Aircraft Detection and Classification using Satellite Imagery (ADCSI)

By

MUHAMMAD AHMAD

MUHAMMAD SAFIULLAH

UMAIR AMIN

TALHA TARIQ

Supervised by:

DR. HASNAT KHURSHID

Submitted to the faculty of the Department of Electrical Engineering,
Military College of Signals, National University of Sciences and Technology, Islamabad,
in partial fulfilment of the requirements of a B.E Degree in Electrical (Telecom) Engineering.

June 2023

In the name of ALLAH, the Most benevolent, the Most Courteous

CERTIFICATE OF CORRECTNESS AND APPROVAL

This is to state that the thesis work contained in this report officially

"Aircraft Detection and Classification using Satellite Imagery"

is carried out by

MUHAMMAD AHMAD

MUHAMMAD SAFIULLAH

UMAIR AMIN

TALHA TARIQ

under my supervision and in my judgement, it is fully ample, in scope and excellence, for the degree of Bachelor of Electrical (Telecom.) Engineering in Military College of Signals, National University of Sciences and Technology (NUST), Islamabad.

Approved by

Supervisor

Lt Col. Dr Hasnat Khurshid

Department of EE, MCS

Date: ___________

DECLARATION OF ORIGINALITY

We hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification in either this institute or anywhere else.

ACKNOWLEDGEMENTS

Allah Subhan’Wa’Tala is the sole guidance in all domains.

Our parents, colleagues and most of all, supervisor, **Lt Col Dr Hasnat Khurshid,** without your guidance it would have been a challenging undertaking.

The group members, through all adversities, worked steadfastly.

Plagiarism Certificate (Turnitin Report)

This thesis has an **<u>11%</u>** similarity index. The Turnitin report endorsed by Supervisor is attached.

Muhammad Ahmad
325013

Muhammad Safiullah
325049

Umair Amin
325040

Talha Tariq
325029

Signature of Supervisor

ABSTRACT

In the modern era, satellite or drone imagery is easily accessible. There are several uses for such images, including the detection and identification of desired targets like aircraft, convoys, trains, and trucks. It can also be used to identify infrastructure, like runways, storage buildings, Air bases and Airports. Our effort adds the identification and classification of aircraft in Google Earth Imagery as another extremely effective use of overhead imagery, broadening the scope of these applications. This can be very helpful for locating and documenting an aircraft in a particular area. In this study, a large number of multi-resolution satellite images were used to train the Convolutional Neural Network-based machine learning algorithm YOLO V5. By selecting training parameters optimized by learning from multiple literature sources and testing them, the models were trained. After a period of extensive model training and achieving desirable accuracy, two user interfaces were developed. Users can detect in real-time when browsing Google Earth or any other source of overhead imagery with the use of UI Live Detection Mode. In Google Earth's Auto-Scan mode, a predefined area is automatically scanned, and all detections are recorded along with classification information. Along with the pixel values of aircraft in the Google Earth image that was acquired, the UI can precisely provide the specific geographic coordinates of the aircraft.

Table of Contents

List of Figures

Chapter 1: Introduction

Engineering is a required field that has had a tremendous impact on society's development and progress. By applying scientific knowledge and skills, engineers design and develop new technologies that can solve complex problems and enhance people's lives across various industries, including healthcare, transportation, entertainment, and National Security.

Automation is a prime example of a technological advancement resulting from engineering research, which has transformed the way we live and work. Machines can now be programmed to perform tasks once done by humans, increasing efficiency, accuracy, and productivity. Automation has made its way into many industries, from manufacturing to healthcare, and has decreased labor costs while improving safety.

Automation is one such technology that has been widely employed to strengthen security measures in various domains. It involves the use of machines that are programmed to make decisions based on real-time data analysis and historical trends.

Automation software has been developed by engineers to help monitor and control various security systems in real-time. For instance, it can be used to analyse overhead imagery by satellites and drones to detect important national assets like Buildings, Convoys and aircraft. This technology can provide a crucial piece of intelligence information.

Overall, engineering and automation software has played a crucial role in enhancing national security by providing adequate and efficient solutions to identify and mitigate potential threats. As technology continues to evolve, engineers will undoubtedly continue to develop new and innovative solutions to protect and enhance national security.

1.1 Overview

In the modern world, intelligence gathering has emerged as a critical aspect of national security and foreign policy. The complex and rapidly changing geopolitical landscape, coupled with the advancements in technology, have made the gathering of accurate and timely intelligence more critical than ever before. Intelligence gathering enables governments and policymakers to make informed decisions on matters of national importance, ranging from counterterrorism and border security to economic policies and trade negotiations. It also plays a vital role in international diplomacy, enabling countries to anticipate and respond to emerging threats and to maintain a strategic advantage over their rivals. With the increasing dependence on technology and the growing interconnectedness of the world, intelligence gathering has become even more challenging, and the need for sophisticated and innovative techniques has become more critical than ever before.

In today's world, the use of satellite imagery has become increasingly important in various fields, including national security, urban planning, and environmental monitoring. The detection and classification of aircraft using satellite imagery is one such application that has been proved innovative intelligence-gathering source and gives leverage to one state over another.

1.2 Problem Statement

Pakistan is a third-world underdeveloped country with constrained financial resources. Only available satellite PRSS-1 (Pakistan Remote Sensing Satellite-1) can capture high-resolution optical and remote sensing images of the Earth and is designed to support various applications, including natural resource management, urban planning, disaster management, defense and security. Lacking a standalone automatic system primarily designed for reconnaissance of crucial intelligence assets has created a great void in the modern world of intelligence gathering. The following are the points which establish the problem statement.

1. There is no independent, autonomous system for Identifying and Classifying national assets.
2. Satellite Imagery is very expensive, and very few support resolutions which are required to identify concerned objects.
3. Manual work by Image analysts is a burden on the national exchequer.
4. Image analysts' manual labor has very low efficiency.

1.3 Proposed Solution

The primary goal of our proposed solution is to Automatically Detect and Classify national assets like aircraft using Google Earth's Satellite Imagery. According to the Google Earth blog, Google Earth updates its images once a month [1]. The free-to-use updated and high-resolution imagery will Detect and Classify Aircraft effectively.

1.4 Working Principle

The project primarily uses machine learning methods, automated browsers, and image processing principles. The project is divided into various modules, and each module is interwoven with the module after it. The following is a list of the modules:

- Datasets Organization
- AI models training and Fine-tuning.
- Browser Automation
- Output Extraction
- Integration
- GUI presentation

1.4.1 Datasets Organization:

A Dataset used for detecting and classifying aircraft needs to have a large number of images captured from overhead with different zoom levels. The dataset would need to be diverse, including different types of aircraft, such as commercial planes, military jets and unmanned aerial vehicles (UAVs), as well as varying weather conditions, lighting and backgrounds.

1.4.1.1 Custom Aircraft Dataset:

The project uses a Custom build dataset of various types of aircraft collected from various internet sources. The images were gathered, filtered and annotated to obtain the coordinates of the object of interest. This dataset would be used for training the object detection model.

1.4.1.2 Synthetic Aircraft Dataset:

For Classification Purposes, images with higher resolution and large numbers are required. The images of one class of aircraft are rare to be found on the internet. To deal with the scarcity of datasets, we synthetically generated images of an aircraft. The dataset would be used for training the Classification model.

1.4.2 AI models training and Fine-tuning:

The datasets prepared for Detection and Classification were fed to the Object detection model and Image classification model, respectively, through the application of machine learning techniques.

1.4.2.1 YOLO algorithm:

YOLO (You Only Look Once) is a real-time object detection algorithm that uses a single convolutional neural network (CNN) to simultaneously predict bounding boxes and class probabilities for each detected object in an image or video frame.

Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images. [3]

In addition to object detection, YOLO can be used for image classification tasks by modifying the network architecture to output class probabilities without the associated bounding boxes.

Our project uses the YOLOv5 algorithm to train the datasets. This prepares object detection and image classification models.

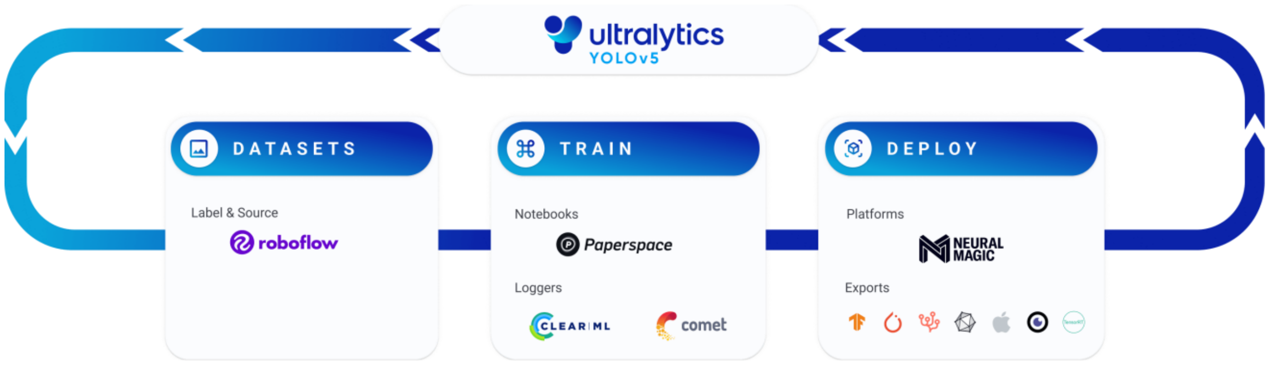


Figure 1 – Yolov5 architecture

1.4.3 Browser Automation:

To automatically scan an area in Google Earth, the browser will be automated using Selenium browser automation. It will also provide essential information like coordinates of the area.

1.4.4 Output Extraction:

The outputs are extracted based on the aircraft detected. The coordinates of detected aircraft in an image will be fed to the Image classification model along the image to find which aircraft it is. The detections, their classification result, along with other findings will be documented as a CSV file.

1.4.5 Integration:

The different modules are then integrated into one stand-alone entity. This stand-alone entity is essential for a compact solution.

1.4.6 GUI presentation:

The visual demonstration of the project is done through the aid of two GUIs (graphical user interfaces).

This information presented and documented includes the serial number of the detection, the snapshot in which the aircraft was found so it can be viewed later, bounding box coordinates, Score of Confidence, the classifier findings, which include the likelihoods of each class against the aircraft, latitude and longitude, a link to Google Earth that takes users directly to the area where the aircraft was found, and, lastly, the location details of the under-sampled area.

Two GUIs are:

1.4.6.1 GUI 1 (Multi Screen for Live Detection)

It would be for a user manually browsing through Google Earth, and the main screen will be projected to another screen in which the detections of the area will be shown. It will mitigate the missing of aircraft as prone to the human eye.

1.4.6.2 GUI 2 (Selenium based)

The coordinates of the area to be scanned will be given, and it will automatically scan the area without human intervention. The session will log all the details mentioned above to a folder containing the captured images of the aircraft.

1.5 Objectives

1.5.1 General Objectives:

"To build an innovative state-of-the-art software powered by Machine Learning (ML) and Browser automation techniques, providing an autonomous tool to Detect and Classify Aircraft with greater efficiency and productivity."

1.5.2 Academic Objectives:

- Development of an accurate and efficient algorithm for automatically detecting and classifying aircraft.
- Addressing a real-world problem
- To implement Machine Learning, Image Processing techniques and simulate the results.
- To increase productivity by working in a team
- To design a project that contributes to the welfare of society and National Security

1.6 Scope

This project finds its scope in intelligence gathering using overhead imagery. It has been developed in such a way that it is a general-purpose software to identify any object for which the model has been trained, and it will scan the area for detection using Google earth.

Its applications include:

- **Early warning** for movements of aircraft in enemy Territory
- The detection of foreign aircraft on hostile bases
- Identify enemy **ORBAT** (Order of Battle)
- **Pre-emption** against possible enemy strikes
- Counter nefarious enemy intentions.

1.7 Deliverables

1.7.1 Eagle View

It gives an eagle view to observe and gather intelligence using a combination of image processing, machine learning and browser automation techniques.

1.7.2 Object of interest:

It can detect the object of interest by using the same combination of image processing and machine learning techniques. By detecting the object of interest, we mean detecting any kind of object. The limitation is only the dataset of that object, for which also a solution has been given in the form of Synthetically generation.

1.7.3 Data Analysis Report:

The project will generate a comprehensive report detailing the analysis of the data collected during the aircraft detection and classification process. The report will include detailed information on the accuracy of the detection and classification system, as well as any areas for improvement.

1.8 Relevant Sustainable Development Goals

The project falls under **SDG 9**

“Support domestic technology development, research and innovation in developing countries.”

By developing software that utilizes cutting-edge technology such as satellite imagery, image processing and machine learning techniques, this project can contribute to the advancement of infrastructure and innovation in the field of transportation and surveillance. Overall, the project can have a positive impact on the achievement of the SDGs by promoting the use of innovative and sustainable technologies in the field of intelligence and surveillance.

The project is sponsored by the Intelligence Agency, which will indeed be used for the enhancement of National Security.

1.9 Structure of Thesis

Chapter 2 contains the literature review and the background and analysis study this thesis is based upon.

Chapter 3 contains the organization of Datasets

Chapter 4 contains the training and evaluation of AI Models

Chapter 5 describes GUIs in detail

Chapter 6 contains the conclusion of the project.

Chapter 7 highlights the future work of this project.

Chapter 2: Literature Review

The introduction of a novel product involves the modification and enhancement of features that were previously present in similar products. The process of conducting a literature review holds significant importance in the progression of an idea towards the creation of a novel product. Similarly, in the context of product development, conducting a comprehensive analysis of similar projects is imperative. The present study is segmented into subsequent sections.

- Existing solutions
- Research

2.1 Existing Solutions

There are several existing solutions to this problem. However, all are hardware-based systems like ADS-B: Automatic Dependent Surveillance-Broadcast (ADS-B), Synthetic Aperture Radar (SAR), Electro-Optical (EO) Sensors and Multi-Spectral Imaging.

No existing solution employs Convolutional Neural Networks (CNNs) to do this job.

2.2 Research

The literature review for Aircraft Detection and Classification using Satellite Imagery includes the following:

1. Research on object detection and classification techniques: This includes research on YOLO (You Only Look Once) object detection technique and other deep learning models that are commonly used in object detection and classification.
2. One Stage Vs Two Stage Detectors: Whether to do detection and classification in one stage or employ separate models for these tasks.
3. Studies on satellite imagery: This includes research on different types of satellite imagery, such as optical and SAR (Synthetic Aperture Radar), and their applications in object detection and classification.
4. Applications of object detection and classification in defense and security: This includes studies on the use of object detection and classification in defense and security applications, such as detecting and identifying military assets and monitoring the movement of troops.
5. Studies on integrating satellite imagery and other data sources: This includes research on integrating satellite imagery with other data sources, such as aerial imagery, ground-based sensors, and social media data, to improve object detection and classification.
6. Studies on the use of object detection and classification in disaster management: This includes research on object detection and classification to identify and monitor areas affected by natural disasters, such as floods, earthquakes, and wildfires.

2.2.1 Research on object detection and classification techniques:

Deep learning neural networks, known as convolutional neural networks (CNNs), are frequently employed for image and video processing. CNNs are built to automatically learn hierarchical representations of visual information from raw pixel input. They are inspired by the structure of the visual cortex in humans and animals.

CNNs have emerged as a popular and successful method for image classification, object recognition, and segmentation applications due to their capacity to learn and extract features from raw data automatically.

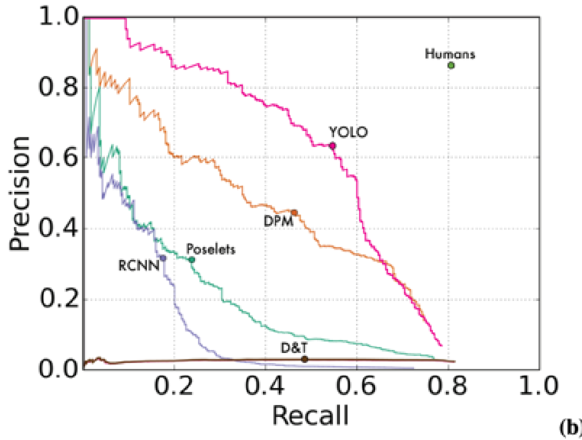
2.2.1.1 Object Detection Models

Machine learning models called object detection models are created to find and identify things of interest in images and videos. Object detection models include details on the location and limits of specific items inside an image, unlike image classification models, which merely assign a label to the entire image.

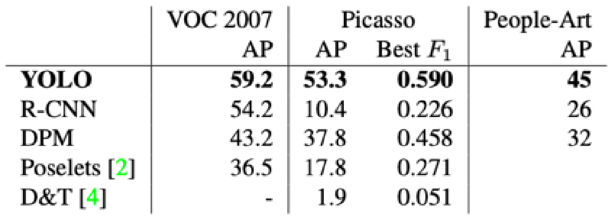
There are various kinds of object detection models, including more established deep learning models like region-based Convolutional Neural Networks (R-CNN), Single Shot Detectors (SSD), and You Only Look Once (YOLO), as well as more conventional computer vision methods like template matching and Haar cascades.

2.2.1.2 Comparison with other models

The problem this project aims to solve needs a state-of-the-art Object Detection model which can detect aircraft in the images. The performance of various Object detection models was analyzed, and YOLOv5 stood first. It had greater accuracy and inference time in comparison with other models.



(a) Picasso Dataset precision-recall curves.



	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

Figure 2a – Performance of different models

Yolov7, which was released in July 2022, performs exceptionally well, but hitherto it had some bugs related to running on Windows OS; it worked perfectly well on Linux-based OS. So, we worked with Yolov5.

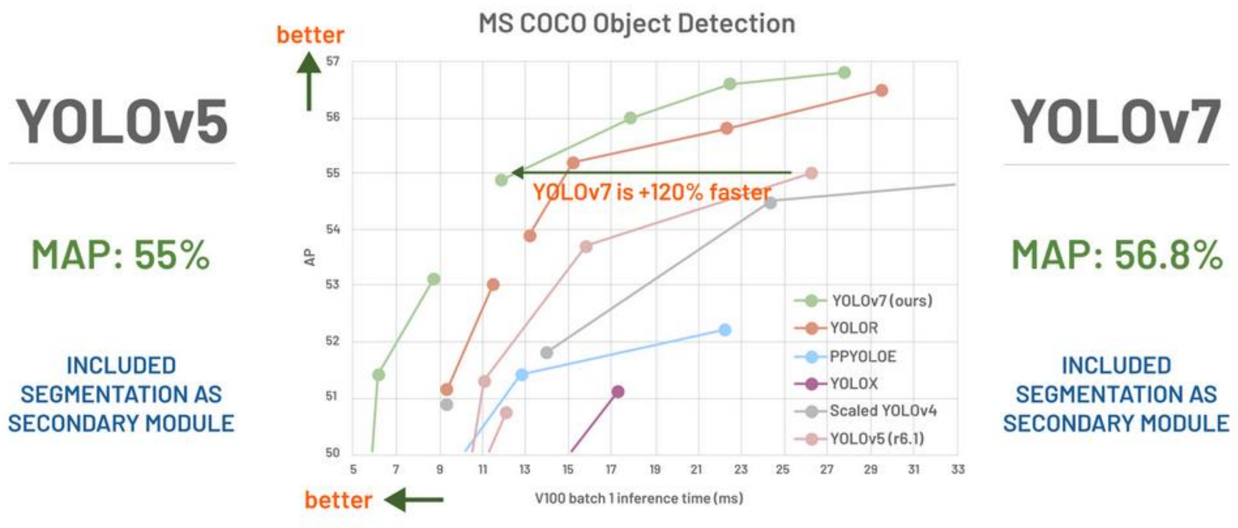


Figure 2b – Performance comparison of YOLOv5 and YOLOv7

2.2.1.3 Image Classification Models

ML Models for categorizing and identifying objects and situations in images are called "Image classification models". These models are tested on a collection of labelled image datasets, where each image has been given one or more class labels that describe its content.

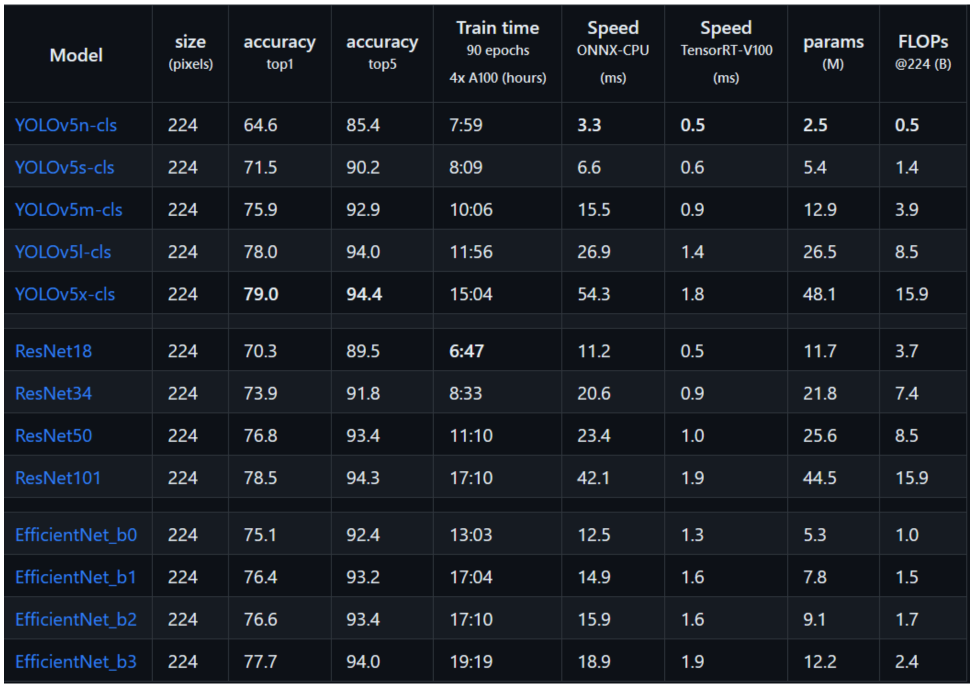
Convolutional neural networks (CNNs), a type of deep learning neural network, are frequently used in image classification models to examine an image's visual characteristics and determine which class it belongs to. Each layer in the processing process retrieves progressively sophisticated information from the input image, which is processed through several layers. The image is then categorized into one or more pre-established categories using these qualities.

Object identification in images and videos, medical imaging, driverless cars, and security and surveillance systems are just a few of the many uses for image categorization models. Due to improvements in deep learning techniques and the accessibility of vast datasets for training these models, they have improved in accuracy and efficiency.

2.2.1.4 Comparison with other models

YOLO is a one-stage detector; it detects the object and classifies it. However, to have greater accuracy, we used a separate Image Classification model. Object Detector was used to detect Aircraft alone, and the Image Classification model was implied to classify the aircraft within various classes.

The performance of various image classification models was analyzed, and the Yolov5-Cls model showed promising results.



Model	size (pixels)	accuracy top1	accuracy top5	Train time 90 epochs 4x A100 (hours)	Speed ONNX-CPU (ms)	Speed TensorRT-V100 (ms)	params (M)	FLOPs @224 (B)
YOLOv5n-cls	224	64.6	85.4	7:59	**3.3**	**0.5**	**2.5**	**0.5**
YOLOv5s-cls	224	71.5	90.2	8:09	6.6	0.6	5.4	1.4
YOLOv5m-cls	224	75.9	92.9	10:06	15.5	0.9	12.9	3.9
YOLOv5l-cls	224	78.0	94.0	11:56	26.9	1.4	26.5	8.5
YOLOv5x-cls	224	**79.0**	**94.4**	15:04	54.3	1.8	48.1	15.9
ResNet18	224	70.3	89.5	**6:47**	11.2	0.5	11.7	3.7
ResNet34	224	73.9	91.8	8:33	20.6	0.9	21.8	7.4
ResNet50	224	76.8	93.4	11:10	23.4	1.0	25.6	8.5
ResNet101	224	78.5	94.3	17:10	42.1	1.9	44.5	15.9
EfficientNet_b0	224	75.1	92.4	13:03	12.5	1.3	5.3	1.0
EfficientNet_b1	224	76.4	93.2	17:04	14.9	1.6	7.8	1.5
EfficientNet_b2	224	76.6	93.4	17:10	15.9	1.6	9.1	1.7
EfficientNet_b3	224	77.7	94.0	19:19	18.9	1.9	12.2	2.4

Figure 2c - Comparison of yolov5-cls with other classification models

2.2.2 One Stage Vs Two Stages:

There are two distinct methods for object detection in computer vision: one-stage and two-stage.

You Only Look Once (YOLO), and Single Shot Detectors (SSD) are examples of one-stage detectors that are intended to accurately anticipate the class and position of objects in a single run through the network. They typically predict object classes and bounding box coordinates simultaneously using several convolutional layers and anchor boxes. For real-time applications requiring high frame rates, one-stage detectors are typically faster than two-stage detectors.

Object detection is done in two steps using two-stage detectors like region-based Convolutional Neural Networks (R-CNN) and their derivatives. Prior to processing the image, areas of interest (ROIs) that are likely to contain objects are found. The object class and bounding box coordinates are then determined by analyzing and categorizing these ROIs. Region proposal networks (RPNs) are frequently used in two-stage detectors to create ROIs and a separate CNN to categorize the objects contained in each ROI. Although two-stage detectors are slower and require more calculations than one-stage detectors, they are typically more accurate.

In conclusion, although two-stage detectors are more accurate but slower, one-stage detectors are faster but less accurate. Depending on the needs of the application, such as the necessary accuracy and speed, one must choose between these two options.

To have greater accuracy for Aircraft detection Yolov5 object detection model was selected, and for the Classification of aircraft separately trained Yolov5-Cls Image Classification model was used. Due to fewer data on Aircraft classes, it was not possible to train only the object detection model to accurately detect and classify aircraft. Separating the process yielded greater accuracy as detecting the aircraft was the only job for the object detection model, and classifying it was for the Classification model. It also made the project general-purpose to integrate different models of various objects where scarcity of Dataset is an issue.

Chapter 3: Datasets

A dataset is defined as a compilation of information utilized for training, testing, and assessing machine learning models. A dataset is a collection of various types of data, including but not limited to images, text, numerical values, or a combination of these.

Labelled datasets are a common practice in machine learning, wherein each data point in the dataset is associated with one or more class labels or target values. These labels or values serve as a reference for the machine learning model to generate an output or prediction for the given data point.

3.1 Custom real-world Aircraft Dataset

The significance of large and varied datasets in the training of machine learning models cannot be overstated. The size and quality of a dataset can significantly influence the performance of a machine-learning model, thereby enhancing its accuracy and robustness. The preparation and refinement of data are typically necessary to guarantee the uniformity, precision, and appropriateness of the dataset for utilization in machine learning algorithms.

As the project was to be deployed in the real world, we had to gather as much real-world data as possible. So overhead imagery of aircrafts from various sources on the internet was acquired. The datasets were different in resolution, annotations format and some of them were not annotated. Un annotated dataset was annotated with the help of Roboflow.

3.1.1 Dataset Collection / Acquisition

The data we want to collect is entirely dependent on the issue we are seeking to resolve. The coaching knowledge cannot be surpassed by a machine learning algorithm. As a result, attention must be paid to gathering the best possible image data for the machine learning models. Computer vision models can be trained using one of the most extensive picture datasets and deep learning image data, thanks to a large image classification dataset. The range of services for collecting and annotating image data for use in machine learning and deep learning applications is extensive. The simplest technique to gather photographs for training the model is to use snipping software to crop images of aircraft, but there is surplus data in the area around the aircraft that could skew the model's results.

The training dataset is contaminated with extra data, or "Outliers," which are not needed to train the model. Figure 3a below displays an illustration of an outlier.

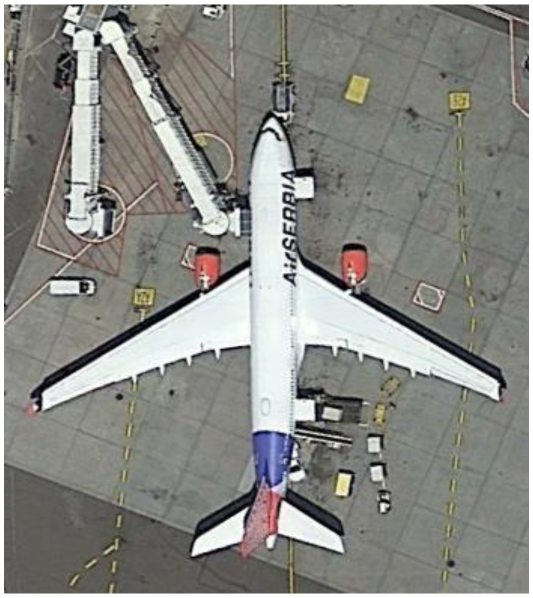


Figure 3a: Image with outliers.

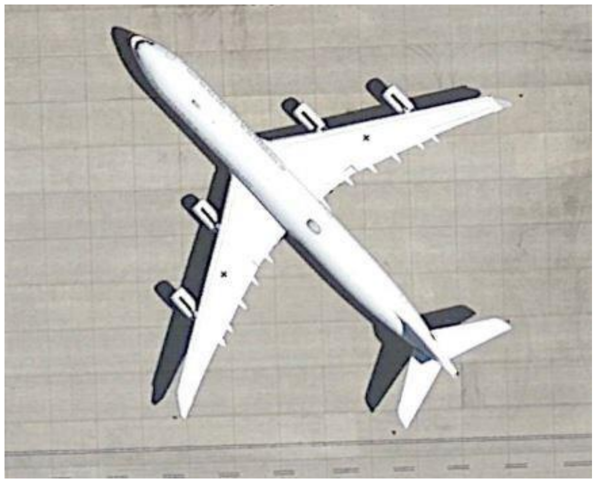


Figure 3b: An image without outliers.

3.1.2 Data Munging and Feature Extraction (Image- Preprocessing)

Data pre-processing, also known as data munging or data cleaning, is an essential step for machine learning engineers. Most ML engineers devote a significant amount of time to it before developing the model. Software like Photoshop and other similar programs can be used for data preparation. Outlier detection, missing value treatments, and removing undesired or noisy data are a few examples of data pre-processing.

Like that, "image pre-processing" refers to actions on images that are performed at the most basic level of abstraction. Pre-processing aims to improve the picture data by reducing unwanted distortions or enhancing specific visual properties necessary for subsequent processing and analysis tasks.

The following is a list of the four categories of image pre-processing techniques.

1. Brightness adjustments and pixel brightness changes
2. Transformations of Geometry
3. Image Segmentation and Filtering
4. Image restoration and Fourier transformation.

In our case, we focused on the following:

- Noise Removal
- Sharpening
- Removal of Outliers

3.1.2.1 Noise Removal

Digital photographs are susceptible to several kinds of noise. Errors in the picture capture process lead to pixel values that do not accurately reflect the accurate intensities of the actual scene, which is what is known as noise. Depending on how an image is produced, noise can be added in several different ways. For instance: If a picture is captured from a satellite, it may be blurred because of clouds and unfavorable weather.

The noise would come from the film grain if the image was scanned from a film photograph. In addition, the film's condition or the scanner itself may have caused noise.

One can eliminate kinds of noise using linear filtering. For this use, some filters, including averaging or Gaussian filters, are suitable. An averaging filter, for instance, can be used to reduce image noise caused by grain. Local differences brought on by grit are lessened since each pixel is adjusted to the average of the pixels in its immediate vicinity.

3.1.2.2 Sharpening

Resolution and acutance are two variables that together makeup sharpness. The resolution is uncomplicated and objective. It simply refers to the image file's size in pixels. The more pixels an image has, the better its resolution and the sharper it can be, all other things being equal. Acutance is slightly more challenging. It is a purely arbitrary measurement of edge contrast. The acutance has no unit.

How clearly defined the details in a picture are, especially the little elements, determines how sharp the image is.

Therefore, sharpening is a method for making a picture appear sharper. Acutance must be increased to increase perceived sharpness. Edge contrast must be added if you want your image to appear sharper.

3.1.2.3 Removal of Outliers

Finding outliers is a difficult undertaking. By looking at uncertainty measurements, outliers can be found. However, there are various kinds of outliers. An outlier, for instance, can be a sample that deviates from the norm. For instance, if we want to differentiate planes from the dataset and enter the data with vehicles and tunnels around the plane, we may get "outliers" since the class estimate is unclear. Examples of the outliers were supplied in figures 3a and 3b for reference. The outliers were previously described in depth in section 2.2.

3.1.3 Labelling of Dataset

The model being used for Object Detection is YOLOv5; it is machine learning under supervision. Therefore, labelling the data was required before training the model on the data. For tagging photos and generating bounding boxes around the object of interest in an image, there are numerous programs and tools available. The bounding box coordinates and class labels for each object in the image are included in each annotation.

Some of the famous annotation tools are as follows:

- SuperAnnotate
- VGG Image Annotation Tool
- Supervise.ly

- Labelbox
- Visual Object Tagging Tool (VoTT)
- LabelImg
- Roboflow

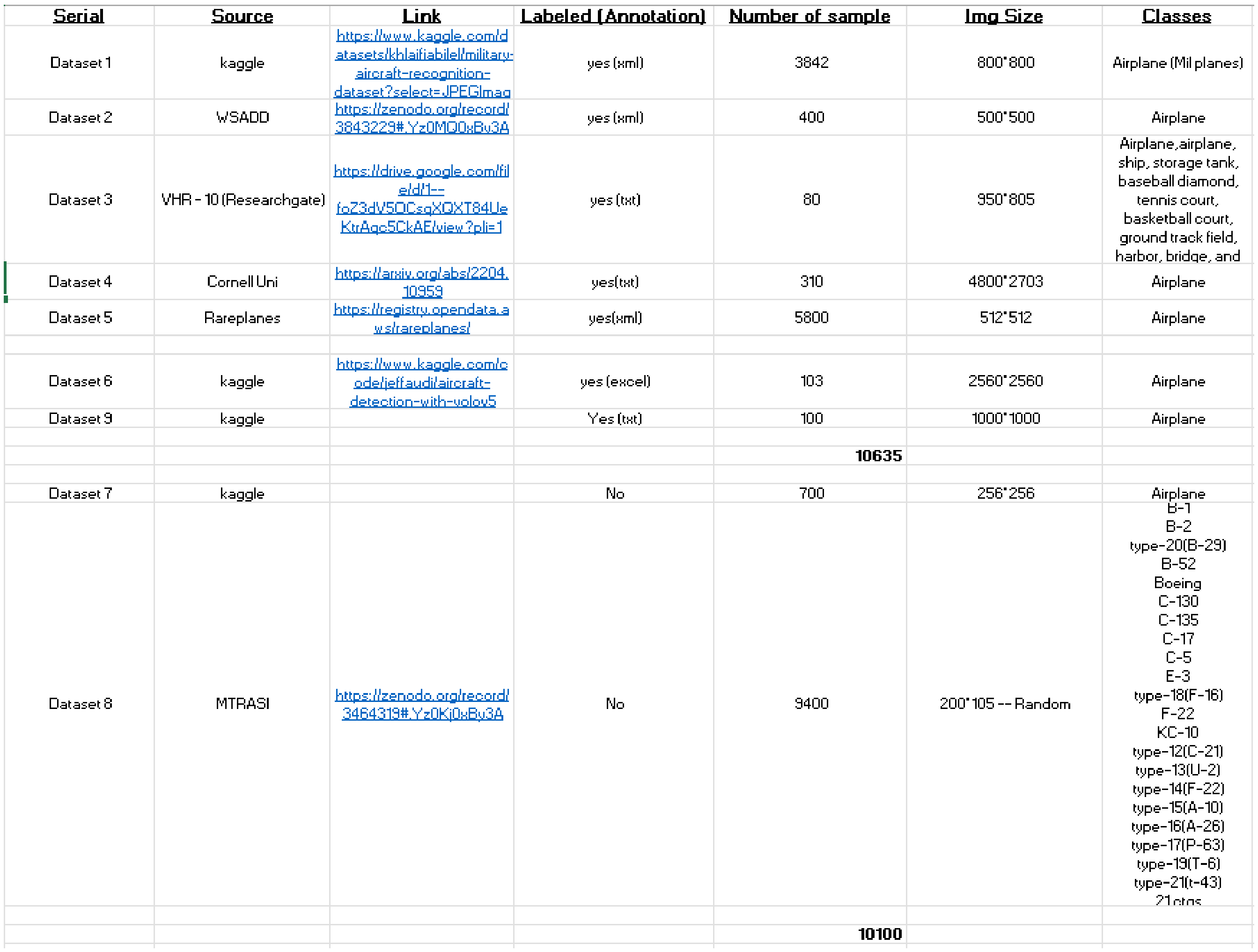
3.1.3.1 Why Roboflow?

The software used for annotation was Roboflow. The following are the key justifications for choosing it:

- A simple user interface
- Free to use with paid upgraded packages
- Cloud computing and storage
- Multiple forms of labelled data are available (which can be downloaded or used by creating a link to the prepared dataset on any web platform)

3.1.4 Acquired Datasets Details:

Below are the details of all acquired images from the internet. Manually cropped images have also been included in the unannotated images folder. They were annotated with the help of Roboflow.



Serial	Source	Link	Labeled (Annotation)	Number of sample	Img Size	Classes
Dataset 1	kaggle	https://www.kaggle.com/datasets/khlaifiabilel/military-aircraft-recognition-dataset?select=JPEGImag	yes (xml)	3842	800*800	Airplane (Mil planes)
Dataset 2	WSADD	https://zenodo.org/record/3843229#.Yz0MQ0xBy3A	yes (xml)	400	500*500	Airplane
Dataset 3	VHR - 10 (Researchgate)	https://drive.google.com/file/d/1--foZ3dV5OCsqXQXT84UeKtrAqc5CkAE/view?pli=1	yes (txt)	80	950*805	Airplane,airplane, ship, storage tank, baseball diamond, tennis court, basketball court, ground track field, harbor, bridge, and
Dataset 4	Cornell Uni	https://arxiv.org/abs/2204.10959	yes(txt)	310	4800*2703	Airplane
Dataset 5	Rareplanes	https://registry.opendata.aws/rareplanes/	yes(xml)	5800	512*512	Airplane
Dataset 6	kaggle	https://www.kaggle.com/code/jeffaudi/aircraft-detection-with-yolov5	yes (excel)	103	2560*2560	Airplane
Dataset 9	kaggle		Yes (txt)	100	1000*1000	Airplane
				10635		
Dataset 7	kaggle		No	700	256*256	Airplane
Dataset 8	MTRASI	https://zenodo.org/record/3464319#.Yz0Kj0xBy3A	No	9400	200*105 -- Random	B-1 B-2 type-20(B-29) B-52 Boeing C-130 C-135 C-17 C-5 E-3 type-18(F-16) F-22 KC-10 type-12(C-21) type-13(U-2) type-14(F-22) type-15(A-10) type-16(A-26) type-17(P-63) type-19(T-6) type-21(t-43) 21ctgs
				10100		

Figure 4 – Acquired Dataset Details

3.1.5 Annotations Conversion

The YOLOv5 object detection model uses an annotation file in .txt format. So, all the annotated dataset of different format was to be converted to a Txt format. Python codes were written to convert annotations from XML and Pascal VOC formats and to check whether the annotations were correct. All the annotations were checked, and any irregularity was dropped using the custom-written code mentioned in Annexure C.

3.1.6 Data Augmentation:

Data augmentation enlarges the dataset to include more training data and enhances model generalizability. Techniques for enhancing data include random cropping, flipping, scaling, and color adjustments. Yolov5 training API provides these options by default. Furthermore, they can be modified by hyperparameters file in the API.

3.1.7 Prepared Dataset:

To train the Object Detection Model, a real-world dataset of overhead images collected, as mentioned above, was segregated into three folders for training, validation, and testing uses using a custom Python code. Background images were incorporated into the dataset to increase the model's precision in differentiating between identical structures, such as aircraft and other similarly shaped items. These background images allowed the model to better learn the distinguishing characteristics of the intended objects and helped decrease false positives.

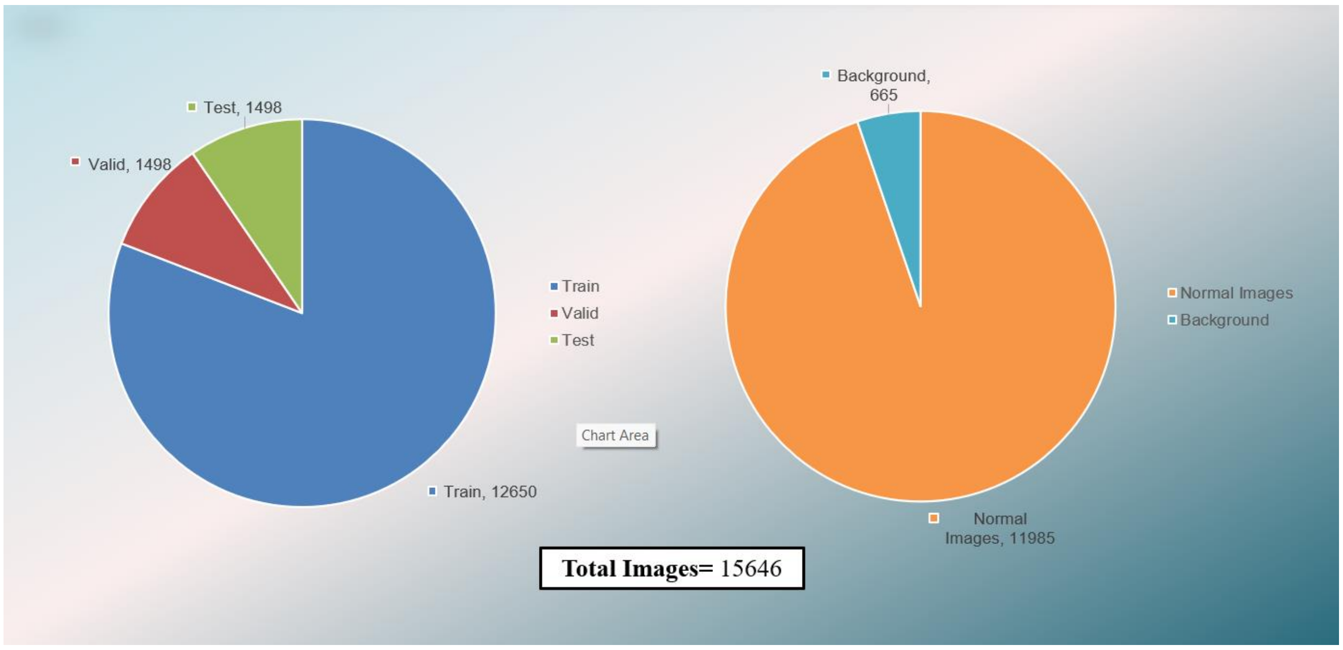


Figure 5a – Detection Dataset Split

3.2 Synthetically generated Aircrafts Dataset

The term "Synthetic Dataset" refers to a certain kind of dataset that is produced artificially, frequently using computer graphics or other simulation techniques. Machine learning models can be trained using synthetic datasets, which have various advantages over real-world datasets.

3.2.1 Significance:

1. Control: Synthetic datasets can be produced with complete control of the distribution of the data and the underlying ground truth, allowing for the customization of the generated data for certain use cases and scenarios.

2. Range: Because synthetic datasets can be developed and enhanced in any way that is appropriate for a certain purpose, they can offer a broader range of data than real-world datasets.

3. Privacy: Since synthetic datasets do not contain sensitive information about specific individuals, they can aid in preserving privacy, which is crucial in applications like healthcare or finance.

4. Scalability: Synthetic datasets may be produced in big numbers and with a wide variety, which can be helpful for deep learning models that need a lot of data to train.

5. Cost: The creation of synthetic datasets is less expensive than the time- and money-consuming process of gathering and classifying real-world datasets.

In cases when real-world datasets may not be available or adequate, synthetic datasets can be used to produce huge quantities of high-quality labelled data for machine learning model training. They also give a mechanism to create certain situations to test models, which helps with research and development.

3.2.2 Tools and Framework:

- Adobe Photoshop CC 2022
- Sketchfab
- Cinema 4D

3.2.3 Generation ways:

There are several ways to create synthetic datasets:

Generative adversarial networks (GANs): GANs are deep learning models that can generate new data by learning the underlying distribution of the data. GANs are commonly used in image and video generation tasks and can produce highly realistic synthetic data.

Rule-based methods: Rule-based methods involve generating data based on specific rules and constraints. For example, a synthetic dataset of floor plans for buildings could be generated based on specific architectural rules and design principles.

Simulation: Simulation involves creating a virtual environment that mimics real-world scenarios to generate synthetic data. This approach is commonly used in robotics and autonomous vehicle applications, where it is difficult to obtain real-world data.

Hybrid methods: Hybrid methods involve combining multiple approaches to generate synthetic data. For example, a hybrid method could involve using GANs to generate images and then using data augmentation to create variations of the generated images.

Based on the required requirements of Our Project, the techniques we employed used to create a synthetic dataset were Using Data Augmentation and Simulating the Virtual Environment for an Aircraft.

3.2.4 Dataset Preparation:

Adobe Photoshop CC 22 was used to prepare the dataset.

- For synthetic dataset generation, 3d Models of different aircraft were downloaded from sketchfab.com and imported into **Cinema 4d,** where materials and shaders were applied to the model,
- Using **Adobe Photoshop,** the finalized models were then color corrected and oriented to replicate overhead imagery of an aircraft at an airbase. Desired Shadows in reference to the lighting position were set, and compositing elements were added to the image to add photorealism,
- Samples of Random background images were taken, keeping in mind the different geographical aspects needed for precise detection and training of the classifier model.
- The aircraft in the images were then Rotated, Scaled and Skewed at different orientations to cater for all possible conditions and augmentations.

- A Photoshop Composer script was created and used to randomize all the different planes, orientations and backgrounds in various combinations, which generated a dataset of 3040 images of 8 different aircraft.

After the generation of the dataset, it was split into the format of folders supported by yolov5 using code in Annexure E. As it is an image classification model, they do not require separate annotation files. All images of one type of aircraft are inserted in a folder with its name.

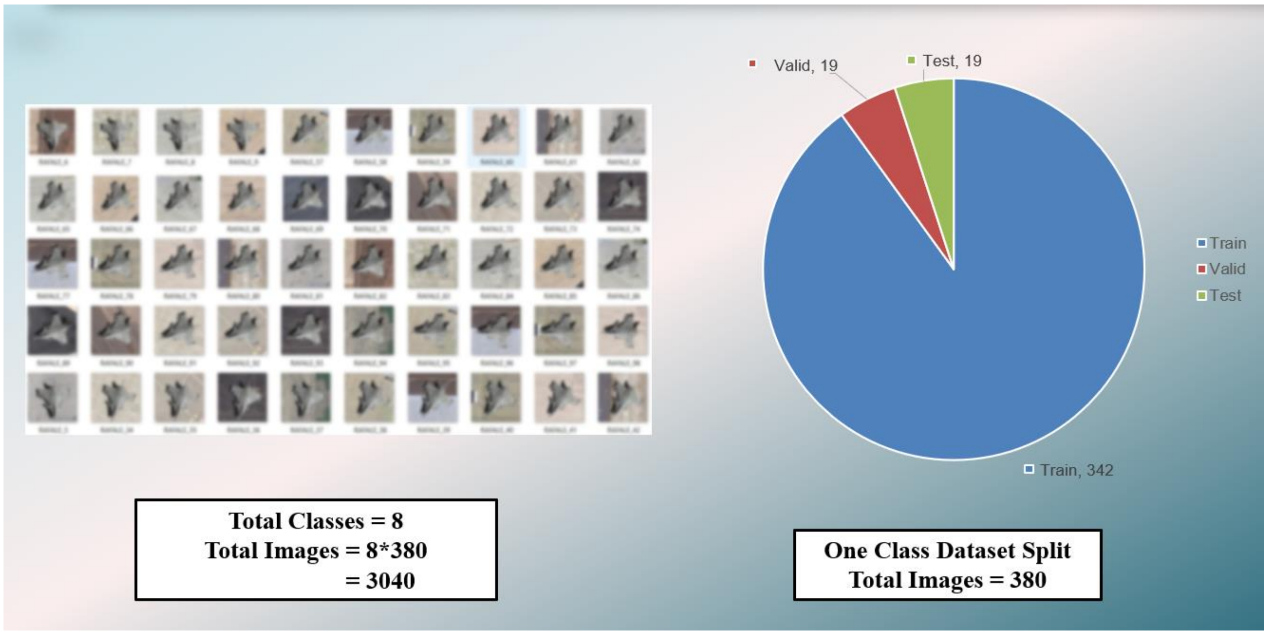


Figure 5b – Classification Dataset Split

3.3 Challenges:

The following challenges were faced while preparing datasets:

- Non-availability of a large dataset. Smaller datasets were combined to make a bigger one.
- Non-Uniformity in ground truth format. All of them were converted to yolo supported format by our custom-written codes.
- Formatting of unlabeled data.
- Detection vs Classification Accuracy / Inference Speed tradeoff
- No available datasets of a specific aircraft

Chapter 4: Training AI Models

As explained in detail above in the Research section, we are using Yolov5 for the generation of Object Detection and Image Classification models.

4.1 Training Object Detection Model:

The steps involved in training Object Detection Model using Yolov5 and the prepared dataset are given below. Before that, some concepts have been explained.

4.1.1 Training Parameters:

Training parameters for the YOLOv5 object detection model determine how the model learns and adjusts its weights during the training process. Here are some key training parameters for YOLOv5:

- **Batch size:** This parameter determines the number of images that the model will process in each training iteration. Larger batch sizes can help to speed up the training process but require more memory, while smaller batch sizes are slower but require less memory.
- **Learning rate:** The learning rate determines the step size that the optimizer takes during the training process to update the model weights. A higher learning rate will

result in larger weight updates and faster convergence but can also lead to instability and divergence.

- **Weight decay:** This parameter determines the amount of regularization applied to the model weights during training, which can help to prevent overfitting. A higher weight decay will result in more regularization, while a lower weight decay will result in less regularization.
- **Epochs:** determine the number of times the entire training dataset is used to train the model. Increasing the number of epochs can help to improve the model's performance but also increases the risk of overfitting.
- **Optimizer:** The optimizer determines the algorithm used to update the model weights during training.
- **Loss function:** The loss function measures the difference between the predicted and ground truth bounding boxes and class labels during training.

Optimizing these training parameters is critical for achieving the best performance of the YOLOv5 model. It is important to experiment with different values of these parameters to find the optimal combination for the specific problem being solved.

4.1.2 Pretrained Model Comparison:

YOLOv5 comes with several pre-trained models that can be used for various object detection tasks. These pre-trained models are trained on large datasets and can be fine-tuned for specific use cases. Here are some of the pre-trained models available in YOLOv5:

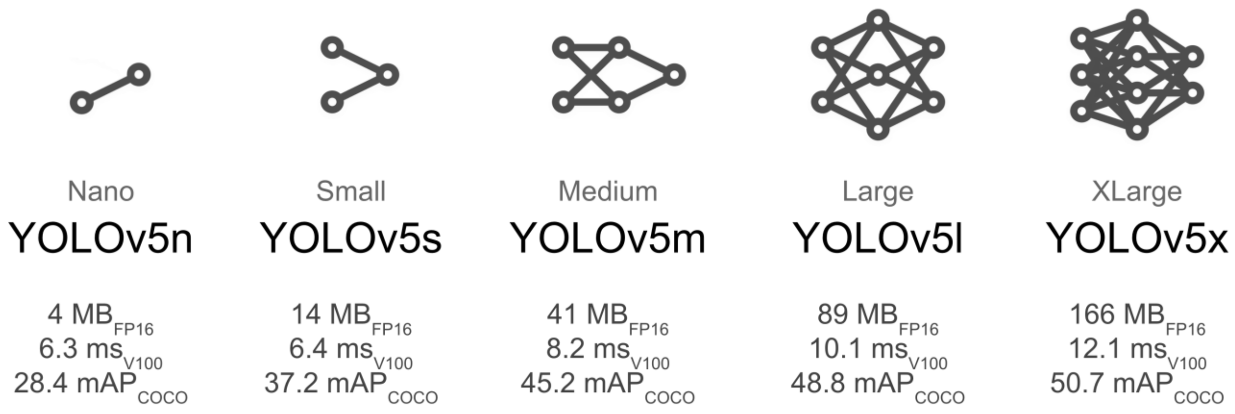


Figure 6 – Pre-trained Yolov5 Detection Models comparison

We chose yolov5s, the second smallest and fastest model available. Greater accuracy comes with a trade-off with inference speed. Large models are more accurate, but their detection speed is slow.

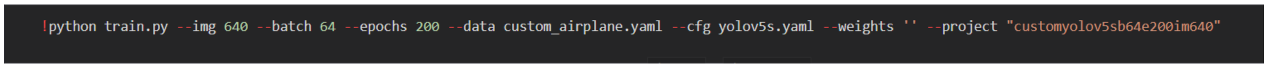
4.1.3 Training Setup:

For training a YOLOv5 object detection model, a High-end GPU-enabled Python environment is required. It can be done on a PC or web-based environment. Google Colab was used, which is a free cloud-based service provided by Google that enables users to run Python code and other popular machine learning frameworks in a web-based environment. It provides access to a high-end CPU, GPU, and TPU (Tensor Processing Unit).
It can run Jupyter notebooks. The other thing is Yolov5 API.

4.1.4 Yolov5 API:

The yolov5 GitHub repository provides an API to train object detection models.

The following command was used to train the model.



```
!python train.py --img 640 --batch 64 --epochs 200 --data custom_airplane.yaml --cfg yolov5s.yaml --weights '' --project "customyolov5sb64e200im640"
```

4.1.4.1 Img Size:

The image size was kept at 640*640. Resolutions of all images were plotted to check for a median value to which all images would be resized. The majority of images had a resolution of 640*640. The API automatically resizes all images.

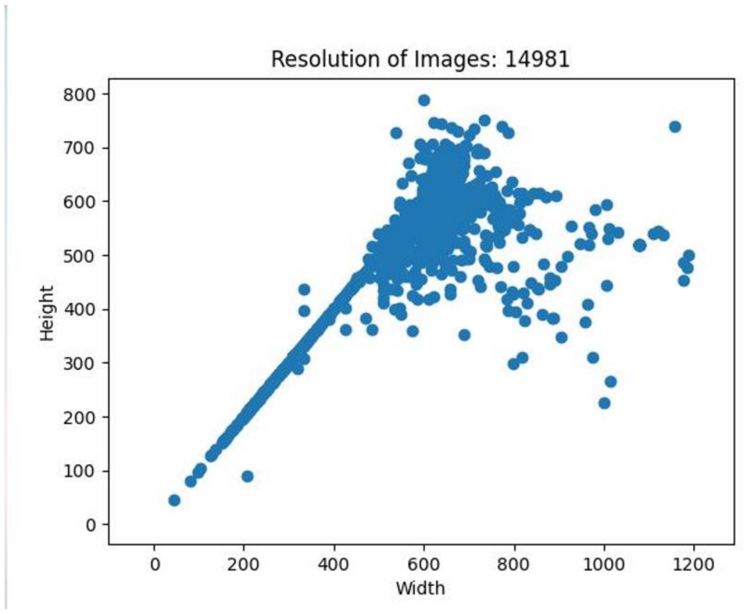
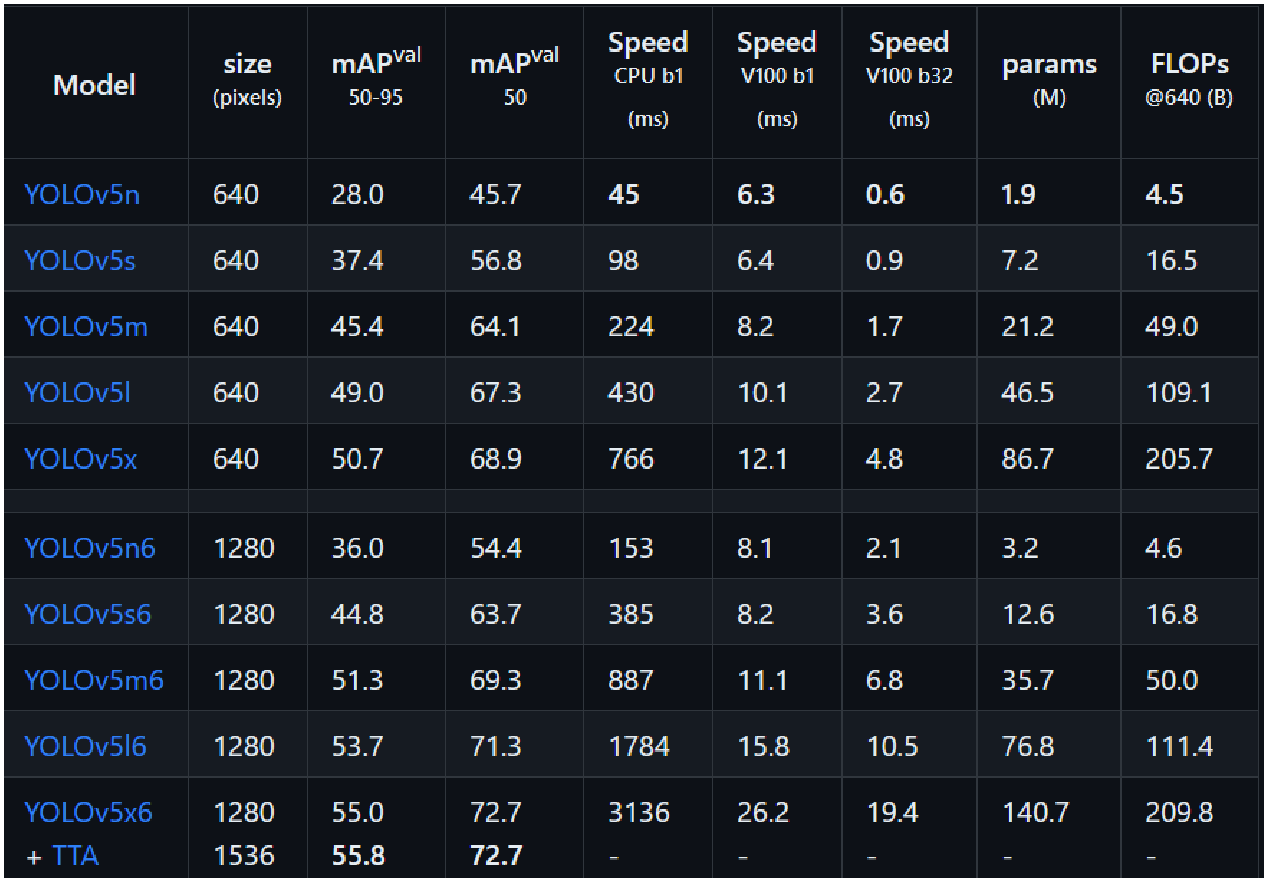


Figure 7 – Plot of resolution of images

The documentation also provides image sizes of the given pre-trained models to get maximum efficiency.



Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

Figure 8 – Performance comparison of yolov5 models

4.1.4.2 Batch Size:

The Google Collab provides a free GPU of 12 GB memory. A batch size of 64 utilizes a maximum of 11.7 Gb GPU memory. The batch size can be increased with higher GPU memory.

4.1.4.3 Epochs:

The epochs are set after hit and try; the maximum number is when the model starts overfitting. Overfitting is a common problem in machine learning where a model is trained too well on the training data to the point where it begins to memorize the training data instead of learning to generalize to new data. This leads to the model performing poorly on new, unseen data, despite performing very well on the training data.

4.1.4.4 Data:

This parameter takes a configuration file (.yaml), which contains folder paths of the train, valid and test folders and class names.

4.1.4.5 Cfg:

The cfg parameter takes an input of the configuration files of pre-trained weights. If we want to train the yolov5s model, the input will be “yolov5s.yaml”.

4.1.4.6 Weights:

This enables the algorithm to use a pre-trained model. If empty, it trains the model from scratch.

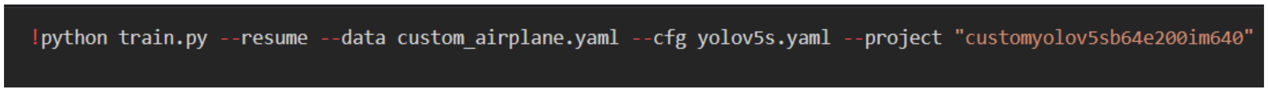
4.1.4.7 Project:

The name of the folder in which it will store all files related to training and final weights is also there.

4.1.4.8 Resume:

Google colab’s free session expires after some time. Training a large dataset is not an easy task. Luckily yolov5 offers a resume option which resumes the training from where it left off. When the session closes, it also deletes the files. As a workaround, Google Drive was mounted to Collab and yolov5 git was cloned to the drive so that the files would be saved even after the session closed. It enabled the use of resume functionality.

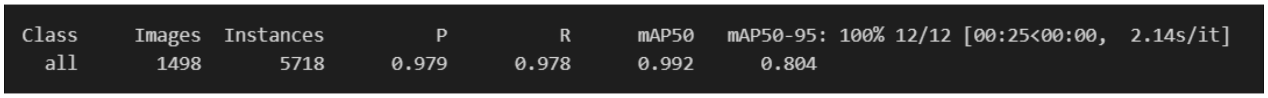
The following command was used to resume the training:



```
!python train.py --resume --data custom_airplane.yaml --cfg yolov5s.yaml --project "customyolov5sb64e200im640"
```

4.1.5 Training results:

The model was trained for 117 epochs and yielded 80% test accuracy.



```
Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 12/12 [00:25<00:00,  2.14s/it]
  all       1498       5718      0.979      0.978      0.992      0.804
```

The following results were plotted by our custom-written code mentioned in Annexure D, which presents various metrics.

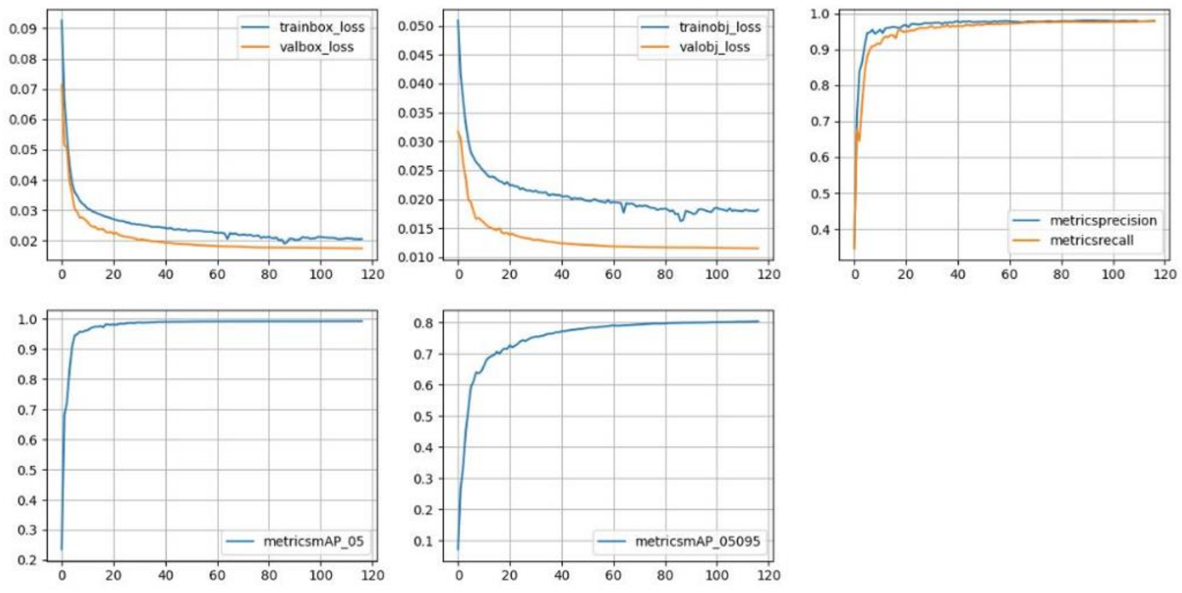


Figure 9a – Detection model training results plot

There are several metrics that are commonly used to evaluate the performance of object detection models like YOLOv5. Some of the most common metrics are:

1. Precision: Precision is the fraction of true positive detections out of all the positive detections. It measures how accurate the model's predictions are.

2. Recall: Recall is the fraction of true positive detections out of all the actual positives in the dataset. It measures how well the model can detect objects in the dataset.
3. Intersection over Union (IoU): IoU is a measure of how well the model can predict the bounding boxes of objects. It is the ratio of the intersection of the predicted bounding box and the ground truth bounding box to the union of both bounding boxes.
4. Average Precision (AP): AP is a metric that measures the accuracy of object detection models. It is calculated by computing the precision and recall for each object class at different confidence thresholds and then taking the average.
5. Mean Average Precision (mAP): measures the precision and recall of the predicted bounding boxes against the ground truth bounding boxes. MAP is usually calculated by averaging the AP (Average Precision) scores over different IoU (Intersection over Union) thresholds. mAP@0.5 (or mAP05095) refers to the MAP calculated with an IoU threshold of 0.5, which is one of the commonly used thresholds for object detection evaluation. It measures the average precision of the predicted bounding boxes that have an IoU of at least 0.5 with the ground truth bounding boxes.

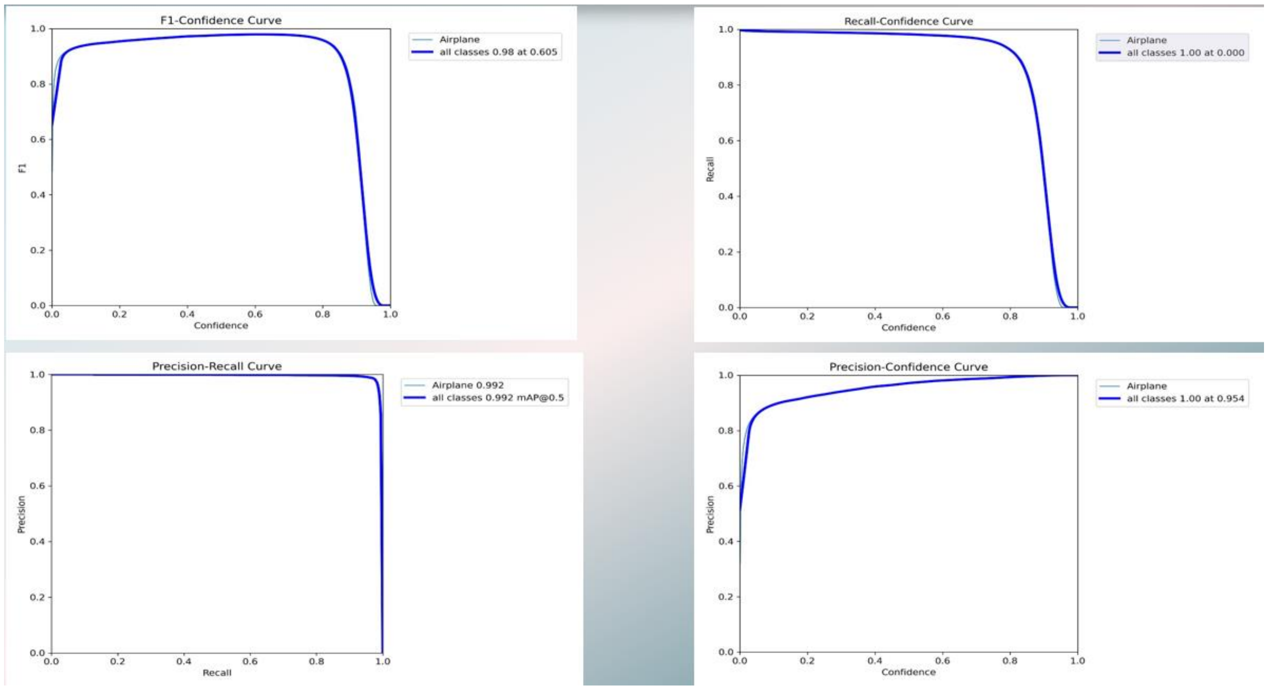


Figure 9b – Detection model training results curves

These are different types of curves that are commonly used to evaluate the performance of object detection models. Here's an explanation of each of these curves:

1. **F1-Confidence curve:** The F1 score is a commonly used metric to evaluate the overall performance of an object detection model. It is a combination of precision and recall. The F1-Confidence curve plots the F1 score against different confidence levels for the model's predictions. This curve can be used to determine the optimal confidence threshold that maximizes the F1 score.
2. **Recall-Confidence curve:** Recall measures the percentage of true positive detections that the model has correctly identified. The Recall-Confidence curve plots the recall against different confidence levels for the model's predictions. This curve can be used to determine the optimal confidence threshold that maximizes recall.

3. **Precision-Recall curve:** Precision is the percentage of true positive detections that the model has correctly identified out of all the detections it has made. The recall is the percentage of true positive detections that the model has correctly identified out of all the ground-truth objects in the image. The Precision-Recall curve plots precision against recall for different confidence thresholds. This curve can be used to evaluate the trade-off between precision and recall and to determine the optimal confidence threshold that balances the two.
4. **Precision-Confidence curve:** The Precision-Confidence curve plots precision against different confidence levels for the model's predictions. This curve can be used to determine the optimal confidence threshold that maximizes precision.

4.1.6 Conclusion:

All the metrics to judge the model’s performance have been explained in detail above. The model training yielded **80%** of test accuracy, as shown in figure 9a. F1-Confidence curves give a threshold of 0.60 with 98% surety. When integrating the model, the threshold can be set to 0.60, and it won’t miss real aircraft.

The model didn’t overfit because ample time was given to dataset preparation. Overfitting can occur when a model is too complex relative to the amount of training data available or when the model is trained for too many iterations or epochs. It can be prevented by using techniques such as regularization, early stopping, and data augmentation. But the dataset was itself diverse. It could have been trained more, but due to constrained resources, it was stopped at 117 epochs. Still, it yielded great accuracy.

The test and train loss were following a downward trend which means it could've been trained more on this dataset before overfitting could occur.

4.2 Training Image Classification Model:

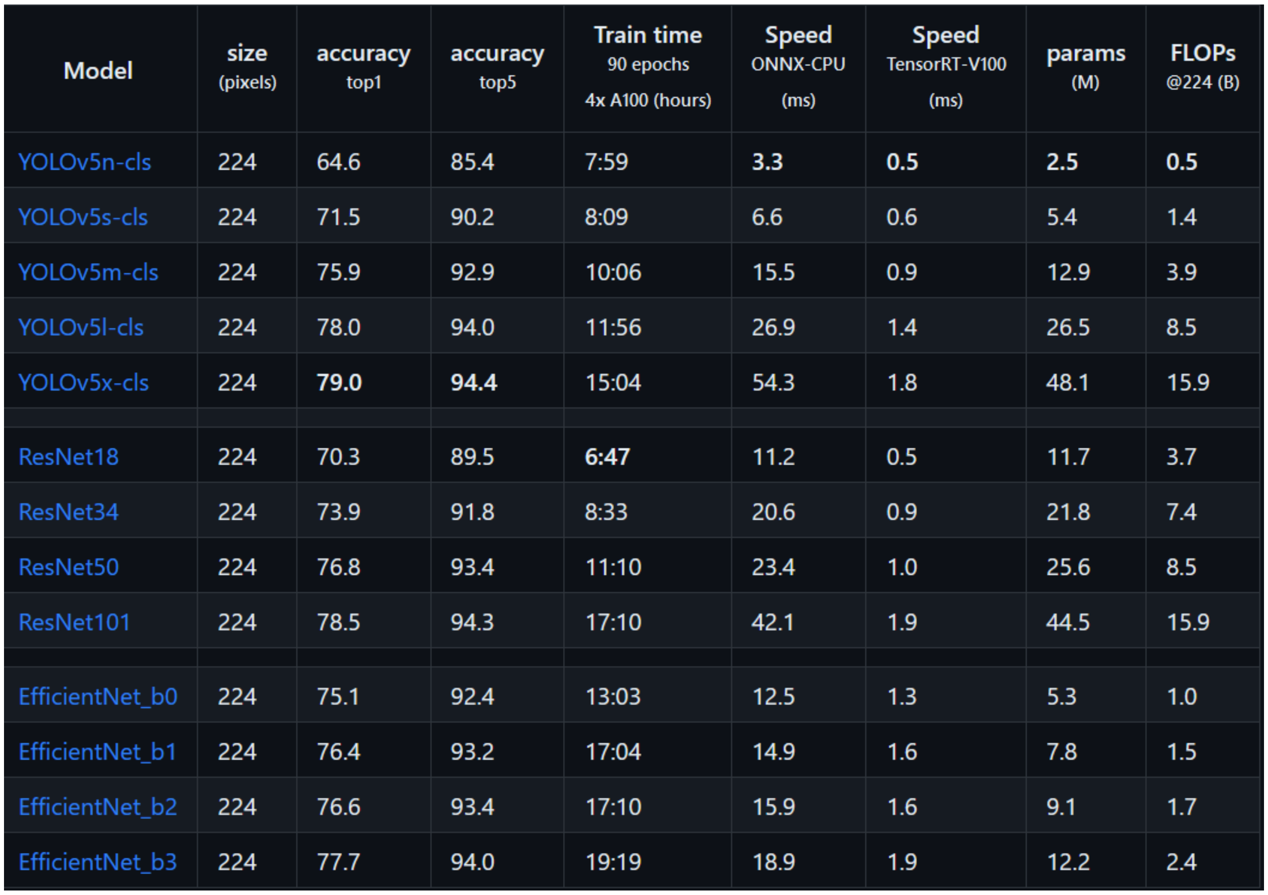
The steps involved in training an image classification model on a synthetic dataset are given below.

4.2.1 Training Parameters

The training parameters of image classification model training are as same as those of object detection, which has been discussed in detail in 4.1.1.

4.2.2 Pretrained models comparison

Following is the comparison of pre-trained yolov5-cls models. We used the yolov5x model as it was large enough to learn complex patterns of different aircraft. It also did not consume more significant resources to train and run inference.



Model	size (pixels)	accuracy top1	accuracy top5	Train time 90 epochs 4x A100 (hours)	Speed ONNX-CPU (ms)	Speed TensorRT-V100 (ms)	params (M)	FLOPs @224 (B)
YOLOv5n-cls	224	64.6	85.4	7:59	**3.3**	**0.5**	**2.5**	**0.5**
YOLOv5s-cls	224	71.5	90.2	8:09	6.6	0.6	5.4	1.4
YOLOv5m-cls	224	75.9	92.9	10:06	15.5	0.9	12.9	3.9
YOLOv5l-cls	224	78.0	94.0	11:56	26.9	1.4	26.5	8.5
YOLOv5x-cls	224	**79.0**	**94.4**	15:04	54.3	1.8	48.1	15.9
ResNet18	224	70.3	89.5	**6:47**	11.2	0.5	11.7	3.7
ResNet34	224	73.9	91.8	8:33	20.6	0.9	21.8	7.4
ResNet50	224	76.8	93.4	11:10	23.4	1.0	25.6	8.5
ResNet101	224	78.5	94.3	17:10	42.1	1.9	44.5	15.9
EfficientNet_b0	224	75.1	92.4	13:03	12.5	1.3	5.3	1.0
EfficientNet_b1	224	76.4	93.2	17:04	14.9	1.6	7.8	1.5
EfficientNet_b2	224	76.6	93.4	17:10	15.9	1.6	9.1	1.7
EfficientNet_b3	224	77.7	94.0	19:19	18.9	1.9	12.2	2.4

Figure 10 – Performance comparison of pre-trained yolov5-cls models

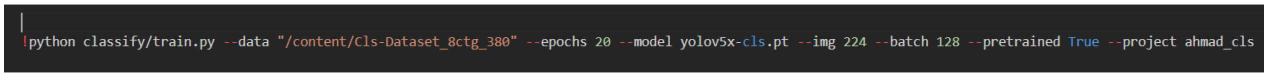
4.2.3 Training Setup

The same setup as mentioned in 4.1.3 was used.

4.2.3 Yolov5-Cls API

The API to train the classification model is also in the same GitHub repository.

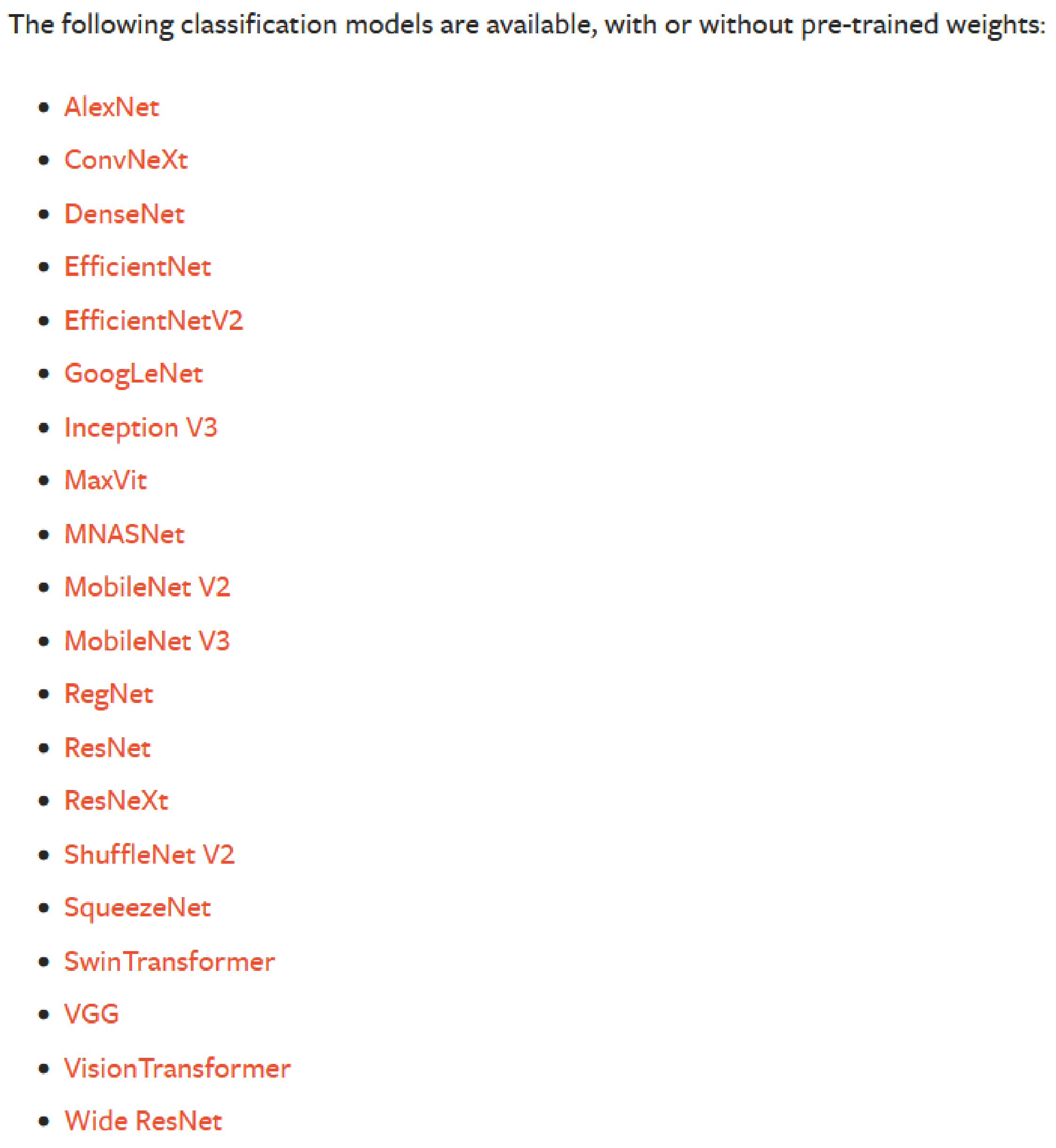
The following command was used to train the image classification model.



```
!python classify/train.py --data "/content/Cls-Dataset_8ctg_380" --epochs 20 --model yolov5x-cls.pt --img 224 --batch 128 --pretrained True --project ahmad_cls
```

The model parameter takes an input of pre-trained image classification model. We have used the pre-trained option now, as the dataset was not large enough for the model to be trained from scratch. The pre-trained model also reduces the time of training.

It supports all Pytorch image classification models and can be used to train any of them.

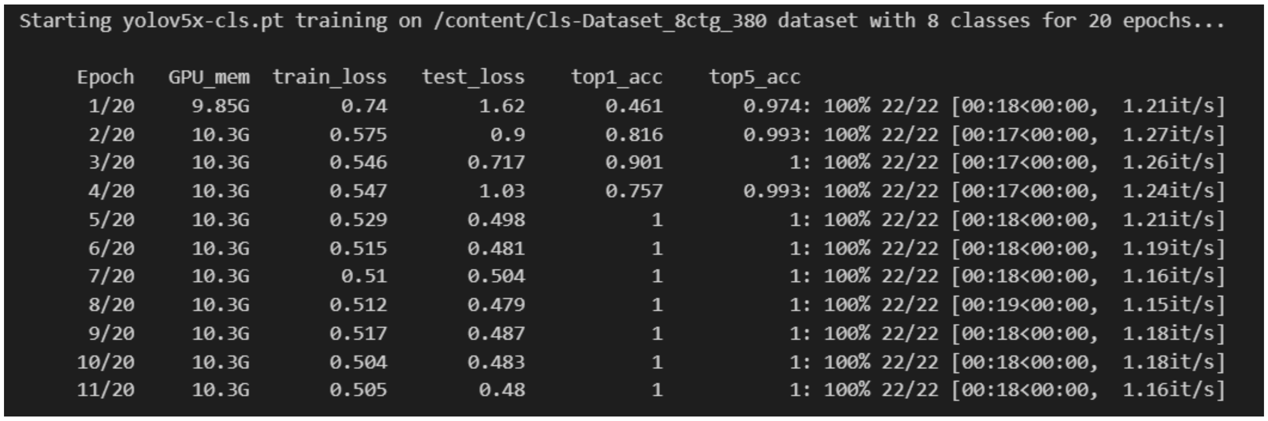


The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

4.2.4 Training Results

The classification API does not produce results and log files as object detection in which deep insights into the model's performance metrics are given. However, the following are the snapshots of training.



```
Starting yolov5x-cls.pt training on /content/Cls-Dataset_8ctg_380 dataset with 8 classes for 20 epochs...

     Epoch   GPU_mem  train_loss   test_loss    top1_acc    top5_acc
      1/20     9.85G        0.74        1.62       0.461       0.974: 100% 22/22 [00:18<00:00,  1.21it/s]
      2/20     10.3G       0.575         0.9       0.816       0.993: 100% 22/22 [00:17<00:00,  1.27it/s]
      3/20     10.3G       0.546       0.717       0.901           1: 100% 22/22 [00:17<00:00,  1.26it/s]
      4/20     10.3G       0.547        1.03       0.757       0.993: 100% 22/22 [00:17<00:00,  1.24it/s]
      5/20     10.3G       0.529       0.498           1           1: 100% 22/22 [00:18<00:00,  1.21it/s]
      6/20     10.3G       0.515       0.481           1           1: 100% 22/22 [00:18<00:00,  1.19it/s]
      7/20     10.3G        0.51       0.504           1           1: 100% 22/22 [00:18<00:00,  1.16it/s]
      8/20     10.3G       0.512       0.479           1           1: 100% 22/22 [00:19<00:00,  1.15it/s]
      9/20     10.3G       0.517       0.487           1           1: 100% 22/22 [00:18<00:00,  1.18it/s]
     10/20     10.3G       0.504       0.483           1           1: 100% 22/22 [00:18<00:00,  1.18it/s]
     11/20     10.3G       0.505        0.48           1           1: 100% 22/22 [00:18<00:00,  1.16it/s]
```

4.2.5 Conclusion

As the dataset was synthetically generated, it still gave an exceptionally good performance. Preparing a dataset which can give higher accuracies in real-world testing scenarios a lot of labor is required.

Anyone who wants to prepare a large, diverse dataset can follow the steps given in previous chapters and include images of all kinds of scenarios as it would have been in the case of a real-world collected dataset.

4.3 Challenges

Following engineering challenges were faced:

- Fewer data quantity and quality of classification dataset.

- Hardware constraints; training both models was computationally intensive and required powerful hardware. Resource constraints limited the complexity of the model, the size of the dataset and the length of the training process.
- Model interpretability: models can be complex and challenging to interpret. Understanding how the model makes predictions and identifying sources of errors was a challenging task.

Chapter 5: GUIs

To integrate both models and to solve the problem, two different Graphical User Interfaces were created.

5.1 GUI 1 (Multi Screen for Live Detection)

It would be for a user manually browsing through Google Earth. The main screen will be projected to another screen in which the detections of the area will be shown. It will mitigate the missing of aircraft as prone to the human eye.

5.1.1 Block Diagram:

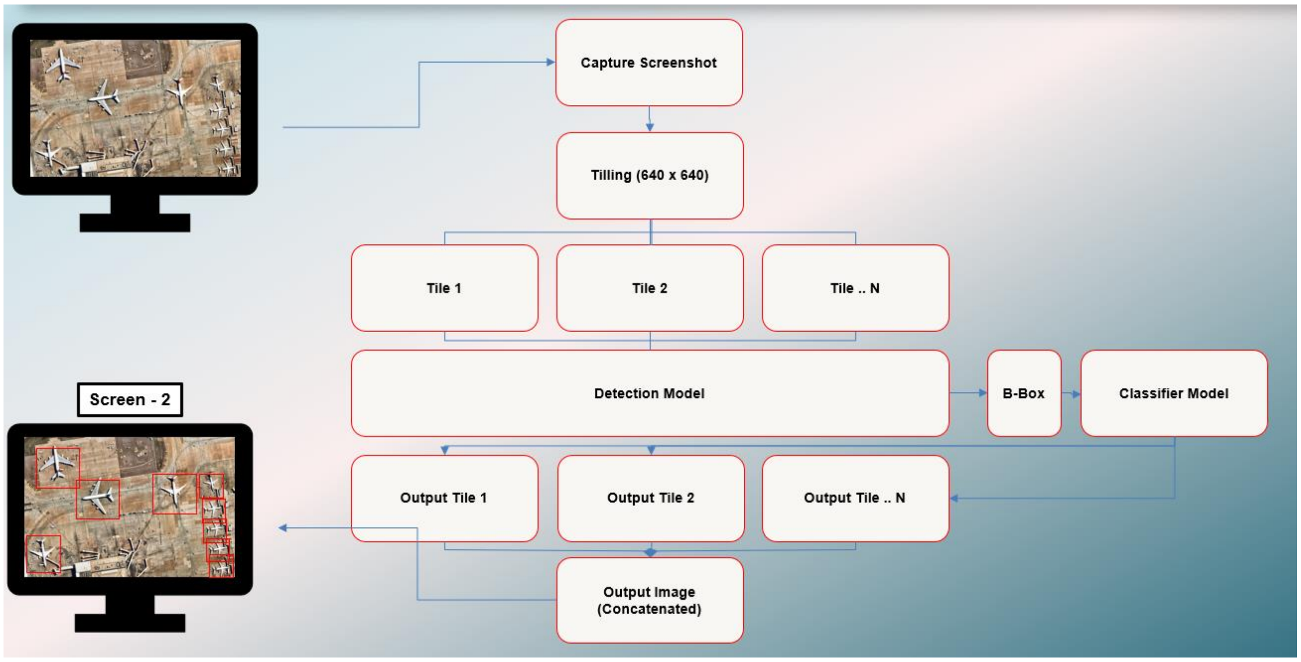


Figure 11 – Block Diagram of GUI 1

5.1.2 Concept:

This Python script uses the YOLOv5 object detection model and a custom classification model to detect objects and classify them in real-time using screenshots of a specified window.

1. The script first defines several helper functions, including one to pre-process the input image by cropping it and one to crop a bounding box around a detected object for classification.
2. The script then loads the YOLOv5 object detection model and sets the confidence threshold for non-maximum suppression.
3. If the user chooses to use classification, the script loads the custom classification model and its corresponding class names.
4. The script then enters a loop where it captures screenshots of the specified window, pre-processes them, and runs the object detection model on them to detect objects.
5. For each detected object, the script optionally crops a bounding box around it and runs the classification model on it to classify the object.
6. The script then draws bounding boxes around the detected objects and displays the object class and classification results (if classification is enabled) on the image. The output image is then displayed in a window.
7. The script continuously loops and captures screenshots until the user closes the output window.

Overall, the script provides a simple and effective way to detect and classify objects in real-time using deep learning models.

5.1.3 Libraries and helper functions:

The code imports several libraries to perform various operations:

- **os:** This library provides a way of using operating system-dependent functionality like reading or writing to the file system.
- **cv2:** This library is used for computer vision and image processing operations, such as image resizing, drawing shapes and text on images, and loading image files.
- **numpy:** This library is used for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with an extensive collection of high-level mathematical functions to operate on these arrays.
- **Torch:** This library is used for deep learning and machine learning operations. It provides several data structures for storing and manipulating large amounts of numerical data, as well as methods for defining, training, and evaluating machine learning models.
- **pathlib:** This library is used for working with file paths. It provides a way to represent file paths as objects, which makes it easier to work with file paths in a platform-independent way.
- **windowcapture (custom):** This is a custom module that provides a way to capture the current screen and return it as a NumPy array.

The helper functions defined in the code are:

- **pre_process_image:** This function crops the image to a specific size by slicing the NumPy array and returns the resulting image as a NumPy array.

- **crop_one_box:** This function crops a specific region of the input image using the coordinates of the bounding box specified by x1, y1, x2, and y2, resizes the cropped image to 224x224, and returns the resulting image as a tensor.
- **parse_detections:** This function takes the output of the object detection model, the current screenshot image, and the optional classifier model and draws bounding boxes and class probabilities on the input image. If a classifier model is provided, it also uses to predict the class probabilities of each object detected. The function returns the modified image as a numpy array.

5.1.4 Working:

The working of this script is explained in detail below:

- When started, it asks about whether to use Classifier for the classification of detected objects.
- It loads the object detection model and if selected also loads the image classification model.
- It sets the “model.conf” parameter, which is the NMS confidence threshold.
- Then it enters in a while loop.
- It gets a screenshot of the current screen using a custom-written code which gives an FPS of 40. It was necessary to have a fast speed of screenshots, or else it would have produced a significant lag on the projected screen.
- It preprocesses the image. Crops the extra screen of the browser other than google earth imagery. It is necessary because otherwise, it would confuse the model with unwanted similar-shaped icons and affect detection.

- It then divides the image of 1920*1080 resolution into tiles of 640*640 size. 640 resolution size was chosen as the model was trained on this size. Tiling was done because an image of greater size than the model was trained on would have to be resized to the training image's size. Greater resolution size images can be passed, but the detections are not as accurate. The pixel ratio of aircraft gets smaller, and the model is unable to detect them. So tiling was done so as not to disturb the pixel ratio and pass each portion of the area as a whole without resizing it.
- It then iterates between the tiles and passes the tile from the model, converting the detections of torch format into an array using pandas.
- Passes each detection into the "parse_detections" function along the tile image. It returns the image with bounding boxes drawn on them. The tile image is stored in precisely the place from where the tile was taken.
- If the classification is selected, the detected coordinates of the object are passed to the "crop_one_box" function to get the cropped image, and that image is passed to the classification model. It is an iterative process, and all detections are cropped. Certain necessary checks to prevent runtime errors are also inserted.
- After the iteration among tiles is complete, an output image with bounding boxes drawn is formed. It is projected to another screen using OpenCV.
- The script prints FPS on the terminal.

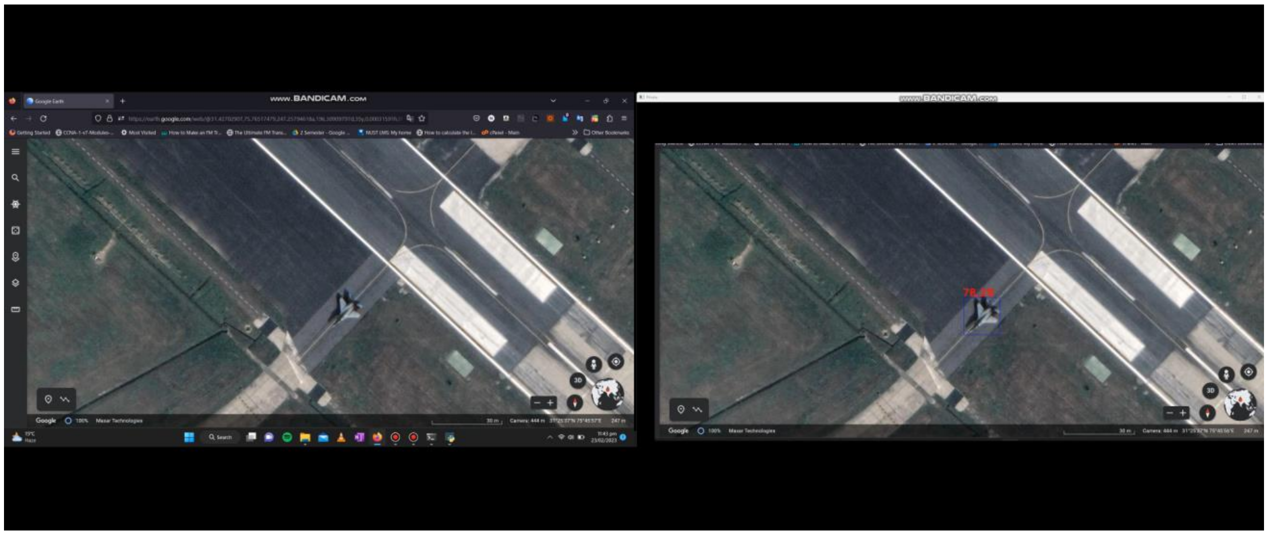


Figure 12 – GUI 1 Display Pane

As shown in the above image, the left screen is where the user is operating, and the screen is captured, passed through the model, and displayed on the right screen.

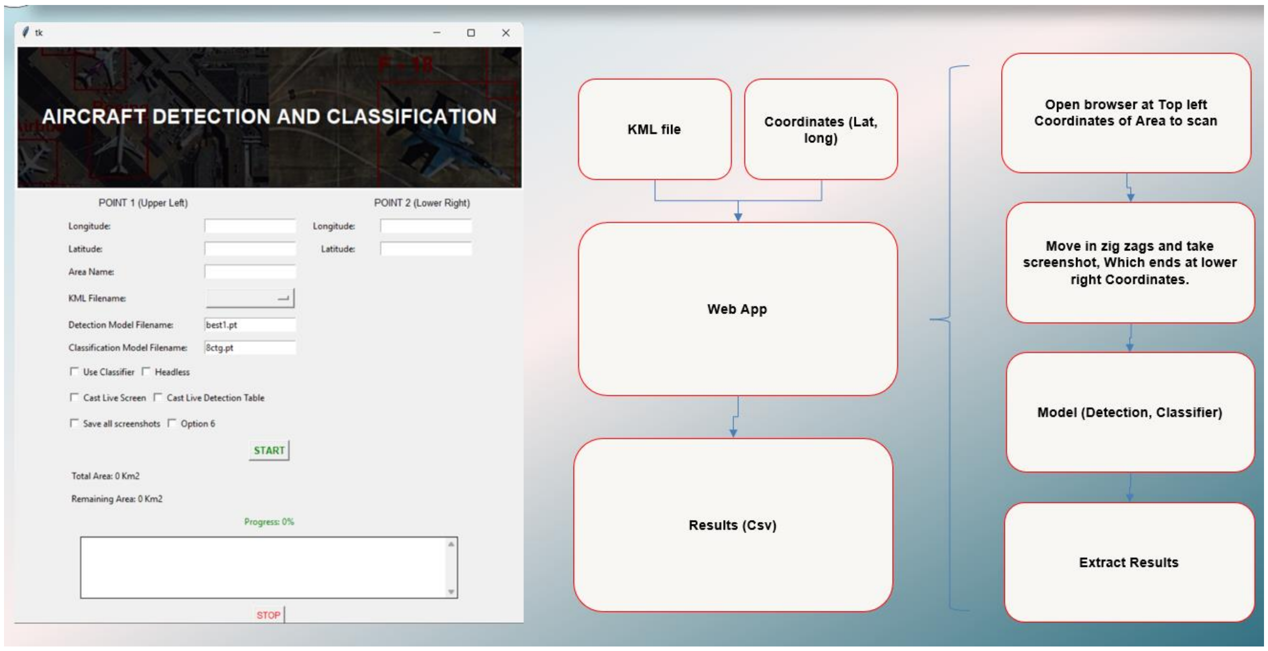
5.1.5 Conclusion

It is an object detection and classification application that can be used for general purposes. To detect any kind of object and shape while browsing through a computer, the internet or any application that displays on the screen, it can be used. The object of interest's model has to be trained, and interfacing is simple. This GUI does not solve the problem definition we have established above, as it requires manual operation. However, it was the requirement of agencies. The code has been provided in Annexure B.

5.2 GUI 2 (Selenium based)

This is the GUI that solves our problem and is a general-purpose product that can be used for any kind of object detection and classification autonomically. The coordinates of the area to be scanned will be given, and it will automatically scan the area without human intervention. The session will log all the details mentioned above to a folder containing the captured images of the aircraft and a CSV file.

5.2.1 Block Diagram



Detection Model Filename: best1.pt

Classification Model Filename: 8ctg.pt

Total Area: 0 Km2

Remaining Area: 0 Km2

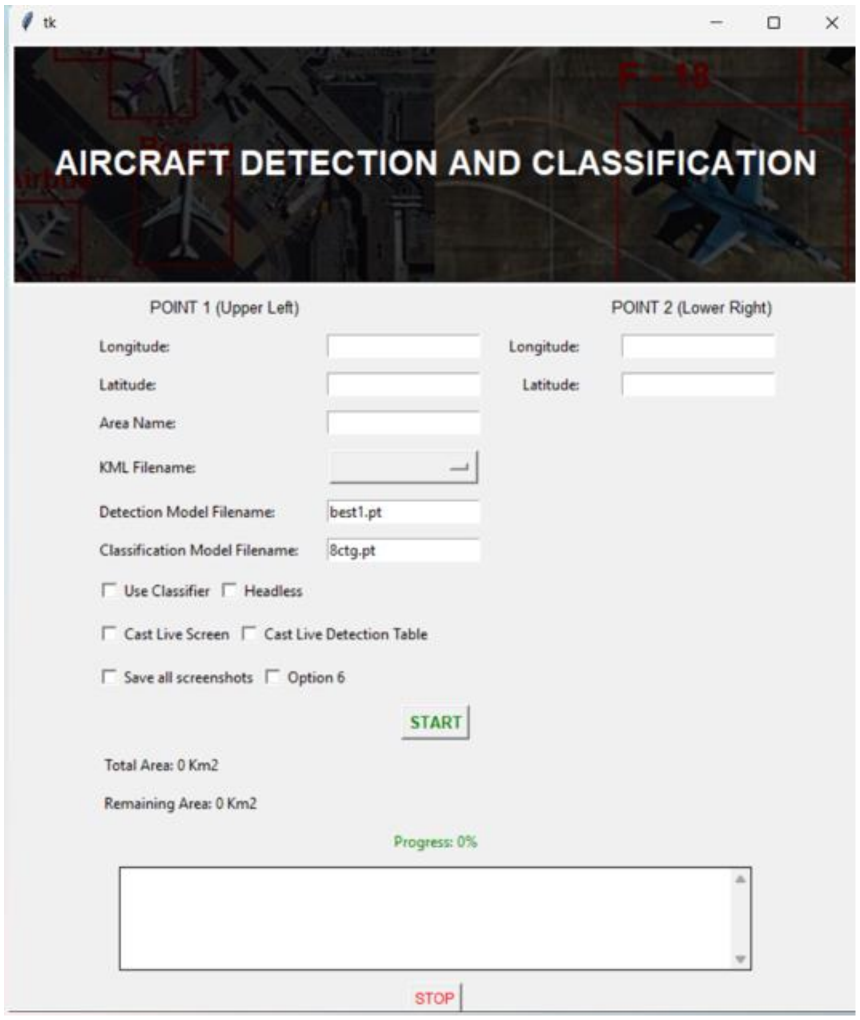
Figure 13 – Block Diagram GUI 2

5.2.2 Concept

This Python script uses the techniques of Browser automation by Selenium and Yolov5 object detection and classification to scan an area of google earth.

The script first defines several helper functions, including one to pre-process the input image by cropping it and one to crop a bounding box around a detected object for classification, send controlling commands to Selenium Browser, log all information in a CSV file and present information in windows of Tkinter.

1. The script loads the components as set in the GUI, which are:
    a. Point1 Longitude and Latitude
    b. Point2 Longitude and Latitude
    c. Area Name
    d. KML file (Generated by Google Earth Project option. It exports the coordinates of placemarks placed on the map)
    e. Detection Model (For object detection)
    f. Classification Model (For object classification)
    g. Use Classifier Checkbox (Whether to use classifier or not)
    h. Headless browser Checkbox (Starts the session without displaying the browser. Used to deploy on a server)
    i. Save all Screenshots Checkbox (Saves all screenshots of the area)
    j. Cast Live Detection on another screen Checkbox (Projects the browser to another screen and shows detection on them in real-time)
    k. Cast CSV Table Checkbox (Displays the log file as a separate window)



POINT 1 (Upper Left) | POINT 2 (Lower Right)

Longitude: | Longitude:

Latitude: | Latitude:

Area Name:

KML Filename:

Detection Model Filename: best1.pt

Classification Model Filename: 8ctg.pt

☐ Use Classifier ☐ Headless

☐ Cast Live Screen ☐ Cast Live Detection Table

☐ Save all screenshots ☐ Option 6

Total Area: 0 Km2

Remaining Area: 0 Km2

Progress: 0%

Figure 14 – The main window of GUI 2

2. Users have been provided with such vast options so that the application can serve a general purpose.
3. It checks all options' validity as all of them are interlinked. For example, if the Headless option is checked, then there is no point in casting the screen as the browser is not displaying anything, and the process is being run in the background.
4. It then setups the browser and sets its settings.
5. Loads the detection and, if checked, classification models. Sets NMS threshold.

6. Creates a CSV file, sets its header and places it in a folder under Area Name with the timestamp of session initiation.
7. Before starting a while True loop, it casts the screen to another screen using the same procedure as done in GUI 1, but in it, the screenshot is captured from the selenium library. It saves the preprocessing time to remove any unwanted details of the captured screenshot as it gives only the imagery. It also enables to have a headless option as if a complete screen was to be captured; a display browser was necessary to capture it. With this option, it can be deployed to a server with no display.
8. It also starts, if checked, the tkinter window to show the CSV file.
9. The while loop starts when the coordinates of the upper left point have been loaded. The quality of the map is dependent on the internet connection. A necessary check to check for blurred images has been placed so that it does not start the scan on blur images.
10. The loop starts a move right, move down, move left, move down and move right scan motion on the browser. It captures screenshots in between and processes them.
11. After the lower right coordinate has been reached, it quits the operation.

5.2.3 Libraries and Helper Functions:

Many libraries have been used, which made this product possible. For a better understanding of their usage, we need to understand what they are made for. These are all the libraries used:

- **os:** A Python module for interacting with the operating system. It provides a way to perform operations like navigating the file system, creating and deleting files and directories, and more.

- **Selenium:** A web testing framework used for automating web browsers. It provides a way to simulate user interactions with a web page, such as clicking buttons, filling out forms, and navigating between pages.
- **Msvrct:** To capture keyboard characters. Used to break the loop.
- **pyautogui:** A Python library for automating mouse and keyboard actions. It provides a way to simulate mouse clicks and movements, as well as keyboard presses and typing.
- **Time:** A module that provides various functions for working with time in Python. It can be used to measure how long a piece of code takes to run, delay program execution for a certain amount of time, and more.
- **cv2:** A library for computer vision in Python. It provides functions for image processing and analysis, such as image manipulation, object detection, and feature extraction.
- **CSV:** A module for working with CSV (comma-separated value) files in Python. It provides a way to read and write CSV files, which are commonly used for storing and exchanging data in a tabular format.
- **Torch:** A machine learning library for Python. It provides tools for building and training neural networks, as well as tools for data processing and analysis.
- **Torch.nn.functional:** A module within the PyTorch library that provides various functions for building neural networks. It includes functions for activation functions, loss functions, and more.

- **geopy:** A Python library for working with geolocation data. It provides tools for geocoding (converting addresses to latitude and longitude coordinates) and distance calculation between two points.
- **Threading:** A module in Python for creating and managing threads. It provides a way to run multiple codes simultaneously within a single program.
- **screeninfo:** A Python library for getting information about the user's display(s). It provides tools for getting the resolution, refresh rate, and other information about the screen(s) connected to the computer.
- **Tkinter:** A Python library for creating graphical user interfaces (GUIs). It provides tools for creating windows, buttons, menus, and other interface elements.
- **Pandas:** A library for data analysis in Python. It provides tools for working with tabular data, such as reading and writing data from various file formats, filtering and sorting data, and more.
- **textwrap:** A module in Python for formatting text. It provides functions for wrapping text to fit within a certain width, filling text with whitespace, and more.
- **argparse:** A module in Python for parsing command-line arguments. It provides a way to specify and parse command-line arguments passed to a Python script.
- **sys:** A module in Python for interacting with the Python interpreter. It provides a way to access system-specific variables and functions, such as the command-line arguments passed to the Python script.
- **pathlib:** A module in Python for working with file system paths. It provides an object-oriented way to manipulate paths rather than using string operations.

5.2.4 Working

Its working has been described below in modules for better explanation and understanding.

5.2.4.1 Scanning

The main challenge to automatically scan the area was to decide how it would go on as we wanted to automate Google Earth. The API of google earth is not free. So, we were left with the only option to automate the browser. The browser shows the same imagery offered by the API.

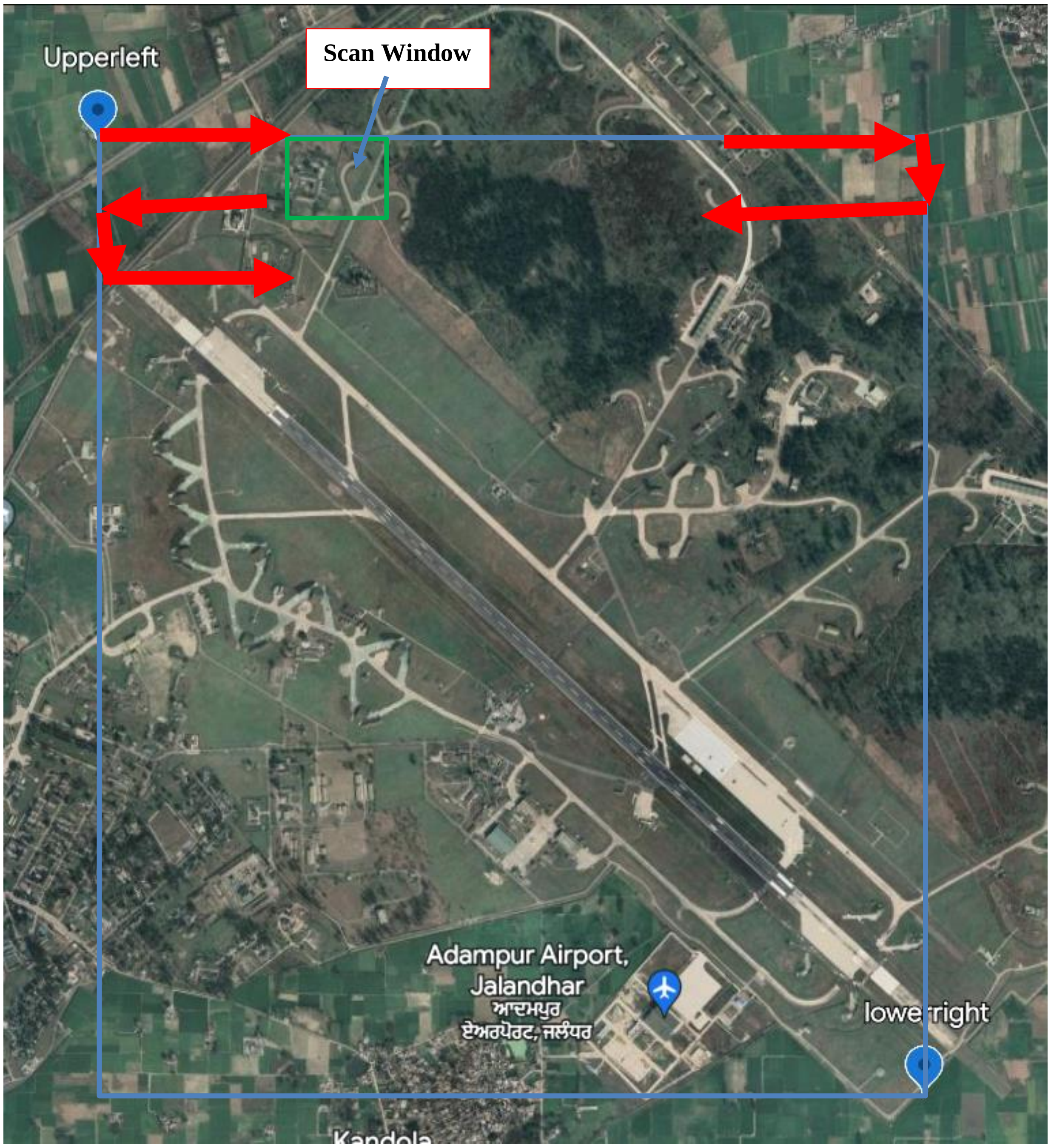
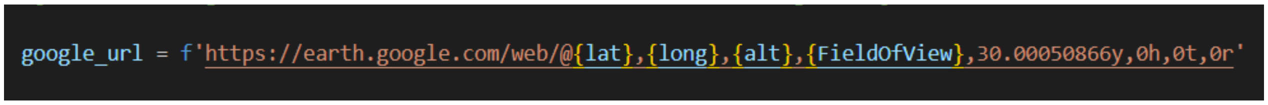


Figure 15 – Conceptualized Scan Motion of GUI 2

The above image shows the conceptualized motion of the scan on Google Earth.

The upper left placemark and Lower right placemark contains coordinates, and only two pair are enough to define a scan area. They are exported by Google Earth as KML files. The application has options to either enter coordinates or a KML file.

The program loads the coordinates of the Upper left placemark in a customized Google Earth link.



```python
google_url = f'https://earth.google.com/web/@{lat},{long},{alt},{FieldOfView},30.00050866y,0h,0t,0r'
```

0h,0t,0r makes the map exactly overhead without any tilt or orthogonal view.

This link is opened in the browser, and it leads to the upper left corner. From there keyboard right key press is initiated, which moves it to the right. Keypress is optimized to just cross the scanned area and bring a new unscanned area. The motion continues till it reaches the longitude of the lower right placemark. After reaching it moves the window down and starts moving it to the left till the longitude of the upper left placemark.

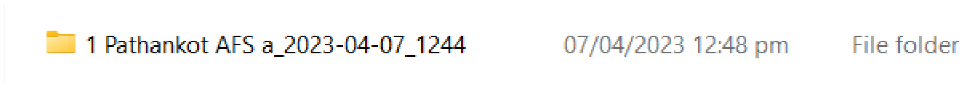
The scan will continue till it reaches the latitude of the lower right placemark. Hence in this way, the scan is automated.

The current coordinates are extracted by the Selenium browser library, which gives the current URL of the page. This way around has been used to get the current coordinates of the map area being shown. It was not possible other than this because OCRing the lower bar, which shows coordinates, was also tested, but it did not yield satisfactory results. Inspecting the page was also not possible as it employed Shadow DOM, which blocks extracting information.

The area to be scanned also depends on the browser size. The browser on Windows is 1920*1080 in size, so it yields images of the same resolution. If we increase the image size, then the area scanned in a unit of time can increase and thus scan speed of the area would be enhanced. It can be done if it is deployed on a Windows server. Selenium supports bigger browser resolutions when deployed on a server with headless option.

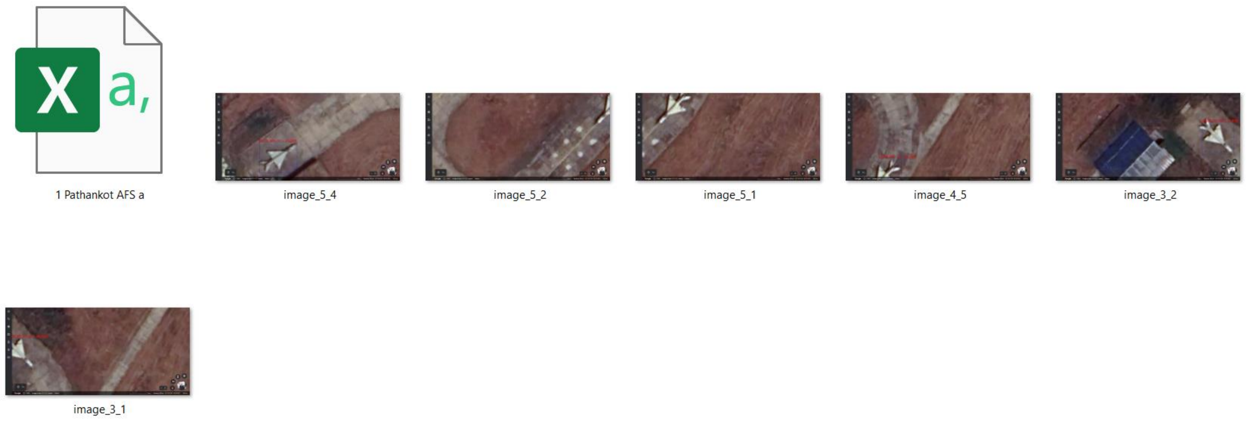
5.2.4.2 Logging

Documenting all information is an integral part of this project. It creates a folder under the Area name and timestamp of session initiation. All files are stored in it.



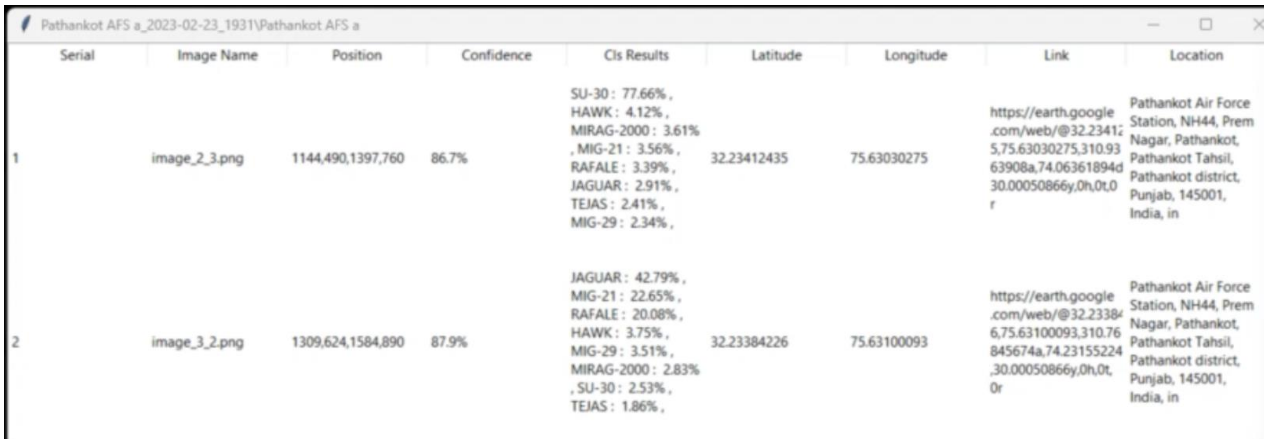
1 Pathankot AFS a_2023-04-07_1244 07/04/2023 12:48 pm File folder

It contains:



- All captured snaps of detected aircraft. If "save_all_screenshots" is selected, it saves all captured screenshots, whether they contain any detection or not.
- A CSV file which contains:
    - The serial number of the detection

- The snapshot in which the aircraft was found so it can be viewed later
- Bounding box coordinates, where in an image the object is
- Score of Confidence
- The classifier findings, which include the likelihood of each class against the aircraft
- Latitude and Longitude of the area
- A link to Google Earth takes users directly to the area where the aircraft was found
- Location details of the under-sampled area

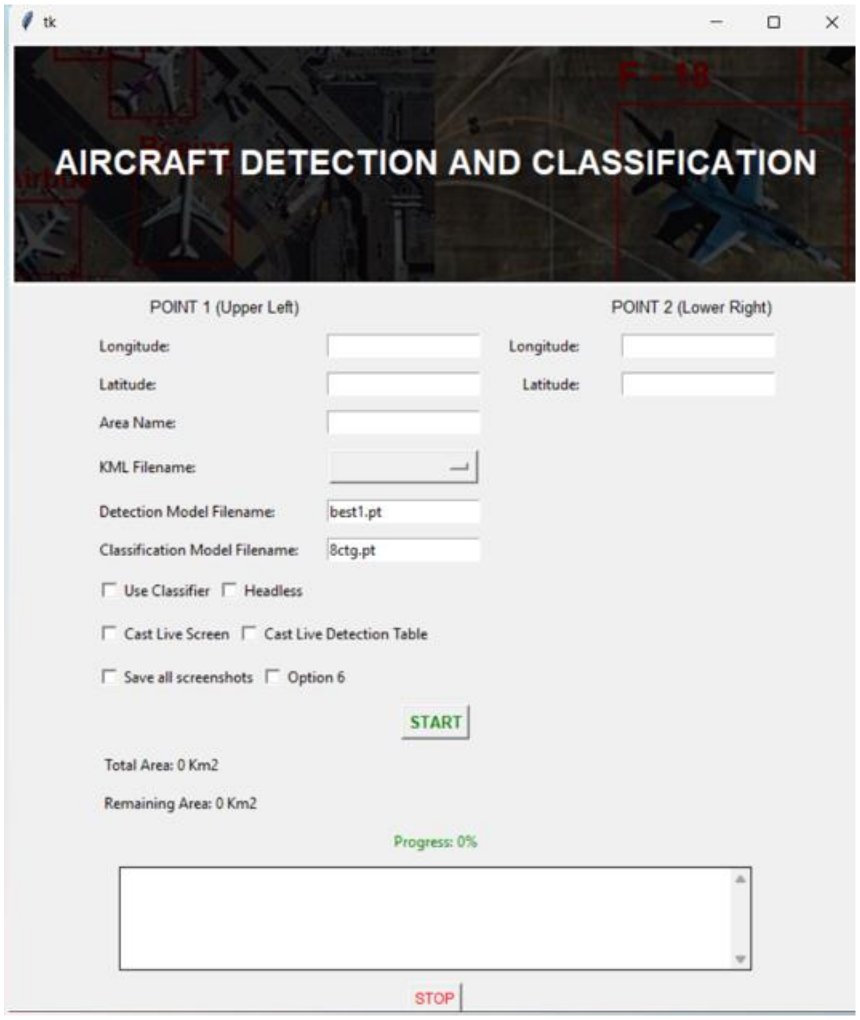

Pathankot AFS a_2023-02-23_1931\Pathankot AFS a

Serial	Image Name	Position	Confidence	Cls Results	Latitude	Longitude	Link	Location
1	image_2_3.png	1144,490,1397,760	86.7%	SU-30 : 77.66% , HAWK : 4.12% , MIRAG-2000 : 3.61% , MIG-21 : 3.56% , RAFALE : 3.39% , JAGUAR : 2.91% , TEJAS : 2.41% , MIG-29 : 2.34% ,	32.23412435	75.63030275	https://earth.google.com/web/@32.234125,75.63030275,310.9363908a,74.06361894d30.00050866y,0h,0t,0r	Pathankot Air Force Station, NH44, Prem Nagar, Pathankot, Pathankot Tahsil, Pathankot district, Punjab, 145001, India, in
2	image_3_2.png	1309,624,1584,890	87.9%	JAGUAR : 42.79% , MIG-21 : 22.65% , RAFALE : 20.08% , HAWK : 3.75% , MIG-29 : 3.51% , MIRAG-2000 : 2.83% , SU-30 : 2.53% , TEJAS : 1.86% ,	32.23384226	75.63100093	https://earth.google.com/web/@32.233846,75.63100093,310.76845674a,74.23155224,30.00050866y,0h,0t,0r	Pathankot Air Force Station, NH44, Prem Nagar, Pathankot, Pathankot Tahsil, Pathankot district, Punjab, 145001, India, in

Figure 16 – Results Display Pane of GUI 2

5.2.4.2 Presentation

The project has three possible windows for presenting information. One of them, the main GUI, has been developed on the tkinter python library. It serves as the central control for the project. It initiates the session and stops it when necessary.



POINT 1 (Upper Left)

POINT 2 (Lower Right)

Longitude:

Latitude:

Area Name:

KML Filename:

Detection Model Filename: best1.pt

Classification Model Filename: 8ctg.pt

Use Classifier Headless

Cast Live Screen Cast Live Detection Table

Save all screenshots Option 6

Total Area: 0 Km2

Remaining Area: 0 Km2

Progress: 0%

It provides information as Total Area and Remaining area in kilometers and Progress in percentage. The text box below shows all output by script. It gives more insight into what the script is doing at the moment.

Another window which shows the log file live is also made on Tkinter. The project logs everything in the CSV file, and this window reads it and displays it. This window is run as another process, so it does not slow the main script.

Lastly, the third window projects the working of the script on another screen. It is to demonstrate and show what and how the script scans an area. It captures the screenshot of the main window and projects it to another screen, just like in GUI 1.

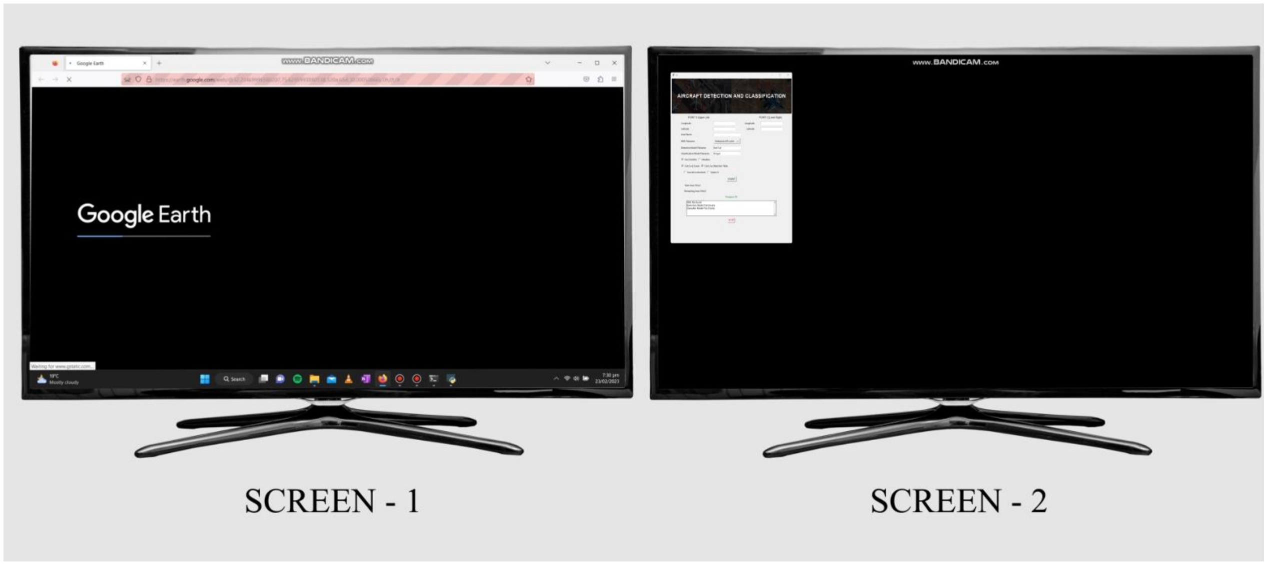


Figure 17a – Proposed Display of GUI 2

Below the image is the screenshot of screen 2. It has a table window and a live projection window.

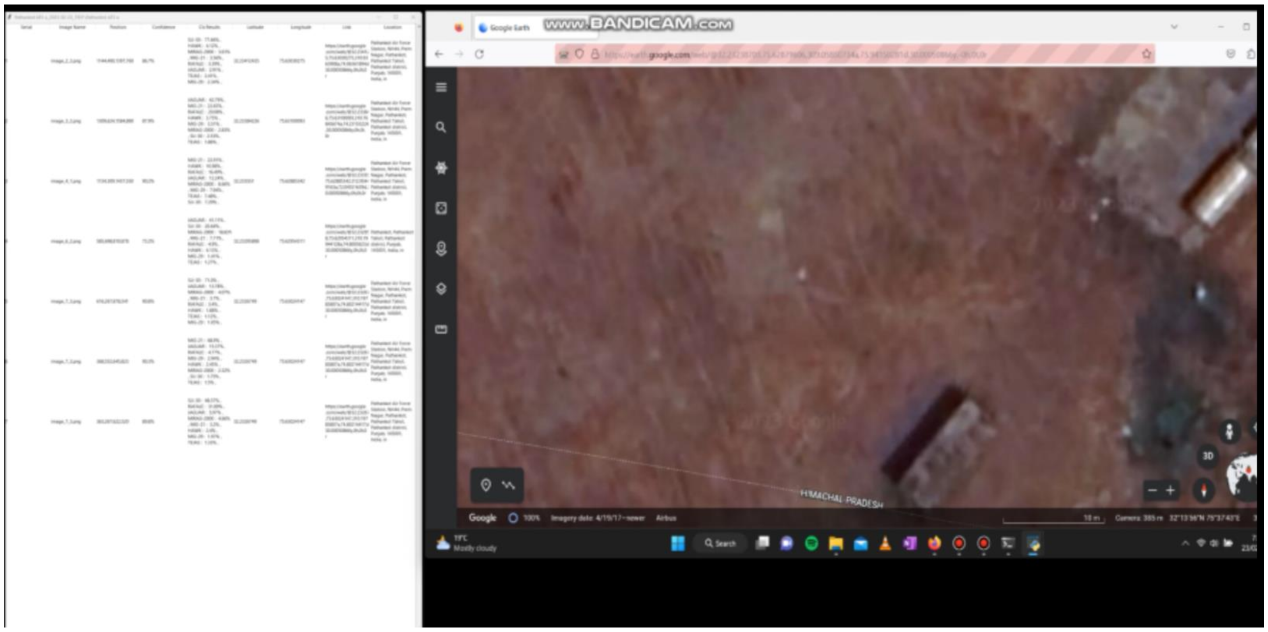


Figure 17b – Snapshot of GUI 2

5.2.5 Conclusion

This application can serve to detect and classify any object on Google Earth. It has thoroughly addressed the problem posed by intelligence agencies. It still has space for future work, like addressing the altitude and field of view adjustment according to the terrain. It requires more testing on a vast, diverse area.

It is ready to be deployed on a Windows server and can give excellent scan speed due to the bigger resolution size of the browser, as discussed above. The code has been provided in Annexure A.

Chapter 6: Conclusion

In this thesis, we discussed a novel intelligence-gathering system that can Detect and Classify any object smartly and more efficiently than the typically manual labor of an image analyst. Our proposed system has an advantage over other traditional systems due to the latest algorithms used for the detection of objects of interest. Techniques used in our proposed system, Image Processing techniques, Browser Automation and Machine Learning included algorithms such as YOLO, which were used to scan and detect the objects of interest; they are also briefly explained, including their working and importance. The purpose of increasing productivity is being achieved by using modern techniques. Additionally, the objectives; of accurate detection and classification and making the application general purpose are attained. Hence saves time as well as is more accurate.

Our proposed system, ADCSI, is cost-effective as it is purely made for the core purpose of serving Pakistan, a system that could be beneficial to its National Security. Else, similar hardware solutions provided and used by other countries are very costly.

Chapter 7: Future Work

Future additions that can be made to this project are:

- Other objects of interest can be detected and classified using this software. For example, Ships, Artillery Guns, trucks, airstrips and convoys.
- This project utilizes Google Earth as its source of imagery. It can be modified to scan Geotiff files produced by satellites or other sources.
- Deployment on a remote Windows-based server, which can make this application easily accessible and can have more incredible scan speed.
- Deployment of this technology on GIS.

References and Work Cited

1. https://techcult.com/how-often-does-google-earth-update/
2. https://www.cameralyze.co/blog/yolov7-architecture-explanation
3. https://github.com/ultralytics/yolov5/releases
4. https://pytorch.org/vision/stable/models.html
5. https://towardsdatascience.com/how-to-train-a-custom-object-detection-model-with-yolo-v5-917e9ce13208
6. Deep Multiple Instance Learning for Airplane Detection in High Resolution Imagery Mohammad Reza Mohammadi [Deep multiple instance learning for airplane detection in high-resolution imagery | SpringerLink]
7. Airplane Detection Based on Unsupervised Deep Domain Adaptation in Remote Sensing Image [https://doi.org/10.21203/rs.3.rs-2088221/v1]
8. Two-Stage Deep Learning Approach to the Classification of Fine-Art Paintings [https://ieeexplore.ieee.org/document/8675906]
9. Jain, A. (2020). Popular Classification Models for Machine Learning. Analytics Vidhya. Retrieved May 1, 2023, from https://www.analyticsvidhya.com/blog/2020/11/popular-classification-models-for-machine-learning/
10. Object Detection: Models, Architectures & Tutorial [2023]. (n.d.). V7Labs. Retrieved May 1, 2023, from https://www.v7labs.com/blog/object-detection-guide

Dataset links

1. https://www.kaggle.com/datasets/khlaifiabilel/military-aircraft-recognition-dataset?select=JPEGImages
2. https://zenodo.org/record/3843229#.ZE_TGM5By3B
3. https://drive.google.com/file/d/1--foZ3dV5OCsqXQXT84UeKtrAqc5CkAE/view?pli=1
4. https://arxiv.org/abs/2204.10959
5. https://registry.opendata.aws/rareplanes/
6. https://www.kaggle.com/code/jeffaudi/aircraft-detection-with-yolov5
7. https://zenodo.org/record/3464319#.ZE_TJs5By3B

Annexures

Annexure A

```
import os
from selenium import webdriver
from selenium.webdriver.firefox.options import Options
import msvcrt
import pyautogui
import time
import cv2 as cv
import csv

import torch
import time
import torch.nn.functional as F

from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="fypgoogleearth")

from GUI2_files.extra_helper import *
from GUI2_files.model_helper import *
from GUI2_files.windowcapture import *

import threading
import screeninfo

#Csv screen
import tkinter as tk
from tkinter import ttk
import pandas as pd
import textwrap

#Sys arg
import argparse
import os
import sys

from datetime import datetime

import pathlib
temp = pathlib.PosixPath
pathlib.PosixPath = pathlib.WindowsPath
```

```python
# Change the working directory to the folder this script is in.
os.chdir(os.path.dirname(os.path.abspath(__file__)))

def print_and_flush(data):
    print(data)
    sys.stdout.flush()

def csv_screen(filename):
    def wrap(string, length=20):
        string = str(string)
        return '\n'.join(textwrap.wrap(string, length))


    def update_table():
        global df
        global update_csv_screen
        global destroy_csv_screen
        if update_csv_screen:
            # read the csv file and convert it into a dataframe

            df = pd.read_csv(f'{filename}.csv')
            df = df.reset_index(drop=True)
            # clear the Table
            table.delete(*table.get_children())
            # populate the Table with the updated dataframe values
            for i, row in df.iterrows():
                row = list(row)
                row = list(map(wrap, row))
                table.insert("", i, values=((row)))
        if not destroy_csv_screen:

            root.after(1000, update_table) # check for updates every 1000ms (1
second)
        else:
            root.destroy()
            sys.exit()

    filename = f"{current_folder_name}\\{filename}"
    # read the csv file and convert it into a dataframe
    df = pd.read_csv(f'{filename}.csv')
    df = df.reset_index(drop=True)
    # create the tkinter window
    root = tk.Tk()
    root.title(f"{filename}")


    # create the Table
    table = ttk.Treeview(root, show='headings')

    # define the columns
    table["columns"] = list(df.columns)

    # format the columns
    for col in table["columns"]:
        table.column(col, width=150, anchor="w")
        table.heading(col, text=col)
    s = ttk.Style()
    s.configure('Treeview', rowheight=200)

    # populate the Table with the dataframe values
    for i, row in df.iterrows():
        row = list(row)
        row = list(map(wrap, row))
        table.insert("", i, values=((row)))

    # add a scrollbar
    scroll_y = ttk.Scrollbar(root, orient="vertical", command=table.yview)
    scroll_y.pack(side="right", fill="y")
    table.configure(yscrollcommand=scroll_y.set)


    # display the Table
    table.pack()

    root.after(1000, update_table) # start checking for updates every 1000ms (1
second)
    root.mainloop()


def parse_detections(results, tile):
    result_dict = []
    for result in results:
        con = result['confidence']
        con = '{:.1%}'.format(con)
        cs = result['class']
        x1 = int(result['xmin'])
        y1 = int(result['ymin'])
        x2 = int(result['xmax'])
        y2 = int(result['ymax'])
        ########################
        if y2 - y1 <= 0 or x2 - x1 <= 0:
            print_and_flush(f"ERROR H={y2-y1} , W={x2-x1}")
            continue
        color = (0, 0, 255)
        ########################
        if model_classifier != None:

            cropped_image = crop_one_box(tile, x1,y1,x2,y2)
            if cropped_image == None:
                continue
            #Classifier
            results_classifier = model_classifier(cropped_image)
            p = F.softmax(results_classifier, dim=1)  # probabilities
            p = p.numpy().tolist()[0]
            #classifier_results = '{:.1%} C'.format(p[0])+ ' - {:.1%}
M'.format(p[1])
            ###
            classifier_results = ''
            results = sorted(zip(class_names, p), key=lambda x: round(x[1]*100,
2), reverse=True)

            highestprob_class = results[0][0]
            highestprob_p = results[0][1]
            first = f"{class_names[highestprob_class]} :
{round(highestprob_p*100,2)}%"

            for (class_na, pred) in results:
                classifier_results += f"{class_names[class_na]}
:  {round(pred*100,2)}% , "


            ###########
            result_dict.append({'pos':f'{x1},{y1},{x2},{y2}','conf':f'{con}','Cla
ssifier':classifier_results})
            cv.rectangle(tile, (x1, y1), (x2, y2),(255,0,0), 1)
            cv.putText(tile, f'{con} {first}', (x1-2, y1-5),
cv.FONT_HERSHEY_DUPLEX, 1, color, 2) #(frame,text,bottom left corner of text,font
type, font scaling factor,color,thickness of line)

        else:
            result_dict.append({'pos':f'{x1},{y1},{x2},{y2}','conf':f'{con}','Cla
ssifier':'Nil'})
            cv.rectangle(tile, (x1, y1), (x2, y2),(255,0,0), 1)
            cv.putText(tile, f'{con}', (x1-2, y1-5), cv.FONT_HERSHEY_DUPLEX, 1,
color, 2) #(frame,text,bottom left corner of text,font type, font scaling
factor,color,thickness of line)


    return tile, result_dict

def append_to_csv(result_dict,image_name,lat,long,alt,FieldOfView):
    dictt = []
    global detection_counter
    #['Serial','Image Name','Position','Confidence','Latitude','Longitude']
    google_url =
f'https://earth.google.com/web/@{lat},{long},{alt},{FieldOfView},30.00050866y,0h,
0t,0r'
    try:

        location = geolocator.reverse(f"{lat},{long}", language='en')
        address = location.raw['address']
        #address = {key: value for key, value in address.items() if 'ISO' not in
key and 'code' not in key}
        address = {key: value for key, value in address.items() if 'ISO' not in
key}
        address = ", ".join(address.values())


    except Exception as e:
        print_and_flush(f"Exception occured retrieving loc data {e}")
        address = "Connection Error"

    for xx in result_dict:
        detection_counter += 1
        dictt.append({'Serial':detection_counter ,'Image Name':image_name
,'Position':xx['pos'] ,'Confidence':xx['conf'] ,'Cls
Results':xx['Classifier'],'Latitude':lat ,'Longitude':long, 'Link':google_url
,'Location':address})
    with open(csv_filename, 'a', newline='') as f_object:

        dictwriter_object = csv.DictWriter(f_object, fieldnames=csv_fields)

        dictwriter_object.writerows(dictt)

        f_object.close()
```

```python
def cc_screenshots():
    cv.namedWindow('P', cv.WINDOW_NORMAL)
    screentwo = screeninfo.get_monitors()[1]
    screentwo_width = screentwo.width
    screentwo_height = screentwo.height
    cv.moveWindow('P', screentwo_width , 0)
    while True:

        window_screenshot = get_screenshot()
        global show_model_image
        if show_model_image:
            global model_image
            cv.imshow('P', model_image)
            cv.waitKey(1000)
            show_model_image = False


        cv.imshow('P', window_screenshot)
        cv.waitKey(1)
        global stop_screenshot_thread
        if stop_screenshot_thread:
            break



"""
['upperleft,72.79693968164692,33.57384131541002,511.2013507374641',
'lowerright,72.86793319077674,33.53240399828832,512.0759362930472']
"""

# Create an ArgumentParser object
parser = argparse.ArgumentParser()
# Add an argument to the parser with a name and help message
parser.add_argument('--point1_longlat', help='Long,Lat of point 1')
parser.add_argument('--point2_longlat', help='Long,Lat of point 2')
parser.add_argument('--area_name', help='Area Name')
parser.add_argument('--kml_filename', help='KML filename (Mushaf Airbase)')
parser.add_argument('--det_model', help='Name of Detection model (best.pt)')
parser.add_argument('--cls_model', help='Name of Classifier model (best_cls.pt)')
parser.add_argument('--use_cls', help='Use Classifier (T/F)',
action='store_true')
```

```
parser.add_argument('--headless', help='Start program headless (T/F)',
action='store_true')
parser.add_argument('--save_allscreenshots', help='Save all Screenshots (T/F)',
action='store_true')
parser.add_argument('--cast_live_screenshot', help='Cast live screenshots (T/F)',
action='store_true')
parser.add_argument('--cast_csv_table', help='Cast CSV table (T/F)',
action='store_true')
args = parser.parse_args()
p1 = args.point1_longlat
p2 = args.point2_longlat
kml_filename = args.kml_filename
headless = args.headless
keep_nodetection_image = args.save_allscreenshots
screen_cast_check = args.cast_live_screenshot
cast_csv_table = args.cast_csv_table
detection_model_name = args.det_model
classification_model_name = args.cls_model
use_cls = args.use_cls
area_name = args.area_name

#Kml file
if kml_filename is not None:
    if os.path.isfile(f"kml_files\\{kml_filename}.kml"):
        print_and_flush("KML file found.")
        UL_lat, UL_long, LR_lat, LR_long =
get_cord_from_kml(f'kml_files\\{kml_filename}.kml')

    else:
        print_and_flush("KML file not found. Exiting")
        sys.exit()

#Point 1 2 and area name
if p1 and p2 is not None:
    UL_long , UL_lat = p1.split(",")
    LR_long , LR_lat = p2.split(",")

#Det model file check
if detection_model_name is not None:
    if os.path.isfile(f"model_files\\{detection_model_name}"):
        print_and_flush("Detection Model File Exists")
    else:
        print_and_flush("Detection Model Not Found. Exiting")
        sys.exit()
```

```
#Cls Model file check
if classification_model_name is not None:
    if os.path.isfile(f"model_files\\{classification_model_name}"):
        print_and_flush("Classifier Model File Exists")
    else:
        print_and_flush("Classifier Model Not Found. Exiting")
        sys.exit()




#kml_filename = 'Mushaf airbase'
#headless = False
#use_cls = True
#keep_nodetection_image = False
#cast_csv_table = True

options = Options()
options.headless = headless
options.binary_location = r'C:\Program Files\Mozilla Firefox\firefox.exe'
driver = webdriver.Firefox(executable_path='GUI2_files\\geckodriver.exe',
options=options)


right_mov_check = True
left_mov_check = False
up_mov_check = False
down_mov_check = False

#adampur 225a,130d
#google_url =
f'https://earth.google.com/web/@{UL_lat},{UL_long},225a,130d,30.00050866y,0h,0t,0
r'
# parameter=184, area=1817m2
google_url =
f'https://earth.google.com/web/@{UL_lat},{UL_long},320a,65d,30.00050866y,0h,0t,0r
'
driver.get(google_url)
if headless == True: print_and_flush("Headless iniitiated")
driver.maximize_window()


# Model
#Model Object Detection
startTime = time.time()
```

```
yolov5_folder_path = f"{os.getcwd()}\\yolov5"
model_path = f"{os.getcwd()}\\model_files"

model = torch.hub.load(yolov5_folder_path, 'custom',source="local",
path=f'{model_path}\\{detection_model_name}')
model.conf = 0.60  # NMS confidence threshold


# Model Classifier
if use_cls:

    model_classifier = torch.hub.load(yolov5_folder_path,
'custom',source="local",
path=f'{model_path}\\{classification_model_name}').cpu().float()
    class_names = model_classifier.names
    print_and_flush(class_names)

else:
    model_classifier = None

##################
endTime = time.time() - startTime
print_and_flush(f"Models Loaded in {endTime}")

driver.implicitly_wait(50)
#print_and_flush("Waiting 50 seconds so that page loads up...")

##################
#Create a csv file + folder
if kml_filename is not None:
    csv_filename = f'{kml_filename}.csv'
else:
    csv_filename = f'{area_name}.csv'

current_folder_name = csv_filename.removesuffix(".csv")
timestamp = datetime.now().strftime("%Y-%m-%d_%H%M")
current_folder_name = f"{current_folder_name}_{timestamp}"
os.makedirs(current_folder_name)

csv_filename = f"{current_folder_name}\\{csv_filename}"

csv_fields = ['Serial','Image Name','Position','Confidence','Cls
Results','Latitude','Longitude','Link','Location']
with open(csv_filename, 'w') as csvfile:
    # creating a csv dict writer object
    writer = csv.DictWriter(csvfile, fieldnames = csv_fields)

    # writing headers (field names)
    writer.writeheader()

##################

#print_and_flush the total ares sq2
_, _,total_areasq2 = calculate_areasq2(UL_lat, UL_long, LR_lat, LR_long)
if total_areasq2 < 1:
    use_sqmeters = True
    total_areasq2 = total_areasq2 * 1000000
    print_and_flush(f"TotalArea:{round(total_areasq2,3)} m2")
else:
    use_sqmeters = False
    print_and_flush(f"TotalArea:{round(total_areasq2,3)} km2")



rem_areasq2 = total_areasq2

###############################################################################Chec
k if page has loaded
time.sleep(30)
print_and_flush("Starting...after 30 seconds")
if headless == False:
    pyautogui.moveTo(1863, 130, 4)
    pyautogui.click()

element = driver.find_element_by_css_selector("body")
element.click()


# Get the initial URL of the webpage
previous_url = driver.current_url
press_and_hold('right',0.2,driver) # an initial push






global detection_counter
detection_counter = 0
```

```
number_of_row = 0
number_of_col = 0




stop_screenshot_thread = False
if screen_cast_check:

    screenshot_thread = threading.Thread(target=cc_screenshots)
    screenshot_thread.start()
    print_and_flush("Started Live screen window")
else:
    print_and_flush("Not Starting Live screen window")


#CSV screen
global update_csv_screen
global destroy_csv_screen

update_csv_screen = False
destroy_csv_screen = False

if cast_csv_table:
    if kml_filename is not None:
        csv_screen_thread = threading.Thread(target=csv_screen,
args=(kml_filename,))
    else:
        csv_screen_thread = threading.Thread(target=csv_screen,
args=(area_name,))

    csv_screen_thread.start()
    print_and_flush("Started CSV Table window")
else:
    print_and_flush("Not Starting CSV Table window")




#check for showing model passed image
show_model_image = False

# Avg longitude and latitude difference while moving

total_lat_diff = float(UL_lat) - float(LR_lat)
```

```
while True:

    current_url = driver.current_url

    time.sleep(1)
    if current_url != previous_url:
        lat, long,alt,FieldOfView = parse_url(current_url)
        prev_lat, prev_long,_,_ = parse_url(previous_url)

        #Take screenshot here
        time.sleep(1)

        screenshot = driver.get_screenshot_as_png()
        screenshot, blur_score = perform_on_screenshot(screenshot)
        while(True):
            #take screenshot again as the image is blur
            if blur_score > 80:
                break

            time.sleep(5)
            screenshot = driver.get_screenshot_as_png()
            screenshot, blur_score = perform_on_screenshot(screenshot)

        detections = model(screenshot, size=640)
        results = detections.pandas().xyxy[0].to_dict(orient="records")
        if len(results) != 0:
            screenshot,result_dict = parse_detections(results, screenshot)

            append_to_csv(result_dict,f"image_{number_of_row}_{number_of_col}.png
",lat,long,alt,FieldOfView)

            cv.imwrite(f"{current_folder_name}\\image_{number_of_row}_{number_of_
col}.png", screenshot)
            #cv.imwrite(f"image_{number_of_row}_{number_of_col}.png",
pad_model_image(screenshot))

            #Check to start the check to update csv
            update_csv_screen = True

            model_image = pad_model_image(screenshot)
            show_model_image = True
            time.sleep(1)

            update_csv_screen = False
        else:
            if keep_nodetection_image:
                cv.imwrite(f"{current_folder_name}\\image_{number_of_row}_{number
_of_col}.png", screenshot)



        #####################
        if right_mov_check:
            press_and_hold('right',1.5,driver)

            diff_right_mov = float(LR_long)-float(long)

            #print_and_flush(f"Long(right)={number_of_col}")

            number_of_col += 1
            if diff_right_mov <= 0:
                right_mov_check = False
                move_down(driver)
                number_of_row += 1
                number_of_col = 0
                left_mov_check = True


        if left_mov_check:
            press_and_hold('left',1.5,driver)

            diff_left_mov = float(long)-float(UL_long)
            #print_and_flush(f"Long(Left)={number_of_col}")

            number_of_col += 1
            if diff_left_mov <= 0:
                right_mov_check = True
                move_down(driver)
                number_of_row += 1
                number_of_col = 0
                left_mov_check = False



        latitude_diff = float(lat)-float(LR_lat)
        latitude_diff_percent = ((1 - latitude_diff / total_lat_diff) * 100)
        latitude_diff_percent = 0 if latitude_diff_percent < 0 else
latitude_diff_percent
```

```
        latitude_diff_percent = 100 if latitude_diff_percent >100 else
latitude_diff_percent
        latitude_diff_percent = round(latitude_diff_percent,1)
        print_and_flush(f"progress:{latitude_diff_percent}")

        # if use_sqmeters: #total_areasq2
        #     rem_areasq2 -= float(4050)
        # else:
        #     rem_areasq2 -= float(0.00805)

        rem_areasq2 = total_areasq2 - ((total_areasq2 * latitude_diff_percent ) /
100)

        print_and_flush(f"RemArea:{round(rem_areasq2,3)}")

        if latitude_diff <= 0:
            print_and_flush("Destination reached")
            break

        previous_url = current_url



    # Check if a key has been pressed
    if msvcrt.kbhit():
        # Get the ASCII value of the key that was pressed
        key_code = ord(msvcrt.getch())

        # If the key that was pressed was the "q" key, break the loop
        if key_code == ord("q"):
            break

# Close the web browser
driver.quit()
stop_screenshot_thread = True
destroy_csv_screen = True

print_and_flush("quit")
```

```
import os
import cv2 as cv
import numpy as np
import torch
from GUI1_files.windowcapture import *
from time import time
import torch.nn.functional as F

import pathlib
temp = pathlib.PosixPath
pathlib.PosixPath = pathlib.WindowsPath

#250a gives 224*224 image

# Change the working directory to the folder this script is in.
os.chdir(os.path.dirname(os.path.abspath(__file__)))


def pre_process_image(image):

    top = 125
    bottom = 50
    left = 55
    right = 1

    # Get the shape of the image
    rows, cols, channels = image.shape

    # Create a black image with the same shape as the input image
    black_image = np.zeros((rows, cols, channels))

    # Crop the input image using slicing
    cropped_image = image[top:-bottom, left:-right]
    # Assign the values of the cropped image to the corresponding portion of the
black image
    black_image[top:-bottom, left:-right] = cropped_image
    black_image = black_image.astype(np.uint8)

    return black_image


def crop_one_box(frame, x1,y1,x2,y2):
    # Functions Classifier
    resize_C = torch.nn.Upsample(size=(224, 224), mode='bilinear',
align_corners=False)
    normalize_C = lambda x, mean=0.5, std=0.25: (x - mean) / std

    img = (frame[y1-10:y2+10, x1-10:x2+10])
    if img.shape[0] <= 0 or img.shape[1] <= 0:
        return None

    #image resize and normalize
    img = np.ascontiguousarray(np.asarray(img).transpose((2, 0, 1)))  # HWC to
CHW
    img = torch.tensor(img).unsqueeze(0) / 255.0  # to Tensor, to BCWH, rescale
    img = resize_C(normalize_C(img))
    return img



def parse_detections(results, tile):
    for result in results:
        con = result['confidence']
        con = '{:.1%}'.format(con)
        cs = result['class']
        x1 = int(result['xmin'])
        y1 = int(result['ymin'])
        x2 = int(result['xmax'])
        y2 = int(result['ymax'])
        ########################
        if y2 - y1 <= 0 or x2 - x1 <= 0:
            print(f"ERROR H={y2-y1} , W={x2-x1}")
            continue
        color = (0, 0, 255)
        ########################
        if model_classifier != None:

            cropped_image = crop_one_box(tile, x1,y1,x2,y2)
            if cropped_image == None:
                continue
            #Classifier
            results_classifier = model_classifier(cropped_image)
            p = F.softmax(results_classifier, dim=1)  # probabilities
            p = p.numpy().tolist()[0]
            #classifier_results = '{:.1%} C'.format(p[0])+ ' - {:.1%}
M'.format(p[1])
            ###
            classifier_results = ''
            for (class_na, pred) in zip(class_names,p):
                classifier_results += 
f"{class_names[class_na]}:{round(pred*100,2)} , "

            ###########

            cv.rectangle(tile, (x1, y1), (x2, y2),(255,0,0), 1)
            cv.putText(tile, f'{con} {classifier_results}', (x1-2, y1-5), 
cv.FONT_HERSHEY_DUPLEX, 1, color, 2) #(frame,text,bottom left corner of text,font 
type, font scaling factor,color,thickness of line)

        else:
            cv.rectangle(tile, (x1, y1), (x2, y2),(255,0,0), 1)
            cv.putText(tile, f'{con}', (x1-2, y1-5), cv.FONT_HERSHEY_DUPLEX, 1, 
color, 2) #(frame,text,bottom left corner of text,font type, font scaling 
factor,color,thickness of line)


    return tile

#Ask for whether to run classifier
use_cls = False
classifier_check = input("Use classifier? y/n ")
if classifier_check == 'y':
    use_cls = True
else:
    use_cls = False

# Model
#Model Object Detection
startTime = time()

yolov5_folder_path = f"{os.getcwd()}\\yolov5"
model_path = f"{os.getcwd()}\\model_files"

model = torch.hub.load(yolov5_folder_path, 'custom',source="local", 
path=f'{model_path}\\best1.pt')
model.conf = 0.40  # NMS confidence threshold

# Model Classifier
if use_cls:

    model_classifier = torch.hub.load(yolov5_folder_path, 
'custom',source="local", path=f'{model_path}\\cus50.pt').cpu().float()
```

```
    class_names = model_classifier.names
    print(class_names)

else:
    model_classifier = None

##################


endTime = time() - startTime
print(f"Models Loaded in {endTime}")

loop_time = time()
while(True):

    image = get_screenshot() #returns a numpy array
    output_image = image.copy()

    image = pre_process_image(image)

    # Set the window size
    window_size = 640

    # Get the width and height of the image
    image_height, image_width, _ = image.shape

    # Calculate the number of windows needed in the x and y directions
    num_windows_x = image_width // window_size
    #num_windows_x = (image_width + window_size - 1) // window_size
    num_windows_y = (image_height + window_size - 1) // window_size


    # Slide the window over the image
    for i in range(num_windows_x):
        for j in range(num_windows_y):
            # Get the current window
            x_min = i * window_size
            y_min = j * window_size
            x_max = x_min + window_size
            y_max = y_min + window_size
            window = image[y_min:y_max, x_min:x_max]

            # Make predictions on the window
            detections = model(window, size=640)
            results = detections.pandas().xyxy[0].to_dict(orient="records")
            # Draw the bounding boxes on the output image
            output_image[y_min:y_max, x_min:x_max] = parse_detections(results, 
window)



    cv.imshow('Privew', output_image)

    # debug the loop rate
    print('FPS {}'.format(1 / (time() - loop_time)))
    loop_time = time()

    # press 'q' with the output window focused to exit.
    # waits 1 ms every loop to process key presses
    if cv.waitKey(1) == ord('q'):
        cv.destroyAllWindows()
        break

print('Done.')
```

```python
#coding:utf-8
import cv2
import os
import random


label_folder = './labels/'

raw_images_folder = './raw_images/'

save_images_folder = './save_image/'

name_list_path = './name_list.txt'

classes_path = './classes.txt'


def plot_one_box(x, image, color=None, label=None, line_thickness=None):
    # Plots one bounding box on image img
    tl = line_thickness or round(0.002 * (image.shape[0] + image.shape[1]) / 2) +
1  # line/font thickness
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
    cv2.rectangle(image, c1, c2, color, thickness=tl, lineType=cv2.LINE_AA)
    if label:
        tf = max(tl - 1, 1)  # font thickness
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=tf)[0]
        c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
        cv2.rectangle(image, c1, c2, color, -1, cv2.LINE_AA)  # filled
        cv2.putText(image, label, (c1[0], c1[1] - 2), 0, tl / 3, [225, 255, 255],
thickness=tf, lineType=cv2.LINE_AA)

# image_name
def draw_box_on_image(image_name, classes, colors, label_folder,
raw_images_folder, save_images_folder ):
    txt_path  = os.path.join(label_folder,'%s.txt'%(image_name))
    print(image_name)
    if image_name == '.DS_Store':
        return 0
    image_path = os.path.join( raw_images_folder,'%s.jpg'%(image_name))

    save_file_path = os.path.join(save_images_folder,'%s.jpg'%(image_name))
    # flag_people_or_car_data =
    source_file = open(txt_path)
    image = cv2.imread(image_path)
    try:
        height, width, channels = image.shape
    except:
        print('no shape info.')
        return 0

    box_number = 0
    for line in source_file:
        staff = line.split()
        class_idx = int(staff[0])

        x_center, y_center, w, h = float(staff[1])*width, float(staff[2])*height,
float(staff[3])*width, float(staff[4])*height
        x1 = round(x_center-w/2)
        y1 = round(y_center-h/2)
        x2 = round(x_center+w/2)
        y2 = round(y_center+h/2)

        # if class_idx == 0:
        #     draw_people_tangle = cv2.rectangle(image,
(x1,y1),(x2,y2),(0,0,255),2)
        #     cv2.imwrite(save_file_path,draw_people_tangle)
        # elif class_idx == 1:
        #     draw_car_tangle =
cv2.rectangle(image,(x1,y1),(x2,y2),(0,255,0),2)
        #     cv2.imwrite(save_file_path,draw_car_tangle)

        plot_one_box([x1,y1,x2,y2], image, color=colors[class_idx],
label=classes[class_idx], line_thickness=None)

        cv2.imwrite(save_file_path,image)

        box_number += 1
    return box_number


def make_name_list(raw_images_folder, name_list_path):

    image_file_list = os.listdir(raw_images_folder)

    text_image_name_list_file=open(name_list_path,'w')
    for  image_file_name in image_file_list
        image_name,file_extend = os.path.splitext(image_file_name)
        text_image_name_list_file.write(image_name+'\n')

    text_image_name_list_file.close()


if __name__ == '__main__':

    make_name_list(raw_images_folder, name_list_path)

    classes = image_names = open(classes_path).read().strip().split()
    random.seed(42)
    colors = [[random.randint(0, 255) for _ in range(3)] for _ in
range(len(classes))]

    image_names = open(name_list_path).read().strip().split()

    box_total = 0
    image_total = 0
    for image_name in image_names:
        box_num = draw_box_on_image(image_name, classes, colors, label_folder,
raw_images_folder, save_images_folder)
        box_total += box_num
        image_total += 1
        print('Box number:', box_total, 'Image number:',image_total)
```

```python
import matplotlib.pyplot as plt
import numpy as np

dataArray = np.genfromtxt('results.csv', delimiter=',', names=True)

plt.subplot(2, 3, 1)
plt.plot(dataArray['trainbox_loss'], label='trainbox_loss')
plt.plot(dataArray['valbox_loss'], label='valbox_loss')
plt.legend()
plt.grid()


plt.subplot(2, 3, 2)
plt.plot(dataArray['trainobj_loss'], label='trainobj_loss')
plt.plot(dataArray['valobj_loss'], label='valobj_loss')
plt.legend()
plt.grid()

plt.subplot(2, 3, 3)
plt.plot(dataArray['metricsprecision'], label='metricsprecision')
plt.plot(dataArray['metricsrecall'], label='metricsrecall')
plt.legend()
plt.grid()

plt.subplot(2, 3, 4)
plt.plot(dataArray['metricsmAP_05'], label='metricsmAP_05')
plt.legend()
plt.grid()

plt.subplot(2, 3, 5)
plt.plot(dataArray['metricsmAP_05095'], label='metricsmAP_05095')
plt.legend()
plt.grid()

plt.show()
```

Annexure E

```
import os
import shutil

# list of folder names in the same directory as code
folder_names = ['BOEING','HAWK', 'JAGUAR', 'MIG-21','MIG-29','MIRAG-
2000','RAFALE','SU-30','TEJAS']

# percentage of files to be placed in train, test, and valid
train_percent = 0.9
test_percent = 0.05
valid_percent = 0.05

# create Dataset folder
if not os.path.exists('Dataset'):
    os.makedirs('Dataset')

# create train, test, and valid folders in Dataset folder
if not os.path.exists('Dataset/train'):
    os.makedirs('Dataset/train')
if not os.path.exists('Dataset/test'):
    os.makedirs('Dataset/test')
if not os.path.exists('Dataset/valid'):
    os.makedirs('Dataset/valid')

# loop through the folder names
for folder in folder_names:
    # create folder with the same name in train, test, and valid
    os.makedirs('Dataset/train/' + folder)
    os.makedirs('Dataset/test/' + folder)
    os.makedirs('Dataset/valid/' + folder)

    # get all files in the folder
    files = os.listdir(folder)

    # calculate number of files for train, test, and valid
    train_files = int(len(files) * train_percent)
    test_files = int(len(files) * test_percent)
    valid_files = int(len(files) * valid_percent)
    print(f"{folder}: Total images={len(files)}; train={train_files},
valid={valid_files}, test={test_files}.
total={train_files+valid_files+test_files}")
    # copy files to train, test, and valid folders
    for i in range(train_files):
        shutil.copy(folder + '/' + files[i], 'Dataset/train/' + folder)
    for i in range(train_files, train_files+test_files):
        shutil.copy(folder + '/' + files[i], 'Dataset/test/' + folder)
    for i in range(train_files+test_files, len(files)):
        shutil.copy(folder + '/' + files[i], 'Dataset/valid/' + folder)
```

Turnitin Originality Report

Processed on: 01-May-2023 11:40 PM EDT
ID: 2076974441
Word Count: 11453
Submitted: 3

Syndicate 15 By Bilal Janjua

Similarity Index	Similarity by Source	
11%	Internet Sources:	5%
	Publications:	4%
	Student Papers:	8%

2% match (student papers from 30-May-2022)
Submitted to Higher Education Commission Pakistan on 2022-05-30

1% match (Internet from 05-Mar-2021)
https://archive.org/details/peel?and%5B%5D=firstTitle%3AI&sort=titleSorter

< 1% match (student papers from 27-Jun-2013)
Submitted to Higher Education Commission Pakistan on 2013-06-27

< 1% match (student papers from 15-Aug-2022)
Submitted to Liverpool John Moores University on 2022-08-15

< 1% match (student papers from 23-Feb-2023)
Submitted to Liverpool John Moores University on 2023-02-23

< 1% match (student papers from 27-Feb-2023)
Submitted to Liverpool John Moores University on 2023-02-27

< 1% match (student papers from 15-Mar-2023)
Submitted to Liverpool John Moores University on 2023-03-15

< 1% match (student papers from 22-Jan-2023)
Submitted to Nanyang Technological University on 2023-01-22

< 1% match (student papers from 23-Jan-2023)
Submitted to Nanyang Technological University on 2023-01-23

< 1% match (student papers from 15-Jan-2023)
Submitted to Abdullah Gul University on 2023-01-15

< 1% match (student papers from 26-Feb-2023)
Submitted to Babes-Bolyai University on 2023-02-26

< 1% match (student papers from 06-Dec-2022)
Submitted to University of Ruhuna Matara on 2022-12-06

< 1% match (student papers from 01-May-2023)
Submitted to University of Hertfordshire on 2023-05-01

< 1% match (student papers from 21-Mar-2023)
Submitted to University of Hertfordshire on 2023-03-21

< 1% match (Internet from 17-Mar-2023)
https://www.wikiwand.com/en/Draft:Complete_programming_of_python

< 1% match (student papers from 01-May-2023)
Submitted to University of Bolton on 2023-05-01

< 1% match (Maragoni Mahendar, Arun Malik, Isha Batra. "Emotion estimation model for cognitive state analysis of learners in online education using deep learning", Expert Systems, 2023)
Maragoni Mahendar, Arun Malik, Isha Batra. "Emotion estimation model for cognitive state analysis of learners in online education using deep learning", Expert Systems, 2023

< 1% match (student papers from 30-Mar-2023)
Submitted to Leiden University on 2023-03-30

< 1% match (Lecture Notes in Computer Science, 2005.)
Lecture Notes in Computer Science, 2005.

< 1% match (Internet from 23-Mar-2023)
https://pdffox.com/implementation-and-performance-evaluation-of-a-cmis-server-for-open-source-php-based-wcms-pdf-free.html

< 1% match (Bhavan Kumar S B, Guhan S, Manyam Kishore, Santhosh R, Alfred Daniel J. "Deep Learning Approach for Pothole Detection - A Systematic Review", 2023 Second International Conference on Electronics and Renewable Systems (ICEARS), 2023)
Bhavan Kumar S B, Guhan S, Manyam Kishore, Santhosh R, Alfred Daniel J. "Deep Learning Approach for Pothole Detection - A Systematic Review", 2023 Second International Conference on Electronics and Renewable Systems (ICEARS), 2023

< 1% match (Internet from 20-Apr-2023)
https://uobrep.openrepository.com/bitstream/handle/10547/625758/DAWAM%20Edward%20Swarlat%201415601%20FULL%20REPOSITORY%20COPY.pdf?isAllowed=y&sequence=1

< 1% match (student papers from 19-Aug-2021)
Submitted to Visvesvaraya Technological University, Belagavi on 2021-08-19

< 1% match (Internet from 16-Dec-2022)
https://files.osf.io/v1/resources/rvzyc/providers/osfstorage/621700d67f41120253fa16ef?action=download&direct=&version=2

< 1% match (student papers from 14-Sep-2019)
Submitted to Seevic College on 2019-09-14

< 1% match (Internet from 26-Feb-2023)
http://ir.mu.ac.ke:8080/jspui/bitstream/123456789/4272/1/OGINA%20THESIS.pdf

< 1% match (student papers from 21-Jan-2023)
Submitted to Asia Pacific International College on 2023-01-21

< 1% match (student papers from 11-Apr-2023)
Submitted to Coventry University on 2023-04-11

< 1% match (student papers from 13-May-2021)
Submitted to University of Hull on 2021-05-13

< 1% match (Internet from 24-Mar-2023)
https://WWW.coursehero.com/file/95265840/PI-2190822112442-1docx/

< 1% match (Lecture Notes in Computer Science, 2010.)
Lecture Notes in Computer Science, 2010.

< 1% match (Internet from 05-Mar-2023)
http://etd.aau.edu.et/bitstream/handle/123456789/15459/Hassen%20Mohammed.pdf?isAllowed=y&sequence=1

< 1% match (Internet from 18-Nov-2022)
http://etd.aau.edu.et/bitstream/handle/123456789/16748/Yadeta%20Gizaw.pdf?isAllowed=y&sequence=1

< 1% match (student papers from 27-Apr-2023)
Submitted to The Robert Gordon University on 2023-04-27

< 1% match (student papers from 21-Apr-2023)
Submitted to The University of the West of Scotland on 2023-04-21

< 1% match (Fisher, A.. "The dynamics of tree cover change in a rural Australian landscape", Landscape and Urban Planning, 19991201)
Fisher, A.. "The dynamics of tree cover change in a rural Australian landscape", Landscape and Urban Planning, 19991201

< 1% match (student papers from 23-Apr-2023)
Submitted to University of North Texas on 2023-04-23

< 1% match (Internet from 28-Feb-2020)
http://collections.mun.ca/cdm/compoundobject/collection/cns2/id/41678/rec/18

< 1% match (Internet from 10-Jan-2023)
https://jp.mathworks.com/help/wavelet/ug/wavelet-scattering.html

< 1% match (Internet from 07-Oct-2022)
http://www.laspositascollege.edu/gv/pdc/assets/docs/mandatoryflex/archives/fall2019/MentalArchitecture.pptx

< 1% match (student papers from 25-Apr-2021)
Submitted to Purdue University on 2021-04-25

< 1% match (student papers from 05-Jan-2023)
Submitted to University of Sunderland on 2023-01-05

< 1% match (student papers from 28-Mar-2023)
Submitted to University of Surrey on 2023-03-28

< 1% match ()
Al Salem, Aqeel Asaad. "Managing Consistency and Consensus in Group Decision-Making with Incomplete Fuzzy Preference Relations", 2017

< 1% match (Internet from 15-Mar-2023)
https://www.mdpi.com/1424-8220/23/6/3147/htm

< 1% match (student papers from 07-Apr-2023)
Submitted to Cardiff University on 2023-04-07

< 1% match (Internet from 26-Oct-2022)
https://qmro.qmul.ac.uk/xmlui/bitstream/handle/123456789/79350/ZHANG_Xindi_130799436_EECS_PhD_final.pdf?isAllowed=y&sequence=1

< 1% match (Bo Lu, Bingchuan Bai, Xuefeng Zhao. "Vision-based structural displacement measurement under ambient-light changes via deep learning and digital image processing", Measurement, 2023)
Bo Lu, Bingchuan Bai, Xuefeng Zhao. "Vision-based structural displacement measurement under ambient-light changes via deep learning and digital image processing", Measurement, 2023

< 1% match (Internet from 10-Jan-2023)
https://moviecultists.com/where-to-find-bounding-box

< 1% match (Internet from 20-Nov-2015)
http://www.gamefaqs.com/ps/572645-arc-the-lad-ii/faqs/67059

< 1% match (student papers from 26-Apr-2023)
Submitted to Metropolia Ammattikorkeakoulu Oy on 2023-04-26

< 1% match (student papers from 27-Aug-2020)
Submitted to University of Edinburgh on 2020-08-27

< 1% match (Internet from 08-Dec-2022)
https://apo.org.au/sites/default/files/resource-files/2019-05/apo-nid235671_1.pdf

< 1% match (Internet from 16-Dec-2022)
https://github.com/gokulsaraswat/365Days_MachineLearning_DeepLearning/blob/main/README.md

< 1% match ()
Herman, Hilde. "A framework for the design, development and implementation of technology platforms in the South African health context", Stellenbosch : Stellenbosch University, 2019

< 1% match (Internet from 18-Dec-2022)
https://www.icao.int/Meetings/SUR-Technologies/Documents/D1%20F.Apeagyei%20IFATSEA%20Session%203.pdf

< 1% match (Internet from 13-Apr-2023)
https://www.um.edu.mt/library/oar/bitstream/123456789/108246/1/2219ICTICS520000005951_1.PDF

< 1% match (Internet from 20-Jun-2019)
https://zh.scribd.com/doc/146394333/Encyclopedia-of-Flight

< 1% match (Tang, Pengjie, Hanli Wang, and Sam Kwong. "G-MS2F: GoogLeNet based multi-stage feature fusion of deep CNN for scene recognition", Neurocomputing, 2016.)
Tang, Pengjie, Hanli Wang, and Sam Kwong. "G-MS2F: GoogLeNet based multi-stage feature fusion of deep CNN for scene recognition", Neurocomputing, 2016.