Parallel Massive Dataset Cleaning

Jianmeng Yu

4th Year Project Report

Computer Science

School of Informatics

University of Edinburgh

2018

Abstract

This project applies 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 approach to reduce unnecessary work during pre-processing and improve the cleaning algorithm. In this process, efficiency evaluation for different implementations of the machine learning techniques is used to reduce computational time cost. 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, hence 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 environmental effects by analysing raw videos and extracting information from it, so researchers could use it 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). After 5 years of recording, the project collected about 524,000 10-minute video clips, with a total size of 91 TB, and approximately 1.4 billion fish detection 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 their 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 are 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 cleaning was not used on the dataset. In 2016, Yu[3] attempted to add voting constraints on the cleaning algorithm, and evaluated the time cost of the algorithm. The cleaning wasn't applied because the constraint could not improve much to the efficiency of cleaning.

It is evaluated cleaning a 1,000 detection video on a 40-core machine would take 200 seconds. This gives a 8 second per frame per core (8 `s/fc`), with the available 200 4-core machines in student labs, 1 `s/fc` would result in 12 days of computational time.

1.3 Contribution

After translation and modification on the algorithm used, the project managed to improve the efficiency to about 0.5-0.8 `s/fc`. Distributes and completes the aforementioned cleaning algorithm.

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

Data extraction pipeline written in Python was created to partition, extract, and parse the raw SQL dump file to comma separated values stored in plain text files.

The pipeline classifier is also re-written in Python, 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-engineered into Python functions. Some metrics algorithms used is translated from MATLAB for loops to Numpy/Scipy operations to increase efficiency.

F4K project's feature extraction MATLAB code developed by Huang[4] is untranslated and is called within Python using PyMatlab library. Some minor changes like replacing edge extraction algorithm used, and error handling added to one of the unstable

`getCstd()` function.

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

The classification step was originally going to use Pugh's trained SVM and CNN. Due to the problem above, the SVM parameters used are re-calibrated for higher accuracy on unseen dataset. The Python's `sklearn.svm.SVC` were 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 were found in the CNN design, this causes 2 out of 3 CNN trained by Pugh almost useless. Details about the fault and the re-train attempt were included in (TODO). Due to this mistake, an attempt of re-use Yu's previously developed voting classifier to increase the accuracy were made.

(TODO) Possible reconstruction of the SQL and video is needed? Some of the FEIF rejected images still seems to be useful.

1.4 Document Structure

Chapter 2 discusses the designs used and previous work details of 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

Backgrounds

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 analysing, the 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], 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 uses the student lab machines provided by The University of Edinburgh. These machines have 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, a 1 TB and 256 GB disk space on AFS were used for this project.

The DICE machines used in the project does not have a shared memory, this means the project will also need tools for distribution of the task before 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], a 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 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:

- **I Not A Fish** - These detection 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 video 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 detects fish correctly, but misleading information may be extracted, it's unsure these frames should be kept or not.

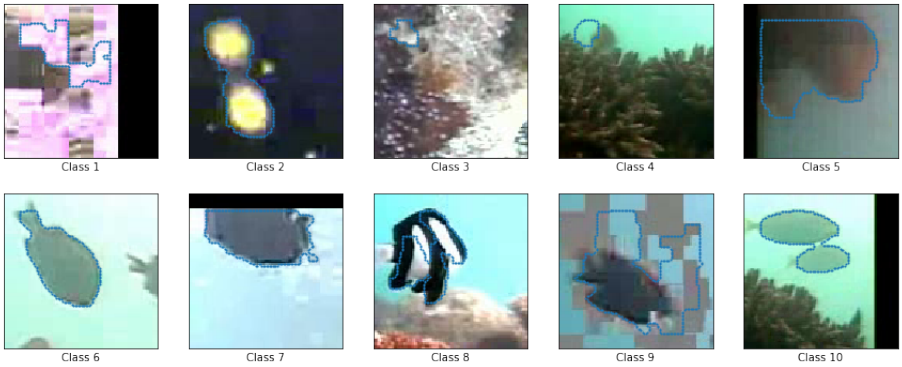

Figure 2.1: Example Detections From Each Class

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 accuracy of the classifiers due to the similarity between the classes, this limitation is 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.

The first limitation is the SQL database, which stores the track and contour information of the image. However the SQL is too large and slow for the project, To make the extraction more sensible, a Python script was used to partition the SQL. This potentially cuts the runtime to about 10% of the original design and removed the need for a SQL database server. More details of this modification is in Section 3.2.

Before sending the data into the classifiers, preprocessing is needed to give a more sensible result. In Pugh's thesis[1], a Frame Edge Indicator Function (FEIF) is used to directly reduce the number of frames 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.

After extracting and reducing features, they are fed into 3 Convolutional Neural Net-

works (CNN) and 10 Support Vector Machines (SVM), then applying a top N algorithm to produce the final result.

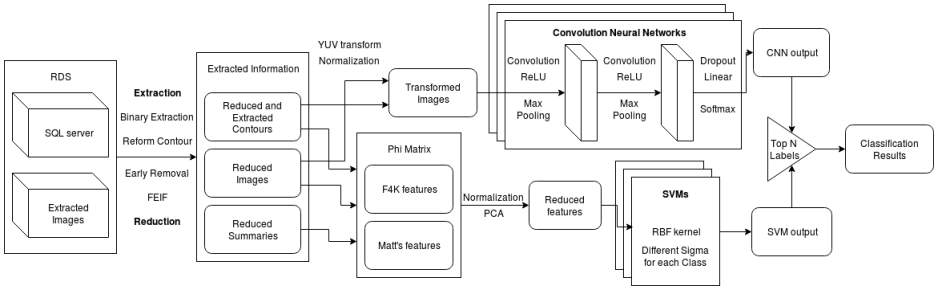

Figure 2.2: Pipeline Classifier designed by Pugh, modified

2.4 Yu's Voting Constraints

In 2016, Yu[3] tried to add a voting constraint on the Pipeline Classifier. Yu evaluated her 3 voting method on the result 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 decrease the True Positive Rate (TPR) by 5% and is not considered useful for the project.

However 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 the coverage, and mistakes in preprocessing steps. Details of this problem is discussed in Section 3.3 and Section 4.5.

Since Yu's evaluation were based on the results of Pugh's experimental classifiers, it is shown that Pugh's classifier would obtain promising accuracy if the mistake in Section 5.3 is fixed. Due to the high cost of training, this project will use a modified version of Qiqi's voting strategy to make use of the results from the over-fitted classifiers.

Chapter 3

Data Source

After the species recognition of the F4K project, three types of output files are stored:

- 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.

3.1 Extracted Images

During the species recognition, the bounding box and contour of each detection are computed. 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.

For each video a `video_id` is generated, it consists of a 32 byte hash of video, a `#`, and the filming date of `YYYYMMDDhhmm` format. For every detection with `w` and `h` both smaller than 90, a process illustrated in Figure 3.1 is applied, where a 100x100 area is selected with top left corner coordinate `w-10` and `h-10` and cropped from the image. If the selected area is out of the range of the frame, those areas will be filled with black pixels. 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`.

There are a total of 396,901 of such videos, consist of 839,465,846 frames, sums up to a total size of 1.14 TB.

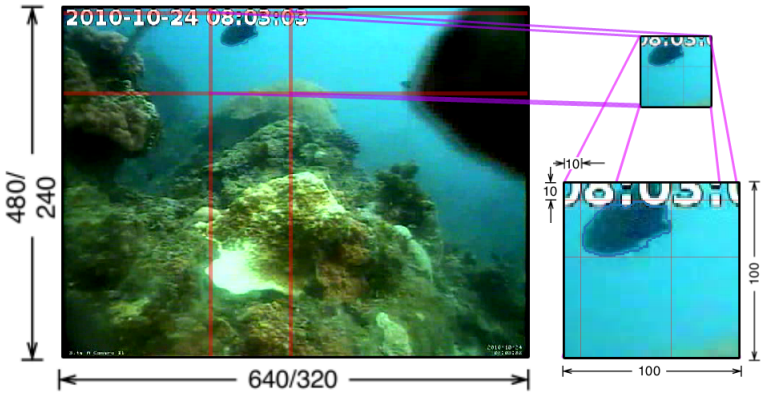

Figure 3.1: Process of Extracting Image

3.2 SQL dump file

The `.sql` dump file comes from the SQL workflow of the F4K project, which means not the only details of fish detection are stored. Other components irrelevant to this project is also stored.

Below is the schema of the ”Fish Detection” table. This table takes about 326 GB, and is the only table that is actually needed for the cleaning.

```
CREATE TABLE IF NOT EXISTS   f4k_db . fish_detection  (
 detection_id  INT(11) NOT NULL AUTO_INCREMENT,
 fish_id  INT(11) NOT NULL,
 video_id  CHAR(45) CHARACTER SET  utf8  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 NULL DEFAULT NULL,
tracking_certainty  FLOAT NULL DEFAULT NULL,
 component_id  SMALLINT(6) NOT NULL,
processed_videos_id  INT(11) NOT NULL
```

There are other tables such as ”Camera_Info” that shows the camera used for each video, with relatively small disk usage.

This `.sql` dump file is stored as plain text files that could be parsed into SQL code that loads the stored data. Under limitations of disk space and access speed, loading a large SQL database dump file into a server and performing 400,000 queries is very unnecessary and time-consuming, hence making it the slowest part of the cleaning. Also, the provided AFS quota could not hold all the information. An alternative is to use python script with standard stream pipeline to parse and partition the SQL dump file into directly usable files.

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 files with its video_id, the information extraction will be much faster without the need to seek through all the records of other videos.

Because each detection is only need once during the extraction, and the dump file is too large for the RAM, standard stream pipeline a more reasonable choice for the parsing.

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

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

are parsed into directly usable list of values, separated by `newline` character.

This process makes the time cost for loading the dataset reduced to an almost negligible amount. In Qiqi's estimate, the loading and preprocessing would take 800,000 hours (25,000 hours on a 32 core machine) in the original design (double the time if considering her calculation error). After the pre-extraction, it only took about 0.3 seconds on average for a frame to be load and processed on a single computational thread. Which is about 70,000 hours, essentially cutting down the time used to 10% of the original design.

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 contains the `x,y,w,h` in original videos, and a chain code to store the fish boundary data in a more compact format.

Also, since the binary file is stored as text file, a different encoding is used so it won't cause parsing fault during loading, for example, 8 consecutive 0 bits, the null character, are stored as two bytes in ascii format of `\0`. A cleaning function is used 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 is found that the binary data is in the following format:

- First 42 (11,11,10,10) bits - `x,y,w,h` of the bounding box.

- Next 11 bits - X-coordinate of the first contour point.
- Following 3 bits - Padding of zeros added to make the length a multiple of 8.
- All other bits - Chain Code of the contour, 3 bits each. Pointing towards next contour point from previous one.

This process allows quick extraction of the contour, loading the image, summaries, and the contour information of a 1000 frame video into memory from AFS only took 1 second now and removes the need to maintain a running SQL server.

3.3 Ground Truth Dataset

In order to train and evaluate the classifiers, Pugh and Yu manually marked a set of detections with the schema above. They chose a subset of the RDS, which is some of the videos having id start with ”13b”. This subset of the RDS 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. Moreover, some set of the detection are marked wrong because of the ambiguity of the classification schema. For example, a lot of the planktons are marked as fishes, and 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.

After extracting some statistics on this dataset, it is discovered that this dataset is heavily biased, it fails to include normal videos filmed at 2 sites. Due to need for re-calibration and future validation. Two subset with 10,000 detections each were marked.

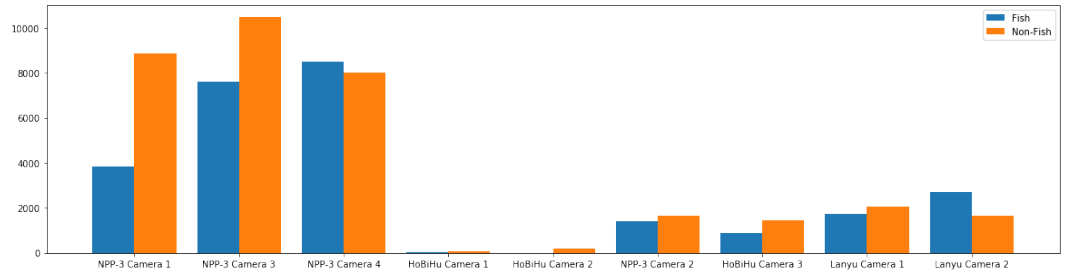

Figure 3.2: Number of Fish/Non-Fish at different sites in original Ground Truth Dataset

Chapter 4

Preprocessing

While the data source problem is solved, the most costly part of the pipeline will be the feature extraction for the SVM part. It's not sensible to send all of the RGB values of a image as direct input for the SVM, hence feature extraction and dimensionality reduction is needed.

To achieve the goal of removing False Positives without losing too many True Positives, some reduction methods are used on the dataset before extraction of the features. Early Video Removal was introduced and an improved version of Frame Edge Indicator Function developed by Pugh was used.

4.1 Translation from MATLAB to Python

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

The F4K feature extraction code have 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 consists of customized code that could not be directly translated to Python.

For example, the Gabor Filter used in the project are 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

re-engineer the whole F4K feature library.

A full translation of the F4K feature could take a few weeks. Since the cleaning algorithm only need to execute once, translation may not be the optimal path to take. After recording performance on various parts of the algorithm, we determined that one of the slowest operation is 2D convolution.

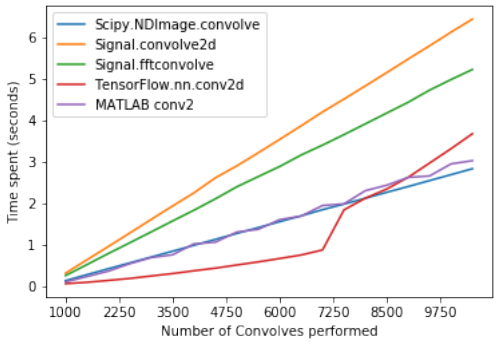

Figure 4.1: Test runtime of different 2D convolve algorithms.

Figure 4.1 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 library, it has a significantly higher memory usage due to the tensor data type. It also have a high initialization cost due to the transformation needed.
- Scipy.NDImage's convolve, giving almost same 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.

4.2 Early Video Removal

During the feature extraction tests, it is discovered that loading a 40,000 frame video and extract features from it would take about 8 GB of memory space. If such video

is processed on a node with RAM less than 8 GB, it will cause serious thrashing, rendering the node unresponsive. For machines having a RAM higher than 8 GB, it will still causes some disruption during loading.

While risking the chance of thrashing, these videos took a longer time to process, and most importantly, they are usually filled with False Positive detections. As discussed in Chapter 3.1, if a camera recorded 30,000 detection in 10 minutes, means that in every frame of the original video, an average of 10 detection is extracted. By looking at these ”outlier” videos, some patterns were found:

- Both cameras at Lanyu site are night-vision cameras. When they film during the night, a lot of light reflecting planktons and small animals close to the camera are recognised as fishes. Videos filmed during night have an average of 6974 detections. A 5% of the total detection comes from such videos, with the 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 change 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 detections. They usually contains lots of good detections.
- Dynamic background - Videos filled with moving vegetation, or refraction of sunlight. They usually contains lots of good detections.

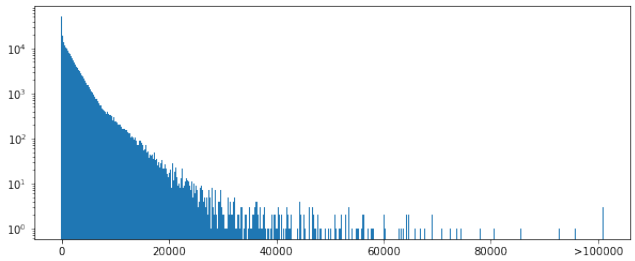

Figure 4.2: Log Scale Histogram of Detection Length

Using the above patterns and observed distribution in Fig 4.2, if we remove all the videos recorded in the night, videos with 40,000 or more frames, and video recorded with above characteristics and 20,000 or more frames. About 8% of the detections can be rejected without need to extract them, saving approximately 200 days of computational time.

4.3 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, a boundary of video is defined, and if the number of the contour points outside of the boundary exceeds 25, the detection is then rejected.

However this function could not achieve the intention on some cases, for example, some videos have a darker frame edge, so even if a fish is cut by boundary, the contour points will be inside the boundary. 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.

Following modification is added to solve the problem, by shrinking the boundary by 2 pixels, 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. About 15% of the dataset is rejected by this upgraded FEIF algorithm, saving 400 days of computational time.

4.4 Feature Extraction

During the species recognition stage of the F4K project, 2626 features were used for computing detection certainty. On that basis, Pugh added 29 new features focused on the edges of the contour. Then 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-occurance 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.
- Rest of the features took almost negligible amount of time to finish.

Unfortunately after checking the features with PCA, these feature all took significant part in the first 50 PCA components, removing any one of the 3 time costly feature will have a high impact on the result.

4.4.1 Pugh's Features

Pugh's part of the generated feature consist of 4 parts: Animation Score, Boundary Curvature, Erraticity, and Gabor Filter on edge. This part of the pipeline is translated into Python as the original feature generation script is incomplete. Some inefficient `for` loops were translated using Numpy library to speed them up.

The Animation Score (AS) is calculated for one whole track, where 5 frames are pick 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), it 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 image generated using contour, instead of the original image used in the F4K feature extraction.

4.4.2 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 eigenvalue greater than 1 is selected. However the only advantage with this criterion is it being easy to calculate, essentially throwing away 30% of informations.

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

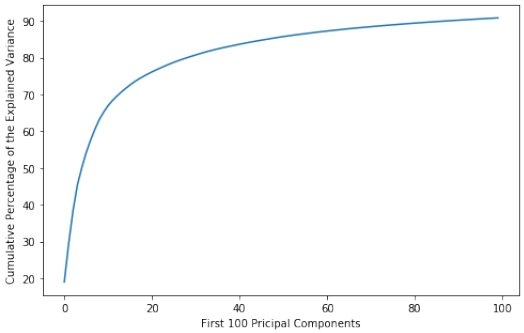

Figure 4.3: Variance Explained

4.5 Image Processing For CNN

In Pugh's pipeline classifier, 3 different CNNs were used on different type of preprocessed images. In `CNN_N`, 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, it is then moved to center of the image.

Before the image is forward into the CNN models, transformation and normalization is performed. Firstly the image is transformed into YUV color space. Then the image is normalized globally with mean and standard deviation of the YUV images. Finally, local spatial normalization was applied on each channel.

However after tests on the CNN trained, the result obtained weren't even close to the ones Pugh obtained. It is discovered that during the preprocessing stage of Pugh, two major mistakes were made. Pugh uses OpenCV's `imwrite` for the extraction of the image, the images are stored in compressed `".jpeg"` format, this leads to drastic change in the result matrix, due to the local normalization step. Another fatal mistake is the image store with OpenCV are in BGR space, where Lua recognize it as RGB image.

This causes the CNN_N and CNN_BC almost useless, classifying almost every unseen data into 1 single class. Some re-train attempt were included in Section 5.3.

Figure 4.4: Images on first stages of pre-process

Chapter 5

Classification

5.1 Ambiguity in Original Classification Schema

For the Pipeline Classifier, 10 classes of different detections were proposed in Section 2.2, but there are many limitations on this schema. For example, some cases could be appearing simultaneously, a detection could have an erratic boundary, and also touching the boundary of a frame. Pugh's thesis didn't state clearly different priority 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 keep class 6 detections.

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

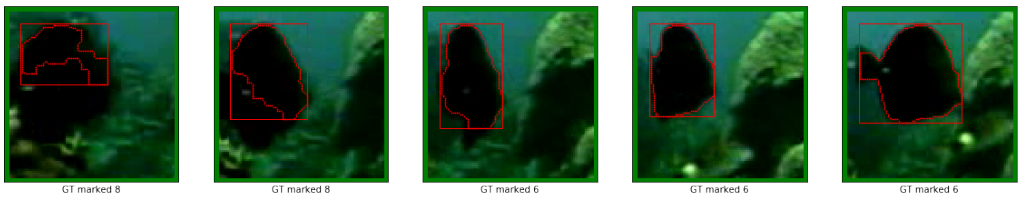

Figure 5.2: A track of detection, changing from class 5 (Unknown) to 6 (Fish).

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 detection might suddenly change to class 7 (partial fish visible) just because they are close to the frame edge.

For the case 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 process of new datasets, it is found that distinguish between these two class are 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 were evaluated again, and unfortunately, both need rework for it to work properly again.

5.2 Support Vector Machines (SVM)

In the original design, Pugh uses ten different SVM classifier with RBF kernel (Radial Basis Function kernel) for marking probability of each class. Each of the SVM 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 of parameter 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 amount of features used and the size of the dataset, training a single SVM could take from 30-minute to 2-hour. Using the new grid search involving C and K-Folding, about 500 new SVM are needed to fit for finding the optimal value, it 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.

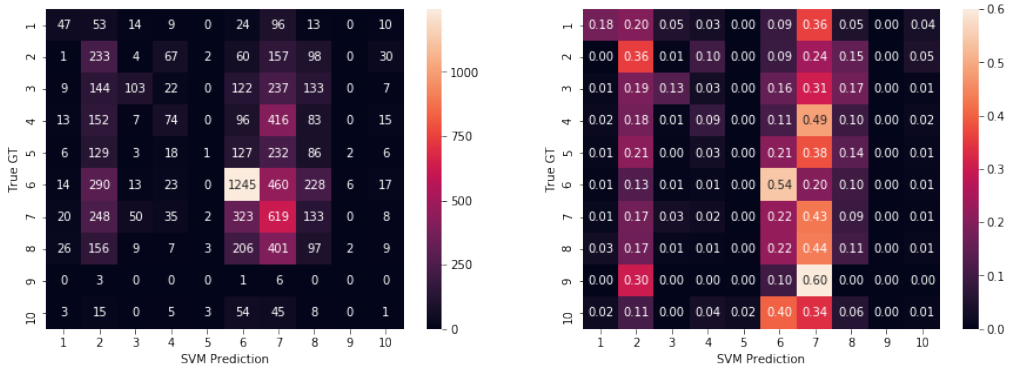

Figure 5.3: SVM's result and normalized result on unseen dataset

5.3 Convolutional Neural Networks (CNN)

As mentioned before in Section 4.5, the result of the CNN were heavily affected by the preprocessing error. This causes the 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 into class 2 or 7.

However even under this fault, the CNN_WC (CNN using image with white mask) still manages to produce reasonable result, and after testing different voting strategies, it's discovered that CNN_BC (CNN using image with black mask) could also contribute in final classification.

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 perform differently than the MATLAB's extraction. After discussion with Pugh he mentioned the extraction procedure were 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 were used instead of the MATLAB ones. OpenCV stores a 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 were not validated, the mistake in color space is not noticed until this project actually uses the trained model.

At the time of the project it is considered the re-train of a new model, the original model were trained with CUDA support, were the machine nodes used in this project does not have a 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-train with the current frame work would take 20 days on 200 machines, not to mention the disruption caused by the high memory usage. It is also considered to apply for a specialized cluster for the re-train, but due to the lack of time, re-train were not used in this project.

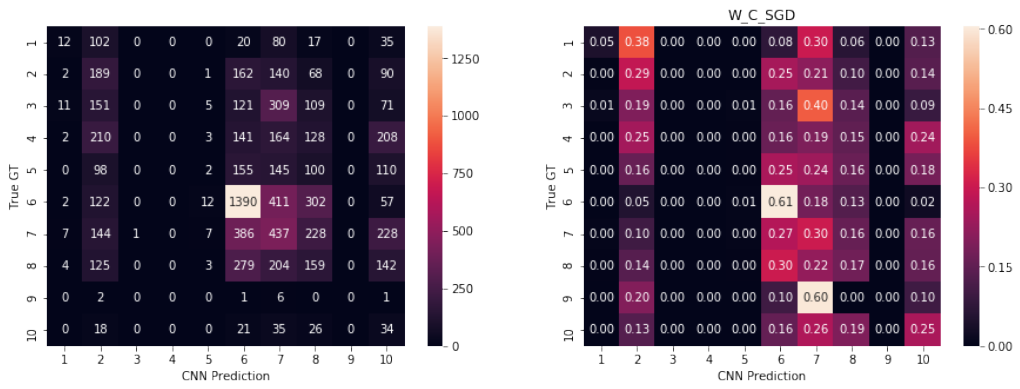

Figure 5.4: CNN_WC's result and normalized result on unseen dataset

5.4 Voting Strategy

As the classifier do 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 method tested is to add a voting method on the predicted probabilities obtained by different classifiers, to test out different combination of the classifier results.

Chapter 6

Parallel Distribution

Originally the deploy of the cleaning task will be in the form of a pipeline. But with sufficient disk space and the need to translate the pipeline parts, the project change the pipeline these stages of processing:

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

Where first two stages are separated due to them cost the most of the time, and the final stage is separated due to the need of experiments to improve accuracy.

6.1 Message Passing Interface for Python (MPI4PY)

In 2005, researchers have tried to add MPI support for Python[8], and extends the capabilities for MPI-2 standard in 2008[9]. The current version of MPI4PY[10] were 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 job 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
```

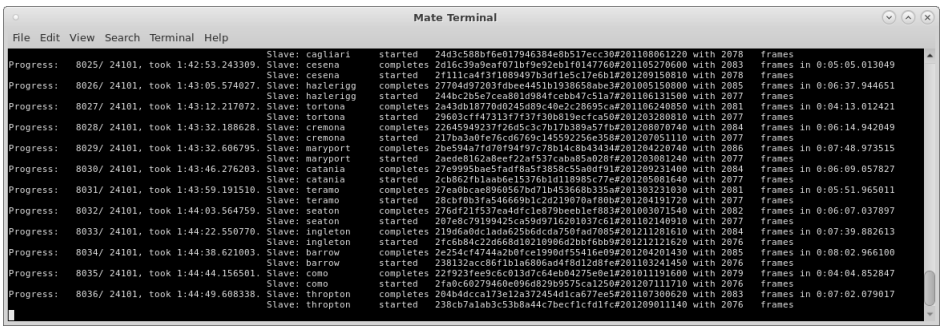

Figure 6.1: Running the MPI program on MATE terminal

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 distributing 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.

For all the ”slave” node, 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 student lab machines provided by The University of Edinburgh. It is important to make sure the cleaning procedure doesn’t affect other user’s work.

A short bash script combining `ping` and `ssh` is used to update the `$HOSTS` to a up-to-date list of available DICE machines.

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. This allows other students to work without disruption by the cleaning.

After testing it on the machines, it is reported that some machines are starting to become unresponsive after running code on it.

Since loading a video with 10,000 frames and all the library function used will take about 3 GB space in physical memory. After unreferencing every variable and force the garbage collecting mechanism, it still leaves 10% of the memory in use. This

eventually piles up and thrash the memory.

A solution is to spawn a sub-process and kill it after it's finished, but this increases the time to reload the python libraries, taking about 5 to 60 seconds per video.

6.3 Error Recovery and Progress Record

MPI allows a more scalable way of distribution of task on multiple computational nodes. However, one of the main disadvantage is the error handling, if one of the MPI process crashes, all of the process 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 master node a `DONE` signal, but with a failure message. The master node will remove the failed slave node and put it's 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" process 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 keeping the progress.

Chapter 7

Conclusion

7.1 Project Outcome

Initially the project goal is to further reduce the size of the RDS dataset from 1.6 TB to an acceptable size, however given the accuracy obtained from actual run, doing so would essentially causing 25% of the good detections lost in the process. Also due to need of maintaining consistency in the `.sql` file and video files, this final step 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 classifier used, taking about 260 GB disk space. Another one is the final binary array of decisions, marking each frame as good detection or else, this is more compact than the other, using about 750 MB disk space.

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

- A 323 GB folder, contains extracted `.sql` records.
- A 510 GB folder, contains normalized and PCA transformed first 100 features.
- About 20,000 marked ground truth dataset, covering the videos both Pugh and Yu misses during 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. Which allows easier visualisation on the dataset, without the need to go through multiple extraction and SQL connection stage in MATLAB.

7.2 Future Work

This project successfully applies the cleaning algorithms designed by Pugh[1], however it did not achieve the same level of accuracy as Pugh's thesis described. In order to increase the accuracy of the classifiers, following improvements will be needed.

7.2.1 More Annotation

As stated in Section 3.3, the current Ground Truth Dataset is heavily biased, which completely misses out the detections at some site of filming. Even after adding 20,000 detections across every site, the dataset still have 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 maybe using an online survey for ground truth, however it is not fully implemented at this stage of the project.

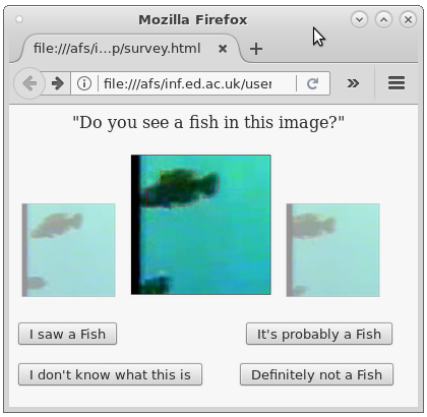

Figure 7.1: A prototype online ground truth interface

7.2.2 Training New CNNs

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

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

7.3 Final Words

Bibliography

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

[2] Daniela Giordano Lynda Hardman Fang-Pang Lin Robert B Fisher, Yun-Heh Chen-Burger. *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] 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.

[10] 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 **DoWork**(VideoID) function of slave code.

If the framework are used in work other than process the videos, (in this project's case: finding SVM parameters), the **VideoIDList** can be changed to adapt the new tasks.

Algorithm 1 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 2 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 4.2, a classification method based on file-name is proposed. If a video have over 30,000 frames and is filmed at a specific time and location, every frame of the video will be marked as non-fish. As each of video of RDS originate 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 case, such video are full of compression error, planktons, or having background extraction fault caused by moving vegetation. These cases were 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 detection such as ones in Fig B.4. This ensures the good fish lost will be less than 1% of the bad detection this method removes.

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

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

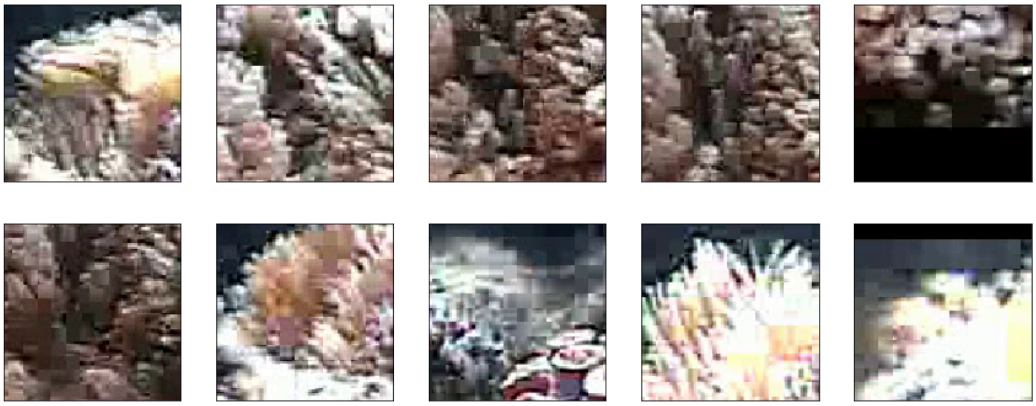

Figure B.3: Sample frames from a video with dynamic background.

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