

Object Detection using Google Earth

Presented By

SUJAYKUMAR REDDY M
20BDS0294

Submitted to

Submitted to Sahaaya Arul Mary S.A
Faculty of Computer Science Engineering
CSE3001 – Software Engineering
Slot : L11+L12



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Abstract

Object detection using Google Earth is a computer vision technology that utilizes state-of-the-art deep learning models to automatically identify and locate objects of interest within satellite or aerial imagery provided by Google Earth. The underlying algorithmic pipeline typically involves multi-scale feature extraction, object proposal generation, and object classification and localization. This process is often executed on high-performance computing infrastructure, such as GPU clusters, to ensure real-time or near-real-time performance.

The detection and classification of objects within satellite imagery involves several challenges, such as occlusion, varying lighting conditions, and complex background scenes. To overcome these challenges, deep learning models, such as convolutional neural networks (CNNs), are trained on large-scale datasets that contain annotated examples of different object categories. These models are then fine-tuned on specific tasks, such as building detection or road extraction, to improve their accuracy and efficiency.

Object detection using Google Earth has a wide range of practical applications, including urban planning, environmental monitoring, land use mapping, and disaster response. For example, this technology can be used to detect changes in urban infrastructure, such as new road construction or building developments, or to identify areas affected by natural disasters, such as floods or wildfires. Moreover, it can also support conservation efforts by enabling the monitoring of biodiversity and deforestation rates, as well as identifying illegal activities, such as poaching or logging.

The proposed project aims to develop an advanced AI-based software tool capable of accurately detecting and classifying a range of real-world objects in Google Street View and 2D/3D views of Google Earth. The tool will be designed to identify "Generic Objects" such as cars, shops, trees, and other objects of interest, with the ultimate goal of providing relevant information to clients.

To ensure the tool's accuracy and effectiveness, the project will incorporate state-of-the-art computer vision and deep learning techniques. Literature surveys of existing research will be conducted to identify best practices and possible improvements. This will be followed by a milestone plan to guide the project's progress.

The tool's practical applications include urban planning, environmental monitoring, land use mapping, and disaster response. For example, it can assist in monitoring urban infrastructure, detecting changes in land use patterns, identifying areas affected by natural disasters, and even monitoring illegal activities such as logging or poaching.

Table of Contents

1	Introduction	1
1.1	Scope of the Project	1
1.2	Google Earth Imagery	1
1.3	Constraints in Google Earth Imagery	2
1.4	Using ML Algorithms in Google Earth Imagery	2
2	Background Analysis	4
2.1	Object Detection in Google Earth Images Using Deep Convolutional Neural Networks	4
2.2	Automated detection of urban change using Google Earth imagery and machine learning	4
2.3	A Deep Learning Approach to Automatic Building Detection in Google Earth Imagery	5
2.4	Object Detection from Satellite Imagery using Deep Learning Techniques . .	5
3	Algorithms Used	6
3.1	A Segmentation Problem	6
3.2	DataSet	6
3.2.1	Dataset Features	6
3.2.2	Labels	7
3.3	UNet Architecture	7
3.3.1	Useases of UNet Architecture	8
4	Software Requirements Specification	9
4.1	Introduction	10
4.1.1	Purpose	10
4.2	Document Conventions	10
4.3	Intended Audience and Reading Suggestions	10
4.4	Project Scope	11
4.5	Overall Description	12
4.5.1	Product Perspective	12
4.5.2	User Classes and Characteristics	13
4.5.3	Product Functions	13
4.5.4	Operating Environment	13
4.5.5	Design and Implementation Constraints	13

4.6	System Features	14
4.6.1	Functional Requirements	14
4.7	External Interface Requirements	15
4.7.1	Software Interfaces	15
4.7.2	Hardware Interfaces	16
4.8	Other Nonfunctional Requirements	16
4.8.1	Performance Requirements	16
4.8.2	Security Requirements	17
4.8.3	Software Quality Attributes	18
5	Diagrams in Software Engineering	19
5.1	Work Breakdown Structure	19
5.2	Gantt Chart	20
5.3	DataFlow Diagram	20
5.4	ER Diagram	21
5.5	Use Case Diagram	22
5.6	Class Diagram	23
5.7	Sequence Diagram	24
5.8	State Chart Diagram	24
5.9	Communication Diagram	25
5.10	Activity Diagram	26
5.11	Component Diagram	27
5.12	Package Diagram	27
5.13	Deployment Diagram	28
6	Implementation	29
6.1	Google Cloud Platform (GCP)	29
6.1.1	GCP Dashboard	29
6.1.2	GCP Service Accounts	30
6.1.3	GCP Identity and Access Management	30
6.2	Data Analysis and Data Visualization	31
6.3	Pre-Procesing Using Patchify	34
6.4	Training using UNet Architecture	36
6.5	Connection to GCP Cloud using key.json	38
7	Testing the Model	41
7.1	Testing the Image from GCP GE	41
8	Conclusions and Future Work	44
8.1	Conclusion	44
8.2	Future Work	44
	References	45
	APPENDICES	45

A	Python Implementation	46
A.1	Libraries	46
A.2	Code	47
A.2.1	Pre-Processing	47
A.2.2	Training	50
A.2.3	GCP Connection GE API	56

Chapter 1

Introduction

1.1 Scope of the Project

The project scope involves the development of an advanced AI-based software tool for object detection in Google Earth and Google Street View. The tool will be designed to identify and classify various real-world objects, including cars, shops, trees, and other objects of interest. The project will employ state-of-the-art computer vision and deep learning techniques to ensure the tool's accuracy and effectiveness.

The project view is to create a tool that can support various industries and domains, including urban planning, environmental monitoring, land use mapping, and disaster response. For example, the tool can assist in monitoring urban infrastructure, detecting changes in land use patterns, identifying areas affected by natural disasters, and even monitoring illegal activities such as logging or poaching. The tool will also enable clients to specify locations of interest and request data from Google API, which will be processed and provided to the client in a user-friendly format.

Google Earth Imagery can be used for a variety of purposes, such as urban planning, environmental monitoring, disaster response, and tourism. For example, urban planners can use the imagery to analyze land use patterns, identify areas for development, and assess the impact of new construction projects on the surrounding environment. Environmentalists can use the imagery to monitor changes in land cover, track deforestation rates, and identify areas affected by natural disasters such as fires, floods, or landslides.

1.2 Google Earth Imagery

Google Earth Imagery is a collection of high-resolution satellite and aerial images that provide a detailed view of the Earth's surface. This imagery is constantly updated and can be accessed through the Google Earth platform, allowing users to explore and navigate the planet from a bird's-eye perspective.

The images in Google Earth Imagery are typically captured by commercial satellite companies or government agencies, such as NASA or the US Geological Survey. These images are often taken using advanced sensors and cameras, which can capture fine details and subtle variations in the Earth's surface, such as terrain features, vegetation, and man-made

structures.

Google Earth Imagery covers almost the entire planet and includes both rural and urban areas. The imagery is available in different resolutions, ranging from 15 meters per pixel (m/p) for some parts of the world, to as high as 15 cm/p for certain urban areas. The high-resolution imagery allows users to see details such as individual buildings, cars, and even people.

The imagery is also a popular tool for virtual tourism, allowing users to explore famous landmarks and tourist destinations around the world. Users can zoom in on specific locations, tilt and rotate the view, and even explore underwater areas through the use of 3D imagery.

1.3 Constraints in Google Earth Imagery

While Google Earth Imagery offers a wealth of data and information, there are several constraints and challenges that need to be considered. Here are some of the major ones:

1. Resolution: The resolution of Google Earth imagery can vary widely, depending on the location and the type of imagery. In some cases, the resolution may not be high enough to detect small objects or features of interest.
2. Cloud Cover: Cloud cover can obscure important details in Google Earth imagery, making it difficult to obtain accurate information about certain areas.
3. Quality: Google Earth imagery quality can vary depending on the source and age of the imagery. In some cases, the imagery may be outdated or low-quality, which can affect the accuracy of the analysis.
4. Image distortion: Imagery can be distorted due to the terrain and the angle of the camera when the image was captured. This can affect the accuracy of the analysis, especially when attempting to measure distances or identify specific features.
5. Data Availability: Not all areas of the world have high-quality Google Earth imagery available. This can limit the scope of analysis for certain regions or countries.
6. Data privacy: Google Earth imagery may capture sensitive or private information, such as military installations or private property. Careful consideration must be given to privacy concerns when using this data.
7. Computational resources: Processing and analyzing large volumes of Google Earth imagery can be computationally intensive, requiring high-performance computing resources and specialized software tools.

1.4 Using ML Algorithms in Google Earth Imagery

There are several potential use cases for using machine learning (ML) algorithms in Google Earth imagery to detect rooftops.

1. Urban planning: City planners and developers can use ML algorithms to identify rooftops in satellite imagery to better understand the current urban landscape and to plan future developments.
2. Disaster response: In the aftermath of natural disasters, such as earthquakes, floods, and wildfires, ML algorithms can be used to quickly identify damaged buildings and prioritize search and rescue efforts.
3. Energy efficiency: ML algorithms can be used to identify rooftops that are suitable for solar panel installations, which can help to increase energy efficiency and reduce reliance on fossil fuels.
4. Insurance: Insurance companies can use ML algorithms to assess the risk of damage to rooftops from severe weather events, which can help to inform pricing and coverage decisions.
5. Property assessment: Real estate companies and property assessors can use ML algorithms to identify rooftops and assess the value of properties based on factors such as size, condition, and location.

ML algorithms can help to automate the process of rooftop detection in Google Earth imagery, enabling faster and more accurate analysis of the data. This can lead to better decision-making in a variety of industries and domains.

Chapter 2

Background Analysis

2.1 Object Detection in Google Earth Images Using Deep Convolutional Neural Networks

The paper[1] proposes a deep convolutional neural network (CNN) approach for object detection in Google Earth images, particularly for detecting buildings. The paper highlights the limitations of existing methods that rely on manual feature extraction and classification, and proposes that deep learning techniques can improve building detection in Google Earth images. The authors provide details of their methodology, which involves training a CNN model on a large dataset of annotated Google Earth images, and discuss the steps involved in preparing the dataset and training the CNN model. They then evaluate the performance of the model using a test set of Google Earth images and compare it to existing methods. The results show that the proposed method outperforms existing methods in terms of accuracy and scalability, demonstrating the potential of deep learning techniques for object detection in Google Earth imagery.

2.2 Automated detection of urban change using Google Earth imagery and machine learning

The paper [3] provides a comprehensive review of traditional and machine learning-based approaches for detecting urban changes using Google Earth imagery. The authors discuss the advantages and limitations of various techniques, including image differencing, object-based change detection, and supervised and unsupervised machine learning methods. The authors also highlight the potential of deep learning-based methods, such as convolutional neural networks (CNNs), for urban change detection due to their ability to learn complex features from the images.

2.3 A Deep Learning Approach to Automatic Building Detection in Google Earth Imagery

The paper [5] proposed a method on a test dataset of Google Earth images and compared its performance with other state-of-the-art methods. The results showed that their method outperformed other methods in terms of accuracy and processing time. The paper concludes that the proposed method has the potential to be used for various applications, including urban planning, disaster management, and environmental monitoring. The authors suggest that future research could focus on improving the accuracy of the model by incorporating additional data sources, such as LiDAR and multi-spectral imagery, and exploring the use of transfer learning to adapt the model to different geographical regions.

2.4 Object Detection from Satellite Imagery using Deep Learning Techniques

The paper [4] presents a method for object detection from satellite imagery using deep learning techniques. Object detection from satellite imagery faces several challenges such as lighting variations, cloud cover, and variations in object size and orientation. To overcome these challenges, the paper proposes a deep learning model based on the Faster R-CNN architecture, which is trained on a large dataset of satellite imagery. The model uses a convolutional neural network (CNN) to extract features from the input image, followed by region proposal generation and object classification. The performance of the proposed method was evaluated on a publicly available dataset of satellite imagery, and the results showed that the method achieved higher accuracy and faster processing time compared to other methods.

Chapter 3

Algorithms Used

3.1 A Segmentation Problem

Image segmentation is a process of dividing a digital image into multiple segments or regions, each of which corresponds to a distinct object or part of the image. The goal of image segmentation is to simplify the image representation by partitioning it into meaningful regions, which can then be used for analysis, manipulation, or understanding.

There are several methods for image segmentation, including thresholding, edge detection, region growing, clustering, and machine learning-based methods. Thresholding is a simple method that segments an image by setting a threshold value and classifying pixels based on whether they are above or below the threshold. Edge detection methods identify boundaries between regions by detecting sharp changes in image intensity. Region growing methods start from a seed point and iteratively add neighboring pixels that meet certain criteria. Clustering methods group pixels based on their similarity in feature space. Machine learning-based methods use algorithms such as decision trees, random forests, and deep neural networks to learn the mapping between image features and segment labels

3.2 DataSet

Segmented Labels are available in [2] The LandCover.ai (Land Cover from Aerial Imagery) dataset is a dataset for automatic mapping of buildings, woodlands, water and roads from aerial images.

3.2.1 Dataset Features

1. land cover from Poland, Central Europe
2. three spectral bands - RGB
3. 33 orthophotos with 25 cm per pixel resolution (9000x9500 px)
4. 8 orthophotos with 50 cm per pixel resolution (4200x4700 px)
5. total area of 216.27 km²

3.2.2 Labels

1. classes: building (1), woodland (2), water(3), road(4)
2. areas: 1.85 km² of buildings, 72.02 km² of woodlands, 13.15 km² of water, 3.5 km² of roads

3.3 UNet Architecture

The U-Net architecture is a type of neural network that is commonly used for image segmentation tasks. It was specifically designed for biomedical image segmentation, but has also been used for other types of image segmentation.

What makes the U-Net architecture unique is its use of a contracting path and an expansive path. The contracting path is a series of convolutional and pooling layers that reduce the spatial resolution of the image, while also increasing the number of feature channels. This helps the network learn high-level features from the input image.

The expansive path is a series of upsampling and convolutional layers that increase the spatial resolution of the image, while reducing the number of feature channels. This helps the network create a segmentation map that has the same spatial resolution as the input image.

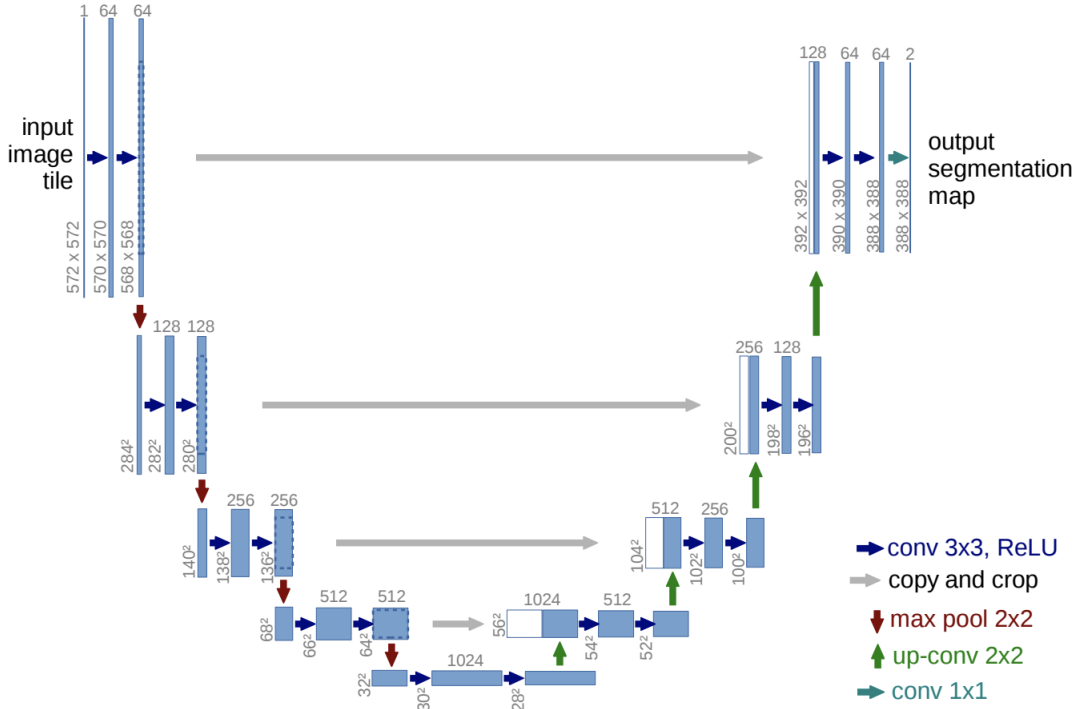


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

3.3.1 Usecases of UNet Architecture

The U-Net architecture is particularly useful for segmentation problems because it can effectively capture both local and global features. The encoder network can capture global features such as context and image-level information, while the decoder network can capture local features such as object boundaries and fine details. The skip connections help propagate information across the network, improving the accuracy of the segmentation. Additionally, the U-Net architecture is computationally efficient and can be trained on relatively small datasets, making it a practical choice for various segmentation tasks.

1. **High Accuracy:** The U-Net model has shown to achieve high accuracy in various image segmentation tasks, including medical image segmentation and object detection in satellite imagery.
2. **Efficient use of data:** The architecture makes efficient use of the available training data by utilizing the skip connections. The skip connections allow the network to combine low-level and high-level features, which helps in reducing the problem of vanishing gradients.
3. **Reduced Overfitting:** The architecture uses data augmentation techniques, such as flipping, rotating, and zooming, which helps in preventing overfitting.
4. **Fast and Easy Training:** The architecture is relatively easy to train, and training times are faster compared to other complex architectures.

Chapter 4

Software Requirements Specification

for

Object Detection using Google Earth

Version 1.2

Prepared by : SujayKumar Reddy M
(20BDS0294)

Sahaaya Arul Mary S.A

4.1 Introduction

4.1.1 Purpose

The usage of **Google Earth** and **Google Street View** is not very habitated by the users. In fact we can do many things for example Detecting Rooftops, Detection of more Heavy Vehicles to reduce the Pollution, in these applications this product is specifically intended for. As when the Vehicle is in transit we cannot track the vehicle in google earth because google earth is a satellite imagery where we cannot watch live changes. According to my research the google earth satellite imagery updates the images for every 1 to 3 years and NASA's Livestat project updates for every 16 days. This application is very much useful for drones to detect the roof-tops and all other Static Objects. My tool will be an interface for the research community for analyzing the vegetation in a particular area within a period of 2-3 years and Rooftop Detection for the drones to deploy a model and other various static applications. My tool will be an interface for all the entities which are available and they can use this to detect the produced Objects and this is the main concept of my model/tool.

4.2 Document Conventions

GEE	Its an API Service by Google named as Google Earth Engine.
GSV	Google Street View is an API Service Provided by the GEE and it mainly concentrates on Street Photography useful for AR/VR Apps
Static Objects	The Object which cannot be changed over 2-3 years of time like Buildings, farms and Vegetation within a particular reach.
Tffite	An Tensorflow Model which is used to Generate the Supervised Learning Approach for the Static Object Detection

4.3 Intended Audience and Reading Suggestions

This Software Requirement Specification is for Myself for the future reference, Professor and testers. This SRS is done according to the template given by the Professor. Issues List is provided at the end of the document at Appendix C. Further, the discussion will provide all the internal, external, functional, and also non-functional information about "Object Detection from Google Earth Images".

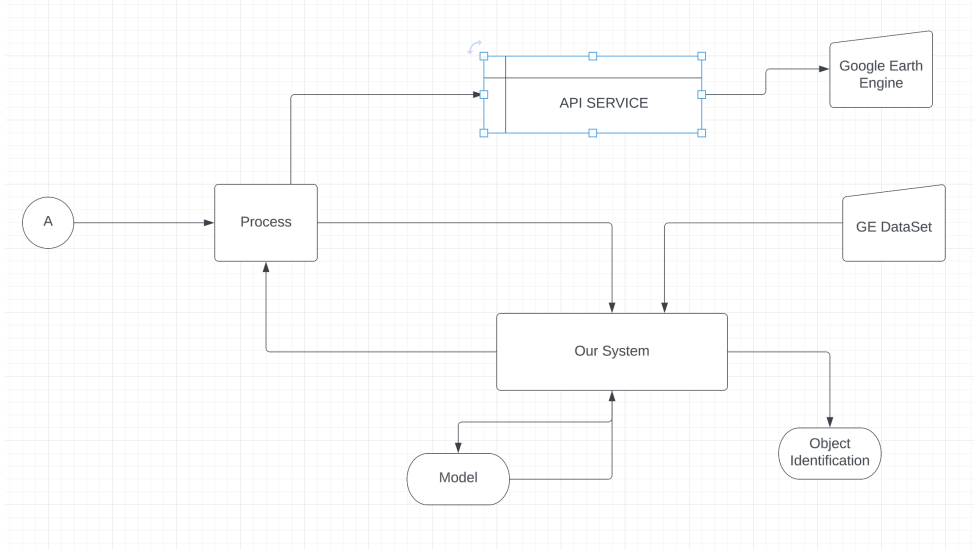


Figure 4.1: Entire work-flow

4.4 Project Scope

The Major Stakeholders are the ...

1. User
2. GEE
3. Object Identification
4. Time
5. Tracking

GEE will take care with the communication of the satellites and takes in the data of satellite image processing. We will only consider that processed image as the input to our model or tool. The GEE API Service will give us an API Services where we can navigate for a zoomed in image to process the image carefully. The Service would be connected with our System which will detect the Objects and track the motion of the detected object.

Object Identification is an Machine Learning Model which detects the objects by Open CV2 it marks the object and classifies as an provided class label.

The User communicates with the GEE Service through our interface and gives us the starting co-ordinates and starting image to identify.

The Time is important because we are considering the static imagery of the Google Earth Engine. We need the images which are useful for real time data by using various sensors but the dataset which we use to train our data.

Figure 1.1 (Entire work-flow) is the overview of the project. Connection of all the entities are dependable to each others. This gives the simple idea about the functional activities of the project.

Google Earth Engine will be a static input for our web application because the model which we trained for.

So, every entity is vary much interactive with each other.

The project scope for Object Detection from Google Earth Images would typically include the following:

1. **Object Detection Algorithm:** Develop a computer vision algorithm to detect objects in Google Earth images using techniques such as convolutional neural networks (CNNs), region-based convolutional neural networks (R-CNNs), and You Only Look Once (YOLO) algorithms.
2. **Image Dataset:** Create or obtain a dataset of Google Earth images for training and testing the object detection algorithm. The dataset should include a variety of different objects in different environments and under different conditions.
3. **Model Training:** Train the object detection algorithm using the image dataset. This will involve adjusting the parameters and architecture of the model until it can accurately detect objects in the images.
4. **Model Deployment:** Integrate the trained object detection model into an application that can be used to process Google Earth images and detect objects within them.
5. **User Interface:** Develop a user interface that allows users to select an image from Google Earth, process it with the object detection model, and view the results. The interface should be intuitive and easy to use.
6. **Performance Evaluation:** Evaluate the performance of the object detection model by comparing its results to a ground truth dataset. This will involve measuring metrics such as precision, recall, and accuracy.
7. **Optimization:** Optimize the performance of the object detection model by making changes to the algorithm, dataset, or other aspects of the system as necessary.

4.5 Overall Description

4.5.1 Product Perspective

Google earth is a large system and My Tool integrates into this particular API Services for providing various analysis of geo-spatial data. Main goal of this project is to minimize the workflow of using drones to calculate the static data instead we can use our tool to Identify the static objects for data analysis.

4.5.2 User Classes and Characteristics

This Object Detection has basically 2 types of users.

- Researchers
- Business Entities

The Researchers working on SOLAR Energy they have the very useful application to know how much area does the each village has so we can implement it by comparatively.

The Business Entities like Amazon can use the models to detect the rooftops to deliver the packages for per user.

4.5.3 Product Functions

The Product should be able to detect the rooftops for the drones in-order to land the package for asset management services. If the Object is the Farming Lands then the Drone would find the optimal point where all the agriculture sensors would be able to communicate within the distance.

Before using the main function of the software result process, users have to be registered. The Products primary source is the Machine Learning model. Result is the main feature of all. It contains the Object which is marked by the Open CV2.

4.5.4 Operating Environment

The website will be operate in any Operating Environment - Mac, Windows, Linux etc.

4.5.5 Design and Implementation Constraints

As the ML model is the primary source we consider that as a Design.

- From the Unprocessed Image from the GE.
- Supervised Learning Techniques.
- Class Labels and Unsupervised Learning Techniques.

From the GE Image we need to detect the static objects and find a suitable environment with color grading which would provide us an Undefined colored so, it acn be easily seen by the Sensors or Microcontrollers in the Drones.

Every Image would happen to work with utmost accuracy to detect the static images.

Design and implementation constraints for object detection from Google Earth can vary depending on the specific use case and requirements. However, some common constraints and considerations are:

- **Image Quality:** The quality and resolution of the satellite images in Google Earth can have a significant impact on the accuracy and reliability of object detection. Poor quality images with low resolution, or images that are obstructed by clouds or other factors, can make it difficult to accurately detect objects.
- **Object Variability:** The variability of objects in satellite images can also impact the accuracy and reliability of object detection. Objects may vary in size, shape, orientation, and appearance, which can make it challenging to train object detection algorithms to recognize them.
- **Computational Resources:** Object detection can be computationally intensive, and large amounts of data need to be processed in real-time to ensure fast and accurate results. It is important to carefully consider the computational resources required to run the object detection algorithm, as well as the hardware and software infrastructure required to support it.
- **Data Privacy and Security:** Google Earth images may contain sensitive information that needs to be protected, and it is important to ensure that the object detection algorithm is designed and implemented in a way that protects this information.
- **Data Storage and Management:** The amount of data generated by object detection from Google Earth images can be large, and it is important to have a robust and scalable data storage and management system in place to handle this data.

These are some of the key design and implementation constraints for object detection from Google Earth. However, there may be additional constraints and considerations that are specific to your use case and requirements.

4.6 System Features

4.6.1 Functional Requirements

1. The object detection software should be able to process images from Google Earth and extract relevant information for object detection. This may involve implementing algorithms for image pre-processing, such as image resizing, and image normalization.
2. The object detection software should be able to detect objects of interest in the images, such as buildings, vehicles, and roadways. This may involve implementing object detection algorithms, such as deep learning-based object detection, and training the algorithms on annotated image data.
3. The object detection software should be able to classify the objects detected in the images into relevant categories, such as type of building, type of vehicle, or type of roadway. This may involve implementing object classification algorithms and training the algorithms on annotated image data.

4. The object detection software should be able to locate the objects in the images and provide the coordinates of the objects. This may involve implementing object localization algorithms, such as bounding box regression, and training the algorithms on annotated image data.
5. The object detection software should be able to generate outputs, such as annotated images or data reports, that provide information about the objects detected in the images. This may involve implementing output generation algorithms and user interfaces for visualizing the outputs.
6. The object detection software should provide user interfaces for interacting with the software, such as uploading images, configuring the object detection algorithms, and viewing the outputs. This may involve implementing graphical user interfaces, command line interfaces, or application programming interfaces (APIs) for integrating the object detection software with other software systems.

4.7 External Interface Requirements

TensorFlow Lite (TFLite) is a lightweight version of TensorFlow, designed to run on resource-constrained devices such as mobile phones and embedded systems. The TFLite object detection model can be integrated into various external user interfaces to provide object detection capabilities to end users. Some common external user interfaces for TFLite object detection models are:

1. Mobile Applications
2. Web Applications
3. Embedded Systems.
4. Robotics

These are some of the external user interfaces that TFLite object detection models can be integrated with. The specific user interface that you choose will depend on your use case and requirements. It is important to carefully consider the requirements of your application and the resources available on your target device when selecting an external user interface for your TFLite object detection model.

4.7.1 Software Interfaces

1. Mobile SDKs: TFLite object detection models can be integrated into mobile software development kits (SDKs), allowing developers to build mobile applications that perform object detection on mobile devices.
2. Web API: TFLite object detection models can be deployed as web APIs, allowing object detection to be performed over the internet. This can be useful for building cloud-based object detection systems that can be accessed by a variety of clients, such as mobile applications, web applications, and desktop applications.

3. Robotics API: TFLite object detection models can also be integrated into robotics APIs to provide object detection capabilities for autonomous robots.
4. Embedded API: TFLite object detection models can be integrated into embedded APIs, allowing developers to build custom embedded systems that perform object detection.

4.7.2 Hardware Interfaces

The Google Earth API can be integrated with various hardware interfaces to provide mapping and visualization capabilities to end users. Some common hardware interfaces for the Google Earth API are:

1. Web Browsers: The Google Earth API can be integrated into web browsers to provide interactive mapping and visualization capabilities to end users. This allows users to access Google Earth in their web browsers and interact with maps and satellite imagery.
2. Embedded Systems: The Google Earth API can be integrated into embedded systems, such as GPS devices and in-vehicle navigation systems, to provide mapping and visualization capabilities in these environments.
3. Robotics Hardware: The Google Earth API can also be integrated into robotics hardware, such as autonomous robots, to provide mapping and navigation capabilities for these systems.

These are some of the hardware interfaces that the Google Earth API can be integrated with. The specific hardware interface that you choose will depend on your use case and requirements. It is important to carefully consider the requirements of your application and the resources available on your target device when selecting a hardware interface for the Google Earth API.

4.8 Other Nonfunctional Requirements

4.8.1 Performance Requirements

The performance requirements for object detection from Google Earth images will depend on a number of factors, including the size of the images, the complexity of the objects being detected, the desired accuracy and speed of the object detection process, and the resources available on the hardware platform where the object detection is being performed. Some of the key performance requirements for object detection from Google Earth images include:

1. Computational Power: Object detection algorithms can be computationally intensive, and therefore a powerful processing platform is necessary to perform object detection in real-time. This may involve using high-performance CPUs, GPUs, or other specialized hardware such as Tensor Processing Units (TPUs).

2. **Memory:** Object detection algorithms require large amounts of memory to store the image data and intermediate results of the object detection process. Therefore, the hardware platform should have sufficient memory to support the object detection process.
3. **Network Bandwidth:** If the object detection is being performed in a cloud environment, the network bandwidth between the cloud and the client device may also be a factor in the performance of the object detection process.
4. **Latency:** The latency of the object detection process is an important factor in the overall performance of the system. Latency refers to the time it takes for an object detection request to be processed and the results to be returned to the client. Low latency is important for real-time applications where the results of the object detection process need to be available immediately.
5. **Accuracy:** The accuracy of the object detection process is also a critical factor in the overall performance of the system. High accuracy is important for applications where the object detection results are used to make important decisions, such as in autonomous vehicles or surveillance systems.

These are some of the key performance requirements for object detection from Google Earth images. The specific performance requirements for your application will depend on the details of your use case and the desired outcomes. It is important to carefully consider the performance requirements of your application and the resources available on your hardware platform when designing and implementing an object detection system from Google Earth images.

4.8.2 Security Requirements

Security is an important consideration in any application that involves processing and storing sensitive information, including object detection from Google Earth images. Some of the key security requirements for object detection from Google Earth images include:

1. **Data Privacy:** The privacy of the image data being processed and stored must be protected. This may involve encrypting the image data and ensuring that it is only accessible by authorized individuals.
2. **Data Integrity:** The integrity of the image data must be protected to prevent unauthorized changes to the data. This may involve implementing data integrity checks and audit trails to detect any unauthorized changes to the data.
3. **Access Control:** Access to the image data and the object detection results must be controlled to ensure that only authorized individuals have access to the data. This may involve implementing user authentication and authorization mechanisms, such as usernames and passwords, to control access to the data.

4. Network Security: The network over which the image data is transmitted and the object detection results are returned must be secure to prevent unauthorized access to the data. This may involve implementing encryption algorithms, such as SSL/TLS, to secure the data transmission.
5. Physical Security: The hardware platform where the object detection is performed must be physically secured to prevent unauthorized access to the image data and the object detection results. This may involve implementing physical security measures, such as security cameras, access controls, and secure data storage facilities.

These are some of the key security requirements for object detection from Google Earth images. The specific security requirements for your application will depend on the sensitivity of the image data and the desired level of security. It is important to carefully consider the security requirements of your application and to implement appropriate security measures to protect the image data and the object detection results.

4.8.3 Software Quality Attributes

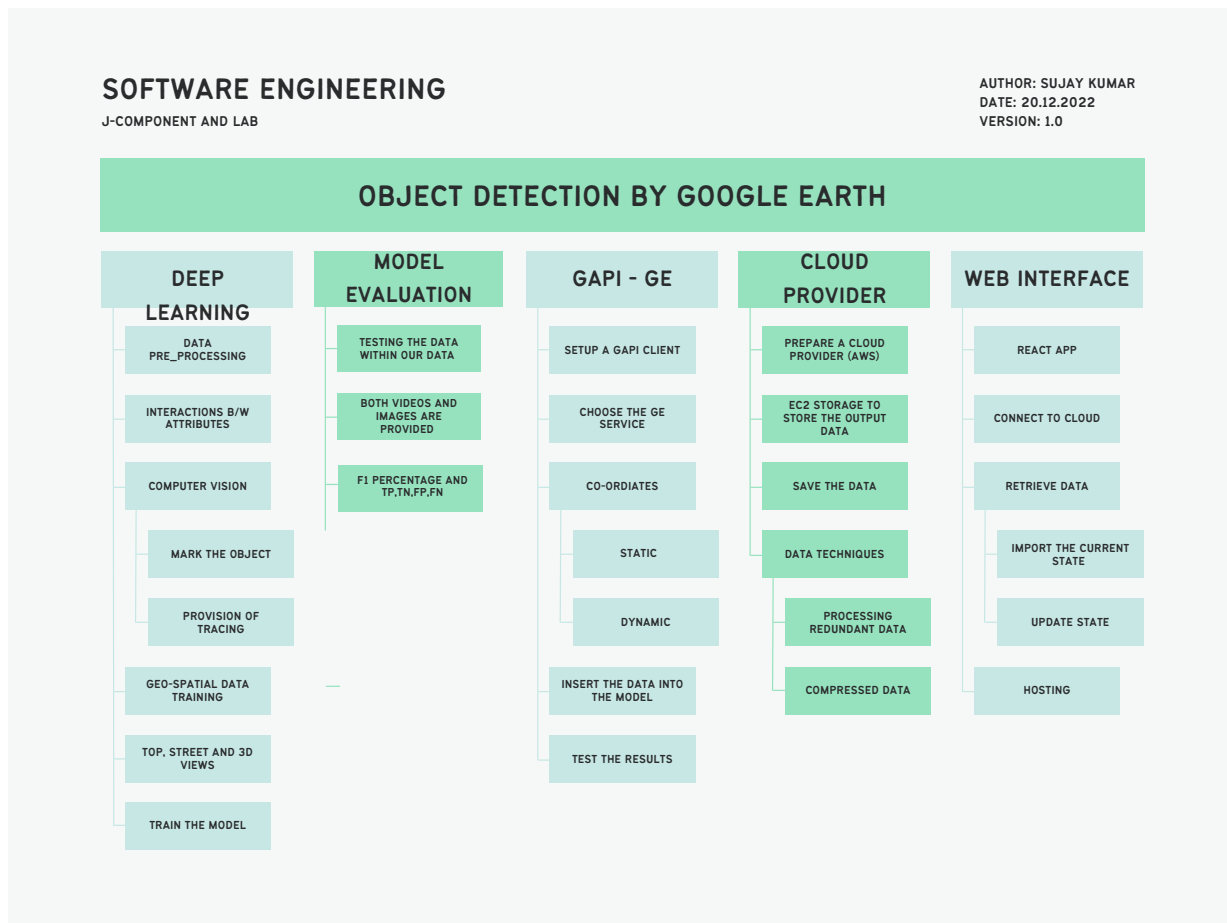
Software quality attributes are characteristics of software that are important to ensure that the software is fit for its intended use. In the context of object detection from Google Earth images, some of the key software quality attributes include:

1. Usability: The object detection software should be easy to use and understand for the end-users. This may involve providing clear and concise user interfaces, user-friendly error messages, and intuitive navigation.
2. Reliability: The object detection software should be reliable and should perform its intended functions accurately and consistently. This may involve implementing robust error handling mechanisms, redundant components, and regular software testing.
3. Performance: The object detection software should perform efficiently and quickly, especially for real-time applications. This may involve optimizing the algorithms for the hardware platform, reducing latency, and improving computational performance.
4. Scalability: The object detection software should be able to handle increasing volumes of image data and users without degradation in performance. This may involve implementing scalable algorithms and architectures, load balancing, and distributed computing.
5. Maintainability: The object detection software should be maintainable, so that it can be updated and improved over time. This may involve writing clear and well-documented code, using modular design patterns, and following software development best practices.

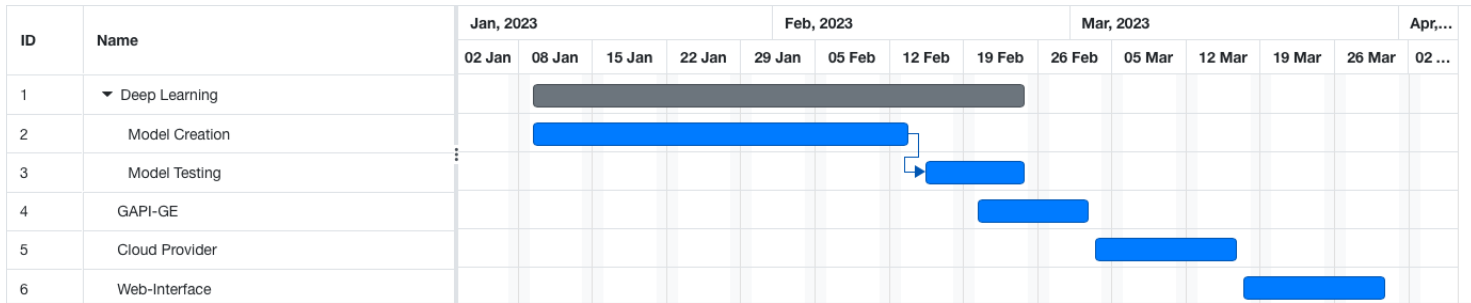
Chapter 5

Diagrams in Software Engineering

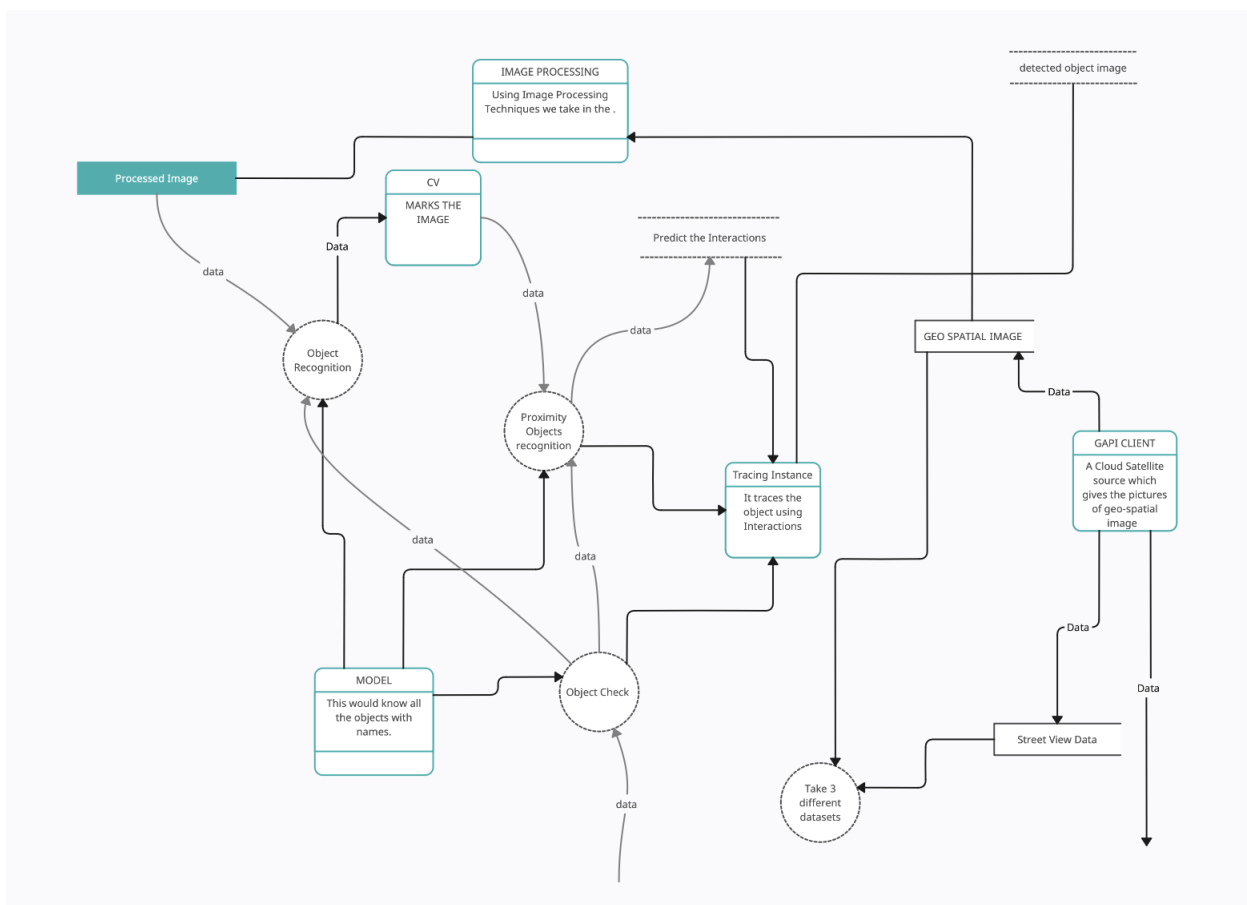
5.1 Work Breakdown Structure



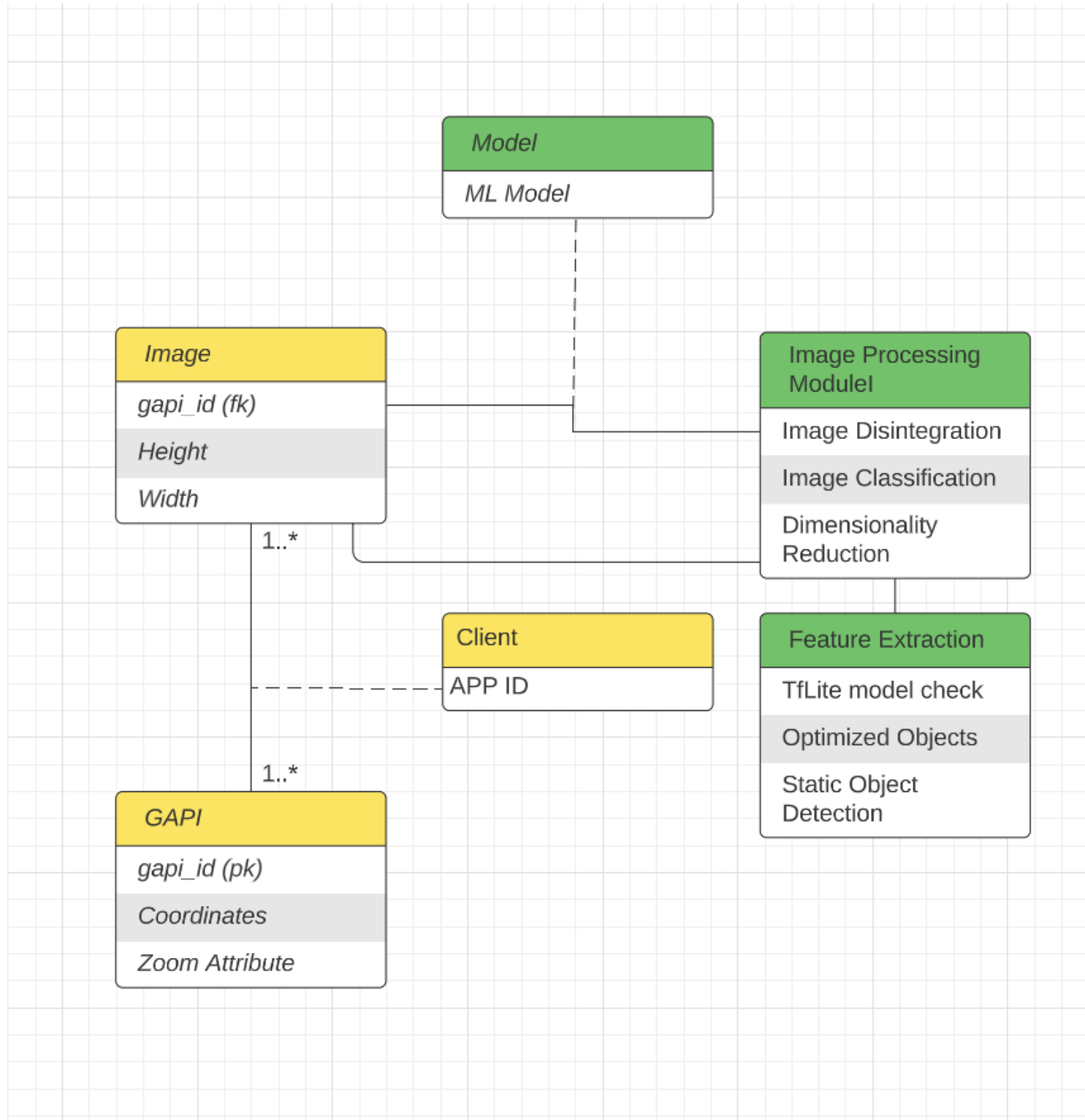
5.2 Gantt Chart



5.3 DataFlow Diagram



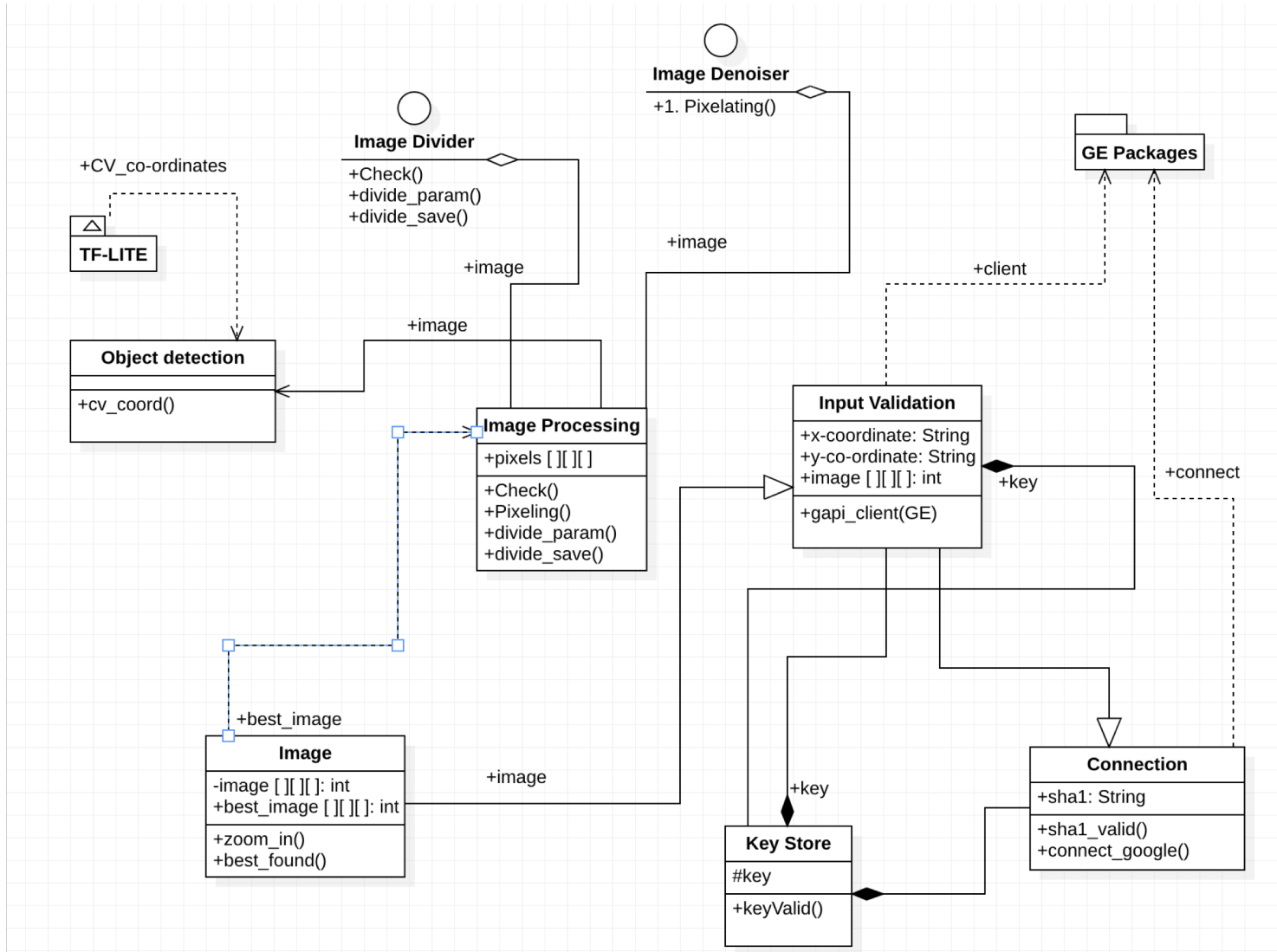
5.4 ER Diagram



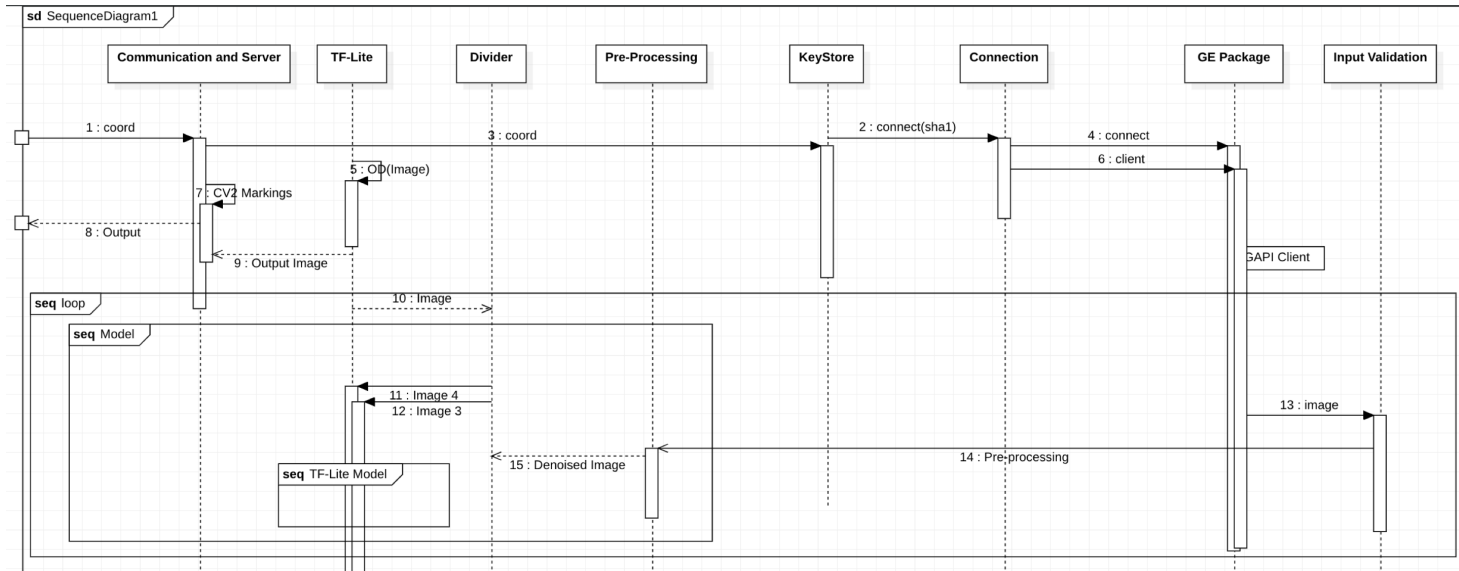
```
graph TD
    subgraph "OBJECT DETECTION"
        direction TB
        ID([Image Divider])
        IDN([Image Denoiser])
        OP([Object Detection])
        IP([Image Processing])
        CV2([CV2])
        IV([Input Validation])
        C([Connection])
        GE([GE Packages])
        KS([KeyStore])
        ID -.->|«extend»| IP
        IDN -->|+Processed Image| IP
        IP -->|+Image| OP
        OP -.->|+Detected Image| IDN
        OP -->|+Image| CV2
        OP -->|+Image| C
        C -->|+conn_sha| IV
        IV -->|+co-ord| customer
        C -.->|«extend»| GE
        C -.->|«extend»| KS
    end
    customer((customer))
    admin((admin))
    GAPI((GAPI))
    customer --- ID
    admin --- IDN
    GAPI --- C
```

The diagram illustrates the architecture of the OBJECT DETECTION system. It features a central system boundary containing several use cases and their interactions. The use cases are represented by ovals: Image Divider, Image Denoiser, Object Detection, Image Processing, CV2, Input Validation, Connection, GE Packages, and KeyStore. The actors are represented by stick figures: customer, admin, and GAPI. The interactions are as follows: The customer actor is connected to the Image Divider use case. The admin actor is connected to the Image Denoiser use case. The GAPI actor is connected to the Connection use case. Within the system boundary, the Image Divider use case has an «extend» relationship with the Image Processing use case. The Image Denoiser use case provides a «+Processed Image» output to the Image Processing use case. The Image Processing use case provides a «+Image» output to the Object Detection use case. The Object Detection use case provides a «+Detected Image» output to the Image Denoiser use case. The Object Detection use case also provides a «+Image» output to the CV2 use case and the Connection use case. The Connection use case provides a «+conn_sha» output to the Input Validation use case. The Input Validation use case is connected to the customer actor and provides a «+co-ord» output. The Connection use case has «extend» relationships with the GE Packages and KeyStore use cases.

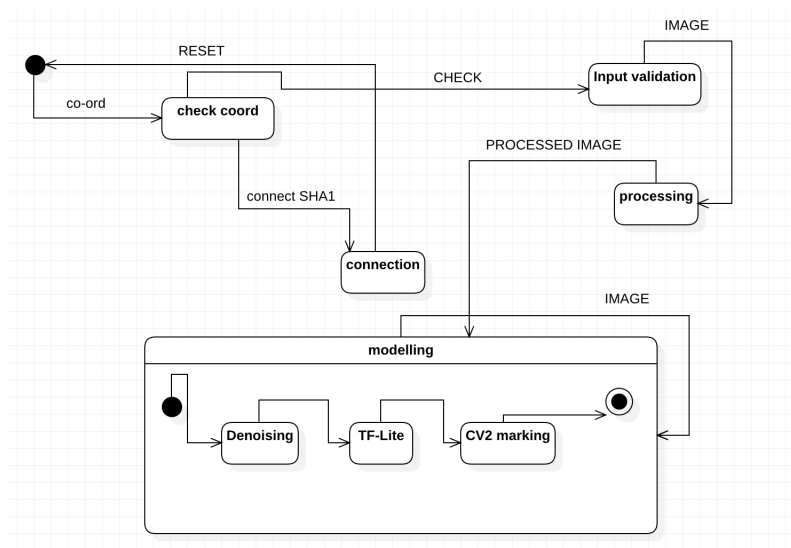
5.6 Class Diagram



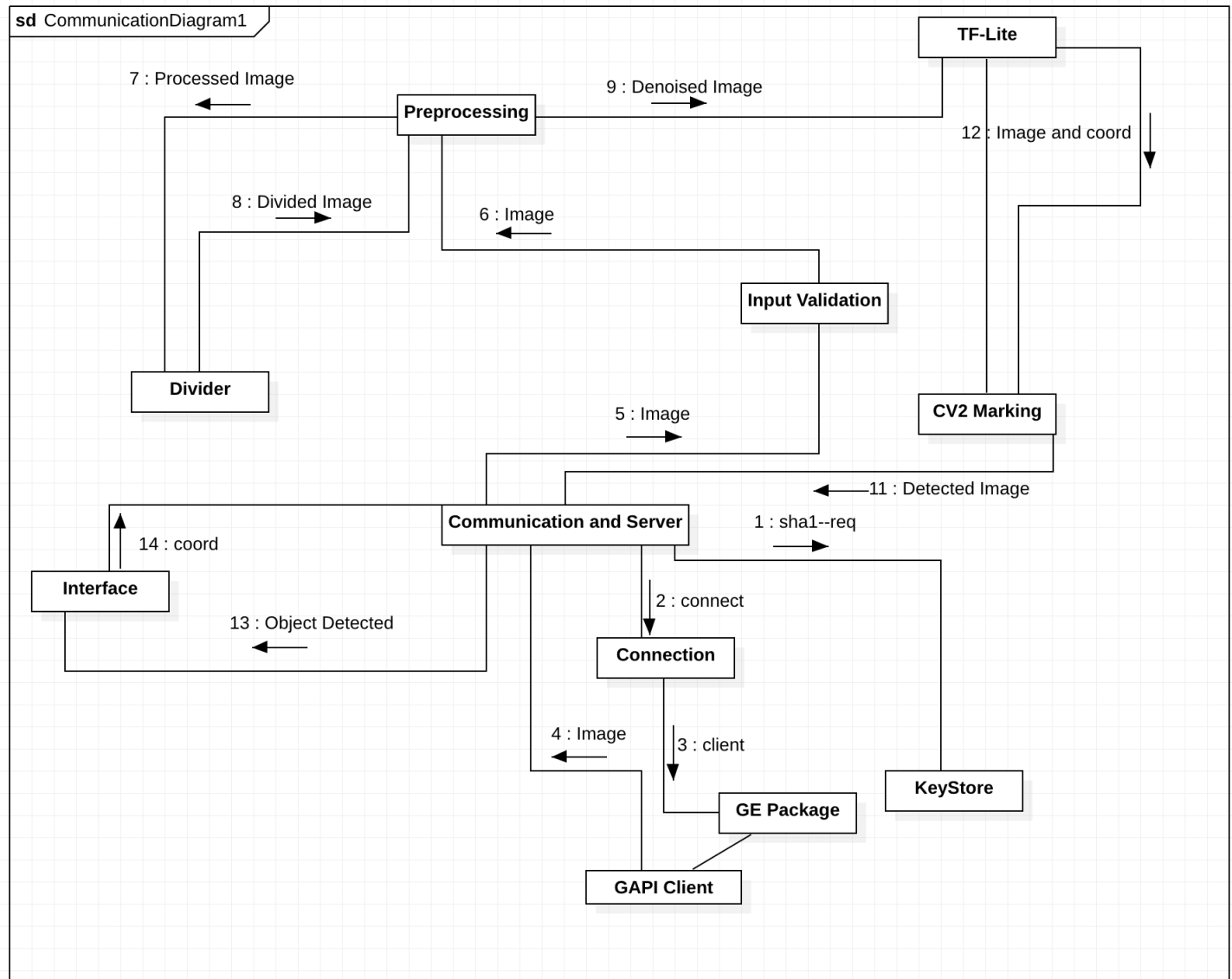
5.7 Sequence Diagram



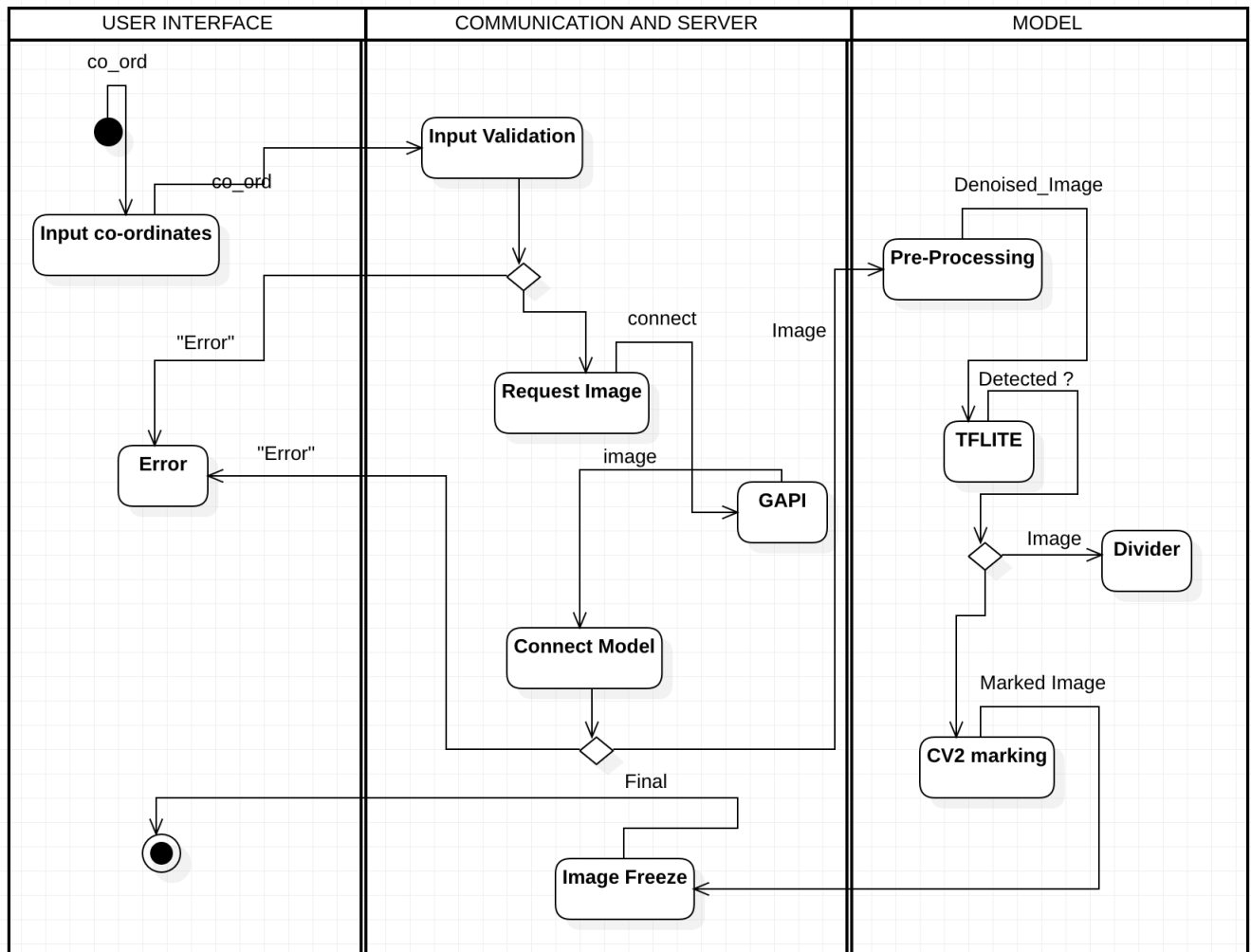
5.8 State Chart Diagram



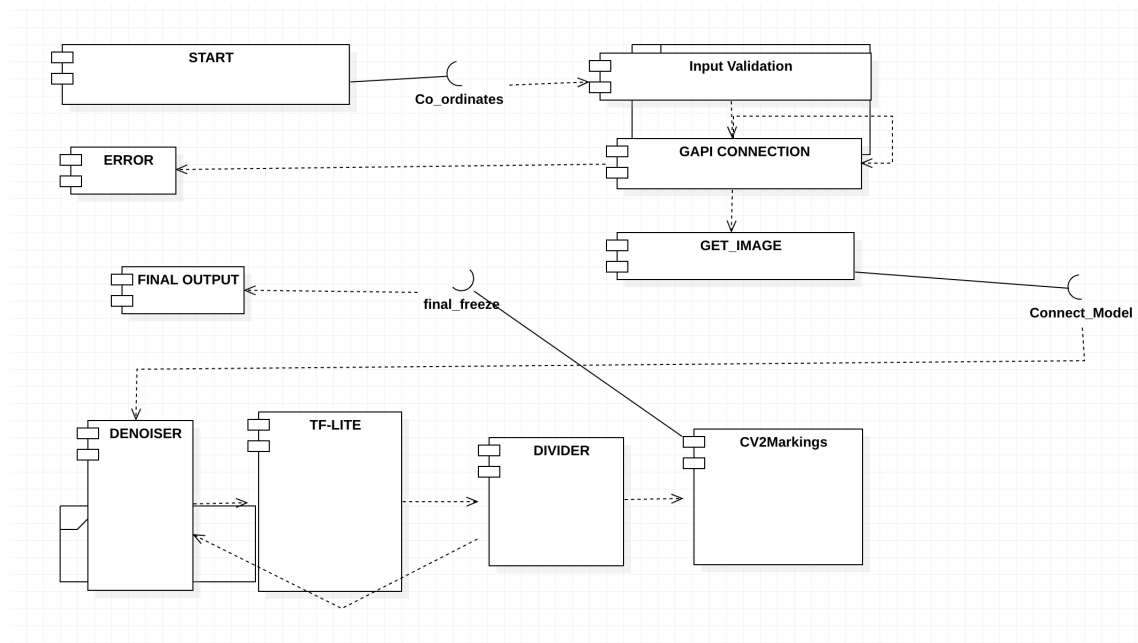
5.9 Communication Diagram



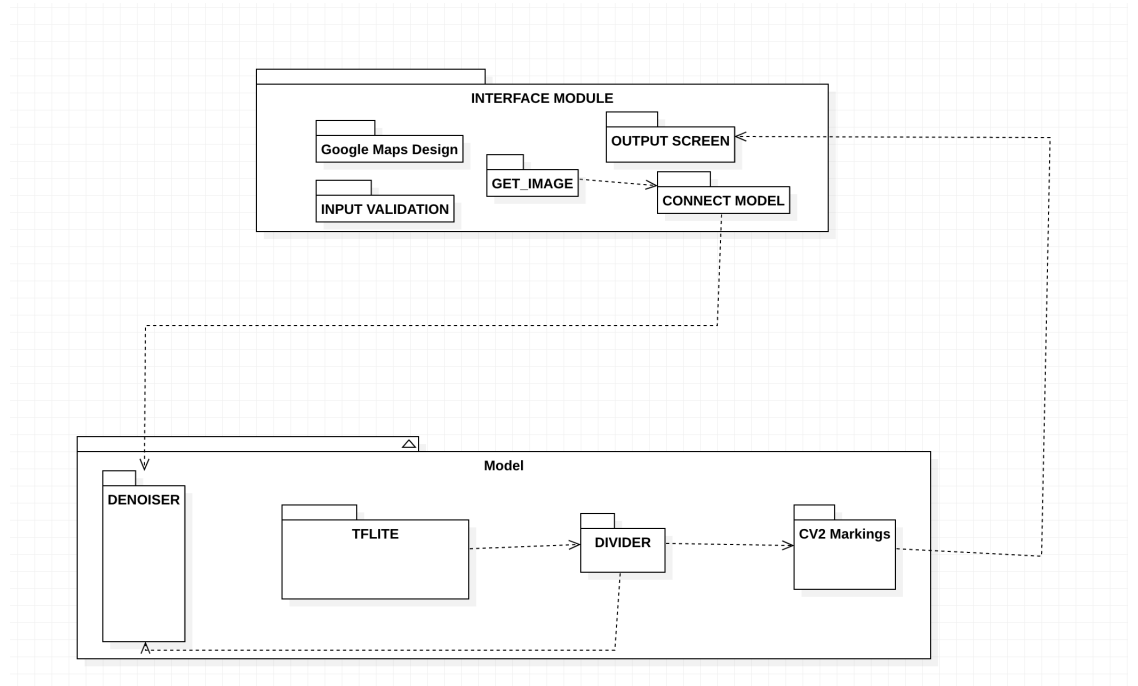
5.10 Activity Diagram



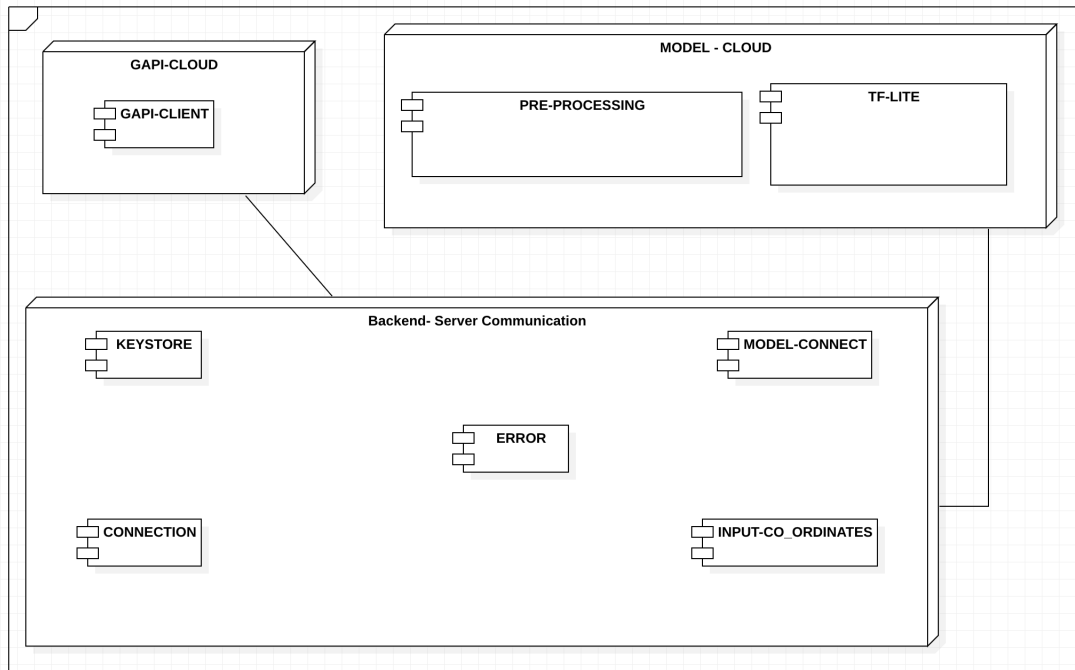
5.11 Component Diagram



5.12 Package Diagram



5.13 Deployment Diagram

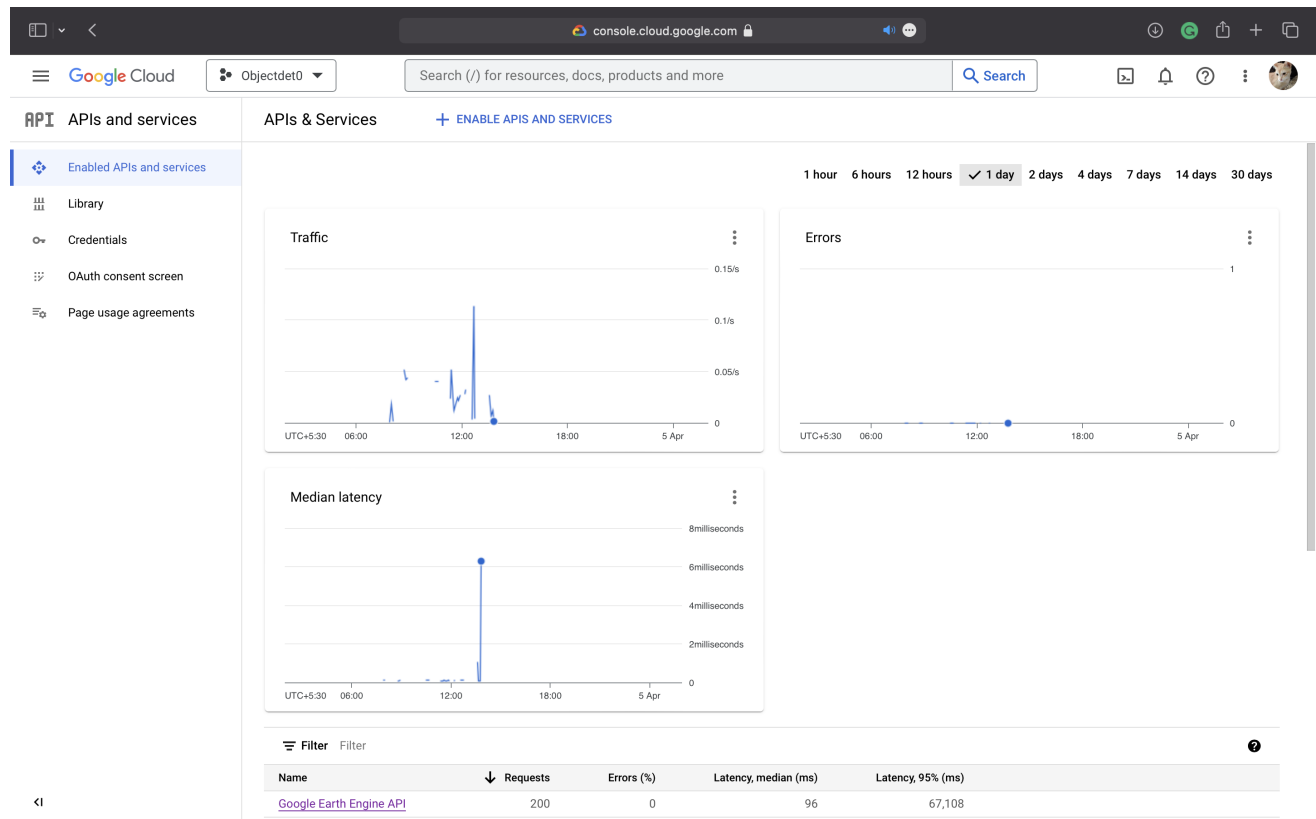


Chapter 6

Implementation

6.1 Google Cloud Platform (GCP)

6.1.1 GCP Dashboard



6.1.2 GCP Service Accounts

The screenshot shows the Google Cloud Console interface for project Objectdet0. The left sidebar contains the navigation menu with 'APIs and services' selected. The main content area is titled 'Credentials' and includes links for '+ CREATE CREDENTIALS', 'DELETE', and 'RESTORE DELETED CREDENTIALS'. Below this, there is a section for 'API keys' which is currently empty. The 'OAuth 2.0 Client IDs' section shows one client ID: 'EE Notebook - suzzay19@gmail.com' with a creation date of '4 Apr 2023' and a type of 'Desktop'. The 'Service Accounts' section shows two service accounts: 'sujay-164@objectdet0.iam.gserviceaccount.com' and 'sujay-gcp@objectdet0.iam.gserviceaccount.com'.

Name	Creation date	Restrictions	Actions
No API keys to display			

Name	Creation date	Type	Client ID	Actions
EE Notebook - suzzay19@gmail.com	4 Apr 2023	Desktop	450664548293-8u78...	[Edit] [Delete] [Download]

Email	Name	Actions
sujay-164@objectdet0.iam.gserviceaccount.com	sujay	[Edit] [Delete]
sujay-gcp@objectdet0.iam.gserviceaccount.com	sujay_gcp	[Edit] [Delete]

6.1.3 GCP Identity and Access Management

The screenshot shows the Google Cloud Console interface for project Objectdet0, specifically the 'IAM and admin' section. The left sidebar contains the navigation menu with 'IAM and admin' selected. The main content area is titled 'IAM' and includes links for '+ GRANT ACCESS' and 'REMOVE ACCESS'. Below this, there is a section for 'Permissions for project Objectdet0' which includes a filter bar and a table of permissions. The table shows three principals: 'sujay-164@objectdet0.iam.gserviceaccount.com' (Owner), 'sujay-gcp@objectdet0.iam.gserviceaccount.com' (Owner), and 'suzzay19@gmail.com' (Owner).

Type	Principal	Name	Role	Security insights	Inheritance
[Service Account]	sujay-164@objectdet0.iam.gserviceaccount.com	sujay	Owner		[Edit]
[Service Account]	sujay-gcp@objectdet0.iam.gserviceaccount.com	sujay_gcp	Owner		[Edit]
[User]	suzzay19@gmail.com	Sujay Kumar	Owner		[Edit]

6.2 Data Analysis and Data Visualization

```
1 // Determine 10-year mean LSTs
2
3 /**** Start of imports. If edited, may not auto-convert in the playground.
4 ****/
5 var VisPar = {"opacity":1,"bands":["constant"],"min":-5,"max":40,"palette"
6 :["002bff","00ff36","fbff00","ff0000"]};
7 /***** End of imports. If edited, may not auto-convert in the playground.
8 *****/
9 //variables
10 var numyears=10; //number of years
11 var firstyear=2005; //first year (beast after 2002)
12
13 //function oneyear mean
14 var oneyearmean=function(MYD,MOD){
15   var MYDm = MYD.mean();
16   var MODm = MOD.mean();
17   var LST_do=MODm.expression( //im transferring to celcius here so I don't
18     have to worry about different band names
19     '(0.02*LST - 273.15)',{
20       'LST' : MODm.select('LST_Day_1km')
21     });
22   var LST_no=MODm.expression(
23     '(0.02*LST - 273.15)',{
24       'LST' : MODm.select('LST_Night_1km')
25     });
26   var LST_dy=MYDm.expression(
27     '(0.02*LST - 273.15)',{
28       'LST' : MYDm.select('LST_Day_1km')
29     });
30   var LST_ny=MYDm.expression(
31     '(0.02*LST - 273.15)',{
32       'LST' : MYDm.select('LST_Night_1km')
33     });
34   var LST = ee.ImageCollection([LST_dy,LST_ny,LST_do,LST_no]).mean();
35   //going with image collection to get the mean, cause than i don't have to
36   //figure out how to work with the 'no value' pixels
37   return LST;
38 };
39
40 //function 10 year mean month
41 var meanmonth=function(from,to,month){
42   var LST2=ee.List([]); //dummy to fill with LSTs
43
44   for (var year = firstyear; year < firstyear+numyears; year++) {
45     var quarter_from = ee.Date(year.toString() +from);
46     var quarter_to = ee.Date(year.toString() +to);
47     if (month == 12) {
48       var yy=year+1;
49       quarter_to = ee.Date(yy.toString() +to);
50     }
51     var MYD = ee.ImageCollection('MODIS/MYD11A1')
```



```

47 .filterDate(quarter_from, quarter_to);
48 var MOD = ee.ImageCollection('MODIS/MOD11A1')
49 .filterDate(quarter_from, quarter_to);
50 LST=oneyearmean(MYD,MOD);
51 LST2=LST2.add(LST);
52 }
53 var LST3=ee.ImageCollection(LST2);
54 return LST3.mean();
55 };
56
57 //start main code
58 var LST= ee.Image.constant(0); //dummy to fill with LSTs
59 for (var month=1;month<13;month++){
60     var m =month+1;
61     var daypermonth=31;
62     if (month == 2){
63         daypermonth=28;
64     } else if (month == 4 | month == 6 | month == 9 | month == 11) {
65         daypermonth=30;
66     }
67     var from=('-' +month.toString()+'-01');
68     var to=('-' +m.toString()+'-01');
69     if (month == 12) {
70         var to=('01-01');
71     }
72     LST=LST.add(meanmonth(from,to,month).multiply(ee.Image.constant(
        daypermonth))));
73 }
74 //print('LST',LST);
75 LST=LST.expression(
76     '(LST/365)',{
77     'LST' : LST
78 });
79 Map.setCenter(0, 0, 1);
80 Map.addLayer(LST,VisPar,'LST');
81 Export.image(LST, 'LST', {
82     scale: 1000,
83     maxPixels: 1e10
84 });

```



6.3 Pre-Processing Using Patchify

Preprocess_1.ipynb

File Edit View Insert Runtime Tools Help Last saved at April 4

Comment Share

Connect

+ Code + Text

pip install patchify split-folders

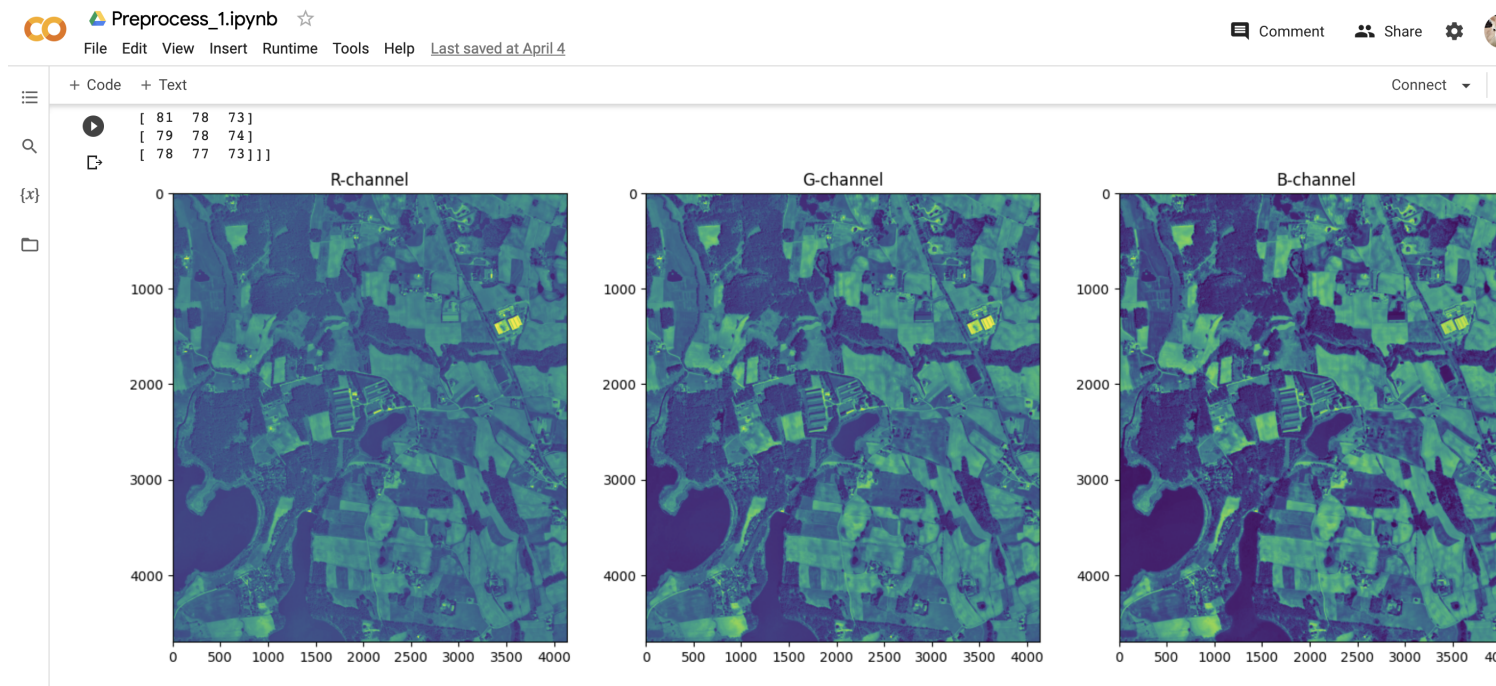
Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>
Collecting patchify
 Downloading patchify-0.2.3-py3-none-any.whl (6.6 kB)
Collecting split-folders
 Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.9/dist-packages (from patchify) (1.22.4)
Installing collected packages: split-folders, patchify
Successfully installed patchify-0.2.3 split-folders-0.5.1

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from PIL import Image
from patchify import patchify
import splitfolders
import random
from keras.utils import to_categorical
```

```
[ ] from google.colab import drive
drive.mount('/content/drive')
img=cv2.imread("/content/drive/My Drive/Objectdet0/fyp/landcover.ai.v1/images/N-33-60-D-c-4-2.tif")
plt.figure(figsize=(18,10))
plt.subplot(131)
plt.title("R-channel")
print(img)
plt.imshow(img[:, :, 0])
plt.subplot(132)

plt.title("G-channel")
plt.imshow(img[:, :, 1])
plt.subplot(133)
plt.title("B-channel")
```

Display a menu





6.4 Training using UNet Architecture

```
Train_1.ipynb ☆
File Edit View Insert Runtime Tools Help Last saved at April 4

[ ] # make a new folder to save model
try:
    os.makedirs(root_dir+"models")
except:
    print("Directory already available, so not created")

Directory already available, so not created

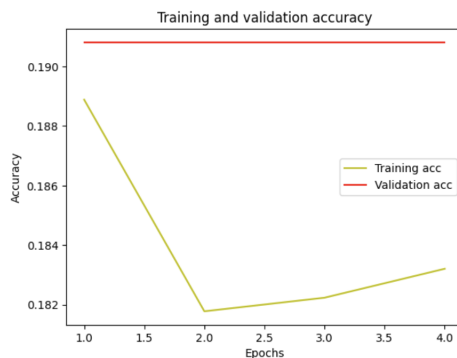
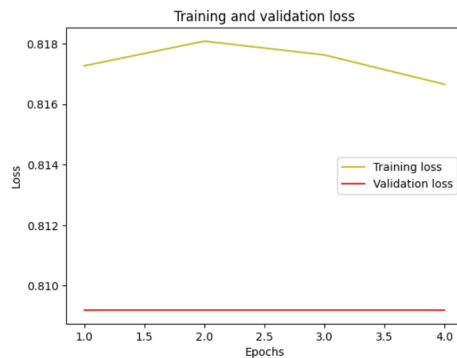
▶ history = model.fit(
    X,Y,
    steps_per_epoch=steps_per_epoch,
    epochs=20,
    verbose=1,
    callbacks=[earlystopping],
    validation_data=(X_test, Y_test),
    validation_steps=val_steps_per_epoch
)

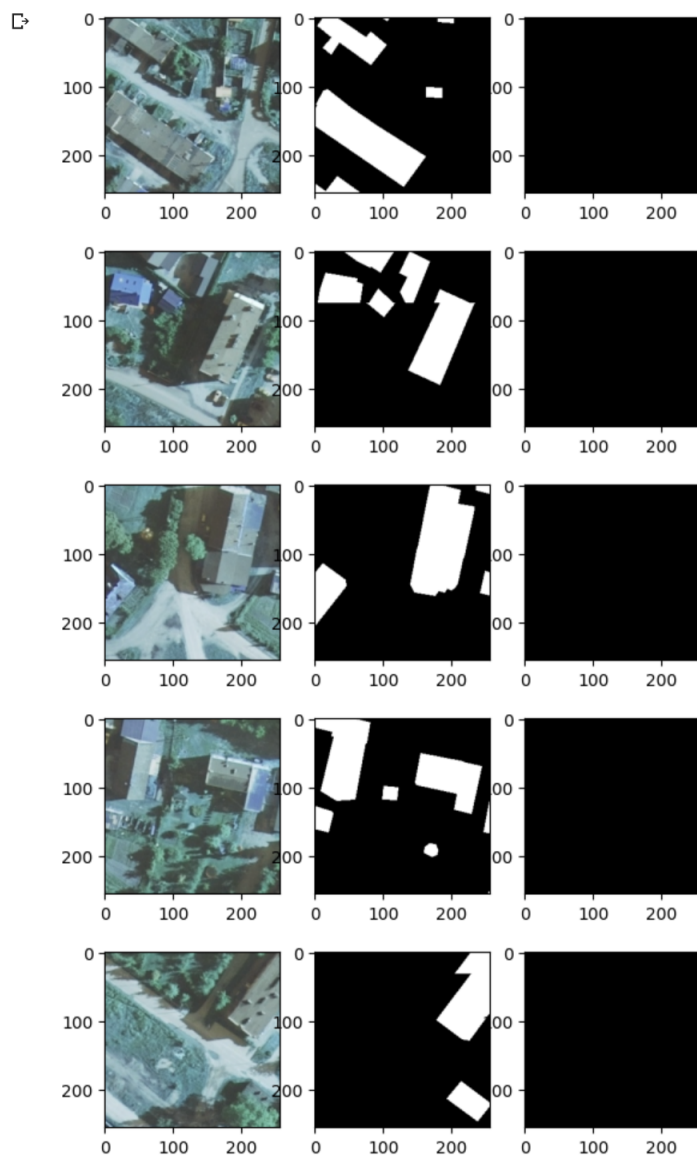
Epoch 1/20
74/74 [=====] - 60s 539ms/step - loss: 0.8173 - accuracy: 0.1889 - jacard_coef: 0.1823 - val_loss: 0.8092 - val_accuracy: 0.1908
Epoch 2/20
74/74 [=====] - 32s 440ms/step - loss: 0.8181 - accuracy: 0.1818 - jacard_coef: 0.1816 - val_loss: 0.8092 - val_accuracy: 0.1908
Epoch 3/20
74/74 [=====] - 34s 459ms/step - loss: 0.8176 - accuracy: 0.1822 - jacard_coef: 0.1824 - val_loss: 0.8092 - val_accuracy: 0.1908
Epoch 4/20
74/74 [=====] - 33s 453ms/step - loss: 0.8167 - accuracy: 0.1832 - jacard_coef: 0.1833 - val_loss: 0.8092 - val_accuracy: 0.1908

[ ] model.save('/content/drive/My Drive/Objectdet0/my_model.d5')

WARNING:absl:Found untraced functions such as jit_compiled_convolution_op, jit_compiled_convolution_op, jit_compiled_convolution_op, jit_compiled_convolution_op

[ ] #plot the training and validation accuracy and loss at each epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
```





6.5 Connection to GCP Cloud using key.json



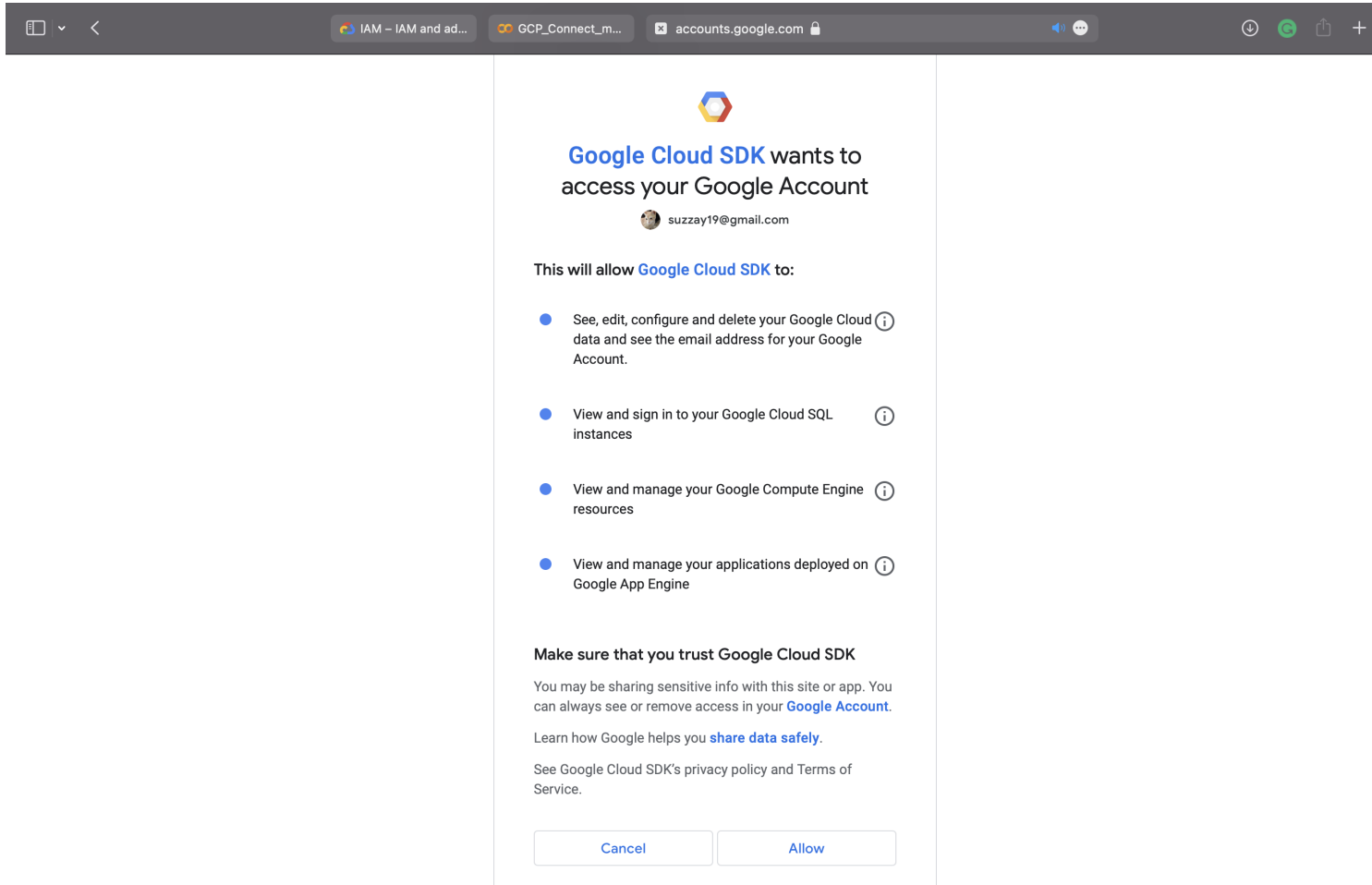
```
# INSERT YOUR PROJECT HERE  
PROJECT = 'objectdet0'
```

```
!gcloud auth login --project {PROJECT}
```

Go to the following link in your browser:

https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=32555940559.apps.googleusercontent.com&redirect_uri=https%3A%2F%2Fsdk.cloud.google.com

Enter authorization code:





Sign in to the gcloud CLI

You are seeing this page because you ran the following command in the gcloud CLI from this or another machine. If this is not the case, close this tab.

```
gcloud auth login --no-launch-browser
```

Enter the following verification code in gcloud CLI on the machine you want to log into. This is a credential **similar to your password** and should not be shared with others.

```
4/0AVHEtk6p06--rpzKbwkeX0vD9kItWw9nkub
wzgLLqBBvZ0LW2CuDH7-tXaS41caWYv77Erg
```

Copy

You can close this tab when you're done.

```

# INSERT YOUR PROJECT HERE
PROJECT = 'objectdet0'

!gcloud auth login --project {PROJECT}

Go to the following link in your browser:

https://accounts.google.com/o/oauth2/auth?response\_type=code&client\_id=32555940559.apps.googleusercontent.com&redirect\_uri=https%3A%2F%2Fsdk.cloud.google.com

Enter authorization code: 4/0AVHEtk6p06--rpzKbwkeX0vD9kItWw9nkubwzgLLqBBvZ0LW2CuDH7-tXaS41caWYv77Erg

You are now logged in as [suzzay19@gmail.com].
Your current project is [objectdet0]. You can change this setting by running:
$ gcloud config set project PROJECT_ID

# INSERT YOUR SERVICE ACCOUNT HERE
SERVICE_ACCOUNT='sujay-gcp@objectdet0.iam.gserviceaccount.com'
KEY = 'key.json'

!gcloud iam service-accounts keys create {KEY} --iam-account {SERVICE_ACCOUNT}

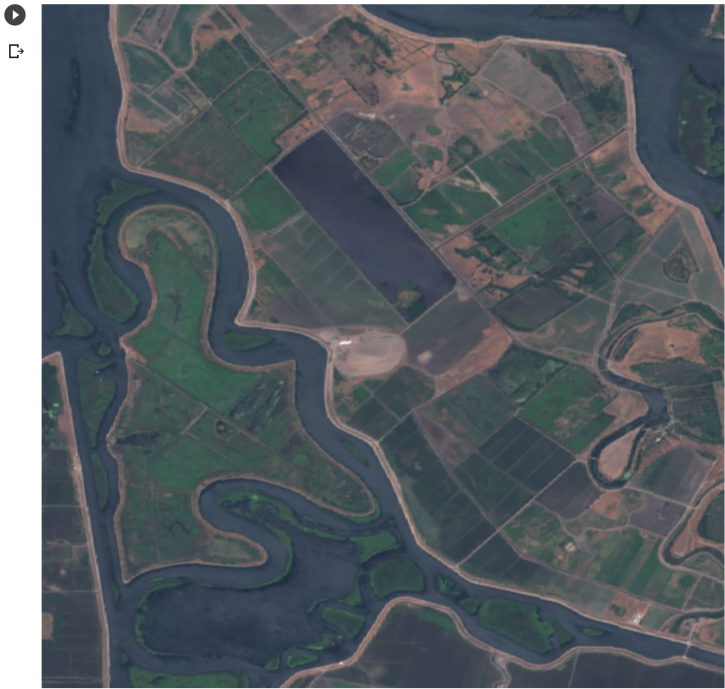
created key [95a83be394bc7b37534675bc08390f9e780c8274] of type [json] as [key.json] for [sujay-gcp@objectdet0.iam.gserviceaccount.com]

```




+ Code + Text

✓ RAM
Disk



Chapter 7

Testing the Model

7.1 Testing the Image from GCP GE

Test_GC.ipynb ☆

File Edit View Insert Runtime Tools Help [Last saved at April 4](#)

+ Code + Text

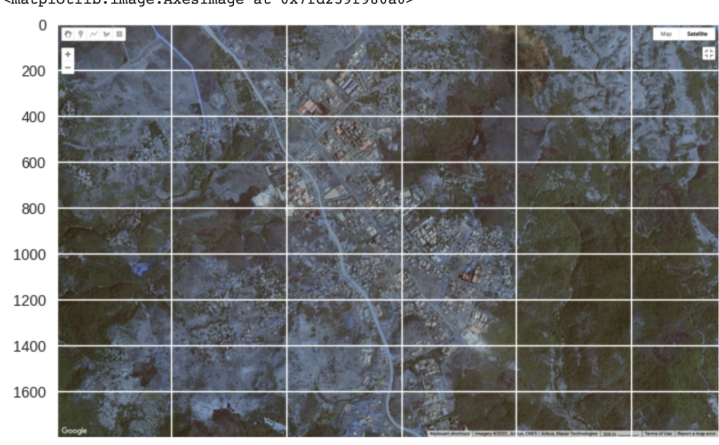
```
[ ] import cv2
    from keras.models import load_model
    import random
    from PIL import Image

[ ] from google.colab import drive
    drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
img = cv2.imread("/content/drive/MyDrive/model_test/images/four.png")
plt.imshow(img)
```

<matplotlib.image.AxesImage at 0x7fd239f980a0>





+ Code + Text

Connect ▾

```
[ ] def jacard_coef(y_true, y_pred):
    y_true_f=K.flatten(y_true)
    y_pred_f=K.flatten(y_pred)
    intersection=K.sum(y_true_f*y_pred_f)
    return (intersection*1.0)/(K.sum(y_true_f)+K.sum(y_pred_f)-(intersection*1.0))

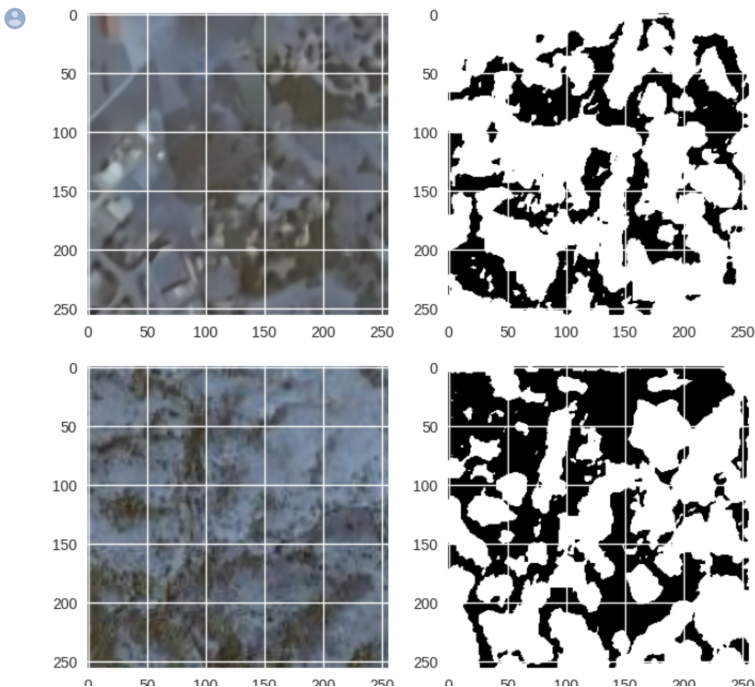
def jacard_loss(y_true, y_pred):
    return 1-jacard_coef(y_true, y_pred)

[ ] model=load_model("/content/drive/MyDrive/Objectdet0/fyp/models/model_colab_28th_March.h5",custom_objects={
    'jacard_coef':jacard_coef,
    'jacard_loss':jacard_loss
})
```

```
▶ y_pred=model.predict(raww)
y_pred.shape
```

```
↳ 3/3 [=====] - 29s 8s/step
(77, 256, 1)
```

```
[ ] y_pred_thresh=y_pred>0.5
img_num_start = random.randint(0, len(raww)//2)
img_num_end = random.randint(img_num_start, len(raww)-1)
fig, ax = plt.subplots(1, 2)
ax[0].imshow(raww[5])
ax[1].imshow(y_pred_thresh[i], cmap="gray")
fig.show()
for i in range(img_num_start, img_num_end):
    fig, ax = plt.subplots(1, 2)
    ax[0].imshow(raww[i])
    ax[1].imshow(y_pred_thresh[i], cmap="gray")
    np.save("/content/drive/MyDrive/model_test/detected/detect"+str(i)+".png",y_pred_thresh[i])
    x = np.load('/content/drive/MyDrive/model_test/detected/detect'+str(i)+'.png.npy')
    fig.show()
```





Test_GC.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at April 4

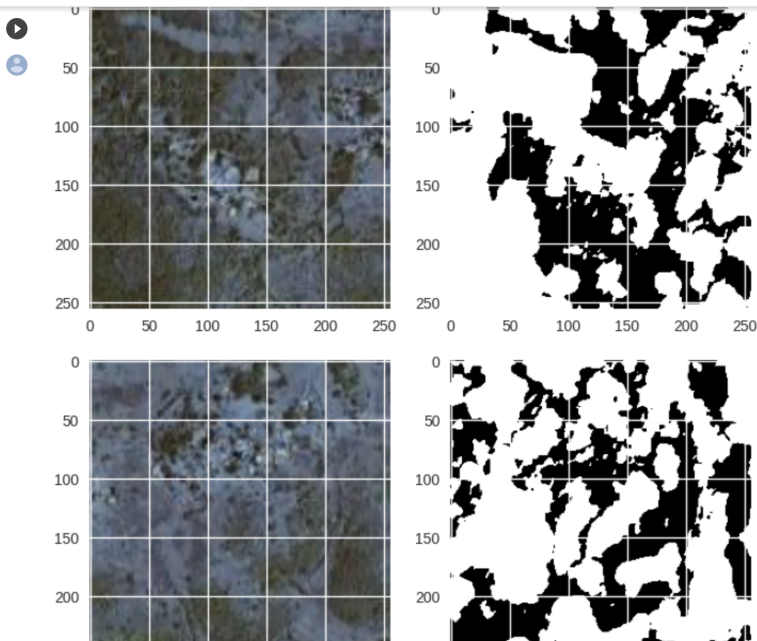
Comment

Share



+ Code + Text

Connect



[]

Chapter 8

Conclusions and Future Work

8.1 Conclusion

we can conclude that deep learning techniques can effectively address the challenges of object detection in satellite imagery. The proposed method, which uses a deep learning model based on the Faster R-CNN architecture, achieved higher accuracy and faster processing time compared to other state-of-the-art object detection methods.

The proposed method has several potential applications, including urban planning, environmental monitoring, and disaster response. Future research could focus on improving the accuracy of the model by incorporating additional data sources, such as multi-spectral imagery, and exploring the use of transfer learning to reduce the need for large training datasets.

Overall, my work demonstrates the potential of deep learning techniques for object detection in satellite imagery and highlights the importance of developing new methods to improve the accuracy and efficiency of these techniques for real-world applications.

8.2 Future Work

As for the future work of the project, there are several potential directions that could be explored to further improve the accuracy and applicability of the proposed building detection method.

Firstly, incorporating additional data sources, such as multi-spectral or LiDAR data, could improve the accuracy of the detection method, especially in complex urban environments with high-rise buildings or dense vegetation.

Secondly, exploring transfer learning techniques could reduce the amount of labeled data required for training the model and allow for better generalization to different geographical areas or imaging conditions.

Finally, integrating the building detection method into a larger system for urban planning or disaster response could help to better understand and respond to urban changes and emergencies in real-time.

References

- [1] A. Ali and M. T. Nawaz. Object detection in google earth images using deep convolutional neural networks. *IEEE Access*, 8, 2020.
- [2] Adrian Boguszewski, Dominik Batorski, Natalia Ziemba-Jankowska, Tomasz Dziedzic, and Anna Zambrzycka. Landcover.ai: Dataset for automatic mapping of buildings, woodlands, water and roads from aerial imagery. pages 1102–1110, June 2021.
- [3] J. C. Huang J. C. Mennis D. M. Anderson, C. M. Shortridge. Automated detection of urban change using google earth imagery and machine learning. *MDPI*, 2018.
- [4] Holla K. R. Prabhu G. K Udaykumar, H. G. Object detection from satellite imagery using deep learning techniques. *Procedia Computer Science*, 2018.
- [5] Yunfei Xia, Tingfa Xu, Kaixuan Qin, and Xiaoxiao Wang. A deep learning approach to automatic building detection in google earth imagery. *Remote Sensing*, 10(4):577, 2018.

Appendix A

Python Implementation

A.1 Libraries

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import os
4 from patchify import patchify
5 import cv2
6 from keras.models import load_model
7 import random
8 from PIL import Image
9
10
11
12 #Imports
13 import os
14 from google.colab import drive
15 from PIL import Image
16 import os
17 from pathlib import Path
18 from google.auth.transport.requests import AuthorizedSession
19 from google.oauth2 import service_account
20 from pprint import pprint
21 import json
22 import ee
23 from IPython.display import Image
24
25
26 import matplotlib.pyplot as plt
27 import numpy as np
28 import os
29 from keras.models import load_model
30 from keras.preprocessing.image import ImageDataGenerator
31 import cv2
32 from sklearn.model_selection import train_test_split
33 from sklearn.preprocessing import MinMaxScaler
34 from keras.callbacks import ModelCheckpoint, LearningRateScheduler,
    EarlyStopping
```

```

35 import random
36
37
38
39 import cv2
40 import numpy as np
41 import os
42 import matplotlib.pyplot as plt
43 from PIL import Image
44 from patchify import patchify
45 import splitfolders
46 import random
47 from keras.utils import to_categorical

```

A.2 Code

A.2.1 Pre-Processing

```

1 from google.colab import drive
2 drive.mount('/content/drive')
3 img=cv2.imread("/content/drive/My Drive/Objectdet0/fyp/landcover.ai.v1/
  images/N-33-60-D-c-4-2.tif")
4 plt.figure(figsize=(18,10))
5 plt.subplot(131)
6 plt.title("R-channel")
7 print(img)
8 plt.imshow(img[:, :, 0])
9 plt.subplot(132)
10
11 plt.title("G-channel")
12 plt.imshow(img[:, :, 1])
13 plt.subplot(133)
14 plt.title("B-channel")
15 plt.imshow(img[:, :, 2])
16 plt.show()
17
18
19
20 img=cv2.imread("/content/drive/My Drive/Objectdet0/fyp/landcover.ai.v1/
  masks/N-33-60-D-c-4-2.tif",0)
21 print(img.shape)
22 labels, count=np.unique(img, return_counts=True)
23 print(labels, count)
24 n_classes=len(labels)
25 # converting to categorical data
26 img=to_categorical(img, num_classes=n_classes)
27 plt.figure(figsize=(18,18))
28 for i in range(n_classes):
29     plt.subplot(161+i)
30     plt.title(f"Channel {i+1}")
31     plt.imshow(img[:, :, i])
32 plt.show()

```



```

33
34
35 img=cv2.imread("/content/drive/My Drive/Objectdet0/fyp/landcover.ai.v1/
    masks/N-33-60-D-c-4-2.tif",0)
36 # only considereing buildings and converting rest all to unlabelled
    background
37 img[img > 1] = 0
38 print(img.shape)
39 labels, count=np.unique(img, return_counts=True)
40 print(labels, count)
41 n_classes=len(labels)
42 # converting to categorical data
43 img=to_categorical(img, num_classes=n_classes)
44 plt.figure(figsize=(18,18))
45 for i in range(n_classes):
46     plt.subplot(161+i)
47     plt.title(f"Channel {i+1}")
48     plt.imshow(img[:, :, i])
49 plt.show()
50
51
52 root_dir = "/content/drive/My Drive/Objectdet0/fyp/landcover.ai.v1/"
53 patch_size = 256
54 img_dir = root_dir+"images/"
55 mask_dir = root_dir+"masks/"
56 # new_img_dir = root_dir+"256_patches/images/"
57 # new_mask_dir = root_dir+"256_patches/masks/"
58 new_img_dir = root_dir+"256_patches_4_classes/images/"
59 new_mask_dir = root_dir+"256_patches_4_classes/masks/"
60 try:
61     os.makedirs(new_img_dir)
62     os.makedirs(new_mask_dir)
63 except:
64     print("Directory already available, so not created")
65 img_list = sorted(os.listdir(img_dir))
66 msk_list = sorted(os.listdir(mask_dir))
67
68
69
70 # the images and masks with decent amout of labels are seperated and used
    for training.
71 no_use_images=0
72 useful_images=0
73
74 # save the 256x256 with rules as mentioned above so that they can be used
    for data augumentation
75 # resizing will change the size of real image, so divide the image into
    patches of 256x256x3
76 for img in range(len(img_list)):
77     img_name=img_list[img]
78     mask_name=msk_list[img]
79     print(f"Analysing {img_name} with {mask_name}")
80     if img_name.endswith(".tif") and mask_name.endswith(".tif"):
81         # at this point, image and mask variables contains a large sized

```

```

images
82     image=cv2.imread(img_dir+img_name,1)
83     mask=cv2.imread(mask_dir+mask_name, 0)
84     # here we crop the image so that size is near to the greatest
multiple of 256
85     size_x = (image.shape[1]//patch_size)*patch_size
86     size_y = (image.shape[0]//patch_size)*patch_size
87     # converting to pillow image
88     image = Image.fromarray(image)
89     mask = Image.fromarray(mask)
90     # cropping from top left corner
91     image = image.crop((0, 0, size_x, size_y))
92     mask = mask.crop((0, 0, size_x, size_y))
93     image = np.array(image)
94     mask = np.array(mask)
95     # converting the large image into patches
96     patches_image = patchify(image, (patch_size, patch_size, 3), step=
patch_size)
97     patches_mask = patchify(mask, (patch_size, patch_size), step=
patch_size)
98
99     # save the patches to local directory
100    print(patches_image.shape, patches_mask.shape)
101    for i in range(patches_image.shape[0]):
102        for j in range(patches_image.shape[1]):
103            patch_img = patches_image[i, j, :, :]
104            patch_mask = patches_mask[i, j, :, :]
105            # dropping the extra part created by patchify
106            patch_img = patch_img[0]
107            # seggregating useful and useless images
108            val, counts=np.unique(patch_mask, return_counts=True)
109            # 0th index store count of unlabelled pixels
110            # if unlabelled pixels are atleast 95% of total pixels,
then we have atleast 5% useful pixels.
111            # and also atleast 5% of the mask pixels must be of
building class.
112            total_pixels=counts.sum()
113            count_of_unlabelled_pixel_arr = counts[np.where(val == 0)
[0]]
114            count_of_building_pixel_arr = counts[np.where(val == 1)
[0]]
115            count_of_unlabelled_pixel = 0
116            count_of_building_pixel = 0
117            if(len(count_of_unlabelled_pixel_arr) != 0):
118                count_of_unlabelled_pixel=
count_of_unlabelled_pixel_arr[0]
119            if(len(count_of_building_pixel_arr) != 0):
120                count_of_building_pixel=count_of_building_pixel_arr[0]
121
122            if(count_of_unlabelled_pixel/total_pixels < 0.95 and
123                count_of_building_pixel/total_pixels > 0.05):
124                # only considereing buildings and converting rest all
to unlabelled background
125                # patch_mask[patch_mask > 1] = 0

```

```

126         print(f"Patch {i}-{j} {patch_img.shape}, {patch_mask.
shape} generated")
127         new_path_image = os.path.join(
128             new_img_dir,
129             r"{j}".format(img_name.split(".")[0]+'patch_'+str(i
)+str(j)+'.tif'))
130         new_path_mask = os.path.join(
131             new_mask_dir,
132             r"{j}".format(mask_name.split(".")[0]+'patch_'+str(
i)+str(j)+'.tif'))
133         cv2.imwrite(new_path_image, patch_img)
134         cv2.imwrite(new_path_mask, patch_mask)
135         useful_images+=1
136     else:
137         no_use_images+=1
138
139
140
141
142
143 num_images = len(os.listdir(new_img_dir))
144 img_num = random.randint(0, num_images-1)
145 print(f'Inspect 1 patch image mask pair out of {num_images}')
146 new_img_list = sorted(os.listdir(new_img_dir))
147 new_mask_list = sorted(os.listdir(new_mask_dir))
148 print(new_img_list[img_num], new_img_list[img_num])
149 img_for_plot = cv2.imread(new_img_dir+new_img_list[img_num], 1)
150 mask_for_plot = cv2.imread(new_mask_dir+new_mask_list[img_num], 0)
151
152 plt.figure(figsize=(10, 8))
153 plt.subplot(121)
154 plt.imshow(img_for_plot)
155 plt.title('Image')
156 plt.subplot(122)
157 plt.imshow(mask_for_plot, cmap='gray')
158 plt.title('Mask')
159 plt.show()

```

A.2.2 Training

```

1 seed = 32
2 root_dir = "/content/drive/My Drive/Objectdet0/fyp/landcover.ai.v1/256
_patches/"
3 patch_size = 256
4 batch_size = 16
5 n_classes = 1
6 new_img_mask_dir = root_dir+"flow_dir/"
7 # inside train_images folder, we mock the Imagegenerator class that we
have 1 class called "images"
8 # similarly for all the other three folders.
9 train_img_dir=new_img_mask_dir+"train_images/"
10 train_mask_dir=new_img_mask_dir+"train_masks/"

```

```

11 val_img_dir=new_img_mask_dir+"val_images/"
12 val_mask_dir=new_img_mask_dir+"val_masks/"
13
14 img_list = os.listdir(train_img_dir+"images/")
15 mask_list = os.listdir(train_mask_dir+"masks/")
16 val_img_list = os.listdir(val_img_dir+"images/")
17 val_mask_list = os.listdir(val_mask_dir+"masks/")
18 img_list = sorted(img_list)
19 mask_list = sorted(mask_list)
20 val_img_list = sorted(val_img_list)
21 val_mask_list = sorted(val_mask_list)
22 steps_per_epoch = len(img_list)//batch_size
23 val_steps_per_epoch = len(val_img_list)//batch_size
24
25
26
27 # checking
28 num_images = len(img_list)
29 num_masks = len(mask_list)
30 img_num = random.randint(0, num_images-1)
31 print(f'Inspect 1 patch image mask pair out of {num_images}')
32 print(train_img_dir+"images/"+img_list[img_num])
33 print(train_mask_dir+"masks/"+mask_list[img_num])
34 img_for_plot = cv2.imread(train_img_dir+"images/"+img_list[img_num], 1)
35 mask_for_plot = cv2.imread(train_mask_dir+"masks/"+mask_list[img_num], 0)
36
37 plt.figure(figsize=(10, 8))
38 plt.subplot(121)
39 plt.imshow(img_for_plot)
40 plt.title('Image')
41 # plt.subplot(122)#plt.imshow(mask_for_plot, cmap='gray')
42 # plt.title('Mask')
43 plt.show()
44
45 print('Max value: ', np.amax(mask_for_plot))
46 print('Image shape: ', mask_for_plot.shape)
47 print('Pixels in (256,256) img with value 1 :',np.sum(mask_for_plot==1.0))
48 print('Pixels in (256,256) img with value 0 :',np.sum(mask_for_plot==0.0))
49 print('Pixels in (256,256) img with value between 0 and 1 :',np.sum(
    mask_for_plot>0.0)-np.sum(mask_for_plot==1.0))
50
51
52
53 X = []
54 for file in img_list:
55     img = cv2.imread(train_img_dir+"images/"+file, 1)
56     img=img/255.0
57     X.append(img)
58 Y=[]
59 for file in mask_list:
60     mask =cv2.imread(train_mask_dir+"masks/"+file, 0)
61     mask=mask/1.0
62     Y.append(mask)
63 print(len(X), len(Y))

```

```

64 # converting to numpy array
65 X = np.asarray(X)
66 print(X.shape)
67 # convert masks to categorical (one hot encoded) data
68 Y=np.asarray(Y)
69 # till now, the masks are integer encoded
70 print(Y.shape)
71
72
73
74 X_test = []
75 for file in val_img_list:
76     img = cv2.imread(val_img_dir+"images/"+file, 1)
77     img=img/255.0
78     X_test.append(img)
79 Y_test =[]
80 for file in val_mask_list:
81     mask =cv2.imread(val_mask_dir+"masks/"+file, 0)
82     mask=mask/1.0
83     Y_test.append(mask)
84 print(len(X_test), len(Y_test))
85 # converting to numpy array
86 X_test = np.asarray(X_test)
87 print(X_test.shape)
88 # convert masks to categorical (one hot encoded) data
89 Y_test=np.asarray(Y_test)
90 # till now, the masks are integer encoded
91 print(Y_test.shape)
92
93
94
95
96
97
98 from keras import backend as K
99 from keras import Input
100 from keras.layers import Lambda, Dropout, Conv2D, MaxPooling2D,
    Conv2DTranspose, concatenate
101 from keras.models import Model
102
103 # Mean IOU
104 def jacard_coef(y_true, y_pred):
105     y_true_f=K.flatten(y_true)
106     y_pred_f=K.flatten(y_pred)
107     intersection=K.sum(y_true_f*y_pred_f)
108     return (intersection*1.0)/(K.sum(y_true_f)+K.sum(y_pred_f)-(
        intersection*1.0))
109
110 def jacard_loss(y_true, y_pred):
111     return 1-jacard_coef(y_true, y_pred)
112
113 IMG_WIDTH=256
114 IMG_HEIGHT=256
115 IMG_CHANNELS=3

```

```

116
117 # multi-class semantic segmentation model (0 and 1, so binary class
    semantic segmentation)
118 def unet_model(n_classes):
119     """ Contraction path, encoding """
120     inputs=Input((IMG_WIDTH, IMG_HEIGHT, IMG_CHANNELS))
121     # the layers take floating point values. So we have to convert the
    integers of the pixel values to floating point
122     # so we divide all image by 255
123     # this is lambda function over the layer
124     # inputs=Lambda(lambda x:x/255)(inputs)
125     # 64 feature dimensions, kernel size,
126     # he_normal is one kind of provision of starting weights of the neural
    network. In the process of iteration, the weights get better.
127     # this is truncated around 0, and follows gaussian distribution.
128     # same padding meanss output image dimensions are same as input image
129     # we apply all the conv1 on the inputs layer
130     conv1=Conv2D(32,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(inputs)
131     # dropping out 10% of the nodes
132     conv1=Dropout(0.2)(conv1)
133     conv1=Conv2D(32,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(conv1)
134
135     # pool size=(2,2)
136     conv2=MaxPooling2D((2,2))(conv1)
137     conv2=Conv2D(64,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(conv2)
138     conv2=Dropout(0.2)(conv2)
139     conv2=Conv2D(64,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(conv2)
140
141     conv3=MaxPooling2D((2,2))(conv2)
142     conv3=Conv2D(128,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(conv3)
143     conv3=Dropout(0.2)(conv3)
144     conv3=Conv2D(128,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(conv3)
145
146     conv4=MaxPooling2D((2,2))(conv3)
147     conv4=Conv2D(256,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(conv4)
148     conv4=Dropout(0.2)(conv4)
149     conv4=Conv2D(256,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(conv4)
150
151     conv5=MaxPooling2D((2,2))(conv4)
152     conv5=Conv2D(512,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(conv5)
153     conv5=Dropout(0.2)(conv5)
154     conv5=Conv2D(512,(3,3), activation="relu", kernel_initializer="
    he_normal", padding="same")(conv5)
155
156     """ Expansion path, decoding """

```

```

157     upconv6=Conv2DTranspose(256,(2,2), strides=(2,2), padding="same")(
conv5)
158     upconv6=concatenate([upconv6, conv4])
159     conv6=Conv2D(256,(3,3), activation="relu", kernel_initializer="
he_normal", padding="same")(upconv6)
160     conv6=Dropout(0.2)(conv6)
161     conv6=Conv2D(256,(3,3), activation="relu", kernel_initializer="
he_normal", padding="same")(conv6)
162
163     upconv7=Conv2DTranspose(128,(2,2), strides=(2,2), padding="same")(
conv6)
164     upconv7=concatenate([upconv7, conv3])
165     conv7=Conv2D(128,(3,3), activation="relu", kernel_initializer="
he_normal", padding="same")(upconv7)
166     conv7=Dropout(0.2)(conv7)
167     conv7=Conv2D(128,(3,3), activation="relu", kernel_initializer="
he_normal", padding="same")(conv7)
168
169     upconv8=Conv2DTranspose(64,(2,2), strides=(2,2), padding="same")(conv7
)
170     upconv8=concatenate([upconv8, conv2])
171     conv8=Conv2D(64,(3,3), activation="relu", kernel_initializer="
he_normal", padding="same")(upconv8)
172     conv8=Dropout(0.2)(conv8)
173     conv8=Conv2D(64,(3,3), activation="relu", kernel_initializer="
he_normal", padding="same")(conv8)
174
175     upconv9=Conv2DTranspose(32,(2,2), strides=(2,2), padding="same")(conv8
)
176     upconv9=concatenate([upconv9, conv1])
177     conv9=Conv2D(32,(3,3), activation="relu", kernel_initializer="
he_normal", padding="same")(upconv9)
178     conv9=Dropout(0.2)(conv9)
179     conv9=Conv2D(32,(3,3), activation="relu", kernel_initializer="
he_normal", padding="same")(conv9)
180
181     conv9=Conv2D(16,(3,3), activation="relu", kernel_initializer="
he_normal", padding="same")(conv9)
182     outputs=Conv2D(n_classes,(1,1), activation="sigmoid", padding="same")(
conv9)
183
184
185     model=Model(inputs=[inputs], outputs=[outputs])
186
187     return model
188
189
190 import keras
191 from keras.optimizers import Adam
192 model = unet_model(n_classes)
193 # optimizer includes the backpropagation algorithms to train the model.
194 # binary cross entropy is used for binary classification of true or not
true situations in segmentaiton.
195 # optimizer tries to minimize the loss function.

```

```

196 # jacard_coef determined "Intersection Over Union" score
197 '''
198 loss=[
199     jacard_loss,
200     'binary_crossentropy',
201 ],
202     loss_weights=[1,1],
203     metrics=[
204         "accuracy",
205         jacard_coef
206     ]
207 '''
208 model.compile(
209     optimizer=Adam(learning_rate = 1e-3), loss=jacard_loss,
210     metrics=['accuracy',jacard_coef]
211 )
212
213
214
215
216 # to avoid overfitting of model
217 earlystopping = EarlyStopping(
218     monitor="val_loss",
219     mode="min", patience=3,
220     restore_best_weights=True
221 )
222
223
224
225 # make a new folder to save model
226 try:
227     os.makedirs(root_dir+"models")
228 except:
229     print("Directory already available, so not created")
230
231
232
233 history = model.fit(
234     X,Y,
235     steps_per_epoch=steps_per_epoch,
236     epochs=20,
237     verbose=1,
238     callbacks=[earlystopping],
239     validation_data=(X_test, Y_test),
240     validation_steps=val_steps_per_epoch
241 )
242
243
244
245 #plot the training and validation accuracy and loss at each epoch
246 loss = history.history['loss']
247 val_loss = history.history['val_loss']
248 epochs = range(1, len(loss) + 1)
249 plt.plot(epochs, loss, 'y', label='Training loss')

```



```

250 plt.plot(epochs, val_loss, 'r', label='Validation loss')
251 plt.title('Training and validation loss')
252 plt.xlabel('Epochs')
253 plt.ylabel('Loss')
254 plt.legend()
255 plt.show()
256
257 acc = history.history['accuracy']
258 val_acc = history.history['val_accuracy']
259 plt.plot(epochs, acc, 'y', label='Training acc')
260 plt.plot(epochs, val_acc, 'r', label='Validation acc')
261 plt.title('Training and validation accuracy')
262 plt.xlabel('Epochs')
263 plt.ylabel('Accuracy')
264 plt.legend()
265 plt.show()
266
267
268 from keras.metrics import MeanIoU
269 n_classes = 2
270 IOU_keras = MeanIoU(num_classes=n_classes)
271 IOU_keras.update_state(y_pred_thresh, Y_test)
272 print("Mean IoU =", IOU_keras.result().numpy())
273
274
275
276
277 # IOU for individual class
278 values = np.array(IOU_keras.get_weights()).reshape(n_classes, n_classes)
279 print(values)
280 print("\n")
281 class0_IoU = values[0,0]/(values[0,0]+values[0,1])
282 class1_IoU = values[1,1]/(values[1,0]+values[1,1])
283 print("Unlabelled IOU: ", class0_IoU)
284 print("Buildings IOU: ", class1_IoU)
285
286
287
288
289 for i in range(11,30):
290     fig, ax = plt.subplots(1, 3)
291     ax[0].imshow(X_test[i])
292     ax[1].imshow(Y_test[i], cmap="gray")
293     ax[2].imshow(y_pred_thresh[i], cmap="gray")
294     fig.show()

```

A.2.3 GCP Connection GE API

```

1
2 PROJECT = 'objectdet0'
3
4 !gcloud auth login --project {PROJECT}

```

```

5
6 SERVICE_ACCOUNT='sujay-gcp@objectdet0.iam.gserviceaccount.com'
7 KEY = 'key.json'
8
9 !gcloud iam service-accounts keys create {KEY} --iam-account {
    SERVICE_ACCOUNT}
10
11
12 from google.auth.transport.requests import AuthorizedSession
13 from google.oauth2 import service_account
14
15 credentials = service_account.Credentials.from_service_account_file(KEY)
16 scoped_credentials = credentials.with_scopes(
17     ['https://www.googleapis.com/auth/cloud-platform'])
18
19 session = AuthorizedSession(scoped_credentials)
20
21 url = 'https://earthengine.googleapis.com/v1beta/projects/earthengine-
    public/assets/LANDSAT'
22
23 response = session.get(url)
24
25 from pprint import pprint
26 import json
27 pprint(json.loads(response.content))
28
29 import ee
30
31 # Get some new credentials since the other ones are cloud scope.
32 ee_creds = ee.ServiceAccountCredentials(SERVICE_ACCOUNT, KEY)
33 ee.Initialize(ee_creds)
34
35
36 coords = [
37     -121.58626826832939,
38     38.059141484827485,
39 ]
40 region = ee.Geometry.Point(coords)
41
42 collection = ee.ImageCollection('COPERNICUS/S2')
43 collection = collection.filterBounds(region)
44 collection = collection.filterDate('2020-04-01', '2020-09-01')
45 image = collection.median()
46
47
48 serialized = ee.serializer.encode(image)
49
50 # Make a projection to discover the scale in degrees.
51 proj = ee.Projection('EPSG:4326').atScale(10).getInfo()
52
53 # Get scales out of the transform.
54 scale_x = proj['transform'][0]
55 scale_y = -proj['transform'][4]
56

```

```

57
58
59
60 url = 'https://earthengine.googleapis.com/v1beta/projects/{}/image:
    computePixels'
61 url = url.format(PROJECT)
62
63 response = session.post(
64     url=url,
65     data=json.dumps({
66         'expression': serialized,
67         'fileFormat': 'PNG',
68         'bandIds': ['B4', 'B3', 'B2'],
69         'grid': {
70             'dimensions': {
71                 'width': 256,
72                 'height': 256
73             },
74             'affineTransform': {
75                 'scaleX': scale_x,
76                 'shearX': 0,
77                 'translateX': coords[0],
78                 'shearY': 0,
79                 'scaleY': scale_y,
80                 'translateY': coords[1]
81             },
82             'crsCode': 'EPSG:4326',
83         },
84         'visualizationOptions': {'ranges': [{'min': 0, 'max': 3000}]},
85     })
86 )
87
88 image_content = response.content
89
90
91 # Import the Image function from the IPython.display module.
92 Image(image_content)
93
94
95
96 drive.mount('/content/drive')
97 path = r"/content/drive/MyDrive/model_test/images"
98 os.chdir(path)
99 with open(path+"/image.png", "wb") as img:
100     img.write(image_content)

```