B.Sc. Computer Science
Computer Science Department

An Incremental and Parallel Aerial Disaster Data Crawler

Candidate

Andrew Hin Chong Yau

Student ID 700012512

Supervisor

Dr. Chunbo Luo

University of Exeter

Academic Year
2022/2023

Abstract

The state of the current climate is causing the weather to become progressively volatile to the extent certain regions can no longer handle the intensity and frequency of natural disasters. As prevention is growing less viable, the knowledge to help mitigate disasters is an asset. The web is now the central wealth information in the world but most of it is disorganised and unusable. This paper implements web crawling methods to organise and coalesce aerial disaster drone images to be practically used during disasters and gather sets of data to be utilised for research and development.

Contents

List of Figures

List of Tables

1 Introduction

The objective of this project, specified in the literature review is to create a web crawler that efficiently gathers raw aerial drone data of disaster sites available within the world wide web, seeking to support two facets of humanitarian and disaster relief management fields[1]:

- Disaster Preparation: Gathered drone images can be utilised as data sets to train AI specialised in enhancing real-time disaster support operations. Examples include AI that can identify where victims are most like to be located, locations which are still risk factors or accessibility of different forms of transport, allowing on-site operators to make the most informed responses possible.
- Disaster Response: Up-to-date data is continuously and swiftly gathered due to parallel and incremental features, allowing aerial images taken by 3rd parties of an on-going disaster to be acquired and utilised by on-site operators as soon as possible, dramatically increasing the data coverage of the disaster area from different viewpoints.

1.1 Motivation

According to the Centre for Research on the Epidemiology of Disasters (CRED) [2], 7,348 natural disasters (natural hazards resulting in significant deaths, homelessness and economic loss etc.) were recorded globally by the Emergency Events Database (EM-DAT) spanning 2000-2019, whilst between 1980-1999, 4,212 natural disasters were recorded. This indicates a 75% increase in natural disasters within 20 years, and the annual increase is predicted to grow[3] from 400 in 2015 to 560 by 2030 equating to a 40% increase.

The increased disaster frequency[2] may render climate change adaptation and current national and local strategies for disaster risk reduction obsolete in many countries. Therefore, the bottleneck in disaster prevention generates a need to optimise humanitarian and disaster relief actions in preparation to mitigate the effects of the inevitable disasters when they occur.

Progress on humanitarian and disaster relief methodologies for immediate responses post disaster at present are centred around AI that is capable of supplying data and suggestions to on-site operators efficiently. In order to train these AI to generate models adequate for real world use, a large quantity of real world disaster data is a core requirement.

The largest source of almost all data including disaster data and images is the web, composed of nearly 2 billion websites[4, 5] with Google indexing 30-50 billion individual web pages from these sites [5]. In addition the web is growing at an exponential rate doubling every 9-12 months and the changing rate of web pages with small changes is approximately 40% weekly, with a change rate of 7% weekly for web pages changed by a third or more [6].

Web crawlers are the systems for coalescing this data and for this project. The intended web crawler implementation is a focused, parallel and incremental hybrid web crawler with the main focus being an incremental characteristic since aerial drone images of natural disasters are not posted or updated regularly. This design will attempt to accrue the best URLs on the web that provide the most disaster images at a constant rate. Parallelism is utilised in order provide scalability and improved efficiency by increasing download rates, enabling the web crawler to adapt to the assumed disaster data growth accompanying web growth. The ′focused′ aspect is accredited to the targeting of aerial disaster imagery and URL quality identification.

1.2 Aims

This report will display the implementation and research on the efficiency of a hybrid web crawler that attempts to combine fragments of three different types of web crawlers, considering the trade off between the advantages and disadvantages of every feature, and skewering the advantages towards specialisation pertaining to disaster images. The following characteristics will be evaluated:

- Portability and Accessibility - The executability of the system should be designed to be easily transferred to another device, requiring the least amount of setup (such as APIs) possible without sacrificing performance in order to manipulate large amounts of data [7]. This is done to match the resources available on an individual scale, allowing research to be done on the effect of the methodologies used at a small scale prior to any potential expansion to vaster applications or big data.
- Scalability and Reliabilty - The system should effectively assign its resources to maximise the download rate of data and a steady influx of data should be obtained.
- Accuracy - The ratio of disaster images to non-disaster images should be as high as possible, reducing the preprocessing requirement of the generated data sets. Thereby, accelerating the training and modelling pipeline for a user's AI.
- DoS Mitigation and Conscientious Application - The network load targeting specific web domains must be minimised in order to prevent any possible damage to the services and any information obtained from domains that convey restrictions on crawling must be obeyed.

2 Project Specification

2.1 Requirements

This chapter will encapsulate the fundamental idea of what every functionality in the web crawler does, why they are required, and finally what characteristics (referred to in the aims 1.2 section) will be optimised.

2.1.1 Breadth First Search Crawl Module

The crawl module is the foundation of the system which retrieves all the web pages data and extracts the relevant disaster and URL information. Therefore, as the most heavily utilised functionality, maximising efficiency in terms of speed and reliability is the key focus of this program.

2.1.2 Database to Store Images and URLs

The database system is required to store all the disaster images and the information of individual URL required to access them and determine their authority on disaster images. The information data must be sorted and retrieved when called throughout and manipulated after every crawl. Therefore, it should be well structured and condensed, storing all the data in the smallest types possible. The information of saved images must also be saved, enabling the images to be copied into a data set according to the requirements specified by the user.

2.1.3 Focused Web Crawler

The web crawler must be capable of identifying disaster images to a high degree of accuracy since it is the only method of quantifying the relevance of a URL. This is done by applying a binary classification model which identifies whether an image scraped from a website is a disaster image or not. The requirements for this functionality are a convolutional neural network architecture model and a disaster images data set in order to train the model. The model's accuracy is determined by the quality of the data used and its compatibility with the model.

2.1.4 Incremental Features

The quality of every URL must be determined in order to sort all URLs in rankings, where higher ranked URL will be searched again after a certain time frame. The method of calculating this must be balanced in order to prevent the values of older and proven URLs from increasing continuously due to time, blocking newer valuable URLs from being assigned their proper significance. High quality web pages are likely to be continuously updated but the time will vary for each page. If the optimal time for each update can be predicted, an efficient and reliable source of disaster images is built, with unnecessary traffic towards each page being mitigated.

2.1.5 Crawl Module Parallelisation

The parallelisation of the crawl module is essential as it is the main method of maximising the crawl and download rate. The overhead should be minimised by selecting a suitable parallel method between multi-processing and threading. In addition, a reasonable number of workers must be selected in order to attain the optimum performance as too many workers would generate a large amount of overhead due to context switching between each other.

2.1.6 Dataset Generator

The capability of copying images that match specified time frames to a directory is necessary for users to easily obtain the correct data for use in their own applications. This will be a standalone program that may be run separate from the web crawler, thereby allowing new images to be immediately copied without stopping or re-initialising the web crawler if it is active.

3 Design

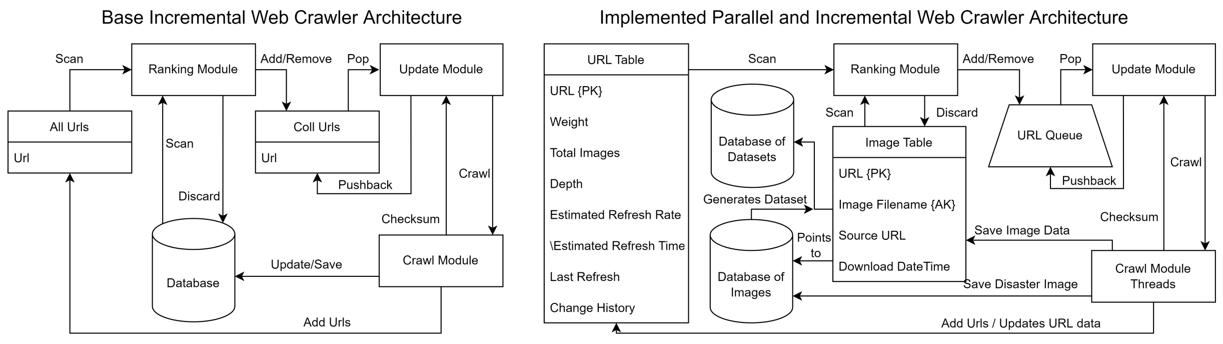


Figure 3.1: Base and Proposed Incremental Web Crawler Architecture

This chapter specifically details how every functionality is implemented in depth, beginning from the general implementation of features and subsequently detailing the proposed experimental design which expands on the fundamentals and differs from other works.

3.1 Base Web Crawler

A basic web crawler [8] traverses the web similar to expanding a graph search consisting of nodes (web pages) and edges (URLs). The crawler downloads the web pages by following the hyperlinks found in individual pages, starting with a few seed URLs (seed URL queue), and then reaching out to the other URLs (added to queue) extracted from the seed URLS. This process is repeated until the crawler decides to stop, or specified end conditions are met.

The foundational design of this web crawler utilises search engines in order to begin extracting urls from the internet using keyword driven queries of specified disasters and tags. Search engines will be used as they are resources that identify the quality of URLs by applying the PageRank algorithm [9] to URL usage data such as the URLs clicks and the query or category searches in the search engine's database. This reduces the 'embarrassment metric' optimisation mathematical model[10] which refers to the potential of the queried information found being irrelevant to the topic.

Overall, this will generate the best set of seed URLs that theoretically possess the highest relevance to disasters with simple URL requests and the least amount of resources and time possible. Since these URLs have the highest relevancy, it may be assumed that the relevancy of URLs extracted from them are also relevant to disasters, but the relevancy will decrease at a linear rate from the seed URL. Following this assumption, these URLs are scraped in a breadth first search[11] fashion, implemented by assigning a level value to every URL found (1 + Parent URL Depth) with seed URLs being initialised with a depth of 0.

3.1.1 CNN Binary Classification Model

Classifiers are components of focused crawlers [12, 8] that categorise web pages by quantifying the information scraped such as the number of times certain words appear in the HTML or the actual structure of the HTML etc. These all contribute to the prediction of a web page's relevancy, therefore determining whether a URL should be searched or not. Differing from these, the proposed design uses a deep convolutional neural network (CNN) binary classifier which will be trained to identify, count and save all appearances of disaster site images on a web page.

A CNN learns [13] when an image is passed through a convolutional layer where it can be defined as a 2D matrix of ones and zeros for each pixel depending its RGB value. Features in the image as convolution feature values can be calculated by multiplying fragments of a specified dimension of pixels in the image matrix by a convolution filter matrix. The size of the filter is defined in the layer and a convolution feature matrix is generated by applying this operation throughout the image by shifting by a pixel for every feature.

The activation function ReLU is subsequently applied to the feature matrix, removing negative features from the matrix. Pooling through a 2D pooling layer is then used to reduce the dimensions of the value matrix by half, storing the highest feature in each corresponding segment, causing features present in the image segment to be more well defined.

The CNN for this project will run this procedure 3 times in a row to promote the learning of each feature but with a low number of filters since larger parameters would cause more features to be learnt than necessary in order to classify an image into 2 classes, disaster image or not. This would also reduce the amount of overhead required for this computation, which is highly beneficial when applied to every image scraped on the web.

Finally, all of the feature maps are flattened into a linear vector of features which can be condensed using two dense layers into a single float value between 1 and 0 using a sigmoid function where 0.5 defines the border for the binary classes.

3.2 Focused

Utilising the trained model, every image scraped from the web must be classified within the crawl module and tallied to update the weight of the URL. Following the update of the URL, the new hyperlinks found on the site will be added to a URL table with a fraction of the newly updated URL's weight. This weight is determined by an inheritance coefficient since the children of a relevant URL should me more likely to be relevant.

3.3 Incremental Features

An incremental crawler [8] continuously crawls the web, with a priority on maintaining the freshness of the local collection of URLS and improving the quality of the local collection which is revisited periodically according to a calculated priority. During its continuous crawl, low quality pages in the local collection may be purged to make space for new pages which may contain higher quality pages.

As depicted in Figure 3.1 it is split into three main modules (Crawl Module, Ranking Module and Update Module). All URLs records all discovered URLs, and Coll URLs records the URLs that are/will be in the Collection. Coll URLs contains all the URLs which are already crawled and is implemented as a priority-queue in which URLs are kept according to their priority score. The Ranking Module chooses URLS to be added to Coll URLs after considering All URLs and the Collection to make the refinement decision. It must also schedule a replacement task when a page not present in Coll URLs, is given a higher rank than a page within Coll URLs. This new URL is placed in Coll URLs with highest priority, so that the Update Module can crawl the page immediately.

3.3.1 Ranking Module

The ranking of a is dependent on the weight of the URL URLs with the highest weights are scanned from the URL table and stored in a list, where the number of URLs scanned is determined by the initialised URL queue size. The URLs in this list are cross checked with the URLs in the URL queue and deleted if the URL already appears in the queue. Values in the URL queue but not in the list of ranked URLs should be purged since these URL have fallen out of the rankings and would cause the specified URL queue size to be exceeded. Following the purging, all of the new ranked URLs are pushed into the URL queue with the priority of each URL being the seconds from their estimated refresh time to the current time.

3.3.2 Update Module

The priority of the URLs in the URL queue is solely dependent on the estimated refresh time of the URL, where the highest priority URL has exceeded its refresh time from the current time the most. The estimated refresh time of a URL is calculated using the formula[14]:

$$\Delta t = (1 - p_c/p_g) * u(p_c - p_g) + (1 - p_c/p_l) * u(p_l - p_c) * t_n \tag{3.1}$$

where:

$$t_n = CurrentRefreshTime \tag{3.2}$$
$$p_c = HistoricalDegreeOfSuccess \tag{3.3}$$
$$p_l = LowerBound \tag{3.4}$$
$$p_g = UpperBound \tag{3.5}$$

$$u(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

This formula is designed to adjust the refresh time of each URL depending on the historical data of the 10 most recent refreshes. Historical data is stored for every URL in the database as a string. '1' is appended to the start of the string whenever a new disaster image is added to the database and '0' is added otherwise. The number of '1' values multiplied by 0.1 provides the historical degree of success utilised in the formula. The lower and upper bounds define the ranges of success for the refresh time to increase or decrease; exceeding the upper bound will decrease the time and dropping below the lower bound will increase the time. This formula will only be applied onto the refresh time of a URL when the historical search data is full.

The fact that the weight of a URL is always increased by the number of new disaster images obtained, causes an inequality between the rankings of old and new URLs as the old URLs may accumulate weight beyond their actual quality. In order to counter this issue, I have decided to implement the use of a decay coefficient in conjunction with the refresh time equation designed by Sharma[14].

This decay coefficient is a constant that is multiplied against the weight of the URL under the condition that the historical success exceeds the lower bound. The exceeding of the lower bound means that the correct refresh time has been found where at least 1 disaster image is obtained with a 70% chance per refresh, thereby increasing the URL weight. The decay coefficient will counter this increase, keeping the true value of the URL in equilibrium. It is also applied when the historical degree Of success is 0, since no new images we retrieved in 10 attempts, the URL is assumed to be 'dead' and therefore, the weight should decrease pushing it out of the rankings

The update module will continuously pop the highest priority item from the URL queue and returns the popped URL if the estimated refresh time has passed. If the refresh time has not passed, the URL is pushed back into the queue and the priorities of the entire queue is updated. If the queue is full and none of the refresh times have passed, the crawler has learnt most of the best URLs in it's collection. At this point, the crawling scope of the crawler has fully transitioned from breadth first search into best first search, causing the system to become more reliable with a constant influx of images as new disaster images should be extracted every scheduled refresh time. Whilst no queued URLs are being refreshed, the resulting free resources are used to scrape new URLs in the system to continue retrieving images and potentially discover new URLs with a quality.

3.4 Database

The database's functions for adding and updating URLs, immediately run the underlying calculations required for the incremental ranking and update modules, stores them. These calculation refer to the URL weight updates for ranking and a URL's estimated refresh time and rate. This allows the state for the entire crawler to be saved and therefore, be fully reinitialised whenever the system is restarted. The decreased initial weight via parent URL depth defined as a base breadth first search functionality, increased initial weight defined by the focused functionality and decay coefficient weight multiplier from the incremental functionality are also immediately applied. Overall, the centralisation of the manipulation of all of the core variables in the system, grants clarity to the relations between all of the data since it is easy to observe. Any mathematical inconsistencies should also be viewable.

Differing from the general design of an incremental database, the URL Table (All URLs) is stored as a table within the database instead of being a list, this allows all of the data required for ranking to be immediately sorted after being called from the database.

Every URL in the proposed URL table possesses 8 values as shown in Figure 3.1:

- URL (String): The full hyperlink URL serving as the primary key
- Weight (Integer): The weight and heuristic of the URL serving to rank the URL
- Total Images (Integer): The total images scraped by the URL alone.
- Depth (Integer): The distance of the URL from the seed URL, equivalent to the number of parent URLs required to traverse back up the web graph.
- Estimated Refresh Rate (Integer): The number of seconds required to pass before refreshing the URL.
- Estimated Refresh Time (Datetime): The actual time that must be exceeded to refresh the URL.
- Last Refresh (Datetime): Records the date and time of the last instance of the URL refreshing. Therefore, URLs with null values have never been scraped.
- Change History (String): The history of the URL is a binary value stored as a string, which is updated every time the url is refreshed by the web crawler.

3.5 Parallel Design

Parallelism is achieved using threads

4 Development

4.1 Technology

4.1.1 Database

Diverging from the initial design specified in the literature review, in which the third party MySQL database resource is used to store the scraped images and relevant data. The scraped images are stored normally within a specified directory and dataframes from the python pandas library are used to easily and efficiently manipulate the data. The pandas library is used as it is very fast and efficient with small amounts of data to up to 1GB of RAM [7]. When implemented using the URL table and Image tables as pointers with the most efficient variable types as specified, over 8 million URLs can be handled without and significant performance drops. The overall versatility is also vastly higher [15] regarding the functions it can apply and mutability of dataframes.

Finally, the MySQL database has a low portability due to requiring set up unlike pandas since it requires the SQL database itself to be installed, initialised and connected to prior to use. The MySQL database is far superior in terms of scalability and performance will large amounts of data but at an individual scale, pandas more suitable for this implementation,

4.1.2 Python Requests Library to Selenium

Post observation of the initial performance of the web scraper and manual cross checking scraped image and hyperlink URLs, it could be observed that many URLs are never obtained. This is due to the loading of page content dynamically using JavaScript where the data is not present in the base HTML of certain websites. In order to combat this issue, the Selenium package is utilised to create a web driver which accesses URLs with a browser, allowing the URL pages to load and be subsequently scraped after a specified time frame.

4.1.3 PriorityQueue Class Object to Heap Queue Algorithm

The usage of the PriorityQueue class object from the queue library for the URL queue was succeeded by the base heap queue algorithm[16] as it boasts a far higher versatility since it is stored as a list and therefore manipulated like a list. This enables a simpler and more efficient update procedure of priority values since the index of specific URLs in the queue can be located and updated directly, rather than popping URLs from the queue and re-pushing them with updated priorities. URL values can be obtained using list comprehension for cross checking with ranked URLs and finally, queue items can be removed without popping URLs by using list indexing.

4.1.4 Crawl Module

The Selenium library is used to generate a web driver to access the specified URLs and obtain the URLs dynamically loaded HTML source code. The Beautiful Soup package is then used to parse the retrieved HTML code, so the hyperlinks and image URLs can be identified. The image URLs are extracted under the condition that the images are JPEG, JPG or PNG file types and kept if the trained disaster model identifies the image as a disaster, allowing an entry for the image to be stored in the image table. The URL data for the scraped URL can now be updated in the URL table as the number of new disaster images is known. The regular hyperlinks found on the page can then be filtered for black listed URLs and added to the URL table if they are not in the table.

The largest drawback of this method is the dependency on every page's individual load times [17] which is are dependent on a multitude of factors such as slow running JavaScript, high network load or ongoing traffic to a URL or generally slow network connection speeds either client or server side. Therefore, to keep the weights of all of the URLs as correct as possible. Every crawler thread will wait for 5 seconds prior to data extraction.

4.2 CNN Binary Image Classifier

4.2.1 Disaster Images Dataset Acquisition

Two data sets were acquired for the training of the model by downloading images from google. The target images to be classified were; earthquakes, landslides, volcanic eruptions, floods, hurricanes, tornadoes, blizzards, tsunamis, cyclones and finally wildfires. The first data set consisted of 250 images for each category that were manually classfied, totaling 2500 images. The opposing image set for the not disaster classification was completely random but with the same quantity. This was done in order to give every type of disaster an equal chance to be learnt.

The second data set relied entirely on google's image search engine to find disaster images using the keywords, and filtering different image sizes for each query. This dataset accrued 16000 images but were not manually checked, this was done to test if a higher quality model could be trained as a dataset of this size may theoretically cause the non-disaster images that managed to enter the set to be less significant in the overall training.

4.2.2 Overfitting Handling

4.2.3 Batch Normalisation

[18]

4.2.4 Dropout

[19] Dropout of 0.5 between every hidden layer and kernel constraints of 3

This model suffers from overfitting in which the model transitions from learning how to identify the class of an images in general, to specialising in identifying the class of images in the dataset. This presents an issue where new images classified by the model are incorrect which is represented as the validation loss of the model

5 Testing

The testing of the CNN Binary Image Classifier performance over 2 Datasets
The testing of the overall web crawler

6 Results

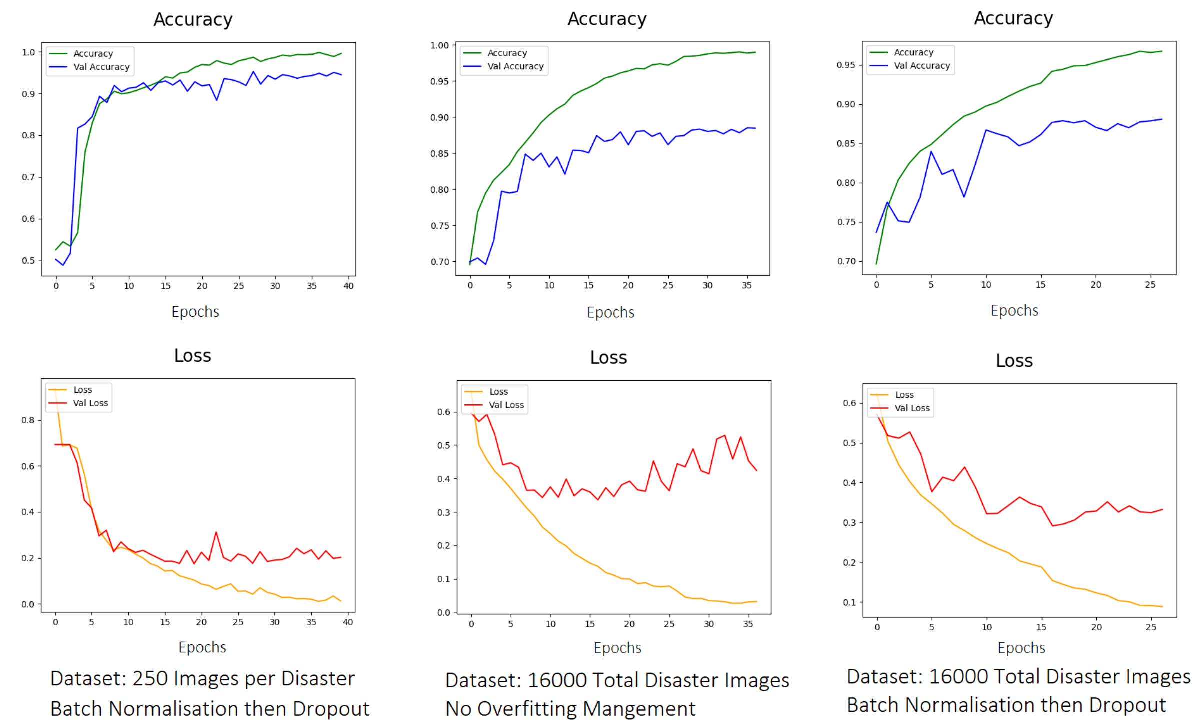
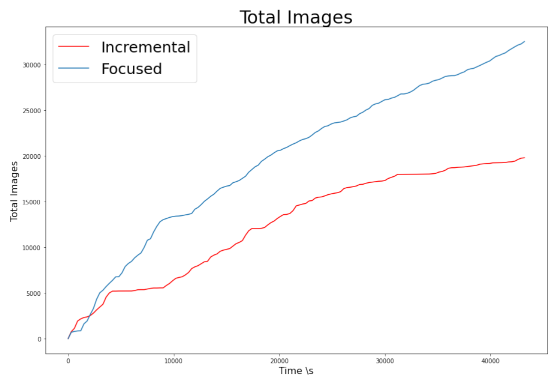


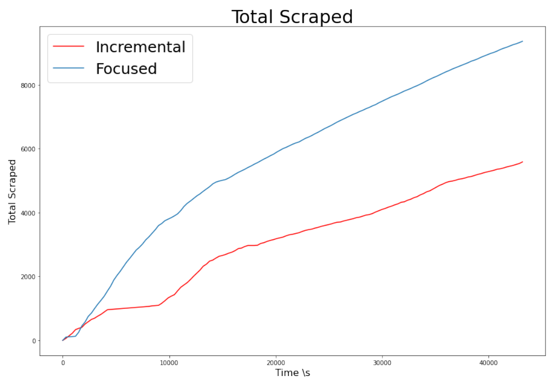
Figure 6.1: CNN Binary Classifier Datasets and their Performance

CNN Lowest Validation Loss Epochs				
Dataset	Loss	Accuracy	Validation Loss	Validation Accuracy
250 Images per Disaster	0.0417	0.9868	0.1762	0.9526
16000 Disaster Images Overfitted	0.1357	0.8738	0.3376	0.9465
16000 Disaster Images	0.1262	0.9499	0.3534	0.8788

Table 6.1: CNN Lowest Validation Loss Epochs



(a) Total Images



(b) Total Scraped

Figure 6.2: Incremental and Focused Crawler Performances

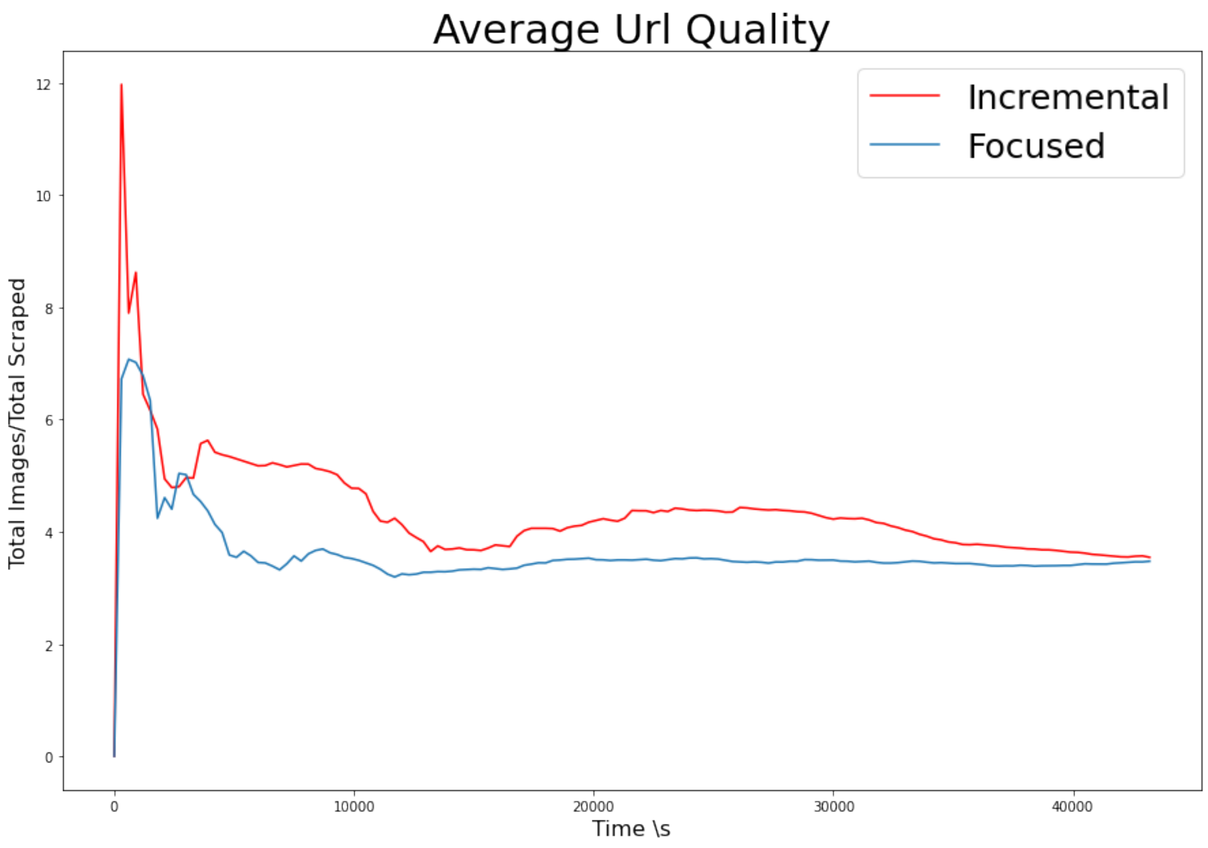


Figure 6.3: Average Url Quality

7 Evaluation

6.1 This model suffers from overfitting in which the model transitions from learning how to identify the class of an images in general, to specialising in identifying the class of images in the dataset. This presents an issue where new images classified by the model are incorrect which is represented as the validation loss of the model

7.0.1 Overfitting Handling

7.0.2 Batch Normalisation

[18]

7.0.3 Dropout

[19] Dropout of 0.5 between every hidden layer and kernel constraints of 3

8 Conclusion

8.1 Future Directions

The development of a secondary data storage system integrated with the current program or a complete alternate version with the current program as the base would be beneficial for

References

[1] Wenjuan Sun, Paolo Bocchini, and Brian D. Davison. “Applications of artificial intelligence for disaster management”. In: *Natural Hazards* 103.3 (2020), pp. 2631–2689. DOI: 10.1007/s11069-020-04124-3.

[2] Nima Yaghmaei. *Human cost of disasters: An overview of the last 20 years, 2000-2019*. United Nations Office for Disaster Risk Reduction, 2020, pp. 7–11.

[3] United Nations Office for Disaster Risk Reduction. *Global Assessment Report on Disaster Risk Reduction 2022: Our World at Risk: Transforming Governance for a Resilient Future.* Tech. rep. 7bis Avenue de la Paix, CH1211 Geneva 2, Switzerland: United Nations Office for Disaster Risk Reduction, 2022.

[4] Ogi Djuraskovic. *How Many Websites Are There? – The Growth of The Web (1990–2022)*. Sept. 2022. URL: https://firstsiteguide.com/how-many-websites.

[5] Deyan Georgiev. *How many websites are there in 2022? [updated guide]*. Oct. 2022. URL: https://techjury.net/blog/how-many-websites-are-there/.

[6] Rahul Kumar, Anurag Jain, and Chetan Agrawal. “SURVEY OF WEB CRAWLING ALGORITHMS”. In: (2016). DOI: 10.5281/zenodo.3674658. URL: https://dx.doi.org/10.5281/zenodo.3674658.

[7] Admond Lee. *Why and how to use pandas with large data*. Nov. 2018. URL: https://towardsdatascience.com/why-and-how-to-use-pandas-with-large-data-9594dda2ea4c.

[8] Shruti Sharma and Parul Gupta. “The anatomy of web crawlers”. In: *International Conference on Computing, Communication Automation*. 2015, pp. 849–853. DOI: 10.1109/CCAA.2015.7148493.

[9] Joost Berkhout. “Google’s PageRank algorithm for ranking nodes in general networks”. In: *2016 13th International Workshop on Discrete Event Systems (WODES)*. 2016, pp. 153–158. DOI: 10.1109/WODES.2016.7497841.

[10] H. Bullot, S.K. Gupta, and M.K. Mohania. “A data-mining approach for optimizing performance of an incremental crawler”. In: *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*. 2003, pp. 610–615. DOI: 10.1109/WI.2003.1241279.

[11] Abhinav Garg, Kratika Gupta, and Abhijeet Singh. "Survey of Web Crawler Algorithms". In: *International Journal of Advanced Research in Computer Science; Vol 8, No 5 (2017): May-June 2017; 426-428 ; 0976-5697 ; 10.26483/ijarcs.v8i5* (June 2017). URL: http://www.ijarcs.info/index.php/Ijarcs/article/view/3321.

[12] Anish Gupta and Priya Anand. "Focused web crawlers and its approaches". In: *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*. 2015, pp. 619–622. DOI: 10.1109/ABLAZE.2015.7154936.

[13] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[14] A K Sharma and Ashutosh Dixit. "Self Adjusting Refresh Time Based Architecture for Incremental Web Crawler". In: *IJCSNS International Journal of Computer Science and Network Security* 8.12 (Dec. 2008), pp. 349–354.

[15] Aditya Parameswaran. July 2022. URL: https://ponder.io/pandas-vs-sql-food-court-michelin-style-restaurant.

[16] Mokhtar Ebrahim. Aug. 2022. URL: https://likegeeks.com/python-priority-queue/.

[17] pingdom. June 2021. URL: https://www.pingdom.com/blog/website-load-time-in-2020/.

[18] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[19] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.