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 Matt Pugh in 2015 on a massive parallel scale, to remove the large amount of False Positive fish detections in the Fish4Knowledge (F4K) dataset[2], without losing too many True Positives.

Also, 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 putting the process onto 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, improve the cleaning algorithm, and evaluating efficiency of different implementations of the machine learning techniques used. A more detailed roadmap this project is provided in the 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 finding out time answering my questions on the project, and precious advices 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

1 Introduction **1**

1.1 Document Structure 1

1.2 Fish4Knowledge Data Set 1

1.3 Classification Schema 2

1.4 Pipeline Classifier 4

1.5 Contribution 5

2 Background **7**

2.1 Big Data 7

2.2 Translation from MATLAB to Python 7

2.3 Message Passing Interface for Python (MPI4PY) 9

2.4 CPU Hogging and Memory Thrashing Prevention 9

2.5 Error Recovery and Progress Record 10

3 Data Source **13**

3.1 Extracted Images 13

3.2 SQL dump file 14

3.2.1 Standard Stream based Extraction Script 15

3.2.2 Translation of Binary Data 15

3.3 Ground Truthing Dataset 16

4 Preprocessing **17**

4.1 Early Video Removal 17

4.2 Frame Edge Indicator Function (FEIF) 19

4.3 Feature Extraction 19

4.3.1 Matthew's Feature 20

4.3.2 F4K Feature 20

4.3.3 PCA Analysis 20
4.4 Image Processing For CNN 20

5 Classification **21**
5.1 Support Vector Machines (SVM) 21
5.2 Convolutional Neural Networks (CNN) 21
5.3 Evaluation 21
5.4 Potential Candidate of classifiers 21

6 Conclusion **23**
6.1 Results 23
6.2 Future Work 23
6.3 Final Words 23

Bibliography **25**

A Sample of Lengthy Videos **27**

Chapter 1

Introduction

This project applies the previous work of Matthew Pugh[1] and Qiqi Yu[3] on massive parallel scale (detail in Chapter 2), while trying to reduce the computational cost of the cleaning algorithm. The main goal of this project is to produce a cleaned subset of a 1.6 TB dataset for future researchers.

1.1 Document Structure

Chapter 2 discussed some concept, designs used and the previous work 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 contains the conclusions and possible future work needed for the project.

1.2 Fish4Knowledge Data Set

The Fish4Knowledge (F4K) project, funded by EU's Seventh Framework Programme (FP7), studies environmental effects by analysing raw videos and extract information

about observed fishes 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 such as Nanwan National Park (NPP), Lanyu, and Houbi Lake (HoBiHu) in Taiwan. After 5 years of filming, the project recorded about 524,000 10-minute video clips, with a total size of 91 TB, and approximately 1.4 billion fish detection were found in the videos, we call this the F4K Original Data Set (FDS).

In attempt to reduce the dataset, F4K project developed and applied a species recognition algorithm, which extracts all detections as 100x100 RGB images and it's description files, reducing to approximately 839 million detections. These summary files have a combined size of 1.6 TB, this is called Reduced FDS (RDS), more detailed composition of these are described in Chapter 3.

The above reduction get rids of some of the False Positives (object that are not fish, recognized as fish) from FDS, unfortunately, there are still a lot of False Positives, a classification schema is created to identify the detections.

1.3 Classification Schema

According to previous work of Matthew, 10 different classes were used to mark the training dataset, and later used in different classifiers for fitting. Fig 1.1 shows manually picked example from each of these classes.

The 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 detection usually have a rigid square shape.

2 Illumination Artefact - Lighting changes recognized as fishes, some are caused by refraction of turbid water, some are caused by light reflecting planktons.

3 Background Vegetation - Some of the video are captured with dynamic background, where the swaying plants are recognized as fishes.

4 Others - Everything else, this includes large floating matters, empty contours created by previous algorithms.

5 Unknown - Because of reasons like lighting 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 fishes have good boundaries, and is 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 for the 100x100 boundary.

8 Bad Boundary - The fish is clearly captured, but the boundary extracted is useless for research.

III A Fish, but not useful - These frames detects fishes correctly, but misleading information may be extracted, it's unsure these frames should be kept or not.

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

10 Multiple Fish with shared contour.

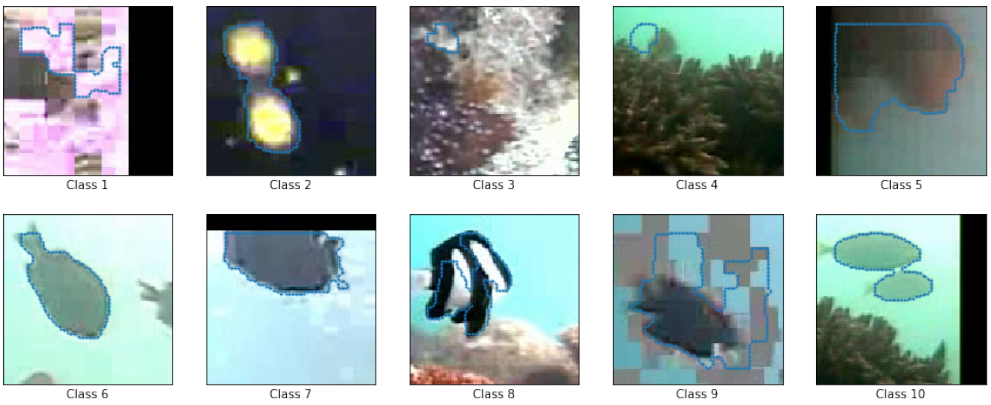

Figure 1.1: Example Detection From Each Class

1.4 Pipeline Classifier

The main part of the project is to translate and apply the pipeline classifier, designed by Matthew. However under limitations, the pipeline itself could not be applied directly on 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 were used to partition the SQL. This potentially cuts the runtime to about 10% of the original design, and removed the need of a server. More details of this modification is in Section 3.2.

To further reduce the cost of cleaning, and attempt of translating the pipeline into Python was used, however the effect isn't significant, more details are in background Section 2.2.

Before sending the data into the classifiers, preprocessing is needed to give a more sensible result, which is the slowest part of the pipeline after removing SQL server. In Matt's thesis, a Frame Edge Indicator Function (FEIF) is used to directly reduce the amount of frames need to classify, after improvements and new addition in Chapter 4, cleans out about 20% of the detection.

After extracting and reducing features, they are fed into 3 Convolutional Neural Networks (CNNs) and 10 Support Vector Machines (SVMs), then applying them a top N algorithm is used to produce final result. This part is mostly trivial since the classifiers are already trained by Matt. Some further evaluation about other potential classifiers are in Chapter 5.

Figure 1.2 below is a diagram of the work flow of the pipeline classifier:

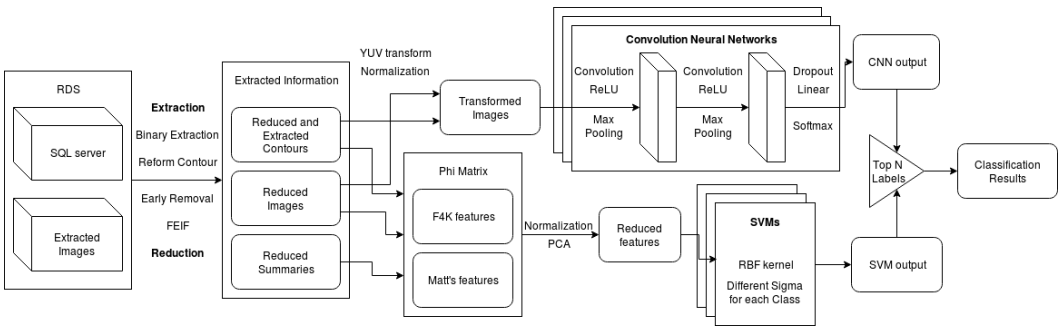

Figure 1.2: Pipeline Classifier designed by Matthew, modified

1.5 Contribution

During this project, the parallel task distribution programme is based on a public GitHub repository - mpi master slave - created by user ”luca-s”[4], minor changes were made to the work queue and protocol for crash recovery.

Data extraction pipeline written in Python was created to partition, extract, and parse the raw SQL file to comma separated values. Functions necessary for extraction and ones needed for visualization are written into two small libraries.

(TODO) Cleaned ground truthing dataset, as previous Matt and Qiqi seems to use different criteria for cleaning.

The pipeline classifier is re-written in Python, unfinished part of the original pipeline is implemented, some completed components in the pipeline like Matt's feature extraction metrics and algorithms were translated.

F4K project's feature extraction MATLAB code is untranslated, and is called within Python using PyMatlab library. One edge extraction algorithm were replaced and error handling were added to one of the unstable `getCstd()` function.

The classification step will use Matthew's trained SVM and CNN. (TODO) Since the ground truthing dataset isn't correct and complete, re-training might be needed.

(TODO) Ask bob if reconstruct of the SQL and video is needed, some of the FEIF rejected image still seems to be useful.

Chapter 2

Background

2.1 Big Data

Big data is one of the most hot 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 library with work distribution framework, for example: Hadoop Image Processing Interface[5], Apache Spark based 4Quant[6], and other tool kits for parallelization.

Due to the limit of the scale, the project could not use a dedicated server for the cleaning, the Distributed Informatics Computing Environment (DICE) with Scientific Linux 7 (SL7) are used instead. The DICE machines are provided by The University of Edinburgh, in this project, approximately 200-300 student lab DICE were used. Also, a 256 GB disk space quota on Andrew File System (AFS) was applied for this project.

2.2 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 seems to become an unreasonable solution. The F4K feature extraction code developed by Huang have about 5,000 lines of code in MATLAB.

Usually most of the MATLAB code have equivalent library function from SciKit-

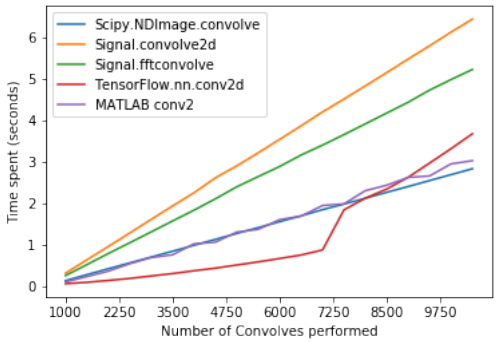

Figure 2.1: Test runtime of different 2D convolve algorithms.

Learn/Scipy, however most of Huang's work consist of customized part that could not be directly translated to Python.

For example, the Gabor Filter used by both Huang and Matthew, are written by Ahmad Poursaberi from Tehran University, a different implementation (where scale of the filter is rotated, while their variance isn't) is used. This essentially requiring the project to re-engineer the whole F4K feature library.

Full translation could take a few weeks for the F4K feature, also because of the cleaning algorithm only need to run once, expected increase in performance need to be calculated to figure out which approach is faster. After time recording on various part of the algorithm, it's identified one of the slowest operation is 2D convolution.

Figure 2.1 shows the test result of various 2D convolve algorithm. During the test, 2000 random 100x100 array is generated to simulate images used, and 5 random 5x5 filter is used, simulating both 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 is faster than MATLAB.

- TensorFlow's conv2d, while faster than all other library, it has a significantly high memory usage due to the tensor data type, potentially causing thrashing on machines. Also because of the high initialization time cost, it can't be used for each frame individually.
- Scipy.NDImage's convolve, giving almost same performance as MATLAB.

After above 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 part of the pipeline will be translated

to Python.

2.3 Message Passing Interface for Python (MPI4PY)

Since the project uses student lab DICE machine as processor nodes, the task distribution system need be scalable, as the machines available may fluctuate. A short bash script combining `ping` and `ssh` is used to update the `$HOSTS` to a up-to-date list of available DICE machines.

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

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

The python program will be started on above 4 machines, with the first one (basso) being the ”master”, distributing task to ”slaves”, which is every other node in `$HOSTS`.

While ”slave” node doesn't receive a `EXIT` signal, it will keep sending master `READY` signal, and after receiving work data from ”master”, it sends a `DONE` signal back to ”master”, and then goes back to the `READY` loop. The ”master” creates a work queue of all the videos need to be processed, and send it to ”slave” with `START` whenever it receives `READY`, it keeps sending the work until the work queue is empty, when every work is marked as finished, it broadcasts `EXIT` signal.

For all the ”slave” node, Python's MultiProcessing and MATLAB's ParPool is used to fully utilize every core's computational power.

2.4 CPU Hogging and Memory Thrashing Prevention

The project take place on public student lab provided by The University of Edinburgh, it is important that the cleaning algorithm is only running in the background, and not affecting other user's work.

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

The process is further tested on an empty lab with 20 machines, after running the extraction overnight, the lab is only slightly warmer than other labs. This problem could be solved simply by ventilation system installed in the lab.

Another problem is more problematic, loading a video with 10,000 frames and all my library function will took 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.

2.5 Error Recovery and Progress Record

The project is running on massive scale, 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" a `DONE` signal, but with a failure message. The "master" will remove that "slave" node and put the job it fails and put it back to the start of the work queue, allowing it to be re-scheduled again.

In 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.

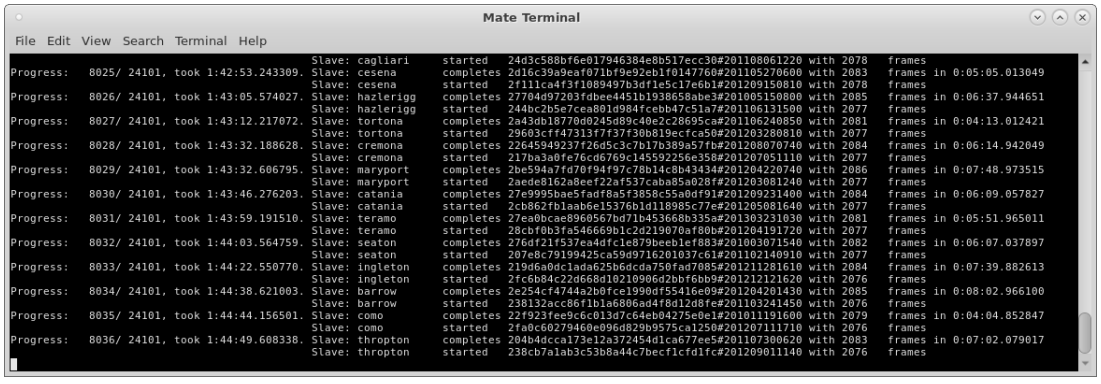

Figure 2.2: Running the MPI program on MATE terminal

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 3

Data Source

After the species recognition of the F4K project, three types of output file is 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 is 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 area 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.

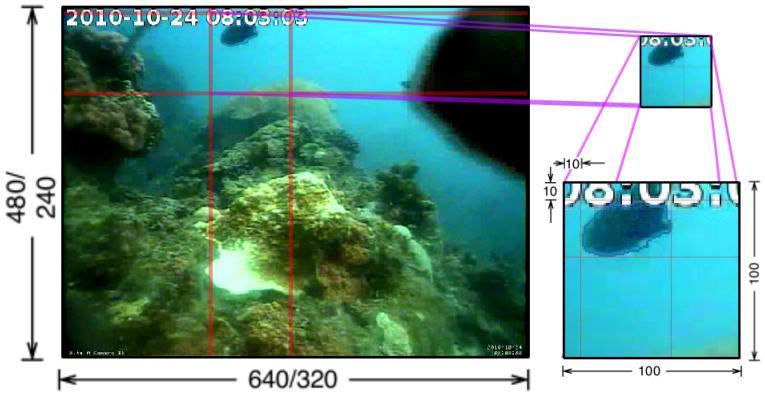

Figure 3.1: Process of Extracting Image

3.2 SQL dump file

The `.sql` dump file comes from the SQL workflow of the F4K project, which means not only details of fish detection is 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 are actually needed in 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 file 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 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 dump file into directly usable files.

3.2.1 Standard Stream based Extraction Script

Since each record needed for the cleaning are independent (given they have different video_id). If each detection are stored into a corresponding file with it's 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 almost negligible amount. In Qiqi's estimate, the loading and preprocessing would took 800,000 hours (25,000 hours on a 32 core machine) in original design (double the time if considering her calculation error). After the extraction it only took about 0.3 second 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.

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 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 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 Truthing Dataset

In order to train and evaluate the classifiers, Matt and Qiqi manually marked a set of detections with the schema above. They chose a subset of the RDS, which is all the videos having id start with ”13b”. This subset of the RDS consists of 97 video files, and 154,042 detections.

However the marking was not able to cover all of the detection because of it’s size. Moreover, some set of the detection is 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 problem) 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.

Some special videos this ground truthing dataset doesn’t cotain was included, for example, video with high transmission error rate at Lanyu Site, videos filmed at stormy times (blurry camera) at HoBiHu site, and a video full of illumination artefact at camera 4 of NPP-3 site.

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 method on the dataset is used before extraction of the feature, Early Video Removal was introduced and improved version of Frame Edge Indicator Function developed by Matthew was used.

4.1 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 video took 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 minute, means that in every frame of the original video, an average of 10 detection is extracted. By looking into these ”outlier” videos, some patterns were found:

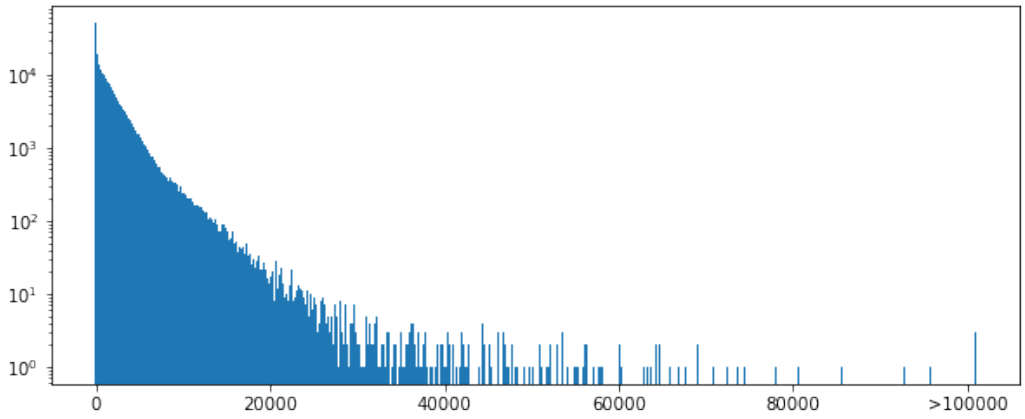

Figure 4.1: Log Scale Histogram of Detection Length

- Both Camera at Lanyu site are night-vision cameras, when they film during the night, a lot of light reflecting planktons 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 happens at NPP-3 site camera 2 during June 2012 to August 2012. The camera seems to be falling 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 with 200,000 detection, consists of lots of repeating frames, possibly caused by bugs in previous extraction processes.

There are also some good video 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.

Using the above patterns, 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. Samples of rejected videos are in Appendix A.

4.2 Frame Edge Indicator Function (FEIF)

In Matthew's thesis, 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 pixel, then increasing the limit of 25 to 40, and added new restriction: reject all the points with 25% of the points touching the boundary. About 15% of the dataset is rejected by it, saving about 400 days of computational time.

4.3 Feature Extraction

During the species recognition of the F4K project, 2626 features were used for computing detection certainty, on that basis, Matt added 29 new features focused on the edges of the contour. Then dimensionality reduction is applied to the features to reduce dimension of 2655 to 28.

This process takes about 0.3 seconds for each frame, cutting down the components 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.

4.3.1 Matthew's Feature

This 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 where translated using Numpy library to speed it up.

Animation Score 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 part isn't very useful because of the dynamic background of the image.

Boundary Curvature, (TODO) I'm not actually sure how these works but my translated code produce the same result.

$E = mc^2$

4.3.2 F4K Feature

4.3.3 PCA Analysis

4.4 Image Processing For CNN

In Matt's pipeline classifier, 3 different CNN 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 all 3 type of images, and transformation to YUV color space is used, then the Y channel is normalized with mean and variance of ground truth dataset. A final color space of Y'UV is used for the CNNs.

Different to SVM feature, preprocessing for CNN took almost no time using OpenCV's image transformation.

Figure 4.2: Images on different stage of pre-process

Chapter 5

Classification

5.1 Support Vector Machines (SVM)

5.2 Convolutional Neural Networks (CNN)

5.3 Evaluation

5.4 Potential Candidate of classifiers

Chapter 6

Conclusion

6.1 Results

6.2 Future Work

6.3 Final Words

Bibliography

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

[2] Fish4knowledge homepage. http://fish4knowledge.eu/. Accessed: 2018-01-08.

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

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

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

[6] 4quant homepage. http://4quant.com/. Accessed: 2018-01-10.

Appendix A

Sample of Lengthy Videos

The videos below are some sample frames.

Words

Paragraphs

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

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

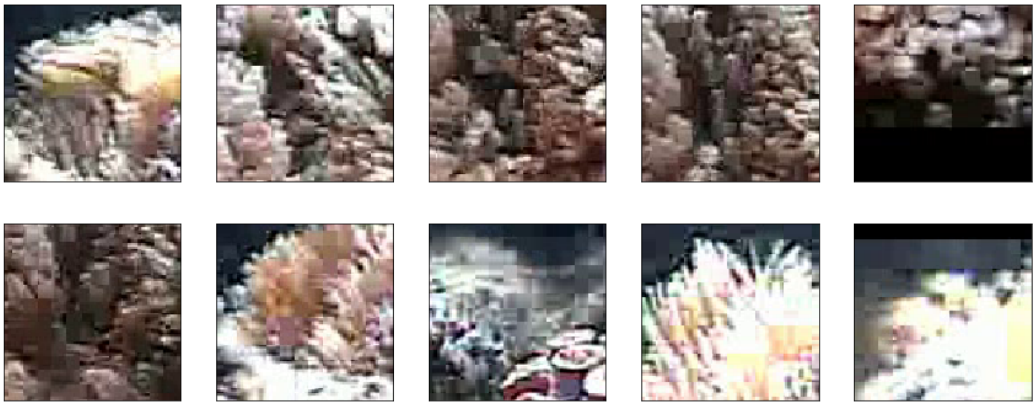

Figure A.3: Sample frames from a video with dynamic background.

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