UNIVERSITY OF WARWICK

Automated Image Geolocation OSINT

GROUP REPORT

Authors
Matthew WIGHT
Daniel FREDRICKSON
Lara GOSS
Lukasz STANASZEK

Supervisor
Khalil CHALLITA

ABSTRACT

Image-based open-source intelligence (OSINT) is a vital tool for search-and-rescue, verifying image details and locating criminals. Current image OSINT frameworks can be tedious to use and highly complex, requiring extensive training and technical knowledge. This report documents the design and development process of AutoOSINT, a user-friendly desktop app capable of performing image geolocation OSINT by identifying a variety of visual clues in an image, including famous landmarks, road signs, mountains and car number plates.

KEYWORDS

OSINT, Image Processing, Computer Vision, Automated Geolocation, Feature Extraction, Deep Learning, Landmark Detection

April 30, 2024

Contents

Motivation

Open-source intelligence has long been a vital tool in a variety of fields including military intelligence, criminal investigations and cutting-edge journalism. Its use predates the internet: early OSINT operations were conducted by meticulously collecting newspapers and radio broadcast reports from around the world in an attempt to find any photos or articles that might give away information about a target[1]. Since the birth of the internet, OSINT has become an increasingly technical area, and experts follow complex frameworks[2] to extract information hidden within sources. This is often an extremely time-consuming task, making the process slow and expensive.

AutoOSINT is a user-friendly desktop app designed to assist in the process of conducting OSINT operations on images for geolocation. It has a variety of easy-to-use automated features which can be used alone or in conjunction with each other to predict the location of an image of any outdoor scene. The aim of this project is to make the OSINT process more efficient for users and to make OSINT more approachable for non-technical users. Intended use cases for the AutoOSINT app include:

- Search and rescue operations, where a user may be lost and must be geolocated using visual clues.
- Verifying location information about an image received by a source that is not trusted.
- Locating criminals who may have posted photos to social media giving away details about their location.

AutoOSINT is able to detect a variety of visual clues in an image for geolocation of both urban and mountainous scenes. Detectable features include famous landmarks, car number plates, mountain topology, road signs, and image metadata. The app is also designed to be easy for a non-technical user to use, while also providing enough utility and detail in predictions that it would benefit an OSINT expert.

Background

Commonly Used OSINT Tools

Popular online tools used for OSINT include search engines and web page scrapers. Modern search engines often have a range of advanced features which can be helpful for an expert looking to uncover information about a target. For example, reverse image search can be used to find visually similar images that may contain the same buildings, logos or people. Social media scrapers such as StalkScan[1] are often used in OSINT to scrape social media profiles for location tags and information included in captions and comments attached to an image. However, this technique comes with some ethical issues as it could be seen as an invasion of privacy, especially if any social media accounts scraped are private accounts.

Open source databases can also be helpful in order to contextualise information about an image. The GeoNames[2] user-editable geographical database consists of over 25 million geographical names describing over 12 million unique features, along with additional information about each feature. If an expert is able to identify a landmark (such as a city, hospital or university) in an image, they can use this tool to geolocate it.

Weaknesses of Traditional OSINT Tools

Conducting manual searches using search engines is often too time-consuming for the purposes of OSINT, and the results set is likely to be too wide for a human to individually check every website for relevant information. It is common to automate the gathering process by means of a web crawler to automatically collect this information according to some set of pre-determined rules. A weakness of this approach is that they can be difficult to implement, especially if scraping from a large number of websites, due to differences in HTML formatting and web page layout. It also still suffers from the problem of the results set containing an extremely large amount of data.

Another common way to collect web data is to use an API. For example, Google's programmable search API provides automated access to the results from a given query. APIs have the advantage of being easier to implement and being more ethically sound than a web scraper (because they require permission in the form of an API key). However, by nature they can only be used by experienced programmers, and they are often rate-limited which reduces the volume of data that can be gathered[3]. Therefore, it can be useful to use multiple search APIs during the OSINT process in order to elicit as much information as possible.

After any relevant photos, videos and metadata have been collected, a human expert can review and analyse the results in order to make relevant inferences. This could relate to people or objects in the image, the location a photo was taken at, the time it was taken or any other clues relating to the target of the OSINT operation. Due to the sheer volume of data available online, this is generally very time-consuming and is considered the most challenging aspect of OSINT[4].

It should also be noted that traditional OSINT methods are typically complex and can only be carried out by trained experts. Therefore, non-technical users are generally not capable of using any of

[1] https://www.spoofbox.com/en/tool/stalk-scan/how-to-scan-facebook-profiles
[2] https://www.geonames.org/

these tools for accurate image geolocation. The requirement for training directly leads to increased costs and slower progress for OSINT operations.

Existing OSINT Automation Tools

While many OSINT tools aim to automate a step of the intelligence collection process, much of the work is still left for a human expert to complete, which makes the process slow. In 2020, Zoder proposed a modular framework called Pantomath[5] to provide all the functionality needed throughout the whole OSINT process. It consists mainly of existing tools, integrated to form a practical solution for cyber threat intelligence. For example, it features a geolocation module which uses APIs to query 14 different IP geolocation services in the hope of receiving more reliable results by querying a range of sources. With multiple sources, the framework can also give a reliability score based on the variance in the results obtained. One limitation of Pantomath is its lack of graphical user interface for easy and intuitive use.

There exist several automated OSINT frameworks designed to be used on a more industrial scale: Maltego[3] is a link analysis software used for OSINT, initially released in 2007. It offers real-time data mining and information gathering, with discovered targets presented as a directional graph. The most significant advantage of Maltego is its state-of-the-art visualisation capabilities. Spiderfoot[4] is a similar automated OSINT framework which is able to continuously analyse a large volume of data elements.

A recently proposed machine learning model, PIGEON, is able to accurately geolocate Google Street View images, with 40% of its predictions being within 25 kilometres of the true location. In testing, it was more accurate than most humans. However, PIGEON does not feature a user interface for non-technical users and must instead be implemented by a programmer via an API.

[3] https://docs.maltego.com/support/solutions/articles/15000019166-what-is-maltego-
[4] https://www.spiderfoot.net/attack-surface-monitoring/

Advantages and Limitations of Existing Automated OSINT

The main motivation behind implementing automated systems for OSINT is reducing the amount of human effort required. As previously discussed, manual collection of data for OSINT is highly time-consuming, making OSINT a very expensive operation. Similarly, manual analysis of images and videos is a slow process and liable to mistakes because it relies on the knowledge and observations of a human.

Another significant advantage of automation is convenience for the end user: an automated system can aggregate a large number of OSINT tools into a single program with an intuitive user interface. This can help to make the job of a human expert much easier when conducting OSINT operations.

It has been suggested that there are four main challenges associated with the use of automated OSINT systems[6]:

- **Availability** – the desired information may not be available from any source.
- **Interpretation** – the user's intentions may be misread by the system.
- **Formulation** – the automated system may fail to return the required information due the search formulation.
- **Confusion** – the user may fail to recognise the relevance and significance of the results.

One solution suggested for all of these issues was to instead implement a semi-automated system which would allow users users to pose queries and give any additional information they might have throughout the OSINT process. This could help the automated system focus on the most relevant aspects of the data input for its analysis.

Project Requirements

Considering these issues surrounding OSINT research in its current state, a platform aiming to rectify a number of these would fulfil the following system requirements. These include both customer and *developer-facing* components. Customer-facing requirements are given in standard font. *Developer Facing Requirements are given by italics.* Any changes made to these requirements during research and development due to unanimous team decisions are given in red text.

Functional

Ref	Requirement	Justification
F1	**Input** F1.1) Should be able to upload an image of a location for OSINT analysis. *Should be able to input an image of file types: PNG, JPEG. The application should be able to accept images of size 1920x1080.* **(Must have)** F1.2) Should be able to input clues to the system. *Should be able to input additional data about an uploaded image/location. This is information known by the user beforehand - such as approximate location or time. This will be assumed true.* **(Should have)**	F1.1) There is a large number of different image types that store information in unique ways. The program shouldn't be limited to a single file type as there are many image sources and circumstances. This will maximise program applicability. F1.2) The user may have additional information regarding the uploaded image that has been obtained externally (such as verbal or paper information). This could increase the accuracy and speed of analysing an image. Re-prioritised from **Must have** to **Should have** due to changes in perceived importance of this feature.
F2	**Authenticity Verification** F2.1) The program should check if the uploaded image is generated by AI. *The program should check if the whole (or part of) the image is generated by AI.* **(Won't have)** F2.2) The system should check if the image was tampered with. *The program should detect compression-based artefacts and copy-move instances within the image.* **(Should have)**	F2.1) With the rise of generative AI, images should be screened. Fake/generated images should be identified and discarded. This will help stop the spread of disinformation and wasteful processing. De-prioritised entirely from **Should have** due to clarification of project purpose. F2.2) Examining the image for tampering will prevent misleading investigations and reveal real-world contextual information (i.e. whether someone is cooperating or trying to hide something).

F3	**Metadata Extraction** F3) The program should be able to extract useful information from the image metadata. *The program will collect metadata such as date, GPS coordinates and the camera ID - if such data is available.* **(Must have)**	F3) Information isn't only stored visually on an image, many images also contain very useful information as metadata. This can be information that narrows down (or even completely reveals) the location and date/time of the information. Information found via metadata extraction will not be taken as ground truth, but will be combined with other analysis results to avoid conclusions being drawn from incorrect or injected metadata.
F4	**Image Enhancement** F4) The program should enhance the image to reveal more image detail. *The program will use standard image enhancement/processing techniques such as:* F4.1) Gamma Correction **(Should have)** F4.2) Contrast Equalisation **(Should have)** F4.3) Sharpening/Blurring **(Should have)** F4.4) Spatial Filtering **(Could have)** F4.5) Further Noise Removal **(Could have)**	F4) An uploaded image may be distorted or poor quality. To capture useful results it may be necessary to pre-process the image and enhance it. The most common image enhancements (listed to the left) can improve image quality and help reveal more information. For example, brightening an image will reveal more detail in the dark areas, and removing noise can make text clearer.

F5	**Urban OSINT** F5.1) The program should recognise notable landmarks and tourist destinations. *The application should recognise well-known unique buildings and natural formations and match the photo to the location.* **(Must have)** F5.2) The program should perform a reverse image search to identify potential locations. *The program will use available APIs to perform a reverse Google image search. The results will provide similar images to identify common features (both in image data and metadata). such as common points of interest, source websites and captions.* **(Must have)** F5.3) The system should be able to recognise street signs and street names. *The program will extract location names from street signs and identify the location from given names.* **(Could have)** F5.4) The system should capture registration plate information from any visible car plates on an image. *The program will extract registration numbers from cars and pass them to another component which will obtain data about the car and infer location/date information.* **(Should have)** F5.5) The program should match faces against social media posts to identify people of interest. *The system will identify people of interest through social media. The system will then use the location history and other images for verification.* **(Won't have)** F5.6) The system should recognise logos and brand names and use them to verify location. *The program will identify stores by logo/name and perform feature matching between the uploaded photo and potential locations.* **(Could have)**	F5.1) Arguably, the easiest case of geolocation is when in a publicly known location like a tourist destination. Unique and distinguishable locations should be easily identifiable hence this is a must. F5.2) One of the most useful geolocation tools available to OSINT analysts is Google reverse image search. The Google search algorithms are extremely powerful and a really good starting point when there is very little information. Hence this feature must be introduced via API calls. F5.3) A good way to obtain a location from a photo is to identify street names on street signs. Therefore, being able to identify street signs and extract text would be a powerful geolocation technique. F5.4) Reading Registration plates would be a beneficial tool for two reasons. Firstly, the registration plate structure can hint at possible photo locations. Secondly, the car is sometimes the subject of interest and information may be wanted about it. This means a registration plate look-up could be run to learn about the vehicle. So this is a very useful tool. F5.5) Deprioritisation from **Could have** due to the consideration of potential ethical issues regarding the misuse of the platform for stalking. F5.6) Logos and brand names on buildings are a reliable and simple approach to verifying location as they stand out and rarely change location, this is a common OSINT geolocation technique.

F6	**Wilderness OSINT** F6.1) The program should use mountain range shape to identify location *The system should feature match an image against potential mountains to identify exact location* **(Should have)** F6.2) The application should use water bodies as a reference for geolocation *The program should use the relative position and shape of water bodies to geolocate an image.* **(Won't have)** F6.3) The system can use stars and constellations in a night photo to estimate general image location. *The program will use known star navigation techniques to estimate a set of countries that the night photo could have been taken from.* **(Won't have)**	F6.1) Geolocation in rural areas is typically more challenging because there are fewer identifiable features compared to urban environments. However, mountain ranges are a more uniquely distinguishable aspect of rural scenes which can be used for geolocation, and tools already exist to identify mountain ranges, so the finished project should include this feature. F6.2) Similar to mountains, bodies of water tend to be recognisable, and tools exist to help implement this feature, so this should be included. Deprioritised from **Should have** due to poor trade-off between difficulty of implementation and value added to predictions by this feature. F6.3) The positions of stars in the sky can be used to estimate the location on Earth of a scene. However, the human effort required to implement this feature would be high and would only be usable on scenes with a clear night sky, so it won't be included.
F7	**Ranking Algorithm** F7) The program should rank location suggestions from most likely to least likely (but still possible). *The system should feature a confidence measure. This will be calculated from all OSINT analysis techniques. It will be used to sort suggested locations.* **(Must have)**	F7) It would be detrimental to the purpose of the application to assume that suggestions will always be 100% accurate. It is much more useful for the user to list a number of likely suggestions along with the system's corresponding measure of confidence in each prediction.
F8	**Output** F8) The system should display a report of the OSINT findings to the user. This includes likely locations and times. *The program should allow a user to select from multiple levels of detail, this will provide a range of information obtained from the image.* **(Must have)**	F8) The user will want to see the results of their analysis, otherwise there would not be a point in using the system. Giving users the option to select from multiple levels of detail will tailor the image analysis output to the objective of each individual user - some users may require more complex and in-depth information while others may only be interested in basic findings.
F9	**Automation** F9) The system should allow the user to run all analysis tools automatically on a given image. **(Must have)**	F9) The user may want to see the results of all analysis tools. They should not be expected to run each tool individually as this is time consuming and is detrimental to the application's user experience.

F10	**Exact Location Prediction** F10) The system should be able to work out the exact location at which an image was taken. **(Won't have)**	F9) A common goal of geolocation is to pinpoint the location from which an image was taken. This feature was suggested by the project supervisor but not was included after research into Depth and Pose Estimation which highlighted the extreme complexity involved in developing a solution.

Table 1: Project Functional Requirements

Non-Functional

Ref	Requirement	Justification
NF1	**Learnability** NF1) The application should be intuitive to use. *A new non-technical user should be able to learn how to navigate the system within 15 minutes of start-up.* **(Must have)**	NF1) The system should be intuitive - this would mean fast training for new users. This would result in the system being integrated by more analysts on a wider scale. Furthermore, less documentation would be required to assist the user. A system that is too complex deters from introduction into the workplace.
NF2	**Usability** NF2) The system must have a Graphical User Interface which is easy to navigate. *A user should be able to navigate to any interface in under 30 seconds from any other interface.* **(Must have)**	NF2) The purpose of the system is to streamline and accelerate OSINT research, and may be used by OSINT researchers who themselves are not technical experts. System interface design should reflect this and not impede non-technical users.
NF3	**Responsiveness** NF3.1) Feedback should be given to users informing them if actions have been successful. *Include and display system messages showing the status and progress of image processing or search queries.* **(Should Have)** NF3.2) The system should remain operational 99.9% of the time. *System downtime should be very rare, ideally solely in the case of critical updates.* **(Should have)**	NF3.1) The system should keep the user well informed to help them keep track of the status of their requests. It should also inform them of errors/failures caused by malformed input or misuse of system features. NF3.2) The application will be used by researchers around the world in varying timezones, often where timing is critical. System downtime should be minimised to enable maximal global usage of the platform.
NF4	**Testability** NF4) The system must be tested modularly throughout development. During testing, there must be normal, extreme and erroneous tests on all features. *Each individual feature (as split in functional requirements) should be able to be tested individually. The entire, complete system can be tested as a whole.* **(Must Have)**	NF4) The system should be easily testable to achieve a bug-free application which functions as intended 100% of the time. This will ensure a high-quality product, with higher precision and reliability. It will also save the development team valuable time and effort during the testing phase.
NF5	**Scalability** NF5) The system should be able to handle large amounts of concurrent image uploads and requests. *This should be achieved without a significant loss in system performance.* **(Should Have)**	NF5) The system should be able to handle a large number of images and requests as it will be used by a large number of analysts at one time, where each one may upload and make searching and processing requests for a large number of images.

NF6	**Security** NF6) The system should always remain up-to-date. *The system should be continually maintained to use the latest versions of associated technologies.* **(Should Have)**	NF6) Up-to-date technologies are often faster and more secure than older technologies. Maintaining an up-to-date system minimises issues relating to poor performance and poor security.

Table 2: Project Non-Functional Requirements

Project Management

Team Structure

Regarding the overall team structure for the production of the project, a predominantly flat hierarchy has been applied to the team, with everyone's opinions holding equal weight. Each team member is expected to contribute fairly and is responsible for delivering their assigned tasks, thus introducing an accountability culture to the development process. Moreover, regarding the working structure of the group, sub-teams have been created for developing each tool using a collaborative paired-programming strategy - dubbed the *responsible-review system*.

The group has assigned a **Project Manager (Lukasz Stanaszek)** to monitor overall progress and oversee the integration of each development stage. This is achieved by managing individual work progress, identifying any time underestimates in the original schedule, and assigning additional team members to tasks where needed. In addition, the Project Manager is responsible for managing everyone's roles, creating and monitoring communication systems, and ensuring each team member's strengths were being utilised at every stage.

To ensure the responsibilities and accountability of each team member are equal, despite this Project Manager role, other administrative and organisational project roles have been delegated across the team, including an **Admin Contact, a Document Manager, a Customer Contact, a Meetings Coordinator and a Scrum Master**. This group structure, in which the project's management responsibilities are distributed evenly across team members, ideally prevents any member from becoming overworked, improving team morale and cohesion.

The team has assigned an **Admin Contact (Matthew Wight)** who oversees all communications with the Module Organiser. Another assigned role is the **Document Manager (Matthew Wight)**, overseeing and handling the project's code and written documents. This includes ensuring version histories are saved and collating each team member's different sections. The team also has a **Meetings Coordinator (Lara Goss)**, who would is responsible for organising the weekly group/supervisor meetings, and for taking the meeting minutes.

Furthermore, the team is working alongside a customer who, although they are not directly involved in the project development, still play a crucial role in the overall project's success. During this project's development, the chosen customer would be a business-orientated person who could provide new perspectives to project ideas. After each development phase, they will be given some use cases to test and provide feedback. Upon the project's completion, possible end-users could include search-and-rescue organisations attempting to locate missing people or media companies wanting to verify image location to eradicate fake news. These end-users would be less interested in the technical aspects of our software, and more in its usability and efficiency. This means the project's customer(s) would operate with a business mindset when providing advice during development. A team member has been assigned as the **Customer Contact (Lukasz Stanaszek)**; they are responsible for liaising and organising monthly meetings with the customer.

The final role which has been assigned within the team is the **Scrum Master (Dan Fredrickson)**, responsible for managing the incremental Scrum methodology utilised by this project, including increment and sprint design, planning and development. Their role aids team communication and collaboration, ensuring continuous development and, ultimately, helping the team produce high-value deliverables

to the customer in a timely manner. Thus, in this project, the Scrum Master is directly responsible for increasing agility, improving teamwork, and achieving the project's objectives within the Scrum framework.

Meeting Strategy

The team hold weekly group meetings to make design decisions for upcoming development phases and to discuss individual progress since the previous meeting. These regular meetings ensure substantial and consistent communication between team members whilst facilitating progress monitoring and preventing any development stage from falling behind schedule. Although it would be preferable to have more regular meetings, team members' time constraints and varying schedules made this infeasible without regular team absence.

The team also schedule regular meetings with the project supervisor to discuss any questions which arise and to receive guidance and advice regarding the project's development progress.

During these meetings, minutes are taken on Overleaf to document decisions for future use or in case of team absences in the meeting.

Expectations for Key Stakeholders

To ensure the project would meet the stakeholder's expectations, the group identified the **main factors which will determine the success of the project**. It was important to identify these key components at an early stage; they provide clear and distinct success criteria that will be available for the team to compare and evaluate project progress against at each phase. This process will prevent the team from restricting the project scope if unexpected time delays arise, thus holding the team accountable to the project's initial goals and purpose. Conversely, if the project were to exceed these initial aims, the completion of these factors could be used to highlight and justify the project's success.

There are several types of criteria which can be used to evaluate the success of the project, covering the completion of aims and objectives, the work plan, deliverables, and broader outcomes. Each of these success factors must be considered within the context of this project to identify the stakeholders' expectations for project success.

Firstly, the team has formed a **set of aims and objectives**. The **amount of these which are completed**, particularly the 'must have' requirements, will provide the stakeholders with a quantitative measure of success. It is in the stakeholders' interest to receive a multi-purpose tool with greater applicability. This can be achieved with more features/tools - the number of which can be definitively quantified, making it a good all-around measure.

Furthermore, a majority of the above objectives are technical in nature and introduce functionality into the end product. Each of these features inherently comprises a set of indicators that measure its performance. This can range from **feature effectiveness to speed and reliability**. For each feature, these tangible results can be evaluated against the stakeholder's expectations, allowing the team to monitor the project's success by **examining feature quality**.

For the team to be successful in these aspects, **adherence to the work plan** is crucial. The regular meetings will allow the group to monitor project progress against the devised schedule. The stake-

holders should expect some deviations from the designed work plan and should not view these as failures. However, the team's management of these time delays will be used as a success factor because they indicate the team's ability to adapt to unforeseen circumstances. To achieve this, the team plans to maintain a safe and collaborative working environment, based on mutual trust within the team.

The final success factor for the project will be the **fulfilment of broader outcomes**. As the stakeholders expect the project to have a positive effect on the wider community, success will be determined by the team's ability to produce a helpful product which can be used by both OSINT experts and individuals with less technical experience.

Overall, the degree to which these criteria are completed will provide a **quantitative measure of success concerning stakeholder's expectations**; this will be used for project evaluation at the end of the project.

Methodology

Scrumban[5], an Agile[6] Incremental[7] Test-Driven Development[8] methodology, has been adopted for this development-focused project.

The selected methodology is a mashup of existing methodologies. The most beneficial aspects of each have been adopted to complement the specific circumstances of the project. This can be demonstrated in this breakdown:

- **Incremental** - A general project plan with end goals is initially planned. Finer implementation details are deferred and incrementally developed in response to customer feedback, internal evaluation and new findings.
- **Scrum** - Once increments are planned, they are broken into more manageable and quick sprints focusing on delivering value.
- **Kanban** - Progress is tracked and managed using a Jira Kanban board, this gives team members autonomy and limits the time wasted from idling.
- **Agile/XP** - Implementation of a backup/safety system to prevent delays from team member absence - known as the Responsible-Review pair programming system (see below).
- **Test-Driven Development** - With limited OSINT experience, the team would start by tackling simpler geolocation cases and work up in difficulty. The test location cases guided the feature selection and thus maximised value delivery from each increment/sprint.

Methodology Justification

Incremental development introduces a time-efficient and value-focused approach to delivering a project of significant scale. Firstly, the project becomes **more manageable by decomposing the problem into smaller disconnected chunks**. This can be observed in the current project as the collection of potential features is considerable, with limited members and time the most valuable features had to be selected.

Secondly, the incremental and iterative approach allows developers to **respond quickly should any requirements and priorities be subject to sudden change** - such as in response to customer feedback. Should an increment be cancelled or require amendment, the time wasted planning the increment would be minimised leading to an overall reduction in redundant and wasteful development. This way, all **features will progress the project towards its defined objectives**.

Thirdly, deferring planning of increments will allow the team to **adapt plans according to lessons, experiences and findings** from previous increments. This is likely to occur due to high levels of uncertainty in a team new to OSINT. This nicely highlights that incremental planning and development allows team members to **gain required topic knowledge and make more informed decisions** by making them later.

[5] https://www.agilealliance.org/scrumban/
[6] https://www.agilealliance.org/agile101/
[7] https://www.agilealliance.org/glossary/incremental-development/
[8] https://www.agilealliance.org/glossary/tdd/

Fast-paced development is ideal when **clearly defined requirements** are available - such as this project. Scrum streamlines the development process by **reducing documentation** and dividing the work into small well-defined deliverables which can be focused independently. They can be continuously developed and integrated with direction and goal in mind. This makes it a **fast, safe and value-driven** development methodology. As a side benefit, team member **engagement and productivity should be boosted** by the sense of accomplishment of frequently concluding a sprint and delivering tangible results.

The above approach is complemented by structuring increments/sprints based on **goal/test case studies** - a form of test-driven development. The wide range of world case studies can be roughly separated into potential features which would solve the case studies. The ultimate goal is to produce a solution which can handle all the cases, however some are more difficult than others. For instance, geolocating an urban photograph with a landmark in the background is an easier task than a rural, mountainous region with a lack of obvious visual clues. The iterative development plan therefore involves **building a foundation of simpler case studies first**, and then **iterating towards the more complex examples**. This reiterates the focus on developing team members' OSINT topic knowledge to assist in the harder cases.

An important project circumstance that should be highlighted is the small development team size. This factor poses a danger to the project as losing a team member could be devastating. In response, the Agile concept of **Pair Programming** will be implemented in the form of a **Responsible-Review approach**. This will be detailed in the **Risk Analysis section** of this document and involves one team member carrying out the primary research and development for a given feature, with a second team member closely following the work in the case of any unforeseen absence or unavailability of the primary feature developer.

Code Management and Version Control

Given the Agile methodology planned for managing the project workload, including the potential for the requirements and priorities to change, it is possible that many different versions of the project code could exist at one time according to the changes required. Furthermore, the project will be developed by a team of developers, where each developer may be working on a different project feature. It would be inappropriate for all developers to work on the same version of the code-base, since changes made by one developer may affect those planned by another. **Code management and version control is therefore crucial for the success of the project** and will be accomplished using GitHub[9] and Git[10].

GitHub is a hosting website for online code repositories, allowing the entire project code base to be uploaded and saved to a remote server. This remote code-base can then be downloaded and replicated by each of the project's developers, whilst also **acting as a backup to prevent any loss of critical code in the event of local hardware failures**.
As part of the risk assessment, the group has also designed a safe and convenient backup plan, which includes pushes to the GitHub repository and merges where necessary. We have also decided on a medium-term backup option which will be performed on a monthly basis, where code and reports will be duplicated onto a personal hard drive.

Jira provides support to link with GitHub so that branches created for feature implementation can be linked with that specific feature on the Jira Kanban board. This improves development organisation and

[9] https://github.com
[10] https://git-scm.com

reduces the risk of any branches remaining in one Kanban state for too long without a team member realising.

Git is a version control system which tracks any changes made within files. Its usage facilitates the creation of code branches where multiple variations of the project code-base can exist and be edited simultaneously, by different people on different systems. This is critical for a project such as this one, where multiple developers will be working on different features, and where any new addition may risk breaking the main project application. Creating a branch for this purpose using Git acts as a 'safety net' so that isolated development can be carried out, and the new feature would only be merged into the main application once any compatibility issues or bugs are resolved. Methods are also provided for tracking the changes made to code over time, as well as for switching between and restoring older code versions.

User Stories and Work Item Management

User stories[11] are the smallest unit of work in an agile framework, and are end goals expressed from the perspective of the customer or end user. The use of user stories will **articulate how each work unit will deliver value to the customer** through its implementation into the project. Similar to the use of sprints, the successful implementation of each user story will create momentum within the development team, whilst driving creative solutions and keeping focus on the end user.

User Stories and other work items will be managed using a **Kanban**[12] **board** on Jira[13]. The application of Kanban alongside Scrum[14] allows for the flow of work to be visualised and improved, as well as more clearly assigned to various members of the team. The project Kanban board separates work units into the following categories depending on their status:

- **Backlog:** This is the list of work units that have a high likelihood of being worked on in the future. It is also the staging area where specs and requirements are fleshed out.
- **Design & Research:** This contains the work units which require User Interface designs and other pre-requisites before they can be developed and implemented.
- **To Do:** This section contains the prioritised work units which are ready for development. Too few tasks in this section means earlier sections should be swarmed to get more tasks ready.
- **In Progress:** This is the list of tasks that are being actively worked on. Each developer should have at least one item in this state.
- **Code Review:** This state contains the tasks which have been developed and are waiting for peer-review by another team member, meaning that each task is reviewed by at least 2 members of the development team before being tested. This section should be prioritised if it ever contains too many tasks.
- **Testing:** This section focuses on Quality Assurance for developed and peer-reviewed tasks, ensuring that they meet the requirements and will function as intended when pushed to the main build of the project.

[11] https://www.atlassian.com/agile/project-management/user-stories
[12] https://www.atlassian.com/agile/kanban
[13] https://www.atlassian.com/software/jira
[14] https://www.scrum.org/resources/what-scrum-module

- **Completed:** This contains the tasks which have been successfully shipped to production.

The project Kanban board was managed by the Scrum Master as they planned the tasks for each sprint. An important goal was to protect the team from over-committing and scope creep. A snippet of the Kanban Board can be seen in Figure 1.

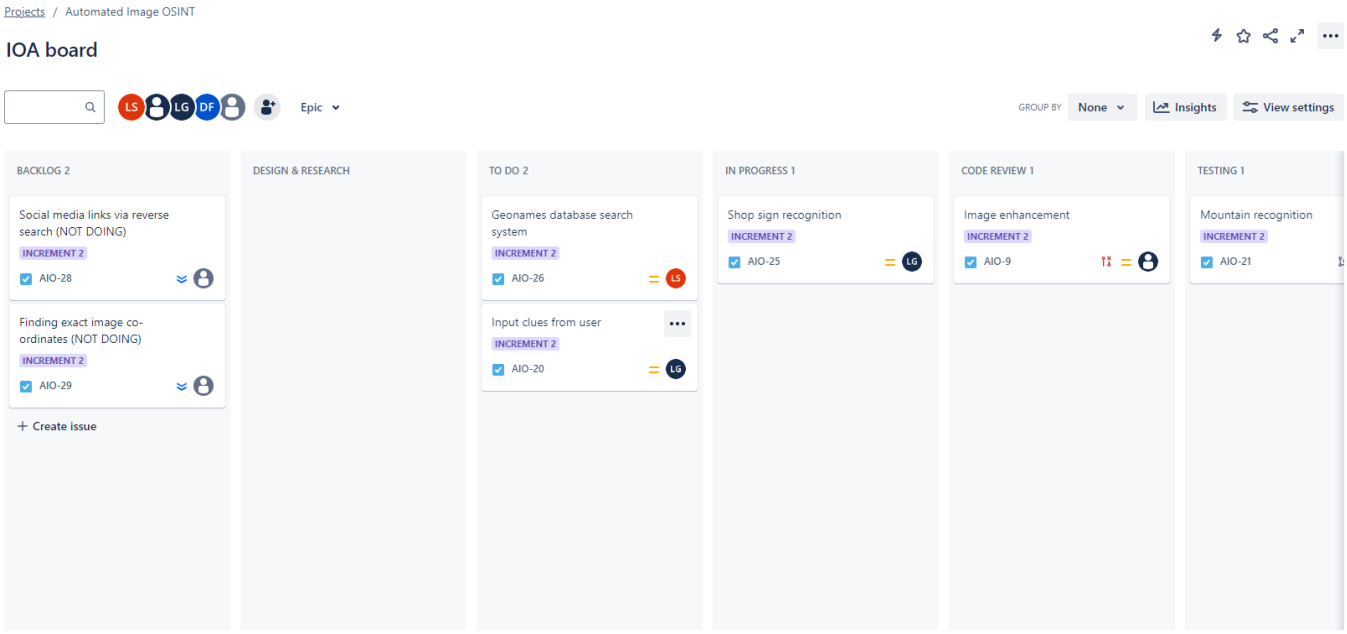


Figure 1: Snippet of JIRA Kanban board

Work Breakdown

The project has a very large potential scope and requires close management throughout development in order to prevent scope creep. The defined project scope, outlined during Requirements Analysis, can be neatly partitioned into sets of independent deliverables which can be developed incrementally. **Project feature increments**[15] **have been further grouped into sprints**[16] which run throughout project development. The final increment and sprint distribution can be observed below and in Figure 2.

Incr.1 **Initial Feature Design and Development, and Graphical User Interface**

- Spr.1 High Priority Feature Design and Development: Reverse Image Search, System Architecture Skeleton (stack) and ANPR
- Spr.2 Graphical User Interface
- Spr.3 Lower Priority Feature Design and Development: Image Authenticity Verification and Metadata Extraction

Incr.2 **Difficult to recognise cases - specific urban cases and mountainous region**

- Spr.1 Interface Improvements - including automation of features, Image Enhancement, Road and Shop sign recognition.
- Spr.2 Mountainous Region Geolocation, User clue input
- Spr.3 Confidence ranking system

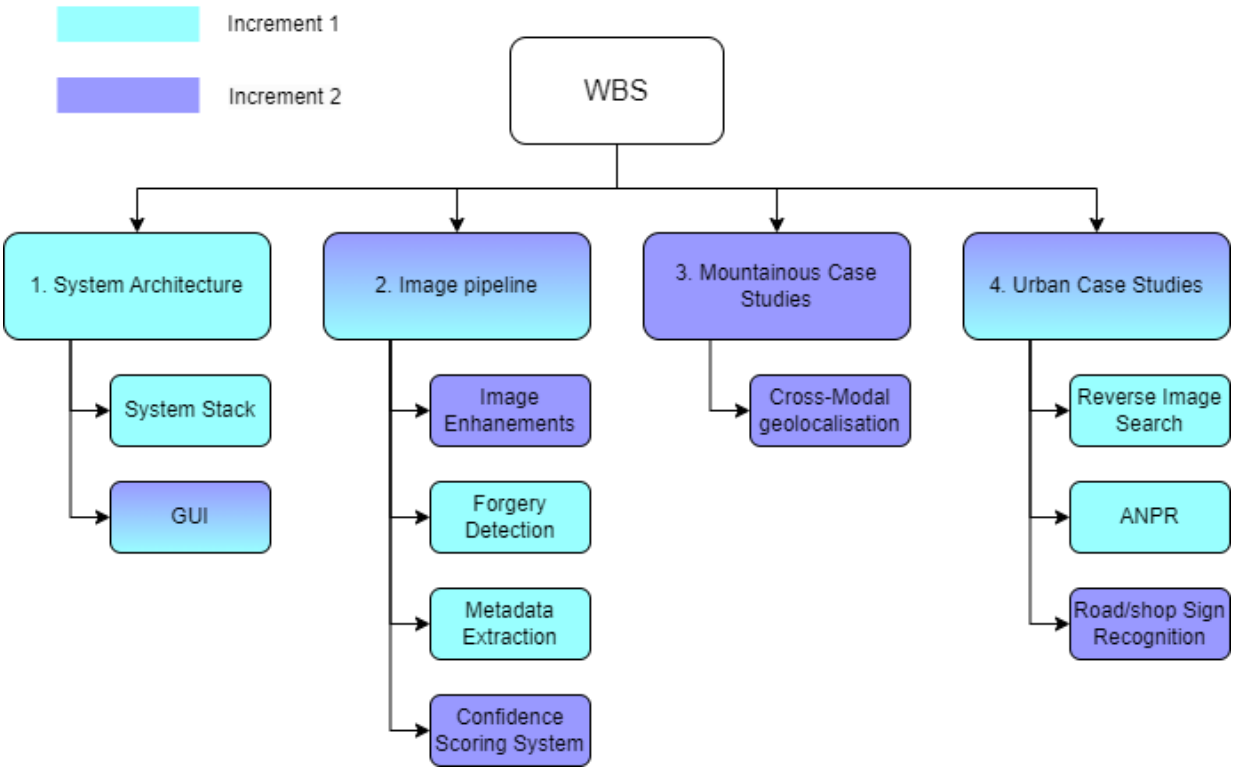


Figure 2: Project Work Breakdown Structure

[15] https://www.scrum.org/resources/what-is-an-increment
[16] https://www.atlassian.com/agile/scrum/sprints

Schedule

The final project schedule is depicted in Figure 3. A few differences can be observed compared to the specification and mid-project progress report schedules - these can be referenced in the appendix).

1. The AI Detection feature has been dropped as outlined in the finalised system requirements.
2. The confidence scoring System has been pushed back to the end of increment 2. This feature is equally as important however the team deemed it wiser to develop this feature once all its inputs (the other features) have been developed. This would prevent speculation on input type and shape and thus prevent integration difficulties.
3. Landmark recognition was merged with reverse image search. It quickly became apparent that reverse image search is capable of recognising famous landmarks so a separate feature was not required.
4. The UI was developed further in the second increment. This was added as a result of feedback from the supervisor and second marker during the progress presentation feedback.

Changes indicate that the project was flexible and that the team could respond to challenges in an effective and controlled manner. The cause of the changes ranged from unforeseen circumstances to delays and plan reconsideration, however, all reasons were justified. The purpose of an elastic scope and timeline is to prevent failure and hit the customer goals. This means the changes were acceptable and in certain cases desired.

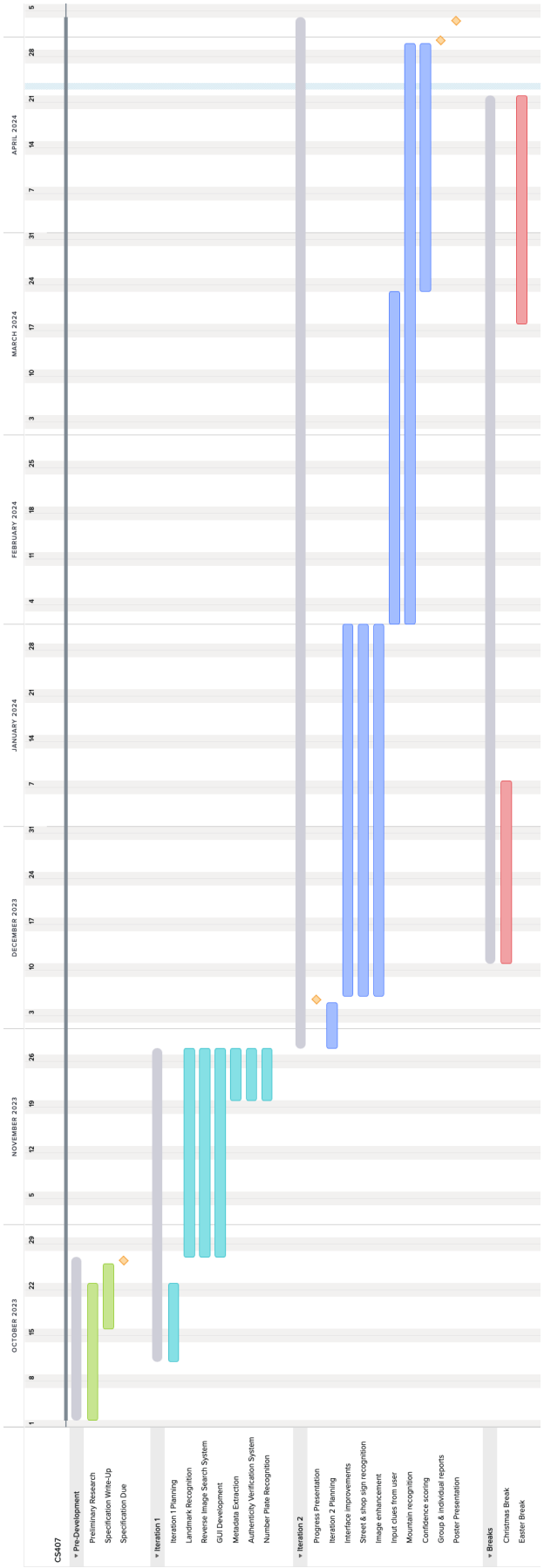


Figure 3: Final Project Schedule

Risk Assessment

Risk Management Plan

As part of the project, it was the group's responsibility to conduct a risk assessment to ensure the project's success. The approach employed is described below:

1. **Risk identification** - An Ishikawa diagram was used to identify potential risks. The standard 4 risk categories are employed.
2. **Risk ranking** - Using FMEA, each risk was ranked on likelihood and severity on a scale of 1-10, this produces an FMEA score between 1 and 100. Higher values represent higher risk.
3. **Mitigation** - A set of mitigations was implemented into the development methodology, management strategy or technology. The mitigations aimed to either minimise the likelihood or limit the impact of the risks.
4. **Updated risk rankings** - Following the same approach as with point 2), a new FMEA value was produced with the mitigations taken into consideration.
5. **Risk conclusions** - The results of all identified risks were used to determine whether the project is viable and can succeed. Once so, critical areas requiring extra care and focus were identified.

Risk identification

To identify risks, an Ishikawa diagram was employed. The standard four categories of "people", "policy", "process" and "plant" were included, as suitable for a software engineering project. The most common and applicable risks were covered, however, this risk list was not exhaustive. Very unlikely or negligible effect risks were not included as it was believed they whould have an insignificant effect on the project. This diagram can be observed in 4.

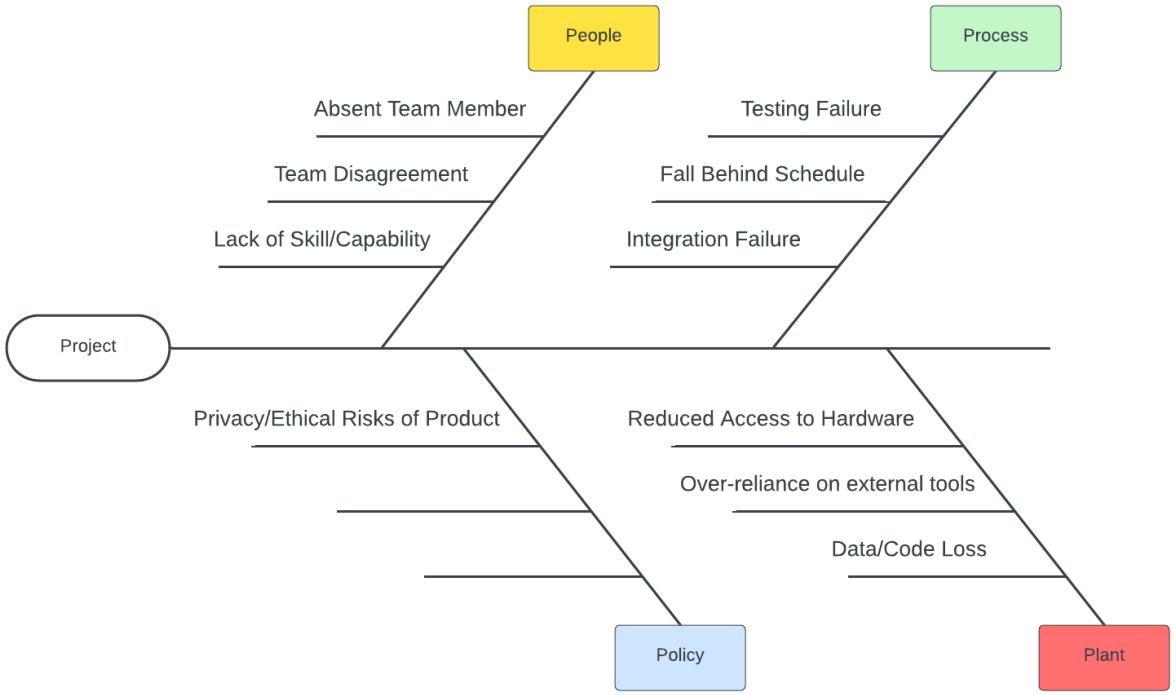


Figure 4: Ishikawa Diagram

Risk ranking and mitigations

FMEA was utilised to rank the risks as it is a methodical approach to comparing risks. Rather than abstractly comparing risks, numerical values between 1-10 were added for likelihood and severity (with higher values meaning higher likelihood and severity). By multiplying these values, a combined FMEA value for each risk is produced. Using agreed bounds also provides a definitive risk classification (high, medium, low, etc). This is a more formal, accurate and traceable approach which highlights which risks to focus mitigations on. Furthermore, justifications for choices can be made for each value. The FMEA table - with mitigations and revised FMEA scores - is displayed in Figures 5 and 6.

Risk	Effect	Likeli-hood	Severity	FMEA	Mitigation(s)	Revised Likeli-hood	Revised Severity	Revised FMEA
People								
Short term team member absence. For example - due to illness.	May lead to minor delays - mainly for the feature the member will be responsible for.	7	4	28	Develop a plan and schedule which accomodates for time delays. Adopted the responsible-review approach*.	7	2	14
Long term team member absence. For example - leaving the team permanently.	May lead to major delays and limited team capability to meet deadlines. Particularly severe for our team due to small size from the beginning.	4	8	32	Design a modular schedule and scope which can be easily altered depending on progress and deadlines. The incremental development methodology should complement the plan as it will minimise wasted time from redundant feature design and planning.	4	6	24
Team disagreement/conflict.	May cause project delays from resolving disagreements and reduce end product quality due to miscommunication. This includes incompatible features or contradicting functionality.	4	4	16	Get to know each other better by organising extracurricular activities. Conducted lengthy in-detail discussions on end product vision before even starting any design to eliminate future disagreement.	2	4	8
Lack of skills/capability.	Team members may not have enough experience/knowledge to complete certain tasks.	6	5	30	Identify team member strengths and weaknesses and exploit this knowledge to match members to best features. Pair programming (under the responsible-review approach*) should mean transfer of skills and reduce dependence on individual members.	4	4	16
Policy								
Pivacy/ethical risk of developing an intellignece gathering tool.	Certain sources may be unethical or even illegal. This could have repuational consquences for the university or legal implications.	6	6	36	Clear the project with supervisor and other university admin. Research safe OSINT guidelines. Compile a thorough list of whitelisted and blacklisted sources to use (i.e don't use dark web). Constantly monitor, consult and report progress, tools and sources with supervisor.	2	4	12

Figure 5: FMEA (part 1)

* Responsible-review approach: where each feature has two people involved in its development: a "responsible" person who carries out the main development and research for the feature and a “review" person which is someone who is closely following the feature, if the person responsible becomes unavailable, then the review member becomes responsible and should be well informed on the feature to carry on.

Risk	Effect	Likeli-hood	Severity	FMEA	Mitigation(s)	Revised Likeli-hood	Revised Severity	Revised FMEA
Process								
Testing failure - This includes missing bugs or passing features that do not match requirememnts.	Passing a faulty application or one that does not meet the success critieria means goals are not met and the product has limited utility.	4	7	28	Design a detailed test plan to test all features with different data types - aim is to identify bugs. Produce a list of various case studies (image scenarios) to test product utility and perfromance. Stay in close contact with customer/supervisor to check the product meets the specification.	2	7	14
Falling behind schedule	Final product may not include all planned feaures	5	4	20	Take a more cautious approach to selecting features to implement (when using MoSCoW have more "could" features). Be flexible to switch between features being worked on. Introduce buffer period in schedule to account for delays.	2	3	6
Integration Failure	Bringing all features together may take longer than expected or may cause unexpected effects (negative). For example APIs may be unclear or incompatible.	5	6	30	Implementation of an incremental development strategy means that features will be implemented gradually over time, so not leaving integration towards te end. Furthermore, the incremental methodology encourages design of the overall system and abstract feature connections, this should ensure compatability and clearly defined APIs.	3	5	15
Plant								
Reduced access to hardware resources (university provided) This may be caused by item busyness or setup delay.	Can lead to significant delays in project if team cannot reliably and flexibly access/utilise hardware recources like GPUs.	6	4	24	Minimal mitigations can be implemented, very dependend on department. Best course of action is to apply early.	5	4	20
Dependence on external tools	Project will likely make use of external tools and libraries, if they are no longer supported then product may no longer function. This will require time/effort to replace.	4 (in short term)	7	28	Make use of trusted, well documented and popular tools/libraries. Suggest a list of alternatives in the eventuality that a tool becomes unavailable.	4	5	20
Data/Code Loss	Very dangerous to project as may lead to full or partial loss of progress, which could cause extreme delays and require additional effort/cost to return to same point.	4	8	32	Established a secure data/code backup plan: This includes constant use of version control software (Git and GitHub), regular local backups on each persons device - with potential for cloud storage should we deem it necessary, and irregular HDD backups.	2	2	4

Figure 6: FMEA (part 2)

Risk Conclusions

After performing the risk analysis, introducing mitigations and updating the FMEA values, it was the group's definitive conclusion that **the project is viable**. The risks did not pose excessive threat which would lead to complete project failure. All mitigated risks have an FMEA value below 25. This is a **"low risk"** classification. Overall, the project should be **safe with the mitigations** in place.

However, some special considerations should be highlighted:

- Due to the team size being smaller than average, additional **effort must be placed on securing a reliable and backed-up development approach**. The highest updated FMEA score (24) was achieved by the long-term absence of a team member, this is because dropping to 3 members will severely limit the team's ability to proceed. Progress with 4 members must be reliable, and not dependent on one person, hence, the **introduction of the responsible-review system**.

  This is a **pair programming approach**, where two developers closely follow each feature, with a "main" responsible developer and a secondary backup ("review") developer. This will have negative implications on the development rate, however, it is deemed necessary to prevent a complete stop to progress.

- Reduced access to university hardware took joint second place for the highest updated FMEA place (20). Additional **care should be dedicated to securing the necessary CPUs/GPUs** to train machine learning models. Some features within the project are likely to be trained ML models or neural networks to achieve sufficient complexity and accuracy - making powerful GPUs necessary. The project cannot proceed without them. Access to these can however be inflexible, especially closer to deadlines, so care must be placed on when the GPUs may be required. The plan/schedule could reflect this and make use of them early on.

- The third risk of interest - with a joint second-highest updated FMEA value - is the over-dependence on external tools. The fourth-year project is relatively short. This means that the chance of a tool becoming unavailable is less likely (though not impossible) meaning that in the short-term the likelihood of the risk is low, however for a long-term project this would require much more consideration. Nonetheless, it is deemed wise to **find alternatives to each tool** used in case such a situation arises.

Additional steps

To further secure the project against risk, it was valuable to create a RACI matrix to establish responsibility and accountability. A general RACI matrix could be designed early in the project, the responsibilities of the member were determined based on their selected team roles. Below, in Figure 7, is the general RACI matrix which shows the general responsibilities.

Accountability would be more critical when distributing work. This needed to be done on a "per feature" basis. However, as mentioned in the methodology section, the exact design and selection of features would be performed in an agile manner - at the start of an increment. This unfortunately meant no feature RACI could be produced at the start of the project. Once the features were selected, and allocated to an increment/sprint, A RACI matrix was produced. The matrices for each feature can be observed below in Figure 8. The "consulted" person is the backup "review" developer in our responsible-review pair programming approach:

Responsible	Accountable	Consulted	Informed

	Matthew	Dan	Lara	Lukasz
Progress Tracking (Trello Management)	Consulted	Consulted	Consulted	Responsible Accountable
Communications with Supervisor	Responsible Accountable	Consulted	Consulted	Consulted
Communications with Customer	Consulted	Consulted	Consulted	Responsible Accountable
Documents Management	Responsible Accountable	Consulted	Consulted	Consulted
Meetings Coordination	Consulted	Consulted	Responsible Accountable	Consulted
Scrum Management	Consulted	Responsible Accountable	Consulted	Consulted

Figure 7: General RACI Matrix

Responsible	Accountable	Consulted	Informed

	Matthew	Dan	Lara	Lukasz
Increment 1				
Program Stack	Consulted	Responsible Accountable	Informed	Informed
GUI	Consulted	Responsible Accountable	Informed	Informed
Reverse Image Search	Informed	Consulted	Informed	Responsible Accountable
ANPR	Responsible Accountable	Informed	Consulted	Informed
Metadata Extraction	Informed	Consulted	Responsible Accountable	Consulted
Authenticity Verification	Responsible Accountable	Informed	Informed	Consulted
Increment 2				
Interface Improvements	Consulted	Responsible Accountable	Informed	Informed
Road Sign Recognition	Responsible Accountable	Informed	Consulted	Informed
Shop Sign Recognition	Consulted	Informed	Responsible Accountable	Consulted
Mountainous Geolocation	Informed	Informed	Informed	Responsible Accountable
Image Enhancements	Responsible Accountable	Consulted	Informed	Consulted
Input Clues from User	Informed	Consulted	Responsible Accountable	Informed
Confidence Scoring System	Responsible Accountable	Informed	Informed	Consulted

Responsible = Responsible / Accountable Review = Consulted

Figure 8: Increments RACI Matrix

Ethical and Legal Considerations

Legally, there are no such considerations for this project. Although the project utilises external APIs and libraries, the platform does not violate any copyright and licensing laws because these tools are in the public domain or are open-source resources.

GDPR concerns need not be raised for the application either since the application's database which contains user-uploaded images is stored locally, within the application itself, on the user's device. The platform's 'About' section informs users that the use of third-party APIs for image analysis necessitates the sharing of image data with these parties and that the user's continued use of the application constitutes their agreement. Furthermore, it is assumed that images containing identifiable information will not be uploaded since such images would not be of interest for automated OSINT analysis.

This platform does have some social, ethical and professional issues which must be considered. The system makes automated OSINT more accessible to the wider public. In some aspects, this is beneficial and several companies have already proven this. One such organisation is Trace Labs which claims that they are 'crowdsourcing OSINT to find missing persons' [17]. By providing a platform to connect their 11,777 global members, they have assisted 380 cases through their coordinated OSINT investigations, working alongside law enforcement agencies. However, the wider accessibility provided by the platform also introduces ethical concerns, and our product could be used for malicious purposes such as stalking. These concerns are exemplified by a 2013 case study[7] detailing the public's attempt to identify the Boston bomber. Here, the 'spread of OSINT, out of the government office to businesses, and casual users'[8] resulted in Reddit users conducting 'online witch hunts' and falsely accusing and naming suspects. Subsequently, this caused detrimental effects on these named people, highlighting our recognition that the misuse of automated OSINT tools such as this platform may result in serious social and ethical issues.

[17] https://www.tracelabs.org/

Tools and Resources

A brief overview of the tools used throughout development. Usage of tools and resources are discussed in-depth in the Project Management, and Feature Implementation sections of this report.

Project Management Tools

- **Git/GitHub**[18]**:** version control tool to house online code repositories.
- **Jira**[19]**:** agile project management tool with Kanban board and GitHub integration.
- **LaTeX/Overleaf**[20]**, Microsoft Office**[21]**:** tools for collaboratively producing and maintaining reports and documentation.
- **Lucid Chart**[22]**, Draw.io**[23]**, TeamGantt**[24]**:** free online diagram creation tools.
- **Microsoft Teams**[25]**:** online communications channel with video calls and file sharing.

Hardware Resources

- **GPUs/CPUs:** provided by The University of Warwick for batch processing on a variety of CPUs and GPUs. Used to train machine learning models. [26]
- **Personal hard drive:** offline data and code backups.
- **Phone camera:** generating test examples.

Data Resources

- **Google Street View**[27]**:** geotagged outdoor images used for testing.
- **HuggingFace**[28]**:** machine learning model and dataset sharing platforms used to import pre-trained models and perform model training.
- **Wikimedia Commons**[29]**:** online repository of free-use images, used for some testing.

[18] https://github.com/
[19] https://www.atlassian.com/software/jira
[20] https://www.overleaf.com/
[21] https://www.microsoft.com/en-gb/microsoft-365
[22] https://www.lucidchart.com/pages/
[23] https://app.diagrams.net/
[24] https://app.teamgantt.com
[25] https://www.microsoft.com/en-gb/microsoft-teams/group-chat-software
[26] https://warwick.ac.uk/fac/sci/dcs/intranet/user_guide/batch_compute
[27] https://www.google.com/streetview/
[28] https://huggingface.co/
[29] https://commons.wikimedia.org/

System Technologies

- **Electron:** desktop application packaging and local server hosting.
- **Django:** web-application host.
- **SQLite3:** database management.
- **Jinja:** HTML templating.
- **Bootstrap:** pre-built CSS components and page layout.
- **FontAwesome:** pre-built CSS icons.
- **jQuery:** HTML DOM access simplification.
- **AJAX:** asynchronous JavaScript requests and CSS alteration.

Software Libraries and APIs

Python Libraries

- **Numpy, Pillow, MatPlotLib, ImUtils, Pandas:** basic data storage and manipulation.
- **OpenCV, Scikit-Image, Scikit-Learn:** computer vision, data processing and image processing.
- **PyTesseract:** implementation of Tesseract OCR engine.
- **PyTorch:** neural network implementation.
- **GeoPy:** geolocation tools.
- **ExifRead, PiExif:** metadata extraction.
- **OverPy:** OpenStreetMap queries.
- **SpaCy:** natural language processing.
- **Django packages:** web app development framework.

APIs

- **Azure Visual Search:** reverse image search.
- **Tesseract OCR:** optical character recognition.
- **Google Cloud Vision:** label extraction, text detection and optical character recognition.
- **GeoNames:** geographical place name database.
- **Overpass:** OpenStreetMap queries.
- **OpenLayers:** integrated interactive map.

Research and Literature Review

Application Framework

The project application framework was decided to either consist of a web application, or a desktop application.
TechTarget defines a web application as an "application program that is stored on a remote server and delivered over the internet through a browser interface"[9]. The use of a web app would bring a number of benefits, such as:

- **Accessibility** - Web apps can be accessed from all web browsers so users would be able to access their OSINT analysis from any of their devices.
- **Efficient development** - The same version works across all modern browsers and devices so there is no need for several different software iterations for multiple platforms.
- **User simplicity** - They don't require a download so are easy to access whilst having no need for end-users to perform software updates or manage their hard drive capacity.
- **Scalability** - The vast majority of web application data is stored in the cloud, meaning no need to invest in additional physical storage capacity to run web apps.

There were a number of issues to consider with a web application, however:

- **Storage** - A large capacity of cloud storage would be required to store the images uploaded by each user for analysis.
- **Reduced speed** - Image analysis would have to be performed on the web-app server itself. Depending on the number of users accessing the web-app and performing analysis, this could dramatically impact performance and even crash the application.
- **Security** - Web applications are available for anybody in the world to access which presents a greater security risk than a native application. This becomes a larger problem when we consider that login information would be stored, which may not be necessary for a native application.

For this project, a combination of web-app and native application features would be deemed most appropriate. A hybrid approach was therefore adopted, where a web-app would be packaged into a desktop application for installation on the user's device.
This would eliminate the potential storage, speed, and security issues presented by a web application. Uploaded images would be stored within the application on the user's device; the computational power of the user's device would be used for hosting the application and for running analysis scripts; and the application server would be locally hosted and at a minimal risk of security attacks.

Two frameworks were investigated to accomplish this: **Electron** and **Tauri**.
Electron is a framework produced by the OpenJS Foundation to create desktop applications using web technologies (HTML, CSS, JS), rendered via a version of the Chromium browser and a Node.js runtime environent backend, as shown in Figure 9 below. Popular applications which make use of Electron include Visual Studio Code, GitHub Desktop, and the Atom text editor. It was released in July 2013 and supports Windows, MacOS, and Linux operating systems. Owing to its decade of use, Electron has extensive documentation and tutorial material online catered to a vast array of desired application functionalities.

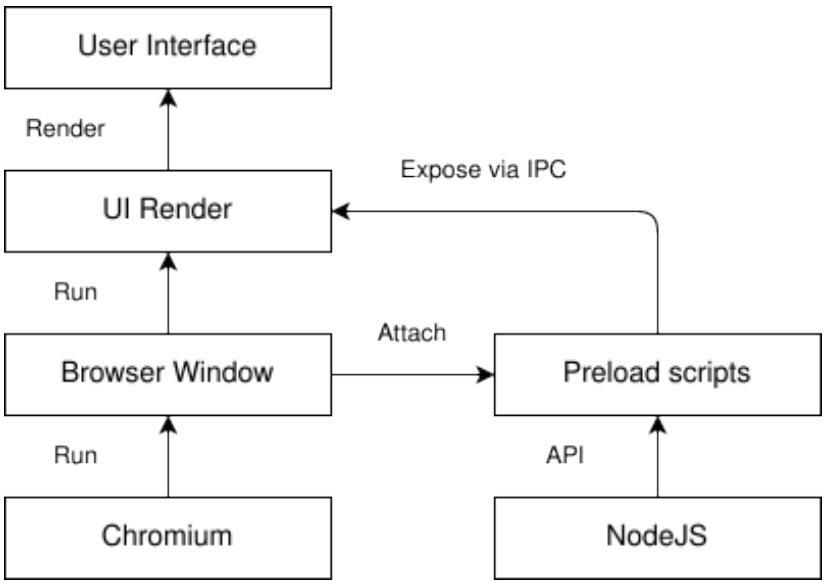


Figure 9: Diagram of Electron's component interaction and functionality.

Tauri aims to accomplish the same ends as Electron whilst being a more lightweight alternative. It makes use of the system WebView to display application content and has a Rust backend, shown below in Figure 10. Tauri's primary advantage over Electron is that its app installer produces a far smaller bundle than its rival - 34x smaller in Levminer's real-world application[10] - and uses less RAM whilst running. However, released in 2020, Tauri is not nearly as popularly used as Electron with few online tutorials which extend beyond Tauri's basic setup.

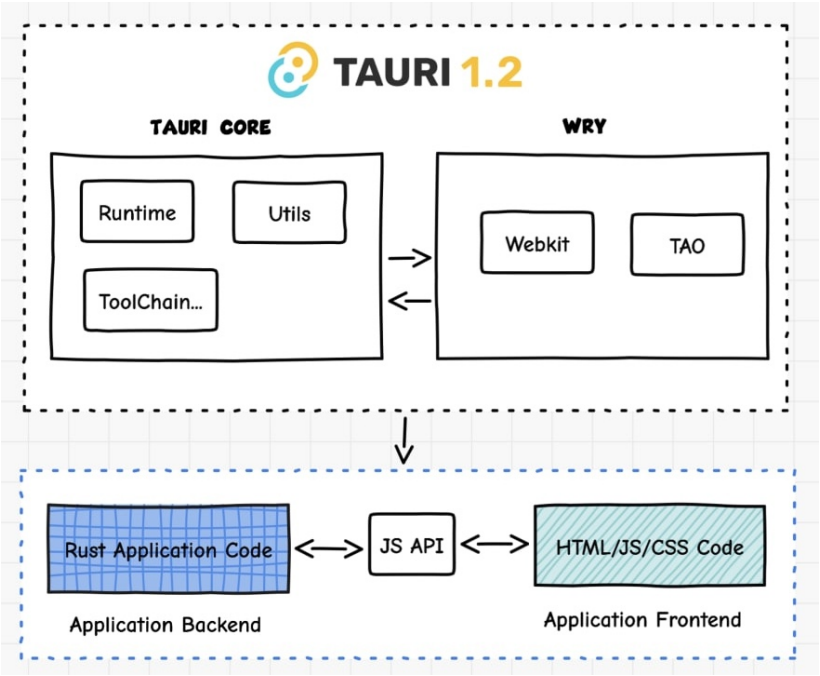


Figure 10: Diagram of Tauri's component interaction and functionality.

It was for this reason that **Electron** was chosen as the primary application framework. Extensive documentation and numerous online examples were both critical for the development of the system in a professional and timely manner, as well as the ability to write backend code in a programming language

which was familiar to all team members (Javascript). It would be very time consuming and inefficient to have all team members learn Rust, and having one person learn Rust for backend production would dramatically increase project risk should that one member become unavailable for future work.

Backend Technologies

The necessity for the user's images to be stored on their device within the application itself, as detailed above, meant that a backend framework had to be found which would interface well with Electron and a local database. Two options presented themselves: **Django** and **Flask**.

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design with a 'batteries included' approach. Its primary goal is to make the creation of complex and data-driven websites more simple through the employment of reusable components, less code, and the principle of "Don't Repeat Yourself". Its primary structure used to serve HTTP requests is laid out in Figure 11.

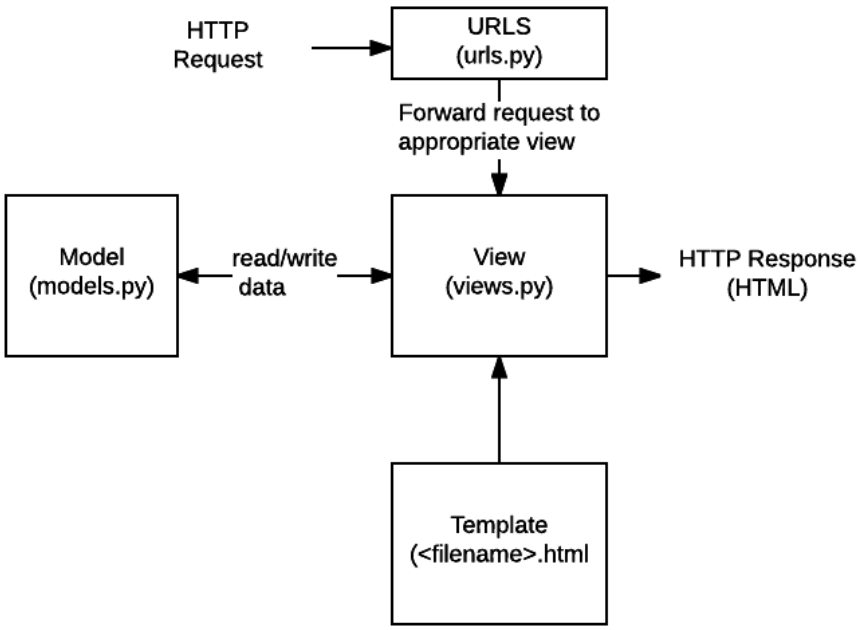


Figure 11: Diagram of Django's frontend and backend functionality with Object Relational Mapping for database models.

On the other hand, **Flask** is described as a micro web framework which does not require any tools or libraries. Unlike Django, it does not come with a database abstraction layer or form validation functionality. It is best suited for small and relatively simple web applications and gives the developer a greater degree of control over site functionality, and is illustrated in Figure 12.

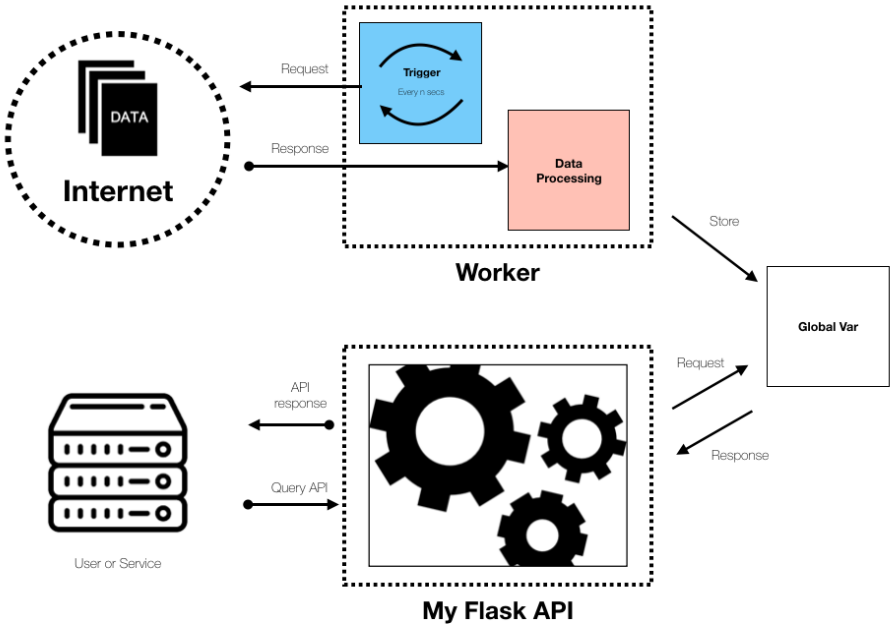


Figure 12: Diagram of Flask's functionality with outsourcing to extensions and libraries to provide back-end models.

Django was deemed the most suitable framework for the project given its built-in support for interfacing with database systems and more advanced support for dynamically-loaded HTML pages which would be required for the displaying of analysis later.

Frontend Technologies

React is a JavaScript library which enables the building of the user interface based on components. The key advantage of using React for this platform would be that, when the page changes, only the parts which have changed are re-rendered. Unchanged DOM elements are left alone, resulting in a more responsive user experience.

AJAX is a set of JavaScript technologies which enables asynchronous updates to web pages in a similar way, but requires more code and is not as easy to use as React. Its installation is far simpler, however, especially for established projects.

When compared with vanilla JS, React provides considerable benefits for platforms requiring a great amount of dynamic page content. However, its installation is complicated when combined with both Electron and Django since each requires its own configuration, and application entry-points must be combined in a non-trivial manner. In the case of this project, the only asynchronous updates to perform were considered to be those from the analysis buttons, which could be accomplished using AJAX, so React was deemed an over-complication whilst AJAX was used instead.

The component-based UI which would be available through React was supplemented with the use of the **Bootstrap CSS framework** which provides a number of pre-built features such as file upload interfaces and buttons, as well as a pre-configured grid-based layout shown in Figure 13. This allowed the

construction of the application interface without having handle any complicated CSS for configuring page layout.

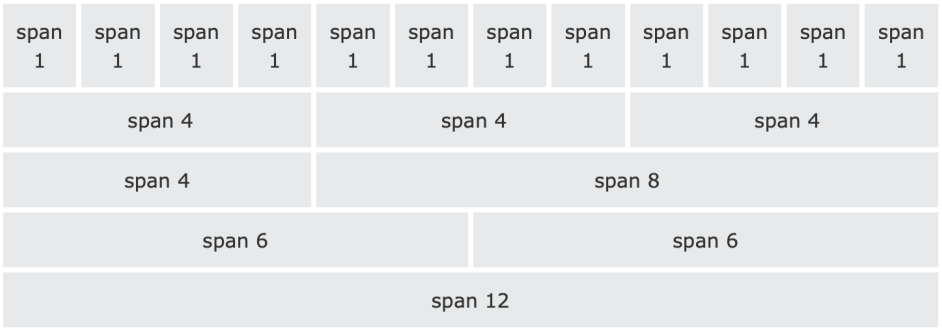


Figure 13: Illustration of the Bootstrap grid system, showing combinations of columns widths in various rows.

Metadata Extraction

'Metadata is data that describes data'[11] which, if present, can be crucial for digital investigation processes like 'image retrieval, content description and geolocation'[12]. This is because image metadata provides useful, hidden information like when the image was taken, which camera model captured it and, most helpful to this project, the location of the image.

Metadata Structure

Within images, this metadata can be broadly categorised into 'technical, descriptive, and administrative'[11]. Firstly, descriptive data is additional metadata produced by the camera software; these tags would include information like the camera model which can then be used to 'potentially verify who took a picture'[11]. However, as this verification is beyond the scope of this project, this descriptive data will not be utilised extensively. Another type of descriptive data is the creation date and time which can be used to help narrow down the geolocation process. By visually comparing light levels in the image with the time of day, certain global regions can be eliminated due to incorrect time zones. Similarly, if the image contains any seasonal weather indicators, the date metadata tag can be used to remove several countries. For example, if the creation date is January and the image has snow present, the image was most likely taken in the northern hemisphere as these countries are in winter. Next, administrative data contains information regarding file ownership and permissions; it is unlikely that this data will be useful for image geolocation.

Finally, technical data covers 'information about the camera that took the photo and the settings it used'[11], including the camera's GPS data which will be the most indicative metadata tag for geolocation. Images with GPS data present will provide the easiest cases for geolocation as they can be used to immediately calculate an image's exact location, improving the geolocation process's efficiency. Furthermore, as most modern photographing software 'is built in with metadata'[13], there is a wider availability of images containing GPS data, especially considering that most people use their mobile phones, which include more and more accurate GPS sensors, as their primary cameras.[11] Hence, the utility of GPS metadata extraction for geolocation is continually increasing as it is becoming much more likely for images to have these GPS tags present.

Aside from the different types of metadata, there are also several formats used to store image metadata including 'Exchangeable Image File Format (EXIF), International Press Telecommunications Council's Information Interchange Model (IPTC- IIM) and Adobe Extensible Metadata Platform (XMP)'[14].

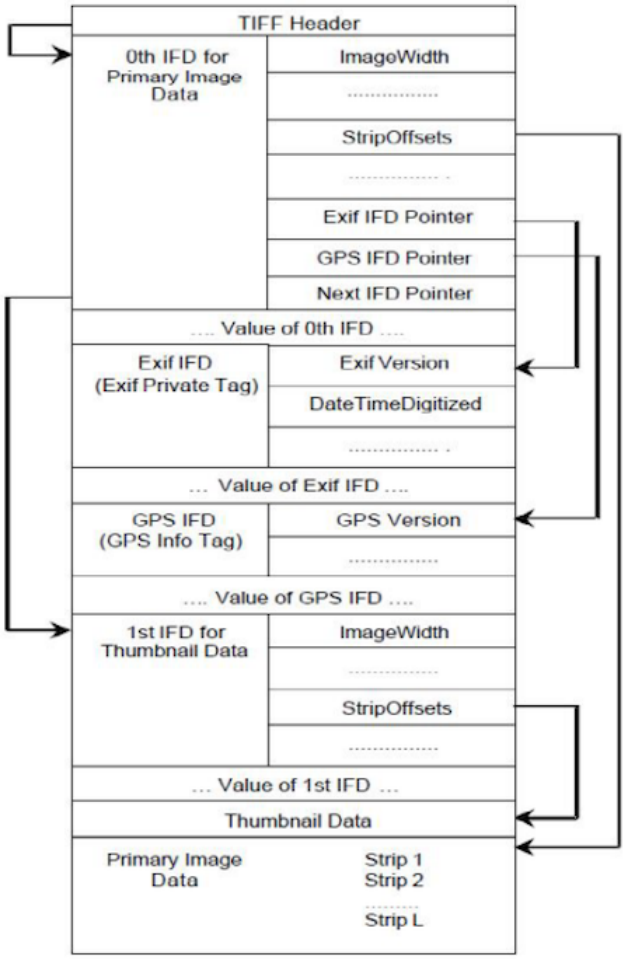


Figure 14: Structure of EXIF metadata[15].

EXIF metadata is one of the most popular format types as most phones and cameras 'automatically add relevant EXIF metadata in the process of image acquisition.'[13]. Notably, the GPS data is a type of EXIF metadata, meaning that EXIF metadata will be the most important format type for satisfying this project's geolocation aim.[15] However, EXIF data is relatively easy to alter as there is a wide variety of freely available software packages[12]. This must be considered during the development process of this project.

IPTC-IIM, created in 1991, is a format used for 'transmitting news text documents and photos'[14]. It contains IPTC headers which are inserted by software like Adobe Photoshop into many photo files. Since the introduction of XMP, the IPTC standards have defined XMP as 'the successor to the IIM-based IPTC header'[14]. XMP, 'serialized in XML and stored using a subset of the W3C Resource Description Framework (RDF)' is a metadata model which provides a standardised method for processing extensible metadata, allowing users to 'embed metadata into a file itself'[14]. One benefit of XMP is that it allows each software program to amend its metadata information to the final digital file. The combination of XMP and IPTC-IIM can be used to store textual location information such as the street name[12].

Ideally, an image would only have a given metadata property defined in one metadata format, like how certain properties are unique to the EXIF container. However, due to the evolution of standards causing certain applications to become outdated, there can be implementation inconsistencies across different software tools and varying size limitations on properties, creating challenges for metadata extraction which must be considered when designing a universal extraction tool. Across these three metadata formats, only the 'Copyright', 'Description', 'Creator' and 'Date/Time' tags are available to all. For images where multiple formats are present, EXIF metadata will be the preferred format in this project as the current standards for metadata extraction prioritise EXIF over the others[14].

The structure used for storing metadata varies between image types. PNG and JPEG are two of the most common image formats online. JPEG files have 3 main sections which are all written according to the EXIF specification: EXIF APP1 for reading EXIF; XMP APP1 for reading XMP and Photoshop APP13 for reading IPTC-IIM.[14] By contrast, PNG files separate their metadata storage in chunks where XMP is often 'stored as a iTXt/uncompressed or zTXt/compressed'[16] chunk and EXIF is stored in the eXIf chunk. As JPEGs are the 'default format for most cameras'[11] they will be the primary focus of this metadata extraction tool.

Challenges

When extracting metadata, there are some major challenges and considerations which must be accounted for. It is essential that the resource used to perform metadata extraction does not manipulate the image and considers 'possible departures from the EXIF specification'[13] to preserve accuracy. For instance, image resizing could remove certain stored metadata, thus causing important information to be omitted from the extraction process and not utilised. Moreover, not all images have relevant metadata still attached. For example, Twitter, like many other photo-sharing platforms, 'removes any metadata the image has'[11] for user-security reasons. This limits the range of images suitable for metadata extraction, especially as the use of these platforms grows, and thus the overall utility of this tool for geolocation. Arguably, the most important consideration regards the validity of the present metadata. Advancements in metadata manipulation tools have made it easier to alter this data and override with fake metadata which can be very difficult to trace and detect afterwards.[12]

Thus, within the context of this project, fake GPS data would cause drastic implications to the accuracy and reliability of the geolocation results from metadata. There are some existing methods which can be used to provide some authentication including Kakar and Sudha's shadow detection process[12] for creation time authentication, analysing digital signatures and metadata hash values, and creating machine learning models to detect anomalous metadata patterns which indicate forgery. However, these processes are extensive to implement due to the complex nature of the required training dataset and are beyond the agreed-upon scope of this project. Therefore, to mitigate this forgery risk, if the geolocation result from metadata extraction is significantly different to those determined from the other features, it will be assumed that the image contains fake metadata.

Existing GPS Metadata Extraction Methods

The Acharya et al. paper 'Geo-Locating an Image Using EXIF Data'[15] created a Python-based metadata extraction system to read the input image's GPS data to show the exact location on a map using Google Maps API. First, Acharya et al. focus on EXIF extraction. They argue that using the Pillow library allows them to efficiently load images in different file formats, whilst providing the 'capability to handle geo-tagged images'[15]. Moreover, they detail how the Python 'exif' module provides 'effortless retrieval of

metadata'[15] as they simply use the open(), read() and process_file() functions to create a dictionary of EXIF tags and values. Then, they can quickly 'access specific tag values by their tag names'[15]. To retrieve the GPS metadata from this dictionary, they use the 'GPS GPS Latitude' and 'GPS GPS Longitude' tags which return the coordinates in degrees, minutes and seconds form. These are then converted into decimal degrees format so that they will be compatible with the Google Maps API.

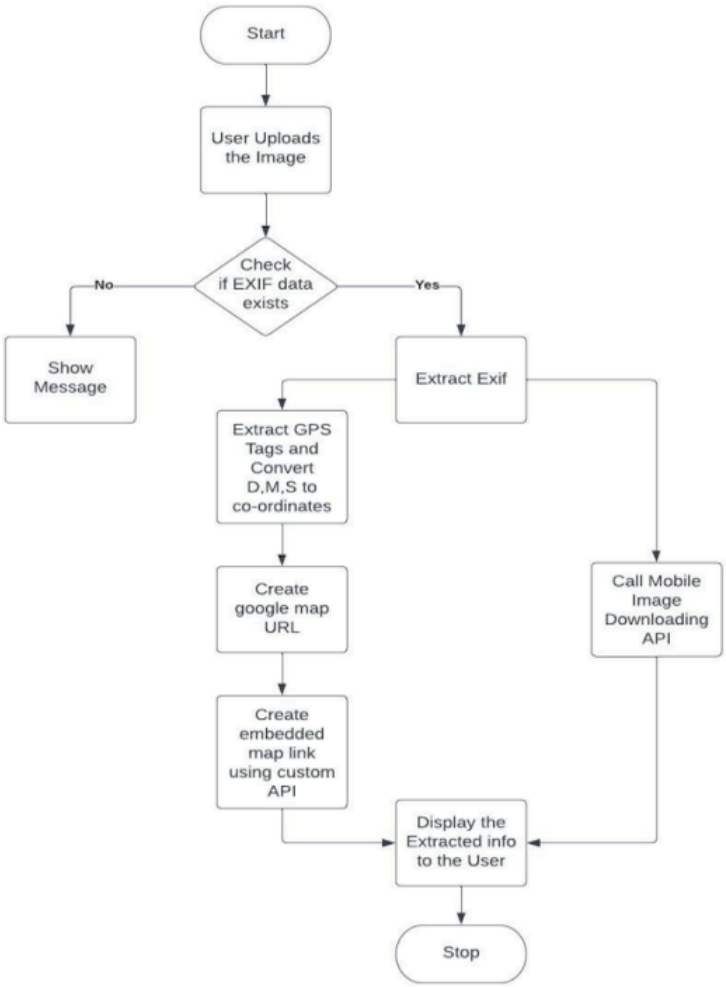


Figure 15: Architecture of the proposed metadata extraction system[15].

Although Acharya et al. provide an efficient automated process from metadata extraction to location visualisation, they do not provide solutions for images with partial or no GPS data present. It would have been better if they chose to analyse other available metadata to attempt to infer location clues to provide the user with a rough idea.

By contrast, Riggs et al. propose the paper 'Image Mapping through Metadata'[11] in which they have developed a metadata extraction program using Java instead of Python and then, similarly to Acharya et al., they have used Google Maps API to visualise the calculated image locations. They have a Java back end which reads and filters multiple files in a user's folder and extracts metadata for all of the found images using the 'metadata-extractor' library. Here, they retrieved the geolocation information but also generated a string of all found metadata, unlike Acharya et al.. For the front end, they have combined JavaFX and HTML5 to produce an interactive map graphical user interface that simultaneously displays the location of the images and all their extracted metadata. Their solution

boasts competitive computation times by performing metadata extraction in parallel; on average it would take '21.79 seconds to completely ... extract the metadata ... and load a maps interface'[11] for 100 input images.

Another successful solution was presented by Toevs's 'Processing of Metadata on Multimedia using ExifTool'[17] paper. Toevs focuses primarily on using the ExifTool command line application to extract GPS coordinates, but then also extends this to include the 'BeautifulSoup' Python library, similar to Acharya et al., for metadata extraction from images sourced from websites. They thoroughly justify their use of the ExifTool application by detailing how it 'contains hundreds of command modifiers'[17] for metadata manipulation and providing a robust metadata manipulation method by utilising ExifTool's flexibility with various file types. However, their solution focuses mainly on command-line operations and batch-processing techniques which, although ideal for larger-scale metadata analysis, the user input and technical expertise required for effective use limits their method's utility for this project's metadata extraction feature.

Moreover, Toevs extends this solution, using 'BeautifulSoup' to search 'http and https webpages for content' and then download the images covering a variety of media types including 'cgi, gif, jpeg, mp3...' [17]. The desired file types can be specified beforehand to provide a versatile program which can effectively be modified to accommodate the requirements of several different research investigations. Even with this expansion to their other command line program, there are still some drawbacks to their approach as the program downloads all the images found on the desired webpage onto the user's computer, which could cause storage issues. Moreover, the program still requires the user to input a command to specify which tags to extract. Although this does provide flexibility for the user, for this project, this hinders this paper's utility as this project aims to create an automated system.

Overall, considering all of these approaches, Toevs's research on the ExifTool provides a much more extensive and versatile metadata extraction tool for all image types compared with the more specialised Python and Java implementations presented by Acharya et al. and Riggs et al.. However, for this project, the improved automation and user-accessible solutions of the latter programs outweigh these ExifTool benefits. Furthermore, between Acharya et al. and Riggs et al.'s programs, there is minimal difference in their suitability for this project as both were very effective, but, as Toevs also chose to use Python when they extended their program, it can be justified that a Python metadata extraction tool will be suitable for this project as there are multiple studied approaches and libraries to choose between during implementation. To improve on Acharya et al.'s program, the wider range of metadata collected by Riggs et al. will be mimicked in this project's solution, as this will allow indirect geolocation clues to be gathered and thus aid the overall geolocation process.

Image Enhancement Techniques

Basic image enhancement techniques such as sharpening and blurring can be used to exaggerate critical details of an input image and filter out unwanted noise. It has been shown that computer vision models (such as convolutional neural networks) sometimes perform better at classification tasks when sharpening and brightness adjustment preprocessing steps are applied before training and testing[18].

Image sharpening filters typically first apply a filter to remove fine details from the image (such as a blur), leaving only the coarse details. These coarse details are then partially subtracted from the original image to emphasise the fine details in the image[19].

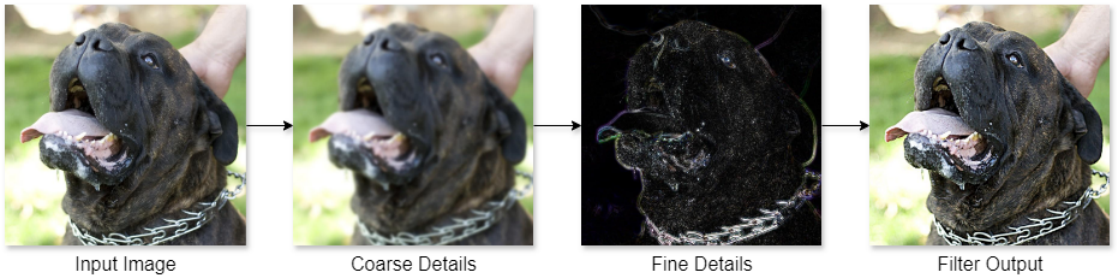


Figure 16: Process of applying a sharpening mask[19].

Sharpening filters can therefore usually be specified by a 2D kernel which is applied as a convolution over an input image to produce a sharpened version. Furthermore, a blur can be applied by replacing the kernel's values with their reciprocals[20].

A linear sharpening filter uses a linear kernel to amplify fine details. This approach has a very low computational cost but can sometimes lead to overly noisy results. By comparison, an unsharp mask filter applies a Gaussian blur to the input image to extract the coarse details. This tends to produce a more subtle and subjectively more visually appealing effect, although it can still lead to excessive noise and distortion of edges in some cases.

Another sharpening method is bilateral sharpening. It makes use of a bilateral filter to extract coarse details while preserving edges. This approach can be used to enhance the visibility of edges in an image while minimising the amount of noise added[21].

Image Splicing Detection

In addition to verifying the geographical location of an image, we can verify its authenticity by checking whether it has been tampered with. Forgery in JPEG images can be achieved using image editing software such as Adobe Photoshop by copying a region from one image onto the target image[22].

Splicing in JPEG images can be detected using a double-compression method. The JPEG compressed input image is re-compressed at varying levels of JPEG compression and saved, then the difference is taken between the input image and each of the re-compressed versions. If the re-compressed version was compressed at or close to the same quality as the original compression, the difference image will be close to zero everywhere (because the results of the two compression runs will look the same). If a region in the input image was compressed at a different compression quality q to the rest (due to image splicing), this region only will be zero in the difference image of the re-compressed version compressed at quality q[23]. This is called a JPEG 'ghost'. Therefore any ghost regions seen in the difference images indicate that the input image was spliced.

JPEG ghosts can generally be detected visually by an expert, but the structural similarity measure SSIM (or 'energy') can also be used for automatic detection. We can simply take the SSIM value pairwise between each of the calculated difference images: if any SSIM value falls below some threshold, then a JPEG ghost is likely to exist in the series of difference images.

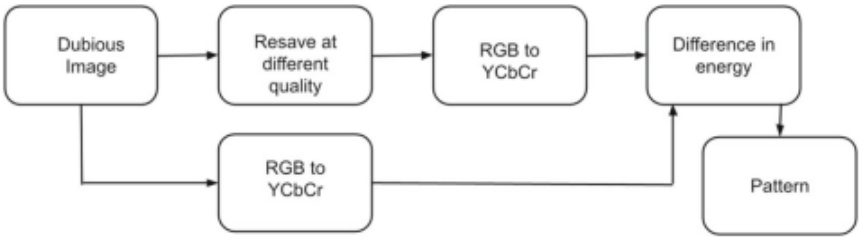


Figure 17: Overview of the JPEG ghost detection method for splicing detection[23].

Morphological Processes

Morphological operations are a set of operations which can be performed on binary images. They can be used to obtain structural information from an image and can serve as a useful and time-efficient method to detect edges, objects and text from complex images. The two basic morphological operations are erosion and dilation, which can be used to remove noise and jagged edges from a binary image:

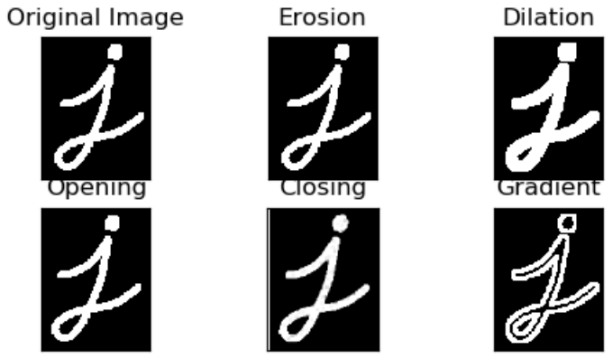


Figure 18: Visual illustration of morphological operations[24].

Morphological opening (dilation of erosion) and closing (erosion of dilation) are two composite operations which are frequently used as a basic tool in computer vision to enhance or detect specific structures, such as edges and corners, in an image[25]. These operations have also been defined for greyscale images.

A sequence of morphological operations can be used to detect regions in an image which are likely to contain number plates (by looking for light-coloured rectangular regions containing edge-dense dark-coloured text). First, two binary masks are created—one containing light regions and another containing dark edges. These masks are combined through a binary `AND` operation, then rectangular regions with an aspect ratio close to 4 are extracted and analysed[26].

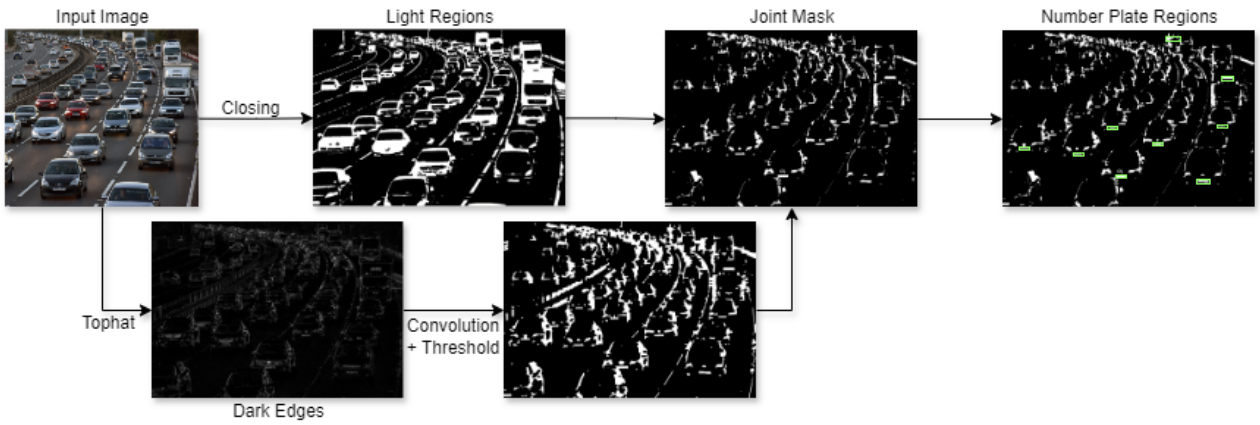


Figure 19: Process of detecting number plate regions in an image.

Optical Character Recognition

Optical character recognition (OCR) is the problem of detecting and translating physical handwritten or printed text into a digitally encoded string of characters. This problem is widely applicable to many real-life tasks including automatic document scanning, machine translation and a variety of OSINT tasks[27]. As a result, OCR is a well-researched area and there exist a variety of powerful OCR tools and libraries, some of which are publicly available to use for free.

A commonly used open-source OCR tool is the Tesseract text recognition engine, which can be used directly via the command line or through a fully-featured API[28].

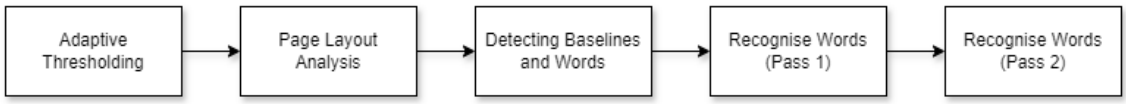


Figure 20: Architecture of the Tesseract OCR engine[29].

Tesseract performs two character recognition passes sequentially—one through a static classifier and another through a dynamic classifier—to maximise the accuracy of each character prediction[29]. The static classifier breaks the characters into smaller shapes and converts the pixel arrays representing each shape into a feature vector before classification. This improves Tesseract's performance on damaged images and partially obscured characters. The dynamic classifier is trained on the output of the static classifier and exists to make the engine more robust to various font types. These features make Tesseract capable of recognising text in a wide range of environments.

Automated number plate recognition is a relatively well-explored problem in computer vision. However, solutions are typically tuned towards a specific environment due to the variations in the features of number plates across different countries. Although the ANPR functionality in the AutoOSINT app will focus primarily on European and American number plates, the joint approach of using morphological operations and the Tesseract OCR engine has been shown to be capable of detecting and analysing number plates from across the world[30]. This approach should allow the system to be easily expanded upon so that it can recognise a wider range of number plates in the future, given further development.

Google Vision

Google Cloud Vision API is a ready-to-use computer vision API compatible with REST architectures that aims to allow developers to easily integrate common vision detection features within applications. Features include facial recognition, explicit content tagging, image labelling and optical character recognition. The API features are cloud-based and easily scalable, making them ideal for commercial use, although Google charges for heavy use of this service[31].

Text Detection

One of the primary features of Cloud Vision is text detection: given an image file or URL input, it is able to generate a dictionary of image regions containing text, along with annotations. The API allows for up to 1,000 free queries per month. Unlike Tesseract OCR, the text detection feature included with Google Vision API is able to extract text from large outdoor scenes, which removes the need for locating regions of text before processing.

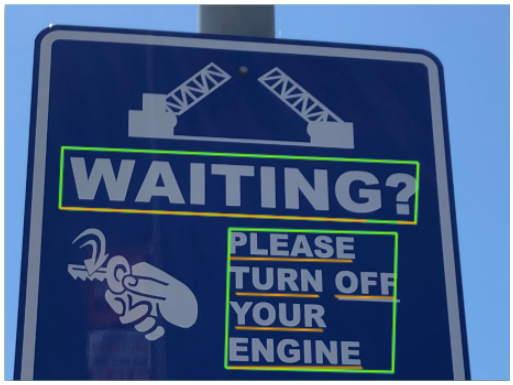


Figure 21: Demonstration of Google Vision API text detection[31].

Furthermore, Google Vision API text detection was found to be more accurate in many cases than alternatives such as Tesseract OCR[18]. However, due to its cost, its use in this project will be limited and other automated tools will be utilised where possible.

Label Extraction

In addition to text detection, Cloud Vision features a label extraction function. When given an outdoor image, the API for this tool returns a collection of descriptive terms, such as 'street', 'town' and 'neighbourhood', along with a normalised score for each term. This label detector is able to detect general objects, themes and locations in an image[32], although the training methods and source code are not published, so a full list of all possible descriptive terms is not available.

Bing Visual Search

Bing Image Search

Bing is an online platform owned by the Microsoft Corporation. It is most known for its search engine product which offers search capability on the World Wide Web. Recent years have seen the development of said tool from standard search functionality to include more complex querying parameters and introducing **image search** (submitting an image as the query as opposed to text). The exact method by which engines like Google or Bing perform image searches is a closely guarded secret - it is certainly a complex system - however, for the project it is enough to abstract this information out.

Nonetheless, both these features have made it incredibly easy to perform OSINT - image search has become a staple in an OSINT expert's toolbox. Experts at such corporations have been pouring millions of dollars into developing the search algorithms - something that the project would unlikely rival in any capacity. Thus, including an image search feature which exploits such powerful algorithms would be very informative and achieve positive results at a lower development ost (mainly in terms of time).

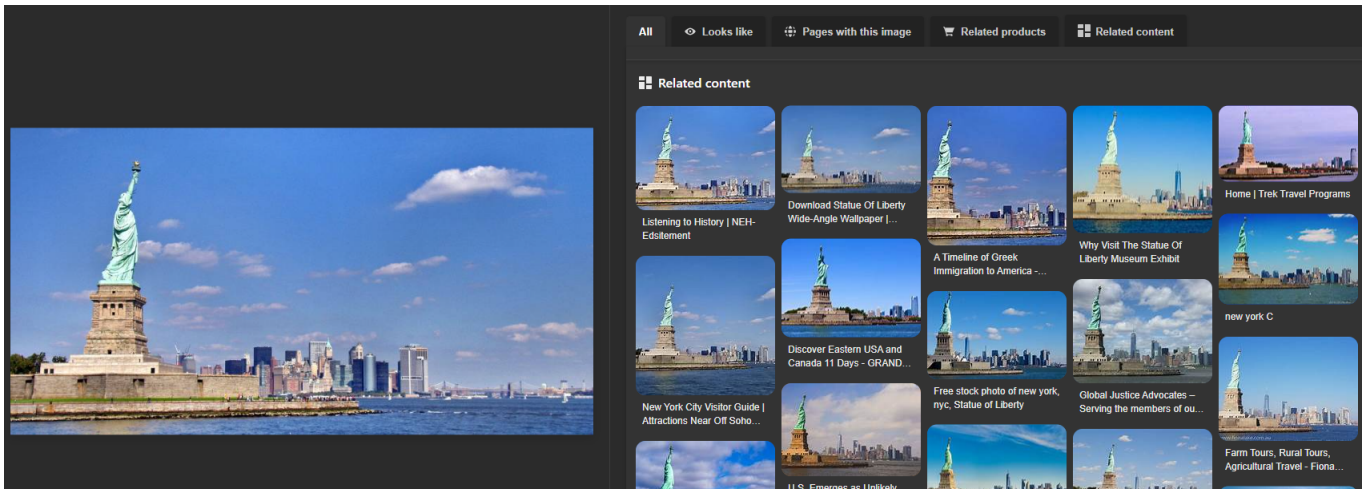


Figure 22: Demonstration of Bing Image Search

Azure

Azure[30] is a cloud computing service also owned by Microsoft Corporation. Without going into excessive detail, cloud computing is the idea of flexible and secure resource access on demand. Cloud computing is often provided by a company which sets up infrastructure on a large scale - large enough to benefit from economies of scale. This includes services like storage and access to communal GPUs for high-performance computation.

As part of Microsoft's centralisation project, a large number of Microsoft services are being moved to Azure. This includes an API service for Bing's Image Search called "Visual Search". The group was able to set up a free version of the Azure Account and use the Bing Visual Search API up to a certain daily limit. However, for greater demand and traffic, a paid package may need to be selected.

[30] https://azure.microsoft.com/en-gb/

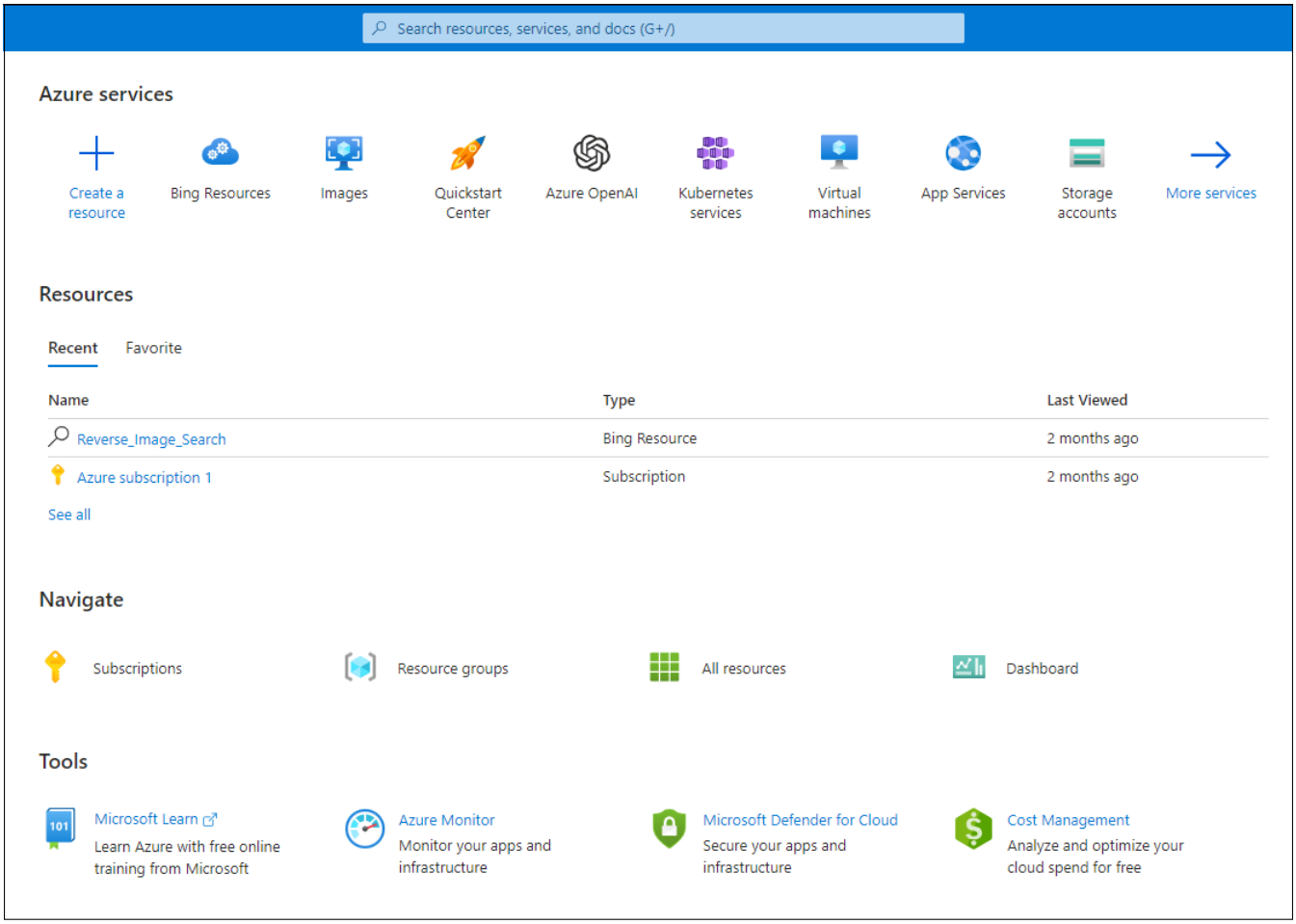


Figure 23: The Azure interface with a set up visual search resource

Alternatives

Other image search APIs exist online. Two worth mentioning are Google image search on Google Cloud[31] and SerpAPI[32]. However, both require payment, to be used. Due to limited resources, the project has utilised the free Azure version, but in the future, the used API could be exchanged, especially as there is an indication that Google image search is more powerful and accurate.

Key Point Matching

Whilst Bing Visual Search does return the top most similar images to a query image, it does not return a metric to indicate similarity. Some of the images could be erroneous and the information provided to them would skew the collected data. Hence an image similarity metric needed to be selected/developed to verify which images are useful and how useful.

Some considerations:

- A 3D scene/location can be photographed from multiple perspectives
- Imaging parameters like lighting, FOV, rotation and scale can be different per image

[31] https://cloud.google.com/
[32] https://serpapi.com/

- Parts of an image could vary over time but still depict the same location

From the above, it can be concluded that the image comparison method needed the ability to recognise features from different perspectives and is robust to varying imaging parameters. Luckily such algorithms exist, namely: SIFT.

SIFT

SIFT (Scale-Invariant Feature Transform)[33] is a computer vision algorithm which extracts key points and descriptors from images. It was introduced by David Lowe in 1999 and was instantly recognised for its groundbreaking robustness to changes in scale, rotation, illumination, and viewpoint. Such properties make SIFT ideal for applications such as object recognition, image stitching, and 3D reconstruction.

SIFT Algorithm Overview:

1. **Scale-Space Generation**: Generate the image at multiple scales by performing Gaussian blurring at different standard deviations and taking a difference of Gaussians.

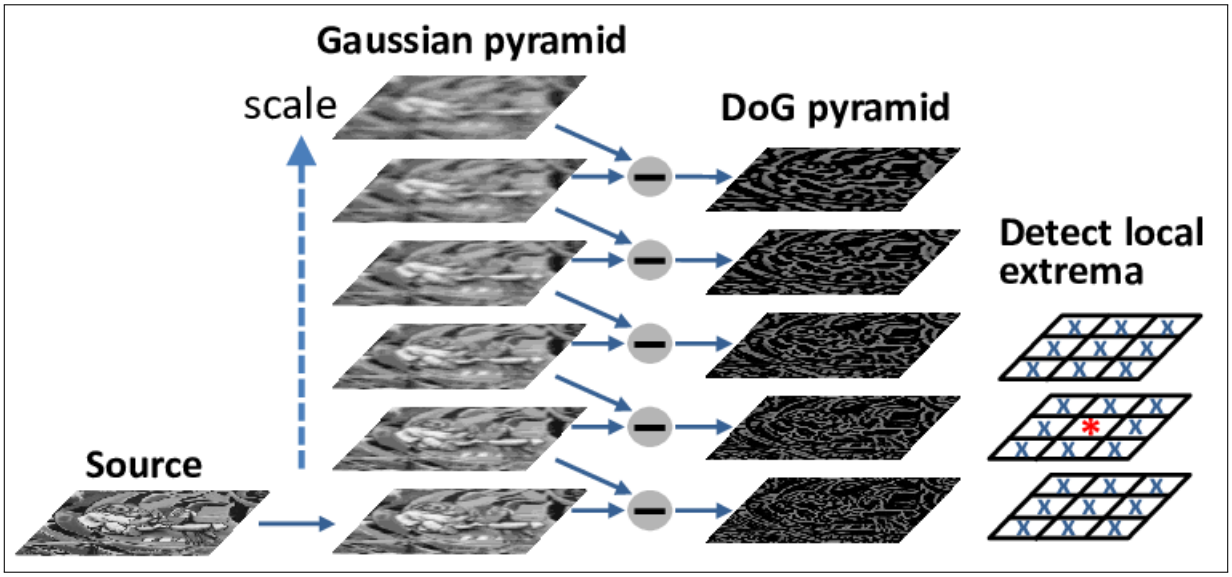


Figure 24: Scalespace creation and extrema detection

2. **Extrema Detection**: Identify keypoints by searching for local extrema in the scale-space (pixels which stand out in a single scale and between adjacent scales).
3. **Keypoint Localization**: Filter out low-contrast and edge keypoints.
4. **Orientation Assignment**: Assign an orientation to each keypoint based on local image gradient histograms - introduces rotation invariance.
5. **Descriptor Generation**: Create a descriptor for each keypoint using local image gradients - introduces invariance to viewpoint, scale and illumination changes.

Once keypoints and their associated descriptors are identified two images can be compared by comparing these keypoints, the number of matched keypoints will indicate how similar the scene in two images is.

Image Information

The ultimate goal of reverse image search is to extract information from similar images to reveal/narrow down the location of the query image. Bing Visual Search sources images to the query images whilst SIFT verifies the usefulness of each suggested image. HThe next step is information extraction. The question therefore is how can images be used to reveal geographical data? Or in other terms, in what different ways is information stored/conveyed by an image?

- Geographical location can be stored in the image representation itself - as image data, such as visible place names or text revealing directions.
- Metadata is intended data that is purposefully attached to images to record the image creation, this includes information such as time or coordinates.
- Image captions and image names often describe a photograph scene and unintentionally reveal the image location. Whilst these channels were not meant for extracting location data, they often do contain valuable data.
- Related information on the image's source webpage. A deeper analysis can be performed on the content here to determine location.

For reverse image search, the most applicable sources of information are the last two. It is possible to use these channels to suggest possible locations. This was a theory that was explored by the implementation.

Mountain Geolocation Data

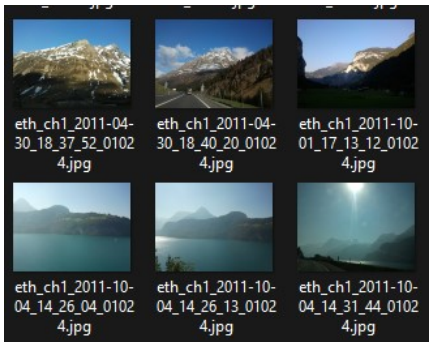
Localised Photographs

One of the biggest obstacles in creating a wilderness/mountain geolocation model is the lack of standardised photograph datasets of mountain areas. The phrase "standardised photograph datasets" needs to be broken down and each issue analysed:

- **Shortage of Photographs**: There is simply a lack of photographs relative to the mountain coverage on Earth. While mountain photographs are abundant, humans have barely scratched the surface of capturing Earth's mountainous regions. To put the scale into perspective, approximately 25% of the Earth is classed as mountain regions according to UNEP[33]. This is 128,000,000 km2 of area to be photographed. This is a colossal task and does not even consider that the regions should be photographed from various perspectives, that obstacles will limit the range that each photograph can depict and that most existing photographs captured are from accessible locations. This means that spatial coverage is not only sparse but also frequently uneven.

  It is important to highlight that most places have been mapped (as will be covered in the "Digital Elevation Model" section) but not every location has been captured from an aerial view, let alone a "human eye" perspective.

- **Missing Geolocation Data/Labels**: Potentially the biggest difficulty with obtaining valid mountain data is the "datasets" component of the larger issue. While many photographs exist, few are well-labelled and documented with data such as coordinates, elevation or orientation. Without such data, images depicting mountainous areas have no utility for geolocation. Hence at this stage, it would be theoretically impossible to construct a library of Earth "photographs" to search through and match queries - with emphasis on the word "photographs".



(a) Mountain photography images

ImageName	Latitude	Longitude	Altitude	Yaw	Pitch	Roll	FOV
28488116812_f5a57ca0f6_k	46.2173	10.1663	439.5	-0.2728	0.243122	0.014817	0.549165
28561570606	46.3463	6.84551	1407	-0.0266	0.087538	-0.01331	1.1531
eth_ch1_04032011388_01024	46.86	9.837	2246.5	-1.3293	-0.07453	0.025975	0.984675
eth_ch1_04032011389_01024	46.86	9.837	2246.5	-2.0097	-0.03249	0.032381	0.983756
eth_ch1_04032011390_01024	46.86	9.837	2246.5	-2.4636	-0.01512	0.03083	0.983893
eth_ch1_04032011391_01024	46.86	9.837	2246.5	-2.8523	-0.02946	0.042805	0.993818
eth_ch1_1332166_01024	46.422	7.402	1939	1.24176	-0.02378	0.036813	0.643872
eth_ch1_2011-04-30_18:37:52	46.5296	9.19213	1620.5	2.23204	0.307024	0.029532	0.92434
eth_ch1_2011-04-30_18:40:20	46.5383	9.2327	1566	2.6975	0.249273	0.098175	0.939313
eth_ch1_2011-10-01_17:13:12	46.5562	7.90146	855.5	-2.0916	0.187911	0.049084	0.939313

(b) Mountain photography metadata

Figure 25: Example Ideal Mountain Photography Dataset

- **Unstandardised Photographs**: Another problem with the available data is that it is not "standardised". The data that is available often comes from many different sources. Collecting the data and verifying its reliability itself poses another obstacle, but different sources mean different methods, technology and purposes. This inherently results in photographs with different resolutions, different sizes, different fields of view and overall varying depictions of mountainous regions from extremely different scenes. The metadata (like GPS coordinates) will also exhibit discrepancies in the accuracy and precision of such data.

[33] https://www.unep.org/interactive/explore-ecosystems/mountains/en/index.php

Whilst the photograph and data variability increases the difficulty of constructing a viable model, it ultimately doesn't prevent one from being made (not considering the previous issues). Nonetheless, it is a point worth considering.

GeoPose3k

To support the above claim, a 2022 study into a new method of geolocation in mountainous areas called Crosslocate[34] found that very few such datasets exist. From those that do exist, there is not enough data to form libraries to geolocate around the world and instead, the sets are limited to specific locations - in this situation the Alps. This could come in handy for smaller-scale geolocation, but there are insufficient photographs to forma a global library of mountains.

A dataset worth mentioning as it is widely used by both Crosslocate and the project is GeoPose3k [35]. GeoPose3k is a dataset of approximately 3000 panorama images (360°images) of the Alps region with annotated data - like elevation, WGS coordinates and orientation. This is an example of an ideal database. Part of this database can be depicted in Figure 25.
Naturally, as time goes, on and technology advances, our capabilities to capture Earth's mountainous regions will drive forward too. This is especially true as the world is experiencing a data boom which means that the volume of data available is growing by the day and the creation is only accelerating. However, an ideal set - such as the one described above - may still be far from being ready. We need a solution for now.

Digital Elevation Model

Despite the usefulness and criticality of photographs for mountain geolocation, photograph images also have associated weaknesses:

1. Weather conditions can drastically change the appearance of a mountainous area. From snow to fog, the landscape can become unrecognisable if unique photograph features and key points become hidden.

2. Photographs can capture anomalous objects in the way of the scenery. Entities such as clouds, trees and buildings can block the depicted scene and limit the number of recognisable features. Photographs would need manual inspection to determine usefulness.

3. Photographs can suffer from imaging degradation, which means their utility is affected by factors such as brightness, resolution and noise. While some of these cases can be partially alleviated with image enhancement and data augmentation, there are cases where images are simply unhelpful.

There are alternative depictions of the environment which do not suffer from such effects. An example to focus on is a **Digital Elevation Model (DEM)**. Digital Elevation Models are topographic maps representing the height/elevation of sampled points on the planet. DEMs can be separated into DTMs (Digital Terrain Models) and DSMs (Digital Surface Models).

Digital Surface Models use methods like LiDAR (Light Detection and Ranging) or photogrammetry to capture the elevation of the earth's surface [36], **including all objects on it, such as trees and buildings**. LiDAR uses laser beams to measure the distance between the sensor and the ground, while photogrammetry uses aerial photographs to create 3D models.

Digital Terrain Models are similar to the above, however DTMs **only capture the elevation of the bare-earth ground**. DTMS will use various forms of pre-processing and post-processing to extrapolate the elevation of the ground under a blocking object [36].

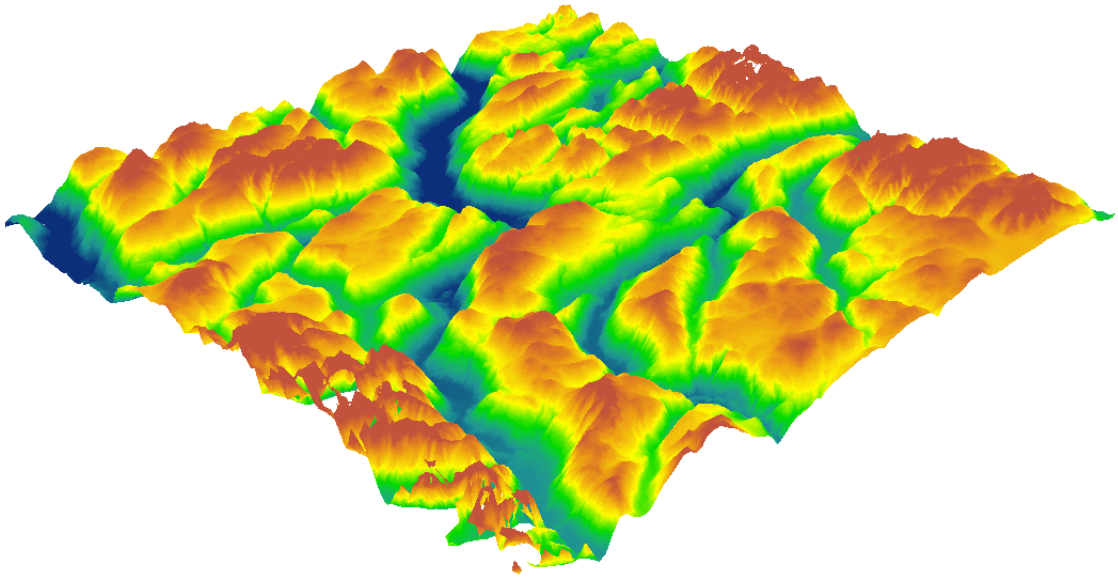


Figure 26: Example Visualisation of a DEM.

The benefit of DEMs is their availability, there is a massive collection of DEMs available online freely with varying sampling rates, precision and accuracy. Selecting the appropriate model is an easy and quick process and the variability of models means that the choice can be tailored specifically to a user's need. DEMs also possess all the required geographical information such as coordinates or allow the derivation of values like yaw or pitch when generating 2D images.

Whilst DEMs do not feature information such as colour or texture, the contour information is the most important when trying to localise a photograph as demonstrated by Crosslocate. Hence DEMs are a crucial resource for generating a library of locations - specifically in the context of this project.

Mountain Horizon Matching

Ground-breaking Concept

In the early 2010s, analysts began to recognise that purely using photographs for geolocation in the wilderness - such as mountains - is virtually impossible. At the same time, theories regarding the relationship between mountain horizon shape and geolocation prompted experimentation with DEMs and image ridge lines.

An influential paper from 2012 called "Large Scale Visual Geo-Localization of Images in Mountainous Terrain" [37] was one of the first attempts to match image ridge shape to DEM representation. The results of the experiment proved fruitful at the time, however in comparison with modern methods they fall behind. Nonetheless, the idea of mapping photographs to DEMs would be ground-breaking and branch into a vast number of studies and variations on the original.

Idea of the Approach

An overview of the process (see Figure 27):

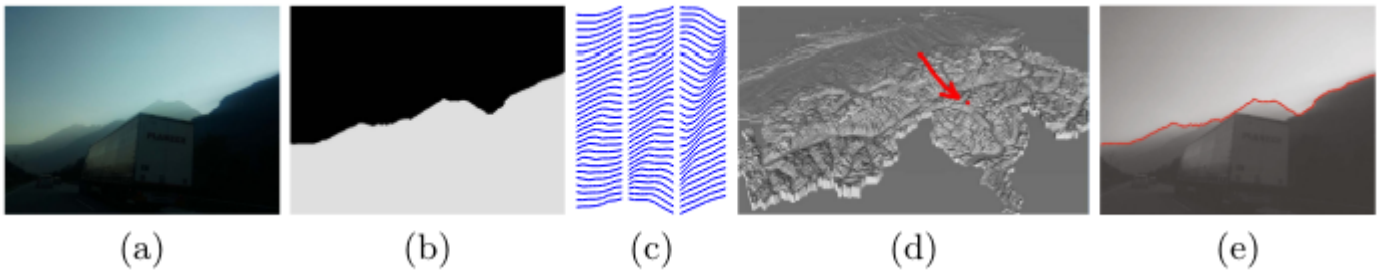


Figure 27: Different stages in the proposed pipeline: (a) Query image somewhere in Switzerland, (b) sky segmentation, (c) sample set of extracted 10◦ contourlets, (d) recognized geo-location in digital elevation model, (e) overlaid visible horizon at estimated position [37].

1. Convert query image into a simplified representation:
   (a) Extract photograph ridge line through sky segmentation
   (b) To achieve robustness to occlusion, noise and systematic errors - like inaccurate focal length, the ridge line will be represented as a set of overlapping 10°contourlets of width w.
   (c) The contourlets are sampled at evenly spaced points producing n feature vectors yi.
   (d) The contourlets are normalised by subtracting the mean of yi vectors and dividing by contourlet width w. This results in camera tilt invariance.
   (e) The contourlets are quantised (each yi falls into a bin which has an associated number).
   (f) The binned contourlet representations are concatenated into a string of numbers - the contour word.
2. Sample a DEM at regular intervals - the collection of points should split the location into a grid of points.
3. For each point extract a 360°panorama representation of the horizon repeat process 1(a) - 1(f) to obtain the location contour word
4. Each location contour word is stored and indexed for quick search and retrieval
5. Obtain a comparison score by comparing each of the i visual words (bins/numbers) in a query and indexed database image using the "contains" equation:

$$D^c(q,d) = \sum_i (w_i \cdot \max(q_i - d_i, 0)) \tag{1}$$

6. Employ a voting system to determine the orientation of the query image within the panorama

At the time of publishing, the results were very positive, overtaking many of the existing solutions in terms of accuracy, speed and robustness. However, not long after releasing the paper, new solutions following a similar structure/approach would overshadow the results obtained. The main weakness of this approach was the feature representation of the mountain library - contourlets were an outdated

feature representation which only captured a small amount of information about a scene.

The importance of the paper came from the problem approach. The critical idea introduced in the study was to build a library/database of sampled locations generated from a DEM and to store them as a representation to be quickly searched through. Many subsequent studies would follow a similar approach, including this project's final implementation.

To summarise, an effective solution to geolocalisation follows a framework which requires 4 items:

1. A query
2. A database of images depicting candidate scenes/locations
3. A descriptor extraction technique - to represent both the database images and query photographs
4. A method of comparing descriptor similarity

Cross-modal localisation - CrossLocate

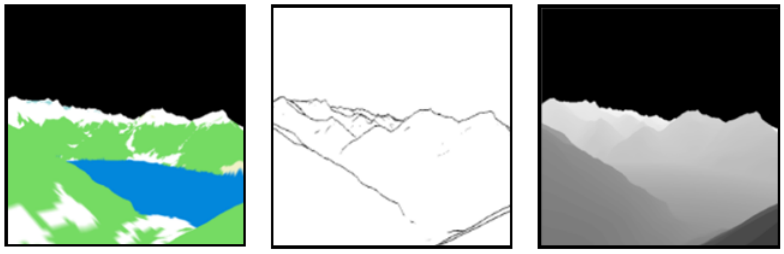
The project's final implementation of a mountainous geolocation system is based on an existing solution published in 2022 as part of a study by the same name Crosslocate [34]. Crosslocate follows a similar approach to the one explored in the previous section but aims to rectify the shortcomings of the previous study with modern technologies and approaches such as Neural Networks.

Whilst the 2012 study [37] uses mountain ridgeline contours as the stored representation of the DEM, Crosslocate explores the effectiveness of a variety of other modalities - such as segmentations, edge silhouettes and depth maps - to create a library locations.

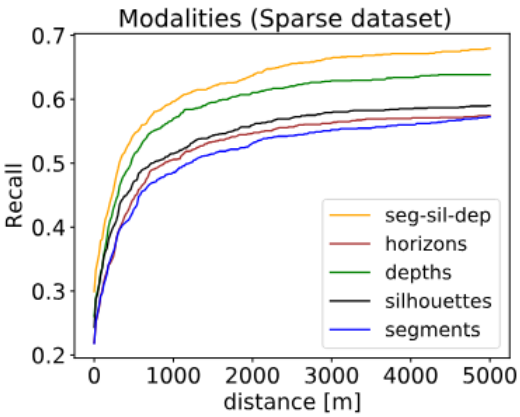
Depth Maps

The first improvement discovered by Crosslocate was the close relation between photographs and depth maps. Depth Maps are images depicting a scene where each pixel value is the distance from the imaging apparatus to the pixel's corresponding point in the 3D scene. Greater values thus represent points in the environment which are further away.

Depth Maps can be obtained by conducting a technique called ray-casting on a Digital Elevation Model (DEM) [38]. Raycasting is a process of projecting virtual "light rays" from the focal point of a camera through each pixel in the camera sensor (or 2D screen) into the 3D scene to determine what is visible along the ray. The process can yield values such as distance - which is used to generate a depth map [39].



(a) Visualisation of the modalities from left to right: segmentations, silhouettes, depthmaps



(b) Modality Performance

Figure 28: Modalities tested by CrossLocate

Naturally, of all the modalities mentioned, depth maps are the most complex and contain the most information about a depicted scene. Not only is the range of values for depth maps larger, but the other modalities such as extracted ridgeline horizons and edge silhouettes can be derived from depth maps - the reverse, is not true. Therefore, when comparing each modality's performance, depth maps outclassed all other individual approaches (as depicted in Figure 28) making them the most suitable modality.

Hence, the optimal solution deemed by the Crosslocate study was to utilise depth maps as the basis of a location database.

Feature Extraction

Another change introduced by Crosslocate is the descriptor creation approach. Whilst [37] used the complex process of creating contour words - which specifically complimented the horizon modality - this approach wouldn't apply to depth maps. This method would waste important depth information. Instead, the authors opted for a modern descriptor/feature extraction approach - Convolutional Neural Networks, or more specifically, Convolutional Encoders.

Given a 3-channel image (either an RGB photograph or a stacked depth map image) the neural network would perform a series of convolutions, max pooling operations and normalisations, which result in a 512 feature vector. This would be the image descriptor. See the crossmodal geolocalisation development section for the final implementation architecture.

The benefit of this approach is that rather than manually identifying useful features, the neural network can select the 512 most important features automatically. The learning and feature extraction are combined and performed by the program. There is much evidence online that machines are quicker and more accurate at feature extraction than humans in this situation, making it an improvement.

With these decisions and implementations in mind, Crosslocate constructs a complete and descriptive dataset for localisation in a mountainous area:

- A **Query Set** of photographs with exact geographical data such as coordinates and orientation.

This is sourced from an expansive and reliable dataset called Geopose3k [35]. The dataset contains 3000+ 360°photographs of the Alps.

- A **Database** of depth maps to be searched through. Various DEMs are available to construct this, the one used in Crosslocate was freely available online [34]. The DEM is sampled at locations corresponding to the Geopose3k coordinate locations and ray cast to produce depth maps.
- A **descriptor/feature representation extraction mechanism**. This is the VGG16-like CNN which produces 512-feature vectors.

It is important to note that Crosslocate conducted the study on 2 different data scales - a sparse dataset of 30,000 database images as a demonstrator of technology and a Uniform dataset of 3 million images for more realistic application.
For this project, the group was unable to secure a powerful enough machine to process data at the Uniform dataset scale. The group had access to a batch computing system, however, the provided storage, memory and GPU processing power were insufficient. Hence the focus of the protect will be to implement a demonstrator focusing on the sparse dataset.

Similarity Metrics

In the last sections, we can observe 3 of the 4 framework components of the localisation framework selected by Crosslocate. The fourth is the metric for comparing descriptor similarity. However, it does not suffice to just consider the distance between descriptors in this case. Since a neural network generates the 512-feature vectors, it must also be considered what a "good" and "bad" representation is. The descriptors need context, which previously was provided by manual feature extraction. This approach requires guidance and instruction for the neural network to learn representations.

Euclidean Distance:
The descriptors produced by the CNN are vectors of size (1x512). This means most of the common mathematical distance equations can be applied to measure the similarity of a query vector and a proposed database depth map vector. A quick, accurate and proven distance metric - selected by Crosslocate - is **Euclidean Distance**. The equation for Euclidean distance between two vectors x and y, both with i elements is:

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2}$$

Triplet Loss
Selection of an appropriate learning/loss function required consideration of what a "good" representation system is. The goals of such metric must be considered:

- If two images are similar/depict a similar scene, then the distance between the two descriptors should be small. If the images are exact, then the ideal distance between the two descriptors should be 0.
- If two images depict two different locations, then the distance between the two descriptors should be large. The more different the 3D environment, the greater the descriptor distance should be.

[34] https://viewfinderpanoramas.org/dem3.html

This implies that if the distance between descriptors of two geographically similar images is less than the representation distance of a mismatching pair of descriptors, then the metric should reward (or not penalise) the representations produced by the network. On the other hand, if the contrary is true, a penalty should be returned.

Two candidate metrics following such a specification exist - contrastive loss and triplet loss. The ideal loss for the nature of localisation is triplet loss. Triplet loss was first introduced as part of Facenet[40], a paper for database retrieval of faces, however, it is equally as effective for geolocation purposes. Triplet loss requires 3 inputs: An anchor (query) descriptor, a positive (matching) descriptor and a negative (non-matching) descriptor. Using a distance metric - like the Euclidean distance - an error between the anchor and the positive/negative examples can be calculated with the following equation:

$$L_{\text{triplet}}(A,P,N) = \max(0, d(A,P) - d(A,N) + \alpha) \tag{3}$$

Where:
A, ***P***, and ***N*** are the anchor, positive, and negative examples respectively.
d(A,P) is the distance between the anchor and positive samples.
d(A,N) is the distance between the anchor and negative samples.
α is the margin hyperparameter - the training process will encourage the distance difference between the representations of the anchor-positive pair and the anchor-negative pair to be at least this amount.

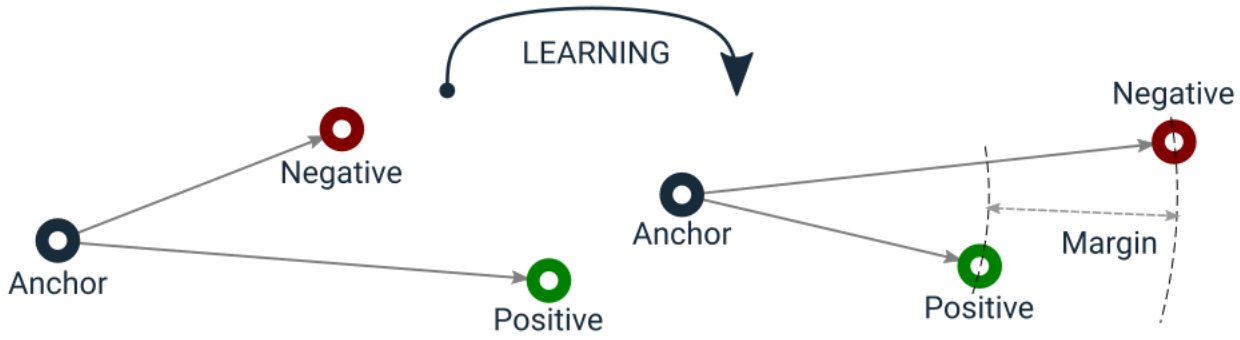


Figure 29: Triplet Loss returns encourages the distance between the anchor-negative pair to be greater than the distance of the anchor-positive pair by a margin.

Using the datasets with the CNN and triplet Loss allowed Crosslocate authors to train models which recognised when a photograph is similar to a depth map. If a similar depth map was identified in the library, then the stored geographic data revealed the location.

Clustering GPS Points

Several of the semi-automated tools produce a list of GPS coordinate predictions. In some cases, such as the road sign analysis tool, it may be difficult or impossible to narrow down the potential locations to a single point. For example, two towns or cities in different geographical locations may share the same name, so the road sign analyser would be unable to determine the exact location of some images. By considering the predictions given by all of the different tools included with AutoOSINT, we can further narrow down the possible locations of the image. Additionally, using all image features should increase the likelihood of making correct predictions.

An intuitive approach to determining structure in a set of data points is clustering. Large and dense clusters are more likely to centre around correct predictions because there is a consensus among multiple OSINT tools.

Haversine Distance

The Haversine distance formula is used to calculate the shortest distance between two points on a sphere given their longitudes and latitudes. It is commonly used in navigation to find the distance between two geographical locations. Given angle θ between two points on a sphere's surface we have:

$$\text{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2}$$

To compute the distance between GPS coordinates, a distance equation can be derived[41]. Given the latitudes ϕ_1, ϕ_2 and longitudes λ_1, λ_2, we can write:

$$d(\phi_1, \lambda_1, \phi_2, \lambda_2) = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_1 - \phi_2}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_1 - \lambda_2}{2}\right)}\right)$$

This distance measure can be used to calculate the pairwise distances of all of the GPS coordinate predictions given by the OSINT tools, enabling us to perform clustering on the data.

DBSCAN

DBSCAN is a density-based clustering algorithm which envisions a set of data points as a graph: two points close to each other are connected by an edge and each point is clustered together with all points reachable from it. Furthermore, sparse data points are classified as noise instead of being added to any cluster[42].

DBSCAN has several key characteristics which make it an ideal clustering algorithm for predicting the image location. The most important are:

- **Robust to noise:** DBSCAN ignores outliers, which is useful as some of the OSINT tools are likely to produce incorrect or misleading results in certain cases.
- **Density-based:** dense groups of points (which should correspond to predictions that are more likely to be correct) are clustered together. Additionally, any number of clusters can be generated depending on the structure of the data, so there is no requirement to estimate the number of clusters beforehand.

- **Few parameters:** The algorithm takes just two parameters, affecting the minimum density of clusters and which points are labelled as noise. Therefore, DBSCAN can be made to be effective with a wide variety of data inputs without adjusting parameters.

Although implementations of the DBSCAN algorithm tend to be slower[43] than some alternative clustering algorithms, this is not a significant issue for this application because the number of data points is relatively small (< 100).

Vision Transformers for Geolocation

The transformer is a type of neural network which was initially applied to natural language processing, using a self-attention mechanism to embed strong high-level representations of language. Text tokens are contextualised, allowing key tokens to be given more importance in a body of text in order to better understand and convey meaning[44]. Very recently, this approach has also been applied to image data in the form of vision transformers (ViTs). ViTs have been shown to outperform state-of-the-art convolutional networks in many image classification tasks when provided a large set of training data[45].

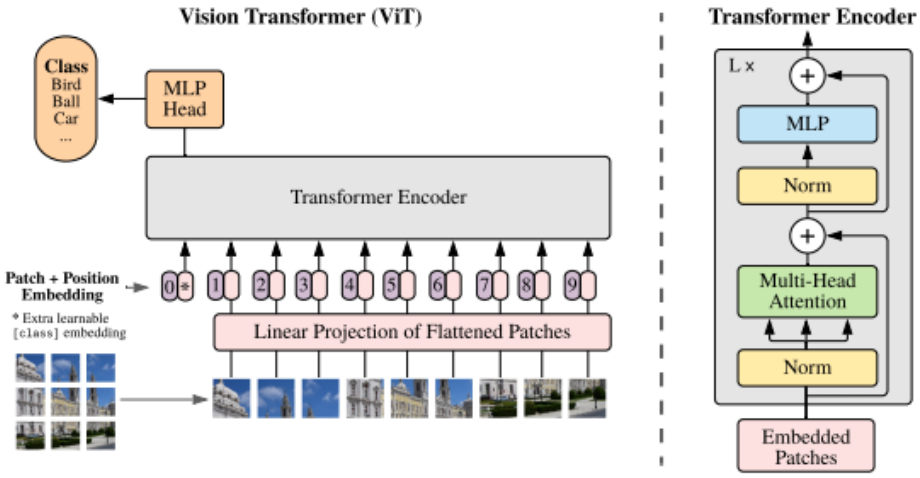


Figure 30: Architecture of the first proposed vision transformer[45].

In a vision transformer, an input image is split into uniform patches and embedded linearly along with their position in the image. The resulting vectors are then fed into a standard transformer model, with a learnable 'classification token' included in order to perform classification.

Due to the large amount of training data required, building a successful ViT model for classification can be highly expensive and time-consuming, especially for more difficult tasks where the number of classes is large[46]. Instead of training a new ViT from scratch, we can utilise existing models trained on vast datasets and with a large amount of computing power.

CLIP and BioCLIP

CLIP is a visual machine learning model which is capable of performing zero-shot classification of images[47]. It is trained on a combined dataset of images with associated text captions. It uses a natural language embedding technique first proposed in 2013 which is capable of effectively and efficiently embedding both semantic and syntactic meaning into word vectors[48]. The model trains the word embeddings along with an image encoder (typically a ViT), learning to correctly pair the text and image vectors

together. After training, CLIP performs zero-shot classification by embedding the words or descriptions of the target classes.

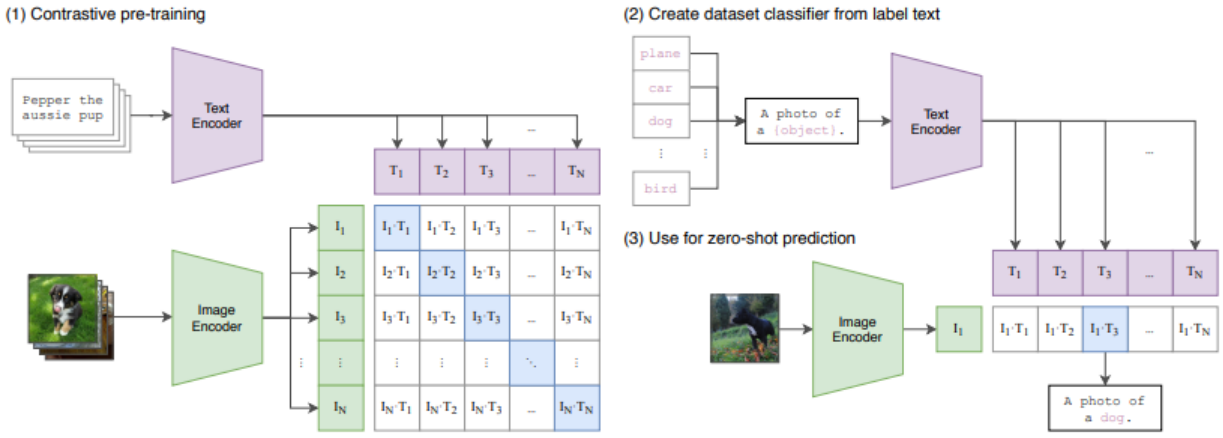


Figure 31: Training and testing process of CLIP[47].

BioCLIP is a CLIP model which utilises a dataset of 10 million images of plants, animals and fungi and taxonomic labels for zero-shot classification[49]. It builds on the original CLIP model and takes advantage of the way that the evolution of species relates to their taxonomic structure (closely related species often have a similar appearance as well as similar taxonomy). The model was shown to substantially outperform previous state-of-the-art models in image classification tasks.

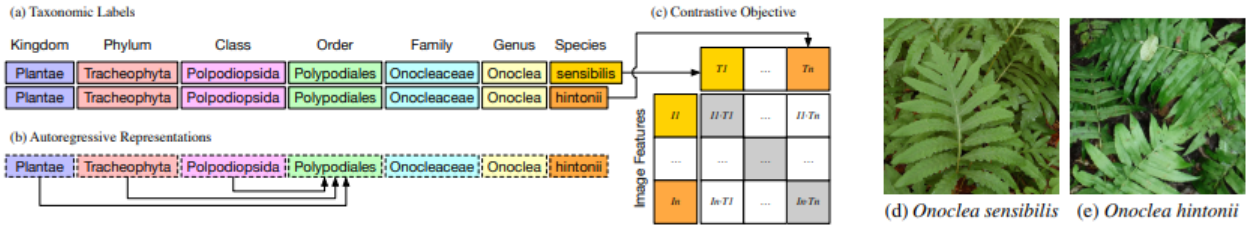


Figure 32: Training of BioCLIP. The two taxa are identical apart from the species. BioCLIP's text encoder naturally embeds this hierarchical structure[49].

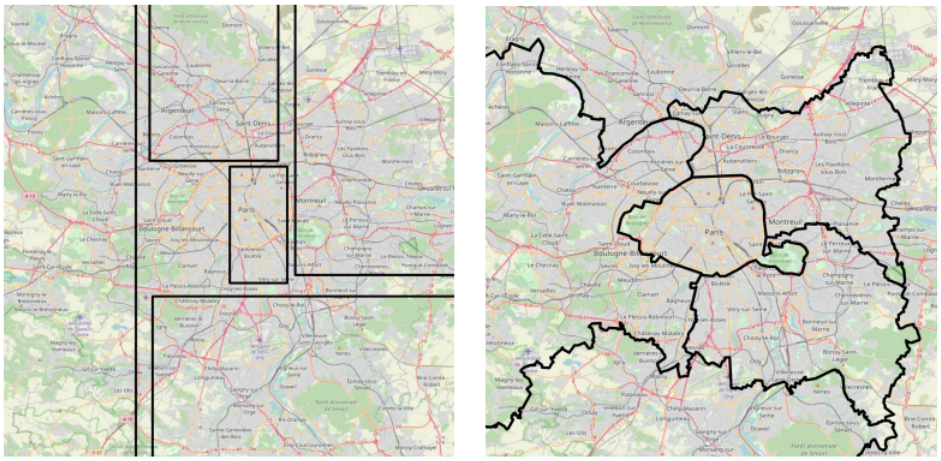
BioCLIP can be used for geolocation because many plant and fungi species are mostly localised to a specific geographic region—this area of study is called phytogeography[50]. Identifying plant species in an image would help to increase certainty of or help to rule out a prediction. Additionally, taxonomic distance and geographical distance between habitats of related species are often (weakly) correlated, which could help to reduce outliers when making predictions[51].

After testing this model on a small test set of images taken from Google Street View[52], we found that the BioCLIP model was not well suited to classification on large outdoor scenes. This is because BioCLIP is trained on close-up images of plants where small details are easily discernible. The examples used in testing for AutoOSINT are lower resolution and generally only feature plants further away from the camera. Instead of BioCLIP, we will consider using a ViT or CLIP model more suited to the specific task of image geolocation.

PIGEON GeoGuessr Bot

PIGEON is a CLIP model trained specifically to play GeoGuessr, a geographic guessing game where users are challenged to solve a single problem: given a Google Street View image taken somewhere in the world, identify its location. Trained on street-level data taken from the game, PIGEON is able to place over 40% of its guesses within 25 kilometres of the target. In a blind experiment against humans, the model was able to rank in the top 0.01% of players of the game[53].

PIGEON makes use of administrative boundary data and CLIP's semantic text embeddings to hierarchically split the world into semantic regions with four different levels of granularity. This allows PIGEON to make meaningful distinguishments between different areas, and especially between different cities.



(a) With naive, rectangular geocells.

(b) With our semantic geocells.

Figure 33: Semantic regions created by PIGEON around Paris, France[53].

Our preliminary testing showed the impressive performance of the model, achieving an accuracy of over 80% on zero-shot classification of around 100 urban scenes collected from Google Street View and Wikimedia Commons. However, model performance was much poorer when tested on a dataset of rural and mountainous regions. A possible explanation for this is that paths and roads are sparser in rural areas, so Google Street View data is not as extensive as in urban areas. Furthermore, performance appeared to worsen in basic tests when the number of classes available for the model to choose from was increased.

Depth and Pose Estimation

Depth estimation is the task of measuring the distance of each pixel in an image relative to the camera. It is a common task in the field of mobile robotics and has previously been utilised to precisely determine the location of a drone in space. This problem is called monocular global geolocation (MGG)[54]. Depth estimation would be useful in this project because it would make it possible to infer the camera's exact position given an image containing a landmark with known geographical coordinates.

Thin Lens Equation

A simple method of calculating camera-subject distance is using the thin lens equation[55]:

$$\frac{\text{real object size}}{\text{distance to object}} = \frac{\text{object height on sensor}}{\text{focal length}}$$

Given the height of an object in the input image, the size of the camera's sensor and the focal length of the lens, the distance to the object can be inferred. Given three objects in the same image, or the bearing of the camera when the image was taken, we can calculate the exact geographical location of the camera's point of view.

The thin lens equation approach consists of four main steps:

1. Identify suitable landmarks in the input image.
2. Determine the height of each landmark. This is done either using height data from a landmark database or by applying the thin lens equation to a reference image where the distances to the landmarks are already known.
3. Calculate the distance of each landmark in the input image relative to the camera using the thin lens equation.
4. Use three or more landmark distances (or camera bearing) to triangulate the exact geographical location of the camera.

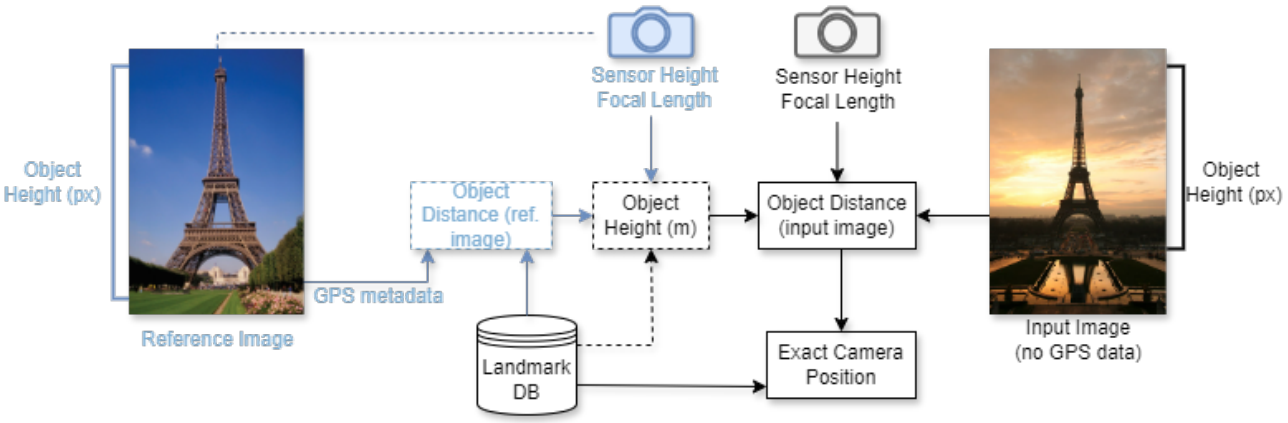


Figure 34: Process of estimating exact camera position from an input image and reference image.

To test the potential efficacy and accuracy of such a model, we measured object height in pixels manually, calculating real-world object height and distance from the camera. The images tested contained

GPS metadata and featured landmarks with known heights (such as the Eiffel Tower) to allow for error measurements.

Using the two images shown in figure 34, the estimated height of the Eiffel Tower (actual height 330 metres) was approximately 195.5 metres—a relative error of more than 40%. Distance estimations were similarly inaccurate and ultimately it was decided that this approach would not be suitable for this project.

There are several factors which affect the accuracy of the inferred distances and heights. The most significant factors that caused this approach to fail were:

- Partial occlusion of landmarks by terrain or other objects.
- Radial distortion caused by the camera lens.
- Imprecise or inaccurate GPS data of reference images.

Additionally, this approach is likely to be impractical for many use cases because it requires knowledge of the camera model used to take the input image, as well as knowledge of the height of multiple landmarks appearing in the image.

Perspective n-Point Pose Calculation

The perspective n-point (PnP) problem is the problem of finding the position and orientation of a camera with respect to a scene object from n correspondence points[56]. The problem is polynomial-time solvable and an implementation of an efficient solution is available in the OpenCV library for Python[57]. However, this method requires knowledge of the scene in 3 dimensions. Because data about objects in an input image is likely to be sparse and difficult to infer from a single image, it was decided that this approach would not be suitable for the task of image geolocation.

Single View Metrology

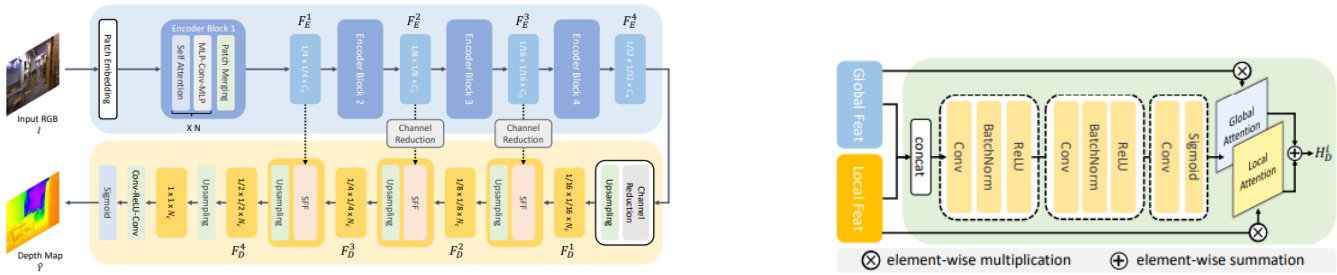
In structured scenes (such as an image taken in an urban environment containing primarily buildings and straight roads), the 3D geometry of objects in the scene can be measured using single view metrology[58]. This technique involves identifying a vanishing point in the image by inspecting straight edges and making measurements on parallel planes. Previous applications for single-view metrology include determining the height of people and objects in security camera footage. The same method can also be used to calculate the position of the camera in the scene.

Because single-view metrology relies on straight parallel lines to measure aspects of a scene, it is unlikely to be effective at measuring images of rural environments. The technique could be more applicable to urban scenes, although it is difficult to verify whether two edges in an image are straight and parallel in real life. In addition, single-view metrology suffers from many of the same camera distortion issues which occur with the thin lens equation approach. It was decided that development time would be better spent on other areas of the project, although there is potential for this technique to be useful if thoroughly implemented in the future.

Global-Local Path Networks

Global local path networks (GLPNs) are a recently proposed ViT-based model architecture which can be used for single-image depth estimation[59]. The model produces both global path (high-level concep-

tual) details and local path (fine-grained) details for an image, then uses a 'selective feature fusion' (SFF) module using attention to integrate the two feature paths.



(a) Overall architecture.

(b) SFF module.

Figure 35: The proposed GLPN architecture[59].

Additionally, GLPN uses a data augmentation called CutDepth which replaces part of the input image during training with the ground-truth depth map to increase variation in the data without affecting the edges in the image. This technique has been shown to improve performance in depth estimation tasks, although training data at long distances was limited[60].

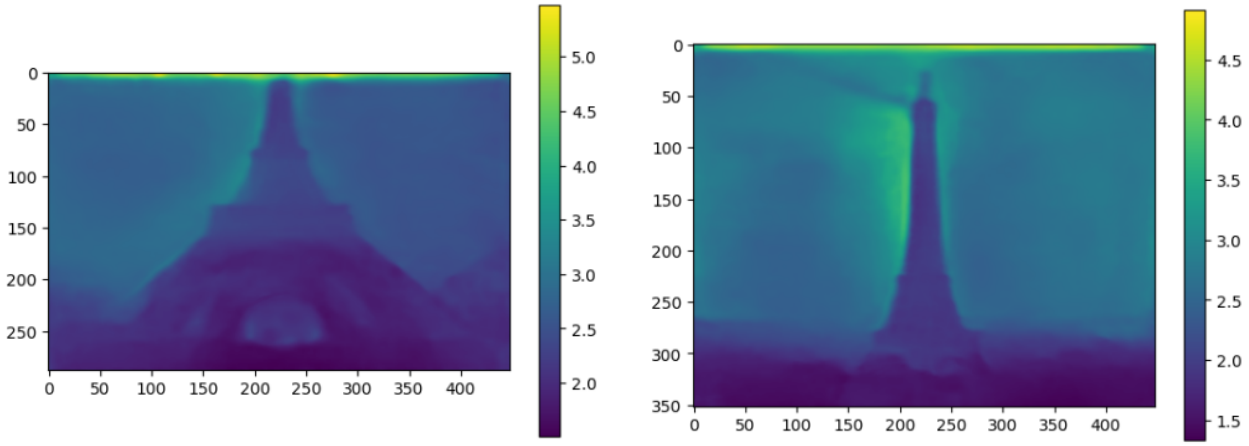


Figure 36: Depth maps generated by GLPN model from two input images.

After testing on a sample test set of outdoor images, it was found that the GLPN model was able to successfully determine relative depth in the images. However, the model was not able to deduce the actual depth at points in the image. For example, figure 36 shows the predicted distance of each pixel from the camera in metres. We can see that the scale of the outdoor scenes was not correctly predicted. Therefore, the model will not perform well for outdoor geolocation tasks.

Implementation

Software Stack Overview

As determined during the application framework research, the system takes the form of a web-app which is hosted on a locally-hosted web server which runs on the user's device. This is then packaged into a desktop application. Figure 37 below illustrates the technologies used for the creation of the platform.

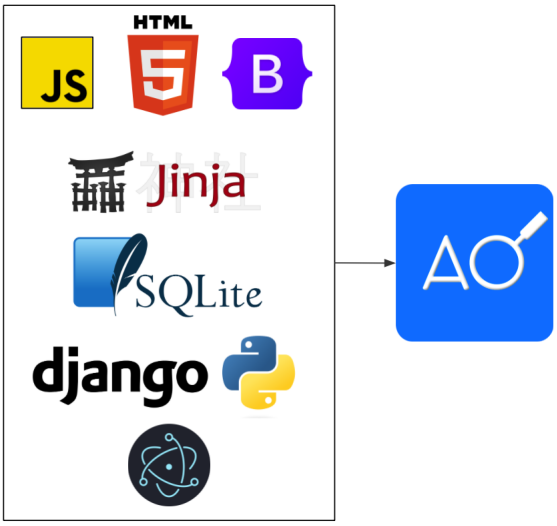


Figure 37: Illustration of the software stack.

The Electron framework hosts a localhost web server on the user's device and launches it using the Chromium browser as a desktop application window. The resulting platform is cross-platform, meaning the same code can be used to create applications on Windows, MacOS, and Linux operating systems.

The Django web framework is used to create a database-driven web-app on top of the Electron-hosted localhost server. The use of Django facilitates the uploading of images for analysis to a locally-stored database managed by SQLite.

Django also supports the employment of a HTML template engine, such as Jinja2, which allows for HTML templates to be written which can be extended by other pages, minimising code repetition and maximising modularity. For instance, the base HTML file of this system consists of the navigation bar and a container for content, and this file is extended by every other HTML file. This displays the navigation bar on every page without requiring the continued repetition of its code.
Another benefit of using template engines is programmatic HTML code generation. Jinja allows for the insertion of Python code into HTML files which means that loops can be used to generate repetitive HTML elements. This is used on the home page where, upon fetching all stored images from the database, a Jinja `{ % for d in data % }` loop is used to write the HTML code for each image.
Similarly, conditional statements can be used. For instance, if the user has not uploaded any images, it is more sensible to prompt them to upload one than to show them a blank screen. Here, we make use of Jinja's `{ % if data % }` and `{ % else % }` constructs to display a different version of the home page depending on the data available.

Finally, used at the top level is vanilla HTML alongside JavaScript with Ajax and jQuery, and a blend of custom CSS, the FontAwesome icon package, and Bootstrap.

JavaScript code is written with the help of the jQuery library for the simplification of HTML element manipulation, and AJAX to allow asynchronous execution of analysis scripts.
Bootstrap is an open-source CSS framework which is used by 19.2% of all websites and provides front-end components for the development of responsive web applications. The system makes use of the Bootstrap navigation bar, its file-upload interface which opens a native window for file selection, as well as its grid-based layout system where HTML `<div>` elements can be given the `col` or `row` class depending on how they are to be laid out on the page. Since Bootstrap is a mobile-first framework, it aims to create dynamic changes to web page appearance depending on the device screen size, ensuring usability and user satisfaction[61]. Custom CSS is applied on top of Bootstrap components in order to refine the appearance and responsiveness of UI elements.

The OpenLayers JavaScript library is used in order to display a map of GPS predictions to the user. It is an open-source alternative to Google Maps and Bing Maps, and vastly improves the user experience when considering that the alternative would be to provide latitude and longitude co-ordinates as results, which would require manual searching to determine their precise location. By using OpenLayers, the user can view the GPS results at a glance, within the application itself, which is particularly helpful if multiple GPS co-ordinates are returned during analysis and must be compared.

Application Structure

The directory tree below illustrates the application structure. Django files are given in orange, Electron files are given in teal, and comments are given in gray, whilst the directories themselves are given in black.

The structure here provides a reference for upcoming explanations of the interactivity between system components, and illustrates the difference between the main Django application in the `autoosint` directory and the sub-application in the `imageupload` directory. The former stores the primary settings for the web-application and the latter stores the majority of the files used for serving the application to the user.

```
./
|-- autoosint/
|   |-- asgi.py
|   |-- settings.py
|   |-- urls.py
|   `-- wsgi.py
|-- data/
|   `-- misc. data stored here
|-- imageupload/
|   |-- scripts/
|   |   `-- Python feature scripts stored here
|   |-- static/
|   |   `-- static web assets stored here
|   |-- templates/
|   |   `-- HTML templates stored here
|   |-- admin.py
|   |-- apps.py
|   |-- forms.py
```

```
|   |-- models.py
|   |-- tests.py
|   |-- urls.py
|   `-- views.py
|-- media/
|   `-- images/
|       `-- database images stored here
|-- db.sqlite3
|-- index.js
|-- manage.py
|-- README.me
`-- requirements.txt
```

System Functionality

There are four primary components to consider when breaking down the functionality and interactivity of the system as a whole: the Electron client, the Django web-app, the database, and the Python feature scripts. A high-level view of the system's functionality is shown in Figure 38 below.

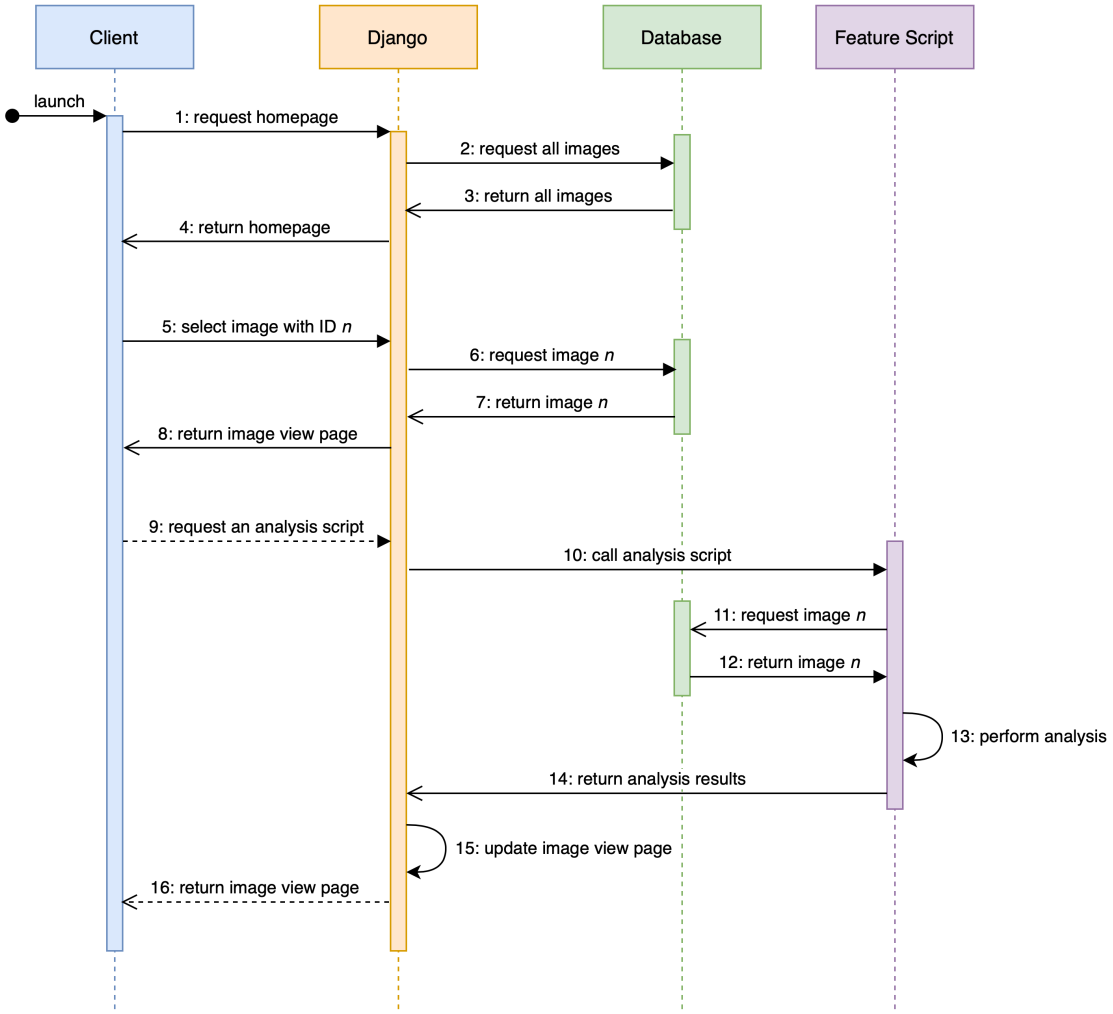


Figure 38: System context diagram showing the interactions between components during use.

Handling of User Navigation

Upon application launch, Electron launches a window pointing to `http://localhost:8000`, as defined within its configuration file `index.js`.

The primary Django application is configured to use the settings defined within the `imageupload` sub-application for all normal site URLs, so upon launch, the sub-app's URL configuration file `urls.py` is consulted to determine which page to serve to the user. Here, the application's URL patterns are defined by the `path(route, view, name)` function. The `route` argument corresponds to the web address path appended to the base URL, and the corresponding view specified by the `view` argument is returned to the client as output. As an example, the URL path followed upon application launch would have a blank route argument, and is given as `path("", views.index, name="index")`. This tells Django that when there is no route appended to the base URL, the returned content is defined by the `index` view.

Application views are defined in the `views.py` file and each takes the form of a Python function and returns the formatted HTML template file for that specific view. For instance, the `index` view which is loaded upon launch returns `render(request, "home.html", {"data" : data})`, where `data` is defined within this view function to contain all user-uploaded images which are stored in the database. In this case, the `data` variable is passed to the `home.html` template for processing with Jinja, and afterwards is returned to the client. The use of Jinja to loop through and display the photos stored in the `data` variable is shown in Figure 39 below.

```
{% for d in data %}
<div class="col-md-4">
    <div class="thumbnail">
        <a class="d-block mb-4 h-100" href="{% url 'image_view' image_id=d.id %}">
            <div class="thumb_img">
                <img class="img-fluid userimage" src="{{ d.photo.url }}" style="width:100%">
            </div>
            <div class="caption center-block">
                <center><p>{{ d.photo.name|basename }}</p></center>
            </div>
        </a>
    </div>
</div>
{% endfor %}
```

Figure 39: An example of Jinja templating on the home page.

Upon selection of an image, as shown above, Jinja is used to create a URL string of the form `/image_view/n` for a photo with ID `n`, and this is the URL path to which the user is taken.

This is the process for any URL accessed by the user, and the required data is passed between views and templates using this discussed method.

Handling of Script Analysis

After selecting an image to analyse from their home page, the user is taken to the image view page where they are able to select which analysis method they wish to run.

```
$("#run_script_reverse, #run_script_ghost, #run_script_plate, #run_script_metadata, #run_script_roadsign").click(function() {
    var scriptType = $(this).attr('id');
    var resultContainer = $("#" + scriptType.replace("run_script_", "results_"));

    $("#loadingModal").show();

    $.get("/" + scriptType + "/", { image_id: image_id }, function(data) {
        processScriptData(data, resultContainer, scriptType);

    }).always(function() {
        postScript()
    });
});
```

Figure 40: The AJAX method used to call the specified analysis script.

Shown above in Figure 40, scripts are called using AJAX `$.get()` requests to the specified script view. The script view functions retrieve the image's file path and pass this to the required script. When the results are returned, the script view uses Django's `render_to_string(template, data_dict)` function which is a variation of the aforementioned `render()` function but stores the processed HTML template as a string. This HTML response string is then returned to the original GET request along with the analysis data returned from the script. This process is shown in Figure 41.

```
def run_script_reverse(request):
    filepath = get_image_path(request)

    from .scripts.ReverseImageSearch import reverseImageSearch
    words, suggestions = reverseImageSearch(filepath)

    html_content = render_to_string("results_reverse.html", {"words":words,"suggestions":suggestions})
    response_data = {
        "html": html_content,
        "words":words,
        "gps_data":suggestions
    }

    return JsonResponse(response_data)
```

Figure 41: The view function used to call the reverse image search analysis script.

The use of AJAX to update the user's page with the analysis results *asynchronously* is very important because it ensures that any previous analysis results obtained by the user are not lost when they run a new script. This would be the case if asynchronous requests were not supported due to the necessity to reload the user's page with each request.

Finally, as Figure 42 shows, the HTML response string and analysis data are then processed within the `processScriptData()` function. Any GPS location data returned by the script is added to the map and to the `gps_data` array for later use during confidence ranking, and any country data is added to the `country_data` array for the same purpose.

The HTML `<div>` which corresponds to the script is then updated asynchronously with the script results, and classes are added to HTML elements in order to update the CSS styling to reflect that the script has been run successfully.

```
function processScriptData(data, resultContainer, scriptType) {
    if (data.gps_data && data.gps_data.length > 0) {
        data.gps_data.forEach(function(x){
            gps_data.push([x['name'],x['lat'],x['lng']]);
            addToMap(x['lat'], x['lng'])
        });
    }
    if (data.country_data) {
        data.country_data.forEach(function(x){
            if (x != ''){
                country_data.push(x);
            }
        });
    }
    resultContainer.html(data.html);
    $("#" + scriptType).find("i").addClass("run");
    $("#" + scriptType).find("span").addClass("run");
}
```

Figure 42: The processScriptData function which handles the resulting analysis template and variables.

This is the way in which all analysis scripts are processed when they are run individually, with the exception of the Confidence Ranking and Image Enhancement scripts. The execution of these scripts is similar but relies on a POST request rather than a GET request due to the quantity of data sent.

When the `Run All Operations` button is pressed, a button click is simulated for each of the analysis script buttons and they are executed in the manner described above.

UI Design & User Journey

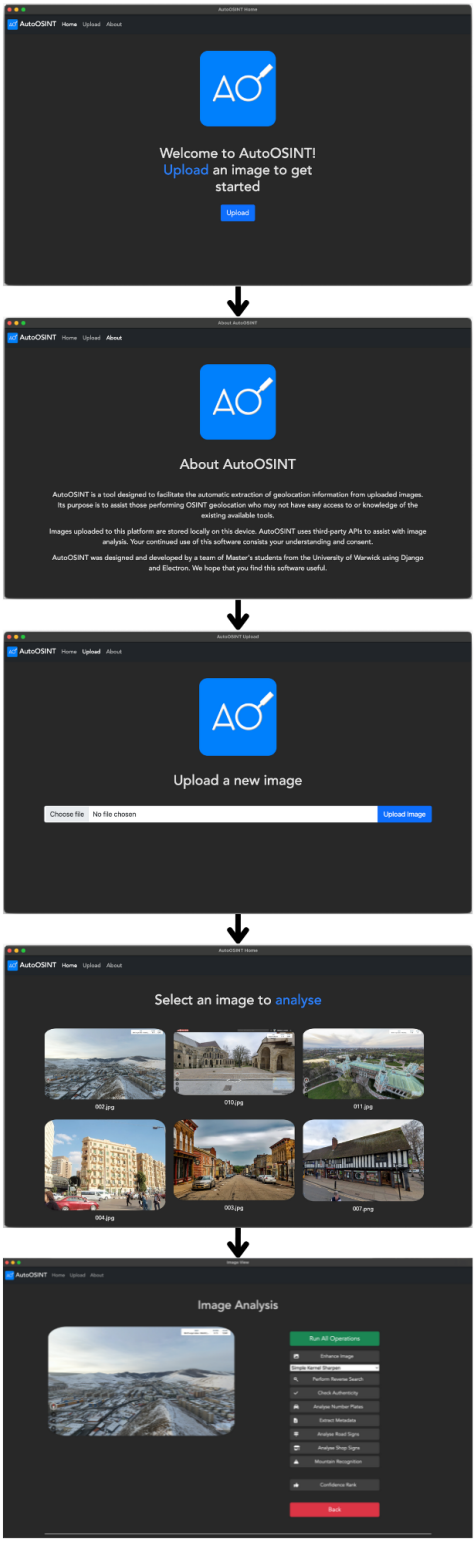
Figure 43 below illustrates the UI screens shown to the user at various stages of the user journey through the application. When the platform is launched for the first time, the user is prompted to upload an image to get started. They are also able to view the Image Upload and About screens via the navigation bar at the top of the page.

Once the user has uploaded one or more images, they are displayed on the Home page, where a prompt is given to select an image to analyse. Upon clicking an image, the user is taken to the Image Analysis page where they are able to select their desired analysis operations, or to run all analysis tools with one button click.

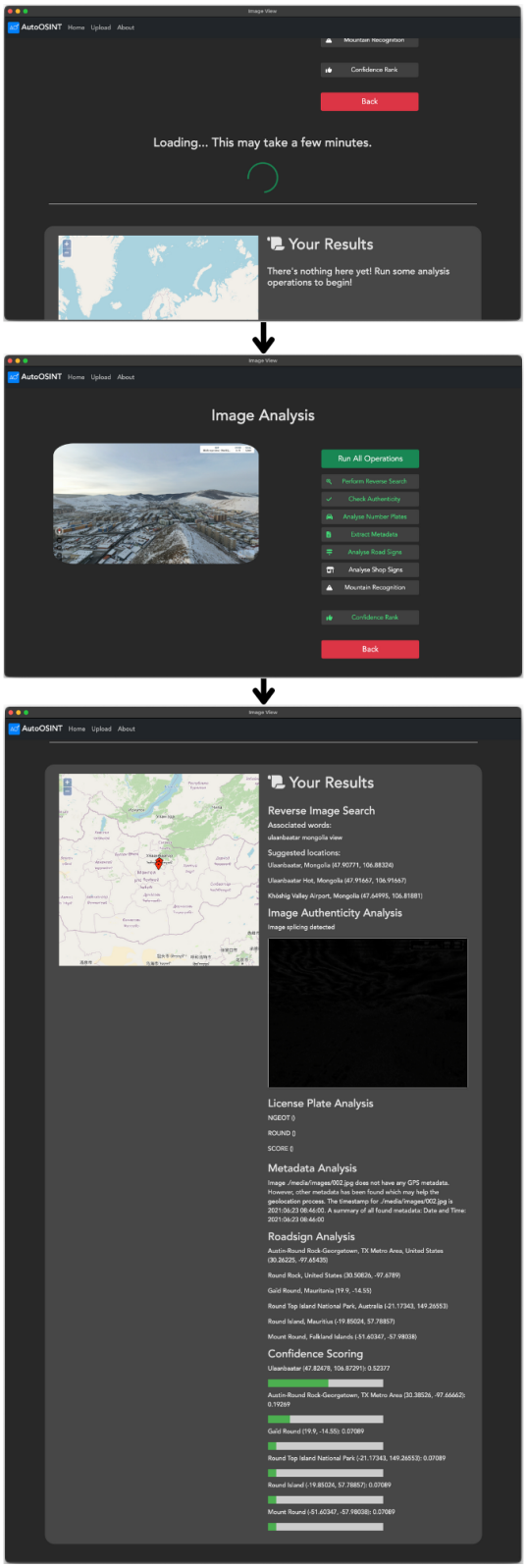
Once analysis has been selected, a loading wheel appears prompting the user to wait since some analysis tools take a few minutes to complete execution. Upon their completion, the buttons for the analysis options which have just been run turn green, providing the user with information about which tools they have not yet explored for the current image.

The results of the analysis are displayed further down the Image Analysis page alongside a map containing markers places on any GPS location data found. Useful raw output from the analysis functions is displayed alongside the map for the user to look through.

Following any and all analysis, the Confidence Ranking script may then be run which displays a graphical illustration of the certainty that the analysed image is located at any suggested location returned during analysis.



(a) Initial Home page, About page, Image Upload page, Home page after uploading images, and Image Analysis page



(b) Loading view during analysis, updated Image Analysis page to reflect successful analysis, and Analysis Results view

Figure 43: Complete application UI and user journey

Image Enhancement

Users a given several image enhancement operations to choose from: simple sharpen, simple blur, unsharp mask, bilateral sharpen, gamma adjustment or contrast enhancement[62]. The sharpening and gamma adjustment operations can be used to make details in the image clearer, and the simple blur can be used for the purposes of noise removal. Once the enhancement has been applied to an image, the new version is saved to a new PNG image file. Any OSINT features applied to the original image can also be applied to the enhanced version.

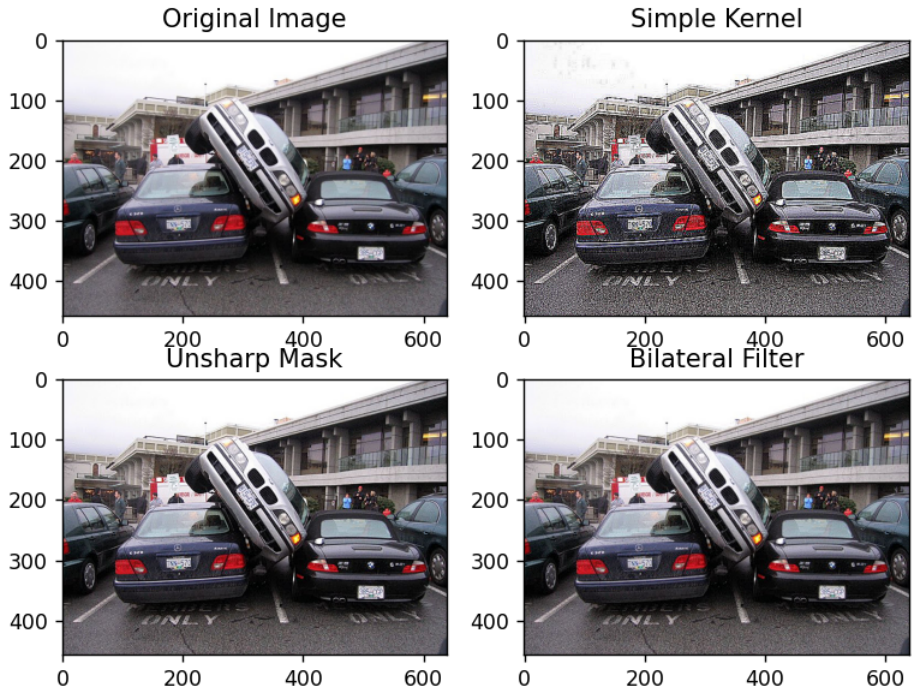


Figure 44: Result of applying sharpening operations on example image.

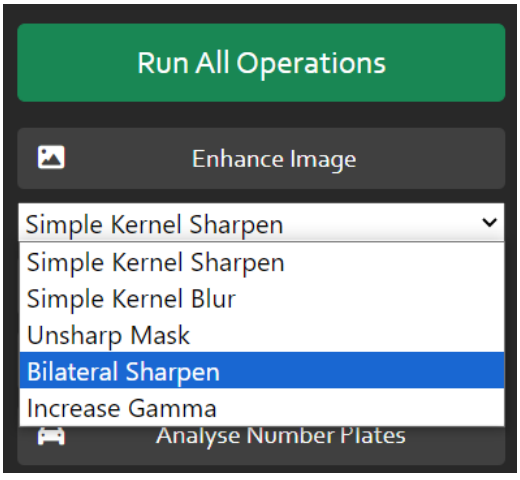


Figure 45: Graphical interface component allowing user to select image enhancement operation.

Reverse Image Search

Bing Visual Search

The first step in the reverse image search feature pipeline is the collection of a list of most similar images found on the web. The image search part of the feature is performed by **Bing Visual Search** which is provided as an API. The endpoint and header structure to send an image as a query request is as follows:

```
BVSENDPOINT = "https://api.bing.microsoft.com/v7.0/images/visualsearch"
HEADERS = {"Ocp-Apim-Subscription-Key":API_KEY, "Content-Type": "multipart/form-data"}
```

Sending the image query to Bing returns a paged JSON object containing various information regarding the query image. One of the fields is the "similar" field containing a list of the top 100 most similar images found and the related data (like caption, filename, source website, etc).

Image Similarity

Once a list of similar images is extracted, each is verified by calculating a similarity value to the query. This determined how reliable the information within the image is. A higher similarity metric meant more reliability and a higher "weight" to the contribution of the final location.

This is achieved using **SIFT**. Keypoints and descriptors are generated for the query image, as well as each candidate image. For each keypoint a SIFT descriptor is calculated: A 16x16 neighbourhood around the keypoint is selected and divided into 16 sub-blocks of 4x4 size. For each sub-block, an 8-bin orientation histogram is created. This produces a 128-bin keypoint/descriptor. A **FLANN (Fast Library for Approximate Nearest Neighbors) matcher** was then used to match said descriptors.

The matches are filtered through the **closest distance/second closest distance ratio**. If the distance to the closest match divided by the distance to the second closest match is greater than 0.75, the match is rejected. This eliminates around 95% of false matches while discarding only about 7% of correct matches, as per Lowe's paper [33].

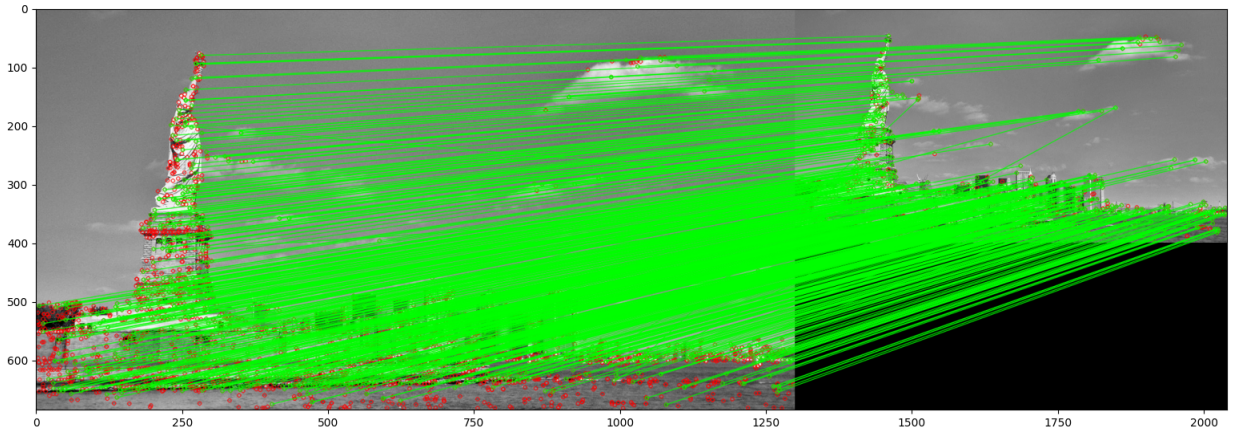


Figure 46: Similarity Metric with SIFT

The similarity of each candidate image to a query image can be inferred from the number of key-point matches. Since Bing is very accurate and discriminative with the suggested images, SIFT is an adequately effective similarity measure capable of recognising when 2 scene depictions are similar - especially considering the scale, rotation and illumination invariance.

At this stage, the feature has access to a collection of verified and ranked images, the remaining step is to extract the image information and suggest a location.

Histogram of Words

Individual words are extracted from the suggested images' **filenames and image captions**. These words are accumulated in a communal dictionary and their counts are retained. This is dubbed the **Histogram of Words**. The words are filtered against the English dictionary to remove common English words. The remaining words are very frequently indicative of the location or image topic. This can be observed in Figure 47.

To prevent anomalous/dissimilar images from skewing the results, a separate histogram is constructed, however, rather than storing word frequency, the number of matched descriptors in the word's "parent image" is added to the word's count. Therefore, closely matching images contribute more to the meaningful words. Therefore, the constructed histogram represents what the most similar images say about the location, as depicted in Figure 48.

Both histograms are used to determine which words are most reliable and describe the location best. These words are used in a search query to identify the location (see GeoNames below).

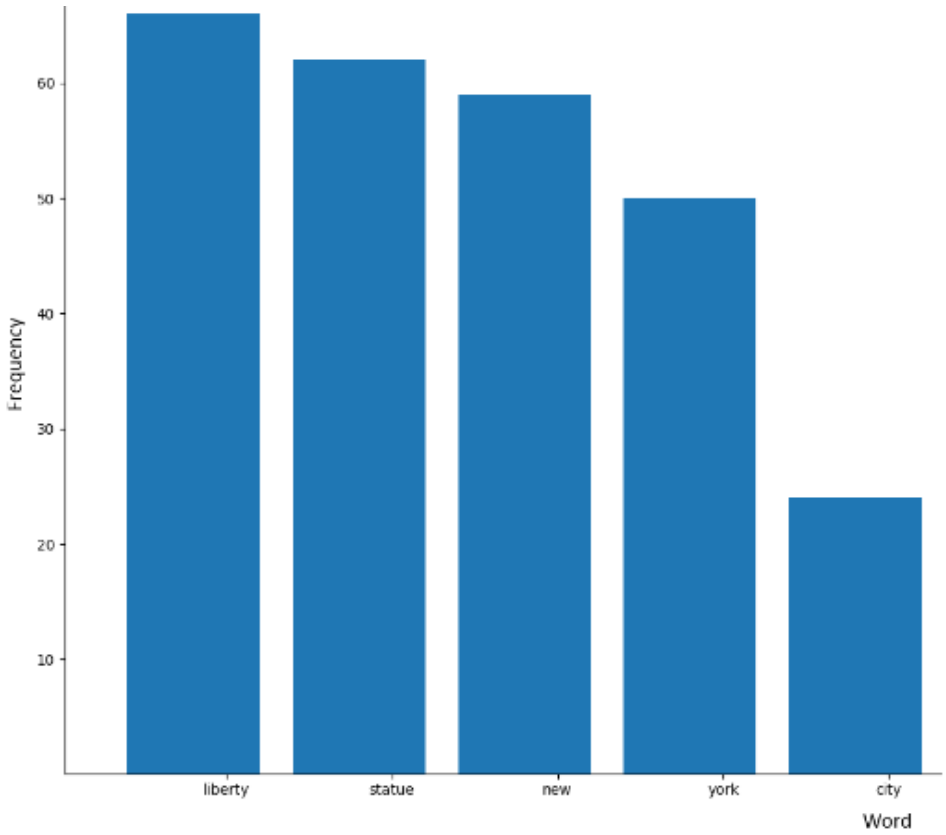


Figure 47: Frequency Word Histogram with SIFT

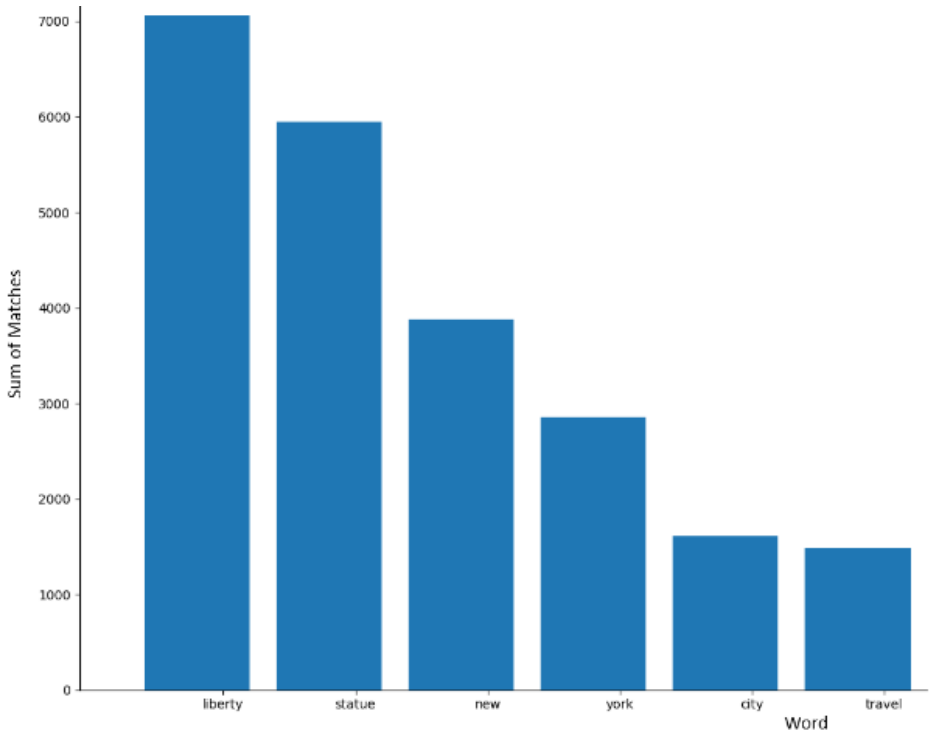


Figure 48: Words Weighted by Matches Histogram with SIFT

GeoNames

Once the words have been extracted, they are passed to an online database of locations known as GeoNames. The API looks as follows:

```
GNENDPOINT = "http://api.geonames.org/searchJSON"
GNUSERNAME = "LStanaszek"
```

The words are submitted as a get request with some additional refinement (the country is set if a country name is detected in the words). GeoNames then returns a JSON object containing a list of possible locations. The object also includes information such as coordinates which are captured.

Metadata Extraction

The metadata extraction module makes use of the concepts and techniques described in the metadata extraction section of the literature review. If there is no metadata attached to the input image, the results section simply displays a message telling the user this fact. If there is attached metadata, the image is first checked for GPS coordinates and then any other relevant attached information. The image's description metadata is checked for named locations, language and timestamps. This module also implements the user input clues feature due to overlapping processing requirements with the description metadata field. The retrieved information from this module is returned as a dictionary and the text output is displayed in the Results section of the image analysis page of the app.

Following the methodology of Acharya et al.[15], the metadata extraction tool was implemented in Python using the Pillow library to process the Exif data. Once the input image is loaded, Pillow's 'getexif' method processes all the available metadata and converts it from the complicated, nested image storage format into a more usable one. Now, the metadata is separated into their respective

subsets which makes it easier to access the individual metadata fields. Then, the metadata tag names are used to extract useful geolocation information for further processing; these include the GPS, camera device, creation date and time, and image description data fields.

GPS Metadata

The metadata extraction tool prioritises GPS metadata as this would provide immediate and exact image geolocation. To search for present GPS metadata, the image is reloaded using the 'piexif' module, instead of the previous PIL library method, because 'piexif' offers better methods for handling GPS metadata to suit the needs of this project. Within the Exif data, the latitude and longitude metadata are grouped into a subset under the tag 'GPS'; 'piexif' is able to understand the individual GPS tags directly, providing a simpler process to parse through the GPS subset for the desired information. Even though there are roughly thirty GPS metadata tags, giving additional information about the GPS satellites and receivers involved, for this project, only the 'GPSLatitude', 'GPSLongitude', 'GPSLatitudeRef' and 'GPS-LongitudeRef' values were extracted as these would be the most relevant and helpful for geolocation.

Once the GPS metadata has been extracted, it is then converted from degrees, minutes, seconds format into decimal degrees format because the decimal degrees format is much more commonplace for mapping API systems. Following this, the coordinates are converted into their written address to increase user-readability as this format will be much more meaningful to the user than the numeric coordinates. To implement this, the 'Nominatim' class from the 'geopy' library was used to perform the reverse geocoding process. The 'Nominatim' class was a suitable choice because it returns detailed addresses including place and road names. It is also highly accurate as 'Nominatim' queries the open-source OpenStreetMap database which provides global coverage of coordinates and their readable addresses.

For images where there is incomplete GPS metadata, the readable address is not calculated, but the extraction tool will still store any available information and will convert to decimal degrees format if possible.

Description Metadata

The metadata extraction tool also processes other metadata fields. Although it is unlikely that these could produce an exact image location, they could still provide information to aid the overall geolocation detection system, particularly when there is no GPS metadata present.

If the 'ImageDescription' metadata tag is present, a series of text processing methods are applied to search for useful information. First, the tool performs language detection to provide some insight into the possible countries the image creator was from. The 'langdetect' library provides the straightforward 'detect' method, based on Google's language-detection library, which uses statistical analysis to identify the used language.

Then, the spaCy natural language processing library is used to parse through the description string for any mentioned locations ('LOC'), like mountains and rivers, or geopolitical entities ('GPE'), like countries and cities. If detected, these will aid the geolocation process when there is no GPS metadata, because they will narrow down the possible global regions, especially if countries or cities are mentioned. Additionally, the description string is parsed for dates ('DATE') and times ('TIME') in case these could provide seasonal insights on the image's country, as mentioned in the metadata extraction section

of the literature review.

Currently, this process can only be used if the detected language is English, because spaCy requires language models to be downloaded before use, impacting the user's computer storage. It was decided that the benefit provided by this feature does not outweigh this storage drawback, and, as such, it can only be called if the description string is in English. spaCy was chosen because it has an intuitive implementation which is fast and efficient. spaCy is accurate on both small and large text extracts, unlike many other libraries which often require a large text extract for reliable language prediction; it is expected that the description metadata will be short in length, meaning that spaCy would be an appropriate implementation choice.

During this design process, the possibility of using the camera metadata for geolocation clues was investigated. Certain camera makes and models are only sold in certain parts of the world. So, similar to how Toevs[17] used the 'BeautifulSoup' library in their metadata extraction process for searching websites, information could be gathered about the camera manufacturers to determine which more likely countries for the image's location. However, the implementation for this was extensive due to the internet searching component and subsequent data analysis; it was unlikely that any meaningful information would be discovered, especially as any gathered could not be significantly relied upon as people travel with their cameras. Thus, this feature was not implemented as it would not be worth the additional computation time.

Number Plate Recognition

Automated number plate recognition (ANPR) is used as one method to narrow down the potential geographical area of the input image. For example, an image containing many cars registered in the United Kingdom is more likely to have been taken in the United Kingdom. However, because cars can travel between countries, the results of this tool are not guaranteed to lead to an accurate prediction. Therefore, it is important to have other prediction tools available to help increase the likelihood of a correct prediction.

The ANPR module first applies a series of morphological operations on the input image to produce two binary masks: one representing the rectangular light-coloured image regions (including number plate backgrounds) and another containing dark edge-dense regions (including number plate text). The masks are then combined with a binary AND operation and the image regions are extracted and stored as contour vectors. The contours are filtered by aspect ratio so that only contours with a similar aspect ratio to a number plate are kept.

Once the number plate regions are extracted, the Tesseract engine is used to perform optical character recognition. The text extracted from each region is looked up in a small locally-stored array: for each row of the array, the first value specifies a country name. The second value is a regular expression which describes every valid registration number for a car registered in the specified country. For each string, the array is iterated through and all countries whose corresponding regular expression accepts the string are added to a list.

```
,   country,        regex,
0,  United Kingdom, "[A-H,J-P,R-Y]{2}[0-9]{2}[A-H,J-P,R-Z]{3}",
1,  Russia,         "[ABEKMHOPCTYX][0-9]{3}[ABEKMHOPCTYX]{2}[0-9]{2,3}",
2,  France,         "[A-H,J-N,P-Z]{2}[0-9]{3}[A-H,J-N,P-Z]{2}",
3,  Italy,          "[A-H,J-N,P-Z]{2}[0-9]{3}[A-H,J-N,P-Z]{2}",
```

Figure 49: Part of the contents of the CSV file containing the number plate lookup dictionary values.

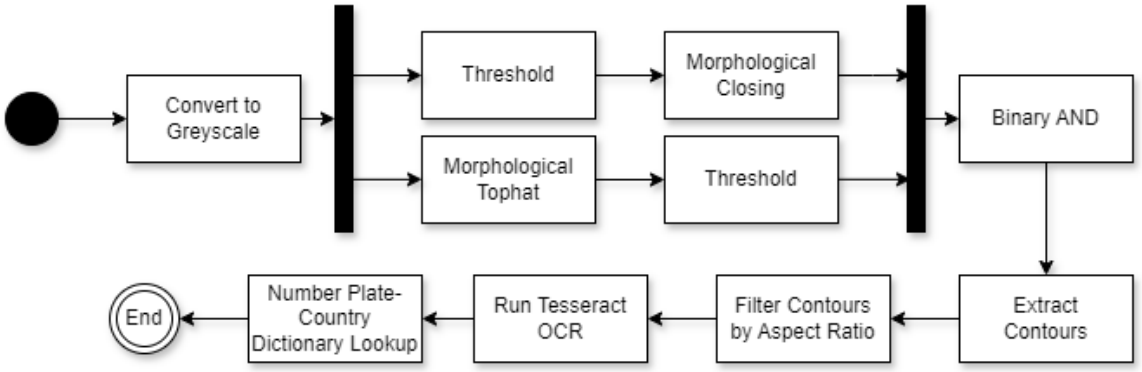


Figure 50: Activity diagram for the number plate recognition and geolocation module.

The entire ANPR process is repeated 10 times with various kernel sizes during the morphological operations. The results of all 10 iterations of the algorithm are aggregated using a counter, then country predictions are ordered descending by the number of occurrences. This step is intended to make the system more reliable, as a prediction which is made in all 10 iterations is more likely to be correct than a prediction which is made in just one iteration. Predictions often differ between iterations due to different kernel sizes detecting potential number plate regions at different scales.

Road Sign Analysis

The Road Sign Analysis module exists to extract any text information from an input image which can be used for geolocation. It uses optical character recognition to identify all text, then searches for geographical place names in the extracted text. This can be an efficient way to narrow down potential locations very quickly, for example if an image contains a road sign pointing to a specific town, city or street. However, because many different geographical locations share the same names, further analysis is often needed to identify an exact location. Furthermore, many road signs are located several kilometres away from the location they are pointing to, so this prediction method tends to be somewhat imprecise.

Optical character recognition is performed by making a call to the Google Vision API for text detection. The number of free calls a user is allowed to make to Google Vision is limited and it was decided that they would be most useful in this module because Google Vision's text detection function is more accurate and is able to perform OCR over an entire scene with a single API call. This is something that free alternatives (such as Tesseract OCR) are not capable of.

The API call returns a list of character strings extracted from the input image. These are then cleaned and tokenised using functions from the SpaCy natural language processing library to make the data easier to parse. The string tokens are then filtered using SpaCy's named entity recognition[63] to extract only geographical place names. For each extracted place name, a call is made to the Geonames

API to find all geographical locations which match the given place name. The full name, country and GPS coordinates of the first five results are stored for display on the map interface included in the app's results section.

Figure 51: Overview of the steps involved in the road sign analysis module.

JPEG Ghost Detection

Although verifying image authenticity is not the main focus of the AutoOSINT app, it was decided that an automated JPEG ghost detection system would be included in order to widen the potential use cases for the system. To detect whether an input image was spliced with another image, the input image is repeatedly compressed at different compression levels. Any image regions which fade in or out based on compression level are considered to be 'ghosts', indicating a potential splicing forgery.

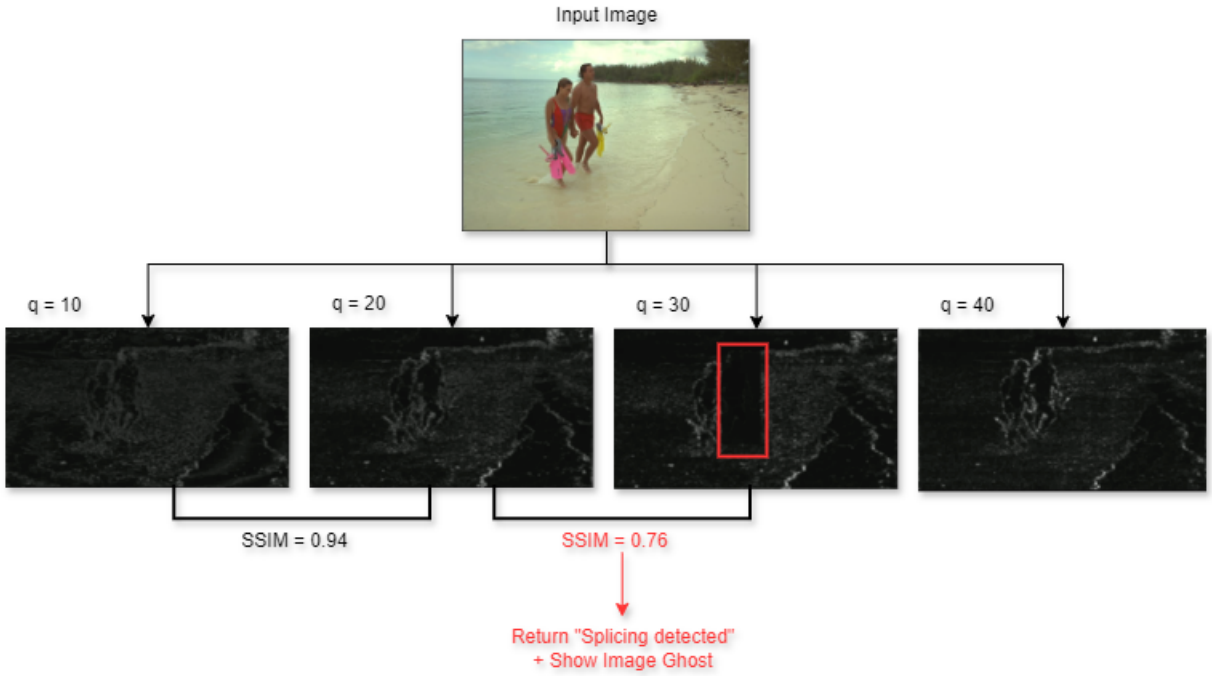


Figure 52: Illustration of flow of JPEG ghost detection system.

JPEG ghosts are detected automatically by sequentially comparing the pairwise SSIM of the compressed image frames. If the SSIM of any two frames is below a threshold value of 0.85, the image is considered to be spliced and the ghost image at the low-similarity frame is displayed in the results section under the input image, along with a message notifying the user that image splicing has been detected.

CrossModal Mountain Geolocation

The project integrates a mountain geolocation feature, based on the researched Crosslocate project. For reasons discussed below, the feature is implemented from scratch based on data and ideas generated in the Crosslocate paper.

Integration of existing solution

Initial efforts focused on integrating the publicly available Crosslocate program found on GitHub. However, 2 serious obstacles hindered the integration of the existing solution:

1. **Missing project components**: The project authors transparently stored all required code on a public GitHub repository. The dataset and its associated metadata should have been stored on the project website. Whilst the image data was available for download from the website, the metadata was incomplete. Although not critical to project completion, the metadata was important in understanding the dataset structure and would speed up the dataset preparation process.

   Fortunately, the group was able to make contact with the authors of the project, enquiring for permission to base a solution around Crosslocate as well as asking for assistance with obtaining the metadata. The request was met and the metadata was uploaded to the original target location. By securing this data and gaining access to the complete Crosslocate dataset, it is estimated that a week of development time was saved.

2. **Dependency incompatability**: Crosslocate was built on Python 3.7, however, AutoOsint is based on Python 3.10. A number of the used libraries were incompatible and dependency version requirements could not be satisfied. It was deemed easier to recreate the Crosslocate program in a newer version of Python and with PyTorch than to downgrade the Python version used by the project and update existing features.

Therefore, the project assumed the task of creating the components to train the Neural Network from scratch, as well as creating packageable code that can predict query image coordinates with trained model weights.

Dataset Preparation

The dataset is obtained from the Crosslocate project website, however the process is described below:

It is important to understand and differentiate between the vocabulary used in the following section. A dataset in this project is a collection of queries (photographs) and a database of locations (in the depth map modality). As mentioned in the previous section, a preprocessed and complete Alps dataset used to create the Crosslocate model was available on the Crosslocate project webpage. The hardware limitations permitted that a model be trained on the smaller "Sparse Dataset". This dataset was derived from another dataset called the GeoPose3k dataset. This is a dataset of 3111 360°panorama images with associated geographic data like WGS coordinates and elevation.

The "Sparse dataset" query set is made up of 3111 photographs from each GeoPose3K image sampled at a specific orientation (yaw) - specifically selected to depict a characteristic mountainous scene with a 60°FOV. The depth map database was generated from the 3111 locations found in Geopose3K. A 3D scene is extracted from a DEM at each coordinate set and depth maps with 60°FOV are generated at 30°yaw intervals. This means at each query location, 12 depth maps are generated, making a database

of size 37332 (3111 x 12). The query photographs and database depth maps are saved in separate folders with attached .csv files containing information about the location of the source image.

The feature is called "cross-modal" mountain geolocation because the model described below learns to recognise the similarity between a photograph and a depth map. These are two different representations/modalities of a scene.

The first obstacle encountered during the development stage was conforming with storage limits. When uncompressed, the dataset takes up 16GB, on the other hand, the university DCS systems provide students with a maximum of 30GB of storage - 20 GB was already occupied with previous projects. This means there was insufficient space to store the dataset. Fortunately, much of the stored data was no longer required and with a storage system clean, enough storage was freed to accommodate the dataset with a margin of freedom.

The next step in preparing the data was to load the queries and database into memory. Another hardware limitation was discovered at this point - there was insufficient memory to load the full dataset. The solution to this was to use a slightly different split between training and test data compared to the Crosslocate selection - no validation set was included as the validation stage to select hyper-parameters was already performed by Crosslocate, and the same hyper-parameters were utilised. The used split can be seen below:

Counts	Train set	Test Set	Total
Query set	1400	500	1900
Database	16800	22800	22800

Table 3: Training and Testing Data Split

At this stage, queries were also resized to 500x500 photographs to reduce memory usage.

Data Augmentation

In an attempt to diversify training data, prevent over-fitting and improve the feature's ability to deal with various image qualities, the data was augmented with common image manipulation techniques. Naturally, photographs and depth maps are different modalities so different augmentations are applicable in either case. This required 2 separate augmentors. The augmentations used (taken from the Crosslocate GitHub) and their degree can be observed in Figure 53. For all augmentations, a default probability of 0.5 is used.

```
dbAugmentator.activateShiftAugmentation(maxHorizontalPercentage=0.05, maxVerticalPercentage=0.025)
dbAugmentator.activateRotationAugmentation(maxAngle=5)
dbAugmentator.activateFlipAugmentation()

qAugmentator.activateShiftAugmentation(maxHorizontalPercentage=0.10, maxVerticalPercentage=0.050)
qAugmentator.activateRotationAugmentation(maxAngle=5)
qAugmentator.activateFlipAugmentation()
qAugmentator.activateBrightnessAugmentation(maxMagnitude=40)
qAugmentator.activateHueAugmentation(maxMagnitude=10)
qAugmentator.activateSaturationAugmentation(maxMagnitude=40)
qAugmentator.activateContrastAugmentation(maxPercentage=0.3)
qAugmentator.activateBlurAugmentation(maxSigmaPercentage=0.003)
qAugmentator.activateNoiseAugmentation(std=10)
```

Figure 53: Augmentations employed by the query augmentor and the database augmentor

Triplet Generation

The Triplet Generator was a class designed to be called in the main training loop to return a new batch of training examples - called "triplets" - with every call of getBatch().

A triplet is a collection of an anchor (query photograph), a positive example (a depth map depicting a similar scene to the query) and negative examples (depth maps not matching the query location) images. Counterintuitively, triplets are not made up of 3 components but 7: 1 query, 1 positive, and 5 negatives. The selection process follows a set of rules to obtain the most useful triplet for each query:

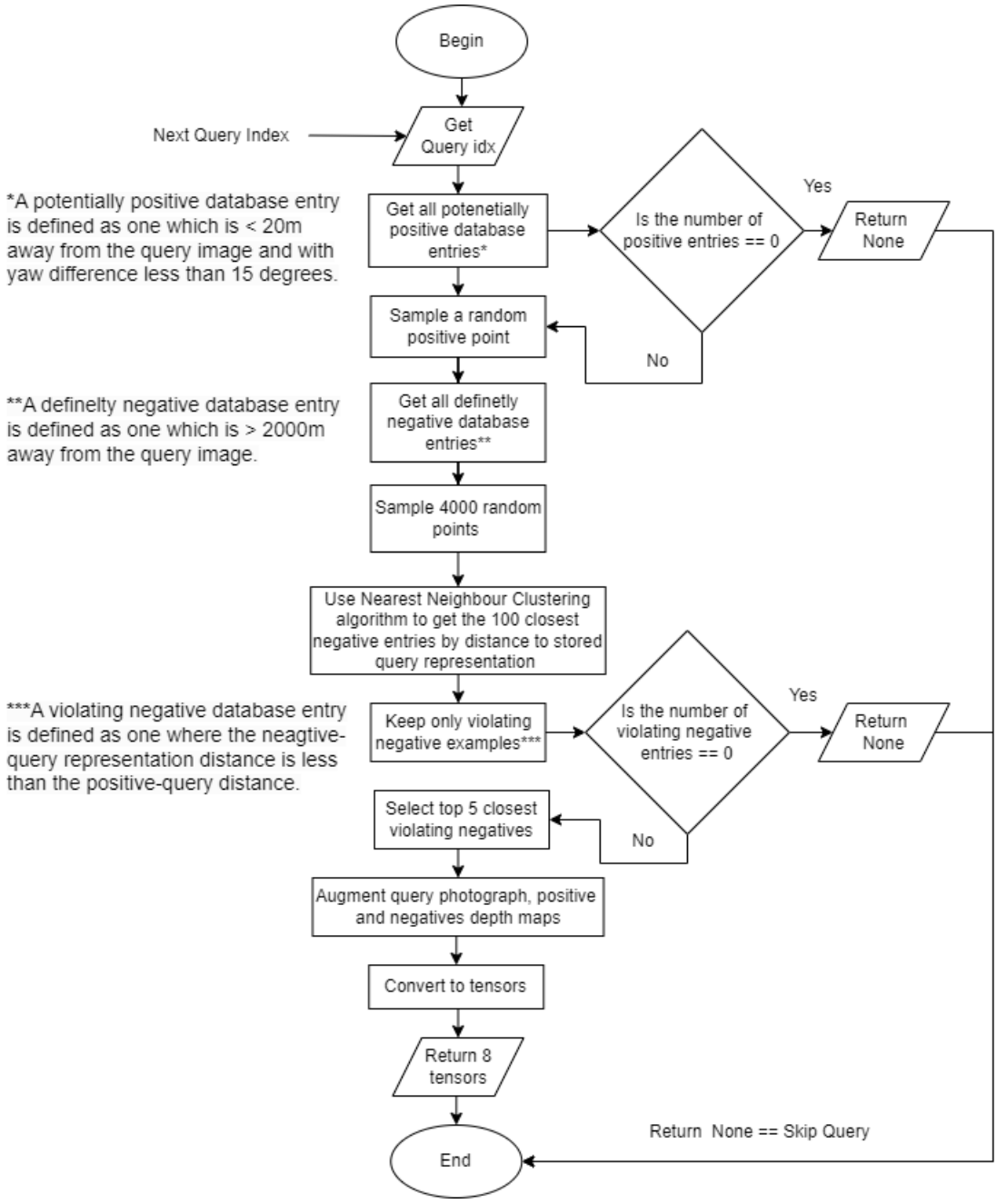


Figure 54: Triplet Generation process

The above process is looped for each query to complete an epoch. To increase the training speed, batches of triplets would be formed and processed at the same time. So calling getBatch() will return a

tensor of shape (batchSize = 3, numberOfExamplesInTriplet = 8, numberOfFeatures=512).

The idea behind triplet loss is to select negatives with descriptors similar to the query descriptor (closer than the distance to a positive example). This is anomalous, as negative examples do not depict the query location and thus the distance between the descriptors should be greater. Selecting this combination of positive and negative examples for the anchor is expected to produce a non-zero loss value which should correct the weights and produce "better" representations where positives are more similar to queries than negatives.

It is important to highlight the 2 terms in the above diagram - potentially positive and definitely negative. Firstly, it must be reiterated that the database depth maps are generated based on the location and orientation provided with the geoPose3K dataset, not from the images themselves. The DEM is a sample of the environment so the exact location of the image may not be available and there may be a few meter disparity between the photograph and depth map depiction.

Secondly, the DEM used to generate the depth maps is a DTM, which means not all objects on the surface are included in the DEM. There is no guarantee that a query photograph has a full view of the mountain range as it can be blocked by various objects including trees, buildings, people and other entities which the DTM will not account for. A potentially positive depth map is thus defined as one within 20 meters and orientated within a 15°yaw of the query. However, a query could be blocked or with a slightly different perspective to a positive depth map - hence the "potentially ". An example of both mentioned phenomena can be seen in Figure 55. People and clouds block the ridge line and the camera perspective varies.

Similarly, definitely negative means database candidates where there is no chance of the same location being depicted. This can be guaranteed more easily as it suffices for the geographical distance between a query and candidate to exceed a certain value, the selected value by both Crosslocate and this project was 5km.

Figure 55: Example of a positive database entry, however not depicting the exact scene.

Neural Network Architecture

The Convolutional Neural Network (CNN) utilised by this project replicates the Crosslocate design in 100%. As covered by the paper, each component has been selected with reason and direction, and an experiment (backed by evidence) revealed that changing any of the components leads to a significant drop in performance.

The network architecture can be depicted in Figure 56. The components are as follows:

- **Input**: The input is a 3-channel 500x500 image. The network layer accepts both photograph images and depth maps; when 1-channel depth maps are used, the depth map is duplicated to the 3 channels.
- **Convolutional Encoder**: The following layers in the neural network follow the VGG16 neural network architecture which is 5 layers of 2 or 3 2D convolutions with ReLU, followed by max pooling operations. This resulted in 32x32x512 image feature embeddings.
- **Descriptor Extraction**: The 32x32x512 feature embedding is passed to an L2 channel-wise normalisation layer, followed by a global max pooling layer for each channel. A final normalisation is applied.
- **Output**: The network returns a 512-feature vector.

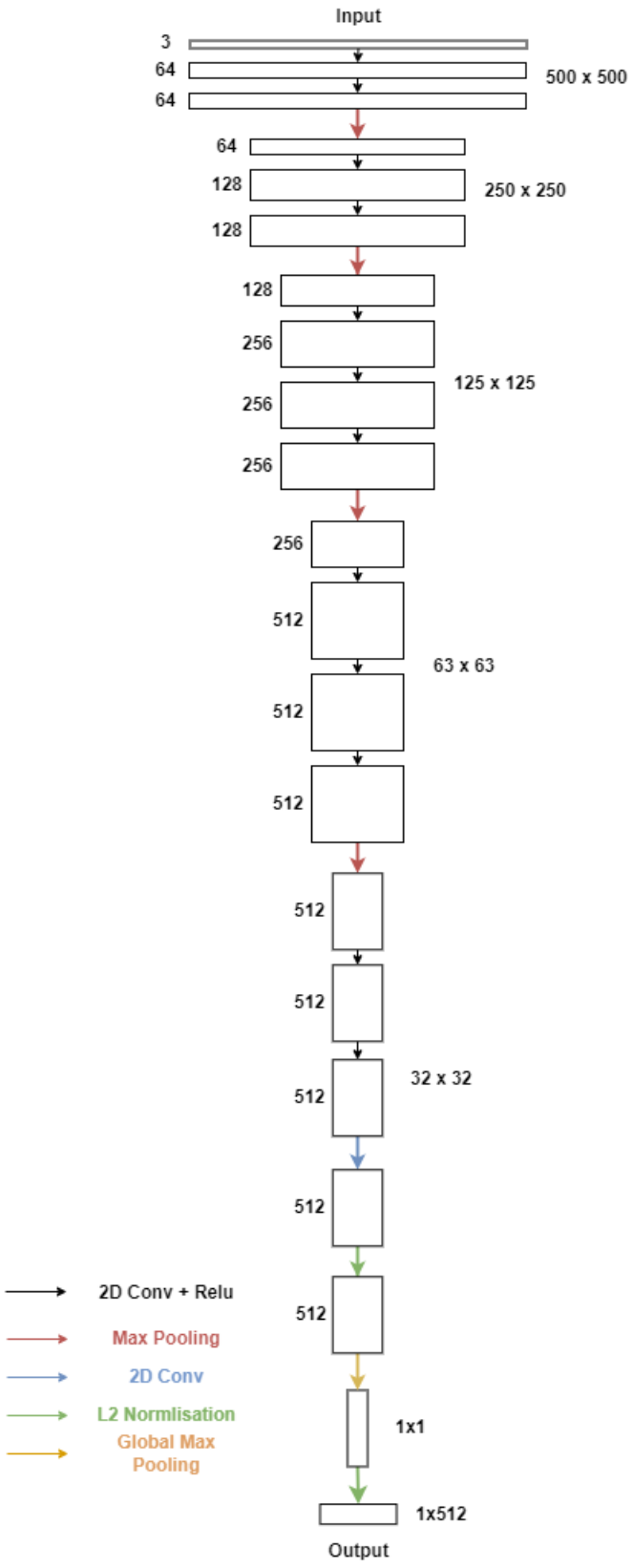


Figure 56: CNN Architecture

Training

Initially, the network weights are initialised with pre-trained weights trained on ImageNet.
The Training Process (for each epoch):

1. Get a triplet batch (of image tensors) from the triplet generator.
2. Pass all the augmented image tensors to the neural network.
3. Calculate triplet loss using the equation:

$$L_{\text{triplet}}(A,P,N) = \max(0, d(A,P) - d(A,N) + \alpha) \tag{4}$$

4. Optimise the weights using the ADAM optimiser.
5. Every 250 triplets, update the stored query and database representations. As explained in the triplet generator section, the triplets are formed by selecting database entries based on the distance of the representations to the query representation. Ideally, the representation of each entry and query should update each batch. However, this would be a very lengthy process as the representations of the full database and query set would have to be recomputed every batch. This is simply infeasible. Instead, the representations are stored and updated every 250 triplets which reduces training time exponentially.

The weights are saved after each epoch for incremental training.

Training Parameters:

- Triplet Generator Parameters:
  - Batch Size = 3
  - Number of Negatives = 5
  - Violating Negative Margin = 0.2
- Flow Control Parameters:
  - Epoch Number = 100
  - Descriptor Update Frequency = 250
- Learning Parameters:
  - Loss = Triplet Loss (with a margin of 0.1)
  - Optimiser = Adam (with a learning rate of 0.00001)

Prediction

Once a model has been trained and a set of weights selected (with some basic validation testing), the model is run on all database images and each of the database images is converted into a descriptor. These descriptors are stored in a CSV along with geographic information (like coordinates and yaw).

When a query is uploaded for geolocation, the model weights are loaded, the image is resized and passed into the neural network. The descriptor that is output is then passed into a KNN algorithm. and the closest K database entry is returned. The K was selected to be 20.

Confidence Ranking System

The confidence ranking system combines the outputs of the individual semi-automated features to provide an easy-to-understand estimate of which geolocation predictions are the most likely to be correct. It takes into account the geographical similarity of different location predictions to each other as well as a weight given to each prediction based on a certainty automatically assigned to it.

First, all country predictions (predictions that state a country but do not specify a GPS location, produced by the number plate analysis module) are dealt with. Using Google Vision's Label Extraction feature, a list of five descriptive words is extracted from the input image. The similarity between these words and the words 'urban' and 'rural' are then calculated using SpaCy's vector-based text semantic similarity measure[64][48]. If the descriptive words more closely match 'urban', then a query is made to the OpenStreetMap API to generate a list of all cities in the predicted country, along with their GPS coordinates. If the descriptive words more closely match 'rural', the list will instead contain the names of all lower-level administrative regions (for example, in the United Kingdom this would correspond to all of the country's counties). The list of names is then fed to PIGEON for zero-shot classification. The predicted name and GPS location given by PIGEON is finally added to the list of GPS coordinates produced by the other automated features for confidence ranking. This processing step exists to convert all predictions to the same level of granularity, making the ranking process simpler.

Once all prediction data has been collected and processed so that it takes the form of a list of named GPS locations, the confidence system performs clustering on the points. The clusterign algorithm used is DBSCAN with Haversine distance used as the distance measure. DBSCAN is used because it allows us to identify only clusters of predictions with a minimum density. The number and size of clusters depends on the prediction data rather than any hyperparameters, and the clustering is robust to outliers which are likely to be produced by some of the automated features (such as the road sign analysis module, which can give incorrect predictions due to different geographical places sharing the same name).

After clustering, the clusters are ranked by their size (the number of points they each contain). A softmax function is applied to the cluster sizes to give a normalised probability value for each cluster prediction. The ranked clusters are finally displayed in the app's results section, with each cluster's softmax value being given next to the GPS coordinates of the cluster's centre and the name of its most central point prediction.

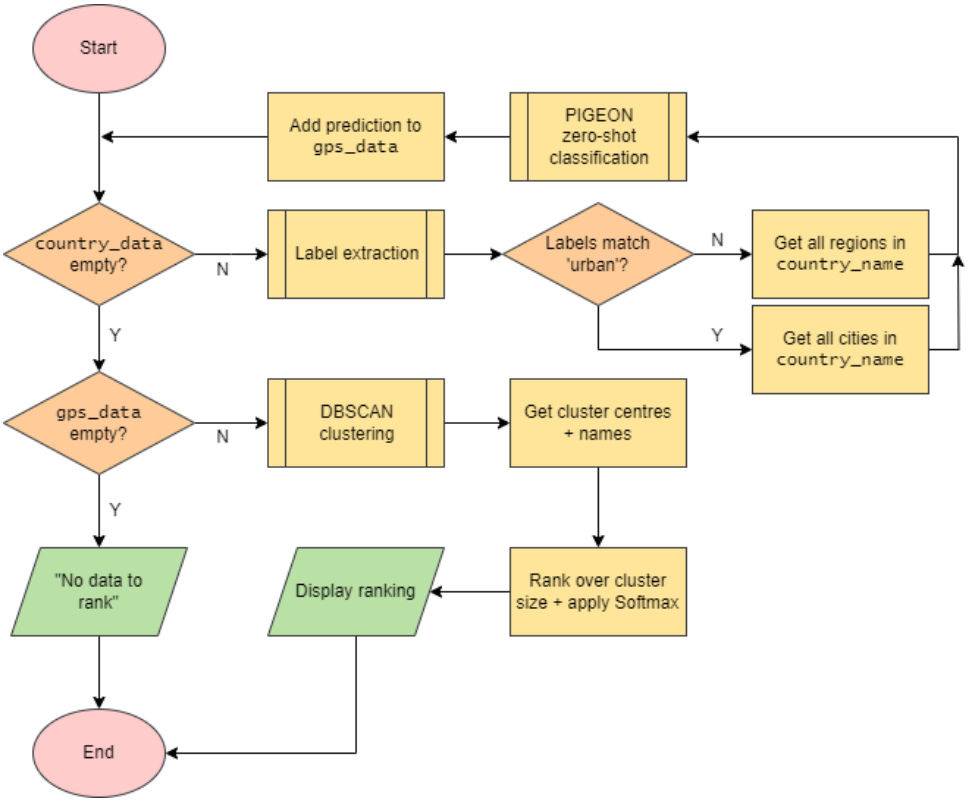


Figure 57: Flowchart outlining the confidence ranking process.

The visual output of the confidence ranking system is an ordered list of location predictions. For each prediction there is a named GPS location, as well as a probability score determined by a Softmax function. Due to the nature of some of the tools (such as the metadata extraction tool, which should always be correct except in cases where metadata was altered), it is difficult to assign a meaningful numerical score for the probability or certainty of each prediction. Therefore, it was important to include the ranking as an additional indicator of prediction quality for the user. This way, a user can choose to conduct further research into location given by the first and most-likely prediction. If the top prediction is later found to be incorrect, the user can consider the alternative predictions given lower down in the ranking. This should minimise the time spent exploring locations which are unlikely to be correct given the input image.

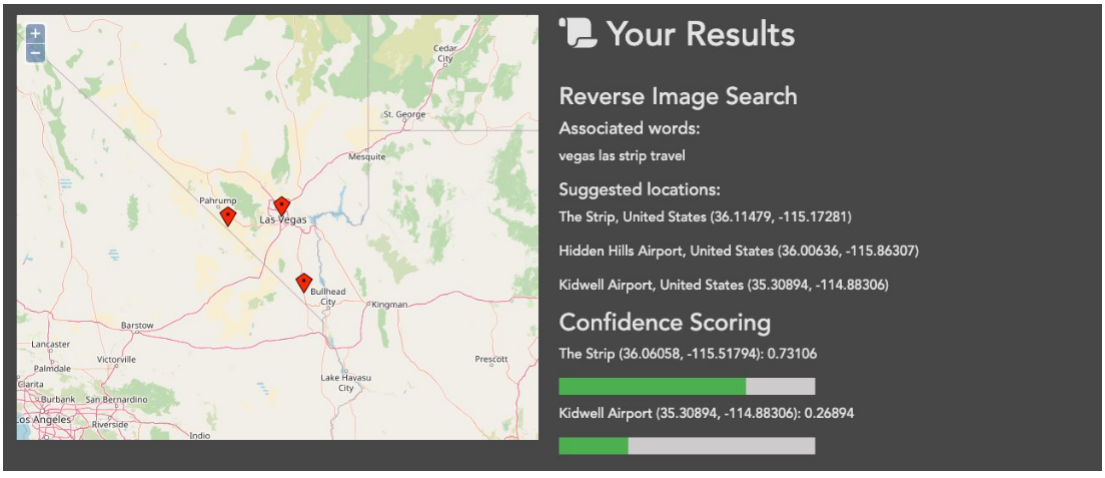


Figure 58: Confidence ranking interface output for an example input image.

Testing

Acceptance Testing

Acceptance tests verify whether a system satisfies the requirements outlined in its initial specification. The entire application is run whilst testing, with a focus on replicating user behaviours and interactions with the system to determine whether the requirements have been met during use in production.

The following tables detail these final acceptance tests to verify whether the project requirements have been met, including *Must Have*, *Should Have*, *Could Have*, and *Won't Have* requirements. As detailed, an Agile Test-Driven Development approach was adopted for system and feature design, and unit testing was therefore performed during development to drive the implementation process in the correct direction.

Functional Acceptance Tests

Test	Spec Ref	Purpose	Input	Expected	Outcome
1	F1.1 (MH)	Check if the user is able to upload a PNG image	Select Upload tab, click 'Choose file', select PNG image file, then click 'Upload Image'	Selected PNG image visible in Home tab	Pass
2	F1.1 (MH)	Check if the user is able to upload a JPG image	Select Upload tab, click 'Choose file', select JPG image file, then click 'Upload Image'	Selected JPG image visible in Home tab	Pass
3	F1.1 (MH)	Check if the user is able to upload an image of size 1920x1080	Select Upload tab, click 'Choose file', select image file with size 1920x1080, then click 'Upload Image'	Selected image visible in Home tab	Pass
4	F1.2 (SH)	Check if the user is able to input clues to the system	Upload + select 'swansea.jpg', then enter 'swansea' in the clue text box	Features are informed to weight results containing 'swansea' as higher priority	Fail - feature not implemented before deadline
5	F2.1 (WH)	Test whether the system checks if an AI-generated image is generated by AI	N/A	N/A	N/A - deprioritised

6	F2.1 (SH)	Test whether the system checks if an non-AI-generated image is generated by AI	N/A	N/A	N/A - deprioritised
7	F2.2 (SH)	Test whether the system detects compression-based artefacts in a spliced image	Upload + select 'beach.jpg', then click 'check authenticity'	'Image splicing detected' + ghost image displayed under Results	Pass
8	F2.2 (SH)	Test whether the system detects compression-based artefacts in a legitimate image	Upload + select 'swansea.jpg', then click 'check authenticity'	'No image splicing detected' + difference image displayed under Results	Pass
9	F3 (MH)	Test whether the system is able to extract GPS and camera information from an image with metadata	Upload + select 'cottage.jpg', then click 'extract metadata'	GPS coordinates of image are displayed under Results, as well as the date and time it was taken, the make and model of the camera, and the address information.	Pass
10	F3 (MH)	Test whether the system is able to extract GPS and camera information from an image with no metadata	Upload + select 'eiffel.jpg', then click 'extract metadata'	'Image ./media/images/eiffel.jpg has no attached metadata'	Pass
11	F4.1 (SH)	Test whether the system is able to perform gamma correction on an image	Upload + select 'eiffel.jpg', select 'increase gamma', and then click 'Enhance image'	'Saved to ./media/images/eiffel_enhanced.png'	Pass
12	F4.2 (SH)	Test whether the system is able to perform contrast equalisation on an image	Upload + select 'eiffel.jpg', select 'increase contrast', and then click 'Enhance image'	'Saved to ./media/images/eiffel_enhanced.png'	Pass
13	F4.3 (SH)	Test whether the system is able to perform sharpening on an image	Upload + select 'eiffel.jpg', select 'simple kernel sharpen' or 'bilateral sharpen', and then click 'Enhance image'	'Saved to ./media/images/eiffel_enhanced.png'	Pass

14	F4.4 (CH)	Test whether the system is able to perform spatial filtering on an image	Upload + select 'eiffel.jpg', select 'simple kernel blur', and then click 'Enhance image'	'Saved to ./media/images/eiffel_enhanced.png'	Pass
15	F4.5 (CH)	Test whether the system is able to perform noise removal on an image	Upload + select 'eiffel.jpg', select 'simple kernel blur', and then click 'Enhance image'	'Saved to ./media/images/eiffel_enhanced.png'	Pass
16	F5.1 (MH)	Test whether the system recognises notable landmarks and is able to match them to a location	Upload + select 'eiffel.jpg', then click 'reverse image search'	Location of the Eiffel Tower presented on map and under Results	Pass
17	F5.2 (MH)	Test whether the system is able to perform a reverse image search	Upload + select 'eiffel.jpg', then click 'reverse image search'	Location of the Eiffel Tower presented on map and under Results along with commonly found search terms	Pass
18	F5.3 (CH)	Test whether the system is able to extract and identify street signs and names from an image	Upload and select 'm25.jpg', then click 'Analyse Road Signs'	Predictions 'Luton', 'St Albans', 'Hatfield' and 'Stansted' displayed under Results, along with their GPS coordinates	Pass
19	F5.4 (SH)	Test whether the system is able to capture registration plate information from any visible car plates in an image	Upload and select 'traffic.jpg', then click 'Analyse Number Plates'	Multiple UK registration numbers displayed + 'United Kingdom' prediction under Results	Pass
20	F5.5 (WH)	Test whether the system is able to match faces against social media posts to identify people of interest	N/A	N/A	N/A - deprioritised
21	F5.6 (CH)	Test whether the system is able to recognise logos and brand names and use them to verify location	Upload + select 'shops.jpg', then click 'shop sign analysis'	Combinations of visible shops are used to determine possible image locations or provide location-based clues to the user	Fail - feature not implemented before deadline

22	6.1 (SH)	Test whether the system is able to use mountain range shape to identify location	Upload + select 'mountain.jpg', then click 'mountain recognition'	Top 20 mountain predictions for given image are displayed on the map and under Results	Pass
23	6.2 (WH)	Test whether the system is able to use water bodies as a reference for geolocation	N/A	N/A	N/A - deprioritised
24	6.3 (WH)	Test whether the system is able to use stars and constellations in a night photo to estimate general image location	N/A	N/A	N/A - deprioritised
25	7 (MH)	Test whether the system is able to rank location suggestions from most likely to least likely (but still possible)	Upload and select 'lasvegas.jpg', then click 'Run All Operations'	An list of GPS ordered by probability. Each prediction is displayed alongside its probability score	Pass
26	8 (MH)	Test whether the system is able to display a report of the OSINT findings to the user	Upload and select 'lasvegas.jpg', then click 'Run All Operations'	Results of each operation displayed under Results	Pass
27	9 (MH)	Test whether the system allows the user to run all analysis tools automatically	Upload and select 'lasvegas.jpg', then click 'Run All Operations'	Results of each operation displayed under Results	Pass
28	10 (WH)	Test whether the system is able work out the exact location at which an image was taken	N/A	N/A	N/A - deprioritised

Table 4: Functional Acceptance Test Table

Non-Functional Acceptance Tests

Test	Spec Ref	Purpose	Input	Expected	Outcome
1	NF1 (MH)	The application should be intuitive to use. A new non-technical user should be able to learn how to navigate the system within 15 minutes of start-up.	A new user is presented with the application and is told to learn to navigate it.	The user should take under 15 minutes to learn to navigate the platform.	Pass - the user takes roughly 5 minutes to become fluent with the system
2	NF2 (MH)	The system must have a Graphical User Interface which is easy to navigate. A user should be able to navigate to any interface in under 30 seconds from any other interface.	A user is presented with the application and told which interface to navigate to.	The user should take under 30 seconds to navigate to the requested interface	Pass - this always takes under 30 seconds
3	NF3.1 (SH)	Feedback should be given to users informing them if actions have been successful. Include and display system messages showing the status and progress of image processing or search queries.	A user selects an image and clicks 'Run All Operations'	A loading wheel is presented during processing. Results are listed for successful operations, information is given for unsuccessful ones.	Pass - as expected
4	NF3.2 (SH)	The system should remain operational 99.9% of the time. System downtime should be very rare, ideally solely in the case of critical updates.	N/A	N/A	N/A - the system is hosted on the user's device so is operational whilst they have the application open and its functionality is reliant upon Python scripts and well-established third-party APIs

5	NF4 (MH)	The system must be tested modularly throughout development. During testing, there must be normal, extreme and erroneous tests on all features. Each individual feature (as split in functional requirements) should be able to be tested individually. The entire, complete system can be tested as a whole.	N/A	N/A	Pass - individual scripts can be run for testing, or the entire system can be tested using the 'Run All Operations' button
6	NF5 (SH)	The system should be able to handle large amounts of concurrent image uploads and requests. This should be achieved without a significant loss in system performance.	N/A	N/A	Pass - the system cannot be used by multiple users at the same time since it is hosted locally on a single user's device.
7	NF6 (SH)	The system should always remain up-to-date. The system should be continually maintained to use the latest versions of associated technologies.	System updates are available	The user runs `npm update` to update all packages	Pass

Table 5: Non-Functional Acceptance Test Table

Feature Performance Testing

Because the methods and tools used by AutoOSINT for performing geolocation differs vastly between urban and mountainous scenes, the testing has been separated into two corresponding parts. The mountainous case studies have been used to test the mountain geolocation module, whereas the urban case studies have been used to test the app's remaining geolocation features.

Urban Case Studies

Urban case studies were collated into a testing set of around 40 images by taking screenshots of various locations in Google Street View and GeoGuessr[35]. Several pictures were also provided by Wikimedia Commons[36]. Images were selected so that they featured at least one of the following visual features:

- A vehicle with a visible and legible number plate.
- A famous landmark (such as the Eiffel Tower, the Statue of Liberty or Machu Picchu).
- A legible road sign containing some geographical place name.

[35] https://www.google.com/streetview/
[36] https://commons.wikimedia.org/

The images were selected so that approximately the same number of images was included in the test set for each type of feature. This helped to test the effectiveness and contribution of each feature towards the predictions.

For each test case, the types of features visible in the image were recorded, along with the true latitude and longitude. The image was then analysed using the urban-oriented automated tools in the AutoOSINT app, and the closest of the top three predictions in the confidence ranking was recorded. The distance between the true and predicted locations were then calculated to give an error value in kilometres.

Test #	Image name	Famous landmark?	Number Plates?	Road signs?	Lat true	Lon true	Lat pred (best)	Lon pred (best)	Pred #	Error (km)
1	eiffel_1	y	n	n	48.86048	2.29032	48.85832	2.29452	1	0.39
2	eiffel_2	y	n	n	48.85399	2.30083	-	-	-	-
3	swansea_1	n	n	y	51.62121	-3.90883	51.60756	-3.98857	1	5.71
4	swansea_2	n	y	n	51.62428	-3.95223	-	-	-	-
5	machu_1	y	n	n	-13.16377	-72.5449	-	-	-	-
6	nyc_1	n	n	y	40.63779	-74.07824	39.97837	-86.11804	2	1023.74
7	nyc_2	y	n	n	40.68918	-74.04358	40.68929	-74.04457	1	0.08
8	warwick_1	n	n	y	52.37372	-1.55262	52.28333	-1.58333	3	10.27
9	warwick_2	n	n	y	52.37915	-1.56085	52.28333	-1.58333	3	10.76
10	m25	n	y	y	51.71663	-0.37841	51.69167	-0.19664	1	12.83
11	m11	n	y	n	51.61074	-0.73358	52.08277	-0.7396	1	52.49

Figure 59: Test results for a subset of urban test cases.

The histogram below in Figure 60 illustrates the distribution of errors between the true and predicted locations.

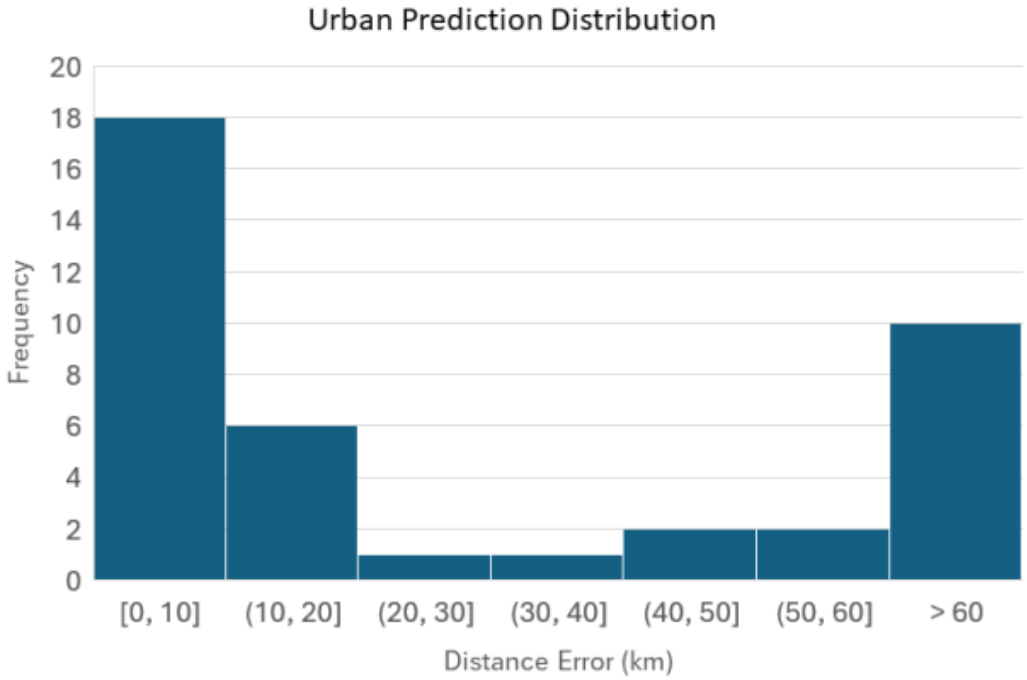


Figure 60: Distribution of top prediction error in urban test cases.

Mountainous Case Studies

Before beginning mountainous location testing, it was important to consider the project's success criteria and determine when a prediction is positive or negative. There are 2 considerations:

1. What is an acceptable error margin - in terms of distance?

2. How many suggestions should the cross-modal model return?

With these factors in mind, a test was constructed where multiple trained models were tested against a disconnected set of mountain photographs (queries) and corresponding database (converted depth maps). This test set consists of 500 locations spread throughout the Alps. The test was conducted where database candidates were positive if at 1000m, 2000m, 3000m, 4000m and 5000m to the query. The models were also able to predict the top 1, 10, 20, 50 and 100 candidates in the database. The best result can be observed below in Figure 61:

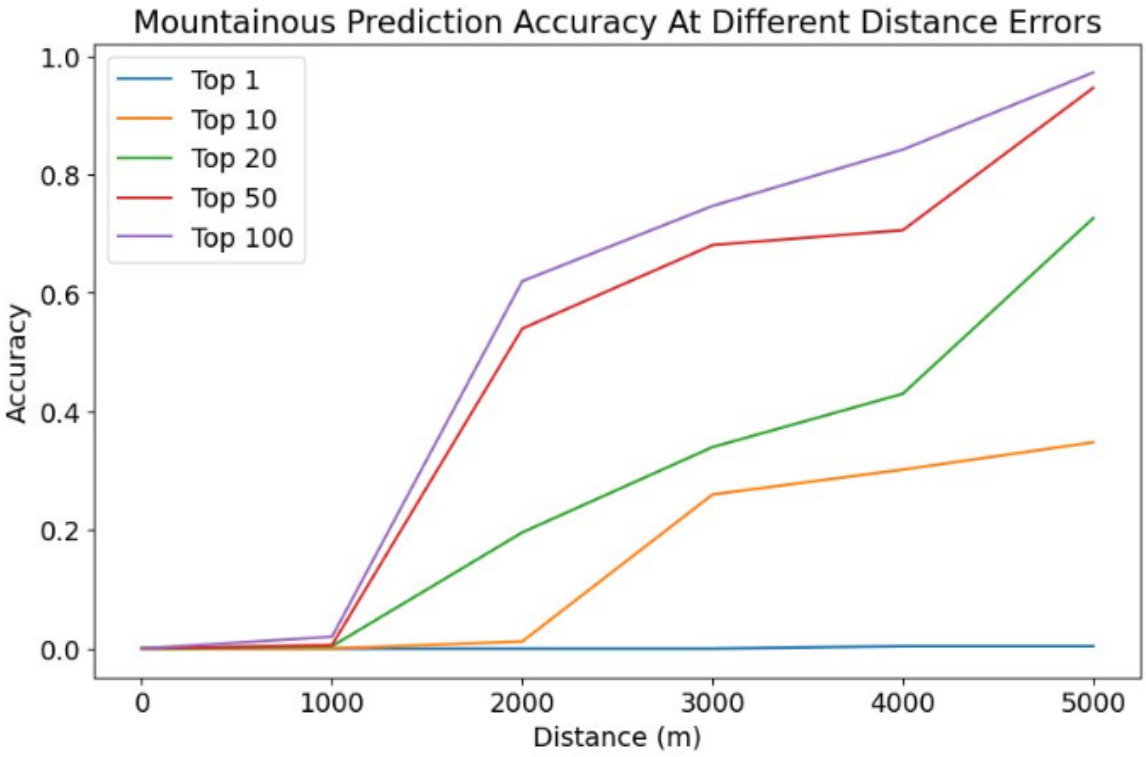


Figure 61: Cross-modal mountain geolocation feature accuracy - with respect to distance error and different suggestion ranges

As can be seen, when suggesting only the top 1 result, the model never produces a correct result, no matter the distance error. However, by increasing the number of suggested database entries, we can observe that the accuracy increases rapidly.

In comparison to the results obtained in the Crosslocate paper, these results are not positive. The model reaches very similar levels when the error distance is 5000m with the top 100 suggestions, however, cross-locate achieves much higher recall/accuracy at lower distances and with fewer suggestions. This can be observed in Figure 62.

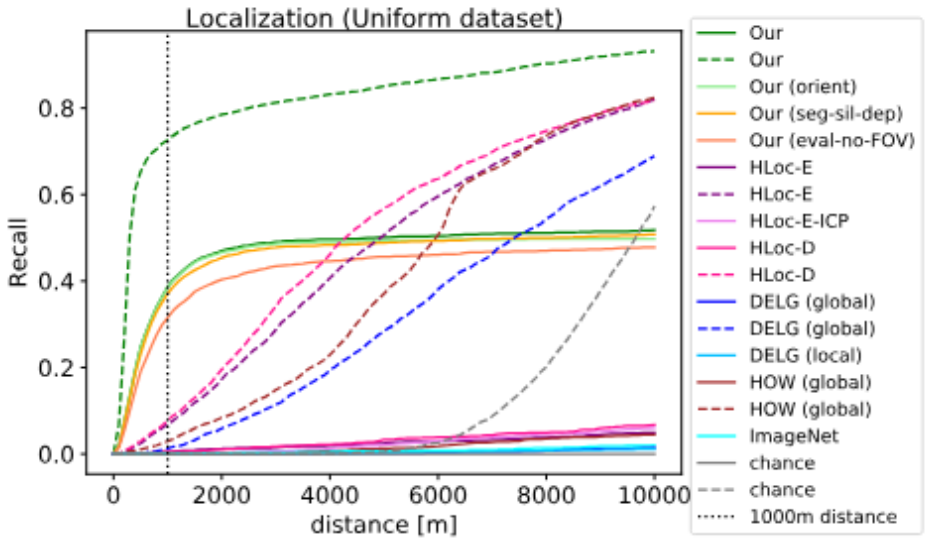


Figure 62: Crosslocate (green) performance against industry standards - dashed is using top 100 suggestions, solid is using top 1 suggestion.

The weaker performance can be attributed to the resource limitations which forced compromise during the training phase. Firstly, the memory restrictions prevented utilising the full dataset for training. The database had to be sampled and thus the data variation was also lower. Secondly, The GPU provided in the batch computation system was much weaker than the one used in the Crosslocate study. Therefore training was much slower - despite using a smaller train set. The number of epochs that the model was trained for was ultimately lower. After training for 96 hours, the model only reached 80 Epochs where the minimum suggested number is 100. Given more time/better resources, it is believed that the performance obtained by Crosslocate could be equalled.

Nonetheless, the model achieves acceptable results considering the size of the database. If a correct prediction is made with 100 suggestions - out of a database with approximately 17,000 entries - this is still a very positive narrowing down of results. Hence the feature was not a complete failure, and partial success.

Project Evaluation

Performance of AutoOSINT

AutoOSINT compared favourably in tests compared to other automated image geolocation OSINT systems in cases where there were identifiable visual clues (such as car number plates and famous landmarks). In other cases, such as in non-mountainous rural scenes where the only visual clues given are related to flora and land topology, the system generally does not perform as well. This was expected due to the limitations of the project and these types of features not being considered by any of the features that were included.

Urban Scenes

The quality of results generated by the urban-oriented tools was promising, although somewhat polarised: in testing, most predictions were either within 10 kilometres of the target or more than 60 kilometres away. This suggests that when the system can correctly identify the clues in an image, it performs well. However, there are a fairly significant number of edge cases where either no clues were detected or the contents of the image were misinterpreted. By implementing more tools for urban scene geolocation, the number of accurate predictions would likely increase.

The features which were most likely to be effective at geolocation on any given urban image were the reverse image search module and the road sign analysis module—these features were very often able to geolocate an image to within 10 kilometres. The number plate detection module was helpful for geolocation in some cases, although it struggled in other cases, even where images contained number plates. This was largely caused by number plates taking up only a small area of most images and therefore being of very low resolution. These number plates were difficult for both humans and optical character recognition software to interpret accurately. Additionally, using number plates for geolocation has several limitations, discussed in the feature overview section of this report.

In testing, 45% of AutoOSINT's urban predictions were within 10 kilometres of the target. This surpasses the 40% of guesses within 25 kilometres which the PIGEON automated geolocation model was able to achieve, although this is not necessarily a fair comparison because the test set used for AutoOSINT was specifically chosen to contain features which the system should be able to detect.

Overall, the quality of the results obtained when testing urban case studies suggests that AutoOSINT would be helpful for both non-technical and expert users for geolocating images in urban areas. However, there is some clear room for improvement in some cases, which could be addressed with future development and extensions.

Mountainous Scenes

Before any evaluation is performed, it should be taken into consideration that the implemented model is a demonstrator of technology. The code doesn't purely predict locations but also offers neural network training. Given the right resources (hardware and dataset). The solution can be used to train a model from scratch. The final model provided in the project is also only capable of handling the Alps region. So for greater-scale geolocation, more models or a wider database would be required.

In regards to the model results obtained, the cross-modal mountain geolocation feature unfortunately did not meet expectations. Since the result was based on a study, the same performance was expected.

However, there exists a wide disparity between the Crosslocate model and the project model. This is arguably the biggest downfall of the project. Despite pouring hours into the training process, the achieved geolocation results were simply disappointing.

Part of the failure can be explained by the hardware-limited training. The model may have been potentially too ambitious for the available resources, which meant that the concept behind cross-modal geolocalisation was simply unattainable from the point of model choice. However, part of the blame can also be attributed to the development process. There should have been more time dedicated to producing the mountain geolocation feature and backup plans introduced in case such an issue arose.

On the other hand, this is an understandably difficult area of expertise which is developing rapidly. So the lower-than-expected results are also forgivable. Furthermore, by examining the results achieved by the project's implementation (Figure 61) and comparing them side by side against some of the industry standards (Figure 62), it can be confirmed that the model still performs better than many implementations. Narrowing down the to 100 potential locations emphasises the "semi-automatic" aspect of our project, as an expert can take over from this stage and have a much easier time than geolocating an entire region.

Interface and Usability

The user interface is overall responsive and easy to use. The app was given to one technical and one non-technical user (a Computer Science student and another student) for the purposes of user acceptance testing and feedback was generally positive. The users were given the instructions included in the user manual and instructed to test the app by uploading and analysing a number of images. Several aspects of the interface and user experience were praised, including the visual feedback given when each button is selected or clicked, and the image operation buttons, which change colour after they have been applied to the selected image. The 'About' tab was also appreciated.

The main criticisms given by the users involved the slow execution of the automated tools, which is a known issue. In addition, one user suggested that it should be possible to delete images from the app to prevent the home page from becoming cluttered.

Efficacy of Development Process

At the start of the project, the group committed to a methodology which draws inspiration from the best parts of many methodologies. Theoretically, this mashup of methods should perfectly fit the circumstances faced by the team, however, it would be valuable to investigate whether this is true. This can teach us, as individuals, how to proceed in the future when undertaking new projects.

Firstly comes our approach to decomposing the project and planning the various phases. We implemented an **incremental approach** whereby we selected features at the start of an increment and only designed them after they were selected. In hindsight, this was a very good decision. Going into the project, all members were new to the concept of OSINT and had much to learn. If the entire system was designed early on, the features would not only fail to fulfil goals and customer requirements, but the team would also struggle to respond to changes. In situations where we would be able to respond, there would have been redundant work performed, and wasted time.

This can be demonstrated with the example of AI Detection. As indicated in the requirements section, the requirement to detect fake images generated by AI was de-prioritised and cancelled from the development backlog. Firstly, this was a response to the customer/supervisor's advice to shift focus on geolocation over the image processing pipeline. The team effortlessly replaced said feature as the design of this feature was deferred and no work had gone into it. This meant no redundant work and thus flexibility in final implementation.

On the other hand, the **initial general system overview lacked detail**. Specifically, the points of connection and details of integration were not clear. Therefore, after the first increment, the team had some struggles with merging the separate features into one system. Contrarily, the group **learned from this difficulty** and decided to integrate more gradually in the second increment, so with a solid foundation in place, this issue no longer persisted. This again highlights the positive of incrementally planning and adapting to difficulties.

The incremental breakdown was followed by an even finer decomposition into sprints using a **Scrumban approach**. The aim of this was to maximise the value delivered by the team. The sprints would be fast and brief which meant a high development rate and frequent evaluations of the implementations against the project goals. Evidence from the schedule indicates that this approach was effective at hitting goals and minimising redundant work, however, it wasn't sustainable. Specifically, during busy periods - like with other deadlines - the group was unable to stick to **sprints** as they were **too intense**. Ultimately, as part of the risk mitigation plan, plenty of spare time was allocated to each sprint/increment so the delays were not critical, however for future reference, sprints should account for the contextual schedules of all members.

In terms of the **Kanban** aspect of Scrumban, the group exhibited exceptional levels of autonomy and flexibility. The sprints gave team members a choice of features to tackle at their convenience and were designed in such a way as to prevent dependency whenever possible. This led to high engagement and minimal idle time. However, occasionally the team would need to be reminded to update their progress on the Jira Kanban board as they would neglect the admin side of the methodology. This led to some confusion, but it was not critical as the group communicated frequently and kept each other up to date.

Other practices at times experienced similar neglect - such as version control or the responsible review system - however, this was kept to a minimum. The high levels of required management and bureaucracy did add overhead, but it is believed to have been worth the safety and management clarity.

As demonstrated, for most parts, incremental Scrumban was very effective, however, it became even more meaningful when combined with **Test-Driven Development**. By approaching the increments with features of gradually greater difficulty, the team could gain the required knowledge and expertise from research and development experience. Individual members would strategise when selecting backlog items such that required tutorials could be completed simultaneously without needing to pause development. Furthermore, with the test cases determined before implementation, the feature contribution towards requirements and customer needs would be verified immediately, which once again highlights the focus on productively delivering value.

From the conception of the project, and even more so after the risk assessment, the group was aware of the danger posed by losing a team member from an already small team. It therefore had to be ensured that throughout the project, there was a focus on delivering value at a good rate whilst also diversifying group skills/knowledge of the system. Whilst delivery rate and expert development are

covered above, cross-training and diversification of the system accountability system were ensured through the **responsible-review system**. This pair programming practice would require each member to learn about features they are not responsible for. In case of member unavailability, progress wouldn't be halted and features still recovered. This was a very prudent approach. The issue with the approach is that each member had additional responsibilities, to an already high workload. Nonetheless, this amount was manageable and the positives outweighed the cost of such actions.

This mitigation plan would come into effect in term 2 as a team member - Lara Goss - would leave the team. The theoretical benefits of the system would be tested and prove incredibly useful. For the features that progressed to the end stage - such as metadata extraction - the team was able to integrate without a problem and troubleshoot when required. However, the responsible-review system isn't a silver bullet. Features that were far from completion and complex unfortunately could not be implemented within the project time-frame. This was not down to a lack of understanding of feature functionality, but rather due to a shortage of time combined with a higher workload being placed on the remaining three team members who already had their own lists of priorities. Nonetheless, it was crucial to save the features that could be saved, and the situation helped us to evaluate the efficacy of our risk management systems whilst illustrating what we could improve upon for future projects.

Overall, we can observe that the methodology adopted was tailored to the project circumstances. It tied in well with group strengths and aimed to reinforce the weak parts (such as the group size). All in all the grand strategy and development practices led to a successful project which means that our approach was positive. The aspects of the methodology that were weaker have been recognised by the team and will be learnt from in future endeavours.

Limitations of AutoOSINT

One weakness of the AutoOSINT app is the running time of some of the automated features. The reverse search, authenticity verification and number plate analysis modules in particular can take up to five minutes to execute, which slows down the OSINT process. Due to time constraints, it was decided that development would be focused mostly on creating a wide variety of features, an intuitive interface and having accurate results. Therefore, there was little consideration for the optimisation of features.

Another limitation of the system is that it is not effective at predicting the location of images with few visual clues. For example, rural scenes with no road signs, cars, famous landmarks, mountains or attached metadata are likely to produce zero location predictions.

Furthermore, predictions given by AutoOSINT do not have the pinpoint precision which might be required in some cases. While many of the predictions are within 10 kilometres of the true location of the input image, the system is not able to predict the exact location of the target. This is due to the fact that depth and pose estimation for geolocation are very difficult tasks to solve given just a single image as input (because a single image can contain perspective-based ambiguity).

The ability to input clues was not included in the final app. This was partially due to time constraints, as well as a revision made to the project's technical requirements where it was decided that the app performs well without this feature included. Without this feature, AutoOSINT may be less effective in cases where the user has any extraneous information about an input image which could be helpful for geolocation.

Another drawback of the system is the poor test results obtained by the mountain geolocation model. Not only is it limited to the Alps, but the hardware-limited training resulted in weaker-than-expected performance.

Extensions and Future Work

One possible extension to the AutoOSINT app is extending analysis to multi-modal inputs, such as images with attached text. Experimental results show that multi-modal models (which analysed images with associated news articles) were more effective at geolocation tasks than image-only classifiers because the texts often contained potential locations for the model to predict[65]. By allowing users to input captions or text found with a given image, the app may be able to predict location with a higher degree of accuracy.

Another area for future work which was discussed in the literature review is the implementation of depth and pose estimation for more precise geolocation. This is a complex task which would require a substantial amount of research and time for development (the tests conducted in the research stages of this project were largely unsuccessful), but the task has been shown to be possible to achieve in at least some cases[58]. This feature would greatly improve the overall precision of the automated geolocation system, helping to further narrow down potential search areas for users.

One of the limitations of the app currently is that some of the features (especially reverse image search, authenticity verification and number plate analysis) are slow to execute. By implementing multi-threading and conducting research into more efficient algorithms, these features can be made to run faster, improving the usability of the app and making the OSINT process more efficient overall.

Due to hardware limitations, the mountain geolocation module was not able to be trained on the full dataset of training images included in the original proposal of the model (discussed in the literature review). Instead, the sparse dataset was used for training, leading to slightly worse performance. Furthermore, the training set consists only of locations in the Alps, so images of other mountainous areas can not be geolocated by this tool. This could be improved upon by building a larger training set composed of a larger number of mountain ranges and using more powerful hardware to train the model.

Another extension to improve the app's usability could be to convert the system into an online web app hosted on a remote server. This would make the setup process significantly faster and simpler as users would no longer be required to host the server locally before running the app. Because the desktop app was built using Electron, which enables desktop apps to be built using web technologies, this change would be relatively simple.

Finally, there are a number of potential features which were discussed in the initial planning phase but ultimately not included in the development timeline. For example, facial recognition and social media web page scraping could be used to help elicit information about an image. Additionally, geolocation through identifying bodies of water and constellation patterns were also discussed. Because of the modular approach used in the design of the AutoOSINT app, such features could be easily developed and integrated into the current version of the app.

Conclusion

Overall, the AutoOSINT app shows promising performance for image geolocation of both urban and mountainous scenes, although there is clear opportunity for improvement and extensions, which could be addressed in the future. The development methodology and overall approach to research and development was moderately successful, with a good level of efficiency and team cohesion despite the various challenges and learning opportunities encountered during the span of the project. The finished project was able to fulfil a large part of the original specification and the app shows some genuine utility, both for non-technical users and OSINT experts looking to geolocate images.

User Manual and Documentation

Installation and setup instructions are specified in the README file, which is included as an appendix to this report as well as in `README.md` in the submission folder and the project's code repository on Github[37].

Once the app is installed and correctly set up, the user experience is designed to be easy and intuitive. The interface consists of three tabs: Home, Upload and About. The Home tab displays all images which have been uploaded to the app. To upload an image, go to the Upload tab, click the 'Choose file' button, select the desired image file in the file explorer and click the 'Upload Image' button.

To perform analysis on an image, go to the Home tab and click on the thumbnail of the image to be analysed. The user can then use the buttons in the interface to run individual OSINT operations, or all operations at once. All results are displayed in the Results section below the image.

The About tab displays some basic information, giving an overview of the app as well as information about how the user's images are used.

[37] https://gitfront.io/r/danfredrickson/3uR1cB4TDZx5/AutomatedOSINT/

References

[1] C. Colquhoun, *A brief history of open source intelligence*, bellingcat, Jul. 2016. [Online]. Available: https://www.bellingcat.com/resources/articles/2016/07/14/a-brief-history-of-open-source-intelligence/.

[2] E. Borges, *What is the osint framework? how can you use it*, Recorded Future, 2024. [Online]. Available: https://www.recordedfuture.com/threat-intelligence-101/intelligence-sources-collection/osint-framework.

[3] H. Gibson, "Acquisition and preparation of data for osint investigations," *Open Source Intelligence Investigation*, pp. 69–93, 2016. DOI: 10.1007/978-3-319-47671-1_6.

[4] N. A. Hassan and R. Hijazi, "The evolution of open source intelligence," *Open Source Intelligence Methods and Tools*, pp. 1–20, 2018. DOI: 10.1007/978-1-4842-3213-2_1. (visited on 10/06/2022).

[5] O. Zoder, "Automated collection of open source intelligence," 2020. [Online]. Available: https://is.muni.cz/th/ww745/sdipr_Archive.pdf (visited on 10/23/2023).

[6] G. R. S. Weir, *Chapter 9 - the limitations of automating osint: Understanding the question, not the answer*, R. Layton and P. A. Watters, Eds., ScienceDirect, Jan. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128029169000099 (visited on 10/24/2023).

[7] BBC, *Reddit apologises for online boston 'witch hunt'*, BBC, 2013. [Online]. Available: https://www.bbc.co.uk/news/technology-22263020.

[8] A. M. Ponder-Sutton, "Chapter 1 - the automating of open source intelligence," in *Automating Open Source Intelligence*, R. Layton and P. A. Watters, Eds., Boston: Syngress, 2016, pp. 1–20, ISBN: 978-0-12-802916-9. DOI: https://doi.org/10.1016/B978-0-12-802916-9.00001-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128029169000014.

[9] TechTarget, *Web application (web app)*, TechTarget, 2023. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app.

[10] Levminer, *Tauri vs. electron - real world applicaiton*, 2022. [Online]. Available: https://www.levminer.com/blog/tauri-vs-electron.

[11] C. Riggs, T. Douglas, and K. Gagneja, "Image mapping through metadata," *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*, pp. 1–8, Oct. 2018. DOI: 10.1109/ssic.2018.8556664.

[12] P. Kakar and N. Sudha, "Authenticating image metadata elements using geolocation information and sun direction estimation," *2012 IEEE International Conference on Multimedia and Expo*, pp. 236–241, Jul. 2012. DOI: 10.1109/icme.2012.82.

[13] P. R. Kumar, C. Srikanth, and K. Sailaja, "Location identification of the individual based on image metadata," *Procedia Computer Science*, vol. 85, pp. 451–454, 2016. DOI: 10.1016/j.procs.2016.05.191.

[14] M. W. Group *et al.*, *Guidelines for handling image metadata (v2. 0)*, 2010.

[15] S. Acharya R, S. J, S. S, V. S Aithal, and H. G S, "Geo-locating an image using exif data," *International Journal of Engineering Applied Sciences and Technology*, vol. 8, no. 1, pp. 43–47, May 2023. DOI: 10.33564/ijeast.2023.v08i01.007.

[16] R. Mills, "Image file formats," in *Image Metadata and Exiv2 Architecture*.

[17] B. Toevs, "Processing of metadata on multimedia using exiftool: A programming approach in python," *2015 Annual Global Online Conference on Information and Computer Technology (GOCICT)*, pp. 26–30, Nov. 2015. DOI: 10.1109/gocict.2015.14.

[18] K. Thammarak, P. Kongkla, Y. Sirisathitkul, and S. Intakosum, "Comparative analysis of tesseract and google cloud vision for thai vehicle registration certificate," *International Journal of Electrical and Computer Engineering*, vol. 12, Apr. 2022.

[19] S. University, *Image processing for photography and vision: Sharpening*, Stanford University, 2020. [Online]. Available: https://web.stanford.edu/class/cs448f/lectures/2.1/Sharpening.pdf.

[20] D. A. Schumacher, "Image smoothing and sharpening by discrete convolution," *Elsevier eBooks*, pp. 50–56, Jan. 1991. DOI: 10.1016/b978-0-08-050754-5.50021-9. (visited on 04/23/2024).

[21] F. Heide, *Image processing*, Princeton University, 2021. [Online]. Available: https://www.cs.princeton.edu/courses/archive/spring21/cos426/lectures/Lecture-1.pdf.

[22] J. Malathi, T. Satya Nagamani, K. N. V. S. K. Vijaya Lakshmi, and P. Rama devi, "Survey: Image forgery and its detection techniques," *Journal of Physics: Conference Series*, vol. 1228, p. 012 036, May 2019. DOI: 10.1088/1742-6596/1228/1/012036. (visited on 08/27/2020).

[23] D. Singh, P. Singh, R. Jena, and R. S. Chakraborty, "An image forensic technique based on jpeg ghosts," *Multimedia Tools and Applications*, vol. 82, Sep. 2022. DOI: 10.1007/s11042-022-13699-x. (visited on 11/24/2022).

[24] *Morphological operations*, Buff ML, https://buffml.com/morphological-operations/ 2022. [Online]. Available: https://buffml.com/morphological-operations/ (visited on 04/19/2024).

[25] N. Jamil, T. Mohd, and Z. A. Bakar, "Noise removal and enhancement of binary images using morphological operations," *International Symposium on Information Technology*, Sep. 2008. DOI: 10.1109/itsim.2008.4631954.

[26] H. Sulehria, Y. Zhang, and D. Irfan, *Mathematical morphology methodology for extraction of vehicle number plates*, 2007.

[27] N. Islam, Z. Islam, and N. Noor, "A survey on optical character recognition system," *arXiv:1710.05703 [cs]*, vol. 10, Oct. 2017. [Online]. Available: https://arxiv.org/abs/1710.05703.

[28] *Tesseract user manual*, Tesseract Documentation, 2024. [Online]. Available: https://tesseract-ocr.github.io/tessdoc/Home.html.

[29] S. Akhil, *An overview of tesseract ocr engine*, 2016.

[30] A. Agbemenu, J. Yankey, and E. Addo, "An automatic number plate recognition system using opencv and tesseract ocr engine," *International Journal of Computer Applications*, vol. 180, May 2018.

[31] Google, *Cloud vision*, Google Cloud, 2019. [Online]. Available: https://cloud.google.com/vision/.

[32] Google, *Detect labels | cloud vision api*, Google Cloud. [Online]. Available: https://cloud.google.com/vision/docs/labels.

[33] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov. 2004. DOI: 10.1023/b:visi.0000029664.99615.94.

[34] J. Tomešek and J. Brejcha, *Crosslocate: Cross-modal large-scale visual geo-localization in natural environments using rendered modalities*, 2022. [Online]. Available: https://cphoto.fit.vutbr.cz/crosslocate/data/paper/CrossLocate.pdf (visited on 04/29/2024).

[35] J. Brejcha and M. Čadík, "Geopose3k: Mountain landscape dataset for camera pose estimation in outdoor environments," *Image and Vision Computing*, vol. 66, pp. 1–14, Oct. 2017. DOI: 10.1016/j.imavis.2017.05.009. (visited on 04/08/2021).

[36] A. Nelson, H. Reuter, and P. Gessler, "Chapter 3 dem production methods and sources," in *Geomorphometry*, ser. Developments in Soil Science, T. Hengl and H. I. Reuter, Eds., vol. 33, Elsevier, 2009, pp. 65–85. DOI: https://doi.org/10.1016/S0166-2481(08)00003-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166248108000032.

[37] A. R. Zamir, A. Hakeem, L. V. Gool, M. Shah, and R. Szeliski, *Large-Scale Visual Geo-Localization*. Springer, Jul. 2016.

[38] S. Frey, F. Sadlo, and T. Ertl, "Explorable volumetric depth images from raycasting," in *2013 XXVI Conference on Graphics, Patterns and Images*, 2013, pp. 123–130. DOI: 10.1109/SIBGRAPI.2013.26.

[39] L. Herman, S. Popelka, and V. Hejlova, "Eye-tracking analysis of interactive 3d geovisualization," *Journal of Eye Movement Research*, vol. 10, May 2017. DOI: 10.16910/jemr.10.3.2. (visited on 04/06/2021).

[40] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015. DOI: 10.1109/cvpr.2015.7298682. [Online]. Available: https://arxiv.org/abs/1503.03832.

[41] E. Winarno, W. Hadikurniawati, and R. N. Rosso, "Location based service for presence system using haversine method," *2017 International Conference on Innovative and Creative Information Technology (ICITech)*, Nov. 2017. DOI: 10.1109/innocit.2017.8319153.

[42] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, 1996. [Online]. Available: https://cdn.aaai.org/KDD/1996/KDD96-037.pdf.

[43] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited," *ACM Transactions on Database Systems*, vol. 42, pp. 1–21, Aug. 2017. DOI: 10.1145/3068335. [Online]. Available: http://www.ccs.neu.edu/home/vip/teach/DMcourse/2_cluster_EM_mixt/notes_slides/revisitofrevisitDBSCAN.pdf.

[44] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, Neural Information Processing Systems, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[45] K. Han, Y. Wang, H. Chen, *et al.*, "A survey on vision transformer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, pp. 1–1, 2022. DOI: 10.1109/tpami.2022.3152247.

[46] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, arXiv.org, Jun. 2021. DOI: 10.48550/arXiv.2010.11929. [Online]. Available: https://www.arxiv.org/abs/2010.11929v2.

[47] A. Radford, J. W. Kim, C. Hallacy, *et al.*, *Learning transferable visual models from natural language supervision*, proceedings.mlr.press, Jul. 2021. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.

[48] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, 2013. [Online]. Available: https://arxiv.org/pdf/1301.3781.pdf.

[49] S. Stevens, J. Wu, M. J. Thompson, *et al.*, *Bioclip: A vision foundation model for the tree of life*, arXiv.org, Dec. 2023. DOI: 10.48550/arXiv.2311.18803. [Online]. Available: https://arxiv.org/abs/2311.18803 (visited on 04/21/2024).

[50] PlantNet, *Plantnet identify*, identify.plantnet.org. [Online]. Available: https://identify.plantnet.org/ (visited on 04/21/2024).

[51] L. Croizat, *Manual of Phytogeography: an Account of Plant-Dispersal Throughout the World*. Springer, Nov. 2013.

[52] Google, *Google street view*. Google Maps Street View, 2015. [Online]. Available: https://www.google.com/streetview/.

[53] L. Haas, M. Skreta, S. Alberti, and C. Finn, *Pigeon: Predicting image geolocations*, arXiv.org, Apr. 2024. DOI: 10.48550/arXiv.2307.05845. [Online]. Available: https://arxiv.org/abs/2307.05845 (visited on 04/22/2024).

[54] F. Gao, F. Deng, L. Li, L. Zhang, J. Zhu, and C. Yu, "Mgg: Monocular global geolocation for outdoor long-range targets," *IEEE Transactions on Image Processing*, vol. 30, pp. 6349–6363, 2021. DOI: 10.1109/TIP.2021.3093789.

[55] L. d. Guzman, *Scene distance estimation using monocular depth estimation*, GitHub, Jan. 2024. [Online]. Available: https://github.com/LanzDeGuzman/Scene-Distance-Estimation-Using-Monocular-Depth-Estimation (visited on 04/13/2024).

[56] R. Horaud, B. Conio, O. Leboulleux, and B. Lacolle, "An analytic solution for the perspective 4-point problem," *Computer vision, graphics, and image processing*, vol. 47, pp. 33–44, Jul. 1989. DOI: 10.1016/0734-189x(89)90052-2. (visited on 04/23/2023).

[57] *Opencv: Perspective-n-point (pnp) pose computation*, docs.opencv.org. [Online]. Available: https://docs.opencv.org/3.4/d5/d1f/calib3d_solvePnP.html.

[58] A. Criminisi, "Single view metrology," *International Journal of Computer Vision*, vol. 40, pp. 123–148, 2000. DOI: 10.1023/a:1026598000963. (visited on 12/11/2020).

[59] D. Kim, W. Ka, P. Ahn, D. Joo, S. Chun, and J. Kim, *Global-local path networks for monocular depth estimation with vertical cutdepth*, arXiv.org, Oct. 2022. DOI: 10.48550/arXiv.2201.07436. [Online]. Available: https://arxiv.org/abs/2201.07436 (visited on 06/21/2023).

[60] Y. Ishii and T. Yamashita, *Cutdepth:edge-aware data augmentation in depth estimation*, arXiv.org, Jul. 2021. DOI: 10.48550/arXiv.2107.07684. [Online]. Available: https://arxiv.org/abs/2107.07684 (visited on 04/21/2024).

[61] A. Schade, *Responsive web design (rwd) and user experience*, 2014. [Online]. Available: https://www.nngroup.com/articles/responsive-web-design-definition/.

[62] D. Kuria, *Master image enhancement techniques with opencv*, MUO, Jun. 2023. [Online]. Available: https://www.makeuseof.com/opencv-image-enhancement-techniques/ (visited on 04/29/2024).

[63] SpaCy, *Named entity recognition*, SpaCy, 2023. [Online]. Available: https://spacy.io/usage/linguistic-features#named-entities (visited on 04/2024).

[64] SpaCy, *Word vectors and semantic similarity*, SpaCy, 2024. [Online]. Available: https://spacy.io/usage/linguistic-features#vectors-similarity (visited on 04/2024).

[65] G. Tahmasebzadeh, S. Hakimov, R. Ewerth, and E. Müller-Budack, "Multimodal geolocation estimation of news photos," *Lecture Notes in Computer Science*, vol. 45, pp. 204–220, Jan. 2023. DOI: 10.1007/978-3-031-28238-6_14. (visited on 04/21/2024).

Appendices

README.md

About

AutoOSINT is a tool to facilitate the automatic geolocation of images.

Installation

Project Dependencies:

- Node.js - see installation instructions here[38]
- Tesseract - see installation instructions here[39]
- GCloud CLI - see installation instructions here[40]
- Python v3.11

Create a GCloud project using the Resource Manager[41]:

- Choose a project ID
- run `gcloud auth application-default set-quota-project [project-id]`, replacing `[project-id]` with the project ID
- run `gcloud auth application-default login`
- enable the Google Cloud Vision API by visiting this link[42], replacing `[project-id]` with the project ID

From the root project directory:

- Project Node dependencies - run `npm install`
- Project Python dependencies - run `pip3 install -r requirements.txt`
- Spacy - run `python3 -m spacy download en_core_web_sm en_core_web_lg`

Execution

From the root folder:

- run `python3 manage.py makemigrations`
- run `python3 manage.py migrate`
- run `python3 manage.py migrate -run-syncdb`

Then:

- in one terminal window, run `python3 manage.py runserver`
- in another terminal window in the same directory, run `npm run autoosint`.

[38] https://nodejs.org/en/download/package-manager
[39] https://tesseract-ocr.github.io/tessdoc/Installation.html
[40] https://cloud.google.com/sdk/docs/install
[41] https://console.cloud.google.com/cloud-resource-manager?walkthrough_id=resource-manager--create-project&start_index=1#step_index=1
[42] https://console.developers.google.com/apis/api/vision.googleapis.com/overview?project=[project-id]

CS407 Meeting Minutes

April 30, 2024

Group Meeting 1

Date: 12/10/2023
Time: 15:00-15:30

Topics Discussed

Admin

- Project name - Automated Image OSINT
- Meetings
  - With Khalil - once a week
  - As a group - once a week.
    To discuss general updates and make group decisions
  - Group availability - Monday, Tuesday, and Thursday
- Project roles
  - Project manager - Lukasz
  - Admin contact - Matthew
  - Customer contact - undecided
  - Document manager - unsure of the details of this role (is this managing our code or our written documents?), Matthew possibly
  - Meetings coordinator - undecided

Specification

- This will be our main focus until week 4 term 1
- Should be 10-30 pages, with a large proportion of this being requirements

Specification Sections:

- Background project statement
  - Project is about finding different tools and combining them.
- Preliminary research - context of project, gap in market etc.
  - This research, on many case studies, will be used to decide the project scope.
- Methodology
  - Incremental methodology with agile aspects, instead of pure agile.
  - We will start by solving easy cases, to build our knowledge and tools, and then increase difficulty as the project progresses
- Objectives (SMART, MoScOw) What we are aiming to do.
- Group Structure/Project management
  - Supervisor must agree to our project management plan
  - Decide on this after we do some research
- Risk management
- Planned project timeline
- Tools and resources - make clear how the available resources will affect our objectives
  - Plan to use Trello to help with documentation. Can be used to help demonstrate our project management and time management
  - Overleaf for documentation writing.
  - Peak Finder website (recommended by Khalil)
- How we will measure success. How the people who are evaluating should determine the success of the project

Initial Project Thoughts

- Our project is about finding a bunch of tools and combining them. Should we assign one member to focus on each different tool?
- What are we actually going to make? (ie. web app, stand-alone app)
  - Would it be easier to make a stand-alone app? We could justify this by saying that in the future could extend to web app
  - Our group has limited experience in making web apps
- How will we get the raw data?
  - Could we just use google map images?
  - Anything online which are available resources to use?

  - Does Kaggle have any resources we could use?
- Should we try and create our own versions of tools for the simple cases?
  - E.g. using our own AI/ML model to interpret and capture street signs/number plates
    * Our model could find a number plate, extract this image section, and then feed it into a number plate interpreter
    * This will improve the quality of our project
- What case studies do we want to tackle?
  - Number plates, street signs, buildings? Or not?
- Are we meant to be fully locating an image or just answering some questions about the image?
- How far do we want to take our project?

Actions To Be Taken

- Lukasz - email and book our supervisor meeting for next week
- Everyone to link Warwick email to Overleaf for access to premium version
- Research to be undertaken - compare several different case studies, e.g. mountains, cities.
- Everyone needs to individually think about how far we want to take our project
  - Might be useful to create a tree-like diagram to help us streamline what we want to do, e.g. what kinds of images
- Unanswered questions
  - What are the exact requirements of the document manager role? (e.g. GitHub documents or our written documents)
  - Are the Peak Finder website's APIs publicly available?

Topics To Be Discussed With Khalil

- Ask for help finding a customer, customer contact role
- Discuss project management
- What are we actually expected to make? (ie. web app, stand-alone app)
- How will we get the raw data?
- Are we meant to be fully locating an image or just be able to answer some questions about the image?
- Should we assign one member to focus on each different tool?
- Thoughts on us implementing our own versions of tools for simple cases.

Next Meeting Plan

- Provisionally scheduled for 19/10/2023 at 15:00
- Confirm project name and group name
- Finalise project roles
- Review research
- Further specification planning

Group Meeting 2

Date: 19/10/2023
Time: 15:00-15:30

Topics Discussed

Specification Writing Responsibilities

- Introduction/Background Project Statement - Matthew
- Preliminary Research - Matthew
- Methodology - Dan
- Objectives - Everyone
- Group Structure - Lara
- Risk Management - Lukasz
- Tools and Resources - Lukasz
- Schedule/Planned project timeline - Dan
- Project Evaluation/How we will measure success - Lara
- References - using standard Latex style

Planned Project Scope

Definite Ideas:

- Street identification/street names
- Number plates
- Shops and brand names - would likely need to use an online service (Google Colab Vision) to provide this instead of implementing our own model;
- Point of interest identification (buildings) - could use the GeoNames service
- Ranking algorithm - give confidence metric to our predictions - this could provide a safer option for a project in regards to utility, e.g. if we can only provide a partial solution, a confidence metric would provide helpful suggestions to the customer
- Reverse image search using existing APIs
- Graphical interface

Maybe Ideas:

- Road markings - This may be a challenge as we are unsure whether there is an existing database we can use for this.
- Photo colour - could we use this for date/time identification? e.g. evening sun, fallen leaves, snow. Identifying the season could be useful to narrow down a time frame.
- User input - if the user already has some information about the photo, it could help us provide a more accurate prediction
- Use of metadata
- Denoising the image - could be helpful to make the image clearer. The algorithm for this wouldn't be too much work and would be applicable to anything. Potentially could use an existing interface online.
- Implementing PeakFinder - can we do this if we create just a program? Potentially could pass in the url.
  - Is there actually much point in using PeakFinder? We need to know the location anyway to use it. Maybe we could extract the data from it.
  - PeakFinder could be an end goal after the urban cases are implemented.
- General locations (city, wild area, beach) - Kaggle has resources for identifying general locations. Could maybe use colours but this might be misleading, for example, parks in urban areas

Definitely Not Ideas (To mention in the evaluating section in project management) :

- Tree identification - Likely too challenging and will be too time-consuming.
- Indoor spaces - likely too hard and challenging for this project, distant trees are nearly indistinguishable

Project Management Plan

Methodology:

- Overall incremental development design, with each feature being designed in an agile manner
  - This will mitigate project delays caused by one feature being abandoned - ie. we can get rid of a feature even if started developing without having delayed other features' development
- Test-driven development style
- Assign one person for each tool development but adopt a collaborative pair programming strategy, with continual group checking in to monitor progress - agile style methodology

General Schedule:

- By progress report - implemented number plate recognition and reverse image search using APIs. Want at least two features functioning so we have something tangible to show.

- Start by implementing features/models one by one - these need to be ready to incorporate into the graphical interface
- Ranking algorithm - developed later because this needs to be compatible with every feature
- Graphical Interface? When?

Using Google Cloud Vision:

- Positives
  - Good for utility purposes to gain extra information about an image. ie. take our image from our model and run into it to gain extra information
  - Large database
- Negatives
  - Using other services introduces risks because we are becoming reliant on the service being available for use throughout the entire project duration

Actions To Be Taken

- Dan - decide a group name
- Finalise project roles after meeting with Khalil
- Everyone - write their sections of the specification
- Everyone - proofread others sections

Topics To Be Discussed With Khalil

- Using PeakFinder? Should we make this our end-case stages?

Next Meeting Plan

- Group meeting provisionally scheduled for 26/10/2023 at 15:00

Supervisor Meeting 1

Date: 20/10/2023
Time: 10:00-11:00

Questions To Ask

- Ask for help finding a customer, customer contact role
  - Role requires time-to-time contact; we give them some use cases and hear what they think
  - Could simply be a friend/family member. 3rd person who has different perspectives to us (e.g. not a computer scientist) Business/journalist could be good.
  - Wider examples - missing person/search and rescue/people who want to verify images taken at an event (e.g. terrorist attack) are true
- Discuss project management - roles, risk management, timeline
  - Test-driven development - good and agreed about test-driven lets us expand our scope as the project goes if time-feasible.
  - Think of each development stage as phases - within each phase apply agile methodology
  - Can move back a phase to add in new features - once we have something that is working well and can be used to demonstrate (for phase)
- What are we actually expected to make? (ie. web app, stand-alone app)
  - Testing could be stand-alone
  - Commercial and long-term - website - very nice interface/very user-friendly
- How will we get the raw data?
  - News, social media, kaggle, have a look into how other applications have trained their models maybe
  - At least 1000 pictures
- Are we meant to be fully locating an image or just be able to answer some questions about the image?
  - Want to extract as much information as possible. Will not be able to locate some images.
- Should we assign one member to focus on each different tool?
  - Did not discuss
- Thoughts on us implementing our own versions of tools for simple cases.
  - Not a good idea as it will waste time
  - If it hasn't already been done - then yes could be useful.
- Using PeakFinder? Should we make this our end-case stages?

  - We could use it by - first, google maps/other sources to try find similar images. Then, as a final check, use PeakFinder.
  - Likes the idea of using the ranking process to send multiple photos into Peakfinder based on these results.
  - Photo compare - angles of mountains etc. Used to exactly locate/provide distance estimate from mountain etc.

Other Topics Discussed

- Project utility - dismissing fake news on social media
- Successful project -
- impossible to create fully automated project - because can never rule out all possibilities (might need tools which aren't in automated system/ journalists etc)
- cross-reference with other sources - social media, news headlines
- defo start with easy cases to ensure project is successful. show easy cases to solve - and how we can exactly locate image . can get exact match instantly
  - famous buildings = easy cases - across different continents?
  - but challenges as some famous buildings have replicas
  - would be good to show a comparison of output for a true and replica photo and show how the project identifies differently - e.g. from using surrounding buildings information and not just using only the main point of interest building
- outputs short report with a percentage confidence - important
- desert/sea = all looks the same so way too hard.
  - sand dunes aren't constant (natural factors like wind - impact image appearance in same location)
- focus on urban areas, mountains for search and research
- look to dcs proceeded resources for gpus/hardware
- format of number plate is good idea as extra step
- military uniform can tell us what army they belong to - could give an idea about which country BUT must consider that impersonators could dress in these uniforms
- out of scope of project - method to verify if photo/video if fake. but worth mentioning. could use an existing available tool to authenticate images
- image enhancement is a good idea
- put most effort into designing a nice interface with all these tools interacting together

- from a project managerial perspective - customers who want product are not interested in technical stuff, they want to see some output from an image. So, interface/report is very important - extract the most amount of info. as possible
- For progress report - the number plates/cars is good to get done by; also alternatively buildings; tourist areas
- risk regarding tools availability
  - unlikely for all tools to not be available anymore
  - normally always an alternative resource providing the same functionality
  - for major tools - unlikely their companies will stop funding their tool (especially due to nature of real-life applications)
  - often offer a paid version as well
- OSINT framework - tracelabs - is an example of need for these kinds of project
- legal etc issues

Group Meeting 3

Date: 02/11/2023
Time: 15:30-16:00

Topics Discussed

- Tasks to do before Christmas
  - Is on Jira, done by Dan
- Lukasz has done a trivial design of the reverse search engine thing. Landmarks will come under the reverse search
  - have a test image
  - pass into API
  - will give a bunch of similar results back
  - image comparison like SSID, SIFT, feature matching algorithms to get an idea of how similar. a lot of these already pre-made for us, scores the reverse search
  - weighted key words (weighted locations)
  - verify against Geonames
  - so does reverse search and also incorporates scoring systems straight away
  - challenge - captions don't always have locations and could have other words. e.g. "image of ..." so need a way to eliminate "false words", using an online tool
  - Lukasz to take this feature
- Number plates
  - OpenCV library to use
  - Lukasz happy to be the "review" side of this process
  - Challenge will be sending it into databases to find out more about the numberplate - ie. need to identify which country database to send image to. Needs research.
  - Extracting text should be simple.
  - Matthew to take.
- Lara metadata
  - Dan review side
- GUI - collaborative
  - Dan to identify a good tool for us to use. Dan doing the preliminary stuff.
  - Need at least a small demo of GUI for presentation
- Image authentication verification system

  - Pre-made tool
  - some existing libraries on GitHub for this
  - Dan to take
- Enhancements if possible if ahead of schedule for presentation - low priority
- On Jira, can create a branch on the Github
  - good to show us on Jira which feature is being developed in each branch on Github
  - Each card has its own naming label - use these labels at the start of each branch etc. To enforce naming system.
  - can attach files which will be useful for our final report
  - everyone must include few user stories on their cards to help with user development
  - when finish code, do a pull request, as linked with code review on cards. for someone else to "responsible review" code. Each card has a reviewer name to be assigned.
  - on backlog, can create new card to put low-priority tasks.
- first increment ending when presentation due
- within two weeks have our individual features done. then a week for integration.
- Reviewers - assigned on Jira,

Group Meeting 4

Date: 10/11/2023
Time: 13:30-14:00

Topics Discussed

- meet with Khalil next week Thursday instead of regular meeting
- working demo for number plates 50% success rate
  - shared number plate format between various places
  - could we query all possible countries using given format and compare car information against car in image?
  - top 10 most populated countries using given format?
  - use restricted version for now until demo with explanation of problem
- use of Electron to produce web-app contained within desktop program
  - cross-platform Windows and Mac
  - uses NodeJS and other standard web technologies (HTML, JS, other JS frameworks e.g. React)
  - easy to export to web-app if required later
  - looking into how to integrate with a Python backend to incorporate our Python features
- Lukasz looking into SerpAPI for reverse images
  - image can't be locally stored, must be stored externally
  - method of programatically uploading to imgur or google drive?
  - SSID to compare general gist of query image and reverse search images
  - when SSID determines closely aligned, use of SIFT (Haar features) to find features shared between images
  - use of Geonames API to get more detailed location information
- aim for features complete by end of next week
- focus on presentation during week 8
  - technical stage: what are the capabilities of the system?
  - go through case studies (one or two case studies which work for all features we have developed)
  - "smart" attire
  - think of team name by then
- remember to update tickets on Jira
- ImageLab might be useful for training data

Topics To Be Discussed With Khalil

- are we on the right track?
- give him general update

Supervisor Meeting 2

Date: 26/11/2023
Time: 15:30-16:00

- presentation = 5th december 10am
- renaming project to include geolocation to make more specific
- discussed what we have been doing
- perceptual hashing techniques - not similar to what we are doing but big companies are looking into these
  - for image classification to find which images are similar
- get image - check if modified and to what extent - difficult task
- mention this but say that this is not part of the project as it is a very complex and difficult task.
  - attackers can take image, modify slightly so that the hash is very different and so not being looked for by these big companies
- how do we deal with modification? in our case we do not deal with this as not part of our aims to try and figure out if modified and to what extent - scope
- very basic base cases for the presentation demo
  - give an example of an image which has been modified - can check with a different tool
  - an example where the image is intact and contains all the metadata
  - an example where the image hasn't been modified but the metadata has been removed - to show all the processes we have developed
- presentation - how much project management to include?
- wayback machine
- social media accounts - Instagram, twitter - only if free to use
- use what other people have already done
- communicate via email, meet at presentation

Group Meeting 5

Date: 23/11/2023
Time: 15:00-16:00

Topics Discussed

- How can we make sure the program knows we know enough to locate
  - e.g here are some suggested words, here are some suggested locations
- after we do the image we should delete it for privacy reasons
  - either delete images after they log off/ users can manually delete/each image stored in a database so just delete after a certain time
- each user will have their own files as stored on users machine - not uploaded to an online server - so privacy reasons not that critical at the moment
- give them option to do all analysis in one and also an option to do all at one go - ie. general analysis
- how long will it take for each analysis - assuming semi-lengthy analysis is okay as complex analysis software - seems reasonable for real-life conditions.
  - progress bar? or estimate time maybe
  - this usually takes xxx minutes note
- for fetching photos to improve efficiency - could cut off after a certain time limit - future thing - Lukasz part of project
- refinement of sections
- integration
  - how challenging would it be for us to incorporate user comment right now
  - can a function be passed in as well for Lukasz at this stage
- output
  - Lukasz - json object - can extract data out of these alternatively
  - on screen - want to display top 5 words from Lukasz's histogram - could be an array or a string. can store these in Django database - keywords and then can loop over this
  - use geonames results as well - take top 3 and then put into a format which is useful
    - key info - name, country, latitude, longitude
    - we suggest looking into these three etc
- requirements.txt - for all pip install stuff
- interactions of features for next iteration

- image altering detection - ghost detection
  - for now we can output this image with the main one and let user decide for themselves, then as a future plan will be to determine ourselves using neural network
- keep updating kanban board
- this week
  - integration
  - lukasz streamline his output
  - presentation start during week - meet Monday in leam

Group Meeting 6

Date: 19/01/2024
Time: 14:00-16:00

Topics Discussed

- Plan for term
- when to do input clues
  - will we take user input as true? not assume
  - keep separate and then use the clues to filter our predictions - give user an option maybe
- feedback said to make system more automated
  - perform all analyses at once and give options for user to do separately still
- how can we predict the exact gps location - google street maps train ? leave to decide later
- separate shop sign and street sign
  - shop sign can help with finding exact gps - google cloud vision can do this
  - CNNs
  - google cloud vision has stuff for street signs too
- from feedback - make notes as to why we have chosen certain methods when developing
  - and why the tools we use is the best one to use out of all the options
- sprint 1
  - interface updates - Dan
  - street signs - Matt
  - shop signs - Lara
  - image enhancement (fairly small) -Matt
  - input clues - Lara
  - mountain stuff - Lukasz (and Dan)
- Lukasz wants to use geonames but create own search function
  - can we use own database if possible - geonames lets you download their database
  - has someone else already created a better search engine
- Can we use a google api
- cloud vision - 1000 free searches a month for each feature

  - returns a list of words from the image - can we use this with geonames
- reverse image search from geonames returns coordinate - can put on map and display
- confidence ranking will be our major contribution
- Social media from feedback - bing api has a filter so could select social media domain
- next week have ideas and system designs ready
- we want to start testing sooner

Group Meeting 7

Date: 20/02/2024
Time: 16:00-:1700

Topics Discussed

- integration
  - how best to integrate react - don't want to if we don't need to
  - using stuff from Lukasz mountain research
- mountains status
  - contour maps - not going to work, trying to predict contours from 2-d images will just not give a good result, just too much computation for what we can currently achieve
  - using reverse image search to match up mountain to location
    * works in certain locations like Mont Blanc
    * works fine with google search for less famous mountains but not so much with bing
    * not a great solution
  - using digital elevation model
    * simple matrix of an area with its heights
    * take image and peform simple skyline .. identify peaks etc
    * essentiallty form curve and feature matrix for skyline and compare with digital elevation model
    * good as maths so range but similar peaks would give false positives
    * from paper in 2012 so fairly old
    * worked well for switzerland but this is a smaller area compared to world
    * so good if we can narrow down area with hints - say if someone lost and sos
    * lots of pre-processing required which takes time
  - using cross-locate library
    * uses deep NNs to match image against its database
    * could try create our own NNs based off this
    * the database gives image of mountain and the exact coordinates of where camera was - handy for projection stuff with distances
  - overall process - need to find with mountain and then project it for distances
    * corrective measures for projection
  - if choosing cross-locate as easiest
    * limited existing datasets for coordinates of camera and mountain for training
    * could be good to show we can do it for a smaller area but explain limitations, data is coming out constantly
  - combine in confidence scoring, maths and digital vision

  - existing papers are quite advanced, especially the mathematics
  - if we can get cross-locating tool working it would be a good for project
    * they want us to use existing tools but how can we use and still contribute code without just copying simply
    * dataset has 3000 photos with depth set, exact camera locations, silhouette maps, they use the three modalities to generate extra data
    * it works because the dataset is so advanced
  - transfer learning could be worth a try
    * but dataset limitation still

- street sign
  - pointers on map done
    * potential locations
    * pass in potential coordinates
    * just from street signs but can integrate others
  - next steps would be user interface

- geonames too restrictive of its searches

- shop signs
  - certain shop brands are only in certain countries
  - with some shops
    * see if api for their shop locations with their company
    * see if they have really close locations for some to narrow down
    * will be more helpful to use with other findings - e.g. done number plates to find it is UK
  - confidence stuff
    * could plot each feature colour coordinates to give visual aid
    * behind the scenes - eliminate down when many many options
    * produce ranked places
  - user interface is important
  - image enhancement
    * some steps been done but not all added to ui as many buttons
    * do we want to feed these into the program or will this be an issue as not all features will want an enhancement
    * could automatically enhance and save and then test which enhancement works best for each feature
      · give user some control over the enhancements used still - dropdown button

Supervisor Meeting

Date: 26/02/2024
Time: 15:00-16:00

Topics Discussed

- Recap of work currently doing
- Discussion about mountain work
  - agreed that it's better to do for a small area than not at all - for future extensions with increasing datasets
  - keep working on the Alps - if algorithm works well on this region then it could be scalable in future
  - algorithm isn't our own - compare another approach with this one to provide justification to accuracy etc
  - using homography and matrix projection too reduce this algorithms error margin - up to us, as long as we have some good results following a strategy then worth it.
  - agreed coming up with a new algorithm is not easy
  - sources of images - website or independently taken. cross-locate team have two datasets. we need to make sure dataset is accurate - attacker could insert false data during training phase so we need to validate datasets used
  - any trouble about topography changes with seasons - the depth map should help prevent this. but they have said snow can increase challenges
  - focus on regions, then seasons and then try and narrow down further to see how solution performs
  - human intervention inevitable at this stage to ensure it matches the true geographic area we are looking for
- discussion about street signs work
  - different continents - seems to work well, works lightly better for English speaking countries
  - challenges with repeated place names in different countries - maybe extract information from buildings surrounding, street, weather, time to infer climate. use multiple sources to form conclusion
- discussions about user interface
  - improvements to prevent page refreshing
  - map use - good visual for the users. ideal displaying multiple possible locations because unlikely whole area will be a good possible solution - e.g. we should be able to locate (narrow down to a few possible locations) a mountain given an area. give a finite number of precise points - unless too challenging (e.g. desert or in snow-covered mountains).

  - how do users narrow-down/conclude correct location - maybe double-click can open another page/web-page to display the location using another source like google-maps for verification

- data will be an issue as not enough and geography always changing. military sector continually re-surveying areas for these reasons - and this data kept secret.
- alps = rescue teams will have access to data. possibility that they share these.
- agrees deep neural network will be best in our case but limited with training time and available data. but accuracy will be much better so will make worth it.
- could be the case that all the possible locations are wrong - realistic in some cases. can then be double checked manually by users / experts in the terrain.
- agrees that this is a semi-automated tool not fully-automated.
- should we spend time trying to detect generative AI - this is a difficult problem here with deep fakes. can be used to poison datasets. worth a mention but don't take into consideration when building our solution. we assume dataset is correct with high accuracy at this point. with metadata- analysists double-check for dataset validity. if we don't make this assumption we will need to go through lots of checks. we assume images haven't been altered - this would make this a different problem/project to solve as would need to prove authenticity.
- regarding finding exact coordinates of camera
  - well-known cases like eiffel tower it should be easier to estimate how far the image was taken from
  - feasible to narrow-down cases. at least give an estimate of distance and the direction.

Group Meeting 8

Date: 11/04/2024
Time: 15:00-16:30

Topics Discussed

- UI: separate tabs (rural, urban, enhancements)
  - checkboxes?
  - will depend on time limit
- duplicate spec in overleaf for final report
- Dan UI
- Matt confidence system
- Lukasz mountains
- Lara start report template
- dev should be finished within a week (17th April)
- poster deadline to send for printing is 24th/25th
- need to bring everything together
- testing to see whether it works, and how well it works
- test by cases (performance evaluation) - aligns with the way we developed it
- link tests to requirements to prove they've been fulfilled (test plan)