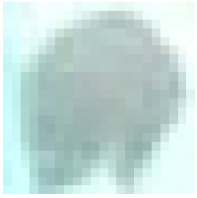

Figure 3.3: Original Image

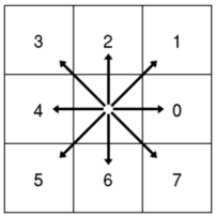

Figure 3.4: Transform Rules[9]

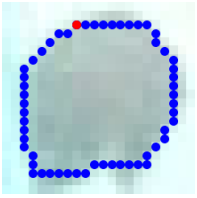

Figure 3.5: Contour Point

This process allows a faster extraction of the contour, which accelerates the dataset loading process so it takes significantly less time. By comparing against Yu's[3] evaluation, this potentially reduces the runtime for about 50%, and more importantly, removes the need for a SQL server, hence gives the cleaning process more portability.

3.3 Ground Truth Dataset

To train and evaluate the classifiers, Pugh and Yu marked a set of detections with the schema in Section 2.2. They chose a subset of the RDS of videos having id start with "13b". This subset consists of 39 video files and 61,101 detections.

However, the marking was not able to cover all of the detections because of its size. Some set of the detections are marked incorrectly because of the ambiguity of the classification schema. For example, a lot of the plankton detections are marked as fish. Also, since class 9 (good detection with problems) is very rare, it's sometimes mistaken as class 5 (Unknown) or 7 (bad boundary). Some of the error patterns are not included in this video set. This limitation is discussed in Section 5.1.

After extracting statistics about this dataset, it is discovered that this dataset is heavily biased. It fails to include normal videos filmed at 2 sites. The distribution of detection on each site is shown in Fig 3.6, note that the detections marked at HoBiHu-site are significantly lower than other sites, and they contain almost no good detections.

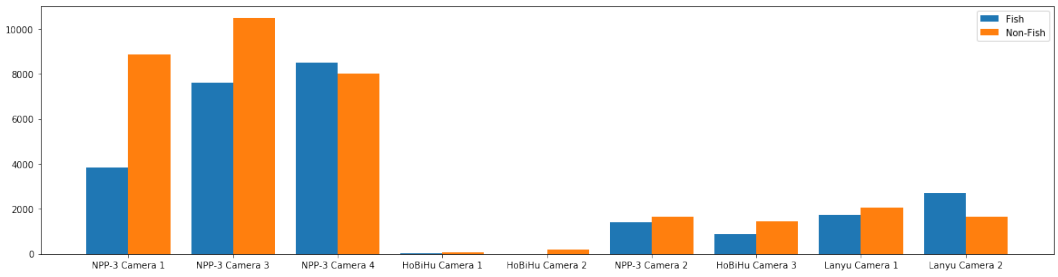

Figure 3.6: Number of Fish/Non-Fish at different sites in original Ground Truth Dataset

3.3.1 Sample Images at Different Sites

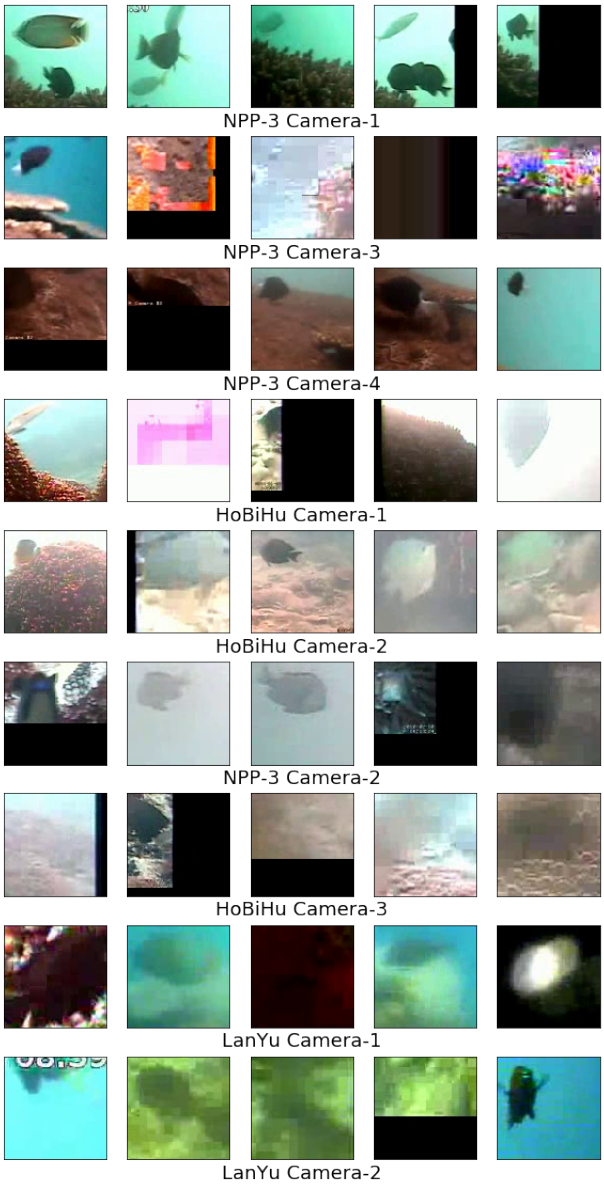

Figure 3.7: Sample Images of Detection in RDS at each site

3.3.2 Additional Ground Truth Dataset

Due to the need for re-calibration and future validation. Two additional subsets of videos, containing a total of 18,242 detections were marked. This includes:

- **Complementary Training Dataset**, containing 9,287 detections of detection.
- **Validation Dataset (site)** - 7,955 detections, having about 1,000 per site.
- **Validation Dataset (random)** - Randomly picked 1,000 detections across RDS.

Chapter 4

Preprocessing

To achieve the goal of removing False Positives without losing too many True Positives, as well as reducing the total runtime, some dataset reduction methods are used on the dataset before extraction of the features:

- **Plankton Removal** - Mark all detections recorded during night as Non-Fish.
- **Frame Edge Indicator Function**, originally developed by Pugh[1] - Marking detections with a percentage of contour points touching the edge as Non-Fish.

The next part of the pipeline will be the preprocessing stages:

- **Feature extraction** and **dimensionality reduction** for the SVM classifier.
- **Colour space transformation** and **normalization** for the images CNN used.

4.1 Plankton Removal (PR)

During the feature extraction tests, it was discovered that loading a 40,000 frame video and extracting features from it would use about 8 GB of memory space. And if such video is processed on a node with RAM less than 8 GB, it will cause thrashing problems, rendering the machine unresponsive. Even on machines with 16 GB memory, it could still interrupt other student's work.

While risking the chance of thrashing, these videos will take a longer time to process, and most importantly, they are usually filled with Non-Fish detections. As discussed in Chapter 3.1, if a camera recorded 30,000 detections in 10 minutes, means that in every

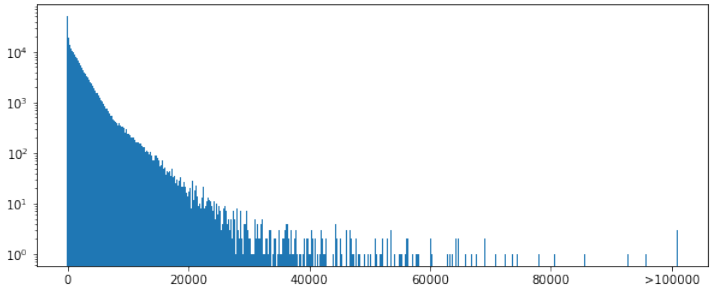

Figure 4.1: Log Scale Histogram of Detection Length

frame of the original video, an average of 10 detections is extracted. Fig 4.1 shows the Log Distribution of the videos with different length, note that only a few videos contain more than 30,000 detections. By looking at these "outlier" videos, some patterns were found (Images of these cases are in Appendix B):

- Both cameras at Lanyu site are night-vision cameras. When they film during the night, a lot of light is reflected from planktons and small animals close to the camera are recognised as fishes. Videos filmed during the night have an average of 6974 detections. 5% of the total detections come from such videos. With a high False Positive rate, these videos can be safely excluded from cleaning.

- Videos full of compression/transmission errors mostly happened at NPP-3 site camera 2 during June 2012 to August 2012. The camera falls down and changed angles every few days. Even if there are no such errors, most of the detections are from moving background vegetation.

- One outlier video had 200,000 detections, consisting of lots of repeating frames, possibly caused by bugs in previous extraction processes.

There are also some good videos with high detections:

- Videos from NPP-3 site camera 3, at January 2010. These videos are captured at a higher frame rate, resulting in more good detections.

- Dynamic background - Videos filled with moving vegetation, or refraction of sunlight. They usually contain lots of good detections.

If we remove all the videos recorded in the night, videos with 40,000 or more frames, and videos recorded with above characteristics and 20,000 or more frames, then about 4.6% of the detections can be rejected without the need to extract them.

4.2 Frame Edge Indicator Function (FEIF)

In Pugh's thesis[1], the FEIF is used to identify if a fish is being partially cut by the frame. In FEIF, the boundary of the video like in Fig 4.2 is used, the time-stamp zone comes from the reject area of the previous species recognition algorithm. If the number of the contour points inside this zone exceeds 25, the detection is then rejected.

However, this function could not achieve the intention in some cases. For example, some videos have a darker frame edge, so even if a fish is cut by the boundary, it could still be accepted. Also, a large fish slightly touching the boundary will be rejected because the 25 limit and small fishes may bypass the heuristics because of the size.

The following modification is added to solve the problem. By padding the boundary with a 2 pixels width, then increasing the limit of 25 to 40, and adding a new restriction: reject all the detections with 25% of the points touching the boundary.

Figure 4.2: Removed Areas (highlighted with red colour) of the modified FEIF

4.3 Remove Rate of Reduction Algorithms

After testing the dataset reduction function on training dataset, the accuracy statistics below were obtained. This manages to remove about 40% of the bad detections, causing about 5-10% of False Negatives.

It was later discovered that there was a significant amount of fish are rejected on videos without the time stamp. Cause of the fault are discussed in Section 7.3.

	PR and FEIF kept	PR and FEIF reject
True Fish	3060	152
Non Fish	3303	1440

4.4 MATLAB code translation

Initially, the project aimed to translate the entire pipeline into Python, however when it came to the feature extraction part, translating the code became an unreasonable solution.

The F4K feature extraction code has about 5,000 lines of code in MATLAB. Usually, for most of the MATLAB functions, an equivalent library function from Numpy/Scipy/SKLearn could be found. Unfortunately, most of F4K feature extraction functions developed by Huang[4] consists of customized code that could not be directly translated to Python.

For example, the Gabor Filter used in the project was written by Ahmad Poursaberi from Tehran University, a different implementation (where the scale of the filter is rotated, while their variance isn't) is used. This essentially required the project to rewrite the whole F4K feature library. A full translation of the F4K feature could take a few weeks.

Since the cleaning algorithm only needs to execute once, translation may not be the optimal path to take. For this purpose, some benchmarking were tested to see if translating could reduce enough runtime on the project.

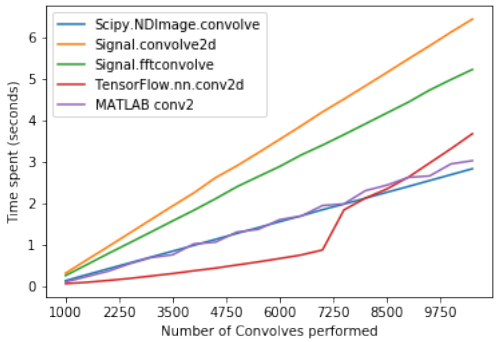

Figure 4.3: Test runtime of different 2D convolve algorithms.

Figure 4.3 shows the tested performance of various 2D convolution algorithm. During the test, 2000 random 100x100 arrays are generated to simulate images used, and 5 random 5x5 filters are used, simulating both the Gabor Filter and CNN used in the pipeline. By enforcing the maximum computational thread to 1, the result shows that only 2 of the chosen algorithms in Python are faster than MATLAB.

- **TensorFlow's conv2d**, while faster than all other libraries, it has a significantly

higher memory usage due to the use of tensor data type. Translation using this would be difficult due to the library being focused on Neural Networks.

- **Scipy.NDImage's convolve**, giving an similar performance as MATLAB.

After this analysis, it's clear that re-engineering the MATLAB code is likely to spend more time, so a library called PyMatlab will be used to compute the F4K features in a MATLAB session instead, while other unfinished parts of the pipeline will be translated to Python.

4.5 Feature Extraction

During the species recognition stage of the F4K project, 2626 features were used for computing the detection certainty. On that basis, Pugh added 29 new features focused on the edges of the contour. After that, dimensionality reduction is applied to the features to reduce dimension from 2655 to 88.

This process takes about 0.3 seconds for each frame, Analysing the computational time shows the following result, sorted by time cost:

- **Co-occurrence Matrix** - these 720 features took about 0.2 seconds to compute.
- **Affine Moment Invariants** - these 105 features took 0.05 seconds to compute.
- **Gabor Filter** - these 160 features took about 0.04 seconds to compute.
- Other features took a negligible amount of time to finish.

Unfortunately, after checking the features with PCA, these feature all took a significant part in the first 50 PCA components, removing any one of these time costly feature will have a high impact on the result.

4.5.1 Huang's Features

Pugh's work was based on the previous work of Huang's[4], where his 2626 features were used to identify the species of a fish. Since this part is not translated, only a list of used feature are provided here:

- **RGB and HSV histograms** - 930 features.
- **Curve Tail Shape, Ratio** - 2 features.
- **Fish Density Static** - 12 features.

- **Co-occurance Matrix** - 720 features.
- **Moment Invariant** - 42 features.
- **Pyramid Histogram of Gradient** - 680 features.
- **Fourier Descriptor** - 15 features.
- **Gabor Filter on Textures** - 160 features.
- **Affine Moment Invariants** - 63 features.
- **Tail/Head Contour Point Ratio** - 2 features.

4.5.2 Pugh's Features

Pugh's part of the generated features consists of 4 parts: Animation Score, Boundary Curvature, Erraticity, and Gabor Filter on contours. This part of the pipeline is translated into Python as the original feature generation script is incomplete. Some inefficient `for` loops were translated using the Numpy library for maximal single-core performance.

- **Animation Score (AS)** is calculated for one whole track, where 5 frames are picked from the track and squared sum of the change in pixels is calculated. This feature isn't very useful because of the dynamic background of the image.
- **Boundary Curvature (BC)**, uses the Curvature Scale Space to measure the change of direction to the contour, the result is blurred with a Gaussian filter and the Skewness and Kurtosis is extracted.
- **Temporal Consistency (Erraticity)** is similar to Boundary Curvature, where 5 frames are picked and mean of Skewness and Kurtosis is recorded.
- **Gabor Filters** applied on binary images generated using contour, instead of the original image used in the F4K feature extraction.

4.5.3 Principle Component Analysis

Originally in Pugh's design, the number of the Principle Components used were selected with Kaiser's Criterion, where only ones with an eigenvalue greater than 1 are selected. However, the only advantage with this criterion is it is easy to calculate, essentially throwing away 30% of the information. This loss could be directly observed from Fig 4.4.

Also due to the storage limit of the project, about 100 features per frame could be stored. With this problem, the first 88 Principle Components were used, this expresses 90% of the variance of the training dataset.

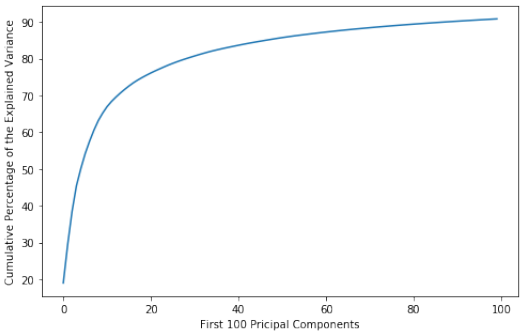

Figure 4.4: Cumulative Variance Explained against Number of Features Used

4.6 Image Processing For CNN

In Pugh's pipeline classifier, 3 different CNNs were used on the different types of preprocessed images. In `CNN_N`, the normal image is used, in `CNN_WC` and `CNN_BC`, a masked image is used, filling the pixels outside the contour with white and black respectively. For the BC and WC images, the detection is then moved to the centre of the image.

Before the image is input into the CNN, transformation and normalization is performed. Firstly the image is transformed into YUV colour space. Then the image is normalized globally with the mean and standard deviation of the YUV images. Finally, local spatial normalization was applied to each channel. This process is shown in Fig 4.5.

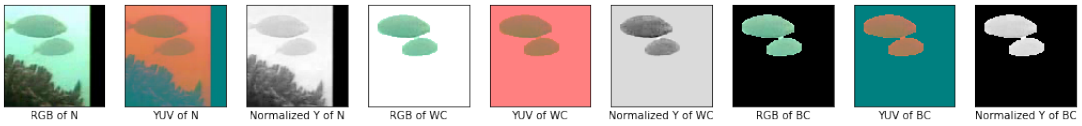

Figure 4.5: Images on first stages of pre-process

After tests on the trained CNNs, the results obtained weren't even close to the ones Pugh obtained. It was discovered that during the preprocessing stage of Pugh, two major mistakes were made due to Pugh's usage of OpenCV's `imwrite` for the extraction of the image.

- The images are stored in compressed ``.jpeg'' format, this leads to a drastic change in the result matrix, due to the local normalization step.
- Another fatal mistake is the images stored with OpenCV are in BGR space, where Lua recognize it as an RGB image.

This fault causes the CNN_N and CNN_BC to be almost useless, classifying a majority of the unseen data into class 2 and 7. Some re-train attempts were included in Section 5.3.

Half of the cleaning time cost comes from the image normalization of the CNN in the original pipeline design. With optimizations applied at the translation stage, the time cost for the pre-processing is drastically reduced to 3 days. This reduction is done by removing of SQL server and translated MATLAB normalization code in Python.

Chapter 5

Classification

5.1 Ambiguity in Original Classification Schema

For the Pipeline Classifier, 10 different classes of detections were proposed in Section 2.2, but there are many limitations on this schema. For example, some cases could be appearing simultaneously. Detection could have an erratic boundary, and also touching the boundary of a frame. Pugh's thesis didn't state clearly the different priorities among classes used.

After looking through the previously marked ground truth data, it is also found that some classes are very similar to others. For example in Fig 5.1, the track has a detection contour that is progressively less erratic. It makes it hard to draw an exact boundary between different classes when classifying manually.

Figure 5.1: Detection changing from class 8 (Bad Boundary) to 6 (Clean Boundary).

Another example of this is shown in Fig 5.2, where initially it is unknown whether the detection is a fish or not given the single frame. This is more problematic because the extraction intends to throw away class 5 while keeping class 6 detections.

Similar problem have also occurred between $classes \in [2,3,4,5,8,6]$. Where class 2,

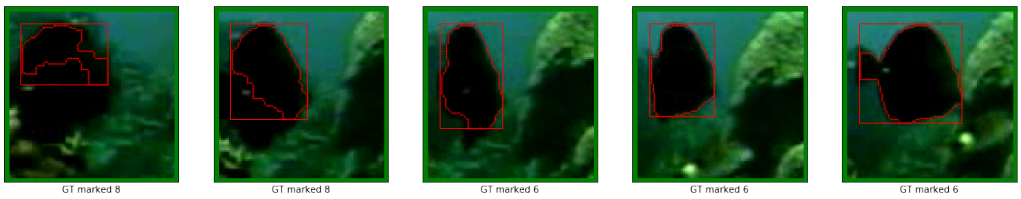

Figure 5.2: A track of detection, changing from class 5 (Unknown) to 6 (Fish).

3, and 4 generally means “I’m not sure what this is, but it’s definitely not a fish”, and on the other side, class 6 represents fish with good detection boundary. Also, any one of the detections might suddenly change to class 7 (partial fish visible) just because they are close to the frame edge.

For the cases of class 2 and 3, they are errors captured by the F4K species extraction’s background removal issues, caused by illumination and background vegetation respectively. During the ground truth processing of new datasets, it is found that distinguishing between these two classes is almost impossible. This indicates that confusion matrix might not be a good indicator whether a classifier performs well or not. In fact, if the classifier performs too well, it would indicate the classifier over-fits on the training dataset instead.

The original goal of the project is to remove non-fish detections from the dataset, hence only fish marked with class 6 and 8 are kept. It is more tolerable for “fish-like” classes like class 9 (Fish with Error) and class 7 (Partial visible) to be classified as fish. On the other hand, class 1 (Compression Fault) classified as class 6 (Good Fish) needed to be penalized heavily.

With this addition to accuracy metrics, the SVM and CNN developed by Pugh was re-evaluated, and unfortunately, both needed rework to work properly again.

5.2 Support Vector Machines (SVM)

In the original design, Pugh used ten different SVM classifiers with RBF kernel (Radial Basis Function kernel) for estimating probability of each class. Each of the SVMs has a parameter setting of:

1. In the Kernel Function $K(x,x') = exp(-\gamma\|x-x'\|^2)$, a γ value around 4-5.
2. Soft Margin Parameter C set to 1.

Translating the SVM design to python is trivial, the package `sklean.svm` provides a easy-to-use implement of the SVM, with only the need to fit the dataset again. The only different between MATLAB and Python's SVM is the parameter γ used, Python use σ for rbf kernel. A simple translation could be used to find the corresponding σ value, using the formula $\gamma = 1/2\sigma^2$.

In Pugh's design, due to the limitations of an incomplete training dataset, the result is heavily biased. After testing on unseen data, it classifies almost every detection as class 7 (partial fish visible). In order to fully utilize the features extracted, another search for parameters over C and σ is needed.

Pugh's search for optimal parameters involves finding $\gamma \in [2^{-5}, 2^{-3}...2^{13}, 2^{15}]$ that gives the highest accuracy among the pre-split dataset, due to the number of features used and the size of the dataset, training a single SVM could take from 30-minutes to 2-hours. Using the new grid search involving C and K-Folding, about 500 new SVM are needed to fit for finding the optimal value, this would take over months to complete. With the help of the task distribution system developed, it took about 4 hours to find the new optimal $\sigma = 10^{-3}$, and $C = 1$. The result of the SVM is shown in Fig 5.3.

Fig 5.3 shows the confusion matrix of the result after testing it on the additional validation dataset, using the original top-N algorithm developed by Pugh[1]. At first, the result looks promising, but after normalizing it is shown that a high percentage of the detections are marked as class 7. It is suspected that the classifier has learned class 7 as its "Base Case", where every detection it doesn't know about is classified as class 7.

Without this interference, the SVM could still obtain reasonable results. Some attempt at removing this problem can be found in Section 5.4.

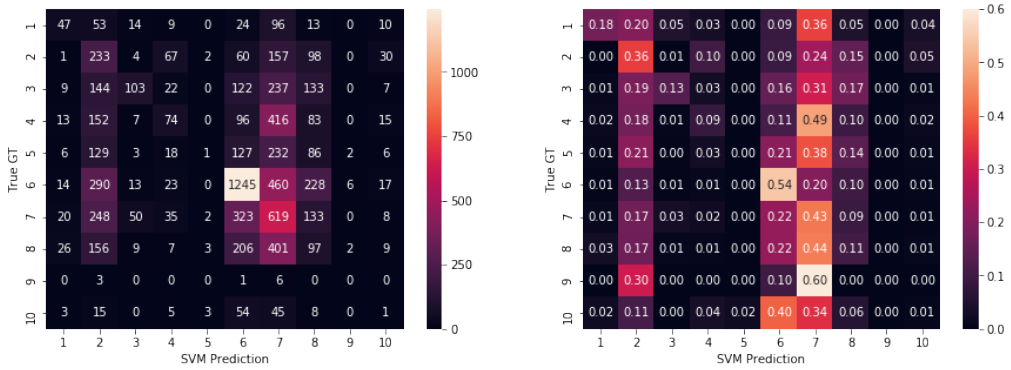

Figure 5.3: SVM's result and normalized result on unseen dataset

5.3 Convolutional Neural Networks (CNN)

As mentioned before in Section 4.6, the result of the CNN were heavily affected by the preprocessing errors. This caused one of the CNN learning too many unnecessary and random features of the image, which produces a poor accuracy on the unseen dataset, marking almost every image it sees as class 2 or 7 detections.

However even under this fault, the CNN_WC (CNN using an image with white mask) still manages to produce a reasonable result, and after testing different voting strategies, it's discovered that CNN_BC (CNN using an image with black mask) could also contribute in the final classification. This can be seen in Fig 5.4, the class 7 problem in SVM can also be found in this graph.

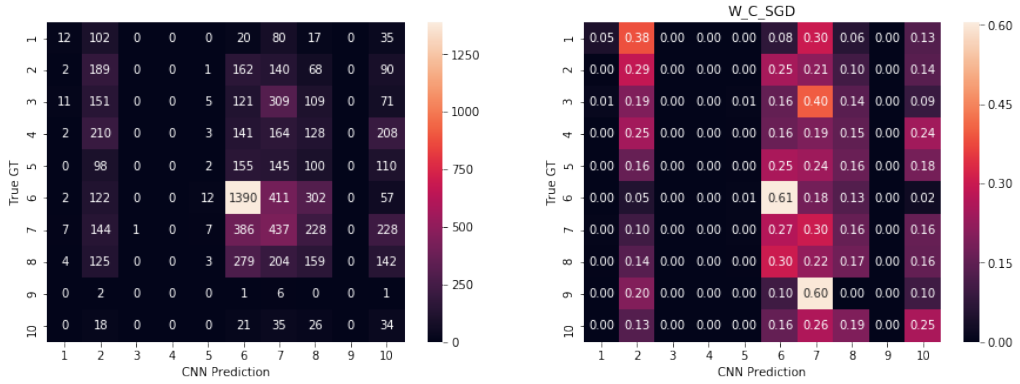

Figure 5.4: CNN_WC's result and normalized result on unseen dataset

Since the classification gives a poor result compared to the accuracy achieved in Pugh's thesis[1], it is first suspected that the translated Python code performed differently than the MATLAB's extraction. After discussion with Pugh, he mentioned the extraction procedure was different in his final pipeline.

His previous work uses MATLAB to experiment and evaluate on different classifiers. However, when it comes to the extraction process, Python's OpenCV library was used instead of the MATLAB ones. OpenCV stores an image in BGR space by default, and Lua loads the extracted images in RGB.

Because of lacking a second set of validation data, the final training was not validated, the mistake in colour space was not noticed until this project actually uses the trained model.

At the time of the project, we considered re-training of a new model. The original

model was trained with CUDA support, but the machine nodes used in this project do not have an NVIDIA video card. After evaluating Pugh's source code, each epoch of the training could take a full day on a single node, and re-training with the current framework would take 20 days on 200 machines, not to mention the disruption caused by the high memory usage. It was also considered to apply for a specialized cluster for the re-training, but due to the lack of time, re-train was not used in this project.

5.4 Voting Strategy

As the classifier does not work as expected as in Pugh's thesis, it is reconsidered to apply the previously failed attempts on increasing accuracy of the classifiers. One of the methods tested is to add a voting method on the predicted probabilities obtained by different classifiers, to test out different combination of the classifier results.

The voting strategy is described in the pseudo-code below. The input: `predict` is the predicted probability outputs of the classifiers, while the other parameters are the options used for searching for the best combinations of the classifiers.

Algorithm 1 Voting Decision

INPUT: predict, useSVM, useCNN, useCNNB, useCNNW, voteNeed, γ
Count $\leftarrow$ 0
KeptClass $\leftarrow$ [6,8]
If useSVM **and** sum(predict.getProb(SVM,KeptClass)) $>\gamma$ **Then:** Count+=1
If useCNN **and** sum(predict.getProb(CNN,KeptClass)) $>\gamma$ **Then:** Count+=1
If useCNNB **and** sum(predict.getProb(CNNB,KeptClass)) $>\gamma$ **Then:** Count+=1
If useCNNW **and** sum(predict.getProb(CNNW,KeptClass)) $>\gamma$ **Then:** Count+=1
If Count >voteNeed **Then:** return True **Else:** return False

After testing through different settings, a Receiver Operating Characteristic (ROC) curve of the voting performance is plotted. The ROC curve plotted in Fig 5.5 shows the True Positive Rate (good fish kept) against False Positive Rate (bad detections kept) at different thresholds (γ value in the pseudo-code).

The project can only afford to lose about 10% of the good fish. With this limit, the best classifier is to use SVM's predicted class 6 probability only. The ROC curve of this classifier is the black curve in Fig 5.5.

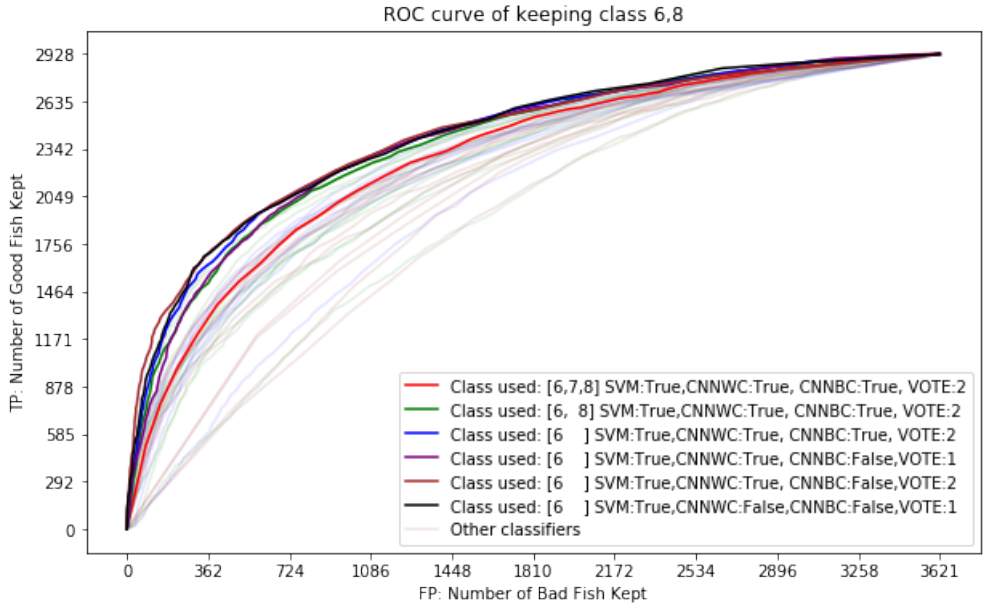

Figure 5.5: ROC curve of different settings for voting algorithm

The chosen gamma value for the criteria is 0.0108, meaning all detection with SVM's class 6 probability higher than 1.1% is classified as a fish. After the method is tested on various datasets from ground truth, and the following statistics are obtained.

Training Dataset	PR reject	FEIF reject	SVM reject	Detection Kept
True Fish	2858	1090	491	26211
Non Fish	1955	10577	7690	19511

Validation Dataset 1	PR reject	FEIF reject	SVM reject	Detection Kept
True Fish	61	91	88	2972
Non Fish	266	1174	553	2750

Validation Dataset 2	PR reject	FEIF reject	SVM reject	Detection Kept
True Fish	1	13	2	373
Non Fish	38	133	101	339

Note the high True Fish reject rate in training dataset, this is because planktons are marked as fish in Pugh's ground-truthing process. The final reduction rate are shown below.

Complete RDS	PR reject	FEIF reject	SVM reject	Detection Kept
Fish	38,512,254	73,686,561	120,124,496	607,142,479
Percentage	4.6%	8.8%	14.3%	72.3%

A further attempt at the reduction of False Negatives is discussed in Section 7.3.

Chapter 6

Parallel Distribution

Originally the deployment of the cleaning task was planned to be in the form of a pipeline. But with sufficient disk space and the need to translate the pipeline parts, the project changed the pipeline into these stages of processing:

- Preprocessing, Feature Extraction and Transformation for SVM
- Classify using SVM and CNN
- Final Classification

The first two stages are separated because they cost most of the computation, and the final stage is separated due to the need for experiments to improve accuracy.

6.1 Message Passing Interface for Python (MPI4PY)

In 2005, researchers added MPI support for Python[8] and extended the capabilities for the MPI-2 standard in 2008[10]. The current version of MPI4PY[11] was implemented in Cython, where the MPI calls were handled in C, ensuring high performance and compatibility.

Mpi-master-slave[5], a small python library with MPI4PY is used to distribute all the jobs among the available machines. For example, if there are 4 machines available, with the following command:

```
>> HOSTS=basso,battaglin,belloni,bergamaschi
>> mpiexec -n 4 -host $HOSTS \
.. python ~/f4k/runMulticoreFeatureExtract.py
```

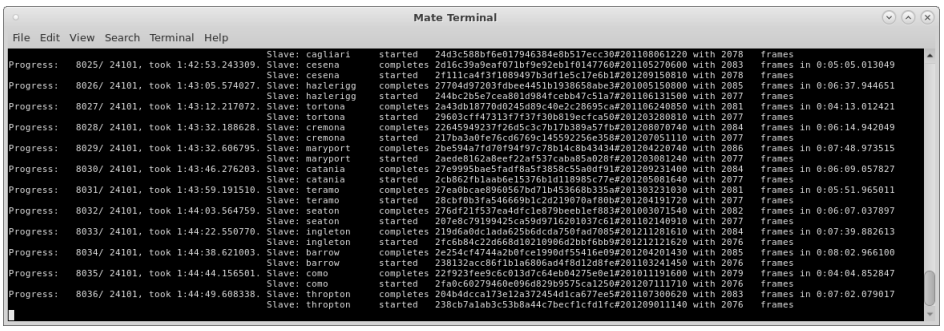

Figure 6.1: Running the MPI program on MATE terminal

The python program will be started on above 4 machines, with the first one (basso) being the master node. The master node keeps a list of video that needs to be processed as a work queue and distributes them to slaves nodes, which is every other node in `$HOSTS`. The slave nodes process the video and store the result on AFS. Upon finish, it notifies the master to receive more tasks. Pseudo-code of the framework can be found in Appendix A.

For all the "slave" nodes, Python's MultiProcessing and MATLAB's ParPool are used to fully utilize every core's computational power.

6.2 CPU Hogging and Memory Thrashing Prevention

The project uses the public lab DICE machines provided by The University of Edinburgh. It is important to make sure the cleaning procedure doesn't affect other users.

To find idle machines, a Python script using Multi-processing pools is used. It spawns a "`ssh MachineName w`" bash process for each machine. With the `ssh`'s X11 forwarding enabled, the results printed out at the target machine is transferred back to the Python script. Then the script checks the length of output obtained from this standard output stream. If that machine has no users logged in, the returned string will have an exact length of 2 lines (containing the headers of query). This script allows the project to obtain a list of available machines within 15 seconds.

For the CPU usage, SL7 provides a `NICE` command, allowing the program to have a higher `NI` value which gives lower priority on CPU scheduling. Even if another student logged into the running machine, this allows them to work without disruption.

After testing it on the machines, it was reported anonymously by other students that

some machines are starting to become unresponsive, and caused work lost after running code on it. It was later discovered to be a memory thrashing problem. Loading a video with 10,000 frames and all the library functions will take about 3 GB space in physical memory. After dereferencing every variable and forcing the garbage collecting mechanism, it still leaves 10% of the memory in use. This eventually piles up and thrashes the memory.

A solution is to spawn a sub-process for each batch of the video cleaning and kill it after it's finished. Doing so would increase the time to reload the Python libraries, taking about 5 to 10 seconds per video. The increase in time cost is almost negligible compared to feature extraction cost, so no further optimizations are made.

6.3 Error Recovery and Progress Record

MPI allows a more portable and scalable way of distributing of tasks on multiple computational nodes. However, one of the main disadvantages is the error handling. If one of the MPI-process crashes, all of the processes running will be forced to exit.

The project is running on a massive scale, so error handling plays a more and more important role in the processing. When an error is caught when processing a video, it immediately sends the master node a `DONE` signal, but with a failure message. The master node will remove the failed slave node and put its job back to the start of the work queue, allowing it to be re-scheduled again.

In the case of master crashes and other situations that MPI process is killed, all of the nodes will stop and some progress recording mechanism is needed for resuming progress. After all the output is stored on the disk, another empty file with `.complete` suffix is created. So the "slave" processes could know if a video is finished processing, or killed before it finishes storage. With this file existence check, repeat work after restart is greatly reduced, hence improving the progress.

Even with these issues addressed, the following event could still cause faults:

- **Rebooted machines**: Some student tend to restart the lab machines before using them, this causes code to terminated by `SIGTERM` or `SIGKILL`.
- **Unplugged machines**: Causes the master node to wait indefinitely for a reply.

With this limitation, the project could only use the framework during night times.

6.4 Time Cost

In Yu's thesis[3], it is estimated the cleaning could take 25,000 hours to complete on a 40-core machine. This evaluation is based on the runtime of cleaning a video with 961 detections. Finish processing these videos took approximately 200 seconds, Yu obtain the estimated runtime by multiplying this amount by the total number of the videos (400,000). Within this 200 seconds, half of the time is used for the SVM feature extraction/transformation, and another half was used for the CNN preprocessing. The final classification took negligible time.

However, this estimate wasn't thorough, since there are 839,000,000 detections and the average length of the video is about 2,000 detections. The actual runtime would double, having about 1,600,000 hours of runtime on a single core.

During the project, about 200 student lab DICE machine was used. The number of available DICE machines varies from 50-250 due to heavy experiment periods of other students. This issue potentially cuts the previous runtime to 1/200, but due to the disruption to/from other students, the project runs the cleaning mostly during night time (12 A.M. to 8 A.M.).

With this reduction, the project finishes the different stages in the following time:

- **SVM preprocessing**: Approximately three weeks, estimated total runtime of about 200 hours.
- **CNN preprocessing and classification**: Approximately one week, estimated total runtime of 70 hours.
- **Other operations**: Like training SVM and generating decision vectors, these all took negligible time (less than 1 day).

Note that the SVM took longer due to the high memory requirement for the cleaning mentioned in Section 4.1.

From the result cleaning time, the translation stage of the project manages to cut down the runtime to 1/8 of the original speed. This reduction is mostly because of removing the SQL server, which has brought the project most reduction in runtime. It might also be the removal of unnecessary I/O (saving images to disk and loading them again for multiple times) during CNN stage. The project benefited mostly from the idle student lab machines. This is equivalent to having 20 40-core machines used in Yu's estimation.

Chapter 7

Conclusion

7.1 Project Outcome

Initially, the project goal was to further reduce the size of the RDS dataset from 1.6 TB to an acceptable size. Given the accuracy obtained from the actual run, doing so would not remove enough False Positives. Also due to the need for maintaining consistency in the `.sql` file and video files, this final removal process is not used.

Instead, this project produces 2 types of `.npy` file (saved Numpy Arrays). One of them is the predicted probability of each class obtained by the 4 major classifiers used, taking about 260 GB disk space. Another one is the final binary array of decisions, marking each frame as a good detection or not. This is more compact than the other, using about 750 MB disk space.

The by-product of the process may be useful for a future cleaning attempt on the dataset, this includes:

- A 1 GB folder, contains Numpy dump of final binary decision vector.
- A 323 GB folder, contains extracted `.sql` records.
- A 510 GB folder, contains normalized and PCA transformed first 100 features.
- About 20,000 marked frames of ground truth dataset, covering the videos both Pugh and Yu missed during the ground truth process.
- A Python mpi application to distribute the classification work on DICE machines, and a Python script to obtain unused DICE machines.

- Translated utility functions in Python, some are from Pugh's MATLAB code and some of them are implemented in this project. This contains several Jupyter Notebooks and two Python libraries. These allow easier visualisation of the dataset, without the need to go through multiple extractions and SQL connection stage in MATLAB.

7.2 Samples of Images Rejected at each stage

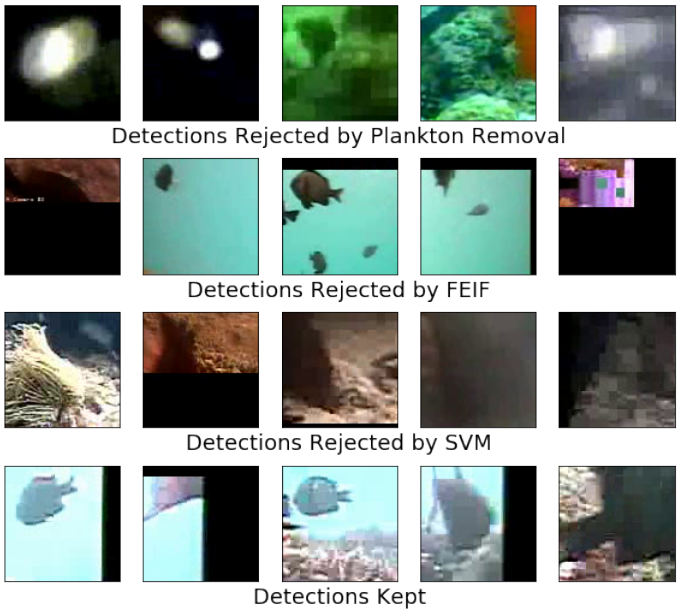

Figure 7.1: Sample of Removed Images at each stage of cleaning

7.3 False Negatives in cleaning algorithm

The cleaning algorithm produces approximately 5-10% of False Negatives (rejected good fish). To reduce the False Negatives, they were scrutinized to see the causing reasons for the fault.

Most of the False Negatives in the cleaning are caused by FEIF used, approximately 2-3% of the RDS were rejected. Fig 7.2 shows the sample of False Negatives in the validation dataset. Note that the time-stamp in absent from the area highlighted with red lines. By looking into the dataset, it is discovered that all the `640x480` resolution videos recorded at NPP-3 sites after 2011-Aug-05 are missing the time stamp.

Figure 7.2: Sample False Negatives (good fish rejected) by FEIF

This absence of time stamp was not expected in early stages of the cleaning process, hence it was not included in the settings for FEIF. Fig 7.3 shows the contours of the False Negative detections. The darker-red area indicates the time-stamp area for `320x240` videos used in the species recognition stage of the F4K project, and the light-red area is newly added to this project.

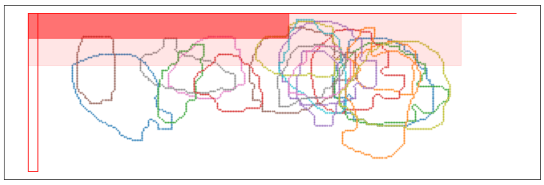

Figure 7.3: Contours in the FEIF time-stamp region

The similar problem also happened to the plankton removal criteria in Section 4.1, where a mistake in the time checking code causes some of the videos recorded at 6-7 A.M. to be removed. However, this only affects about 0.2% of the detections.

While the early removal stages fault causes a lot of class 6 False Negatives, most of the class 8 False Negatives are caused by the SVM used, which is partially considered to be the fault of the classification schema discussed in Section 5.1. Where class 8 stands for good fish with bad contour, this covers detections from slightly erratic contours to completely useless contours. Fig 7.4 shows a sample of such fault. These contour faults are caused by background extraction from previous species recognition of F4K project. Fish with colours similar to the background or stripes usually have broken or segmented contours.

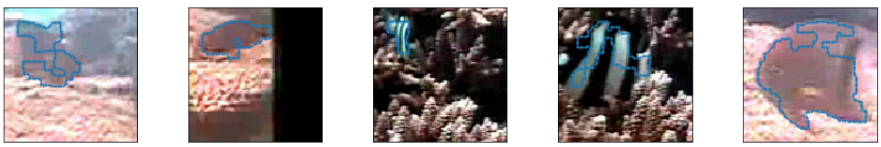

Figure 7.4: Sample False Negatives (good fish rejected) by SVM

7.4 Future Work

This project successfully applied the cleaning algorithms designed by Pugh[1]. Unfortunately, it did not achieve the same level of accuracy as Pugh's thesis described. In order to increase the accuracy of the classifiers, not only we needed to fix the aforementioned faults in Section 7.3, other improvements will also be needed.

7.4.1 More Annotation

As stated in Section 3.3, the current Ground Truth Dataset is heavily biased, which completely misses out the detections at some sites of filming. Even after adding 20,000 detections across every site, the dataset still has incomplete coverage on the RDS.

Pugh also mentioned about this problem in his thesis, where more human annotator might be needed for the project. With the limitation of the current classification schema, the ground truth process is irritating and tedious due to the ambiguity mentioned in Section 5.1.

An alternative may be using an online survey for ground truth. But the survey is not fully implemented at this stage of the project. Fig 7.5 shows the web interface for the survey. A user is shown a detection in a track, and they could use the provided options to send their decision to the database server used.

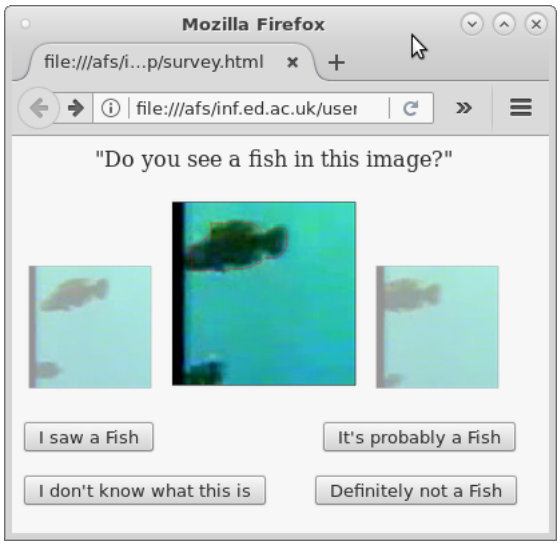

Figure 7.5: A prototype online ground truth interface

7.4.2 Training New CNNs

Due to the image extraction fault in the training stage of the CNN, an over-fitted version of CNN is used in this project. In order to achieve expected accuracy in Pugh's thesis[1], re-training of the network will be needed.

The accuracy obtained by the training dataset on CNN_WC has shown promising accuracy even with a completely wrong colour space. Combining with Pugh's result obtained in the experiment stage, and the difference in performance on Training/Validation dataset obtained in this project, it's almost certain that a well-trained CNN would obtain at least 70% accuracy on the cleaning task.

7.5 Final Words

The project finished the algorithm originally designed by Pugh. However, the expected level of accuracy was not achieved. Nonetheless, cleaning of the full dataset was accomplished, with an estimated removal of 42% False Positives while removing only an estimated 7.5% True Positives. This reduced the dataset from 839,465,790 fish to 607,142,479 fish.

In hindsight, the development process was not careful enough, which is mainly reflected in the FEIF behaviour of the project. While setting parameters, only reduction rate is considered. Details like False Negative Rate was ignored and have led to most of the lost true detections.

Also, the project has trusted the previous result blindly at early stages of the cleaning, which has led to the late discovery of the CNN fault. The validation dataset is marked only after the translation of the algorithms.

With the big data nature of this task, many issues only arise when the project started running the cleaning on the distributed scale. These issues could be avoided if more analysis and debugging of the previous code had been done at the start of the project.

For example, MATLAB's ParPool uses a local lock file to keep track of the parallel tasks. Combined with the shared file system used, lots of MATLAB segmentation fault crashes have been caused by this. Due to the ambiguity of the error messages, this problem was only addressed months after first discovery.

Another example would be the `.sql` preprocessing mentioned in Section 3.2. The project misses one of the special characters in the blob syntax used, which has caused 30% of the contour data to be missing after extraction. This fault also required the project to re-run the SVM extraction stage and two week's work was lost because of it.

Future work on the project is needed to fix these remaining faults:

- Adjust the plankton removal and FEIF criteria mentioned in Section 7.3.
- Create a larger set of Ground Truth dataset.
- Re-train Pugh's Convolutional Neural Network.

Exploration of other machine learning techniques is also needed in order to achieve higher accuracy on the classification.

Bibliography

[1] Matthew Pugh. Removing false detections from a large fish image data-set. Msc dissertation, The University of Edinburgh, 2015.

[2] Robert B Fisher, Yun-Heh Chen-Burger, Daniela Giordano, Lynda Hardman, Fang-Pang Lin. *Fish4Knowledge: collecting and analyzing massive coral reef fish video data*. Springer, 2016.

[3] Qiqi Yu. Adding temporal constraints to a large data cleaning problem. Msc dissertation, The University of Edinburgh, 2016.

[4] Phoenix X. Huang. *Balance-guaranteed optimized tree with reject option for live fish recognition*. PhD thesis, The University of Edinburgh, 2014.

[5] luca-s: mpi-master-slave github repository. `https://github.com/luca-s/mpi-master-slave`. Accessed: 2018-01-18.

[6] Introduction to hadoop image processing interface (HIPI). `http://hipi.cs.virginia.edu/`. Accessed: 2018-01-15.

[7] 4quant homepage. `http://4quant.com/`. Accessed: 2018-01-10.

[8] Lisandro Dalcn, Rodrigo Paz, and Mario Storti. Mpi for python. *Journal of Parallel and Distributed Computing*, 65(9):1108 – 1115, 2005.

[9] Chain code. `http://www.crisluengo.net/index.php/archives/324`. Accessed: 2018-03-30.

[10] Lisandro Dalcn, Rodrigo Paz, Mario Storti, and Jorge DEla. Mpi for python: Performance improvements and mpi-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655 – 662, 2008.

[11] Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124 – 1139, 2011. New Computational Methods and Software Tools.

Appendix A

Master/Slave Framework Pseudo-Code

With this framework design, if we want to process any stage of the pipeline classifier, we only need to overwrite the dummy **DoWork**(VideoID) function of slave code.

If the framework is used in work other than processing the videos, (e.g. finding SVM parameters), the **VideoIDList** can be changed to adapt to the new tasks.

Algorithm 2 Master Workflow

```
WorkQueue ← []
for VideoID ∈ VideoIDList do
    WorkQueue.addTask(VideoID)
end for
while not WorkQueue.done do
    for Slave ∈ mpi.getList(SlaveList,"READY") do
        if WorkQueue.done then BREAK
        mpi.send(Slave,"START",WorkQueue.getTask)
    end for
    for (Data,VideoID) ∈ mpi.getList(SlaveList,"FINISH") do
        WorkQueue.markComplete(VideoID)
        if Data = "Fail" then
            WorkQueue.addTask(VideoID)
        end if
    end for
end while
mpi.broadcast("EXIT")
```

Algorithm 3 Slave Workflow

```
while not mpi.receive("EXIT") do
  mpi.send(Master,"READY")
  if mpi.receive(Master,"START",VideoID) then
    Success ← DoWork(VideoID)
    if Success then
      mpi.send(Master,"FINISH","Success",VideoID)
    else
      mpi.send(Master,"FINISH","Fail",VideoID)
    end if
  end if
end while
mpi.broadcast("EXIT")
```

Appendix B

Sample of Lengthy Videos

In Section 4.1, a classification method based on file-name is proposed. If a video has over 30,000 detections and is filmed at a specific time and location, every frame of the video will be marked as non-fish. As each video of RDS originates from a 10-minute video, having such amount of detection essentially means there are over 50 fish captured per second of original FDS.

For the most of the cases, such videos are full of compression errors, plankton, or having background extraction faults caused by moving vegetation. These cases are shown in Fig B.1, Fig B.2, and Fig B.3.

By manually looking through the samples from the longest videos, a limit of 30,000 is set due to the need of keeping good detections such as those in Fig B.4. This ensures the good fish lost will be less than 1% of the bad detections this method removes.

Figure B.1: Sample frames from a video filmed at night.

Figure B.2: Sample frames from a corrupted video.

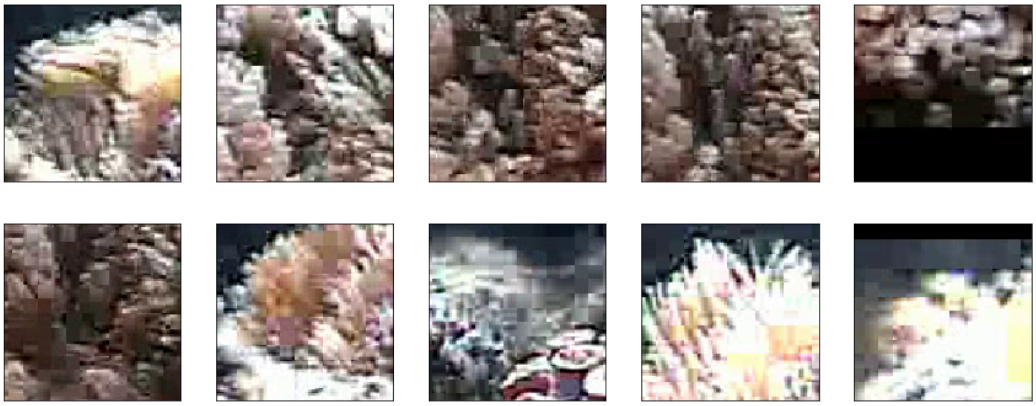

Figure B.3: Sample frames from a video with dynamic background.

Figure B.4: Sample frames from a video with abnormal amount of fish.