Advertisement detection system using machine learning techniques.

Final Project Report

DT211c
BSc in Computer Science (Infrastructure)

Jack Duggan

C16350866

Brian Gillespie

School of Computer Science

Technological University, Dublin

11/04/20

Abstract

This project revolves around the detection of advertisements using machine learning means, creating a system that can apply these means and the evaluation of that system. The motivation behind this system is the turmoil in advertisement deliverance. The use of advertisements to track users as they navigate the Internet and data privacy concerns that arise because of this. Further research into this space has revealed advertisement blockers use the same technology to complete their task. New methods of completing this task should be developed.

This report describes how this system was designed and how the design will be implemented. The backend will utilise two main machine learning algorithms, word2vec, the inception model and how the outputs of these models will be compared. The system will be used through a chrome plugin implemented in JavaScript. The middle layer will then connect these algorithms and the front-end using the flask framework.

The final system will be a chrome plugin that states if there are advertisements on a website and presents a metric for how sure the system is.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

[illegible]

Jack Duggan

09/04/2020

Acknowledgements

I would like to acknowledge the support I have received during the undertaking of this project.

My supervisor, Brian Gillespie, for guiding this project.

For the continued support of my friends and family.

And finally, my fellow students for the sense of comradery and support during my time at TUDublin.

Table of Contents

Table of figures

1. Introduction

1.1. Project Background

In January of 2019, Google proposed a change in their extension manifest version (v3) (1). The point of this update is to make extensions safer by adding greater permission controls. This however had the knock-on effect of "killing" ad block support according to Raymond Hill, the maintainer of uBlock origin and uMatrix (2). A conclusion therefore can be made that extensions cannot be used reliably to protect privacy while browser creators have a vested interest in creating targeted ads.

On the 25th of May 2018, the EUs General Data Protection Regulation (GDPR)(3) came into effect. GDPR has made it mandatory for websites to ask for consent to use the user's data for personalizing content and ads. The problem here is often if you do not agree, you are unable to access the content on the website. In other countries the user is not given the option and consent is implied by the user using the service. This makes ad blocking necessary to protect the privacy of users who want to use such websites.

Observations were made of the turmoil between advertisement suppliers and advertisement blockers, as a result research to find other solutions to the Ad blocking problem commenced. Pi-hole, a DNS sinkhole for advertisements appeared as a promising solution. This program is simple to install but requires an always on server. The problem with this solution is it required prior knowledge of computing whereas browser plugins are a 1-click install meaning they are far simpler for the greater public to use.

1.2. Project Description

To solve the problems mentioned above, a solution was proposed where the system uses a convolutional neural network model to identify images in a website. The output will then be compared against what the text on the page is about. To compare the text and image classification output, word2vec a word embedding 2-layer neural network will be used decide what the text is about. If there is a miss match, we can assume that the suspect image is an advertisement. As time goes on advertisement providers are getting craftier or have the ability to remove the impact of the ad blocking software, as a result more advertisements are slipping through the cracks. This system would get around these issues by not having static ad lists that must be maintained. To implement this system a client in the form of a chrome extension and a server that will host the machine learning models will be developed.

1.3. Project Aims and Objectives

The goal of this project is to decide if a website has adverts or not. To achieve this objective the images and text on the page will be classified using various methods. This classification will decide what the inputted text and image is. The outcome of these two machine learning algorithms will then be compared against each other. This will be done by grouping these words based on what they are. For example, a dog is in the canine category were as classes will be in an item category. These are two separate groups, so it is likely the image is an advertisement. In this project, the concept will be refined and tested as a valid method of detection.

To achieve this, several objectives must be completed. Step one is to gather relevant technology's and test their suitability for the systems application.

1.4. Project Scope

This project is about advertisement detection. Advertisement removal is out of scope for this project, the system will inform the user on how likely an image is or is not an advertisement. For the purpose of this system, the scope will be narrowed to the type of advertisements and text the system will deal with. This can be expanded depending on how accurate the system is at predicting the likely hood of the image being an advertisement. Starting with a small number of advertisement types will allow testing and validate if this is a relevant method of checking for advertisements.

As an application the system must be lightweight and fast. The interface should follow the concepts of universal design.

1.5. Thesis Roadmap

One sentence summary of the following chapters This section will provide a summary of each of the chapters covered in this report.

Literature Review

This chapter delves into existing methods of ad detection. Following this, an examination into current/existing technologies and data that will enable the completion of this project. Finally, this chapter will discuss any other research done for this project.

Experiment Design

This chapter explains the design and overall layout of the system. An explanation each layer of the system will be explained. This includes diagrams on how each part functions.

Experiment Development

The development chapter goes through the entire development process of the prototype system. The middle-tier and backend systems that where discussed in the Design and Research chapters are implemented here.

Testing and Evaluation

In this chapter, the discuss the testing methodologies used and define the parameters for success.

Conclusions and Future Work

This chapter reflects on the completed prototype system and research completed. The conclusions drawn from this work will be used to propel the development of this system.

2. Literature Review

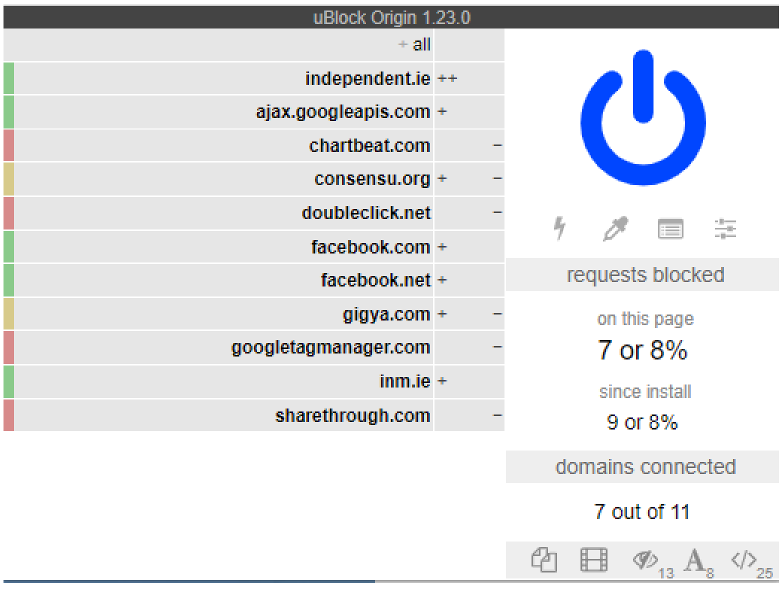
2.1. Introduction

In this chapter, key technologies to the success of the system will be discussed. A Comparison to alternative existing solutions to this problem will be made.

2.2. Alternative Existing Solutions to Your Problem

uBlock Origin

An efficient blocker: easy on memory and CPU footprint, and yet can load and enforce thousands more filters than other popular blockers out there (4). This is a complex interface for an app of this type. This is because the domains of each ad that are blocked are displayed. Control is limited to an on and off button. The app also has html element removing functionality.



uBlock Origin 1.23.0

+ all		
independent.ie	++	
ajax.googleapis.com	+	
chartbeat.com		−
consensu.org	+	−
doubleclick.net		−
facebook.com	+	
facebook.net	+	
gigya.com	+	−
googletagmanager.com		−
inm.ie	+	
sharethrough.com		−

requests blocked

on this page
7 or 8%

since install
9 or 8%

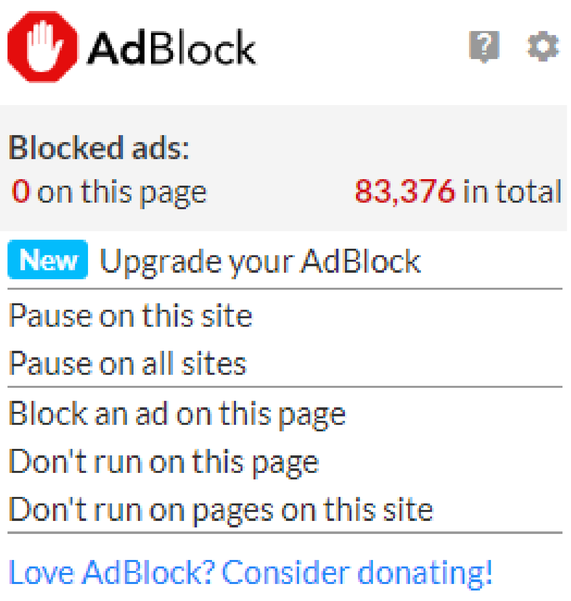
domains connected

7 out of 11

Figure 1 - uBlock origin interface

AdBlock

AdBlock (5) does exactly what it says on the tin. It supports the mainstream web browsers as well as android and iOS. The user interface displays the least information but allows the users to pause block and control running in one simple page.

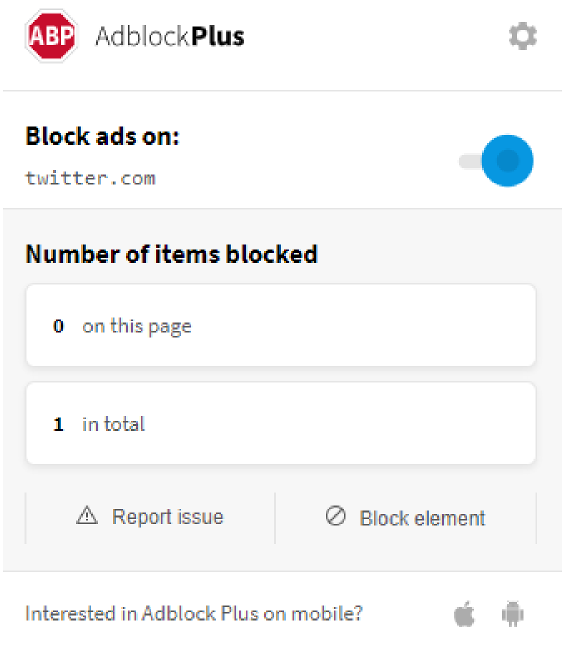

AdBlock

Blocked ads:
0 on this page 83,376 in total

Figure 2 - AdBlock interface

AdBlock Plus

AdBlock plus (6) provides a distraction free browsing experience in the same was as others in this list. AdBlock plus is open-source and gives users access to privacy controls. The interface is very simple, this is a toggle for blocking ads on the current website and a display for advertisements blocked.


AdblockPlus

Block ads on:
twitter.com

Number of items blocked
0 on this page
1 in total

Figure 3 - AdBlock Plus

Pi-hole

Pi-hole (7) is a DNS sinkhole that is used to provide network wide adblocking. This is another system that uses static lists to find and block advertisements. The system is harder to setup then its counterparts as it is hosted on an internal server and client DNS setting must be changed to point to the internal server. The system can also double as a DHCP server.

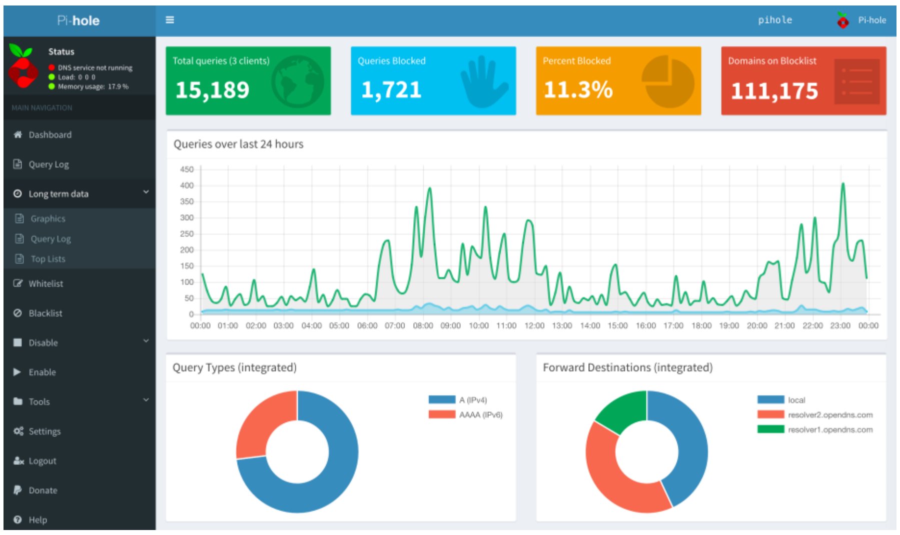


Figure 4 - Pi-hole dashboard

Conclusion

These programs all use a variation of static lists to find adverts. The most common are plugins for common browsers i.e. Firefox, chrome, etc. All these solutions have very simple interfaces that allow the user to see the total number of advertisements blocked on the site. They also allow the user to enable and disable on the fly. All the plugin mentioned have 10,000,000+ users on google chrome making this a very sought-after functionality.

2.3. Technologies you've researched

The following are possible platforms:

Chrome

Chrome is a popular web browser (8). Chrome currently has about 60 percent of the browser market according to statcounter and w3counter (9). These statistics are collected from all websites that use W3Counters or statcounter, these websites collect telemetry using JavaScript allowing insights into the website's performance. Google are a major supplier of advertisements, so have a vested interest in crippling the current systems.

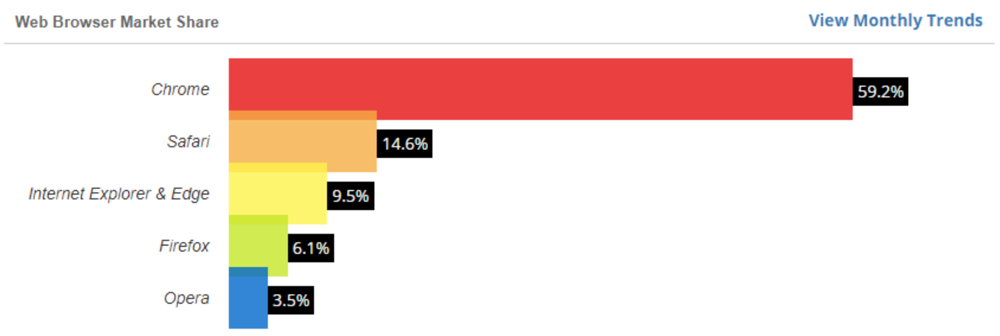


Figure 5 - W3counter report

Firefox

Firefox (10) does not have anywhere near the popularity of chrome. Firefox has a reputation for user privacy and in 2019 have rolled out features such as a VPN and have removed plugins that collect excessive user data. By default, 3rd party tracking cookies and crypto miners are blocked. Firefox is backed by a non-profit.

Conclusion

In conclusion, chrome is the market leader for web browsers. A method that does not uses static lists will be most useful on this platform as this is where the advertisement skirmish is happening. Firefox actively supports user privacy meaning that if the system is successful it should be ported to this platform in the future.

The following are possible programming languages:

Java script

This is the go-to language for extension development along with html and CSS. Java script can be used to modify the DOM on websites this will be used to highlight or completely remove the ads. The TensorFlow.js library allows machine learning models to be developed and run inside of a browser. This would allow the backend systems to be implemented in the browser making the application a 1 tier architecture.

Python

Python is a high-level interpreted language that emphasised code reliability. Python supports many open source frameworks, library's and development tools. Python also allows for rapid prototyping due to the small size of the code. Applications like conda or virtualenv allow for python programs to be isolated from other projects and help manage their dependencies.

C++

C++ is a general-purpose compiled language. C++ is used mainly in systems programming and embed systems because of its features and security. C++ has support for library's and frameworks.

Conclusion

In conclusion, python's ability to produce rapid prototypes and mature library's make it a good language for this system. Python is often used in machine learning and web development tasks. Implementing this solution completely in JavaScript is to much of a risk as the library is not mature and processing limits associated with running code in a browser. JavaScript is the recommended and supported language for developing extensions for the chrome browser because of this it will be used for this purpose.

The following are possible library's and models:

Word2vec

This is a method of converting words to a machine-readable format. Normally to deal with categorical features like words one must embed them into a numerical format but in doing so it loses how the words relate to each other. Word2vec preserves the relationship between the words. Word2vec comes in two flavours, the skip-gram method of word2vec weighs nearby context words more heavily than more distant context words and Bag of words which the order of context words does not influence prediction. (11)

Inception

Inception is a convolutional network that classifies images. Unlike its competitors it goes wider instead of deeper by having multiple convolutions on the same level. Using a pre trained model as opposed to building a new model makes sense when trying to solve this problem as these models are optimized and trained well past anything one person can manage on their own. (12)

WordNet

WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. (13) The linking of the items in the wordnet database and our ability to see what Synset the word is part of will allow the system to make meaningful comparisons between the output of the convolutional neural network and text analysis systems

TensorFlow Lite

This allows TensorFlow models on mobile, embedded, and IoT devices (14). Lite does this by using pre-trained models. These models are designed to execute efficiently on mobile and other embedded devices with limited compute and memory resources. TensorFlow Lite has a JavaScript implementation this should allow the use of the image classification model in the front end of the extension/website. To utilise this library the TensorFlow model must be recompiled for TensorFlow lite. Compiling this model proved problematic, as such this feature was scrapped in favour of implementing the image classification model as a back-end system. In future iterations of this system this feature should be implemented allowing a user to choose between client and server implementations.

The following are possible web frameworks:

The requirements for this framework are simplicity but expandability. This system will serve as the connection between the backed and the frontend of the system. Because of limited resources the framework should be easy to operate and easy to run.

Flask and flask restful

Flask (15) is a micro web framework for python that provide the means to create a web application. It is a micro framework as it does not require any tools or library's to function. It is free and open source. Flask restful builds on this functionality allowing developers to create REST APIs. A rest API may be necessary for this system depending on what portions of this project can be ran in browser. Flask is a micro framework meaning that functionality like logging in and Restful APIs are added in separate libraries or must be implemented by the developer. Flask is not asynchronous, to get this functionality a production web server like apache must be used. If this option is utilised this may become a problem as running machine learning models can be resource intensive and using multiple processes may be a way to speed up the execution of the system.

Django

Django is a web framework for python that includes all the functionality needed to scale and secure a website. Django like flask is free and open source but is more feature rich out of the box. Django touts itself as being a "batteries included" application this means that it as everything needed to create web applications out of the box like URL routings, template engine, Session handling and AJAX support to name a few of the available functions included in the Django framework.

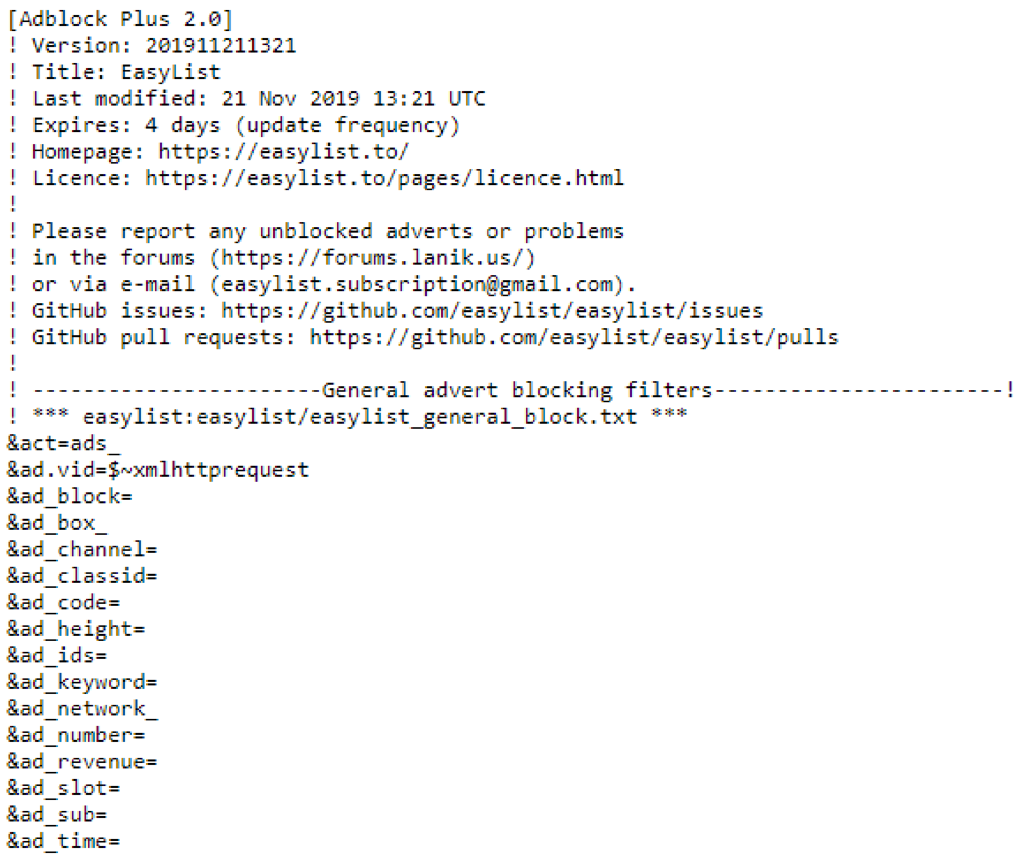
Conclusion

Flask is a far simpler and as a result easier to use as it sets out to be a mini web framework and nothing more. Whereas Django aims to be an all-inclusive platform. As of now I only require a method of communicating between the middle layer and backend because of this the lighter simpler platform will be better for this project.

2.4. Other Research you've done

Ad lists

These are lists made and maintained by the AdBlock community. A popular Ad list, EasyList (16) comes pre-packed and/or tested with many adblockers (AdBlock Plus, AdBlock and uBlock Origin). EasyList is a primary filter list for removing most adverts from international webpages, including unwanted frames, images and objects. It is the basis for many other filter lists. Below is the easy list utilised by AdBlock plus which is popular browser extension.



```
[Adblock Plus 2.0]
! Version: 201911211321
! Title: EasyList
! Last modified: 21 Nov 2019 13:21 UTC
! Expires: 4 days (update frequency)
! Homepage: https://easylist.to/
! Licence: https://easylist.to/pages/licence.html
!
! Please report any unblocked adverts or problems
! in the forums (https://forums.lanik.us/)
! or via e-mail (easylist.subscription@gmail.com).
! GitHub issues: https://github.com/easylist/easylist/issues
! GitHub pull requests: https://github.com/easylist/easylist/pulls
!
! -----------------------General advert blocking filters-----------------------!
! *** easylist:easylist/easylist_general_block.txt ***
&act=ads_
&ad.vid=$~xmlhttprequest
&ad_block=
&ad_box_
&ad_channel=
&ad_classid=
&ad_code=
&ad_height=
&ad_ids=
&ad_keyword=
&ad_network_
&ad_number=
&ad_revenue=
&ad_slot=
&ad_sub=
&ad_time=
```

Figure 6 - easy list

Draw.io - Diagrams.net

Draw.io is an online diagram editor. The application is browser-based and free to use. This application will be utilised for medium fidelity prototypes of the UI of the system. Another application is diagrams for the design and ideas of some systems. As such draw.io will be utilised in the explanation of this system.

2.5. Existing Final Year Projects

Here are some final year projects from previous years that are relevant to this project. These projects are both machine learning projects and as such share some similarities to this project in their development style, technologies and methods used.

Title: Euro Coin Classification. Using Image Processing & Machine Learning

Student: Yumin Chen

Description (brief):

This project aims to investigate the visual features of euro coins and develop models that can represent different natural images. The objective of this project is to investigate the use of machine learning specifically in the context of euro coin classification.

What is complex in this project

Classifying the coin is the difficult part of the project. This is done using both image processing and machine learning. The student implements different algorithms and evaluates them.

What technical architecture was used

OpenCV, Cordova, machine learning and data mining techniques.

Explain key strengths and weaknesses of this project, as you see it.

The student tested a multitude of ml algorithms and evaluated them against each other. I see this as a strength. The student uses Rapid Application Development methodology, prototypes were made quickly but the problem of immediately starting to code is an issue. This weakness in the development style, extra care should be taken in the planning stage for this sort of project.

Title: Detecting Bot Twitter Accounts using Machine Learning

Student: Emmet Hanratty

Description (brief):

In this project the student creates a system to detect and evaluate whether a user account is a bot. The system manages good accuracy, and this will be increased as the data set expands. They system also creates graphs on the data used in prediction.

What is complex in this project

Training a sentiment model based on tweet data. This decides if a tweet is positive, natural or negative.

What technical architecture was used

Python using the Django framework, MySQL, Twitter API and Git

Explain key strengths and weaknesses of this project, as you see it.

The project is not multi-threaded meaning it is slow in the pre-processing stages. Twitter have a limit on the amount of calls you can make to their API, meaning after a certain number of requests the system is unusable for the rest of the day. The system has good accuracy when predicting the state of an account and this accuracy will grow over time.

As stated int the introduction of this section, both projects are machine learning orientated and as such the methods used in completing these projects will be useful in completing this one. The RAD method of software development is a good method for this type of project as it allows refinement of the prototype to take place before it is implemented in the main system. Extra care should be taken in the design chapter to keep this project on track. In the second project discussed the system developed was slow in the pre-processing stage. Finding a method to speed up the classification on images and text like using GPU resources as speed could be a deciding factor in the validity of this project.

2.6. Conclusions

From looking at the current popular solutions for advertisement blocking we can see they all employ a similar approach for this task. With certain companies having a vested interest in curtailing this technology for their own profit and said companies having the means of doing so other solutions should be developed. To develop a solution to this problem possible technologies were researched, and existing Final year projects were looked at to provide guidance.

3. Experiment Design

3.1 Introduction

In this chapter, how the system will interact, look and function will be designed. The main contenders for the software methodologies are discussed. How the system will look for the user is explored. The overview of the backend systems that will be utilised and how these systems will be interacted with the front-end will be decided upon.

3.2. Software Design

Waterfall Methodology

The waterfall model (17) is a phase by phase method of developing software. The current phase must be completed before commencing the next phase. Once a phase is finished it cannot be revisited till the end of the project.

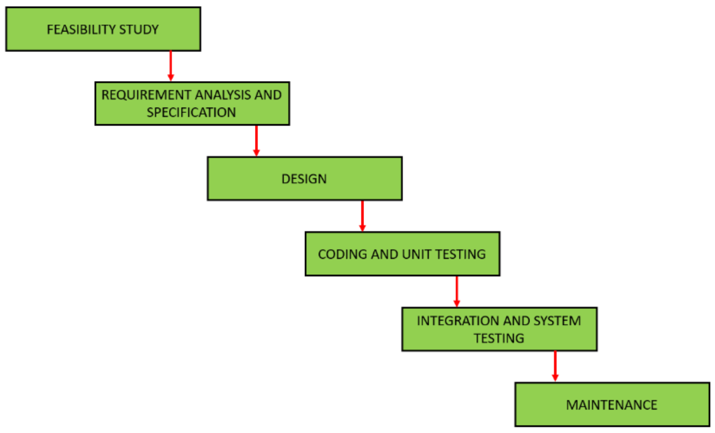

FEASIBILITY STUDY
REQUIREMENT ANALYSIS AND SPECIFICATION
DESIGN
CODING AND UNIT TESTING
INTEGRATION AND SYSTEM TESTING
MAINTENANCE

Figure 7 - waterfall model

An advantage of this model includes clear and understandable milestones. This makes the stages of the process and advancements very clear.

A disadvantage of the waterfall model is that is difficult to redevelop and redesign when the goalpost changes. Once started you are locked into this mode of development till the maintenance phase. On completion of the project, the application may or may not be suitable for its purpose.

Rapid application development

Rapid application development (18) is a form of agile software development. This methodology will allow rapid testing and creation of new approaches.

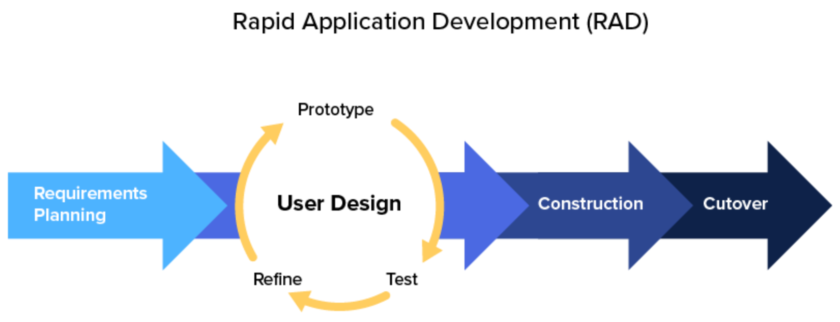


Figure 8 - Rapid Application Development

The main advantage of the RAD methodology is that time between prototypes and their next iteration is short. This allows for rapid development over a shorter period.

The disadvantages of RAD are mainly based around applying the methodology to larger teams. As this is a solo project this is not a problem. Another disadvantage of this methodology is it is not suitable for larger projects.

Both these methodology's have their advantages and disadvantages. Because of this different methodology's suite types different of software development. This project is developed by one person and must take place over a limited time. For these reasons rapid application development was chosen.

Using the Rapid application development method of software design the first step is define what the system aims to accomplish, this has been completed project aims and objectives portion of this report. The second stage of RAD is to prototype designs and develop these prototypes. The general approach to the prototype phase was to divide the system into manageable pieces. These pieces were then built, refined and tested. After Unit testing which will serve as the feedback, these pieces will be grouped together to create the final system.

3.3. Software Test plan

No.	*Feature*	*Testing method*	*Description*
1	Word2vec	Ad hoc	Visual assessment of the same data multiple times.
2	Image classifier	Ad hoc	Manual query of know image types.
3	Flask API	Ad hoc	Curl and completed chrome extension.
4	Compare script	Unit testing	Utilise python's unit test framework.
5	Chrome extension	Ad hoc	Visit websites and test if application gets correct data.

3.4. Front-End

The system will have a similar front end to its alternative solutions. This will be written in html, CSS and JavaScript as these are the default languages used to write extensions for chrome. Across these solutions, the number of ads blocked, a way to stop and restart the application are universal. Below are possible designs for the interface of the application.

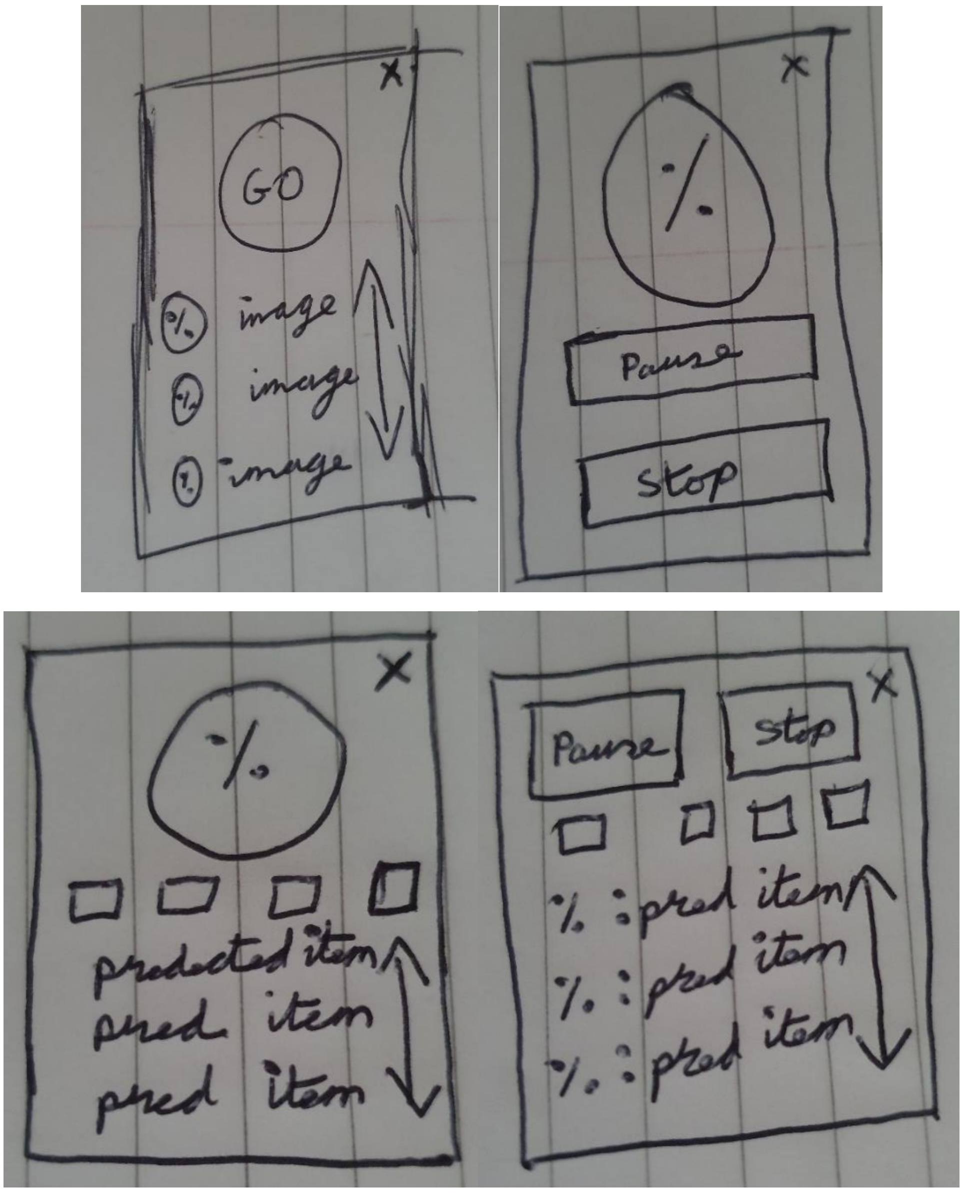


Figure 9 - low fidelity prototype

The next step was to create medium fidelity prototypes. The front end will we simple with one to two different pages. The prototype displays the functionality of the interface and its overall layout. A simple easy to understand interface will server this system the best as it will need to be accessible to all types of users.

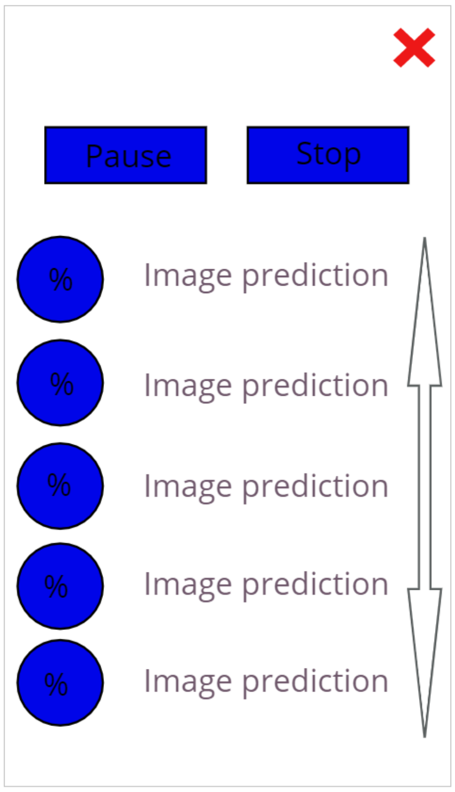


Figure 10 - medium fidelity prototype

Taking the design provided by the medium fidelity diagram this can be converted into html and CSS. The high-fidelity prototype adds an information paragraph to the interface. A subtle purple boarder when the user hovers the buttons was chosen to replace the blue accents of the medium fidelity prototype. This high-fidelity design will then be utilised int the client chrome extension.

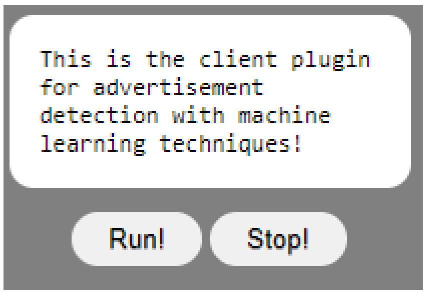

This is the client plugin for advertisement detection with machine learning techniques!

Figure 11 - high-fidelity prototype

The front end is also responsible for the parsing and gathering of the data. This is then posted to the Flask API. The images must be scraped from the website, the collected images will be posted to the Flask API. The Flask API will return the classification of the images. The same scraping will occur for the text however if the text does not contain punctuation marks or is under seventeen characters. This will ensure the titles and menu text is filtered out. The average amount of letters in a word is 4.79 (21), this is based on data gathered from the Google Books Ngram Viewer. Most of the title text that should be filtered out seems to lack punctuation and is below 3 words. The filtered text can then be posted to the text classification system.

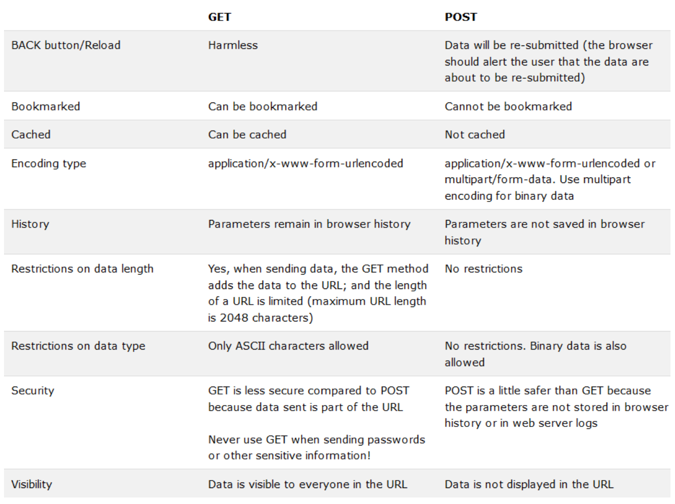
Taking the outputs of these two systems they will be compared using the word net database and a distance formula. The output of the compare will be displayed to the user.

3.5. Middle-Tier

This is the flask API responsible for interacting with the backend systems and the front end. An API is an application program interface which is responsible controlling how systems interact. The API created in the course of this project can be used for other systems that require a word2vec and inception image classification. Allowing other projects using these systems as building blocks to be developed faster.

The API will utilise both get and post requests. The get requests will be responsible for explaining to the user what and how the API should be used. Posting to the API on different routes will allow the chrome extension to interact with the backend systems. This separation will allow lower spec computers to take advantage of these algorithms. Separate clients can be developed for this system allowing for multiple types of implementations.

The method in which the systems communicate is request and response. The client makes a response which will contain the method, resources, headers and content. The server will then do computation and respond with the status code, headers and content. The get method is responsible for requesting data from the server. These requests can be cached and have length restrictions, so they are most suited for supplying information about the API. The post method on the other hand is useful for creating and modifying data. These requests cannot be cached and encapsulate data in the body of the request meaning it is more secure and there are no restrictions on the data length. Images and large portions of text must be sent to this API meaning post requests are necessary. Below is an in-depth comparison from w3schools (20).



	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Figure 12 - get vs post

3.6. Back-End

The backed will consist of 3 systems, image classification, word embeddings.

Image classifier

The image classification system will be based on the inception model. This model is already trained on ImageNet making its predictions already quite accurate. The plan to expand on this model is to retrain it with logos of common companies making it more suitable for these purposes. This backend system may also be moved to the front-end in this iteration depending on how the development progresses. Below is a possible structure for the image prediction.

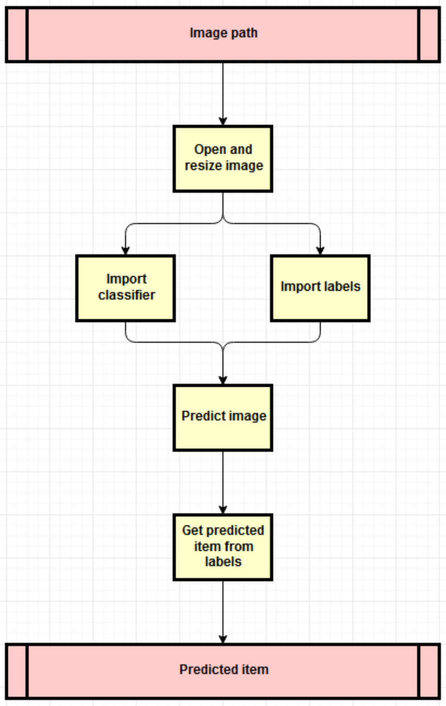


Figure 13 - image classification design

Word2vec

The word2vec (19) portion of the back end will run on a server as it is to intensive to run on the clients. The reason why word2vec is utilised here is because normally when converting word into vectors the relationship between the words are lost. This can be overcome with word2vec. Take the sentence “the red fox jumped over the red car” this can be converted into the vectors in figure 11 but in doing so the relationship between the words are lost. But in figure 12, word2vec is used and as you can see the relationships between the words are preserved. From this we can infer that the words over, jumped and red have a greater relevance to the text and from the graph we can see that these words are related to each other more closely. In the development of this experiment stop words like “The” should be dropped. Stop words don’t add context or value to the outputted vectors and regarding this project a picture cannot be a stop word.

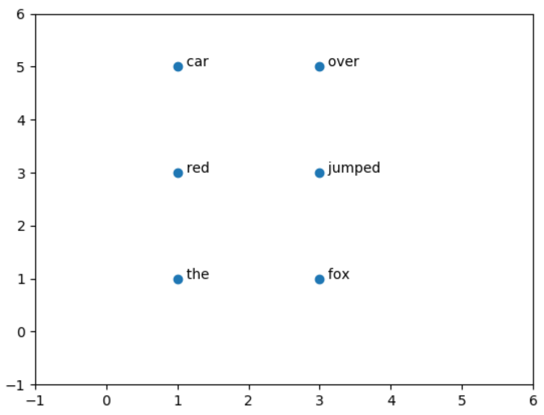


Figure 14 - vectored words

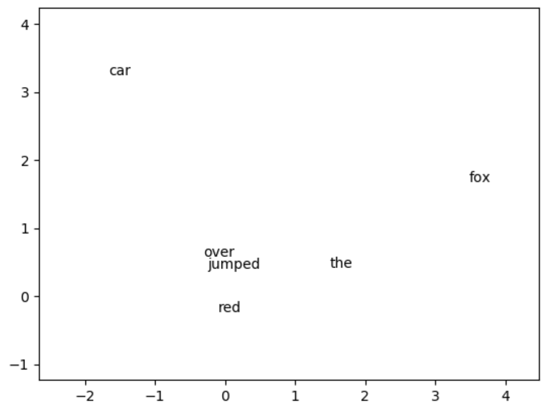


Figure 15 - word2vec vectored words

Each new document will have to be trained separately as each document will produce a different set of vectors. The vectors that are generated each time will be different, but the same basic relationships will be preserved. Word2vec outputs word embeddings by predicting the neighbouring words given a word. Word2vec is a 3-layer neural network which consists of an input layer, a hidden layer and an output layer.

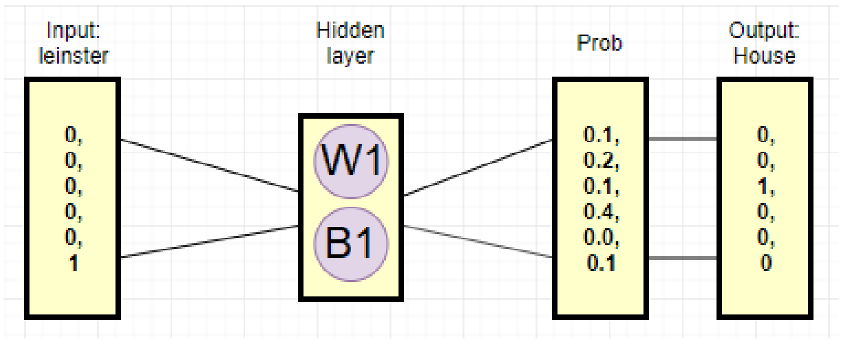


Figure 16 - Word2vec structure

Training the model to predict the next word will never give high accuracy at the task. But the hidden layer which is the vectors used to predict the outcome give us the relationship of the words to each other.

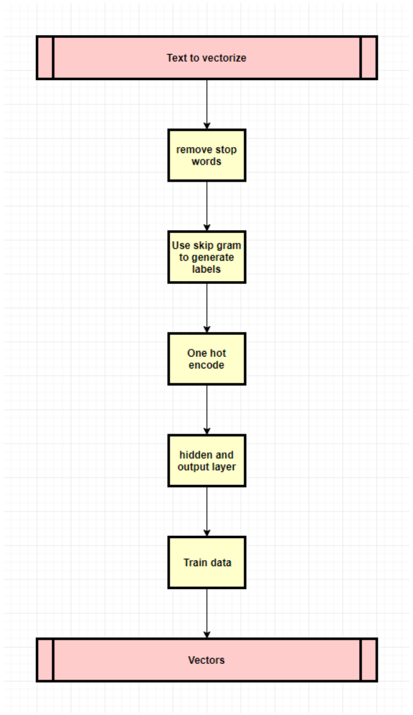


Figure 17 - word2vec design

Compare

This portion of the backend will be responsible for comparing the outputs of the image classification and word2vec. This logic will come in two steps, first the system will decide if the image could be an advertisement or not. This will be a Boolean value based of if the word is in the text. As words can be different but mean the same thing, querying the word net database is necessary. This will allow the system to match a greater variety of words. To do this synsets and lemmas are utilized. Synsets are sets of synonyms and lemmas are the canonical form of the word. The primary method of discerning if words are part of the same category will be the synsets but if this fails lemmas will be used to substitute the data. Below is an example of a synset relationship.

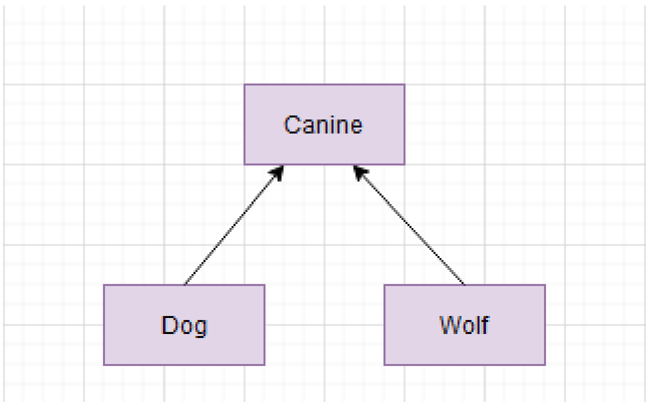


Figure 18 - synset example

In this example both wolf and dog share the synset or category of canine. The system gets the values of 'canine', 'unpleasant_woman', 'chap', 'villain', 'sausage', 'catch', 'support' and 'pursue' for dog. And the values of 'canine', 'womanizer', 'attacker' and 'eat' for wolf. Both lists have the values of canine so therefore the image and text wold have the same classification making the image being an advertisement unlikely. But as stated before this is a Boolean yes or no, dog could have been mentioned in a different context like hot dog. As such this compassion only serves as a starting point to decide how likely this is to be an advertisement.

The second stage of the compare logic is to decide how likely the possible advertisement is to be an actual advertisement. To do this the output of the word2vec algorithm is used. This is a two-dimensional graph of how each word in the text relates to each other. Using the Euclidean distance, we can compare how connected a word is to the document. Therefore, a more connected word will have a smaller total value for the lengths of its neighbouring points. This is a necessary part of this system as the text of a web site may mention something that does not pertain to what the said text is about. For example, a text may mention that they have a new line of clothing and you should buy them. This text will not be as interconnected in the word2vec graph, giving it a higher value meaning it is less likely to be what the subject is about.

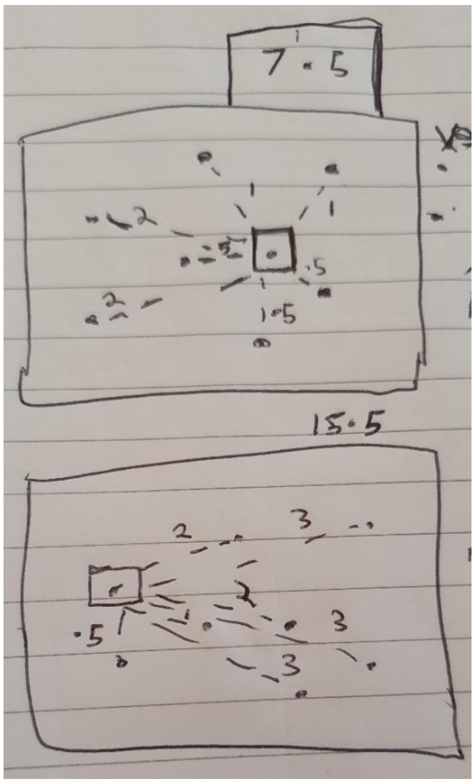


Figure 19 - Euclidean distance total

In the above diagram this idea is shown, the middle point is more connected meaning it will get a lower score whereas the second diagram the point is on the edge meaning the other points are further away making the value higher.

There are a few ways to calculate the distance between two points such as the Euclidean distance, Manhattan distance and Chebyshev Distance. Each of these methods have different quirks as they approach the problem differently. Because the Euclidean distance is the easiest to intuitively understand it was chosen as the method for calculating this metric, but this requires more mathematical operations so in future versions it may be quicker to use the Manhattan distance as this method requires less operations. The Euclidean distance can be equated to traveling along the hypotenuse of a right-angled triangle whereas the Manhattan distance can be equated to traveling along the other two sides.

3.7. Conclusions

In this chapter, the three main back end technologies were discussed, and an overall design was given for each of them. The middle tier will be responsible for connecting the backend to the frontend and translating the gathered information into meaningful data. The design of the UI was explored, fleshed out and settled on. Because of the nature of rapid application development these designs may change as the overall system is iterated upon in the development chapter.

4. Experiment Development

4.1. Introduction

From my research and design, I have decided on a few technologies to use for this system. These technologies may change depending on how this experiment develops. Upon entering a new site, the DOM will be scraped for the relevant data (the images and text of the site). This data is then shipped to the flask API which is responsible for running the necessary backend functions. The back-end function of this system will be the image classifier, text classifier and a compare system.

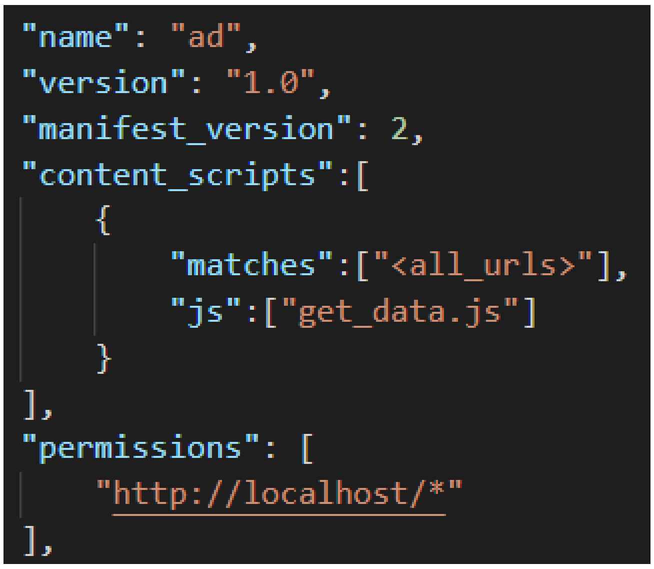
4.2. Software Development

The first step to developing any system is to set up version control. This project is being managed through GitHub at https://github.com/UNIpope/fyp-fix. As the prototype of this system already has a Git repository, we will be extending from that. In order to keep different versions of python libraries from interfering with each other it is necessary to utilise conda or virtualenv. For this project virtualenv was utilized as it was pre-installed, and the developer has experience with this method of package management. Visual studio code has a robust and easy to use implantation of version control along with good python virtual environments implementation as such this is the chosen IDE.

4.3. Front-End

The front-end interface will be implemented in html, CSS and JavaScript. The main html file will be responsible for showing the user information about the system. Upon entering a new website, the chrome extension will run the script responsible for collecting and shipping the data to the middle tier Flask API. The returned information will then be displayed to the user. The chrome extension consists of a manifest file, the popup HTML and CSS and any scripts that should be run.

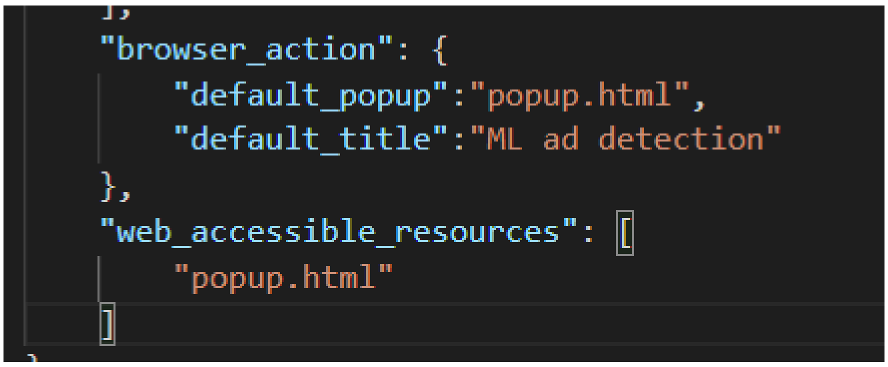
The manifest file is responsible for providing chrome with important information. The required information by chrome is the manifest version, name and version. The name is simply the name of the extension, the version is the version of the chrome extension and the manifest version is the version of the manifest file which as of Chrome 18 should be set to the integer 2. Content scripts allow the specified JavaScript to be run on the specified websites. In the case of this system, the content script is run on all URLs. The specified script is responsible for scraping the data from the website, posting the data to the Flask API and displaying the result to the user. The permissions variable allows the client to communicate with the localhost, this will be changed to the domain of the API server in future iterations.



```
"name": "ad",
"version": "1.0",
"manifest_version": 2,
"content_scripts":[
    {
        "matches":["<all_urls>"],
        "js":["get_data.js"]
    }
],
"permissions": [
    "http://localhost/*"
],
```

Figure 20 - manifest variables

Browser action is responsible for defining what popup and title should be displayed. This links to the popup.html file which displays information to the user. The popup html is made available to edit by the variable web accessible resource. This will allow the popup.html to be updated with relevant advertisement information from the content scripts.



```
    "browser_action": {
        "default_popup":"popup.html",
        "default_title":"ML ad detection"
    },
    "web_accessible_resources": [
        "popup.html"
    ]
```

Figure 21 - manifest popup

When clicking on the icon for this extension it will display the popup html. The popup html is created like any other html website accept it is displayed as a popup. This popup website is responsible for explaining to the user what the plugin does. This plugin is also responsible for starting and stopping the plugin processes. The popup also displays a list that will be updated when the plugin detects advertisements on the website which is only shown when it is updated.

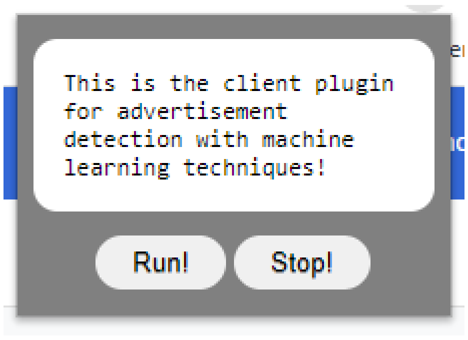
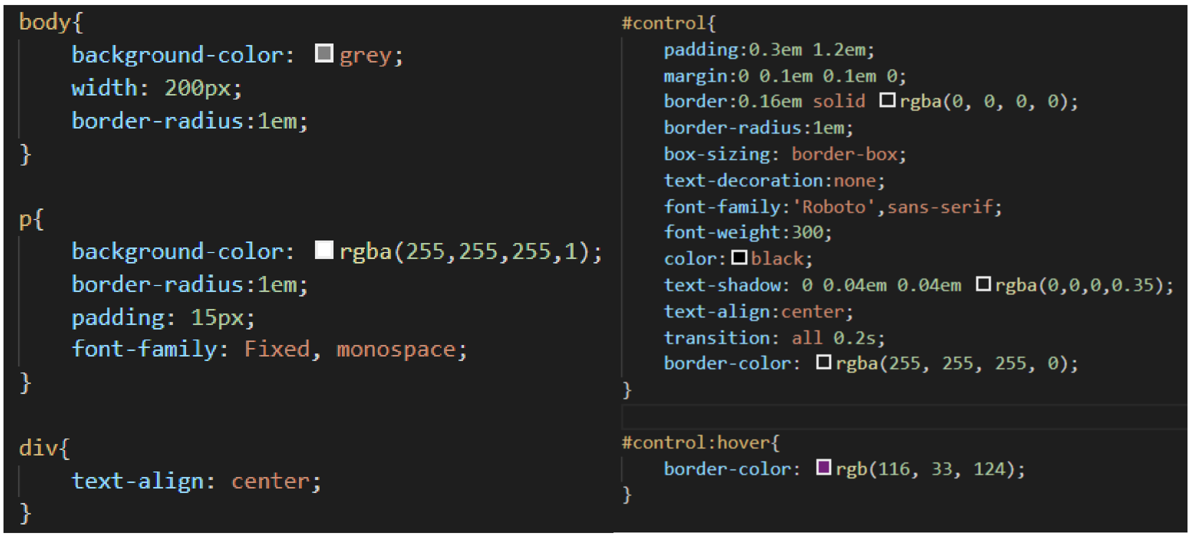

This is the client plugin for advertisement detection with machine learning techniques!

Figure 22 - popup

The html for this is simple, it contains a paragraph element, two button elements and a list element. These elements are linked to an external style sheet. This provides the styling of the extension. A lot of effort was put into getting the look of the Buttons correct as this is the main reason to be in this popup menu.

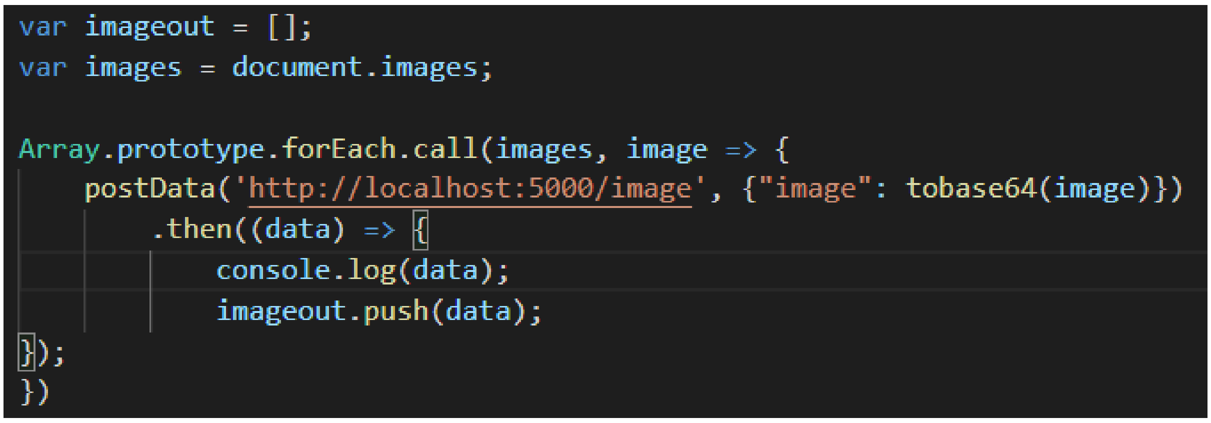


```css
body{
    background-color: grey;
    width: 200px;
    border-radius:1em;
}

p{
    background-color: rgba(255,255,255,1);
    border-radius:1em;
    padding: 15px;
    font-family: Fixed, monospace;
}

div{
    text-align: center;
}
```

```css
#control{
    padding:0.3em 1.2em;
    margin:0 0.1em 0.1em 0;
    border:0.16em solid rgba(0, 0, 0, 0);
    border-radius:1em;
    box-sizing: border-box;
    text-decoration:none;
    font-family:'Roboto',sans-serif;
    font-weight:300;
    color:black;
    text-shadow: 0 0.04em 0.04em rgba(0,0,0,0.35);
    text-align:center;
    transition: all 0.2s;
    border-color: rgba(255, 255, 255, 0);
}

#control:hover{
    border-color: rgb(116, 33, 124);
}
```

Figure 23 - popup CSS

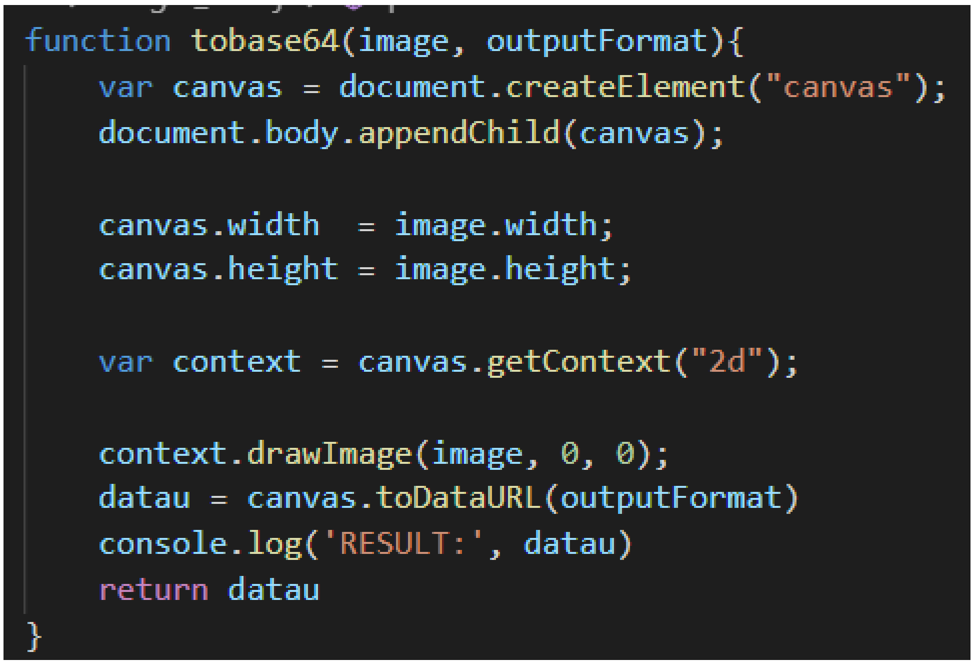
Upon loading a new webpage, the get_data.js script is called. This is the main script of the front-end system. The first step of this script is to collect all the images on the webpage. This is done using document.images, which returns a an array of the images loaded on the website. The foreach() method is used to call the post function on each of the elements in the array. The script waits for the post method to return the image classification. The returned data is logged and added to the image output array.



```
var imageout = [];
var images = document.images;

Array.prototype.forEach.call(images, image => {
    postData('http://localhost:5000/image', {"image": tobase64(image)})
        .then((data) => {
            console.log(data);
            imageout.push(data);
});
})
```

Figure 24 - collect and post images

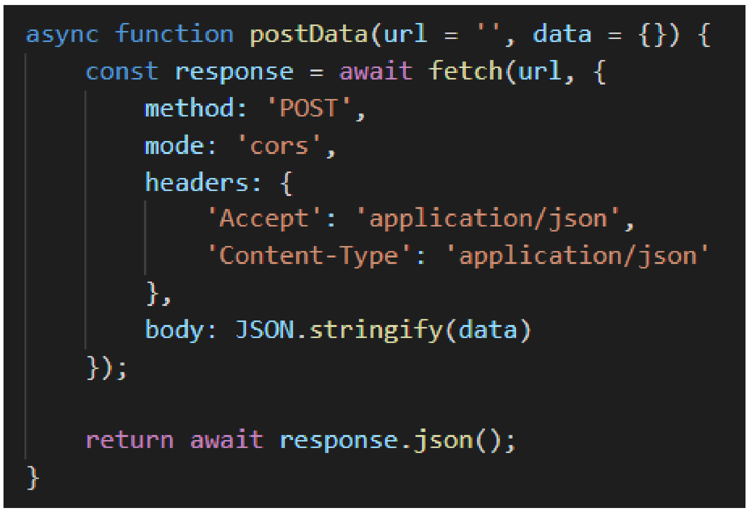
In order to send images via a post request they must be encoded as a base64 string. In order to complete this task a new canvas object must be created; this is appended to the html document. The canvas is setup with a 2d context and the height and width of the image. The image can then be drawn onto the canvas. The resulting canvas can then be converted to URL which is a base64 encoded string. This string is then returned.



```
function tobase64(image, outputFormat){
    var canvas = document.createElement("canvas");
    document.body.appendChild(canvas);

    canvas.width  = image.width;
    canvas.height = image.height;

    var context = canvas.getContext("2d");

    context.drawImage(image, 0, 0);
    datau = canvas.toDataURL(outputFormat)
    console.log('RESULT:', datau)
    return datau
}
```

Figure 25 - base64 encode image

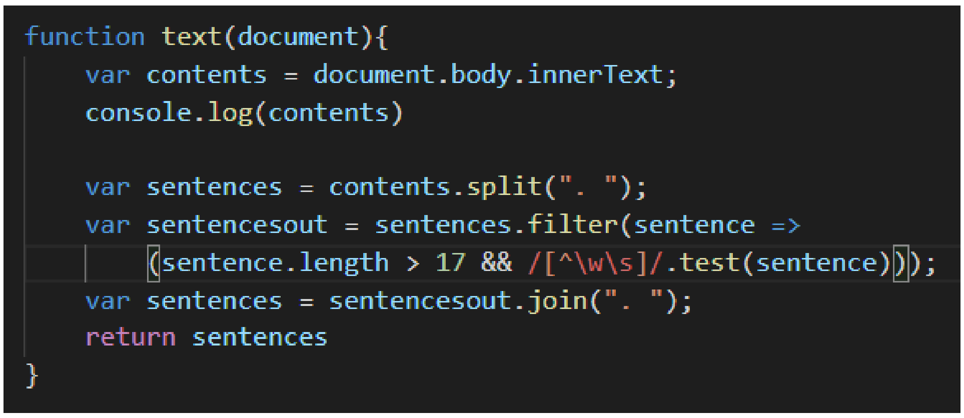
Now that the system has the encode string for the image it can be posted to the Flask API. To complete this task an async function is used. This returns a promise objects that allows for asynchronous behaviour. The script will return the output of this function when the flask server responds. To communicate with the Flask API in this manor a fetch request is utilised. The headers necessary for json are “accept” and “content-type”, this tells the flask server what type of data to expect. CORS mode must be utilised to allow resources to be requested from another domain. The body of the post request is a json formatted version of the data.



```
async function postData(url = '', data = {}) {
    const response = await fetch(url, {
        method: 'POST',
        mode: 'cors',
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
    });

    return await response.json();
}
```

Figure 26 - post function

The next step is to get the text data from the website. The system does this using document.body.innerText. This collects all the text contained in the body element. This data is then split by "." space. Each sentence is parsed for sentences less than three words and punctuation marks. Joining the strings back together will give a cleaned version of the data. This cleaned data is returned and is ready to be posted to the Flask API. Posting is handled in the same way as images, but it goes to the "/w2v" route as opposed to the "/image" route.



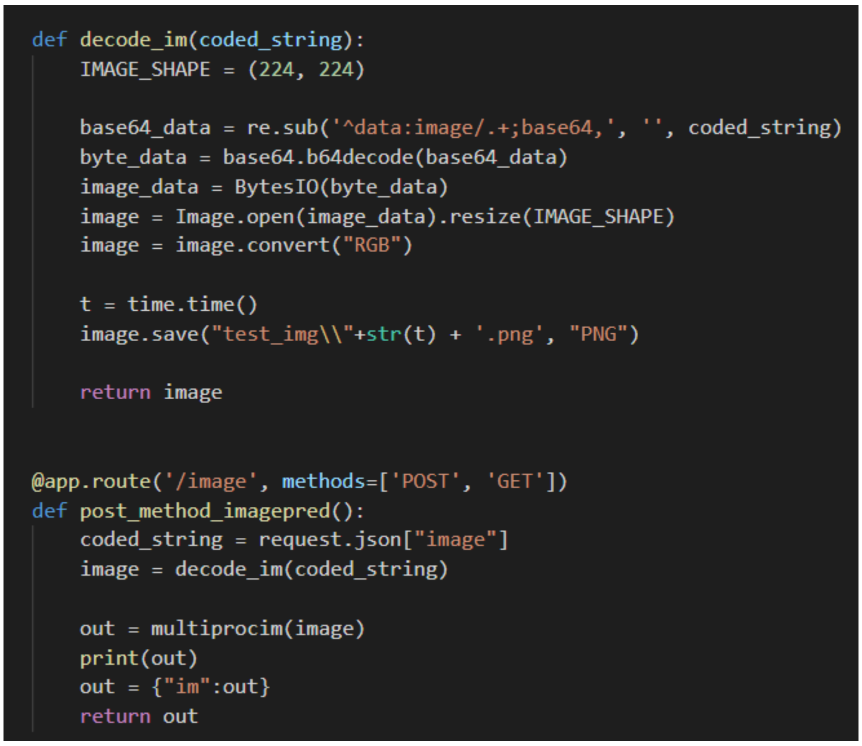
```
function text(document){
    var contents = document.body.innerText;
    console.log(contents)

    var sentences = contents.split(". ");
    var sentencesout = sentences.filter(sentence =>
        (sentence.length > 17 && /[^\w\s]/.test(sentence)));
    var sentences = sentencesout.join(". ");
    return sentences
}
```

Figure 27 - get and clean text

4.4. Middle-Tier

The middle tier of this system is a flask API. This system is responsible for accepting the POST and GET requests from the chrome extension and passing this data to the backend scripts that classify the data and decide if said data constitutes an advertisement or not. Each of these methods have a different route in the flask API. These routes along with the URL state the type of http traffic that the method will accept.

The POST methods of this system import the process management methods of the classifier scripts. When a POST request is made to the URL, Flask will run the method associated with that route which will call the necessary backend script. This data is then returned to flask in a dictionary format and sent back to the chrome extension.

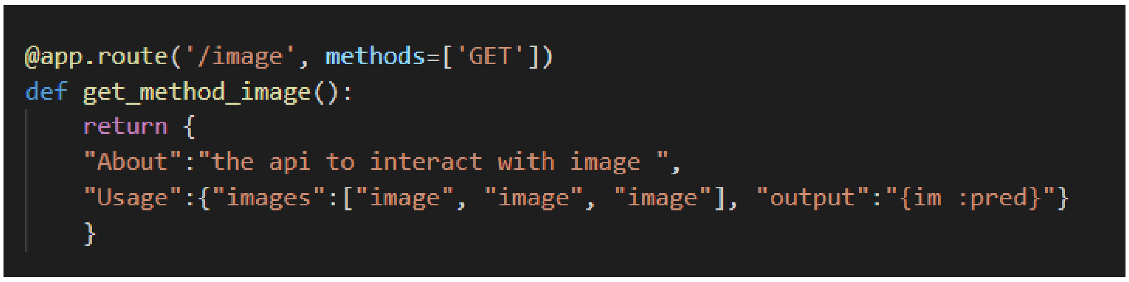


```python
def decode_im(coded_string):
    IMAGE_SHAPE = (224, 224)

    base64_data = re.sub('^data:image/.+;base64,', '', coded_string)
    byte_data = base64.b64decode(base64_data)
    image_data = BytesIO(byte_data)
    image = Image.open(image_data).resize(IMAGE_SHAPE)
    image = image.convert("RGB")

    t = time.time()
    image.save("test_img\\"+str(t) + '.png', "PNG")

    return image


@app.route('/image', methods=['POST', 'GET'])
def post_method_imagepred():
    coded_string = request.json["image"]
    image = decode_im(coded_string)

    out = multiprocim(image)
    print(out)
    out = {"im":out}
    return out
```

Figure 28 - image decode and image route

The above code is responsible for handling the image prediction. To receive images from the client chrome extension, the images are encoded as base 64. To decode these coded strings into an image, the system must strip the identifying string from the coded string. The resulting string is then decoded from base 64. In order to get an image from this the system must convert it into a byte array. As the image declassified takes 3-layer images the image must be converted to the RGB format. The image is also resized to 224 by 224. The image is then saved for testing purposes. The image is then passed to the back-end image classifier and the result of the image classification is send to the client.

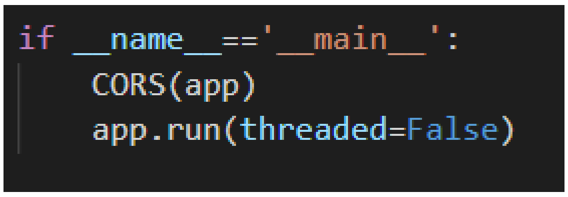
The GET methods are responsible for explaining what the corresponding POST method does and the way in which the data should be structured for that method to be used. The below code is responsible for responding to the get request. When the get request is made the api displays information on what the route is used for, how it should be used and what the expected return type will be.



```
@app.route('/image', methods=['GET'])
def get_method_image():
    return {
    "About":"the api to interact with image ",
    "Usage":{"images":["image", "image", "image"], "output":"{im :pred}"}
    }
```

Figure 29 - get method example

As we are using Flask as a fetch API, CORS (Cross-Origin Resource Sharing) (23) must be used for the chrome extension to communicate with said API. In order to do this, we must import the CORS flask library, set the CORS headers in our flask app to content type and before running the app convert into CORS. By default, flask uses multiple threads to respond to requests. This functionality enables the flask API to respond to requests simultaneously. For this experiment threading must be disabled, this makes flask queue the requests. The backend algorithms utilised use a large amount of ram as such running more than one algorithm at one time will crash the server. Disabling threading is a quick fix for this for the purposes of this experiment. The development server included in flask is not capable of managing threading and multiprocessing in order to get access to those features it is necessary to deploy a production environment, this is out of scope for the current system.



```
if __name__=='__main__':
    CORS(app)
    app.run(threaded=False)
```

Figure 30 - enable CORs and disable threading

4.5. Back-End

Ram usage

The image classifier and word2vec use a large amount of ram. After these systems finishes, they do not release the ram. The research conducted and other options tried to solve this problem are explained in the testing chapter. The solution settled upon is to create a new process for these systems. As the flask development web server is a development server it does not have the functions necessary to manage processes. As such the pythonic way of creating and managing new processes is used. When creating new processes inside the running api the system will hang while it tries to spin up an infinite number of processes. To solve this issue the new processes must be created before the api is started or while the api is executing a function. Trying the former method, the processes can be created and utilised with the intended effect but when closing a process the system lacks the ability to create a new process in its place, leading to the system running out of new processes after n number of calls to the api. As such the latter method of utilising threads was used in the final system. To utilise processes in python the multiprocessing library is utilised. Creating a new process is simple as all that is needed is the process method. This method takes a target function which will be called in the new process and arguments that will be passed to the new process. In order to get data back from the process, the manager class is utilised to get this data back. The manager class allows the creation of dictionary proxies. The dictionary proxy is then utilised to get data back from the process. This allows the contents of the dictionary to be propagated through all the processed with access to the dictionary. As the proxy dictionary is not a dictionary, to return data as a dictionary the system takes the first value of the proxy dictionary which is a python dictionary. The next step is to start the child process and join waits for this child process to terminate. The contents of the dictionary are then returned to the flask api. Because this process is necessary for two of the back-end systems, word2vec and image classification, this explained here. Below is the code used to complete this task in image classification.

The processor used was upgraded to a current model with far more cores. This has sped up both algorithms greatly. Word2vec took around 1 minute and 30 seconds to complete but now takes just under 5 seconds. The above multi-process implementation is still necessary for the back-end systems to clean up after they are finished. The system no longer takes as much ram.

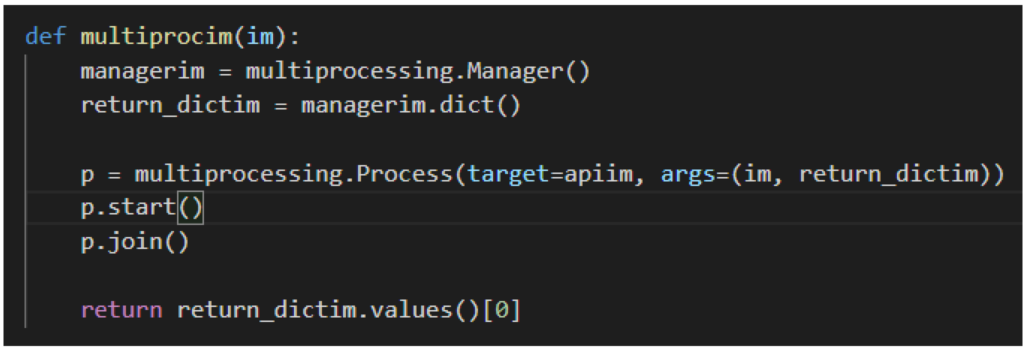

```python
def multiprocim(im):
    managerim = multiprocessing.Manager()
    return_dictim = managerim.dict()

    p = multiprocessing.Process(target=apiim, args=(im, return_dictim))
    p.start()
    p.join()

    return return_dictim.values()[0]
```

Figure 31 – resource use mitigation

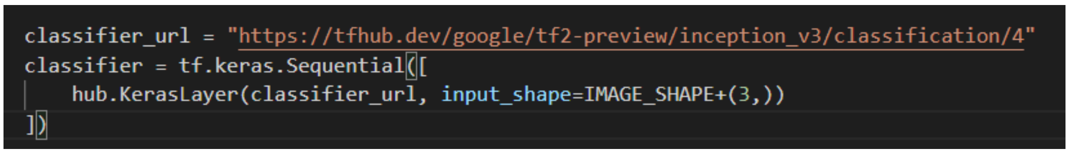
image classifier

In the development of this experimental system this subsystem has been the most problematic. As such, it currently exists as a system that takes in an image and then classifies it with the inception model. The original plan for this back-end system was to have it run on the client side or front end of this experiment. Also, to retrain the inception classifier for advertisement images like brand logos.

These optional requirements failed as saving the checkpoint file for the inception model locally caused problems. Their multiple formats for different use cases. TensorFlow lite uses .tflite files which are produced by taking a saved model and converting it to a TFLite flat buffer file. This file could then be used by a TFLite interpreter to run locally on the client device. Part of the model gets saved but part way through tis process a variable necessary to saving does not exist. This issue may be caused because of versioning of the Keras and TensorFlow library's or the CPU in use for this project is a i5-4670k which is an old chip. Because of time constraints it was decided to utilise inception as a back-end model. This required extra development in the Flask API to allow it to decode base64 encoded images to pillow image objects, allowing a client to send images to the python API.

As stated in the previous chapter, the classification must take place in its own process mitigating the ram issues seen in the earlier development of this iteration. The image object generated from the base64 data is passed to the back-end system and the output of this system will be a dictionary with the label of the image object.

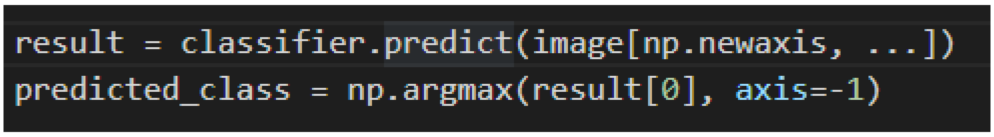
The first step for this new process is to acquire the image classifier. This is done through TensorFlow hub which is a library for reusable machine learning modules. This model is imported as a Keras layer which has the input shape of (224, 224, 3). An image has its resolution which is the x and y lengths but also the RGB colour space. The colour space is represented in three layers stacked on top of each other, the culmination of these three layers makes an image. Different image formats have extra layers as such it is important to make sure the image is in the RGB format this is handled by the Flask API. The layer serves to encapsulate the inception model and this layer is made into a Keras model with tf.keras.Sequential which is a simple model that executes sequentially. The figure below completes this process in python code.



```
classifier_url = "https://tfhub.dev/google/tf2-preview/inception_v3/classification/4"
classifier = tf.keras.Sequential([
    hub.KerasLayer(classifier_url, input_shape=IMAGE_SHAPE+(3,))
])
```

Figure 32 - image classifier

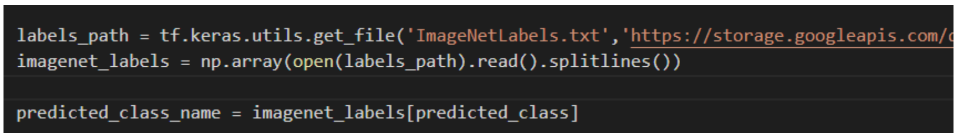
The next step is to predict what the image object could be. To do this the predict function is called of the created classifier. The output of the classifier is the numerical id of the predicted classes with the likelihood of that class being what is predicted. The biggest value in this dictionary will be the predicted class.



```
result = classifier.predict(image[np.newaxis, ...])
predicted_class = np.argmax(result[0], axis=-1)
```

Figure 33 - predict image

The next step is to match the numerical class with the label. This will be the output of this backend system. The image net labels are hosted and can be downloaded from googleapis.com. This file is read into a dictionary. Matching the numerical id from the image classifier with the image net labels outputs the name of the class. This class is returned to the Flask API.



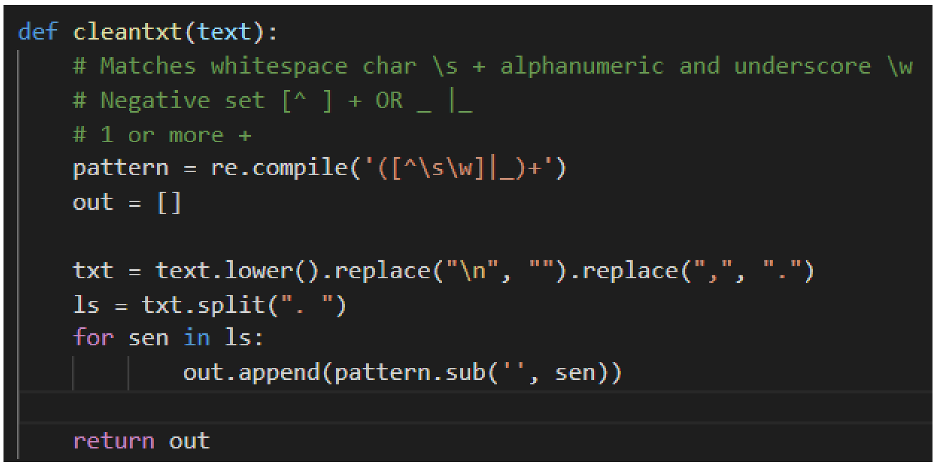
```
labels_path = tf.keras.utils.get_file('ImageNetLabels.txt','https://storage.googleapis.com/
imagenet_labels = np.array(open(labels_path).read().splitlines())

predicted_class_name = imagenet_labels[predicted_class]
```

Figure 34 - match labels

Word2vec

In order to determine the likely hood of a word being an advertisement we must decide on a method to quantify what the text is about. The chosen method for this experiment, as disused in the design chapter, is word2vec as this allows us to change the document into vectors on a graph. This graph shows us each of the words in the text in relation to each of the other words. The skip gram method was chosen as this method predicts the context rather than the word.

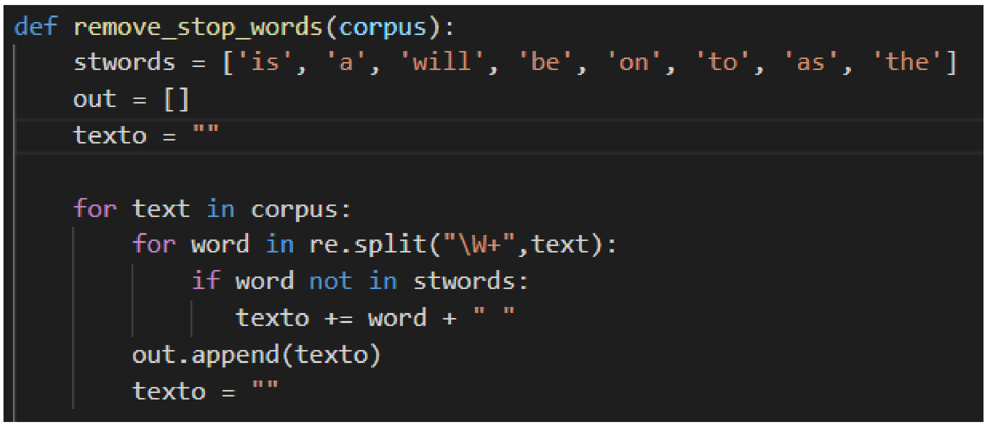
The first steps to implementing this algorithm is cleaning up the data. To do this the system uses a regular expression to find all non-alphanumeric characters and removes them. In order to preserve the structure of the sentences in the text, the text will be split into a list by ". Space" before the regular expression is used to clear up the non-alphanumeric characters. In this portion the trailing new line characters and capital letters are also handled.



```
def cleantxt(text):
    # Matches whitespace char \s + alphanumeric and underscore \w
    # Negative set [^ ] + OR _ |_
    # 1 or more +
    pattern = re.compile('([^\s\w]|_)+')
    out = []

    txt = text.lower().replace("\n", "").replace(",", ".")
    ls = txt.split(". ")
    for sen in ls:
            out.append(pattern.sub('', sen))

    return out
```

Figure 35 - clean text

The next step is to remove the stop words from the text. Stop words are a set of commonly used words in the English language. The reason why these words should be removed is as the removal of these words allows the system to focus on more important words to the text. For the purposes of this experiment the system uses a limited number of stop words though there are many more words that can be added. Removing more stop words may have a negative or positive effect on the overall system as such the system utilises a small number of stop words. Below is the code utilised to accomplish the removal of the stop words.

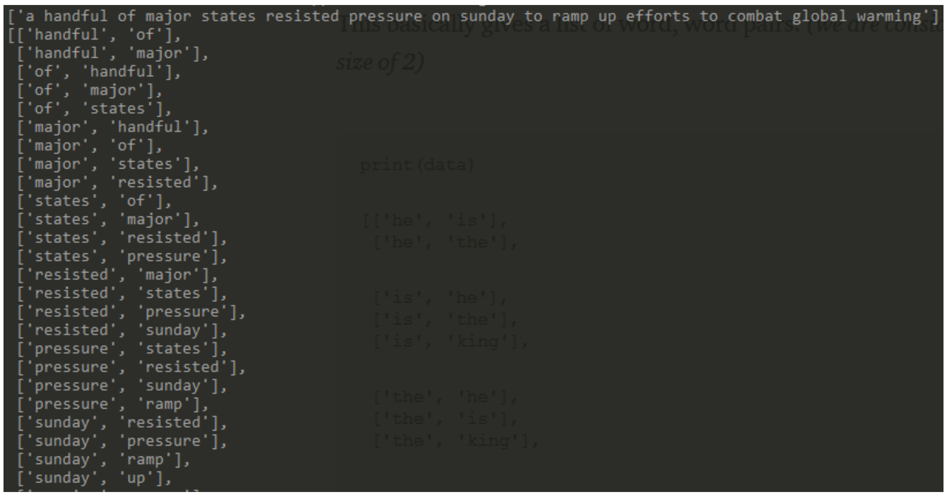


```python
def remove_stop_words(corpus):
    stwords = ['is', 'a', 'will', 'be', 'on', 'to', 'as', 'the']
    out = []
    texto = ""

    for text in corpus:
        for word in re.split("\W+",text):
            if word not in stwords:
                texto += word + " "
        out.append(texto)
        texto = ""
```

Figure 36 - remove stop words

Next the system creates a list of words in the text. To do this the set function is utilized removing the duplicated words. This list in then utilized to create a mapping of a word to an integer. To create this dictionary the enumerate function is utilised. This function returns the location of the item and the item and these items are then inserted into the dictionary.

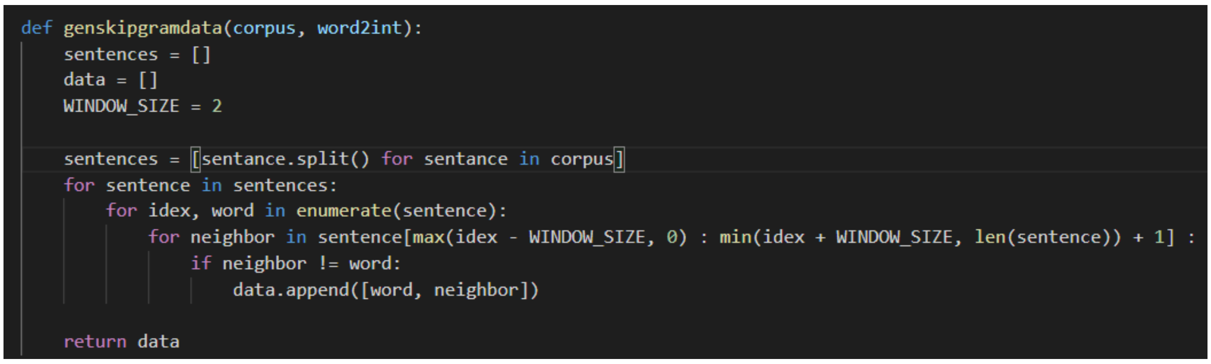
The next function creates the training data used for the word2vec algorithm using the cleaned and stop word less data and the enumerated words. The structure of the training data is a list of word pairs. The word pairs are generated by scanning through a sentence and each word index plus and minus 1 value till the window size is hit. Below is an example of some generated data. As you can see the stop words are removed by earlier processes as such “handful” is the first item in the list. “handful” is related to “of” and “major”. This is repeated for the entire list of sentences.



```
['a handful of major states resisted pressure on sunday to ramp up efforts to combat global warming']
[['handful', 'of'],
 ['handful', 'major'],
 ['of', 'handful'],
 ['of', 'major'],
 ['of', 'states'],
 ['major', 'handful'],
 ['major', 'of'],
 ['major', 'states'],
 ['major', 'resisted'],
 ['states', 'of'],
 ['states', 'major'],
 ['states', 'resisted'],
 ['states', 'pressure'],
 ['resisted', 'major'],
 ['resisted', 'states'],
 ['resisted', 'pressure'],
 ['resisted', 'sunday'],
 ['pressure', 'states'],
 ['pressure', 'resisted'],
 ['pressure', 'sunday'],
 ['pressure', 'ramp'],
 ['sunday', 'resisted'],
 ['sunday', 'pressure'],
 ['sunday', 'ramp'],
 ['sunday', 'up'],
```

Figure 37 - skip gram training data

To generate this data, it is necessary to enumerate the words list. This gives the system the index of the words of the list. Using this index, a range of words are selected and if the words selected are not the current word this relationship is added to the training data. The below code is responsible for executing the above logic.

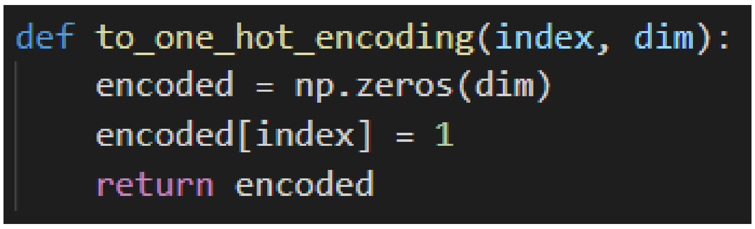


```python
def genskipgramdata(corpus, word2int):
    sentences = []
    data = []
    WINDOW_SIZE = 2

    sentences = [sentance.split() for sentance in corpus]
    for sentence in sentences:
        for index, word in enumerate(sentence):
            for neighbor in sentence[max(index - WINDOW_SIZE, 0) : min(index + WINDOW_SIZE, len(sentence)) + 1] :
                if neighbor != word:
                    data.append([word, neighbor])

    return data
```

Figure 38 - generate skip gram data

The next step is to convert the data into one hot encoded data.

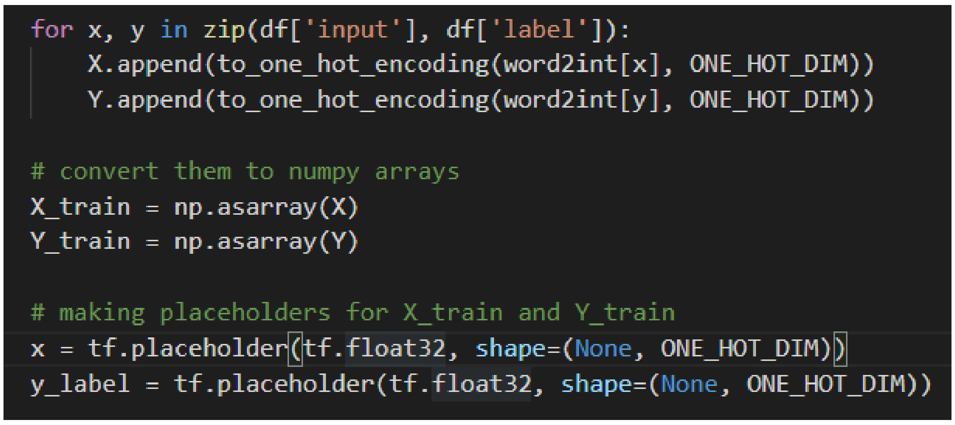
To do this the system gets the count of the words. This is used to set the size of the encoded data which are all zeros. The integer value of the word that was generated earlier in the code is then used to set that index to 1. This is done for all words, giving each its own encoded value. As in the below code.



```python
def to_one_hot_encoding(index, dim):
    encoded = np.zeros(dim)
    encoded[index] = 1
    return encoded
```

Figure 39 - one hot encode

The next step is to generate the computational graph which is done using the TensorFlow library. The first step to create placeholder tensors for the X and Y training data. The x data is the input word and the y data is the target or what the word2vec algorithm will guess. A placeholder is a type of tensor that is used to handle the feeding of a value. This is not evaluated directly and states what data type is expected and what shape the data which is the length of the one hot encoding. A tensor is an object of n dimensional arrays of a float, int or string.

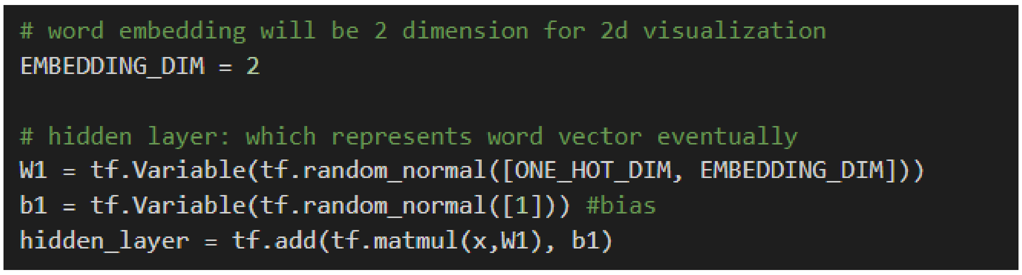


```python
for x, y in zip(df['input'], df['label']):
    X.append(to_one_hot_encoding(word2int[x], ONE_HOT_DIM))
    Y.append(to_one_hot_encoding(word2int[y], ONE_HOT_DIM))

# convert them to numpy arrays
X_train = np.asarray(X)
Y_train = np.asarray(Y)

# making placeholders for X_train and Y_train
x = tf.placeholder(tf.float32, shape=(None, ONE_HOT_DIM))
y_label = tf.placeholder(tf.float32, shape=(None, ONE_HOT_DIM))
```

Figure 40 - train data and placeholder variables

The next step is choosing the word embedding. As this embedding is also the output of this back-end system as such it is important to consider what effects larger and smaller values will have. Since the system is using the Euclidean distance to compare the likely hood of an advertisement the curse of dimensionality applies. This means that as we use more dimensions to represent the data the system will lose the ability to tell these points apart as the points start to appear equidistant from each other. Another problem caused because of the curse of dimensionality is as more dimensions are added the same amount of data does not fill the same amount of space meaning that the system may be adding irrelevant attributes to the data. Testing this backend system is difficult to do in a computing setting as such it is important to be able to visualise the results of this system. Because of these factors the word embedding dimension will be two. This will allow visualisation of the output data and allow the system to effectively calculate the distance from one point to another. This means both better results and administrator oversight of the unsupervised machine learning model.

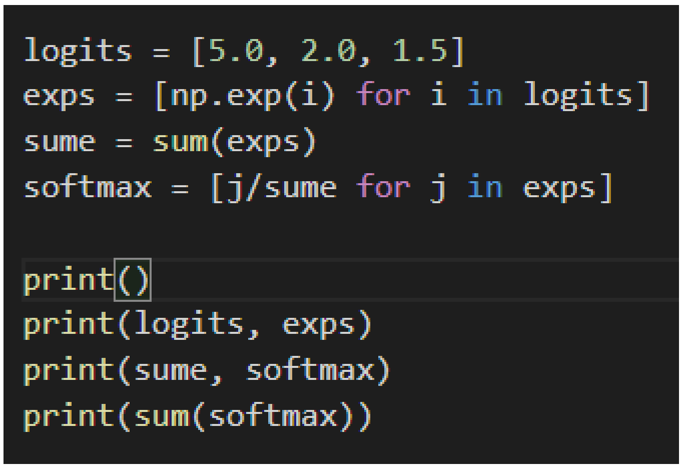
The next step to creating this system is to create the hidden layer or the word2vec algorithm. This consists of W1 and b1 which are the weights and bias respectively. The x placeholder created earlier multiplied by the weights plus the bias will create the hidden layer of the algorithm.



```
# word embedding will be 2 dimension for 2d visualization
EMBEDDING_DIM = 2

# hidden layer: which represents word vector eventually
W1 = tf.Variable(tf.random_normal([ONE_HOT_DIM, EMBEDDING_DIM]))
b1 = tf.Variable(tf.random_normal([1])) #bias
hidden_layer = tf.add(tf.matmul(x,W1), b1)
```

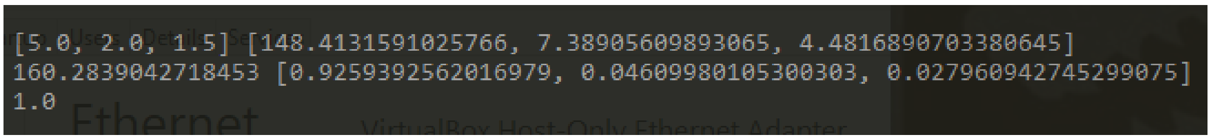
Figure 41 - dimensions and hidden layer

To create the output layer which will be predicting the next word the softmax function is utilized. The input to this function is the hidden layer multiplied by W2 which is the same math as W1 but in the other direction. The bias is the added to this. The softmax returns a tensor the same shape and type as the tensor inputted into the function. The softmax function turns the logit into probabilities of potential outcomes. A logit a tensor of type half or float. This logit must be converted into probabilities in order to select an output word. The process is best explained with an example. In the below code a list of “logits” is defined. The exponentials of each of the logits is calculated using the NumPy function. Exponentials are utilised here a logit is a logarithm of odds and a logarithm ranges from positive to negative infinity. This means when adding logits together it will not produce the correct normalisation. Exponentials solves this as it turns them into values zero and up. This then allows for the percentage to be calculated for each logit using the sum of exponentials divided by sum of exponentials. To do this the TensorFlow function tf.nn.softmax is utilized.



```
logits = [5.0, 2.0, 1.5]
exps = [np.exp(i) for i in logits]
sume = sum(exps)
softmax = [j/sume for j in exps]

print()
print(logits, exps)
print(sume, softmax)
print(sum(softmax))
```

Figure 42 - softmax calculation NumPy



```
[5.0, 2.0, 1.5] [148.4131591025766, 7.38905609893065, 4.4816890703380645]
160.2839042718453 [0.9259392562016979, 0.04609980105300303, 0.027960942745299075]
1.0
```

Figure 43 - softmax output

In the above paragraphs the structure of the algorithm and any cleaning and structing of the data was completed. In the later paragraphs, the training and output of this backend system are explained.

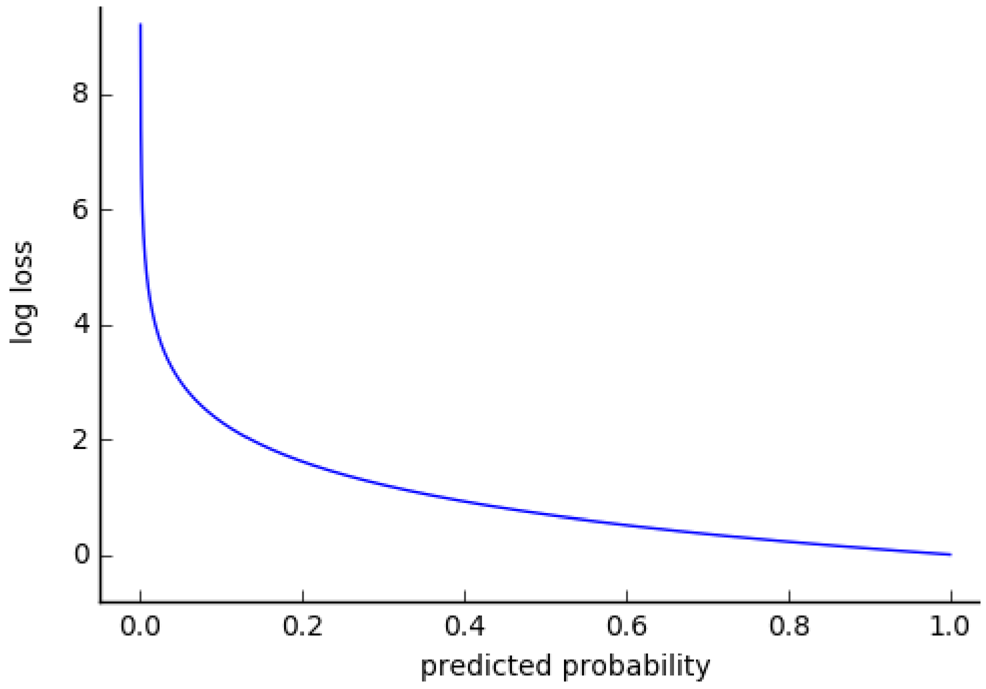
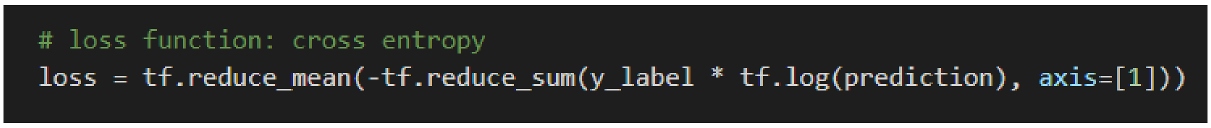


Figure 44 - Log loss or cross entropy loss

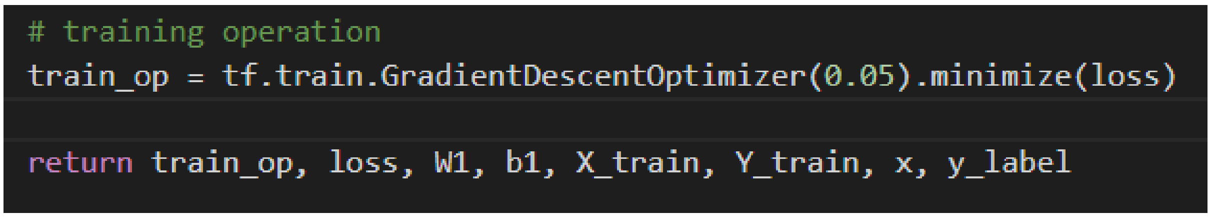
The next step is to create the loss function for the training step of the algorithm. Using softmax it is ensured that the values are compatible with this function. The loss function is responsible for defining how good of a prediction is regarding the expected outcome. The loss function utilised in this model is cross entropy loss or log loss. This function is utilised in models where the output is between 0 and 1. In the above figure, using this method log loss heavily penalises values that have a high predicted probability but are wrong. As the probability goes to 1 the loss slowly decreases. But as the predicted probability decreases the log loss increases rapidly. Below is the code used to implement this function. The first step is to multiply the target data which is the y label by what the log of algorithm predicted. The sum of each row is then calculated and then the mean is then used to get the loss.



```
# loss function: cross entropy
loss = tf.reduce_mean(-tf.reduce_sum(y_label * tf.log(prediction), axis=[1]))
```

Figure 45 - loss function

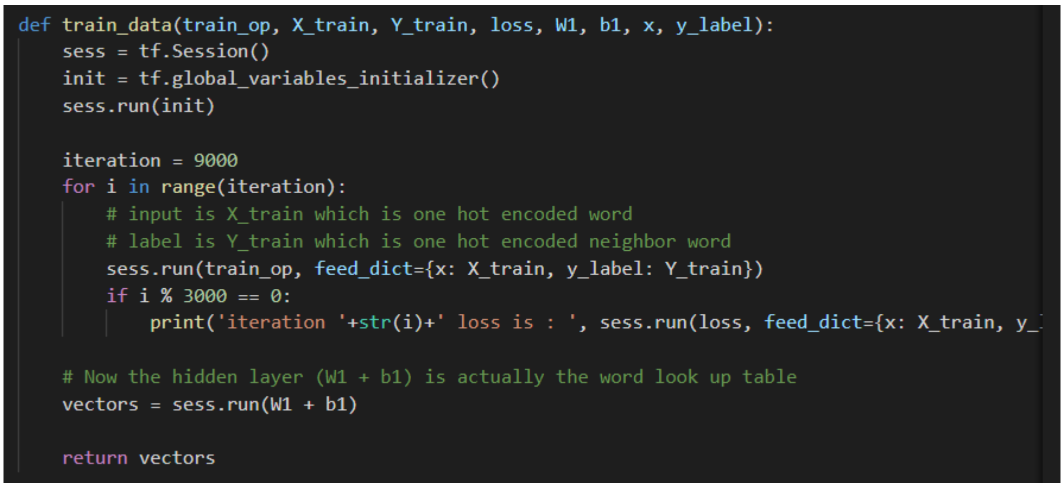
The following step is to create the training operation of the algorithm. The point of this operation is to minimize the loss function. The gradient decent algorithm is one of the most common algorithms used for this step. The learning rate is the size of the step taken to get to the minimum point. A large step will get the program to the point faster, but this may result in the overshooting of the minimum point. After each step the slope of the line is calculated till this equal to zero. Meaning that when the slop hits zero the bottom of the curve is met. In code, the tf.train.GradientDescentOptimizer class is used with a step / learn rate of .05. The minimize method is then called with the argument of the loss function created in the last step. The minimize method computes the gradients and then applies these gradients to the tensor. The training operation, loss function, hidden layer components, the training sets and the placeholder variables for the training sets.



```
# training operation
train_op = tf.train.GradientDescentOptimizer(0.05).minimize(loss)

return train_op, loss, W1, b1, X_train, Y_train, x, y_label
```

Figure 46 - training function

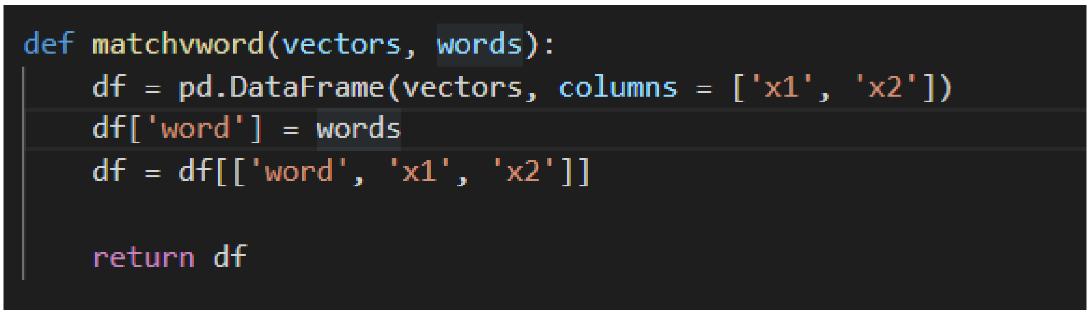
The next stage of this backend system is to train the word2vec model. The first step in training the model is to setup the TensorFlow session and initialiser functions. Only after the initialiser is run do the tensor variables hold any data. The session method creates the environment. This session is then run with the init variables. Next the system trains the model; an iteration count is provided. This value can be adjusted to provide the best time for the system to complete as the accuracy of the model will be quite low, so excessive training provides little value. In later iterations of this project looping indefinitely and breaking out of the loop when the change in the accuracy is below a defined value may save resources. For each iteration, the training method and a dictionary of the training and target values are used to train the model. For administrators the accuracy of the iteration is printed to standard output. When this is completed the system now has a model that can predict the next word based on a given word. This is accomplished through the hidden layer that was just trained. Therefore, the hidden layer is the relationship of the words to each other and extracting this hidden layer gives the vectors necessary to show these relationships mathematically.



```python
def train_data(train_op, X_train, Y_train, loss, W1, b1, x, y_label):
    sess = tf.Session()
    init = tf.global_variables_initializer()
    sess.run(init)

    iteration = 9000
    for i in range(iteration):
        # input is X_train which is one hot encoded word
        # label is Y_train which is one hot encoded neighbor word
        sess.run(train_op, feed_dict={x: X_train, y_label: Y_train})
        if i % 3000 == 0:
            print('iteration '+str(i)+' loss is : ', sess.run(loss, feed_dict={x: X_train, y_

    # Now the hidden layer (W1 + b1) is actually the word look up table
    vectors = sess.run(W1 + b1)

    return vectors
```

Figure 47 - train model

The final step to this backend system is to match the vectors to their respective word. The output of this is what is returned to the flask API.



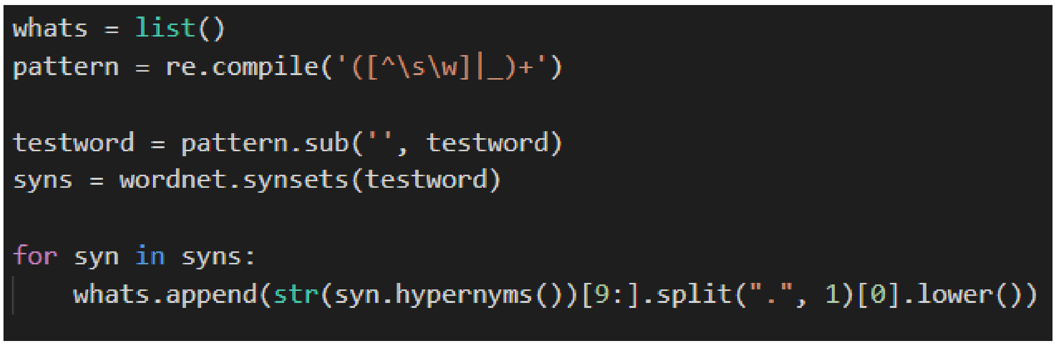
```python
def matchvword(vectors, words):
    df = pd.DataFrame(vectors, columns = ['x1', 'x2'])
    df['word'] = words
    df = df[['word', 'x1', 'x2']]

    return df
```

Figure 48 - match word to vectors

Compare

Comparing the outputs of the afore mentioned algorithms is done using the wordnet database and by using Euclidean metrics. As discussed in the design chapter it is important to be able to compare the outputs of these systems.

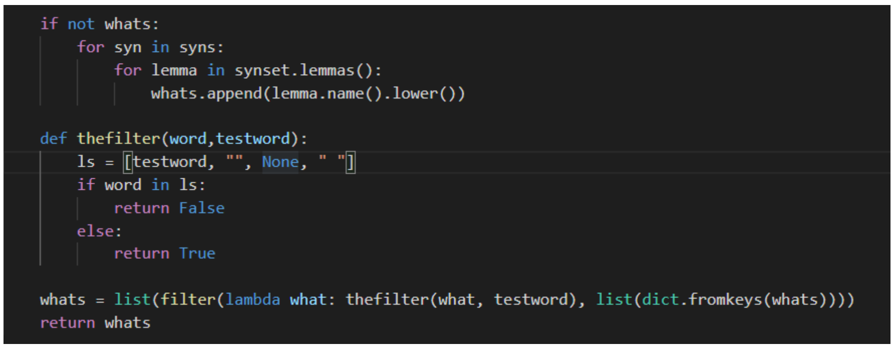
In order to query the wordnet database the word must be cleaned of any non-alphanumeric values. This is done using the re library and the following regular expression '([^\s\w]|_)+'. This expression is compiled so it can be used against the submitted word. The word is then queried against the wordnet database using the synsets method. The returned object is a list of the synsets of that word. This is then looped through and the hypernyms method is used to query the synset for the category of the word. The category is then returned in the wordnet format this is then stripped of said format. To do this the split method is utilised, using the first full stop as the item to split upon. Taking the first item from the split, using the lower method to lowercase the word and removing the first nine letters from the final output as wordnet prepends “hypernym” to the beginning of the output word. These hypernyms are the added to the output list. This logic is executed in code form below.



```
whats = list()
pattern = re.compile('([^\s\w]|_)+')

testword = pattern.sub('', testword)
syns = wordnet.synsets(testword)

for syn in syns:
    whats.append(str(syn.hypernyms())[9:].split(".", 1)[0].lower())
```

Figure 49 - synset and hypernyms

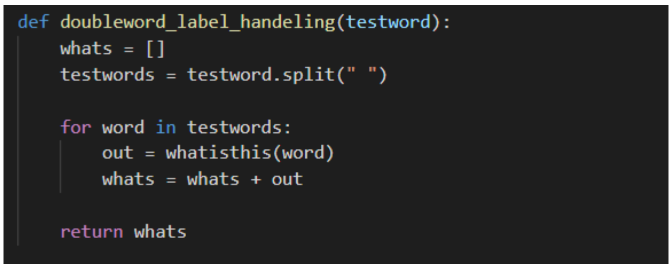
The next step in the method is to have a backup if this fails, as discussed in the design chapter if there are no synsets the system falls back on lemmas which are the dictionary form of the word. then the system filters out any empty, null value and original words. The returned value is a list of similar or synset words to the input word. This is completed by the below code.



```
if not whats:
    for syn in syns:
        for lemma in synset.lemmas():
            whats.append(lemma.name().lower())

def thefilter(word,testword):
    ls = [testword, "", None, " "]
    if word in ls:
        return False
    else:
        return True

whats = list(filter(lambda what: thefilter(what, testword), list(dict.fromkeys(whats))))
return whats
```

Figure 50 - backup and clean output

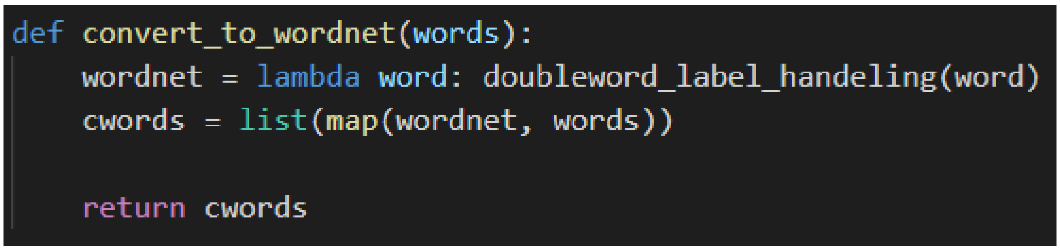
The next step is to deal with passing double words to the system. As discussed in the design and testing chapters this can happen as the labels of the image processing system can be double words like red wine. As red wine can be used to represent the colour or alcohol in speech it is necessary to utilise both words in the matching process. To do this the system splits on spaces, calls the wordnet database querying function and concatenates the output of the wordnet database.



```
def doubleword_label_handeling(testword):
    whats = []
    testwords = testword.split(" ")

    for word in testwords:
        out = whatisthis(word)
        whats = whats + out

    return whats
```

Figure 51 - handle double words

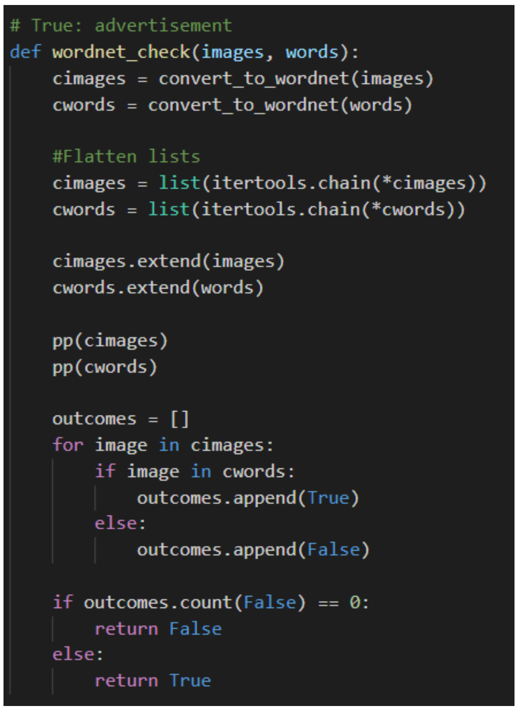
To handle a list of words at once the convert to wordnet function returns a list of the converted words in a list. To get these words a lambda function is created that passes words into the double word handling function. The map method is used to map this lambda function to input words. Below is the necessary code to perform this operation.



```
def convert_to_wordnet(words):
    wordnet = lambda word: doubleword_label_handeling(word)
    cwords = list(map(wordnet, words))

    return cwords
```

Figure 52 - wordnet output

To check if a list of image labels contains an advertisement the function discussed above is called. The output of this function is then flattened to a list and the original words are re added to the list. If the image list is loop through and if the image is in the c words list the value of true or false is appended to a new list. The outputted list then can be counted for the amount of false values and if there are no false values that means there are no advertisements and if there are advertisements true will be returned.

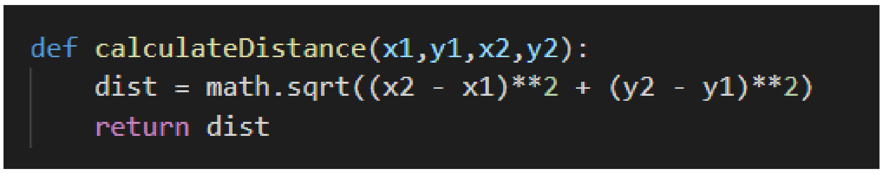


```
# True: advertisement
def wordnet_check(images, words):
    cimages = convert_to_wordnet(images)
    cwords = convert_to_wordnet(words)

    #Flatten lists
    cimages = list(itertools.chain(*cimages))
    cwords = list(itertools.chain(*cwords))

    cimages.extend(images)
    cwords.extend(words)

    pp(cimages)
    pp(cwords)

    outcomes = []
    for image in cimages:
        if image in cwords:
            outcomes.append(True)
        else:
            outcomes.append(False)

    if outcomes.count(False) == 0:
        return False
    else:
        return True
```

Figure 53 - wordnet check

Now that the system knows that there could be advertisements it must work out how likely this is. To do this the vectors from the word2vec system is used. The first step to this is to implement the

Euclidean distance method as a function. This function will take in four values being the two-dimensional points and will return the distance between these two points. As stated earlier in the design chapter Euclidean distance was chosen for its simplicity to understand nut this method required more mathematical operations than other methods. The Euclidean distance is simply x2 – x1 squared plus y2 – y1 squared and then this value is square rooted. Below is the code to complete the Euclidean distance method.



```python
def calculateDistance(x1,y1,x2,y2):
    dist = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return dist
```

Figure 54 - Euclidean distance

4.6. Conclusions

In conclusion, the systems development is chronicled here. The main issues in the development process were the communication between the flask api and the extension, saving or recompiling the image classification model and iterating through arrays in JavaScript. Two of these problems were solved using the for each method to iterate through arrays in JavaScript and implementing CORS in the flask API and chrome extension but saving and recompiling the image classifier was not solved. The back-end systems implemented are word2vec, an image classifier and the compare script. A chrome extension was built, and Flask was used to create an API to link the chrome extension to the backend.

5. Testing and Evaluation

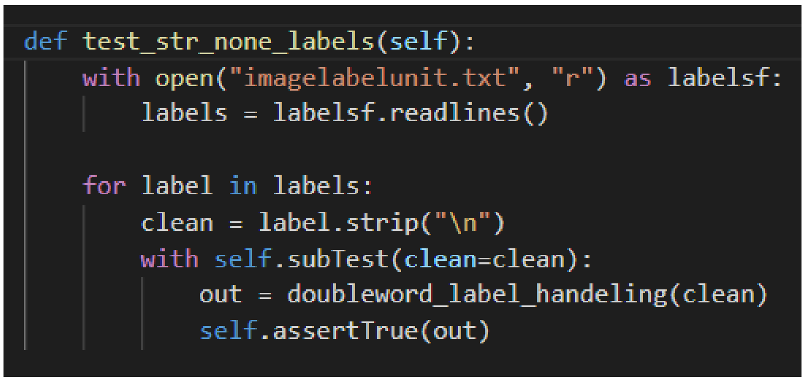
5.1. Introduction

In this chapter, the explanation of how unit testing occurred and how the final system was evaluated.

5.2. System Testing

In software development, testing is an important part of the development process. In the RAD methodology this is especially so. All software has bugs, to mitigate this testing after iterating is essential. Throughout the development of this project both unit testing and ad hoc testing to validate the iterations of the programme.

Unit testing was most effective when applied to the code that decides what a word is. Using the labels of the image classification algorithm this function was tested and various changes were made because of this testing. For example, the original function did not consider double words. This was never a consideration as the word2vec model ensures a one-word format whereas in the image classifier 400 words are double words like red wine. The bug would not have been detected till much later or at all if not for the unit testing.

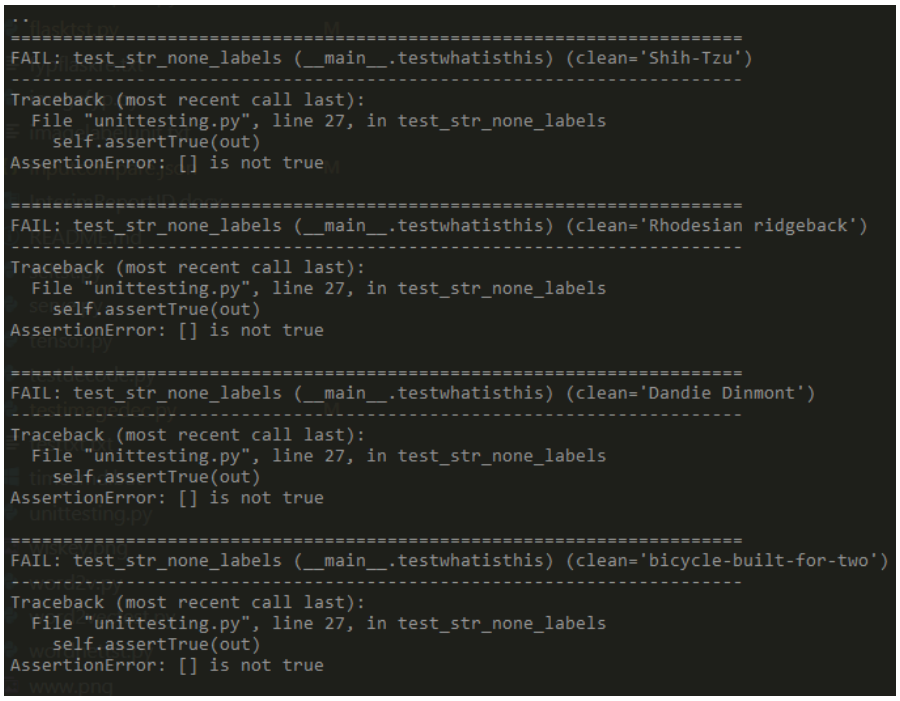


```
def test_str_none_labels(self):
    with open("imagelabelunit.txt", "r") as labelsf:
        labels = labelsf.readlines()

    for label in labels:
        clean = label.strip("\n")
        with self.subTest(clean=clean):
            out = doubleword_label_handeling(clean)
            self.assertTrue(out)
```

Figure 55 - unit test example

The above code is responsible for one of the subtests. The imagelabelunit.txt contains the labels of the image classifier, this is read in and cleaned up. For every word in this list, the wordnet database will be queried if there is no output then the query failed meaning there is a problem. The failure will be reported in the output of the unit test script.

In order to execute the above code, the unit test framework is utilized. The testing functions are incapsulated in a class. This class has the unittest.TestCase passed to it which sub classes unittest.TestCase. Contained in this new class is the methods for testing the system. These methods assert something to be True, False, Equal, etc. This creates the test cases that will be run. In order to run the test script, the main() method is called. This method is used to create a command line interface for the script as seen in the below image. The first 2 dots mean succeeded tests whereas a Fail is a test fail. The failures show what test class failed, what method failed, and the variables used. From this information, the reasons for the failure can be mitigated.



```
..
======================================================================
FAIL: test_str_none_labels (__main__.testwhatisthis) (clean='Shih-Tzu')
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unittesting.py", line 27, in test_str_none_labels
    self.assertTrue(out)
AssertionError: [] is not true

======================================================================
FAIL: test_str_none_labels (__main__.testwhatisthis) (clean='Rhodesian ridgeback')
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unittesting.py", line 27, in test_str_none_labels
    self.assertTrue(out)
AssertionError: [] is not true

======================================================================
FAIL: test_str_none_labels (__main__.testwhatisthis) (clean='Dandie Dinmont')
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unittesting.py", line 27, in test_str_none_labels
    self.assertTrue(out)
AssertionError: [] is not true

======================================================================
FAIL: test_str_none_labels (__main__.testwhatisthis) (clean='bicycle-built-for-two')
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unittesting.py", line 27, in test_str_none_labels
    self.assertTrue(out)
AssertionError: [] is not true
```

Figure 56 - unit testing

As seen above, the current state of the unit test is nine sub checks fail. Many of the words that fail are not in the wordnet database or are irrelevant for the evaluation of this experiment as such this is unnecessary to solve this problem.

An ad hoc method of testing was required for the Flask API as with this application it either works or it doesn't. To do this testing the curl tool was utilised as the client plugin had not been developed at the time. This ended up being a poor method of testing as the chrome plugin utilises CORS which is used when a web app must query a fetch API (23). The apparent successful testing of the API meant that when developing the POST functionally of the chrome extension the problems experienced seemed to purely be on the extension side of the application. After implementing the CORS package in the flask API and adding CORS mode, accept header and body json this issue was solved.

Another problem in testing the API in this manner is was in running the algorithms one at a time and then closing the process. This meant that when running the scripts in the flask API the process wold never close witch is the intended functionality. But the methods to free the resources are not guaranteed to function. These methods include TensorFlow's build in methods like reset_default_graph() and close() but also using library's like numba which have a Cuda implementation to select the GPU and close the current thread. As such, the best way to manage the memory consumption of these scripts is to create a new process and when they are finished processing the data end said process freeing the memory back to the system. This is guaranteed to happen as the process had ended (22).

5.3. System Evaluation

Currently all parts of the system are evaluated separately. Each part requires different approaches to be evaluated.

The image classification is evaluated in two ways. The model is evaluated against the ImageNet dataset and has a 78% accuracy at this task. But these images are not advertisements as such this must be validated against real world data. To complete this evaluation car and food images were focused upon. During this evaluation, the model would return not exactly the corrected or expected answer, but it would be in the same general domain. For example, an image of a drag racer was sent to the image model and the result was a train. Because these images share a synset this is not as big of a problem as it first seems but because of this the overall system will lose accuracy if the article was talking about trains and the advertisement was a car. This is a recurring issue with this method of advertisement detection.

Word2vec is another difficult model to evaluate. For the purposes of this evaluation the same text data was utilised and then run multiple times. The same vector values will not be preserved on each outcome but visually the same general grouping was observed.

5.4. Conclusions

In conclusion, all software has bugs and because of this it is necessary to test for their presence. There are many methods to completing this requirement of software development and not all methods are suitable for certain programs. Choosing the correct methods for software testing can lead to better or worse results for the completed system. The image and text algorithms were evaluated separately.

6. Conclusions and Future Work

6.1. Introduction

In this chapter, various conclusions about the design, development and evaluation were discussed. Future possibilities for this system are also explored.

6.2. Conclusions

Word2vec is an integral part of this system. In the course of researching many blogs and scientific papers were explored on the topic. The continuous bag of word model vs the skip gram model was reviewed, and it was discovered that the skip gram method provided greater weight to nearby context words. This means that the skip gram method provides better word vectors for this use case as such this method was chosen.

Wordnet is another integral part of this system as it provides a way to compare the outputs of the chosen machine learning models. Wordnet is a large lexical database of English nouns, verbs, adjectives and adverbs. These words are grouped into cognitive synonyms or synsets, each of which express a distinct concept. This network can then be queried resulting in a method of grouping the outputs of the machine learning models.

In the design chapter the front-end UI explored using the prototyping process. This process consists of low, medium and high-fidelity designs, the final design is then used to create the front-end application. As the front-end of this system is relatively simple the high-fidelity prototype was implemented and iterated upon in html and CSS. This method of design happened in a natural progression using the RAD method of software development.

After the design of the front-end how the systems would interact with each other was decided. Flask was chosen as the web framework because of its simplicity and support for API implementation. The overall design of the system calls for a request and response architecture. Flask lends itself to this form of design. Each backend system can be called separately, this means that the image classification and wor2vec systems can be used for different systems. The compare system on the other hand is specialised to this problem domain because of this it has a high probability of not being useful to other systems. When implementing the Flask API, a new requirement for the system was discovered, this being the CORs method, thankfully because of the simplicity and support of flask extensions were already implemented allowing the implementation of this feature.

The method in which the outputs of the machine learning models are assessed against each other is an integral part of this system. The design of this system first required a method of grouping similar ideas together this will be handled by wordnet. Now that the system can match the words against each other the likelihood must be calculated. To calculate the likelihood of an image being an advertisement the Euclidean distance is used, this idea is like how the K-Nearest Neighbours Algorithm functions, but the least connected point is most likely to be an advertisement and the most connected is least likely. This allows the program to show the user how sure it is that the image is or is not an advertisement.

When testing the system many issues were found or masked by the testing methodology used. In future versions of this system more effective testing should be utilised. An example of good testing that occurred in the process of the development of this system is the unit testing that was conducted. The unit testing allowed the identification of bugs before the bug was ran into in a live environment. Whereas when ad hoc testing was utilised the bug was masked till the client extension was created. As such the problem in connecting the two systems was assumed to be client side exclusively. This is an example of using poor testing, to avoid this issue in future the same time of requests should be utilised.

This method of advertisement detection has many flaws. The primary flaw is these models take time to execute, this problem can be lessened by using more powerful hardware but will never be as fast as the current systems that use the ad list method of detection. As a result, the final system would be uses as a companion application to already existing solutions. As stated in the evaluation chapter, the system loses accuracy when dealing with similar item types. This problem can be dampened by retraining the model for advertisement specific items like logos, this idea is discussed further in future work.

6.3. Future Work

Browser image classification

In the development of this project this features development was shelved for a later iteration. In future iterations this feature should be implemented, or the user should be able to choose between having the image classification model on their own client vs on a server. For older hardware this may be a necessary step for the system to run effectively. Having the model run on the client’s computer will also remove load from the server and this method will add more privacy for the client user. If not properly regulated the system could be utilised to track and monitor the user, moving the image processing to the client machine is a good way to start mitigating this issue.

Retrain the inception model

The default model is the first runner up for the ImageNet Large Scale Visual Recognition Challenge. ImageNet is an image database organized according to the WordNet hierarchy. Retraining the model for more advertisement-based images would improve the accuracy for this application of the classifier. These advertisement-based images could be company logos. The logo may not be the main thing the image classifier identifies but if the prediction has a high enough percentage the system can be sure that the image is an advertisement. The image can then be removed without having to compare it with the text. This will make the system faster and more efficient as the main system will not have to be queried if the image classification is done in browser.

Adding labels for images like the android and apple app stores and dropping these images would also be useful for the system. Images like the above mentioned provide no value to this system and removing them will improve the systems effectiveness. This could also have a negative effect on the system as using it when an article is about these items would cripple the system. Download our app along with these images are so ubiquitous on many websites on the internet. Thus, an elegant method for handling these types of images should be found. An expanded image scraper could also help handle this issue.

Lan implementation

The current implementation of this project is designed to run on a webserver and to be queried by a chrome plugin this idea came from current implementations of AdBlock in browsers. Other methods to solve this problem exist like Pi-hole as mentioned earlier and Privoxy which is a web proxy for modifying and filtering webpages. Following these projects ideas, A server which sits in-between the clients and the Internet could modify pages achieving a similar solution.

The advantages of this approach would be the privacy of people using the system, as the system is no longer shipping the contents of the user's webpages to an external server. If the server were to collect logs or was compromised this system could be used as another data point for the tracking and identification of the advertisement servers thus nullifying the purpose of the system this new implantation would mitigate this issue. Another advantage would be the mitigation of browser fingerprinting. This is the process of defining who you are by scraping the setup of your browser like the operating system you are running, the browser and its version, the language and time zone, plugins and many more. This new implementation would effectively make everyone on the network identical.

The biggest disadvantage of this approach is the current resources necessary to run the algorithms. The image classification portion of the system has been well optimised and is effective at classifying images but word2vec relies on training a new dataset on each query this takes considerable resources, making the required hardware expensive. As implementations of this algorithm get more efficient this implementation will become feasible to implement in home environments. Another problem with this approach is by default it won't work with https traffic; this would have to be solved to make it an effective solution.

This is another method of mitigating the security concerns associated with sending the data to an off-site server. Running the system in a local network means the user data is parsed locally, this means that the server should not be making any connections to the servers that provide this application on the wider Internet. Even if this were the case the provider domain could be blocked with pi-hole, preserving the security of the user.

Removing advertisements

Now that the system can identify images that it thinks are advertisements the next logical step would be to remove them in some fashion. In its current iteration, this can be done with JavaScript by removing the element from the DOM. The implementation proposed above would remove the advertisements before it even gets to the client computer.

Word clouds

Word clouds are a simple way of visualising the frequency of a word in text. Early in the development of this project the frequency of the words was disregarded as a method to define what the text is about as in testing the top words tended to not describe the text was about and as such the system uses word2vec which takes every word in the text into account. But when it comes to text like blogs, tutorials, speeches, etc this method of text classification becomes much more viable. The method is also far more lightweight than word2vec meaning it could be implemented in browser either replacing word2vec on sites that contain blog in the URL or bolstering word2vec's results on other sites.

User feedback

Implementing a method for user feedback to what the system thinks is or is not an advertisement could be a useful feature. This would allow the system to add more weight to certain words or outcomes of the machine learning models utilised in the system. This could be implemented in a multitude of ways. Regarding image classification, inception returns a multitude of labels with their likelihood of being the item. Most advertisements are focused on one item, but this is not always the case. In these situations, having user input on if the image is or is not an advertisement or, if one of the prompted items from a list is in the advertisement would help this portion of the system to become more accurate. This same approach could be utilized for the word2vec model but instead of adding to the likely hood of something being an advertisement or not, the weight generated by the Euclidean distances could be changed. This could not be a static value for each graph instead it must be some form of multiplier of the weight value. This multiplier could be added or subtracted from the weight value.

Expanded image scraping

When it comes to websites there are many images that do not serve an informational purpose. These images include items like custom close buttons, custom headers, etc. These types images cause problems in the running of this system as they are neither content nor advertisements. They are visual buttons, etc. Users don't see these images as images as they are part of the aesthetic of a site. One of the was these images could be removed is the size of the image. As these images serve the purpose of buttons, they are often quite small in resolution. Removing images sub a certain value could be detrimental though as such testing should be conducted on websites to ensure this will not be the case. Many one colour images are also found in the current scraper. These could be backgrounds and as such fall into the same category as the buttons. Removing images with a single colour could also be completed on the client side of the system.

Bibliography

1. Manifest V3 [Internet]. Google Docs. [cited 2019 Sep 29]. Available from: https://docs.google.com/document/d/1nPu6Wy4LWR66EFLeYInl3NzzhHzc-qnk4w4PX-0XMw8/edit?usp=embed_facebook

2. 896897 - Extensions: Implement Manifest V3 - chromium - An open-source project to help move the web forward. - Monorail [Internet]. [cited 2019 Dec 7]. Available from: https://bugs.chromium.org/p/chromium/issues/detail?id=896897&desc=2#c23

3. General Data Protection Regulation (GDPR) – Official Legal Text [Internet]. General Data Protection Regulation (GDPR). [cited 2019 Sep 29]. Available from: https://gdpr-info.eu/

4. Hill R. gorhill/uBlock [Internet]. 2019 [cited 2019 Dec 7]. Available from: https://github.com/gorhill/uBlock

5. AdBlock. AdBlock [Internet]. [cited 2019 Dec 5]. Available from: https://getadblock.com

6. Adblock Plus | The world's # 1 free ad blocker [Internet]. [cited 2019 Dec 7]. Available from: https://adblockplus.org/en/

7. telekrmor. Home [Internet]. Pi-hole®: A black hole for Internet advertisements. [cited 2019 Dec 7]. Available from: https://pi-hole.net/

8. Google Chrome - The New Chrome & Most Secure Web Browser [Internet]. [cited 2019 Dec 7]. Available from: https://www.google.com/chrome/

9. W3Counter: Global Web Stats - November 2019 [Internet]. [cited 2019 Dec 5]. Available from: https://www.w3counter.com/globalstats.php?year=2019&month=11

10. Firefox Privacy Promise [Internet]. Mozilla. [cited 2019 Dec 7]. Available from: https://www.mozilla.org/en-US/firefox/privacy/

11. Rong X. word2vec Parameter Learning Explained. arXiv:14112738 [cs] [Internet]. 2016 Jun 5 [cited 2019 Dec 5]; Available from: http://arxiv.org/abs/1411.2738

12. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the Inception Architecture for Computer Vision. arXiv:151200567 [cs] [Internet]. 2015 Dec 11 [cited 2019 Dec 5]; Available from: http://arxiv.org/abs/1512.00567

13. WordNet | A Lexical Database for English [Internet]. [cited 2019 Dec 7]. Available from: https://wordnet.princeton.edu/

14. TensorFlow Lite [Internet]. [cited 2019 Dec 7]. Available from: https://www.tensorflow.org/lite

15. Flask-RESTful — Flask-RESTful 0.3.7 documentation [Internet]. [cited 2019 Dec 7]. Available from: https://flask-restful.readthedocs.io/en/latest/

16. EasyList - Overview [Internet]. [cited 2019 Dec 7]. Available from: https://easylist.to/

17. Software Engineering | Classical Waterfall Model [Internet]. GeeksforGeeks. 2018 [cited 2019 Dec 7]. Available from: https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/

18. Rapid Application Development: Definition, Steps, Advantages and Case Study [Internet]. 2018 [cited 2019 Dec 7]. Available from: https://kissflow.com/rad/rapid-application-development/

19. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed Representations of Words and Phrases and their Compositionality. In: Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ, editors. Advances in Neural Information Processing Systems 26 [Internet]. Curran Associates, Inc.; 2013 [cited 2019 Dec 7]. p. 3111–3119. Available from: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

20. HTTP Methods GET vs POST [Internet]. [cited 2020 Apr 1]. Available from: https://www.w3schools.com/tags/ref_httpmethods.asp

21. English Letter Frequency Counts: Mayzner Revisited or ETAOIN SRHLDCU [Internet]. [cited 2020 Apr 1]. Available from: http://norvig.com/mayzner.html

22. CUDA Device Management — numba 0.13.0 documentation [Internet]. [cited 2020 Apr 9]. Available from: http://numba.pydata.org/numba-doc/0.13/CUDADevice.html

23.Cross-Origin Resource Sharing (CORS) [Internet]. MDN Web Docs. [cited 2020 Apr 11]. Available from: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS