MSc Data Science Project Report

An On-Demand Satellite Earth Observation Imagery Retrieval and Analysis Pipeline

Joshua Stephenson

MSc Data Science Project Report, Department of Computer Science and Information Systems, Birkbeck College, University of London 2016-2018

Word count (excluding code and appendices):

11,652 words

Acknowledgements

I would like to thank Tom Jones and Joana Kamenova from the Satellite Applications Catapult for their advice throughout the project and the industry contacts that they have connected me with, my supervisor Steve Maybank for his support, and Sentinel Hub by Sinergise for the research account privileges they granted to me for the purposes of this project.

1 Contents

2 Abstract

Satellite Earth Observation (EO) is the gathering of information about our planet's physical, chemical and biological systems through satellite-based remote sensing. This is a rapidly growing industry that has been bolstered in recent years by technical, economic and political developments, with continued innovation from both governments and the private sector. As a consequence, EO imagery data is becoming available on a much greater scale than ever before.

Initially inspired by the opportunity to use this data to detect and monitor changes in the politically charged regions of the South China Sea, the project provides an on-demand satellite EO imagery retrieval and analysis pipeline tool that can be applied on an international scale.

The tool utilises data from both open-access low-resolution imagery sources and industry leading high-resolution commercial imagery providers, adapting state-of-the-art academic research and cutting-edge tools and techniques to analyse the imagery retrieved and present this to the user.

Supervisor: Steve Maybank

3 Introduction

3.1 Emergence of satellite-based remote sensing

On 4th October 1957 the world's first artificial satellite, Sputnik 1, was successfully launched into low Earth orbit by the Soviet Union [1]. Despite only lasting for 22 days this achievement inaugurated the US-Russia space race and the period of development in this field that followed. By 1980 satellites were being launched on vehicles not wholly state-controlled [2] and since then the industry has seen exponential growth, with around 180 Earth Observation (EO) satellites launched in 2007-2016. An estimated 600 launches by nearly fifty countries [3] is anticipated in the decade to follow with the UK Government launching its first radar satellite, NovaSAR, on 16th September 2018 [4].

Several factors in recent years have supported the expansion of this sector. Significant technological developments such as affordable rocket launch technology, for example Elon Musk's reusable Falcon 9 [5], have made commercial projects economical. Hardware improvements including advances in miniaturisation of sensors have led to lighter satellites and also allowed nano-satellites to collect some of the same data as traditional larger satellites [6]. Large-scale distributed data warehousing and mining capabilities such as Amazon Web Services (AWS) and Hadoop have made processing and analysing data on this scale possible. All of these developments have driven up demand and innovation in the field at an accelerating rate.

3.1.1 Earth Observation and object detection

Earth Observation (EO) is a sub-field of satellite-based remote sensing covering a broad range of land cover and land use applications such as urban change detection, carbon biomass assessment, ocean management, disaster and disease response, and air quality monitoring [7]. This usually involves quantifying some aspect of large areas of land and does not typically depend on very high-resolution data.

In 2014 the US Government effectively loosened restrictions on the spatial resolution of commercial satellite imagery, lowering the highest permitted resolution from 50cm per pixel to 25cm per pixel and allowing DigitalGlobe to sell images at this quality [8] [9]. At this resolution the types of object that can be identified from satellite imagery is vastly increased. Innovative applications, such as measuring seal populations in Antarctica as an indicator for coastal ecosystem health [10], or monitoring disease trends by counting cars in hospital carparks [11] continue to be conceived and tested.

3.2 Project deliverables and objectives

There were two key aims for this project. Firstly, to gain an understanding of satellite EO imagery analysis and the cutting-edge open source and commercial technology that is available. Secondly, to develop a tool that is relevant to the current challenges and opportunities and state-of-the-art research in the field.

The project delivers a proof-of-concept tool enabling a user to retrieve, view and analyse satellite imagery data via an interactive view of a map and User Interface (UI) in a two-stage process.

In the first stage a low-resolution satellite imagery mosaic with minimum cloud-coverage levels is generated for two pre-specified timeframes over a pre-specified area-of-interest (AOI). Once this is returned two types of land cover classification algorithms

are applied to provide the user with an indication of potential change of interest over time.

In the second stage a high-resolution satellite image with minimum cloud-coverage levels is returned over a smaller pre-specified AOI and a state-of-the-art object detection algorithm is applied.

The tool can also serve as a prototyping environment for testing future developments.

Accompanying this tool, a detailed project report is provided that outlines the system design and architecture, the implementation process, and the testing procedures undertaken.

3.3 Project trailer

An exhibition of selected features of the project tool is provided below.

3.3.1 Tool initialisation

After following the set-up steps in section 11.4 and initiating the tool the UI and graphical map in Figure 1 is displayed.

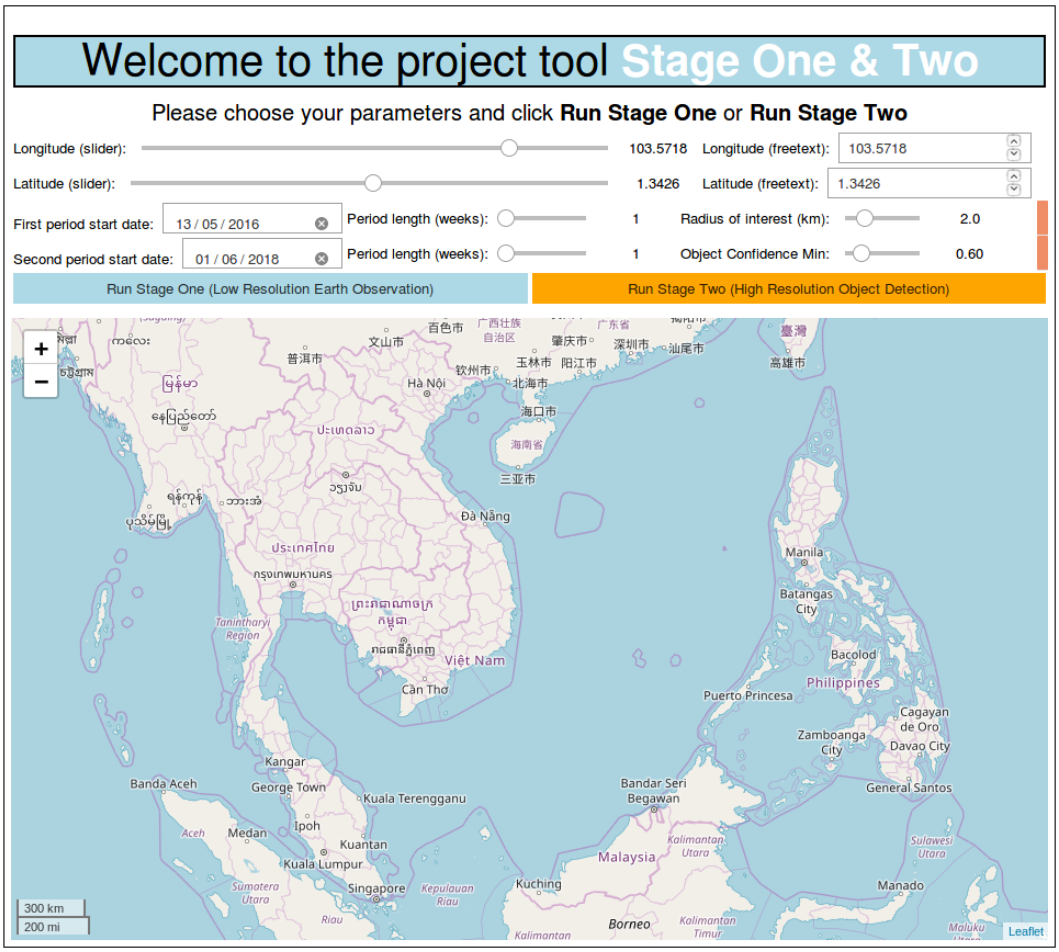

Figure 1 - User Interface and graphical map upon initialisation of project tool

The user can interact with the map by clicking '+' and '-' buttons to zoom in and out, and by dragging the map to pan. If the user clicks on the map the latitude and longitude

coordinates at that point are shown. This functionality is demonstrated in an example over Ho Chi Minh City, Vietnam in Figure 2.

Figure 2 - Zoom, pan, and mouse click functionality of project tool

3.3.2 Stage One

Once the user has identified the latitude and longitude coordinates of their AOI they can interact with the UI to choose the parameters for requesting satellite imagery. At Stage One the project tool will read the input from the 'longitude' and 'latitude' sliders (which have been linked to the freetext boxes), the 'period start date' fields, the 'period length' sliders and the 'Radius of interest' slider. In Figure 3 these have been set to non-default values, and when the 'Run Stage One' button is clicked the process of retrieving and analysing Sentinel-2 satellite imagery will begin.

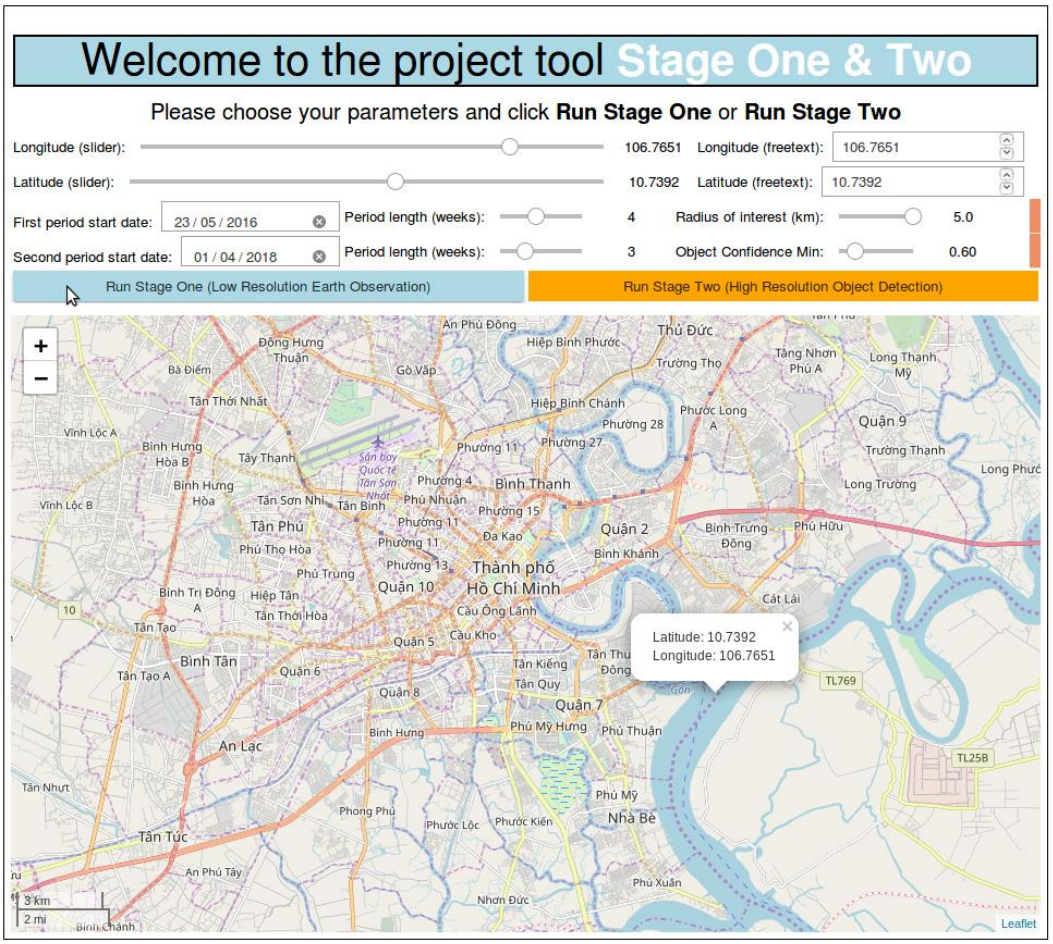

Figure 3 - Parameters chosen and Stage One button highlighted

Once the user has clicked the 'Run Stage One' button the project tool will print the user inputs and retrieve and analyse the satellite imagery, providing progress updates at various stages. If there is more than one image available for a given timeframe the tool will produce a mosaic of imagery prioritising lowest cloud cover. Figure 4 gives an example of the resultant map displaying a low cloud imagery mosaic once this process is complete.

Figure 4 - Stage One low cloud imagery mosaic feature layer

In the top right corner the user can choose which feature layers to display and toggle ‘full-screen’ mode on/off, a close-up is provided in Figure 5.

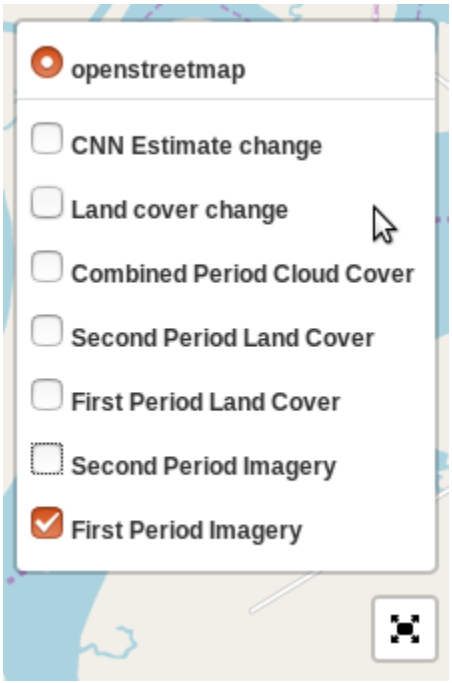

Figure 5 - Close-up of feature layer panel and 'full-screen' button

One type of analysis layers is the result of a Classification and Regression Tree (CART) algorithm. An example of this feature layer is shown in Figure 6. Pixels classified as urban are coloured red, vegetation is coloured green, and water is coloured blue.

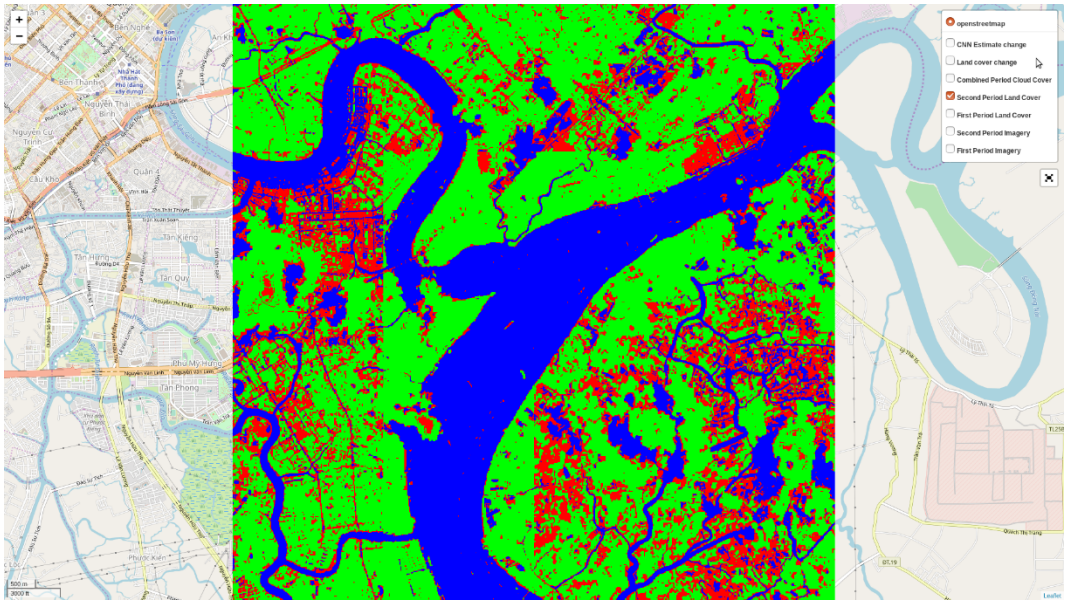

Figure 6 - CART land cover classification feature layer

The CART-estimated land cover proportions of each tile in the imagery mosaic are compared between the two timeframes and the 'Land cover change' feature layer, shown in Figure 7, highlights to the user the areas which have seen the highest amount of change.

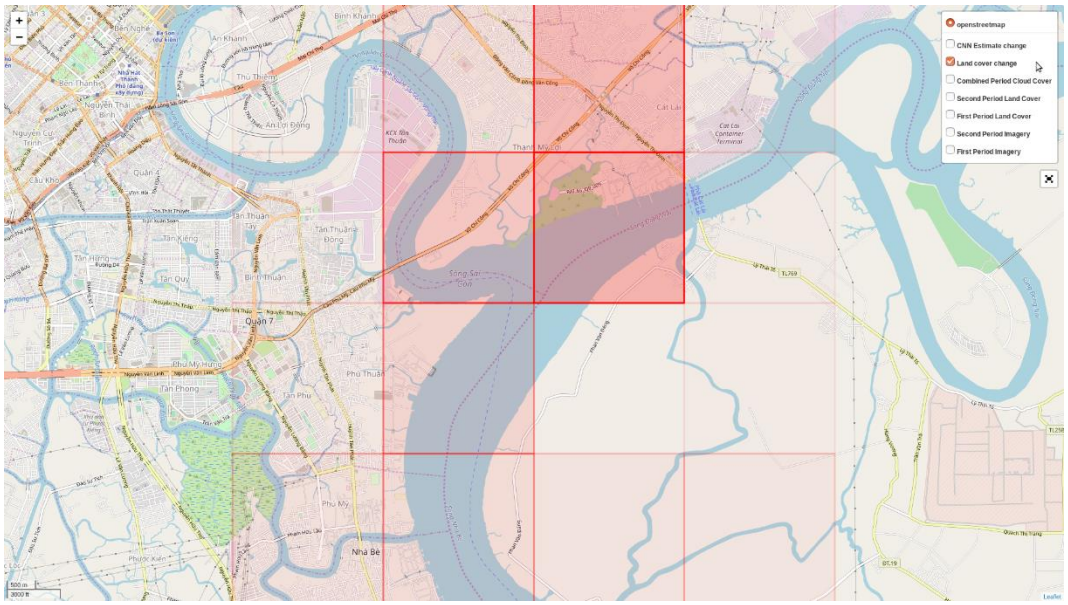

Figure 7 - Land cover change feature layer example

A deep residual Convolutional Neural Network (CNN) is applied to estimate whether tiles contain 'man made structures' or not. The 'CNN Estimate change' feature layer compares these predictions between the two timeframes and highlights the difference to the user in a similar manner to the 'Land cover change' layer. When this layer is displayed a mouse click will show the CNN estimates for each timeframe, as illustrated in Figure 8.

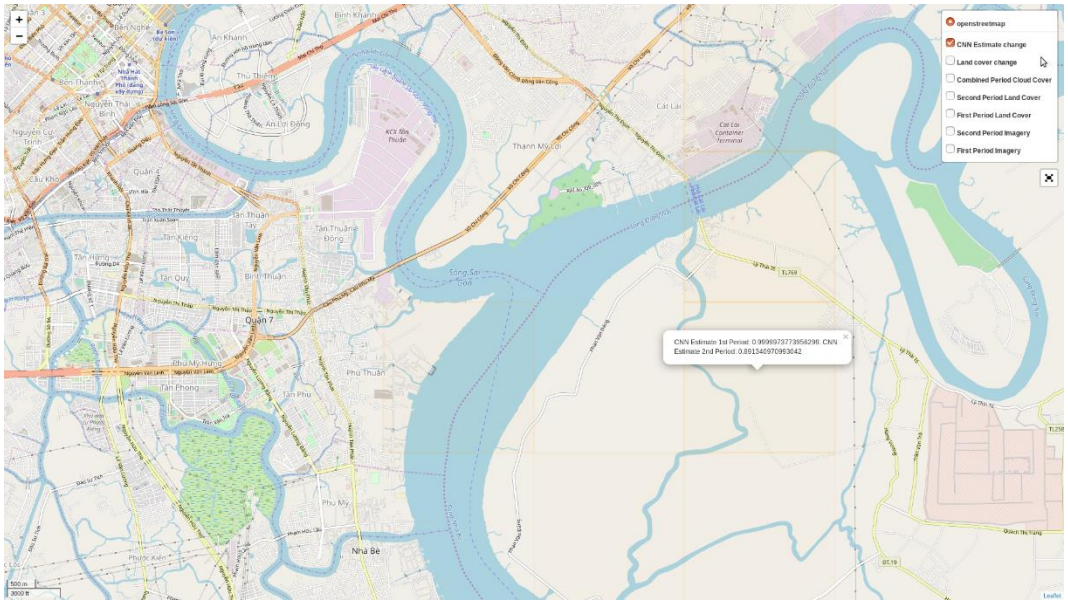

Figure 8 - CNN Estimate change feature layer. Polygon opacity is low due to low change detected.

3.3.3 Stage Two

The user can also initiate Stage Two of the project tool from the same UI. At Stage Two the project tool will read the input from the longitude and latitude sliders and the 'Object Confidence' slider. Figure 9 shows the output of the project tool once the user has set 'Longitude' = 106.6641, 'Latitude' = 10.8171, 'Object Confidence' = 0.5, and clicked the 'Run Stage Two' button.

Figure 9 - Stage Two High Quality image layer with object bounding boxes

At all stages in the pipeline the user can pan the map and change the zoom level. In Figure 10 the imagery and analysis returned at Stage Two has been inspected at a higher zoom level and with 'full-screen' toggled on.

Figure 10 - Closer zoom of Stage Two imagery and analysis layer

3.4 Project evaluation summary

This project was particularly challenging due to my unfamiliarity with the subject matter, the cutting-edge nature of the field, and the scale of the deliverables I set out to achieve. When starting this project I had only a limited knowledge of Computer Vision, satellite EO, or cloud computing technologies, and so to deliver a functional and high-quality tool a significant amount of my time was devoted to independently researching these areas and testing the relevant state-of-the-art technologies in these fields.

Through contacts made via the Satellite Applications Catapult and successful research applications to the European Space Agency (ESA) and AWS I was able to gain access to the cutting-edge tools and industry-leading imagery data necessary to deliver an otherwise cost-prohibitive project.

The project achieved its main goal which was to investigate the field of satellite EO and demonstrate a proof-of-concept tool that accesses and analyses low-resolution open imagery data on a larger scale and then high-resolution commercial imagery data on smaller AOIs following this. The project could be enhanced with a better-quality set of labelled data upon which to train the algorithms that are applied, and from an improved UI. These and other recommendations are provided in more detail in section 11.3.

3.5 Roadmap for the remainder of the report

The subsequent sections of the report detail the relevant outputs from this project. The contents of each are outlined below:

- **Section 4**: Background research and literature review

This section focusses on the context of the report. The background research section details the imagery data available in the field of satellite EO. The literature review section introduces several of the techniques and concepts used in the project.

- **Section 5**: Project Specification

This section outlines the key functional requirements, de-scoped functionality, and key dependencies of the project tool.

- **Section 6**: System Design and Architecture

This section provides a high-level description of the architecture of the project tool and the map rendering and UI design.

- **Section 7**: Implementation

This section details all aspects related to the implementation of the project tool. This includes the technologies, tools, programming languages, and servers used.

- **Section 8**: Testing

This section provides details of testing additional to that in the implementation section, including unit testing and user acceptance testing.

- **Section 9:** Evaluation and Conclusion

This section provides an overall evaluation of the project and includes a critical evaluation, lessons learnt and conclusion.

4 Background research and literature review

This section focusses on the context of the report. The background research section details the imagery data and existing solutions available, following this the literature review section introduces state-of-the-art Computer Vision techniques and concepts that are used in the project.

4.1 Background research

4.1.1 Open satellite data

The first stage of this project focuses specifically on EO using open satellite data. Two of the most prominent open data sources for this type of imagery are the Sentinel-2 and Landsat 8 missions, coordinated by the ESA and National Aeronautics and Space Administration (NASA) respectively. The satellites in both missions host a range of multi-spectrum imaging sensors.

4.1.1.1 Landsat 8 Operational Land Imager (OLI) and Thermal Infrared Sensor (TIRS)

NASA's Landsat program is the longest running enterprise for acquisition of satellite imagery of Earth [12], beginning with the launch of Landsat 1 in 1972 and documenting decades of global change through six successful launches since. The most recent satellite mission, Landsat 8, was launched on February 11th 2013 with a 5-year lifespan and up to 10 years of fuel on board. The mission's main objective is to provide data continuity of the earlier Landsat 4, 5, and 7 missions by capturing timely and high-quality images of all landmass and near-coastal areas on the Earth [13].

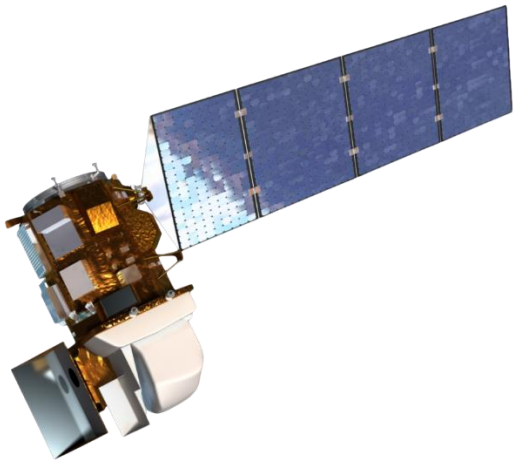

Figure 11 - Landsat 8 Satellite model. Source – NASA [14]

Landsat 8 carries two sensors: the Operational Land Imager (OLI) and the Thermal Infrared Sensor (TIRS). These instruments capture 9 spectral and 2 thermal bands respectively as shown in Table 1, with a swath width of 185 kilometres and revisit time of 16 days.

Table 1 - Wavelengths and Bandwidths of the Landsat 8 OLI and TIRS instruments. Source – USGS [15]

Spatial Resolution (m/pixel)	Spectral Band	Landsat 8	
		Wavelength Range (nm)	Sensor
15	08 – Panchromatic	500 – 680	OLI
30	01 – Coastal/Aerosol	433 – 453	OLI
	02 – Blue	450 – 515	OLI
	03 – Green	525 – 600	OLI
	04 – Red	630 – 680	OLI
	05 – NIR	845 – 885	OLI
	06 – SWIR	1,560 – 1,660	OLI
	07 – SWIR	2,100 – 2,300	OLI
	09 – Cirrus	1,360 – 1,390	OLI
100	10 – LWIR	10,300 – 11,300	TIRS
	11 – LWIR	11,500 – 12,500	TIRS

The data captured by Landsat 8 is downlinked and processed into high-quality data products within 24 hours of acquisition by the Level-1 Product Generation System at the Earth Resources Observation and Science Center. The processing uses inputs from both the sensors and the spacecraft to remove various distortions such as view angle effects, altitude deviations, and Earth curvature. These images are suitable for pixel-level time series analysis [15].

4.1.1.2 Sentinel-2 Multispectral Instrument (MSI)

The Sentinel-2 mission comprises two identical polar-orbiting satellites launched on 23rd June 2015 (Sentinel-2A) and 7th March 2017 (Sentinel-2B), both with a 7-year lifespan and up to 12 years of fuel on board. The pair are phased at 180° to each other, each carrying multispectral 'high-resolution' imaging sensors covering the 13 spectral bands shown in Table 2 for the purpose of land monitoring. They are part of a series of next-generation EO initiatives run by the ESA to support the operational needs of the Copernicus programme [16] and to provide continuity for the current Landsat missions.

Figure 12 - Sentinel-2 Satellite model. Source - ESA [17]

During development of the Sentinel-2 Satellites the ESA and NASA collaborated to cross-calibrate the instruments with those on Landsat 8, with the goal of allowing the scientific community to use data from the two sensor types synergistically [18]. A further aim of the Sentinel-2 mission was to improve upon the existing Landsat 8 Near-Infrared (NIR) band which was found to be heavily contaminated by water vapour and not sensitive enough to certain parameters [17]. Recent studies have shown that Sentinel-2 bands are more accurate than those of Landsat 8 when used for land use and land cover mapping [19] [20].

Table 2 - Wavelengths and Bandwidths of the Sentinel-2 MSI. Source – ESA [17]

Spatial Resolution (m/pixel)	Spectral Band	Sentinel-2A		Sentinel-2B	
		Central Wavelength (nm)	Bandwidth (nm)	Central Wavelength (nm)	Bandwidth (nm)
10	02 – Blue	496.6	98	492.1	98
	03 – Green	560	45	559	46
	04 – Red	664.5	38	665	39
	08 – NIR	835.1	145	833	133
20	05 – Vegetation Red Edge	703.9	19	703.8	20
	06 – Vegetation Red Edge	740.2	18	739.1	18
	07 – Vegetation Red Edge	782.5	28	779.7	28
	8A – Narrow NIR	864.8	33	864	32
	11 – SWIR	1613.7	143	1610.4	141
	12 – SWIR	2202.4	242	2185.7	238
60	01 – Coastal aerosol	443.9	27	442.3	45
	09 – Water vapour	945	26	943.2	27
	10 – SWIR – Cirrus	1373.5	75	1376.9	76

The Sentinel-2 mission captures a wide swath width of 290km and initially had a revisit time of 10 days. Following the successful launch of the second satellite the revisit time was halved to 5 days. It should be noted that as with Landsat 8 and other satellites carrying passive sensors, the quality and usability of imagery is highly dependent on the cloud coverage at time of capture.

Every day the Sentinel-2 Satellites capture 2.4 Terabytes of raw data, which is transmitted in compressed format to the ESA's Payload Data Ground Segment during satellite overpass. The Payload Data Ground Segment decompresses this data, applies radiometric and geometric corrections with sub-pixel accuracy to create a 'Level-1C' Top-of-Atmosphere (TOA) reflectance product [21], and archives the data for online access by users.

For certain geographical regions such as Europe a further processed 'Level-2A' operational product became available in March 2018. This additional processing includes a scene classification and an atmospheric correction to create a Bottom-of-Atmosphere corrected reflectance product. The ESA plans to gradually ramp-up the geographical availability of this product to systematic worldwide coverage in 2018 [22].

4.1.2 Commercial satellite data and existing solutions

At 10 m/pixel the best spatial resolution available from open satellite data is not high enough for detailed object detection. For these purposes the second stage of the tool will need to obtain imagery from other sources. Two of the leading Commercial satellite

data services offering imagery at a higher resolution and also platforms for analysis are Planet Labs and DigitalGlobe.

4.1.2.1 Planet Labs

Planet Labs is a commercial satellite data service that launched their first two satellites in 2013. As of March 2018 they operate more than 175 Dove satellites, 13 SkySats, and 5 RapidEye satellites currently in Earth orbit [23], delivering an industry leading level of coverage and scale with further launches planned for 2018 [24].

Their product offering includes medium, high, and open water resolution satellite imagery monitoring in GeoTIFF format, delivered to consumers either using their GUI platform 'Planet Explorer' or through their API cloud-based platform that integrates geographic information systems (e.g. ArcGIS, QGIS, Boundless).

Planet Lab's imagery data archive dates back to 2009 and has been successfully used to train machine learning and Computer Vision algorithms, for example in their Kaggle competition [25].

The highest resolution imagery that Planet Labs offer is at 0.72m/pixel, captured by the sensors on Skysats 3 – 13 in their satellite constellation launched between June 2016 and October 2017. The sensors have a ~6.6km swath width and capture imagery in 5 spectral bands – Red, Green, Blue, NIR, and Panchromatic.

Figure 13 – SkySat 3. Source – Skybox Imaging [23]

4.1.2.2 DigitalGlobe

DigitalGlobe is a publicly listed company with a similar product offering to Planet Labs. They own a constellation of their own high-resolution satellites [26] and claim to be the world's leading provider of multispectral Earth imagery content, supplying much of the high-resolution imagery which Google Earth and Google Maps have used [27].

DigitalGlobe uses AWS Simple Cloud Storage Service (S3) 'buckets' to store GeoTiff format data, accessed along with computational resources through their Geo Big Data Platform (GBDX) RESTful APIs, and although a costly paid service they offer limited free access in an evaluation account.

The highest resolution they offer is industry leading at 0.31m/pixel. This is captured by sensors on their WorldView-3 and Worldview-4 Satellites, launched in August 2014 and November 2016 respectively. The sensors on both satellites have a swath width of ~13.1km and capture the same spectral bands as Planet Labs' SkySats.

The sensors on WorldView-3 also capture 4 additional ‘visible and near-infrared’ (VNIR) bands: coastal, yellow, red edge, and NIR2; 8 ‘short-wave infrared’ (SWIR) bands that penetrate haze, fog, smog, dust, and smoke; and 12 ‘CAVIS’ (Clouds, Aerosols, Vapours, Ice and Snow) bands for mapping clouds, ice and snow whilst correcting for aerosol and water vapour [28].

Figure 14 - WorldView-3. Source – DigitalGlobe [28]

While both Planet Labs and DigitalGlobe offer platforms in which to analyse their data, their services require significant development and coding from the user and the non-commercial data is limited on each. Another existing solution is Google Earth Engine [29].

4.1.2.3 Google Earth Engine

Google Earth Engine (GEE) is a geospatial data processing/analysis platform, providing cloud-based access to many of the open satellite data sources such as Landsat, Sentinel, MODIS, and more. The engine is also a platform for Google's computational capabilities, extending large-scale satellite data access to those that lack the technical capacity required to utilise traditional supercomputers or commodity cloud computing resources [30].

To test the feasibility of this technology as a basis upon which to build the project tool I investigated using a Python interface for GEE access, which requires a container technology called Docker. Docker containers are useful for a variety of reasons, but Google encourages their use when deploying a Python Development Environment to allow others to be able to easily replicate any behaviour being experienced for collaboration purposes. After facing multiple issues running Docker locally I instead opted to run a Datalab Docker container on Google Cloud Platform. Despite initial success the outcome was still not favourable with the Google developers not actively maintaining critical Python objects such as ee.mapclient [31] which is used in the majority of their code examples and allows users to view data as map tile projections. This object was built on TK, a graphical UI toolkit which unfortunately behaves differently on different machines despite the use of Docker.

Although a valuable experiment, as a result of the cutting-edge nature of both Docker and GEE there are numerous issues still to be resolved for sufficiently reliable functionality of this product. Given the amount of time invested without successful imagery acquisition within a Python Development Environment I concluded that this cloud solution is not viable as a platform for my project. Despite that, the GEE Javascript

code editor and web-based IDE has proven to be a useful tool, revealing the poor South China Sea coverage of Landsat 8 during satellite data imagery exploration as shown in Figure 15, and also through the testing of a 'Classifier' package which handles various supervised classification algorithms such as decision trees, linear regression models, support vector machines, perceptrons, and naive Bayes models; an application of this package is detailed in section 7.4.4.

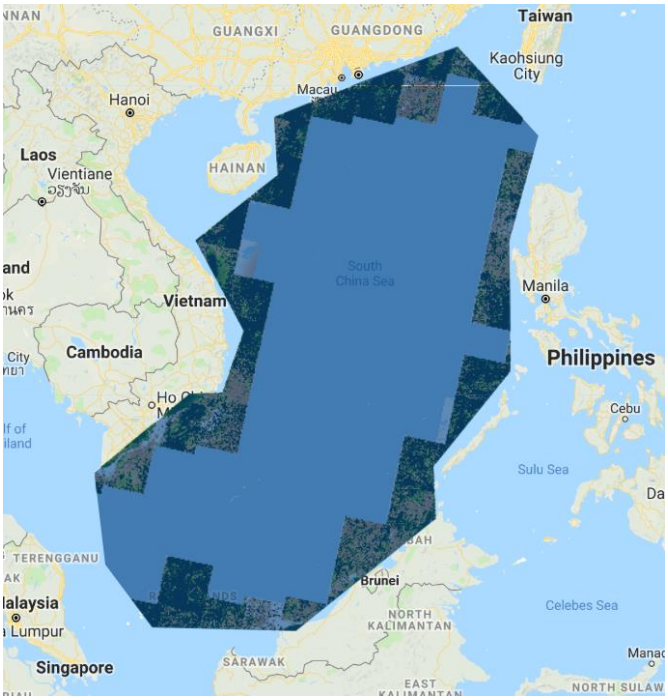

Figure 15 - Landsat 8 imagery cover in the South China Sea. Source – GEE [29]

4.2 Literature review

4.2.1 Computer Vision and Convolutional Neural Networks

Computer Vision is a scientific discipline that studies how computers can efficiently perceive, process, and understand visual data such as images and video. Applications include face and object detection, self-driving vehicles, 3D reconstruction from images and more.

Artificial Neural Networks (Figure 16) are a biologically-inspired programming paradigm, popularly used for deep learning applications that require a computer to learn from observational data.

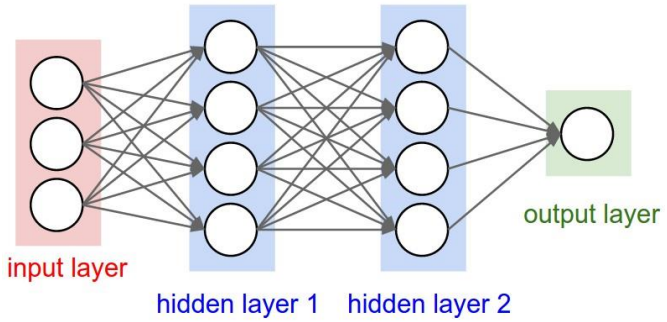

Figure 16 - A regular 3-layer Artificial Neural Network [32]

Their application to image classification underwent a major breakthrough in 2012 when Krizhevsky, *et al*. [33] applied a Convolutional Neural Network (CNN) to accomplish state-of-the-art performance in the 2012 ImageNet Challenge, achieving a winning top-5 test error rate of 15.3%, compared to 26.2% accomplished by the second-best entry.

CNNs are inspired by Hubel and Wiesel's [34] explanation of how mammals visually perceive the world around them using a layered architecture of neurons in the brain. Unlike a regular Neural Network, the layers of a CNN are arranged in 3 dimensions: **width** and **height** which are generally proportional to the pixel dimensions of input images, and **bands** which relates to the number of spectral bands. They take their name from the **convolution layers** they apply that slide or convolve a filter of fixed size over the data. This is an effective technique when the feature order/arrangement of the CNN's input layer is important, as you would expect for pixels in an image. In early layers this allows the CNN to identify basic visual features such as edges or groups of colours. In subsequent layers increasingly complex features such as circles, or eventually faces can be detected.

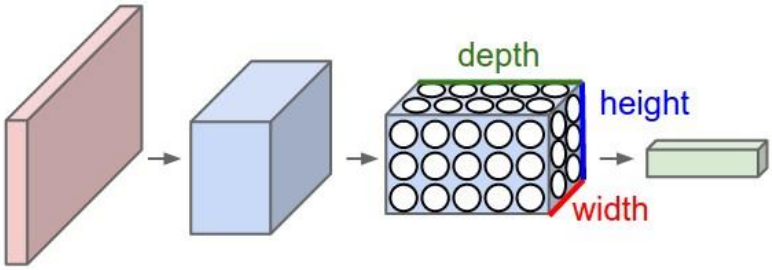

Figure 17 - Structure of a Convolutional Neural Network [32]

After a **convolution layer** it is typical to apply a **Rectified Linear Unit activation layer**, this was found to accelerate performance improvements during training of CNNs compared to other activation layer functions [33].

Another important layer of a CNN is the **pooling layer** that down-samples the spatial size of the representation to reduce the number of parameters and amount of computation in the network. These reductions also control overfitting [32].

4.2.2 Improving Neural Network performance with residual learning

In the field of Computer Vision both current research [35] [36] and the human-level surpassing leading results [37] of the annual ImageNet dataset competition have shown that when training CNNs a principal factor for image classification performance is the number of hidden layers and nodes, known as network depth. However, the more hidden layers of nodes a neural network has, the more difficult it is to train. This is due to the degradation problem whereby accuracy improvements diminish and then degrade rapidly as network depth increases. This issue was tackled in Kaiming, *et al*. [38] using a method called deep residual learning. Their Residual Neural Network (ResNet) took 1st place in 5 of the largest Computer Vision competitions in 2015.

Residual learning eases the training of CNNs and enables their architectures to be substantially deeper by building 'residual blocks' that employ identity mappings as shown in Figure 18.

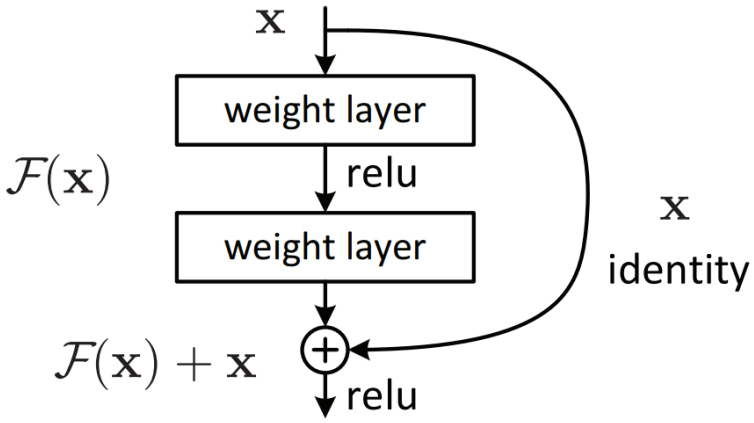

Figure 18 - A residual block [38]

5 Project Specification

This section outlines the key functional requirements, de-scoped functionality, and key dependencies.

5.1 Key functional requirements

5.1.1 UI and interaction requirements

- User can view, pan, and zoom graphical map of the world and obtain latitude/longitude coordinates of AOI
- User can request low-resolution imagery retrieval and analysis within radius of specified latitude/longitude, captured during specific time periods
- User can view, pan, and zoom imagery and analysis layers of map
- User can request high-resolution imagery retrieval and analysis of AOI at specific latitude/longitude

5.1.2 Imagery requirements

- Low-resolution imagery returned to user is mosaic of minimal cloud coverage during period requested
- High-resolution imagery returned to user is that with lowest cloud coverage available over AOI, prioritised by capture date
- Imagery is shown in true colour using Red Green and Blue spectral bands from satellite data
- Imagery returned is layered on graphical map at correct geolocations
- Imagery returned is suitable for application of algorithms

5.2 De-scoped functionality

- UI slider functionality for first/second timeframe imagery mosaic layers
- Specific focus on developments in the South China Sea
- Fine-tuning the CNNs

Section 11.3 provides additional detail on the de-scoped functionality and also a sample of suggested future improvements for the project tool.

5.3 Key dependencies

Table 3 - Key project dependencies and impacts

Dependency	Impacts on
Availability of labelled satellite imagery training data.	Feasibility and accuracy of algorithms.
Sufficient GPU computing capacity.	Ability to train algorithms within limited timeframe.
Active connection to satellite imagery servers and access permissions.	Ability to retrieve data.
Low cloud satellite imagery of AOI.	Effectiveness of tool and algorithms.
Sufficient CPU computing capacity.	Ability to run tool and apply algorithms locally.
Sufficiently reliable/stable software libraries available.	Ability to build a functional tool within limited timeframe.

6 System Design and Architecture

This section provides a high-level description of the architecture of the project tool and the map rendering and UI design.

6.1 Use case description and activity diagram

The core use case is described with a use case description (Table 4) and an activity diagram (Figure 19). The activity diagram assumes that the user provides suitable parameters and satellite imagery is available, the tool has additional exception handling for when this is not the case.

Table 4 - Use Case Description for Project Tool

Use Case Name:	Analyse Satellite Imagery
Description:	A tool which allows a user to retrieve and analyse satellite imagery
Actor:	User
Triggers:	User loads and runs Jupyter Notebook file to display graphical map and UI
Precondition:	User wishes to retrieve and analyse satellite imagery in a specific AOI and perform analysis to compare changes over time and detect objects
Postcondition:	User is able to view satellite imagery and analysis as layers displayed in an interactive map
Main Course (M):	1. User interacts with graphical map and uses mouse click to identify latitude and longitude of AOI 2. User interacts with UI to set parameters: **Latitude coordinate**, **Longitude coordinate**, **Radius of interest**, **First period start date** and **length of this period**, **Second period start date** and **length of this period**. 3. User initiates low-resolution satellite imagery retrieval and analysis by clicking 'Run Stage One' button. 4. The inputs are processed and sent as a Web Coverage Service (WCS) request (7.4.1.1.1) to the Sentinel Hub servers 5. The imagery retrieved is prioritised based on cloud coverage and analysed. 6. The imagery and analysis are displayed on an interactive map and navigated by the user 7. The user may proceed to use mouse clicks to identify latitude and longitude of second AOI 8. User interacts with UI to set parameters for second stage: **Latitude coordinate** and **Longitude coordinate.** 9. User initiates high-resolution satellite imagery retrieval and analysis by clicking 'Run Stage Two' button 10. The inputs are processed and the DigitalGlobe catalogue is searched for candidate image. Metadata is retrieved to prioritise imagery based on cloud cover and capture date 11. For each candidate image a WCS request to the DigitalGlobe servers is attempted in prioritised order based on cloud cover and capture date until an image is successfully retrieved

	12. Analysis is performed on the image retrieved. 13. The imagery and analysis are displayed on an interactive map and navigated by the user
Alternate Course 1 (A1):	1. The user proceeds directly to the second stage of the tool
Alternate Course 2 (A2):	1. The user clicks ‘Run Stage One’ or ‘Run Stage Two’ without setting parameters 2. The tool is run using the default parameters

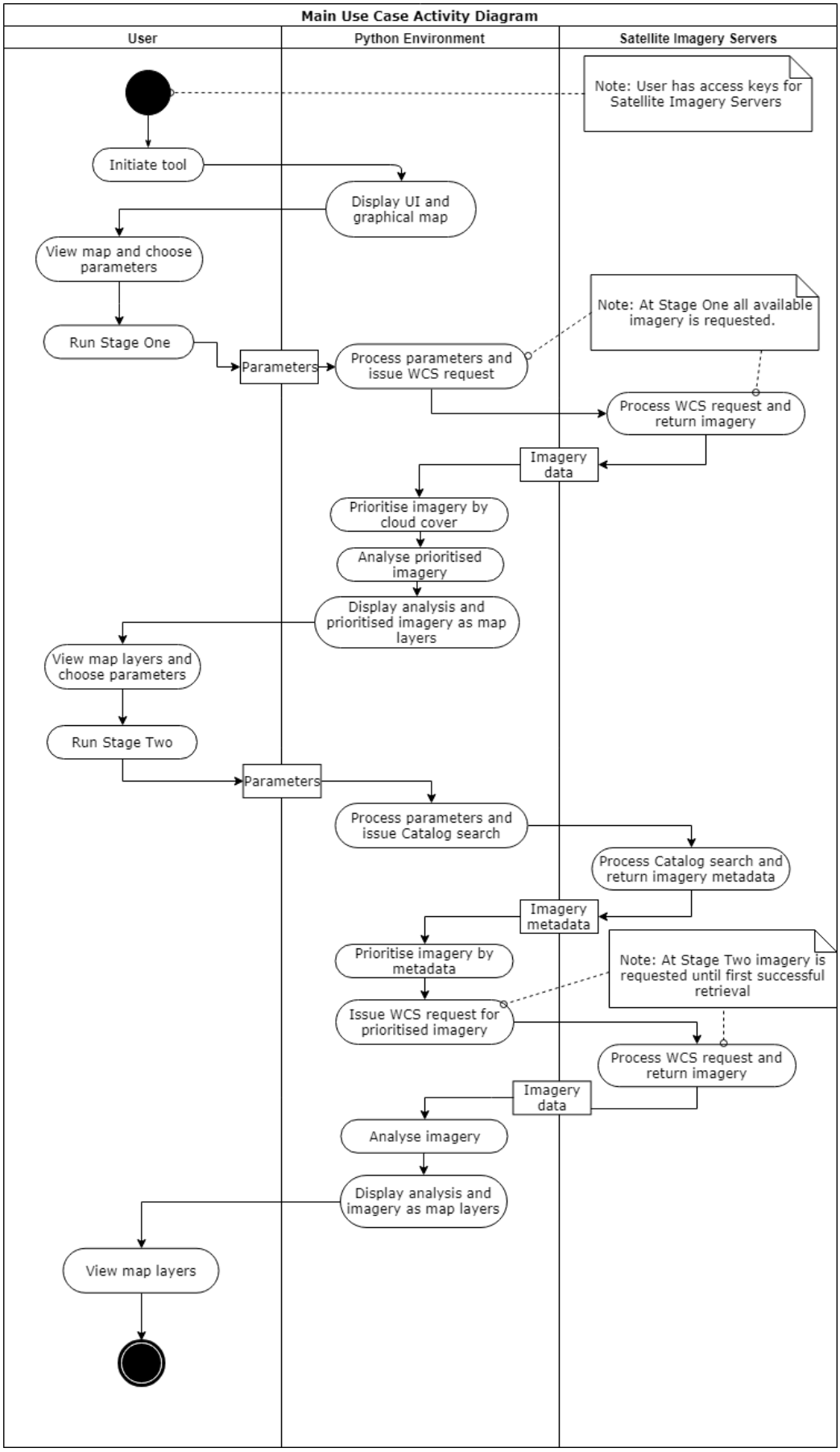

Figure 19 - Activity Diagram showing Main Use Case for Project Tool

6.2 System design and architecture

6.2.1 System description

The tool is designed to be sufficiently light-weight to run locally on the average personal computer for smaller AOIs. However, searches over larger areas or for imagery captured over longer periods of time require higher specifications. As described in Table 4 the client computer retrieves and holds only the required imagery from both the Sentinel Hub and DigitalGlobe databases.

6.2.1.1 Part 1 – The Programming language: Python front-end and back-end

From a software architecture perspective all code used by the tool was written in Python [39], a highly versatile object-oriented programming language that is portable enough to run on many variants of UNIX, Mac, and Windows.

In the project proposal the advantages and disadvantages of MATLAB and Python were compared, with both technologies proving viable for this project. Following further research I found that the tools available for accessing and manipulating satellite imagery data were principally written in Python. Consequentially this project focusses on a solution using this language, however the libraries which are used also access other languages such as HTML and JavaScript.

Python methods and packages provide both the front-end and back-end of the tool: displaying the UI, handling user inputs, sending requests to satellite imagery servers, managing the data which is returned, performing the analysis, and presenting this back to the user.

The open source package/environment management system Conda [40] was used to manage the complex inter-dependencies of the various Python libraries required for the tool. A comprehensive list of the libraries used for the project is provided in section 67.

6.2.1.2 Part 2 – The Tool Environment: A Jupyter Notebook server-client structure

The tool is hosted and run in a Jupyter Notebook browser-based environment [41]. Jupyter Notebook is an open-source web application based on a server-client structure. It is built for developing, documenting, and executing code in Python using an IPython kernel by default.

6.2.1.3 Part 3 - The UI: Folium Library mapping and ipywidgets Library user interaction

The ipywidgets [42] library is specifically designed for Jupyter notebooks and provides access to the IPython kernel's interactive HTML widgets. The tool utilises these widgets to receive, and also restrict, user inputs from within a UI.

The Folium [43] library acts as a Python wrapper for a JavaScript tool called Leaflet [44]. The tool uses Folium to generate several interactive Leaflet maps based on the user input received via the UI widgets.

6.2.1.4 Part 4 – The satellite imagery servers: Sentinel Hub and DigitalGlobe

External imagery servers are accessed by the tool to retrieve satellite imagery data. Low-resolution Sentinel-2 imagery access is provided by Sentinel Hub and downloaded via HTTP from the Copernicus Open Access Hub [45]. High-resolution Worldview-3 imagery is provided by DigitalGlobe, who use AWS for cloud-based storage of all of their content which is accessed through RESTful web service APIs.

6.2.2 Map rendering design

The map is designed from an end-user perspective so as to be as user-friendly as possible. The basemap design uses OpenStreetMap tiles, these were chosen due to the balance between the speed at which they render and the detail they provide.

6.2.2.1 Zoom level

When a map is rendered the zoom level default is based on the resolution of the satellite imagery being returned. For low-resolution imagery this is set at zoom level 14, and for high-resolution imagery this is at zoom level 17. The maximum possible zoom level is also increased from 18 to 21 when high-resolution imagery is displayed to allow closer inspection by the user.

Where imagery is not being displayed, for instance when the tool is initiated, a lower default zoom level is used to allow the user to navigate to their AOI with as few mouse clicks as possible.

6.2.2.2 Latitude and longitude coordinates

When a map is rendered with retrieved satellite imagery the latitude and longitude coordinate defaults for the centre of the map are set to those provided by the user. When the tool is initiated this is set in combination with the zoom level as above to provide a view of the South China Sea (Figure 1).

6.2.2.3 User Interaction

As demonstrated in section 3.3 there are several elements of user interaction included in the map design. The user can navigate the map by dragging it and using the provided buttons to zoom in and out. A plug-in was implemented so that it is possible to extend the map to full-screen mode. Each of the layers, other than the basemap layer, can be displayed or hidden by the user in the layer control widget panel.

6.2.2.4 Map Layers

Every map rendered by the tool includes the basemap layer, which as stated is OpenStreetMap. The key purpose of the basemap layer is to enable the user to orientate themselves and locate their AOI, it is detailed with world land polygons, global ocean polygons, country labels, and at lower zoom levels city, building, and road labels too.

The remainder of the layers provide specialised data based on the user input. These are known as feature layers and are displayed in the order shown in Figure 5. A brief description of each is provided below:

- **Low-resolution satellite imagery layers**

For each timeframe provided by the user a low cloud imagery mosaic is constructed using the visible spectral bands. Each image in a mosaic is grouped together for display as a single layer in the correct geolocation. The brightness of these layers is scaled up by a factor of 3 for visualisation.

- **Low-resolution combined cloud mask layer**

To produce each low-resolution imagery mosaic a cloud mask is calculated, detailed in implementation section 7.4.1.4. The cloud cover is of interest to the user and so the

cloud mask from each timeframe is combined and displayed as a single feature layer, as illustrated in Figure 20.

Figure 20 - Imagery mosaics from two timeframes over the same AOI and resultant combined cloud mask

- **Low-resolution land cover classification layers**

A classification algorithm, detailed in implementation section 7.4.4 is applied to the pixels in each low-resolution imagery mosaic to classify the land cover. This is displayed as a separate feature layer for each timeframe.

- **Low-resolution land cover change analysis layer**

The land cover classification of each timeframe is compared on pixels where the combined cloud mask does not detect cloud coverage. This is displayed as a square polygon with outline and fill opacity levels set based on the level of change detected as illustrated in Figure 21. Areas with high change detection have more clearly marked polygons. The polygons are combined in to a single feature layer.

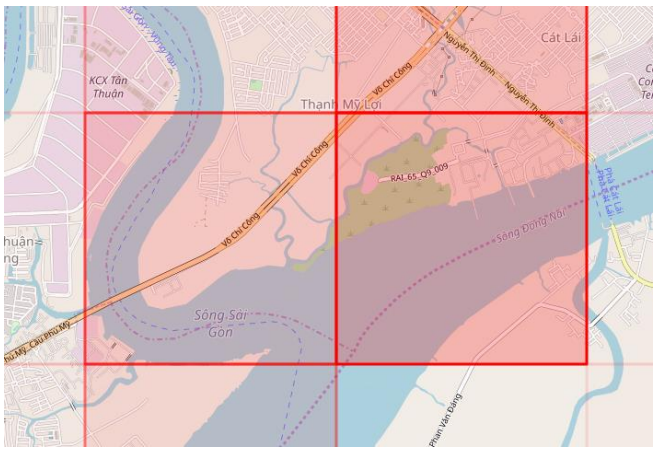

Figure 21 - Example of low-resolution land cover analysis layer

- **Low-resolution CNN change analysis**

The low-resolution CNN is applied to each image in the low cloud imagery mosaic, and the results of which are compared between timeframes. As with the low-resolution land cover change analysis layer, these results are displayed as a combined feature layer of square polygons with outline and fill opacity levels set based on the level of change detected between timeframes.

- **High-resolution satellite imagery layer**

For performance reasons only a single high-resolution image is returned by the tool for each AOI. The imagery is pansharpened and displayed in the correct geolocation.

- **High-resolution Object Detection CNN**

The high-resolution Object Detection CNN is applied and where the prediction confidence of a detected object is greater than the user's pre-specified threshold the tool draws a bounding box around the object on the image and provides the class name of the object identified within the bounding box. The tool displays the image as a feature layer in the map interface for the user to interactively explore.

6.2.3 UI design

The UI design, shown in Figure 22, is based on the core principles of clarity, flexibility, and efficiency.

6.2.3.1 Clarity

The widget labelling and layout is deigned to be as self-explanatory and intuitive as possible. Only the necessary controls are provided for the core tool functionality and a user who is unfamiliar with the software would be able to quickly get to grips with the UI. Default values are set for each widget to indicate sensible inputs. When the user runs the tool it can take several minutes for imagery to be returned and so during this time updates are provided at key points in the process, initially to confirm the user inputs and then on the status of imagery retrieval and analysis.

6.2.3.2 Flexibility

Despite the software being designed with some functional order in mind the tool can run Stage One and Stage Two in any combination, for instance to only request high-resolution imagery or to request low-resolution imagery for multiple AOIs. The user can either type their latitude and longitude coordinates in a freetext box for maximum precision, or use the slider functionality instead. All latitude and longitude combinations are accepted, including those near the North/South poles. Any period of whole weeks, up to a maximum of roughly two months, can be set as the length of each timeframe for low-resolution imagery retrieval to be attempted.

6.2.3.3 Efficiency

The limitations within the UI are set in combination with the underlying Python code to ensure an efficient user experience, and to prevent an inadvertent request for excessive quantities of satellite imagery data. The coordinate slider and freetext boxes are linked and changing one will update the other automatically. When running the tool all widget inputs are stored so that the user does not need to re-enter them between Stage One and Stage Two.

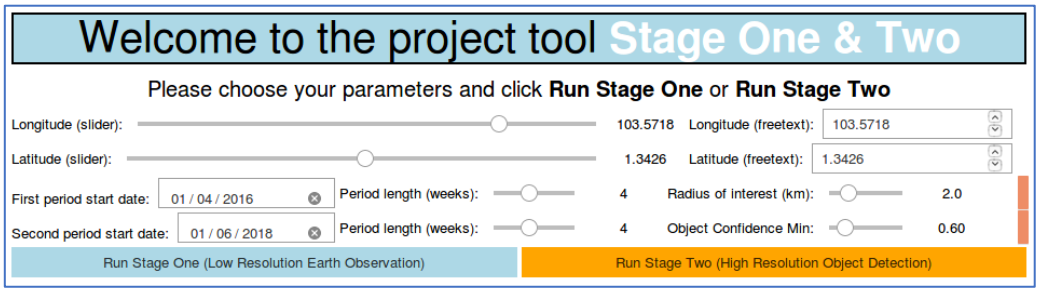

Figure 22 - Close-up of UI

7 Implementation

This section details all aspects related to the implementation of the project tool. This includes the technologies, tools, programming languages, and servers/databases used.

7.1 Programming languages and libraries

As set out in 6.2.1.1, Python is the primary programming language I have used for the tool.

7.2 Databases and servers

Due to the volume of satellite imagery data available and the accelerating velocity at which is continues to be produced it was decided that local data storage would only be suitable for the imagery pertinent to the area of interest at the time of use. The retrieval time for data from external satellite imagery providers was tested and designs put in place to limit the waiting period required when accessing these servers in the tool.

7.3 Map rendering and UI

As set out in 6.2.1.3, a lightweight map rendering solution has been applied which uses the Folium [43] Python library. The ipywidgets [42] Python library is used to manage user interactivity and the tool is hosted in a Jupyter Notebook. Detail of the map rendering and UI implementation is in section 7.4.8.

7.4 Implementation details

To manage the inherent complexity of this multi-stage project I split the implementation in to sub-components. Each sub-component refers to a coding or research element of the implementation that is modular and incremental. To reduce the risk that excessive time spent on one aspect of the project would impact on a later stage a time-limit was assigned to each sub-component which was strictly adhered to. These are ordered to convey all aspects of the implementation to the reader in a meaningful and coherent way. However, in reality several of the components were performed in parallel at times, or revisited following a finding or issue at a later stage.

7.4.1 Acquiring Sentinel-2 satellite imagery

As part of the background research I identified two main satellite missions for which visible-spectrum open data was available, NASA's Landsat 8 and ESA's Sentinel-2. Although both were viable sources for satellite imagery, the Sentinel-2 satellites have a slightly higher resolution in certain spectral bands, a more frequent revisit time, and are due to stay in active use for longer. Also, as mentioned in section 4.1.2.3, it was found in testing that Landsat 8 imagery was not available over the regions that this tool would be primarily focussed on. Having decided to focus on Sentinel-2 the objective of this sub-component was to build a set of Python modules capable of retrieving suitable format Sentinel-2 imagery in specific areas captured during specific periods of time from a web service database.

7.4.1.1 The sentinelhub-py library

The core library used for this aspect of the tool was sentinelhub-py [46], which is written and maintained by 'Sentinel Hub by Sinergise' [47]. My research and testing showed that despite there being multiple libraries offering a similar functionality, a large number of these were not as reliable with either limited functionality or multiple usability issues/bugs. The sentinelhub-py library was the tool of choice, despite being a paid

service, after a successful research application to the European Space Agency's Open Science Earth Observation call: 'The Open Science Earth Observation (OSEO) call offers to scientists the opportunity to exploit at no cost a full archive of EO data for science, applications and technological innovation, by offering Third Party services which exploit state of the art ICT.' [48]

7.4.1.1.1 Open Geospatial Consortium Web Map Service and Web Coverage Service requests

The sentinelhub-py package allows users to make Open Geospatial Consortium (OGC) Web Map Service (WMS) [49] and Web Coverage Service (WCS) [50] requests to their map server. WMS requests serve georeferenced map images, whilst WCS requests serve coverage data. Both types of requests were found to be useful with the WCS request service allowing spatial resolution to be specified; an important factor for the performance of Computer Vision algorithms.

The sentinelhub-py library also enables users to download raw data from AWS in '.SAFE' format [51]. To fully utilise the open source nature of Sentinel-2 imagery data my preference would have been to use the library to retrieve imagery from AWS. Unfortunately during the time that this tool was being developed the access rights to the relevant Sentinel-2 AWS bucket were in the process of changing [52], this would have caused the tool to break. For this reason the tool has been built utilising the OGC WCS request service methods of sentinelhub-py.

7.4.1.2 Sentinel Hub set-up and OGC WCS imagery specification

In order to use the sentinelhub-py library to access Sentinel Hub services a Sentinel Hub account was required, once set up the next stage was to configure an instance ID (alpha-numeric code of length 36) with access to a layer containing all of Sentinel-2's Level-1C data bands using the Sentinel Hub Configurator [53]. Section 4.1.1.2 gives detail on why Level-1C was most suitable processing level for the tool.

The key arguments that the tool passes within the WCS request method are:

- **layer** – The 'BANDS-S2-L1C' layer is specified to access the Sentinel-2 imagery source with all 13 available bands (B01,B02,B03,B04,B05,B06,B07,B08,B8A,B09,B10,B11,B12)
- **resx, resy** – For both x (column) and y (row) resolutions '10m' is requested by the tool, this is the best possible native resolution of some of the bands in Sentinel-2 imagery
- **time** – Either a single date or a range of dates in ISO8601 format can be passed in this argument. In the tool I take a range of dates based on user input.
- **instance_id** – As above, an instance ID was configured to access the required Sentinel-2 data and passed in this argument.
- **custom_url_params** – An issue I had faced using this method was that a 'Sentinel Hub' watermark logo appeared in the bottom left corner of each method returned, significantly impacting on algorithm performance. By passing '{CustomUrlParam.SHOWLOGO: False}' in this argument this logo was removed from the images returned.
- **bbox** – This argument specifies the AOI that the imagery is being requested for within bounding box coordinates. It takes an instance of the sentinelhub-py BBox method which is in World Geodetic System 1984 (WGS84, EPSG:3857)

Coordinate Reference System (CRS) format. The complex approach I've taken to calculate accurate bounding box coordinates based on the user's inputs is discussed in section 7.4.1.3.

The appendix section 11.7.3 provides the Python method that issues these sentinelhub-py WCS requests.

7.4.1.3 Calculating distance-based bounding box coordinates

One of the core functions of this tool is to return imagery suitable for Computer Vision algorithms based on a user's input. These inputs are a point on the map in longitude and latitude WGS84 CRS degrees, and also the distance from this point in km that is to be retrieved and analysed.

To input imagery into the CNN detailed in 7.4.3 it must be of shape 256 by 256 pixels. As the imagery returned by the sentinelhub-py WCS request method is at a resolution of $10m^2$ per pixel this translates to square imagery with precisely $2.56km^2$ surface coverage. The tool scales up the user-provided distance so that the imagery returned can be split in to a square grid containing a whole number of $2.56km^2$ image tiles.

To calculate the bounding box coordinates of this grid the PyGeoTools [54] library is used. This library is a Python adaptation of the java code written by Jan Philip Matuschek [55] which addresses the challenge illustrated in Figure 23 of calculating distances in metres and representing them in WGS84 degrees.

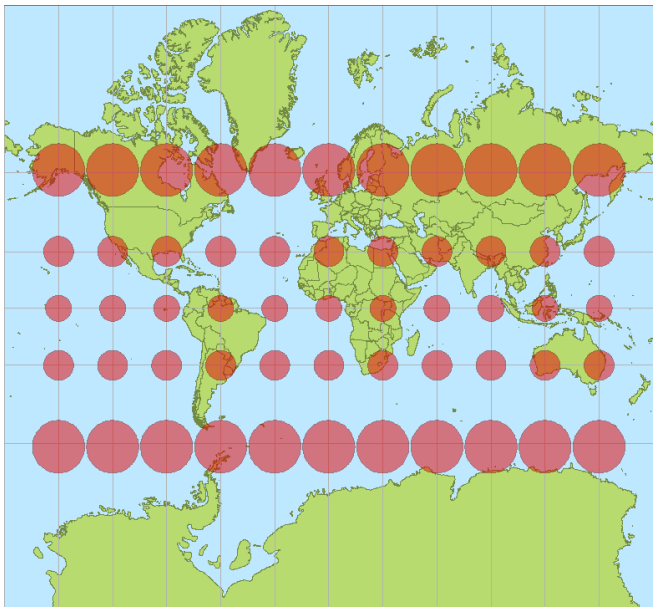

Figure 23 - Variation with latitude of represented distances (in degrees or pixels) on the Mercator projection per actual distances (in meters) on Earth surface. [56]

The PyGeoTools methods were applied to find the minimum and maximum longitude and latitude coordinates of points for which the great-circle distance [57] from the user-provided location on the map is equal to the scaled-up distance calculated in the previous step. These coordinates are used to create the bounding box for the total area to be retrieved. Figure 24 provides a graphical illustration of these calculations and resultant grid.

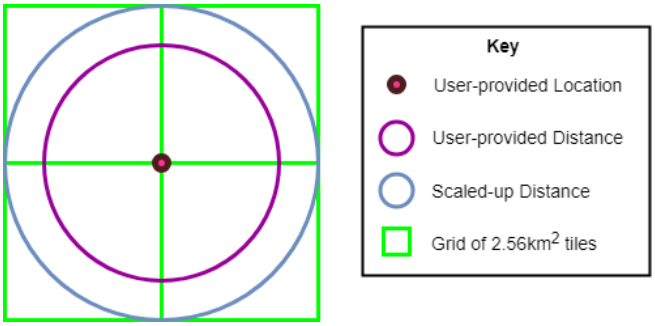

Figure 24 - Graphical illustration of user-inputs and resultant tile grid

The formula used in this algorithm assumes that the Earth is a perfect sphere, however due to the planet's true ellipsoidal shape there is the potential for a very slight error (<0.5%) in the calculation which is maximised at median latitudes. To account for this the images are reshaped where they do not perfectly meet the 256 by 256 pixel specification.

The tool retrieves each grid tile image individually using the sentinelhub-py library's BBoxSplitter method, rather than requesting imagery for the entire area and splitting it locally once retrieved. This approach was taken to reduce the performance impact of large AOI downloads, and to prevent large areas of imagery being returned with no data; a scenario which occurs when a Sentinel-2 image swath only partially intersects the bounding box.

7.4.1.4 Cloud cover algorithm

Due to the high revisit rate of Sentinel-2 satellites it is likely that multiple images will be returned for a given AOI and timeframe. Where this occurs the imagery for each sub-bounding box is prioritised and retained by the tool based on the level of cloud cover present, presenting to the user a mosaic of the lowest possible cloud cover images for the given AOI and timeframe.

The WCS request method accepts a **maxcc** argument which is used to specify the maximum proportion of cloud coverage for returned imagery. In testing this was found to be unsuitable as the **maxcc** cloud coverage is estimated on the entire Sentinel-2 tile and not just for the region defined by the bounding box. I implemented a more effective approach by adapting the algorithm proposed in J. Braaten, *et al*. [58]. The spectral bands used in the original algorithm are based on those captured by the Landsat sensors. However, as the Sentinel-2 instruments were cross-calibrated in their development with Landsat 8 sensors the equivalent bands were found to have satisfactory results in testing. The algorithm is applied to each image by the 'image_cloudMask_cloudScore_date' method, shown in appendix section 11.7.3, to create a cloud mask and prioritise imagery. Figure 25 gives an example of an image with high cloud cover and the cloud mask produced as a result.

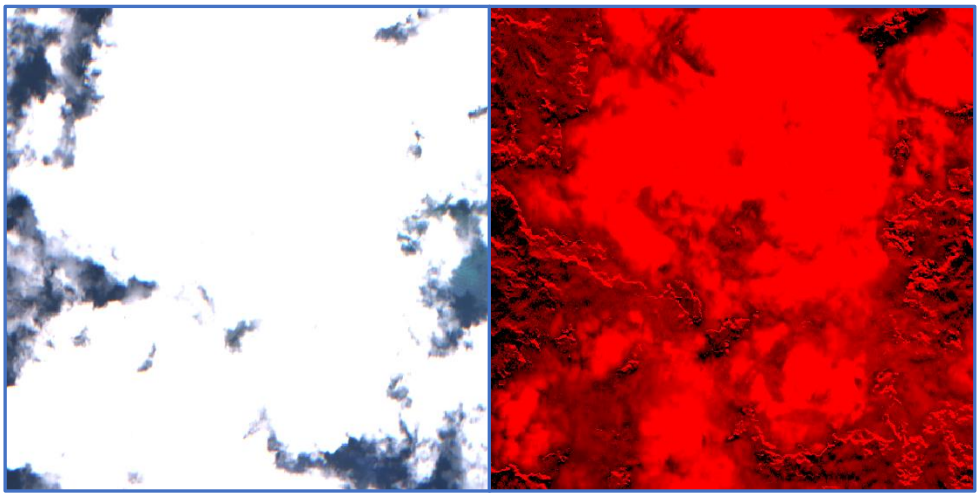

Figure 25 - Example of Sentinel-2 image with high cloud cover and resulting cloud mask

7.4.2 Creating a satellite imagery training dataset

As stated in 5.1 a key functional requirement for the tool was to use a CNN to identify change of interest in satellite imagery. There is no pre-trained Neural Network available for this specific task, and also limited labelled training data available at Sentinel-2 resolution. That which is available is unsuitable, for instance from locations on the planet (e.g. cities in Europe [59]) which are not representative of the geographic location of primary focus for the tool.

Using the label-maker tool from Development Seed [60] I was able to build my own training data set. This was an extensive process and full implementation details are provided in the annex section 11.5.

7.4.3 Leveraging cloud computing power to train a deep residual CNN

7.4.3.1 ResNet50 deep residual architecture

As mentioned in 4.2.2, several of the best performing entries in recent satellite imagery land cover classification competitions [61] [62] used a ResNet50 [38] CNN, and so for this tool the same architecture has been applied. An interactive visualisation of the Resnet50 architecture can be accessed on GitHub [63].

7.4.3.2 AWS cloud computing technology

Training a CNN with as many layers as ResNet50 is a computationally-intensive task to undertake. Locally available hardware was not powerful enough for the purpose and so alternative methods were investigated.

Using cloud compute power was the most cost-effective approach identified, with AWS offering on-demand Elastic Cloud Compute (EC2) instances specifically engineered for 'Machine Learning Use Cases' [64].

The EC2 instance used for training was the p2.xlarge [65] which provides an NVIDIA K80 GPU with over 60 GiB of RAM and an environment preinstalled with Keras, Tensorflow and other libraries required for GPU-powered Neural Network training. I held the labelled data for training within an AWS S3 bucket to reduce upload/transfer times and hence costs once the instance was initiated. To control these services from a local machine I set up a configuration file and used the AWS Command Line Interface tool

[66]. Section 11.6.2 provides further details, such as model accuracy improvements during training.

7.4.4 Utilising Google Earth Engine for land cover classification

Despite using a deep residual architecture, the size of the manually labelled training data set, used as a result of 11.5, limited the accuracy of the CNN and the algorithm was not performing to the required standard to use as the only indicator of change-of-interest for the tool.

Despite issues set out in 4.1.2.3 with GEE I was able to devise a method to train and then transfer a model utilising the GEE Classification library's CART algorithm [67]. This algorithm was trained using pixels I had hand-selected from a composite of cloud-free Sentinel-2 imagery. The three classes I created for this algorithm were 'water', 'vegetation_natural' and 'urban_manmade', and in total over 450 points from countries and regions near the South China Sea were selected as shown in Figure 26.

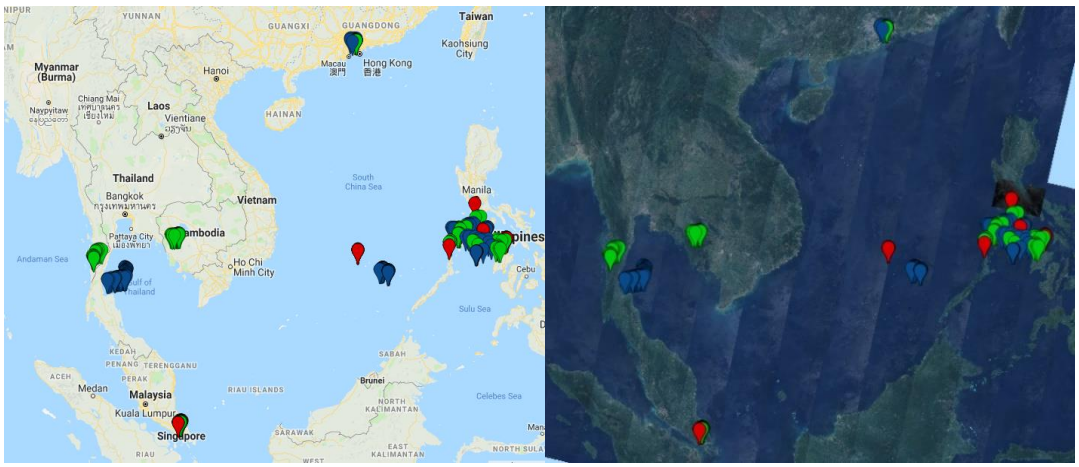

Figure 26 - CART training points selected within GEE. Left – graphical map. Right – Sentinel-2 composite

In contrast to the First Stage CNN, for which the training data available was imagery in the visible RGB spectral bands, this algorithm was trained on the full range of spectral bands captured by the sentinel-2 MSI (Table 2, page 18). The data was split 70/30 for training/testing and Table 5 shows the confusion matrix created when validating the CART on the test data following training. The algorithm applied as scale over an area in Brunei is shown in Figure 27. The JavaScript code I used to produce the CART can be found in section 11.7.1 and I've shared it publicly on GEE [68].

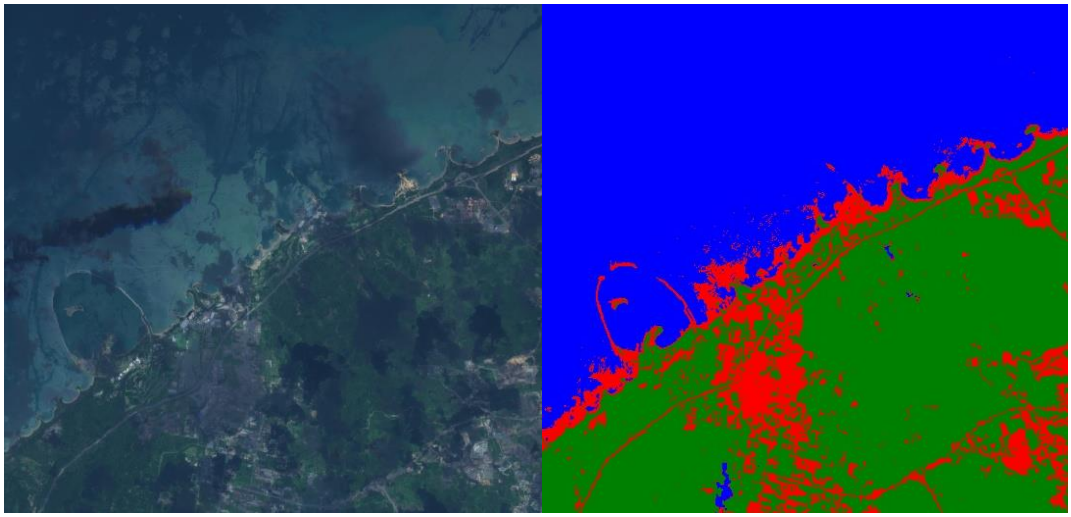

Figure 27 - Side-by-side comparison of Brunei Sentinel-2 composite and CART land cover predictions within GEE

Table 5 – Confusion matrix showing performance of CART on test data

		Predicted Class		
		urban_manmade	vegetation_natural	water
Actual class	urban_manmade	53	*1*	*1*
	vegetation_natural	0	49	0
	water	0	0	47

The implementation process involved taking the JavaScript output of the CART from GEE and building it manually within the project tool, replicating the algorithm by using if/elsif/else logical statements in Python. Within GEE the Sentinel-2 data is in UINT16 format with each spectral band representing TOA reflectance scaled by 10,000, whereas Sentinel Hub returns imagery at true TOA reflectance in the range 0-1. A factor was applied to leaf node values to account for this.

When testing the algorithm on Sentinel-2 data within the tool it was found that the values used to split at each CART leaf node required further scaling. This is in part due to the min() function being used in GEE to produce the cloud-free Sentinel-2 mosaic that the CART was trained on, which unfortunately results in the majority of pixels in the training data being taken from cloud shadows instead.

The process for this additional scaling was iterative, with a range of values proportional to the average pixel value of the target image being applied to the CART leaf node split values, and the resulting land cover classification outputs being visually compared for accuracy. Following this comparison the two images showing the most accurate classification were selected and their scaling factors were taken as upper and lower bounds for the next range of factors to be tested. This was repeated until no improvement was observed in additional iterations. An illustrative example with a code sample is provided in 8.1.1.

Finally, higher cloud coverage in imagery causes the average pixel values to be disproportionately high. To account for the detrimental impact of this on the land cover

classification accuracy an additional scaling factor is also applied to the leaf node values which is based on the cloud mask created in section 7.4.1.4. If the images within a mosaic were selected from within a sufficiently wide timeframe then cloud coverage tends to be low and this factor is close to 1.

7.4.5 Analysing change between timeframes

At this stage a low-cloud imagery mosaic for each timeframe was available, suitable to be used as input for both the CNN and CART algorithm to produce comparison analysis. The tool applies the CNN to each mosaic tile and compares the output predictions for the same geolocation from each timeframe. If there is significant change in the two outputs this is considered to be a good indication of change of interest and can be highlighted to the user.

The CART algorithm is applied and land cover classification is compared only in pixels where no cloud coverage is identified by the cloud mask algorithm in either timeframe's imagery tile. The mosaic imagery tile with highest proportional change is highlighted to the user, as illustrated in sections 3.3.2 and 6.2.2.4.

7.4.6 Retrieving high-resolution imagery from DigitalGlobe

In Stage Two this tool requests and analyses high-resolution satellite imagery. DigitalGlobe (4.1.2.2) was the commercial imagery provider that was chosen over Planet Labs (4.1.2.1) for this stage of the tool for several reasons:

Firstly, the imagery collected by DigitalGlobe's WorldView-3 and Worldview-4 satellites has the highest possible spectral resolution publicly available at 0.31cm/pixel.

Secondly, DigitalGlobe provide a well-maintaned Python Software Developer's Kit called gbdxtools which can be used for ordering imagery and launching workflows on DigitalGlobe's GBDX [69].

Thirdly, in March 2018 the Defense Innovation Unit Experimental (DIUx) and the National Geospatial-Intelligence Agency released a new labelled satellite imagery dataset, xView, as part of the xView Challenge [70]. The imagery was collected from DigitalGlobe's WorldView-3 Satellites and is annotated with over one million bounding boxes across 60 object classes. The dataset was accompanied by benchmark CNN models using Single-Shot Multibox Detector (SSD) [71] architectures that had been pre-trained on the data [72].

7.4.6.1 Imagery access

After submitting a research proposal to DigitalGlobe through contacts made at the Satellite Applications Catapult they have agreed to provide me with access to their imagery for the purposes of this project. Whilst waiting for these permissions to be granted I have used their 'Evaluation Accounts' which provide full use of their services for up to 30 days.

7.4.6.2 Imagery requests

Due to the higher resolution of this imagery the data takes significantly longer to download and process than Sentinel-2 imagery for the same AOI. To manage the impact of this on the performance of the tool a different approach was required for this stage of the project. When a user requests imagery the gbdx.catalog.search method within the

gbdxtools library is used to find metadata on all imagery that has been captured over a given AOI, following this the CatalogImage method is used to retrieve an image.

7.4.6.2.1 Identifying suitable imagery

The conclusion of testing using the in-built Python 'time' module, shown in 8.1.2, was that for practical performance the AOI for high-resolution imagery needed to be between 1/3 and 1/4 the size of the 2.56km^2 sub-areas of low-resolution imagery.

DigitalGlobe use the same CRS format as Sentinel Hub (WGS84, EPSG:3857), and so the same coordinate inputs can be used for both stages of the tool. These inputs are converted to a Well-known text (WKT) format polygon as required by the gbdx.catalog.search method using the shapely.geometry library. The GBDX catalog is then searched with filters set to only return details of Worldview-3 Satellite imagery that intersects the WKT polygon.

The metadata of matching imagery is returned and the tool creates a list of particular properties of interest such as CatalogID (unique identifier used for requesting imagery), capture date, and cloud cover %. The cloud cover property is a suboptimal indicator of image quality as it is based on the entire parent image strip. Unfortunately, in contrast to the Sentinel-2 imagery, it is not feasible to download all image matches and apply a cloud mask locally due to the size of the data.

When GBDX acquire an imagery strip it's called an 'acquisition' and it sits in an archive until it is 'ordered' by a user. The image is then moved from the archive to a server location where it is regularly accessible as a 'product'. Although searching the catalog for candidate imagery returns all 'acquisitions', the CatalogImage request that retrieves the imagery in the next step only looks for 'products' that are immediately available. If a CatalogID is requested that is still only an 'acquisition' the gbdxtools method will return an error. To work with 'acquisition' images they first need to be ordered, which requires premium account level permissions and also takes significant time. This is out of scope for the purposes of this tool.

The gbdx.catalog.search list returned is sorted by descending acquisition date, and then sorted again by ascending cloud cover %. The tool then makes a CatalogImage request for each image in sorted order, with exception handling for the errors returned when resuesting 'acquisitions', until a 'product' image is successfully retrieved. The reason for the two-stage sort is so that if several images have the same cloud cover % a retrieval is attempted for the more recently captured images first. Before sorting the images are grouped by 10% of cloud cover. This grouping is to ensure, for example, an image captured in 2016 with 6% cloud cover does not take priority over an image captured in 2018 with 7% cloud cover.

7.4.6.2.2 Retrieving imagery

As stated, the tool requests imagery using the gbdxtools CatalogImage method. This takes as arguments the CatalogID identified in the previous step, a bounding box to crop the parent strip to the desired AOI, and a flag for whether to return pansharpened 0.31cm/pixel bands (the resolution of the xView dataset that the benchmark SSD CNN model was trained on).

The tool retrieves imagery and converts it into an 8-bit RGB tiff format that is suitable for both displaying as a layer close to true likeness on the graphical map, and also

feeding as input for object detection to the benchmark SSD CNN model. Several methods were tested when attempting to return imagery of this type. Figure 28 shows the bizarre outcome of DigitalGlobe's recommended approach: combining the gbdxtools image.astype() method, which converts the image from 32-bit default to 8-bit, and the image.geotiff() method, which saves the RGB bands (bands 4, 2, and 1) of the image as a tiff file.

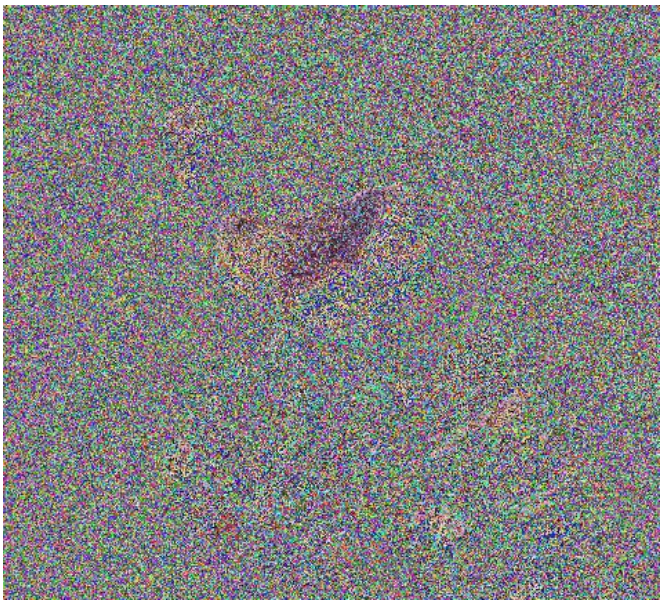

Figure 28 – Applying image.astype('uint8') and image.geotiff methods

The seemingly random colours in Figure 28 are the result of value overflows during the change in precision from the 32-bit default to 8-bit performed by the image.astype() method.

Figure 29 shows the result of another approach which avoids using the image.astype() method, instead passing *spec='rgb'* as an argument in the image.geotiff() method to return an 8-bit RGB tiff that's been contrast stretched to the 2nd and 98th percentile values of the parent strip. A different issue occurs here wherein the image appears to have overcompensated in the red band. This is caused by the contract being stretched based on values of the parent strip rather than the pixels within the returned image.

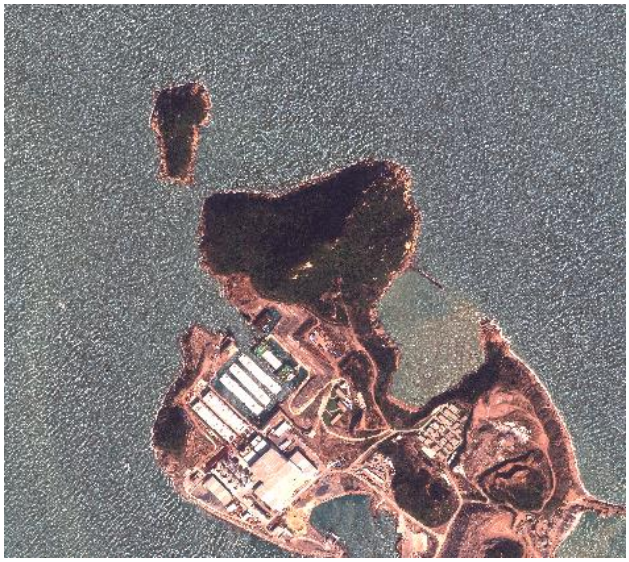

Figure 29 - Applying image.geotiff method with spec='rgb' argument

After further investigation a workaround was identified, which applies the gbdxtools image.rgb() method to create an 8-bit RGB array, and then uses the Python Pillow library to convert and save this image as a tiff for input to the benchmark SSD CNN model. This approach is suboptimal as the tiff that is generated does not contain any of the original GeoTiff geospatial information, however this information can be stored separately and as shown in Figure 30 the image that results from this workaround method is suitable for displaying as a true-colour layer in the tool.

Figure 30 - Applying image.rgb() method and converting with Pillow library

7.4.7 Applying Object Detection CNN to high-resolution imagery

Once an image is acquired at a suitable spatial resolution and in the correct format the tool loads and segments it in to 300*300 pixel chips and runs the benchmark SSD CNN on each chip, producing a maximum of 250 object detection predictions. Where the prediction confidence of a detected object is greater than a threshold pre-specified in the UI by the user the tool draws a bounding box on the image and provides the class name of the object identified within the bounding box.

Once this process is complete the tool displays the image as a feature layer in the map interface for the user to interactively explore.

7.4.8 Building the UI

A simple but effective GUI wrapper was developed to demonstrate the tool using a combination of the Folium [43] and ipywidgets [42] Python libraries.

The ipywidgets library provides interactive HTML widgets specifically designed for Jupyter notebooks. The tool utilises these widgets to receive, and also restrict, user inputs. The tool then utilizes the Folium library to visualize data retrieved/created in Python on an interactive leaflet.js map.

In testing the longitude and latitude sliders were found not to have the required sensitivity to select degrees to the desired granularity of 4 decimal places, and so a freetext box is also provided to the right of each. The freetext boxes and sliders are linked so that changing one automatically updates the other. The tool uses the Folium LatLngPopup method to show the latitude and longitude coordinates of the position on the map where a user performs a mouse click. These coordinates can then be copied manually in to the freetext boxes provided by the user.

These and other features, such as the full-screen option and layer control, were shown in the project trailer in section 3.3 and described in more detail in the design sections 6.2.2 and 6.2.3.

8 Testing examples

The following section provides details of testing additional to that included in the earlier sections of the report.

8.1 Unit testing

Throughout the development of the project tool every Python method that was built required extensive testing to verify that behaviour was as intended. The code sample below is an example of the unit testing applied to confirm that the bounding boxes produced to request imagery from Sentinel Hub matched the specifications required for the project tool purposes.

```
"""
Import required libraries for testing
"""
from shapely.geometry import shape, Polygon, asPolygon
from sentinelhub import BBoxSplitter, CRS
import math
from geoloc.geolocation import GeoLocation

"""
Set example user-provided parameters, repeat with other parameters for additional testing
"""
km_per_tile = 2.56 # intended width and height of returned imagery
user_Lon_deg = -0.1869 # Example longitude coordinate
user_Lat_deg = 51.4937 # Example latitude coordinate
user_dist_km = 3.2 # Example of user-provided distance in KM

"""
Test A - do the functions output the correct grid size and upper-distance:
Precursor to 'bbox_split_grid' method outputs 3
Precursor to 'user_dist_km_upper' method outputs (2.56*3)/2
"""
bbox_split_xy = math.ceil(user_dist_km/(km_per_tile/2)) # using ceiling function to calculate
minimum number of bounding box splits based on user input
user_dist_km_upper = bbox_split_xy*(km_per_tile/2) # distance calculated to ensure user AOI wi
thin bounding box

# output should be 3, and (2.56*3)/2
if bbox_split_xy == 3 and user_dist_km_upper == (2.56*3)/2:
    print("Test A - PASS")
else:
    print("Test A - FAIL")

"""
Test B - is the output bounding box within required accuracy of desired width/height in KM:
Precursor to 'user_large_bbox' method outputs coordinates with distance close to (2.56*3)/2
"""
user_loc = GeoLocation.from_degrees(user_Lat_deg, user_Lon_deg) # Create 'Geolocation' object
user_SW_loc, user_NE_loc = user_loc.bounding_locations(user_dist_km_upper) # Produce bottom-le
ft and top-right coordinates for bounding box


western_midpoint_ratio = (user_loc.distance_to(GeoLocation.from_degrees(
    (user_SW_loc.deg_lat+user_NE_loc.deg_lat)/2,user_SW_loc.deg_lon))) / ((2.56*3)/2) # Create
western ratio for coordinate accuracy testing
northern_midpoint_ratio = (user_loc.distance_to(GeoLocation.from_degrees(
    (user_NE_loc.deg_lat),(user_NE_loc.deg_lon+user_SW_loc.deg_lon)/2))) / ((2.56*3)/2) # Crea
te northern ratio for coordinate accuracy testing

# Output should be very close to 1
if 0.999 <= western_midpoint_ratio <= 1.001 and 0.999 <= northern_midpoint_ratio <= 1.001:
    print("Test B - PASS")
else:
    print("Test B - FAIL")


"""
Test C - are the tiles within the larger bounding box close to desired size:
Precursor to 'user_sub_bbox_coords_v2' method outputs tiles with width and height close to 2.5
6km
```

```
"""
# With larger bbox geo coords calculated we now split to tiles of grid:
user_polyg = asPolygon([[user_SW_loc.deg_lon, user_SW_loc.deg_lat],[user_SW_loc.deg_lon, user_
NE_loc.deg_lat],
                        [user_NE_loc.deg_lon, user_NE_loc.deg_lat],[user_NE_loc.deg_lon, user_
SW_loc.deg_lat]])
# With polygon created as input to splitter method, feed in with previously calculated number
of sub_bboxes
user_bbox_splitter = BBoxSplitter([user_polyg], CRS.WGS84, (bbox_split_xy, bbox_split_xy))

user_sub_bbox_list = []
width_length_count = 0
# extract lon/lat from each bounding box and test distances
for box in range(len(user_bbox_splitter.get_bbox_list())):
    lon1, lat1 = user_bbox_splitter.get_bbox_list()[box].get_lower_left()
    lon2, lat2 = user_bbox_splitter.get_bbox_list()[box].get_upper_right()

    user_sub_bbox_list.append((lon1, lat1, lon2, lat2))
    height_bbox_km = (GeoLocation.from_degrees(lat1,lon1).distance_to(GeoLocation.from_degrees
(lat2,lon1)))
    length_bbox_km = (GeoLocation.from_degrees(lat1,lon1).distance_to(GeoLocation.from_degrees
(lat1,lon2)))

    if 0.999 <= height_bbox_km/2.56 <= 1.001 and 0.999 <= length_bbox_km/2.56 <= 1.001:
        width_length_count += 1 # increments for each tile if accuracy levels confirmed
    else:
        print(height_bbox_km/2.56,length_bbox_km/2.56)

# test whether all tiles are within desired accuracy
if width_length_count == len(user_bbox_splitter.get_bbox_list()):
    print("Test C - PASS")
else:
    print("Test C - FAIL")
```

Output:

Test A - PASS

Test B - PASS

Test C - PASS

8.1.1 Testing the GEE CART land cover algorithm

As described in 7.4.4 a factor was set when importing the CART algorithm from GEE in to Sentinel Hub to account for the different imagery brightness. Figure 31 illustrates an iteration of this testing alongside the original image, in which the second image gives the most accurate classification of land cover. The code used was an adaptation of the 'landcover_cart' method in section 11.7.3.

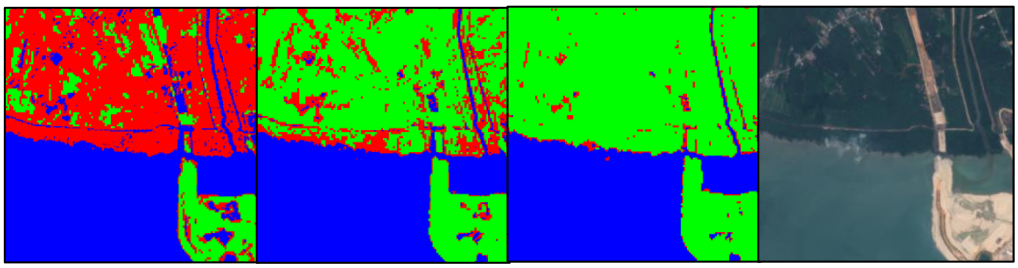

Figure 31 – Illustration of imported CART algorithm classification with varying factors applied, and original image

8.1.2 Retrieval time for RGB imagery from DigitalGlobe

The method that sends requests to the DigitalGlobe servers restricts the size of the AOI to limit the time for data to be retrieved and stored locally. Table 6 below illustrates the run time recorded for various AOI sizes.

Table 6 - Time taken to retrieve and store imagery based on factor applied to original AOI

method	factor applied to AOI	retrieval and storage time (seconds)
GBDX_IMG_Request	1	1428.2145
GBDX_IMG_Request	1/2	466.3325
GBDX_IMG_Request	1/3	191.0298
GBDX_IMG_Request	1/3.5	139.6207

Code example, in which a factor of 1/3.5 is applied:

```
test_factor = 3.5 # Factor to test
sub_size = 2.56/test_factor # Reduce AOI by factor being tested
sub_AOI = user_GBDX_bbox(103.57945528900234,1.3386416396269512,sub_size/2) # Calcul
ate bounding box coordinates of reduced AOI
start = time.time() # Record time that image acquisition and processing begins
img_pan_rgb = CatalogImage('1040010031060B00', pansharpen=True, bbox=sub_AOI) # Req
uest image over reduced AOI from DigitalGlobe that is pansharpened
pan_np = img_pan_rgb.rgb() # Create true-colour image from acquired data
pan_plot = plt.imshow(pan_np)# Visually confirm image quality
pan_im = Image.fromarray(pan_np) # Convert array to Pillow 'Image' object
pan_im.save("image_returned_{}.tif".format(test_factor)) # Convert/save as .tif
end = time.time() # Record time that process completes
print(end - start) # Print time (in seconds)
```

Output:

139.6206760406494

8.2 User acceptance testing

To ensure the UI was handling the user inputs and running the underlying Python methods appropriately the tool was tested manually. A summary of these tests is provided in Table 7 below.

Table 7 - User Acceptance Testing Summary

#	Test Scenario	Expected result	Actual result	Pass/Fail
1	User clicks on map to find Latitude/Longitude coordinates	Tool displays coordinates in location of mouse click	As expected	Pass
2	User provides invalid Latitude/Longitude coordinates	Tool does not attempt to request imagery for provided coordinates	User Interface restricts input to valid coordinates	Pass
3	User provides non-existent dates	Tool does not attempt to request imagery for provided dates	User Interface restricts input to valid dates	Pass
4	'First' time-frame isn't distinctly before 'Second' timeframe	Tool does not attempt to request imagery for provided dates	Tool attempts to request imagery but notifies user	Fail but accepted
5	Imagery is not available for AOI/dates	Tool notifies user that imagery is not available	As expected	Pass
6	Cloud cover too high in imagery for algorithms	Tool notifies user that low cloud imagery is not available	Tool provides cloud mask as layer for inspection	Fail but accepted
7	Date range/AOI too large for acceptable run-time	Tool does not attempt to request imagery for date range/AOI	User Interface restricts input to reduce run-time	Pass
8	User runs Stage One and Stage Two in any combination	Tool requests imagery on basis of user input	As expected	Pass
9	User applies different combinations in layer control	Tool displays only top layer	As expected	Pass
10	Digital Globe or Sentinel Hub server failure	Tool notifies user that connection is not possible	As expected	Pass

Additional analysis and testing examples are provided in section 11.6.

9 Evaluation and Conclusion

This section provides an overall assessment of the project and includes a critical evaluation, lessons learnt and conclusion.

9.1 Critical Evaluation

The project had a very ambitious scope for an independent undertaking within the given timeframe, with papers detailing comparable research citing teams of contributors. I started with a complete lack of experience in satellite imagery analysis, cloud computing and Computer Vision, and the majority of Python libraries implemented were completely unfamiliar at project outset. To deliver a functional tool required a very steep learning curve and also resilience to failure, as the cutting-edge technologies used often proved unreliable to the point of unfeasibility after significant progress had been made and time invested. After facing issues with cloud cover that were not anticipated I produced my own cloud mask algorithm which I could apply to create a low cloud imagery mosaic. This is an active area of research in the field.

The project required me to work with a variety of imagery sources, both open and commercial, each necessitating a different approach for retrieval of suitable data. Several types of analysis are performed on the imagery retrieved. With the low-resolution imagery I used a deep residual CNN architecture as planned in the proposal but complimented this with a multispectral CART following unsatisfactory results. For the high-resolution imagery I have applied a cutting-edge object detection SSD CNN, which only became available after work on the project had started. As might be expected there are various ways in which the accuracy of the algorithms applied could be improved, from acquiring better quality training data to fine-tuning the parameters. I've provided more detail of these approaches and others in section 11.3.

Without a similar existing pipeline combining two types of satellite imagery for comparison it wasn't clear at outset whether the project was feasible, however I have successfully produced a proof-of-concept tool that retrieves and displays both high and low-resolution imagery, performing cutting-edge analysis on each. Taking the above into account, the project has been a very challenging but rewarding learning experience. I have significantly developed my Python programming and other technical skills, and have built a good understanding of the state-of-the-art research and technologies in the field of satellite EO. Section 11.3 provides a number of improvements that would have been made given more time.

9.2 Lessons learnt

Several of the key lessons learnt whilst undertaking this project are summarised below.

9.2.1 Virtual environments

Python libraries are rarely built in isolation and often require specific versions of other libraries to function. As the project progressed the number of libraries used grew to almost 250, listed in section 67, and this created a complex set of version interdependencies. After significant time was lost attempting to manually handle this I learnt to use virtual environments within Conda [40].

9.2.2 'Good enough' and 'nice to have'

At times I had to remind myself that this was a proof-of-concept project and thus every component did not need to be polished. Time was lost on small details that the reviewer will most likely not notice, and knowing when to stop/avoiding perfectionism was a skill I developed to deliver the project in time.

9.2.3 Unreliable technologies

Identifying earlier when a technology/library is unreliable would have saved significant amounts of time. Upon finding a suitable tool on GitHub for instance, activity in the 'Issues' log and date of last commit are good indicators of whether the developers of a library/tool are actively debugging and improving the technology.

9.2.4 Training data quality

Regardless of the design and architecture of the algorithm being trained, good quality data is vital for a high-performance predictive model. A large portion of this project was spent iteratively investigating and improving the quality of training data created for the Stage One CNN, whereas in hindsight the scope of the project could have been changed to allow a different pre-existing data source to be used instead.

9.3 Conclusion

This report has documented the process of researching, designing, and implementing the project tool. The project's goal was to investigate the field of satellite EO and the associated cutting-edge technologies, concepts and techniques, and to produce a relevant proof-of-concept satellite imagery retrieval and analysis pipeline tool from scratch.

The report introduces the field of satellite EO and sets out the project deliverables and objectives. In the background research (section 4) the relevant technologies, imagery data sources, and concepts are discussed. A functional specification is provided and the report details the system design and architecture. The implementation section gives an in-depth breakdown of the key steps and testing that were necessary to successfully build the tool, and further testing is also detailed. Finally, the report provides a critical evaluation of the project deliverables and lessons learnt, and future recommendations are provided in section 11.3.

10 References

[1] NASA, “NSSDCA Master Catalog,” 2017. [Online]. Available: https://nssdc.gsfc.nasa.gov/nmc/. [Accessed 20th March 2018].

[2] ILS, “COMMERCIAL SATELLITE LAUNCH HISTORY,” 2013. [Online]. Available: http://www.ilslaunch.com/node/33. [Accessed 20 March 2018].

[3] EUROCONSULT, “Satellite-Based Earth Observation Market Prospects to 2026,” October 2017.

[4] UK Government, “Surrey satellite firm set for lift off from India,” 16 September 2018. [Online]. Available: https://www.gov.uk/government/news/surrey-satellite-firm-set-for-lift-off-from-india. [Accessed 16 September 2018].

[5] space.com, “SpaceX Rocket Could Be 100-Percent Reusable by 2018, Elon Musk Says,” 10 April 2017. [Online]. Available: https://www.space.com/36412-spacex-completely-reusable-rocket-elon-musk.html.

[6] K. B. M. D. Rainer Sandau, “Small satellites for global coverage: Potential and limits,” in *ISPRS Journal of Photogrammetry and Remote Sensing*, 2010, pp. 492-504.

[7] P. Kansakar and F. Hossain, “A review of applications of satellite Earth Observation data for global societal benefit and stewardship of planet Earth,” *Space Policy,* vol. 36, pp. 46-54, 2016.

[8] DigitalGlobe, “U.S. Satellite Resolution Restrictions – LIFTED!,” 2014. [Online]. Available: http://blog.digitalglobe.com/news/resolutionrestrictionslifted/. [Accessed 20 March 2018].

[9] M. Wired, “U.S. Department of Commerce Relaxes Resolution Restrictions DigitalGlobe Extends Lead in Image Quality,” June 2014. [Online]. Available: http://www.marketwired.com/press-release/us-department-commerce-relaxes-resolution-restrictions-digitalglobe-extends-lead-image-nyse-dgi-1919482.htm. [Accessed 20 March 2018].

[10] Gizmodo, “Scientists Need Your Help Spotting Seals in Antarctica,” 16 April 2018. [Online]. Available: https://earther.gizmodo.com/scientists-need-your-help-spotting-seals-in-antarctica-1825251396. [Accessed 1 May 2018].

[11] E. O. Nsoesie, P. Butler, N. Ramakrishnan, S. R. Mekaru and J. S. Brownstein, “Monitoring Disease Trends using Hospital Traffic Data from High Resolution Satellite Imagery: A Feasibility Study,” PubMed Central, 2015.

[12] Nature, “US government considers charging for popular Earth-observing data,” 24 April 2018. [Online]. Available: https://www.nature.com/articles/d41586-018-04874-y. [Accessed 1 May 2018].

[13] U.S. Geological Survey, “Landsat 8 Data Users Handbook,” 2018. [Online]. Available: https://landsat.usgs.gov/landsat-8-l8-data-users-handbook-section-1. [Accessed 1 May 2018].

[14] NASA, “Landsat 8,” 2018. [Online]. Available: https://landsat.gsfc.nasa.gov/landsat-8/. [Accessed 1 May 2018].

[15] USGS, “Landsat Processing Details,” [Online]. Available: https://landsat.usgs.gov/landsat-processing-details. [Accessed 1 May 2018].

[16] European Commission, “What is Copernicus?,” 2018. [Online]. Available: http://www.copernicus.eu/main/overview. [Accessed 1 May 2018].

[17] European Space Agency, “Sentinel-2 Heritage,” 2018. [Online]. Available: https://sentinel.esa.int/web/sentinel/missions/sentinel-2/heritage. [Accessed 1 May 2018].

[18] NASA, “ESA-NASA Collaboration Fosters Comparable Land Imagery,” 13 February 2013. [Online]. Available: https://landsat.gsfc.nasa.gov/esa-nasa-collaboration-fosters-comparable-land-imagery/. [Accessed 1 May 2018].

[19] G. Forkuor, K. Dimobe, I. Serme and J. E. Tondoh, “Landsat-8 vs. Sentinel-2: examining the added value of sentinel-2’s red-edge bands to land-use and land-cover mapping in Burkina Faso,” *GIScience & Remote Sensing,* vol. 55, no. 3, pp. 331-354, 2018.

[20] L. Korhonen, Hadi, P. Packalen and M. Rautiainen, “Comparison of Sentinel-2 and Landsat 8 in the estimation of boreal forest canopy cover and leaf area index,” *Remote Sensing of Environment,* vol. 195, p. 259–274, 2017.

[21] European Space Agency, “Level-1C Processing,” 2018. [Online]. Available: https://earth.esa.int/web/sentinel/technical-guides/sentinel-2-msi/level-1c-processing. [Accessed 1 May 2018].

[22] European Space Agency, “Level-2A Processing,” 2018. [Online]. Available: https://earth.esa.int/web/sentinel/technical-guides/sentinel-2-msi/level-2a-processing. [Accessed 1 May 2018].

[23] Planet Labs, “PLANET IMAGERY AND ARCHIVE,” 2018. [Online]. Available: https://www.planet.com/products/planet-imagery/. [Accessed 20 March 2018].

[24] Planet Labs, “ROCKETING INTO 2018,” January 2018. [Online]. Available: https://www.planet.com/pulse/rocketing-into-2018/. [Accessed 20 March 2018].

[25] Planet, “Planet: Understanding the Amazon from Space,” Kaggle, 2017. [Online]. Available: https://www.kaggle.com/c/planet-understanding-the-amazon-from-space. [Accessed 20 March 2018].

[26] DigitalGlobe, “DigitalGlobe - Our constellation,” [Online]. Available: https://www.digitalglobe.com/about/our-constellation. [Accessed 20 March 2018].

[27] The New York Times, “Governments Tremble atGoogle's Bird's-Eye View,” December 2005. [Online]. Available: https://www.nytimes.com/2005/12/20/technology/governments-tremble-atgoogles-birdseye-view.html. [Accessed 20 March 2018].

[28] DigitalGlobe, “WorldView-3 Sensor Data Sheet,” 2014. [Online]. Available: https://www.spaceimagingme.com/downloads/sensors/datasheets/DG_WorldView3_DS_2014.pdf. [Accessed 1 May 2018].

[29] Google, “Google Earth Engine API,” [Online]. Available: https://developers.google.com/earth-engine/. [Accessed 20 March 2018].

[30] M. H. M. D. S. I. D. T. R. M. Noel Gorelick, “Google Earth Engine: Planetary-scale geospatial analysis for everyone,” *Remote Sensing of Environment,* vol. 202, pp. 18-27, 2017.

[31] “google/earthengine-api/issues #16,” GitHub, [Online]. Available: https://github.com/google/earthengine-api/issues/16.

[32] A. Karpathy, “CS231n Convolutional Neural Networks for Visual Recognition,” 2017. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed 20 March 2018].

[33] A. Krizhevsky, I. Sutskever and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” 2012.

[34] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *The Journal of Physiology,* p. 106–154, 1962.

[35] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going deeper with convolutions,” 2015.

[37] H. Kaiming, Z. Xiangyu, R. Shaoqing and S. Jian, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” 2015.

[38] H. Kaiming, Z. Xiangyu, R. Shaoqing and S. Jian, “Deep Residual Learning for Image Recognition,” 2015.

[39] Python Software Foundation, “Python,” 2018. [Online]. Available: https://www.python.org/. [Accessed 1 May 2018].

[40] Anaconda, Inc., “Conda,” 2017. [Online]. Available: https://conda.io/docs/. [Accessed 1 May 2018].

[41] Jupyter Team, “The Jupyter Notebook,” 2015. [Online]. Available: https://jupyter-notebook.readthedocs.io/. [Accessed 1 May 2018].

[42] Jupyter Widgets, “ipywidgets GitHub Repository,” 2018. [Online]. Available: https://github.com/jupyter-widgets/ipywidgets. [Accessed 1 May 2018].

[43] R. Story, “GitHub Folium respository,” 2013. [Online]. Available: https://github.com/python-visualization/folium. [Accessed 1 August 2018].

[44] V. Agafonkin, “Leaflet.js,” 2017. [Online]. Available: https://leafletjs.com/. [Accessed 1 May 2018].

[45] European Space Agency, “Copernicus Open Access Hub,” 2018. [Online]. Available: https://scihub.copernicus.eu/. [Accessed 1 May 2018].

[46] Sentinel Hub by Sinergise, “sentinel-py GitHub Repository,” 2018. [Online]. Available: https://github.com/sentinel-hub/sentinelhub-py. [Accessed 1 May 2018].

[47] Sentinel Hub by Sinergise, “Sentinel Hub services,” 2018. [Online]. Available: https://services.sentinel-hub.com/oauth/subscription. [Accessed 1 May 2018].

[48] European Space Agency, “Welcome to the OSEO Call submission area,” 2018. [Online]. Available: https://earth.esa.int/web/guest/pi-community/apply-for-data/ao-s?IFRAME_SRC=%2Fpi%2Fesa%3Fcmd%3Daodetail%26aoname%3DOSEO%26displayMode%3Dcenter%26targetIFramePage%3D%252Fweb%252Fguest%252Fpi-community%252Fapply-for-data%252Fao-s. [Accessed 1 May 2018].

[49] Open Geospatial Consortium, “Web Map Service,” 1999. [Online]. Available: http://www.opengeospatial.org/standards/wms. [Accessed 1 May 2018].

[50] Open Geospatial Consortium, “Web Coverage Service,” 2018. [Online]. Available: http://www.opengeospatial.org/standards/wcs. [Accessed 1 May 2018].

[51] European Space Agency, “SAFE,” 2017. [Online]. Available: http://earth.esa.int/SAFE/. [Accessed 1 May 2018].

[52] Sentinel Hub by Sinergise, “Changes of the access rights to L1C bucket at AWS Public Datasets (Requester Pays),” 6 May 2018. [Online]. Available: https://forum.sentinel-hub.com/t/changes-of-the-access-rights-to-l1c-bucket-at-aws-public-datasets-requester-pays/172. [Accessed 1 July 2018].

[53] Sentinel Hub by Sinergise, “Sentinel Hub Configurator,” 2018. [Online]. Available: https://apps.sentinel-hub.com/configurator/. [Accessed 1 May 2018].

[54] J. Fein, “PyGeoTools GitHub Repository,” 2013. [Online]. Available: https://github.com/jfein/PyGeoTools. [Accessed 1 May 2018].

[55] J. P. Matuschek, “Finding Points Within a Distance of a Latitude/Longitude Using Bounding Coordinates,” 2010. [Online]. Available: http://janmatuschek.de/LatitudeLongitudeBoundingCoordinates. [Accessed 1 May 2018].

[56] K. Stefan, Artist, *a Mercator projection map with Tissot's indicatrices.* [Art]. 2004.

[57] Wikipedia, “Great-circle distance,” [Online]. Available: https://en.wikipedia.org/wiki/Great-circle_distance. [Accessed 1 May 2018].

[58] J. Braaten, W. B Cohen and Z. Tang, “Automated cloud and cloud shadow identification in Landsat MSS imagery for temperate ecosystems,” *Remote Sensing of Environment,* pp. 128-138, 2015.

[59] P. Helber, B. Bischke, A. Dengel and D. Borth, “EuroSAT: A Novel Dataset and Deep Learning Benchmark,” 2017.

[60] Development Seed, “Label Maker GitHub Repository,” 2018. [Online]. Available: https://github.com/developmentseed/label-maker. [Accessed 1 May 2018].

[61] Kaggle, “Understanding the Amazon from Spaze 1st place interview,” 17 October 2017. [Online]. Available: http://blog.kaggle.com/2017/10/17/planet-understanding-the-amazon-from-space-1st-place-winners-interview/. [Accessed 1 May 2018].

[62] R. Minetto, M. P. Segundo and S. Sarkar, “Hydra: an Ensemble of Convolutional Neural Networks for Geospatial Land Classification,” 2018.

[63] ethereon, “Netscope Resnet50 architecture visualisation,” 2018. [Online]. Available: http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006. [Accessed 1 May 2018].

[64] Amazon Web Services, “AWS Instance Types,” 2018. [Online]. Available: https://aws.amazon.com/ec2/instance-types/. [Accessed 1 May 2018].

[65] Amazon Web Services, “P2 Instances,” [Online]. Available: https://aws.amazon.com/ec2/instance-types/p2/. [Accessed 1 May 2018].

[66] Amazon Web Services, “AWS Command Line Interface,” 2018. [Online]. Available: https://aws.amazon.com/cli/. [Accessed 1 May 2018].

[67] Google, “Earth Engine Classification,” 2018. [Online]. Available: https://developers.google.com/earth-engine/classification. [Accessed 1 May 2018].

[68] J. Stephenson, “Google Earth Engine CART algorithm script,” [Online]. Available: https://code.earthengine.google.com/ef76eeb57f620188297cd3a1c0695d7d. [Accessed 1 May 2018].

[69] DigitalGlobe, “gbdxtools GitHub Repository,” 2018. [Online]. Available: https://github.com/DigitalGlobe/gbdxtools. [Accessed 1 May 2018].

[70] DIUx, “xView Dataset,” 2018. [Online]. Available: http://xviewdataset.org/. [Accessed 2018 1 March].

[71] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, “SSD: Single Shot MultiBox Detector,” 2016.

[72] DIUx, “xView baseline models GitHub Repository,” 2018. [Online]. Available: https://github.com/DIUx-xView/baseline. [Accessed 1 May 2018].

[73] Mapbox, “Mapbox Satellite,” 2018. [Online]. Available: https://www.mapbox.com/maps/satellite/. [Accessed 1 May 2018].

[74] OpenStreetMap, “OSM Lab OSM QA Tiles,” 2018. [Online]. Available: https://osmlab.github.io/osm-qa-tiles/. [Accessed 1 May 2018].

[75] OpenStreetMap, “OSM Zoom levels,” 2018. [Online]. Available: https://wiki.openstreetmap.org/wiki/Zoom_levels. [Accessed 1 May 2018].

[76] DigitalGlobe, “DigitalGlobe Blog - Global Basemap,” 12 July 2011. [Online]. Available: http://blog.digitalglobe.com/tag/global-basemap/. [Accessed 1 May 2018].

[77] OpenStreetMap, “OSM Features,” 2017. [Online]. Available: https://wiki.openstreetmap.org/wiki/Map_Features . [Accessed 1 May 2018].

[78] Mapbox, “Mapboz Style Specification - Other filter,” 2018. [Online]. Available: https://www.mapbox.com/mapbox-gl-js/style-spec/#other-filter. [Accessed 1 May 2018].

[79] OpenStreetMap, “OSM wiki Highways definition,” 2018. [Online]. Available: https://wiki.openstreetmap.org/wiki/Highways. [Accessed 1 May 2018].

[80] OpenStreetMap, “OSM Contributors,” 2018. [Online]. Available: https://wiki.openstreetmap.org/wiki/Contributors. [Accessed 1 May 2018].

[81] P. Henderson and V. Ferrari, “End-to-End Training of Object Class Detectors for Mean Average Precision,” *Lecture Notes in Computer Science,* vol. 10115, 2016.

[82] Space Applications Catapult, “The Satellite Applications Catapult,” 2018. [Online]. Available: https://sa.catapult.org.uk/. [Accessed 1 May 2018].

11 Appendices

11.1 List of Acronyms

AOI	Area-of-interest
AWS	Amazon Web Services
CART	Classification and Regression Tree
CNN	Convolutional Neural Network
CRS	Coordinate Reference System
DIUx	Defense Innovation Unit Experimental
EC2	Elastic Cloud Compute
EO	Earth Observation
ESA	European Space Agency
GBDX	Geo Big Data Platform
GEE	Google Earth Engine
mAP	mean Average Precision
MSI	Multispectral Instrument
NASA	National Aeronautics and Space Administration
NIR	Near-infrared
OGC	Open Geospatial Consortium
OLI	Operational Land Imager
OSM QA	OpenStreetMap Quality Assurance
S3	Simple cloud Storage Service
SSD	Single-Shot Multibox Detector
SWIR	Short-wave Infrared
TIRS	Thermal Infrared Sensor
TOA	Top-of-Atmosphere
UI	User Interface
WCS	Web Coverage Service
WGS84	World Geodetic System 1984
WKT	Well-known text
WMS	Web Map Service

11.2 Glossary

Polar-orbit An orbit in which a satellite passes above both poles of the body being orbited on each revolution.

TOA reflectance The measure of light which reflects off the planet at the point of the (satellite) sensor without atmospheric correction.

Swath width The width of the area being imaged on the surface by a (satellite) sensor.

Near-infrared A term referring to the spectral region of infrared light with wavelengths near to that of visible light

Short-wave infrared A term referring to the spectral region of infrared light with a wavelength range immediately following that of near-infrared.

11.3 De-scoped Functionality and Future Improvements

This section provides both the de-scoped functionality and a sample of suggested future improvements for the project tool.

11.3.1 De-scoped functionality

11.3.1.1 Enhanced UI

Due to time constraints a light-weight UI was built using the Folium Python library in which the layers were displayed using a feature layer panel. The original intention was to have slider functionality for comparing the first and second periods at Stage One. Due to the limitations of the Folium library this requirement was de-scoped.

11.3.1.2 Specific focus on developments in the South China Sea

Although the project was initially inspired by the opportunity to detect change in disputed territories of the South China Sea it was found that the majority of change in these regions had taken place before the time period that the relatively recent Sentinel-2 Satellites became active. There continues to be activity in this region, and so the tool could still prove useful for this application in future, and including a longer-running satellite imagery source such as Landsat 8 could expand this analysis historically.

11.3.1.3 Fine-tuning the CNNs

Training a Neural Network often involves an iterative trial-and-error approach, in which various parameters such as ‘learning rate’ and ‘decay’ are tweaked to find the optimal result. Due to time constraints only a selection of Neural Network training parameter variations were tested, falling short of the precise adjustments required to validate the use of the term ‘fine-tuning’.

11.3.2 Future improvements

11.3.2.1 User specified ‘change of interest’ type

The main focus of the tool is to identify where there has been a change from rural/natural land cover to urban/man made. By applying algorithms trained to identify alternative classes the tool could be enhanced with the option to detect other types of change, for example deforestation.

11.3.2.2 Additional satellite imagery data sources

Where low-resolution Sentinel-2 imagery cloud cover is too high, or imagery does not exist for a certain timeframe, other imagery data sources could be used such as Landsat-8 data. A similar approach could be taken for high-resolution imagery, whereby another commercial imagery provider such as Planet Labs could be queried for available imagery where the DigitalGlobe retrievals are not successful. Both enhancements would necessitate retraining the algorithms applied on the data types and resolutions that these imagery sources provide.

The Worldview-4 Satellite imagery data is not available at present, but in future including this source would also enhance the likelihood of a successful DigitalGlobe server retrieval. With premium account level permissions and an extended Use Case the functionality could also be introduced to order DigitalGlobe ‘acquisition’ products as well.

11.3.2.3 Algorithm performance improvements

Due to the rapidly growing nature of the satellite industry there are new labelled data sources are becoming available at an increasing rate. In future it is highly likely that low-resolution labelled imagery will be available that is suitable for the purposes of the tool. In the interim period the manual method applied to label the imagery acquired from label-maker could be expanded upon to improve the performance of the Stage One CNN.

The winning strategies from the current xView challenge, once released, could be adapted to fine-tune the Stage Two CNN for greater object detection accuracy.

The CART algorithm was trained on an imagery mosaic produced using a function in GEE to minimise cloud clover that as a consequence also maximised the chance of the image pixels being within a cloud shadow. An improved approach could be devised to produce a cloud free mosaic in GEE using another method, or alternatively data could be labelled within the Python environment directly from Sentinel Hub, which would enable more powerful methods such as random forests to be applied.

11.3.2.4 Additional UI and Graphical Display improvements

Displaying the prediction confidence level next to the class label within the bounding box of detected objects would allow the user to make a more informed decision on what level to set the confidence threshold at.

A limitation of the Folium library is that it can only pass data from Python to leaflet.js and not vice-versa, and so auto-populating the latitude and longitude selections based on map interactivity was not possible. In future a different library could be used which offers this functionality.

11.4 User set-up manual

To access satellite imagery using the project tool the user must first sign up for accounts at Sentinel Hub [47] and DigitalGlobe [26].

In order to run the tool certain library dependencies must be conformed with, and so within the folder a Virtual Environment file named ‘projectEnvironment.yml’ is stored. With Conda installed the user should then run the following from the command line with the project folder set as the working directory:

```
$ conda env create -f projectEnvironment.yml
$ source activate projEnv
$ jupyter notebook
```

This will activate the Virtual Environment and open a Jupyter Notebook server instance within the default browser.

Next the user should open the ‘Tool.ipynb’ notebook file, enter their Sentinel Hub and DigitalGlobe credentials and run the code. The UI will be displayed and the tool is now initialised and ready to use.

11.5 Implementation process for building a training data set using label-maker (additional detail for section 7.4.2)

The following section describes the implementation process of building a training data set for the Stage One CNN using label-maker from Development Seed [60].

The label-maker tool downloads pre-processed satellite imagery tiles from Mapbox [73] and label information from OpenStreetMap Quality Assurance (OSM QA) tiles [74]. The process of utilising this tool went through several iterations, with each intending to make improvements to the training data acquired and, as a result, the accuracy of the Neural Network being trained with the data. The key implementation details for building the data set are detailed below:

Zoom level - When requesting the imagery through label-maker the zoom level must be specified as an integer between 0 and 19, with larger zoom levels representing higher-resolution imagery. Zoom level 14 represents a spatial resolution of 9.547 m/pixel at the equator [75], which is roughly equivalent to Sentinel-2 imagery which has a maximum resolution of 10 m/pixel. At this zoom level Mapbox provides satellite imagery from a combination of open and proprietary sources including DigitalGlobe's Global Basemap [76]. Section 11.6.1 provides a comparison of imagery at this zoom level with that of Sentinel-2.

Country and **bounding_box** – The label-maker tool requires both country and bounding box for which to request relevant imagery and OSM QA tiles. To improve the training data I requested and combined imagery from multiple locations and countries in the proximity of the South China Sea, illustrated in Figure 32.

Figure 32 - Map highlighting countries (blue) for which satellite imagery was sampled using the label-maker tool. Philippines, Indonesia (Lombok), Indonesia (Borneo), Malaysia, Vietnam, Brunei, China, Taiwan

Classes – The label-maker tool allows classes to be defined as a combination of OSM features [77] using the 'Mapbox GL Filters' [78] syntax. The purpose of the tool is to identify evidence of man-made objects or human change, as such a key challenge was identifying a combination of class features that didn't result in a dataset where **all** land cover tiles contained these features.

Figure 33 illustrates the land coverage of various combinations of OSM features for an area in the Philippines:

- The red tiles show areas for which the representative OSM tiles contain the 'building' label.
- This expands to the purple tiles too when also including tiles where any of: 'man_made', 'military', 'aerialway', or 'aeroway' features are present, or if landuse is any of 'allotments', 'brownfield', 'cemetery', 'commercial', 'construction', 'depot', 'farmland', 'farmyard', 'garages', 'greenhouse_horticulture', 'industrial', 'landfill', 'military', 'orchard', 'plant_nursery', 'port', 'quarry', 'railway', 'recreation_ground', 'religious', 'residential', 'retail', 'village_green', 'vineyard'.
- This expands further to include the orange tiles simply with the addition of 'highways', which is defined by OSM as '... any road, route, way, or thoroughfare on land...' [79]

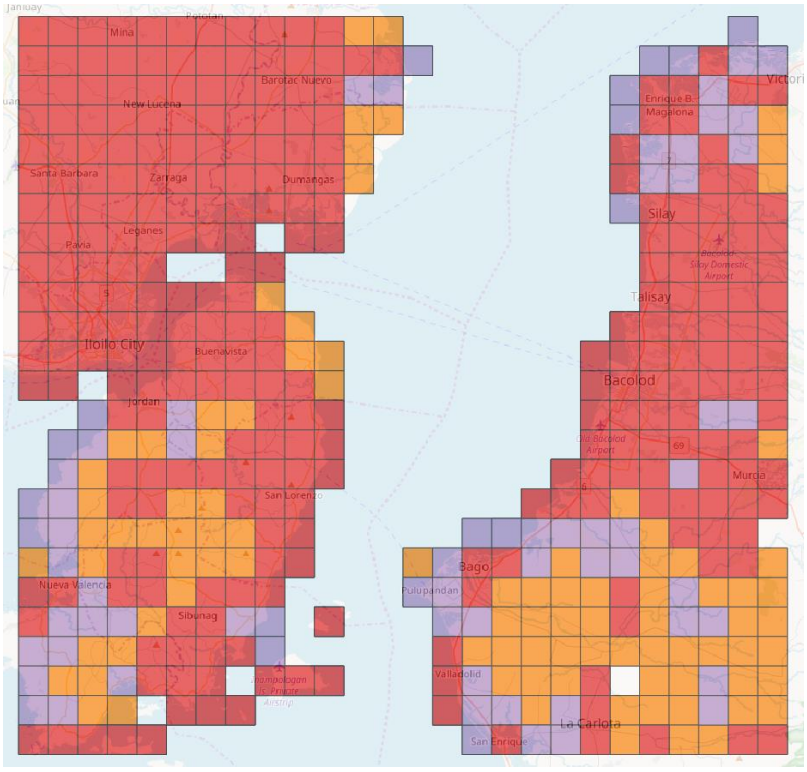

Figure 33 – Illustration of feature coverage within OSM QA tiles

It was decided following this testing that 'highways' are too prolific in OSM tiles and can be found within almost all land tiles at zoom level 14. A balance of covered/non-covered land tiles, whilst also including a range of man-made objects, was achieved by excluding the tiles only containing 'highways' from the list above. The resulting class feature coverage is illustrated by the red and purple tiles in Figure 33. The final syntax used for the classes is shown below:

"classes": [{ "name": "human construction", "filter": ["any", ["has", "man_made"], ["has", "military"], ["has", "building"], ["has", "aerialway"], ["has", "aeroway"], ["in", "landuse", "allotments", "brownfield", "cemetery", "commercial", "construction", "depot", "farmland", "farmyard", "garages", "greenhouse_horticulture", "industrial", "landfill", "military", "orchard", "plant_nursery", "port", "quarry", "railway", "recreation_ground", "religious", "residential", "retail", "village_green", "vineyard"]] }],

Despite carefully designed class syntax and a variety of countries/locations the CNN was performing poorly when applied in testing. An approach taken to improve upon this was to manually cleanse the data of poor-quality images, examples of which are shown in Figure 34.

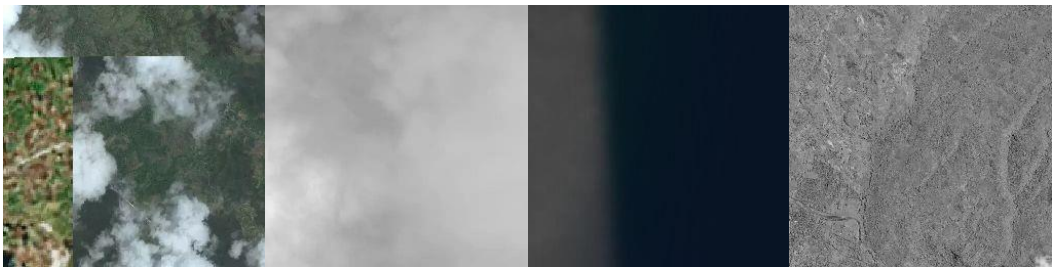

Figure 34 - Examples of poor quality label-maker imagery

The images were visually reviewed in .jpg format, with Table 8 showing resulting impact this had on the size of the dataset. The images were then converted to Numpy arrays and georeferenced with OSM QA tiles for labelling. This was a similar strategy to that used by the team producing the EuroSAT dataset [59].

Table 8 - Impact on size of training data of imagery review

Country	Image count pre-review	Image count post-review	% reduction
Philippines	3,471	1,801	48%
Indonesia (Lombok)	2,008	1,334	34%
Indonesia (Borneo)	1,208	616	49%
Malaysia	830	300	64%
Vietnam	1,150	995	13%
Brunei	554	497	10%
China	1,912	1,207	37%
Taiwan	1,150	995	13%
Total	**12,283**	**7,745**	**37%**

Training the CNN with the imagery cleansed as above led to only a small improvement in the CNN performance. The next point of investigation was the OSM QA tile labelling, unfortunately this labelling proved to be highly unreliable. There are two causes for this; firstly, OSM label quality is entirely dependent on a range of contributors, from organisations and governments to individual volunteers, to provide data and keep it up to date and accurate [80]. Secondly, an inspection revealed a software bug wherein at country boundaries the label-maker tool doesn't download certain OSM QA tiles, as illustrated by Figure 35 which shows the OSM tile coverage is missing in the image on the right near to the border between Singapore and Malaysia. With inaccurate or missing OSM QA tile data the label-maker tool doesn't create the correct class labels for the data set.

Figure 35 – Illustration of full OSM QA tile coverage (left) and missing OSM tile coverage (right)

As a result of this finding I was forced to re-label the data by hand. This was a time-intensive process and with significant portions of the project outstanding I could only re-label a subset of the cleansed imagery data.

11.6 Additional testing and analysis

The following section provides additional relevant testing and analysis to compliment sections 7 and 8.

11.6.1 Imagery comparison between label-maker and Sentinel-2

An imagery comparison was performed over the same geolocation to confirm that the zoom level of labelled imagery acquired using the label-maker tool for training purposes was at the correct resolution and detail to that of Sentinel-2. Figure 36 illustrates the results of this testing. The image on the left from label-maker is too detailed at zoom level 15, whereas the central image from label-maker at zoom level 14 is a close match to the Sentinel-2 acquisition on the right.

Figure 36 - Left to right: label-maker acquisiton at Zoom 15, label-maker acquisition at Zoom 14, Sentinel-2 acquisition from Sentinel Hub

11.6.2 Stage One Resnet50 Classification CNN

One of the indications that the label-maker data was not suitable for the purposes of the project tool was the output of training attempts. Figure 37 illustrates the performance of the model at each epoch when being trained on the data which had been cleansed of poor-quality imagery, but before the OSM QA tile labelling accuracy had been checked. Despite the loss and accuracy rates improving for the training set this pattern was not reflected in the validation set, prompting the investigation that identified missing OSM QA tiles as detailed in 7.4.2.

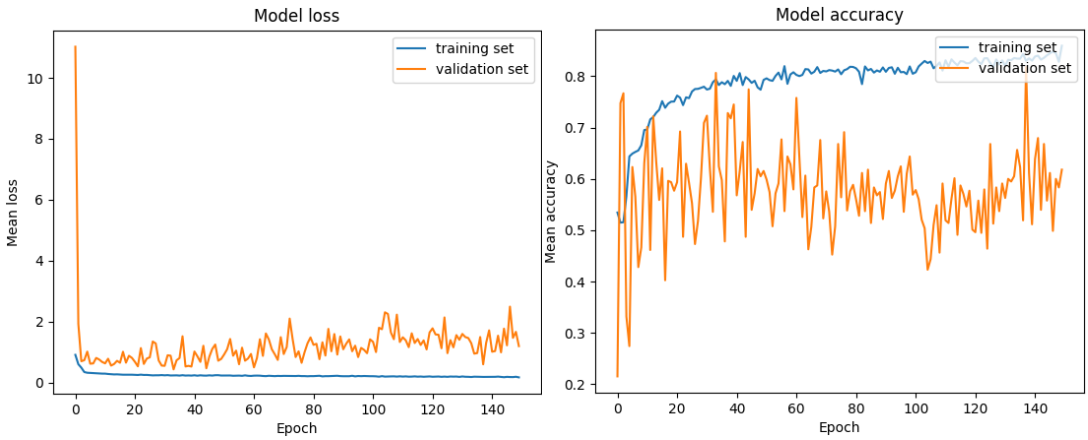

Figure 37 - Loss and accuracy levels at each epoch when training the Resnet50 CNN with cleansed but poorly labelled imagery. The loss function is categorical cross-entropy.

Applying this CNN to the imagery retrieved from Sentinel Hub had unexpected results, as illustrated by Figure 38 in which the confidence of 'man made' is 0.96 in the image on the left and 0.40 in the image on the right.

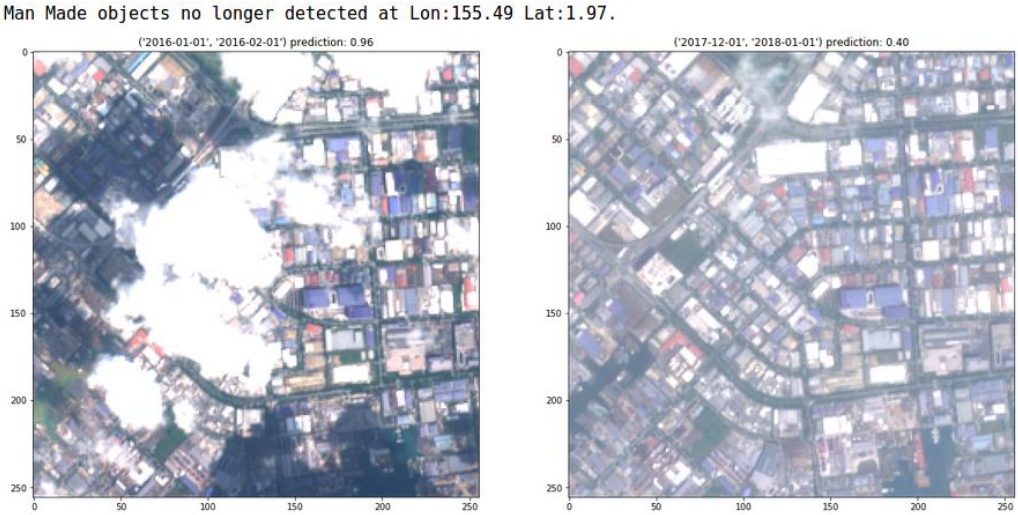

Figure 38 - Illustration of unexpected results when applying an earlier iteration of the Stage One CNN

Despite the manually re-labelled data set being significantly smaller, a stronger correlation between training and validation set loss/accuracy was observed, illustrated in Figure 39.

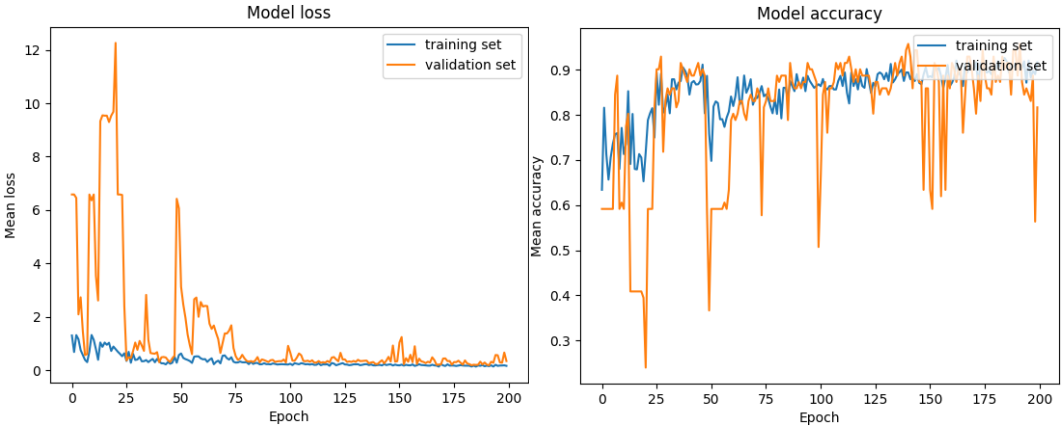

Figure 39 - Loss and accuracy levels at each epoch when training the Resnet50 CNN with manually re-labelled imagery. The loss function is categorical cross-entropy.

11.6.3 Second Stage SSD Object Detection CNN

As part of the DIUx xView Challenge the weights for three benchmark CNN models were released. Each model was trained with the DIUx xView dataset on 4 GPUs for 7 days using interpolated mean Average Precision (mAP) [81] as their backpropagation loss function. The total mAP score across all classes that resulted from this training is provided in Table 9 below.

Table 9 - mAP score across all classes of 'Vanilla', 'Multires', and 'Aug' benchmark CNNs

	Vanilla	Multires	Aug
Total mAP	0.1456	0.2590	0.1549

The 'Multires' benchmark model showed the best performance in testing and has been adapted and implemented at Stage Two in the project tool. Figure 40 shows an example of the bounding box and class labels output when running Stage Two with the AOI over an airport in Vietnam. It was found that small changes to the confidence threshold had a significant impact on the detections that were displayed, due to this a slider for this parameter was provided in the UI.

Figure 40 - Bounding box and class label output from Object Detection CNN

11.7 Program code and algorithms

11.7.1 Google Earth Engine CART Algorithm script

```
// Adapted from code segments reviewed at the SAC [82]
// First create a composite of Sentinel-2 imagery
var s2collection = ee.ImageCollection('COPERNICUS/S2')
    .filterDate('2017-05-01', '2018-05-01')
var S2composite = s2collection.min(); // uses .min() function to minimise cloud cover
// Select the red, green and blue bands which will allow for visual identification of land-use.
Map.addLayer(S2composite, {
    bands: ['B4', 'B3', 'B2'],
    gain: '0.1, 0.1, 0.1'
}, 'S2 Min Composite');

// Merge the three geometry layers created by hand into a single FeatureCollection.
var newfc = urban_manmade.merge(vegetation_natural).merge(water);


// Based on testing the following bands have been used for the classification.
var bands = ['B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8'];
// The name of the property on the points storing the class label.
var classProperty = 'landcover';

// Create a set of training data.
var S2training = S2composite.select(bands).sampleRegions({
    collection: newfc,
    properties: [classProperty],
    scale: 30
});

// Train the CART classifier.
var S2classifier = ee.Classifier.cart().train({
    features: S2training,
    classProperty: classProperty,
});

// Print the architecture of the CART for transfer to Python environment
print('CART, explained', S2classifier.explain());


// Classify the composite to review accuracy on global scale.
var S2classified = S2composite.classify(S2classifier);
Map.centerObject(newfc);
Map.addLayer(S2classified, {
    min: 0,
    max: 2,
    palette: ['red', 'green', 'blue']
}, 'CART estimate');

// Test accuracy of CART model
var withRandom = S2training.randomColumn('random');

// Split the data in to training and testing sets
var split = 0.7; // 70% training, 30% testing.
var trainingPartition = withRandom.filter(ee.Filter.lt('random', split));
var testingPartition = withRandom.filter(ee.Filter.gte('random', split));

// Train the classifier with 70% of the data.
var S2trainedClassifier = ee.Classifier.gmoMaxEnt().train({
    features: trainingPartition,
    classProperty: classProperty,
```

```
    inputProperties: bands
});

// Classify the test set.
var S2test = testingPartition.classify(S2trainedClassifier);

// Print the confusion matrix.
var S2confusionMatrix = S2test.errorMatrix(classProperty, 'classification');
print('Confusion Matrix', S2confusionMatrix);
```

11.7.2 Python libraries

Table 10 provides a comprehensive list of all Python libraries that were required for the project in <library>=<version> format.

Table 10 - Python libraries and versions required for project tool

absl-py=0.4.0	gast=0.2.0	libspatialite=4.3.0a	pthread-stubs=0.4	tk=8.6.8
affine=2.2.1	gdal=2.2.4	libssh2=1.8.0	ptyprocess=0.6.0	toolz=0.9.0
altair=2.2.2	geos=3.6.2	libstdcxx-ng=7.2.0	py=1.5.4	tornado=5.1
appdirs=1.4.3	geotiff=1.4.2	libtiff=4.0.9	pyasn1=0.4.4	tqdm=4.24.0
asn1crypto=0.24.0	gettext=0.19.8.1	libuuid=2.32.1	pyasn1-modules=0.2.1	traitlets=4.3.2
astor=0.7.1	giflib=5.1.4	libxcb=1.13	pycparser=2.18	typing=3.6.4
atomicwrites=1.1.5	glib=2.55.0	libxml2=2.9.8	pydot=1.2.4	urllib3=1.23
attrs=18.1.0	gmp=6.1.2	locket=0.2.0	pygments=2.2.0	utm=0.4.2
automat=0.7.0	gst-plugins-base=1.12.5	mako=1.0.7	pygpu=0.7.6	vcrpy=1.13.0
backcall=0.1.0	gstreamer=1.12.5	markdown=2.6.11	pyhamcrest=1.9.0	vincent=0.4.4
basemap=1.1.0	h5py=2.8.0	markupsafe=1.0	pyjwt=1.6.4	wcwidth=0.1.7
bleach=2.1.3	hdf4=4.2.13	matplotlib=2.2.3	pyopenssl=18.0.0	webencodings=0.5
blinker=1.4	hdf5=1.10.2	mercantile=1.0.4	pyparsing=2.2.0	werkzeug=0.14.1
bokeh=0.13.0	heapdict=1.0.0	mistune=0.8.3	pyproj=1.9.5.1	wheel=0.31.1
boost-cpp=1.67.0	html5lib=1.0.1	mkl_fft=1.0.5	pyqt=5.6.0	widgetsnbextension=3.4.0
boto3=1.7.77	hyperlink=17.3.1	mkl_random=1.0.1	pyshp=1.2.12	wrapt=1.10.11
botocore=1.10.77	icu=58.2	more-itertools=4.2.0	pysocks=1.6.8	xerces-c=3.2.0
branca=0.3.0	idna=2.7	msgpack-python=0.5.6	pytest=3.7.1	xorg-kbproto=1.0.7
bumpversion=0.5.3	imageio=2.3.0	nb_conda=2.2.1	pytest-runner=4.2	xorg-libice=1.0.9
bzip2=1.0.6	incremental=17.5.0	nb_conda_kernels=2.1.1	python=3.5.5	xorg-libsm=1.2.2
ca-certificates=2018.4.16	ipykernel=4.8.2	nbconvert=5.3.1	python-dateutil=2.7.3	xorg-libx11=1.6.5
cachetools=2.1.0	ipython=6.5.0	nbformat=4.4.0	pytz=2018.5	xorg-libxau=1.0.8
cairo=1.14.12	ipython_genutils=0.2.0	ncurses=6.1	pywavelets=0.5.2	xorg-libxdmcp=1.1.2
certifi=2018.8.13	ipywidgets=7.4.0	networkx=2.1	pyyaml=3.12	xorg-libxext=1.3.3
cffi=1.11.5	jedi=0.12.1	notebook=5.6.0	pyzmq=17.1.2	xorg-libxrender=0.9.10
chardet=3.0.4	jinja2=2.10	oauthlib=2.1.0	qt=5.6.2	xorg-renderproto=0.11.1
click=6.7	jmespath=0.9.3	olefile=0.45.1	rasterio=1.0.3	xorg-xextproto=7.3.0
click-plugins=1.0.3	jpeg=9c	openjpeg=2.3.0	readline=7.0	xorg-xproto=7.0.31
cligj=0.4.0	json-c=0.12.1	openssl=1.0.2o	requests=2.19.1	xz=5.2.4
cloudpickle=0.5.3	jsonschema=2.6.0	packaging=17.1	requests-futures=0.9.7	yaml=0.1.7
constantly=15.1.0	jupyter_client=5.2.3	pandas=0.23.4	requests-oauthlib=1.0.0	zeromq=4.2.5
cryptography=2.3.1	jupyter_core=4.4.0	pandoc=2.2.2	s3transfer=0.1.13	zict=0.1.3
cryptography-vectors=2.3.1	kealib=1.4.9	pandocfilters=1.4.2	scikit-image=0.14.0	zlib=1.2.11
curl=7.61.0	keras=2.1.6	parso=0.3.1	send2trash=1.5.0	zope.interface=4.5.0
cycler=0.10.0	kiwisolver=1.0.1	partd=0.3.8	service_identity=17.0.0	grpcio=1.12.1
cytoolz=0.9.0.1	krb5=1.14.6	pathlib2=2.3.2	setuptools=40.0.0	intel-openmp=2018.0.3
dask=0.18.2	libffi=3.2.1	pcre=8.41	shapely=1.6.4	libcurl=7.61.0
dask-core=0.18.2	libgcc=7.2.0	pexpect=4.6.0	simplegeneric=0.8.1	mkl=2018.0.3
dbus=1.13.0	libgcc-ng=7.2.0	pickleshare=0.7.4	sip=4.18	numpy=1.14.2
decorator=4.3.0	libgdal=2.2.4	pillow=5.2.0	six=1.11.0	pycurl=7.43.0.2
distributed=1.22.1	libgfortran=3.0.0	pip=18.0	snuggs=1.4.1	scipy=1.1.0
docutils=0.14	libgfortran-ng=7.2.0	pixman=0.34.0	sortedcontainers=2.0.4	twisted=18.7.0
entrypoints=0.2.3	libgpuarray=0.7.6	pluggy=0.7.1	sqlite=3.24.0	configparser=3.5.0
ephem=3.7.6.0	libiconv=1.15	poppler=0.61.1	tblib=1.3.2	gbdx-auth=0.4.0
expat=2.2.5	libkml=1.3.0	poppler-data=0.4.9	tensorboard=1.9.0	gbdxtools=0.15.11
folium=0.6.0	libnetcdf=4.6.1	proj4=4.9.3	tensorflow=1.9.0	msgpack=0.5.6
fontconfig=2.13.0	libpng=1.6.35	prometheus_client=0.3.0	termcolor=1.1.0	opencv-python=3.4.2.17
freetype=2.9.1	libpq=9.6.3	prompt_toolkit=1.0.15	terminado=0.8.1	sentinelhub=2.4.1
freexl=1.0.5	libprotobuf=3.6.0	protobuf=3.6.0	testpath=0.3.1	tifffile=0.15.1
future=0.16.0	libsodium=1.0.16	psutil=5.4.7	theano=1.0.2	

11.7.3 Python methods and functions

```
def bbox_split_grid(user_dist_km, km_per_tile):
    """
    Calculate the number of whole tiles required to split the user-provided distance
    in to a square grid

    Args:
        user_dist_km : distance from point on map to be retrieved and analysed
        km_per_tile : width/height of tiles to split grid by

    Outputs:
        An integer representing the width/height in whole tiles to split the calculated area by

    """
    bbox_split_xy = math.ceil(user_dist_km / (km_per_tile / 2))
    return(bbox_split_xy)


def user_dist_km_upper(user_dist_km, km_per_tile):
    """
    Calculate width/height in km of total area to be covered based on number of tiles in grid

    Args:
        user_dist_km : distance from point on map to be retrieved and analysed
        km_per_tile : width/height of tiles to split grid by

    Outputs:
        A float representing the width/height in km of total area to be retrieved

    """
    km_upper = bbox_split_grid(user_dist_km, km_per_tile) * (km_per_tile / 2)
    return(km_upper)


def user_large_bbox(user_Lon_deg, user_Lat_deg, user_dist_km, km_per_tile):
    """
    Calculate Bounding Box coordinates of total area to be covered based on user input

    Args:
        user_Lon_deg : User provided longitude coordinate of AOI
        user_Lat_deg : User provided latitude coordinate of AOI
        user_dist_km : distance from point on map to be retrieved and analysed
        km_per_tile : width/height of tiles to split grid by

    Outputs:
        The latitude and longitude coordinates of the lower left and upper right corners of the
total area
        to be retrieved

    """
    user_loc = GeoLocation.from_degrees(
        user_Lat_deg, user_Lon_deg)  # Note: LAT,LON format not LON,LAT format
    user_SW_loc, user_NE_loc = user_loc.bounding_locations(
        user_dist_km_upper(user_dist_km, km_per_tile))
    return user_SW_loc, user_NE_loc


def user_sub_bbox_coords_v2(
        user_Lon_deg,
        user_Lat_deg,
        user_dist_km,
        km_per_tile):
    """
    Calculate the Bounding Box coordinates for each tile in total area to be retrieved

    Args:
        user_Lon_deg : User provided longitude coordinate of AOI
        user_Lat_deg : User provided latitude coordinate of AOI
        user_dist_km : distance from point on map to be retrieved and analysed
        km_per_tile : width/height of tiles to split grid by

    Outputs:
        A list containing the latitude and longitude coordinates of the lower left and
        upper right
        corners of each tile in total area to be retrieved
        A list containing the grid position of each tile
```

```
    """
    # First use earlier function to return larger bbox coords
    user_SW_loc, user_NE_loc = user_large_bbox(
        user_Lon_deg, user_Lat_deg, user_dist_km, km_per_tile)

    # With larger bbox geo coords calculated we can now split.
    # First create polygon to feed to sentinel hub BBoxSplitter
    # method/function:
    user_polyg = asPolygon([[user_SW_loc.deg_lon,
                             user_SW_loc.deg_lat],
                            [user_SW_loc.deg_lon,
                             user_NE_loc.deg_lat],
                            [user_NE_loc.deg_lon,
                             user_NE_loc.deg_lat],
                            [user_NE_loc.deg_lon,
                             user_SW_loc.deg_lat]])

    # With polygon created as input to splitter method, feed in with number of
    # sub_bboxes calculated by bbox_split_grid:
    user_bbox_splitter = BBoxSplitter(
        [user_polyg], CRS.WGS84, (bbox_split_grid(
            user_dist_km, km_per_tile), bbox_split_grid(
            user_dist_km, km_per_tile)))

    # List of BBox created suitable for sentinel hub requests
    user_sub_bbox_list = user_bbox_splitter.get_bbox_list()

    # Information regarding grid position of each bounding box and original
    # large bbox
    user_sub_info_list = user_bbox_splitter.get_info_list()
    return user_sub_bbox_list, user_sub_info_list


def sentinel_bands_and_dates_req(sub_bbox, user_date, INSTANCE_ID):
    """
    Retrieves imagery for a given Bounding Box and date range using the sentinel hub
    WCSRequest method

    Args:
        sub_bbox : bounding box coordinates of tile for which imagery is to be requested
        user_date : the date range of imagery capture required
        INSTANCE_ID : credentials for accessing Sentinel Hub servers

    Outputs:
        A list containing all imagery data available for given parameters
        A list containing the dates of capture for each image

    """
    wcs_all_bands_request = WcsRequest(layer='BANDS-S2-L1C',  # Sentinel 2 imagery source with
                                                              all available bands (B01,B02,B03,B04,B05,
                                                              B06,B07,B08,B8A,B09,B10,B11,B12)
                                       bbox=sub_bbox,  # Which bounding box
                                       time=user_date,  # When imagery requested
                                       resx='10m', resy='10m',  # Metres per pixel, only valid
                                                                  for WCS requests
                                       instance_id=INSTANCE_ID,
                                       # Instance ID created during set-up and
                                       # configuration and set above
                                       image_format=MimeType.TIFF_d32f,
                                       # TIFF_d32f, since Sentinel-2's 13 bands
                                       # can not be held in a png/jpg image.
                                       # By default logo in bottom left, this
                                       # removes that
                                       custom_url_params={
                                           CustomUrlParam.SHOWLOGO: False}
                                       )
    wcs_all_bands_tiff = wcs_all_bands_request.get_data()
    wcs_all_bands_dates = wcs_all_bands_request.get_dates()
    return(wcs_all_bands_tiff, wcs_all_bands_dates)


def clip(a):
    """
    Limits the value of input to range (0,1)

    Args:
```

```
        a : value to be clipped to range (0,1)
    Outputs:
        A float in range (0,1)

    """
    if a >= 0:
        if a <= 1:
            return a
        else:
            return 1
    else:
        return 0


def image_cloudMask_cloudScore_date(
        wcs_all_bands_single_tiff,
        wcs_all_bands_single_date):
    """
    Identifies cloud coverage and calculates two types of cloud mask and cloud score

    Args:
        wcs_all_bands_single_tiff : an individual image
        wcs_all_bands_single_date : the date of capture of the image provided

    Outputs:
        A dictionary object containing the original image, both types of cloud mask
        and cloud score, the date of capture of the image

    """
    # adaptation of algorithm proposed by [Braaten, Cohen, Yang, 2015]
    wcs_tiff_cloud = np.zeros(
        shape=wcs_all_bands_single_tiff[:, :, [3, 2, 1]].shape)
    wcs_tiff_cloud_basic = np.zeros(
        shape=wcs_all_bands_single_tiff[:, :, [3, 2, 1]].shape)
    cc = 0
    cc_basic = 0
    for i in range(len(wcs_tiff_cloud)):
        for j in range(len(wcs_tiff_cloud[i])):

            # calculate whether heavy or medium cloud for each pixel:
            bRatio = (wcs_all_bands_single_tiff[i]
                      [j][2] - 0.175) / (0.39 - 0.175)
            NGDR = (wcs_all_bands_single_tiff[i][j][2] - wcs_all_bands_single_tiff[i][j][3]) / (
                wcs_all_bands_single_tiff[i][j][2] + wcs_all_bands_single_tiff[i][j][3])

            if bRatio > 1:
                v = 0.5 * (bRatio - 1)
                wcs_tiff_cloud[i][j][0] = 0.5 * \
                    clip(wcs_all_bands_single_tiff[i][j][3])
                wcs_tiff_cloud[i][j][1] = 0.5 * \
                    clip(wcs_all_bands_single_tiff[i][j][2])
                wcs_tiff_cloud[i][j][2] = (
                    0.5 * clip(wcs_all_bands_single_tiff[i][j][1])) + v

                wcs_tiff_cloud_basic[i][j][0] = (
                    0.5 * clip(wcs_all_bands_single_tiff[i][j][1])) + v

                cc += wcs_tiff_cloud[i][j][2]
                cc_basic += 1

            elif bRatio > 0 and NGDR > 0:
                v = 5 * math.sqrt(bRatio * NGDR)
                wcs_tiff_cloud[i][j][0] = (
                    0.5 * clip(wcs_all_bands_single_tiff[i][j][3])) + v
                wcs_tiff_cloud[i][j][1] = 0.5 * \
                    clip(wcs_all_bands_single_tiff[i][j][2])
                wcs_tiff_cloud[i][j][2] = 0.5 * \
                    clip(wcs_all_bands_single_tiff[i][j][1])

                wcs_tiff_cloud_basic[i][j][0] = (
                    0.5 * clip(wcs_all_bands_single_tiff[i][j][3])) + v

                cc += wcs_tiff_cloud[i][j][0]
                cc_basic += wcs_tiff_cloud[i][j][0]

            else:
                wcs_tiff_cloud[i][j][0] = 0.5 * \
```

```
                    clip(wcs_all_bands_single_tiff[i][j][3])
                wcs_tiff_cloud[i][j][1] = 0.5 * \
                    clip(wcs_all_bands_single_tiff[i][j][2])
                wcs_tiff_cloud[i][j][2] = 0.5 * \
                    clip(wcs_all_bands_single_tiff[i][j][1])
    img_dict = dict()
    img_dict['tiff'] = wcs_all_bands_single_tiff
    img_dict['cloud'] = wcs_tiff_cloud
    img_dict['cloud_basic'] = wcs_tiff_cloud_basic
    img_dict['cc'] = cc / 65536
    img_dict['cc_basic'] = cc_basic / 65536
    img_dict['date'] = wcs_all_bands_single_date
    return img_dict


def best_image_cloud_dates_dict(wcs_all_bands_tiff, wcs_all_bands_dates):
    """
    For a range of images over multiple dates calculates cloud masks and cloud scores
    and returns image with lowest cloud score

    Args:
        wcs_all_bands_tiff : a list of images
        wcs_all_bands_dates : a list of the date of capture of the images provided

    Outputs:
        A dictionary object containing the lowest cloud cover image, cloud masks, cloud scores a
nd capture date
    """
    images_dict = dict()
    if len(wcs_all_bands_tiff) == 0:
        print("No imagery for selected dates")
        return
    for i in range(len(wcs_all_bands_tiff)):
        images_dict[i] = image_cloudMask_cloudScore_date(
            wcs_all_bands_tiff[i], wcs_all_bands_dates[i])

    # From images_dict apply a lamda function to select index of image with
    # lowest cloud:
    key_min = min(images_dict.keys(), key=(lambda k: images_dict[k]['cc']))

    return(images_dict[key_min])


def folium_bounds(sub_bbox):
    """
    Rearranges tile coordinates to appropriate Folium format for projection

    Args:
        sub_bbox : the bounding box coordinates to be rearranged

    Outputs:
        A list containing rearranged coordinates
    """
    low_left = sub_bbox.get_lower_left()
    up_right = sub_bbox.get_upper_right()

    fol_bound = [[low_left[1], low_left[0]], [up_right[1], up_right[0]]]
    return(fol_bound)


def landcover_cart(wcs_all_bands_single_tiff, cc):
    """
    Applies a CART algorithm pixel-wise to classify land cover in imagery

    Args:
        wcs_all_bands_single_tiff : an individual image
        cc : the cloud cover score for the image

    Outputs:
        A image in numpy array format representing the classification estimate
    """
    B2 = 1
    B3 = 2
    B4 = 3
    B5 = 4
    B6 = 5
    B8 = 7
```

```
    factor_t = 3.3

    im_mean = np.mean(wcs_all_bands_single_tiff[:, :, [
                      0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]], axis=(0, 1, 2))
    mean_factor = ((im_mean * (1 - cc) * factor_t))

    wcs_tiff_CART = np.zeros(
        shape=wcs_all_bands_single_tiff[:, :, [3, 2, 1]].shape)
    for i in range(len(wcs_tiff_CART)):
        for j in range(len(wcs_tiff_CART[i])):

            if wcs_all_bands_single_tiff[i][j][B6] <= 0.385108 * mean_factor:
                if wcs_all_bands_single_tiff[i][j][B8] <= 0.304296 * \
                        mean_factor:
                    wcs_tiff_CART[i][j][2] = 1
                else:
                    wcs_tiff_CART[i][j][0] = 1

            else:
                if wcs_all_bands_single_tiff[i][j][B2] <= 0.951 * mean_factor:
                    if wcs_all_bands_single_tiff[i][j][B8] <= 0.76668 * \
                            mean_factor:
                        if wcs_all_bands_single_tiff[i][j][B2] <= 0.82584 * \
                                mean_factor:
                            wcs_tiff_CART[i][j][1] = 1

                        else:
                            wcs_tiff_CART[i][j][0] = 1

                    else:
                        if wcs_all_bands_single_tiff[i][j][B2] <= 0.917636 * \
                                mean_factor:
                            wcs_tiff_CART[i][j][1] = 1

                        else:
                            if wcs_all_bands_single_tiff[i][j][B5] <= 0.95164 * \
                                    mean_factor:
                                if wcs_all_bands_single_tiff[i][j][B4] <= 0.52226 * \
                                        mean_factor:
                                    if wcs_all_bands_single_tiff[i][j][B2] <= 0.938008 * \
                                            mean_factor:
                                        wcs_tiff_CART[i][j][1] = 1

                                    else:
                                        wcs_tiff_CART[i][j][0] = 1

                                else:
                                    wcs_tiff_CART[i][j][0] = 1

                            else:
                                wcs_tiff_CART[i][j][1] = 1

                else:
                    if wcs_all_bands_single_tiff[i][j][B2] <= 1.007008 * \
                            mean_factor:
                        if wcs_all_bands_single_tiff[i][j][B3] <= 0.9032 * \
                                mean_factor:
                            wcs_tiff_CART[i][j][0] = 1

                        else:
                            wcs_tiff_CART[i][j][1] = 1

                    else:
                        wcs_tiff_CART[i][j][0] = 1
    return(wcs_tiff_CART)


def landcov_proportions(land_cover, cloud_before, cloud_after):
    """
    Calculates the proportion of each land cover type from a CART classification estimate for
    pixels which are cloud-free in both timeframes

    Args:
        land_cover : the CART classification of an individual image
        cloud_before : the cloud cover mask for an image from the first timeframe
        cloud_after : the cloud cover mask for an image from the first timeframe
```

```
    Outputs:
        A list containing the proportions of each landcover type
    """
    man_made = 0
    natural = 0
    water = 0
    tot = 0
    for i in range(len(land_cover)):
        for j in range(len(land_cover[i])):

            # only consider pixels without cloud in either before or after
            if cloud_before[i][j][0] == 0 and cloud_after[i][j][0] == 0:
                man_made += land_cover[i][j][0]
                natural += land_cover[i][j][1]
                water += land_cover[i][j][2]
                tot += 1
    if tot == 0:  # Too much cloud
        return([0, 0, 0])
    return([man_made / tot, natural / tot, water / tot])


def polyline_bounds(sub_bbox):
    """
    Rearranges tile coordinates to appropriate format for polygon projection in Folium layer

    Args:
        sub_bbox : the bounding box coordinates to be rearranged

    Outputs:
        A list containing rearranged coordinates
    """
    low_left = sub_bbox.get_lower_left()
    up_right = sub_bbox.get_upper_right()

    poline_bound = [[low_left[1], low_left[0]], [up_right[1], low_left[0]], [
        up_right[1], up_right[0]], [low_left[1], up_right[0]], [low_left[1], low_left[0]]]
    return(poline_bound)


def process_predict_img(user_image):
    """
    Processes satellite image to correct size and format, CNN to make prediction

    Args:
        user_image : an RGB numpy array

    Outputs:
        CNN prediction in [a,b] format where a,b are in range (0,1) and a+b = 1.
    """
    if user_image.shape != (256, 256, 3):
        user_image = cv2.resize(user_image, dsize=(
            256, 256), interpolation=cv2.INTER_CUBIC)
    # normalize the image
    user_image = user_image.astype('float32')
    img_mean = np.mean(user_image, axis=(0, 1, 2))
    img_std = np.std(user_image, axis=(0, 1, 2))
    user_image -= img_mean
    user_image /= img_std
    # add a column to match expected shape
    test_img_exp = np.expand_dims(user_image, axis=0)
    # make prediction
    prediction = res50_model.predict(test_img_exp)
    return prediction[0]


def first_stage_map_layer_dicts(
        user_Lon_deg,
        user_Lat_deg,
        user_dist_km,
        km_per_tile,
        user_date_start,
        user_date_finish):
    """
    Combines methods to create dictionary objects for display in Folium map

    Args:
```

```
    user_Lon_deg : User provided longitude coordinate of AOI
    user_Lat_deg : User provided latitude coordinate of AOI
    user_dist_km : distance from point on map to be retrieved and analysed
    km_per_tile : width/height of tiles to split grid by
    user_date_start : the date range of imagery capture required for first period
    user_date_finish : the date range of imagery capture required for second period


Outputs:
    A dictionary for each timeframe containing lowest cloud imagery and analysis
    A dictionary containing comparison analysis between both timeframes
"""
# With functions above first create bbox_list:
bbox_list, bbox_info = user_sub_bbox_coords_v2(
    user_Lon_deg, user_Lat_deg, user_dist_km, km_per_tile=2.56)
# Run through bbox_list returning best image in each bbox based on cloud
# coverage

start_master_dict = dict()
for i in range(len(bbox_list)):
    wcs_all_bands_tiff, wcs_all_bands_dates = sentinel_bands_and_dates_req(
        bbox_list[i], user_date_start, INSTANCE_ID)
    start_master_dict[i] = best_image_cloud_dates_dict(
        wcs_all_bands_tiff, wcs_all_bands_dates)
    start_master_dict[i]['land'] = landcover_cart(
        start_master_dict[i]['tiff'], start_master_dict[i]['cc'])
    start_master_dict[i]['fol_bounds'] = folium_bounds(bbox_list[i])

finish_master_dict = dict()
for i in range(len(bbox_list)):
    wcs_all_bands_tiff, wcs_all_bands_dates = sentinel_bands_and_dates_req(
        bbox_list[i], user_date_finish, INSTANCE_ID)
    finish_master_dict[i] = best_image_cloud_dates_dict(
        wcs_all_bands_tiff, wcs_all_bands_dates)
    finish_master_dict[i]['land'] = landcover_cart(
        finish_master_dict[i]['tiff'], finish_master_dict[i]['cc'])
    finish_master_dict[i]['fol_bounds'] = folium_bounds(bbox_list[i])

# Calculates land cover proportions in the images only where cloud cover
# isn't heavy in either before or after

for i in range(len(bbox_list)):
    start_master_dict[i]['land_cov_prop'] = landcov_proportions(
        start_master_dict[i]['land'],
        finish_master_dict[i]['cloud_basic'],
        start_master_dict[i]['cloud_basic'])
    # print(start_master_dict[i]['land_cov_prop'])
print("First Period imagery retrieval and analysis complete.")

for i in range(len(bbox_list)):
    finish_master_dict[i]['land_cov_prop'] = landcov_proportions(
        finish_master_dict[i]['land'],
        finish_master_dict[i]['cloud_basic'],
        start_master_dict[i]['cloud_basic'])
    # print(finish_master_dict[i]['land_cov_prop'])
print("Second Period imagery retrieval and analysis complete.")

# A dictionary holding comparison analysis
compare_dict = dict()

# Loop below takes maximum proportional change across each land use type
for i in range(len(bbox_list)):

    compare_dict[i] = dict()  # set up dict for each image

    norm_dif = [0, 0, 0]  # initialise list for proportions

    # for each land cover type calculate a proportional change
    for j in range(len(finish_master_dict[i]['land_cov_prop'])):
        # case when potential to divide by zero:
        if start_master_dict[i]['land_cov_prop'][j] == 0:
            norm_dif[j] = math.fabs(
                (finish_master_dict[i]['land_cov_prop'][j] -
                 start_master_dict[i]['land_cov_prop'][j]))
        else:
            norm_dif[j] = math.fabs(
                (finish_master_dict[i]['land_cov_prop'][j] -
```

```
                    start_master_dict[i]['land_cov_prop'][j]) /
                   start_master_dict[i]['land_cov_prop'][j])

        # take the maximum land cover change across all types
        compare_dict[i]['land_cov_max_change'] = max(norm_dif)

        # create polygon coordinates for map display
        compare_dict[i]['polyline_bounds'] = polyline_bounds(bbox_list[i])

    # For displaying takes max landcover change over whole AOI and takes
    # proportions for each area of this
    key_min = max(compare_dict.keys(), key=(
        lambda k: compare_dict[k]['land_cov_max_change']))
    for i in range(len(bbox_list)):
        if compare_dict[key_min]['land_cov_max_change'] == 0:
            compare_dict[i]['land_cov_max_change_prop'] = 0
        else:
            compare_dict[i]['land_cov_max_change_prop'] = compare_dict[i]['land_cov_max_change']
/ \compare_dict[key_min]['land_cov_max_change']

    # Create a mask which combines cloud from both images
    for i in range(len(bbox_list)):
        compare_dict[i]['combined_cloud'] = np.zeros(
            shape=finish_master_dict[i]['cloud_basic'].shape)
        for j in range(len(compare_dict[i]['combined_cloud'])):
            for k in range(len(compare_dict[i]['combined_cloud'][j])):
                compare_dict[i]['combined_cloud'][j][k][0] = max(
                    finish_master_dict[i]['cloud_basic'][j][k][0],
                    start_master_dict[i]['cloud_basic'][j][k][0])

    # Mix the land cover from both images to understand what is being taken in
    # to account, and what is masked by cloud
    for i in range(len(bbox_list)):
        compare_dict[i]['non_cloud_landcover'] = np.zeros(
            shape=finish_master_dict[i]['cloud_basic'].shape)
        for j in range(len(compare_dict[i]['non_cloud_landcover'])):
            for k in range(len(compare_dict[i]['non_cloud_landcover'][j])):
                if compare_dict[i]['combined_cloud'][j][k][0] == 0:
                    compare_dict[i]['non_cloud_landcover'][j][k][0] = max(
                        finish_master_dict[i]['land'][j][k][0],\
                                        start_master_dict[i]['land'][j][k][0])
                    compare_dict[i]['non_cloud_landcover'][j][k][1] = max(
                        finish_master_dict[i]['land'][j][k][1],\
                                        start_master_dict[i]['land'][j][k][1])
                    compare_dict[i]['non_cloud_landcover'][j][k][2] = max(
                        finish_master_dict[i]['land'][j][k][2],\
                                        start_master_dict[i]['land'][j][k][2])

    # Calculates land cover proportions in the images only where cloud cover
    # isn't heavy in either before or after

    for i in range(len(bbox_list)):
        finish_master_dict[i]['CNN_pred'] = process_predict_img(
            finish_master_dict[i]['tiff'][:, :, [3, 2, 1]])[1]
        # print(finish_master_dict[i]['CNN_pred'])

    for i in range(len(bbox_list)):
        start_master_dict[i]['CNN_pred'] = process_predict_img(
            start_master_dict[i]['tiff'][:, :, [3, 2, 1]])[1]
        # print(start_master_dict[i]['CNN_pred'])

    for i in range(len(bbox_list)):
        compare_dict[i]['CNN_change'] = math.fabs(
            finish_master_dict[i]['CNN_pred'] -
            start_master_dict[i]['CNN_pred'])
        # print(compare_dict[i]['CNN_change'])
    print("Comparison analysis complete.")
    return(start_master_dict, finish_master_dict, compare_dict)


def first_stage_layers_visualise(
        start_master_dict,
        finish_master_dict,
        compare_dict,
        user_Lon_deg,
        user_Lat_deg):
    """
```

```
    Creates feature layers for display in Folium map

    Args:
        start_master_dict : A dictionary for the first timeframe containing lowest cloud imagery
and analysis
        finish_master_dict : A dictionary for the second timeframe containing lowest cloud image
ry and analysis
        compare_dict : A dictionary containing comparison analysis between both timeframes
        user_Lon_deg : User provided longitude coordinate of AOI
        user_Lat_deg : User provided latitude coordinate of AOI

    Outputs:
        A dictionary for each timeframe containing lowest cloud imagery and analysis
        A dictionary containing comparison analysis between both timeframes
    """
    m = folium.Map(
        location=[user_Lat_deg, user_Lon_deg],
        tiles='OpenStreetMap',
        zoom_start=14,
        control_scale=True
    )

    after_image = folium.FeatureGroup(name='Second Period Imagery')
    for i in range(len(finish_master_dict)):
        folium.raster_layers.ImageOverlay(
            image=np.minimum(finish_master_dict[i]['tiff'][:, :, [3, 2, 1]] * 3, 1),
            bounds=finish_master_dict[i]['fol_bounds'],
            opacity=1
        ).add_to(after_image)

    before_image = folium.FeatureGroup(name='First Period Imagery')
    for i in range(len(start_master_dict)):
        folium.raster_layers.ImageOverlay(
            image=np.minimum(start_master_dict[i]['tiff'][:, :, [3, 2, 1]] * 3, 1),
            bounds=start_master_dict[i]['fol_bounds'],
            opacity=1
        ).add_to(before_image)

    combined_image_cloud = folium.FeatureGroup(
        name='Combined Period Cloud Cover', show=False)
    for i in range(len(start_master_dict)):
        folium.raster_layers.ImageOverlay(
            image=np.minimum(compare_dict[i]['combined_cloud'], 1),
            bounds=start_master_dict[i]['fol_bounds'],
            opacity=1
        ).add_to(combined_image_cloud)

    after_image_CART = folium.FeatureGroup(
        name='Second Period Land Cover', show=False)
    for i in range(len(finish_master_dict)):
        folium.raster_layers.ImageOverlay(
            image=np.minimum(finish_master_dict[i]['land'] * 3, 1),
            bounds=finish_master_dict[i]['fol_bounds'],
            opacity=1
        ).add_to(after_image_CART)

    before_image_CART = folium.FeatureGroup(
        name='First Period Land Cover', show=False)
    for i in range(len(start_master_dict)):
        folium.raster_layers.ImageOverlay(
            image=np.minimum(start_master_dict[i]['land'] * 3, 1),
            bounds=start_master_dict[i]['fol_bounds'],
            opacity=1
        ).add_to(before_image_CART)

    comp_image_land = folium.FeatureGroup(name='Land cover change', show=False)
    for i in range(len(compare_dict)):
        folium.Polygon(
            locations=compare_dict[i]['polyline_bounds'],
            color='red',
            opacity=compare_dict[i]['land_cov_max_change_prop'],
            fill=True,
            fill_opacity=compare_dict[i]['land_cov_max_change_prop'] / 4
        ).add_to(comp_image_land)

    comp_CNN_pred = folium.FeatureGroup(name='CNN Estimate change', show=False)
    for i in range(len(compare_dict)):
```

```
            folium.Polygon(
                locations=compare_dict[i]['polyline_bounds'],
                color='orange',
                opacity=compare_dict[i]['CNN_change'],
                fill=True,
                fill_opacity=compare_dict[i]['CNN_change'] / 4,
                popup='CNN Estimate 1st Period: {}. CNN Estimate 2nd Period: {}'.format(
                    start_master_dict[i]['CNN_pred'],
                    finish_master_dict[i]['CNN_pred'])).add_to(comp_CNN_pred)

        comp_CNN_pred.add_to(m)
        comp_image_land.add_to(m)
        combined_image_cloud.add_to(m)
        after_image_CART.add_to(m)
        before_image_CART.add_to(m)
        after_image.add_to(m)
        before_image.add_to(m)

        m.add_child(folium.LatLngPopup())
        folium.LayerControl().add_to(m)

        plugins.Fullscreen(
            position='topright',
            title='Full screen',
            title_cancel='Exit full screen',
            force_separate_button=True).add_to(m)

        return(m)


    def user_GBDXCatalog_bbox(user_Lon_deg, user_Lat_deg, user_km=2.56 / 7):
        """
        Calculates bounding box coordinates for GBDX imagery request

        Args:
            user_Lon_deg : User provided longitude coordinate of AOI
            user_Lat_deg : User provided latitude coordinate of AOI
            user_km : width/height of AOI to be returned

        Outputs:
            A list containing GBDX format bounding box coordinates for AOI
        """
        user_loc = GeoLocation.from_degrees(
            user_Lat_deg, user_Lon_deg)  # Note: Method takes LAT/LON and not LON/LAT
        user_SW_loc, user_NE_loc = user_loc.bounding_locations(user_km)
        return([user_SW_loc.deg_lon, user_SW_loc.deg_lat, user_NE_loc.deg_lon, user_NE_loc.deg_lat])


    def sorted_CatalogID_list(
            user_AOI_bbox,
            filters=["(sensorPlatformName = 'WORLDVIEW03_VNIR')"]):
        """
        Requests metadata for DigitalGlobe imagery of AOI

        Args:
            user_AOI_bbox : A list containing GBDX format bounding box coordinates for AOI
            filters : the filters to apply to catalog imagery before returning metadata

        Outputs:
            A sorted list containing filtered GBDX imagery metadata over AOI
        """
        wkt_AOI = box(*user_AOI_bbox).wkt
        results = gbdx.catalog.search(searchAreaWkt=wkt_AOI, filters=filters)

        GBDX_results = []
        for i in range(len(results)):
            GBDX_results.append([])
            GBDX_results[i].append(results[i]['properties']['cloudCover'])
            GBDX_results[i].append(
                datetime.datetime.strptime(
                    results[i]['properties']['timestamp'],
                    "%Y-%m-%dT%H:%M:%S.%fZ"))
            GBDX_results[i].append(results[i]['properties']['catalogID'])

        GBDX_results_date = sorted(
            GBDX_results,
            key=lambda image: image[1],
```

```
        reverse=True)
    GBDX_results_date_cloud = sorted(
        GBDX_results_date,
        key=lambda image: math.ceil(
            image[0] / 10))

    return(GBDX_results_date_cloud)


def GBDX_IMG_Request(sorted_CatalogID_list, GBDX_bbox):
    """
    Requests imagery data from DigitalGlobe servers until first successful retrieval

    Args:
        sorted_CatalogID_list : A sorted list containing filtered GBDX imagery metadata over AOI
        GBDX_bbox : A list containing GBDX format bounding box coordinates for AOI

    Outputs:
        A pansharpened image in numpy array format
        The path to the locally stored image
    """
    for i in range(len(sorted_CatalogID_list)):
        try:
            print("Requesting image with catalogID = {}, captured at {}.".format(
                sorted_CatalogID_list[i][2], sorted_CatalogID_list[i][1]))
            img_pan_request = CatalogImage(
                sorted_CatalogID_list[i][2],
                pansharpen=True,
                bbox=GBDX_bbox)
            print("Image successfully retrieved.")
            print("Processing image.")
            print("...")
            pan_np = img_pan_request.rgb()
            pan_im = Image.fromarray(pan_np)
            pan_im.save(
                "Image_prioritised_{}.tif".format(
                    sorted_CatalogID_list[i][2]))
            print("Image with catalogID = {} saved as 'image_prioritised_{}.tif'".format(
                sorted_CatalogID_list[i][2], sorted_CatalogID_list[i][2]))
            img_path = "Image_prioritised_{}.tif".format(
                sorted_CatalogID_list[i][2])
            return(pan_np, img_path)
        except BaseException:
            print(
                "Image not available, attempting next image. NOTE: This image may have increased
cloud coverage")
    print("Imagery not available at given coordinates, consider changing coordinates or ordering
imagery")


def on_button_clicked_stage_one(b):
    """
    Runs first stage of tool when button is clicked

    Args:
        NA

    Outputs:
        Interactive map displaying imagery and analysis
    """
    try:
        # Take weeks that user provides and create date range
        user_date_start = (str(user_date_start_low.value), str(
            (user_date_start_low.value) + datetime.timedelta(weeks=user_date_start_len.value)))
        user_date_finish = (str(user_date_final_low.value), str(
            (user_date_final_low.value) + datetime.timedelta(weeks=user_date_final_len.value)))

        user_dist_km = user_dist_km_UI.value

        user_Lon_deg = user_longitude.value
        user_Lat_deg = user_latitude.value

        print("Running Stage One.")
        print("User inputs: Longitude - {}, Latitude - {}, Radius - {}, \
First Period - {}, Second Period - {}".format(user_Lon_deg, user_Lat_deg, user_dist_km, user_dat
e_start, user_date_finish))
```

```
        print("Retrieving and analysing Sentinel-2 imagery")
        start_master_dict, finish_master_dict, compare_dict = first_stage_map_layer_dicts(
            user_Lon_deg, user_Lat_deg, user_dist_km, km_per_tile, user_date_start, user_date_fi
nish)
        m = first_stage_layers_visualise(
            start_master_dict,
            finish_master_dict,
            compare_dict,
            user_Lon_deg,
            user_Lat_deg)

        print("about to display map")
        with out:
            clear_output()
            display(widgets.VBox(boxes))
            display(m)
        # m.save("html_obj_det_map_{}_{}.html".format(user_Lon_deg,user_Lat_deg))
        #webbrowser.open('file://' + os.path.realpath("html_obj_det_map_{}_{}.html".format(user_
Lon_deg,user_Lat_deg)))

    except TypeError:
        print("Please review parameters.")
    except BaseException:
        print("Unexpected error")


def initialise_UI():
    """
    Initialises User Interface and Map Display

    Args:
        NA

    Outputs:
        User Interface and Interactive Map
    """

    # Header at top of tool
    title = widgets.HTML(
        value="<h1 style='font-size:40px;background-color:LightBlue;text-align:center;border:2px
solid Black;'>\
    Welcome to the project tool <b style='color:White;'>Stage One & Two</b></h1> \
    <p style='font-size:20px;text-align:center;'>Please choose your parameters and click <b>Run
Stage One </b> or <b>Run Stage Two </b> </p>",
        layout=widgets.Layout(
            width='100%',
            height='200%'))

    # Setting style to ensure full label is displayed
    style = {'description_width': 'initial'}  # to display whole label in UI

    global user_longitude

    user_longitude = widgets.FloatSlider(
        value=103.5718,
        min=-180,
        max=180,
        step=0.0001,
        description='Longitude (slider):',
        style=style,
        layout=widgets.Layout(width='66%', height='100%'),
        disabled=False,
        continuous_update=False,
        orientation='horizontal',
        readout=True,
        readout_format='.4f',
    )

    global user_longitude_freetext
    user_longitude_freetext = widgets.BoundedFloatText(
        value=103.5718,
        min=-180,
        max=180.0,
        step=0.0001,
        description='Longitude (freetext):',
        style=style,
        disabled=False
```

```
)

global lon_link
lon_link = widgets.jslink(
    (user_longitude, 'value'), (user_longitude_freetext, 'value'))

global user_latitude
user_latitude = widgets.FloatSlider(
    value=1.3426,
    min=-90,
    max=90,
    step=0.0001,
    description='Latitude (slider):',
    style=style,
    layout=widgets.Layout(width='66%', height='100%'),
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.4f',
)

global user_latitude_freetext
user_latitude_freetext = widgets.BoundedFloatText(
    value=1.3426,
    min=-90,
    max=90.0,
    step=0.0001,
    description='Latitude (freetext):',
    style=style,
    disabled=False
)

global lat_link
lat_link = widgets.jslink(
    (user_latitude, 'value'), (user_latitude_freetext, 'value'))

global user_dist_km_UI
user_dist_km_UI = widgets.FloatSlider(
    value=2.0,
    min=1,
    max=5.0,
    step=0.1,
    description='Radius of interest (km):',
    style=style,
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
)

global user_date_start_low
user_date_start_low = widgets.DatePicker(
    description='First period start date:',
    value=datetime.date(2016, 5, 13),
    style=style,
    disabled=False
)

global user_date_start_len
user_date_start_len = widgets.IntSlider(
    value=1.0,
    min=1.0,
    max=8.0,
    step=1.0,
    description='Period length (weeks):',
    style=style,
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.0f',
)

global user_date_final_low
user_date_final_low = widgets.DatePicker(
```

```
    description='Second period start date:',
    value=datetime.date(2018, 6, 1),
    max=datetime.datetime.now(),
    style=style,
    disabled=False
)

global user_date_final_len
user_date_final_len = widgets.IntSlider(
    value=1.0,
    min=1.0,
    max=8.0,
    step=1.0,
    description='Period length (weeks):',
    style=style,
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.0f',
)

global conf_thresh
conf_thresh = widgets.FloatSlider(
    value=0.6,
    min=0.2,
    max=1.0,
    step=0.05,
    description='Object Confidence Min:',
    style=style,
    #layout=widgets.Layout(width='66%', height='100%'),
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.2f',
)

run_button_one = widgets.Button(
    description="Run Stage One (Low resolution Earth Observation)",
    layout=widgets.Layout(
        width='50%',
        height='100%'),
)
run_button_one.style.button_color = 'lightblue'
run_button_one.on_click(on_button_clicked_stage_one)

run_button_two = widgets.Button(
    description="Run Stage Two (High resolution Object Detection)",
    layout=widgets.Layout(
        width='50%',
        height='100%'),
)
run_button_two.style.button_color = 'orange'
run_button_two.on_click(on_button_clicked_stage_two)

m = folium.Map(
    location=[12, 110],
    tiles='OpenStreetMap',
    zoom_start=5,
    control_scale=True
)
m.add_child(folium.LatLngPopup())

title_box = widgets.HBox([title])
toppest_box = widgets.HBox([user_longitude, user_longitude_freetext])
top_box = widgets.HBox([user_latitude, user_latitude_freetext])
middle_box = widgets.HBox(
    [user_date_start_low, user_date_start_len, user_dist_km_UI])
lower_box = widgets.HBox(
    [user_date_final_low, user_date_final_len, conf_thresh])
button_box = widgets.HBox([run_button_one, run_button_two])

out = widgets.Output(layout={'border': '1px solid black'})
boxes = [
    title_box,
    toppest_box,
```

```
            top_box,
            middle_box,
            lower_box,
            button_box]
        return out, boxes, m


    def draw_bboxes(img, boxes, classes):
        """
        Reference: Adapted from DIUx-xView code - [72] - https://github.com/DIUx-xView/baseline
        Draw bounding boxes on top of an image

        Args:
            img : Array of image to be modified
            boxes: An (N,4) array of boxes to draw, where N is the number of boxes.
            classes: An (N,1) array of classes corresponding to each bounding box.

        Outputs:
            An array of the same shape as 'img' with bounding boxes
                and classes drawn

        """

        name_from_class = dict()
        name_from_class[11] = 'Fixed-wing Aircraft'
        name_from_class[12] = 'Small Aircraft'
        name_from_class[13] = 'Passenger/Cargo Plane'
        name_from_class[15] = 'Helicopter'
        name_from_class[17] = 'Passenger Vehicle'
        name_from_class[18] = 'Small Car'
        name_from_class[19] = 'Bus'
        name_from_class[20] = 'Pickup Truck'
        name_from_class[21] = 'Utility Truck'
        name_from_class[23] = 'Truck'
        name_from_class[24] = 'Cargo Truck'
        name_from_class[25] = 'Truck Tractor w/ Box Trailer'
        name_from_class[26] = 'Truck Tractor'
        name_from_class[27] = 'Trailer'
        name_from_class[28] = 'Truck Tractor w/ Flatbed Trailer'
        name_from_class[29] = 'Truck Tractor w/ Liquid Tank'
        name_from_class[32] = 'Crane Truck'
        name_from_class[33] = 'Railway Vehicle'
        name_from_class[34] = 'Passenger Car'
        name_from_class[35] = 'Cargo/Container Car'
        name_from_class[36] = 'Flat Car'
        name_from_class[37] = 'Tank Car'
        name_from_class[38] = 'Locomotive'
        name_from_class[40] = 'Maritime Vessel'
        name_from_class[41] = 'Motorboat'
        name_from_class[42] = 'Sailboat'
        name_from_class[44] = 'Tugboat'
        name_from_class[45] = 'Barge'
        name_from_class[47] = 'Fishing Vessel'
        name_from_class[49] = 'Ferry'
        name_from_class[50] = 'Yacht'
        name_from_class[51] = 'Container Ship'
        name_from_class[52] = 'Oil Tanker'
        name_from_class[53] = 'Engineering Vehicle'
        name_from_class[54] = 'Tower crane'
        name_from_class[55] = 'Container Crane'
        name_from_class[56] = 'Reach Stacker'
        name_from_class[57] = 'Straddle Carrier'
        name_from_class[59] = 'Mobile Crane'
        name_from_class[60] = 'Dump Truck'
        name_from_class[61] = 'Haul Truck'
        name_from_class[62] = 'Scraper/Tractor'
        name_from_class[63] = 'Front loader/Bulldozer'
        name_from_class[64] = 'Excavator'
        name_from_class[65] = 'Cement Mixer'
        name_from_class[66] = 'Ground Grader'
        name_from_class[71] = 'Hut/Tent'
        name_from_class[72] = 'Shed'
        name_from_class[73] = 'Building'
        name_from_class[74] = 'Aircraft Hangar'
        name_from_class[76] = 'Damaged Building'
        name_from_class[77] = 'Facility'
        name_from_class[79] = 'Construction Site'
```

```
    name_from_class[83] = 'Vehicle Lot'
    name_from_class[84] = 'Helipad'
    name_from_class[86] = 'Storage Tank'
    name_from_class[89] = 'Shipping container lot'
    name_from_class[91] = 'Shipping Container'
    name_from_class[93] = 'Pylon'
    name_from_class[94] = 'Tower'

    source = Image.fromarray(img)
    draw = ImageDraw.Draw(source)
    w2, h2 = (img.shape[0], img.shape[1])

    idx = 0

    for i in range(len(boxes)):
        xmin, ymin, xmax, ymax = boxes[i]
        c = classes[i]
        name = name_from_class[c]
        draw.text((xmin + 15, ymin + 15), str(name))

        for j in range(4):
            draw.rectangle(
                ((xmin + j, ymin + j), (xmax + j, ymax + j)), outline="red")
    return source


def GBDX_to_folium_bounds(sub_bbox):
    """
    Convert GBDX bounding box coordinates to Folium format

    Args:
        sub_bbox : GBDX format bounding box

    Outputs:
        Folium format bounding box

    """
    fol_bound = [[sub_bbox[1], sub_bbox[0]], [[sub_bbox[3], sub_bbox[2]]]]
    return(fol_bound)


def on_button_clicked_stage_two(b):
    """
    Runs second stage of tool when button is clicked

    Args:
        NA
    Outputs:
        Interactive map displaying imagery and analysis
    """
    try:
        # retrieve Lon/Lat from widgets
        user_Lon_deg = user_longitude.value
        user_Lat_deg = user_latitude.value
        conf_thresh_val = conf_thresh.value

        print("Running Stage Two.")
        print(
            "User inputs: Longitude - {}, Latitude - {}, Confidence Theshold - {}".format(
                user_Lon_deg,
                user_Lat_deg,
                conf_thresh_val))

        # return bbox which is 1/7 size of Sentinel-2 sub-bbox
        user_GBDXCatalog_bboxes = user_GBDXCatalog_bbox(
            user_Lon_deg, user_Lat_deg)

        # query catalog for candidate images, and sort so that highest priority
        # image is first
        print("Retrieving list of candidate images")
        sorted_CatalogID = sorted_CatalogID_list(user_GBDXCatalog_bboxes)

        # loop through list until image returned and saved
        print("Candidate images: {}".format(len(sorted_CatalogID)))
        try:
            returned_img, img_path = GBDX_IMG_Request(
                sorted_CatalogID, user_GBDXCatalog_bboxes)
```

```
        except BaseException:
            print("Tool stopped running.")
        # load image saved and perform object detection
        obj_det_img, ob_det_path = run_predictions(output='preditions_txt_{}_{}.txt'.format(
            user_Lon_deg, user_Lat_deg), image_loc=img_path, confidence_threshold=conf_thresh_va
l, checkpoint='models/multires.pb', chip_size=300)

        # Geolocation on Folium map
        bbox_bounds = GBDX_to_folium_bounds(user_GBDXCatalog_bboxes)

        m = folium.Map(
            location=[user_Lat_deg, user_Lon_deg],
            tiles='OpenStreetMap',
            zoom_start=17,
            control_scale=True,
            min_zoom=4,
            max_zoom=21
        )

        obj_ident = folium.FeatureGroup(name='Object Identification')
        folium.raster_layers.ImageOverlay(
            image=np.asarray(obj_det_img),
            bounds=bbox_bounds,
            opacity=1
        ).add_to(obj_ident)

        GBDX_img = folium.FeatureGroup(name='High Resolution Image')
        folium.raster_layers.ImageOverlay(
            image=np.asarray(returned_img),
            bounds=bbox_bounds,
            opacity=1
        ).add_to(GBDX_img)

        GBDX_img.add_to(m)
        obj_ident.add_to(m)

        m.add_child(folium.LatLngPopup())
        folium.LayerControl().add_to(m)

        plugins.Fullscreen(
            position='topright',
            title='Full screen',
            title_cancel='Exit full screen',
            force_separate_button=True).add_to(m)
        m.save(
            "html_obj_det_map_{}_{}.html".format(
                user_Lon_deg,
                user_Lat_deg))
        print("Opening imagery in new tab")
        webbrowser.open(
            'file://' +
            os.path.realpath(
                "html_obj_det_map_{}_{}.html".format(
                    user_Lon_deg,
                    user_Lat_deg)))

    except TypeError:
        print("Please review parameter")
```