CONTEXT-AWARE OBJECT DETECTION IN COMPUTER VISION

Final Year Dissertation

Author

Syed Ayman Naushad

Supervisor

Dr. Radu-Casian Mihailescu

HERIOT-WATT UNIVERSITY

School of Mathematical and Computer Sciences

BSc. Computer Science (Artificial Intelligence)

Declaration

I, Syed Ayman Naushad confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Syed Ayman

Date: November 30, 2022

Acknowledgements

I thank Prof. Radu-Casian Mihailescu for his support and guidance throughout the duration of his supervision of this project. I also thank my parents for their continuous motivation.

Abstract

The presence of an object in real life is rarely in solitude, they co-vary with other objects and its surroundings creating an abundant source of contextual information, such as associations and correlations. Humans build scene representations over time using information gained through contextual inference and analysis to identify regions of interest in visual media. Attempts to reproduce this process using Artificial Intelligence is the inspiration behind the state-of-the-art Object Detection algorithms of today. Context plays a major role in the way humans perceive the world; hence it is rational to assume that the provision of contextual information in the object detection algorithm can only enhance its performance.

In this project, the training algorithm of an object detection model is enhanced by integrating contextual information in the form of 'scene labels', through various methods to identify the most optimal approach. A total of four deep learning models are implemented and from among the two contextually aware object detection models, the best performing model produced a **classification accuracy of 75.14%** and a **mean bounding box IoU of 0.69** on the constructed dataset. This was achieved by virtue of an auxiliary scene classification model used to make image scene predictions in the training phase.

Table of Contents

List of Figures

List of Tables

Abbreviations

AI Artificial Intelligence

ML Machine Learning

ANN Artificial Neural Network

CNN Convolutional Neural Network

SVM Support Vector Machine

RoI Region of Interest

NMS Non Maximum Suppression

IoU Intersection over Union

RCNN Region-based Convolutional Neural Network

AP Average Precision

Bbox Bounding Box

Chapter 1

Introduction

Context, in a general sense has been defined as [25]:

"any information that can be used to characterize an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."

Moreover, context can be considered a statistical property. It provides essential information for perceptual inference, and studies [26] have proved that location, relative size, object arrangements and other contextual information are key factors for human object detection. Furthermore, object recognition was more precise when the strength of the object-scene relationship was high. An understanding and application of this capability to use contextual information in modern day object detection algorithms can greatly help rather than hinder performance.

1.1 Motivation

The motivation behind this project lies in the need to improve the precision and effectiveness of object detection systems. Contextual information, including environmental factors and object relationships, can provide valuable clues that aid in the identification and localization of objects. By administering contextual knowledge, object detection systems can more effectively distinguish between similar objects, detect obstructed objects, and identify objects that are partially hidden. Additionally, contextual information can be used to forecast behaviour, which is advantageous in applications such as autonomous driving or robotics. Previous work in the field has utilized scene understanding techniques to infer contextual information from the environment, while others have incorporated object relationships and contextual cues directly into various stages of the detection pipeline. Ultimately, incorporating contextual information into object detection can lead to more dependable and efficient systems, with numerous practical applications in robotics and artificial intelligence.

1.2 Aim

The aim of the project is to develop and assess contextually aware object detection models using effective deep learning methodologies that take contextual information of an image into account and constructively apply this knowledge to improve their performance in comparison to a standard object detection model.

1.3 Objectives

The objectives of this project include:

- Conduct extensive research on object detection in computer vision and techniques to integrate contextual knowledge from images into the algorithm.
- Incorporate the practical use of contextual information into an object detection algorithm in order to improve performance.
- Evaluate and compare the proposed models with existing models based on relevant performance metrics and draw conclusions about its practicality.

1.4 Organization

This technical report is organized as follows into 5 chapters:

- Chapter 1 is the introduction of the report summarising the context of the project, stating the aim and objectives to be achieved, and the organization of the document.
- Chapter 2 contains the literature review discussing related work in technical literature in the field of object detection in computer vision.
- Chapter 3 provides a comprehensive explanation of the entire implementation of the project.
- Chapter 4 discusses the analysis of the results of evaluation of the developed deep learning models.
- Chapter 5 concludes the report by discussing challenges faced during implementation as well as limitations and future works.

The back matter of this project consists of the list of references of all documents and figures as well as appendices that are cited in the text.

Chapter 2

Background

In this chapter, a comprehensive review of the technical literature on context-aware object detection is outlined. Beginning with an overview of AI and Computer Vision, it then delves into different types of object detection models, including multi-stage, single-stage, and state-of-the-art (SOTA) frameworks. Next, the importance of context in object detection and its various different forms and uses in object detection is discussed. Finally, it presents a review of related works and offers a critical analysis of the current state of the field.

2.1 Overview of Artificial Intelligence

The field of computer science dedicated to the study, implementation and development of machines that imitate cognition is called Artificial Intelligence [1]. The vast field of AI comprises of several sub-fields that share a common goal, which is to implant capabilities such as organisation and classification of new knowledge, and discovery through perception and observation of the environment, into machines; essentially mimicking natural human intelligence [2].

A few examples of such sub-fields would be Neural Networks, Computer Vision, Robotics, Evolutionary Computation, and Natural Language Processing.

2.2 Computer Vision

Computer Vision is the field of artificial intelligence that is focused on attempting to reproduce the capability of human vision in computers [7]. The goal is to enable computers to detect, classify and process objects in images and video the way humans do. Computer vision algorithms are based on pattern recognition, and for it to work efficiently large amounts of visual data is required which, fortunately, is feasible as 720,000 hours of video and 3.2 billion images are shared online daily [8].

Modern computer vision is a branch of deep learning which is, in turn, a subfield of machine learning that makes use of Neural Networks. Artificial Neural Networks (ANNs), simply called Neural Networks, are algorithms inspired by the biological understanding of how the human brain functions. This new biologically inspired method of computing creates massive parallel networks that are trained to solve problems by finding and generalizing patterns the way the human brain does; this is contrary to the more rote approach of traditional programming [9].

2.2.1 Convolutional Neural Networks (CNN)

The type of ANNs that is mainly used to solve image-driven pattern identification tasks such as in computer vision is called Convolutional Neural Networks. These neural networks consist of three types of successive layers: convolutional layers, pooling layers, and fully connected layers [10]. The CNN architecture is given in Figure 2.1.

In the convolutional layer, the CNN makes use of kernels, which are basically 2D matrices of weights stacked together to form 3D filters that move through and convolve with each pixel in the input image to carry out the feature extraction of the input and produce resulting feature maps [10].

The aim of the pooling layers is to reduce the number of parameters; it achieves this by performing dimensionality reduction on the feature maps so as to reduce the computational complexity, this is called downsampling. The scaled down matrix is then sent to the fully connected layers [10].

The fully connected layers flatten the input from the convolutional and pooling layers to a one-dimensional vector which produces output scores that can be converted to probability scores for each class by a softmax layer, these scores are then what is used for classification of the objects in the image [11].

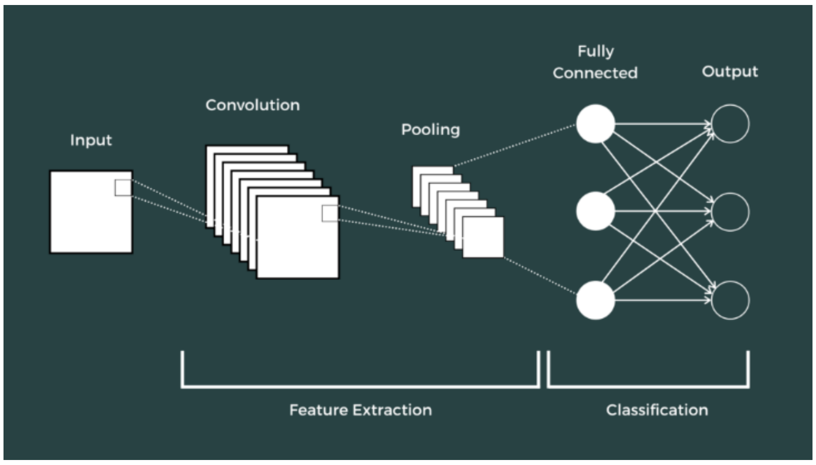


Figure 2.1: Basic architecture of CNN [39]

2.3 Object Detection

Object detection in computer vision has two steps, namely, object localization (to locate an object in an image or video) and object classification. While object classification is relatively easier as it can be done with great accuracy with the help of CNNs, the more challenging task is object localization for which several algorithms have been proposed.

Object detection can either be salient, which entails the identification of only the most discernible object in the image and ignoring its surroundings, or generic, which involves the detection of all existing objects in an image and locating them in the image with a labelled bounding box around each object [13].

Further, generic object detection frameworks are of two types, multi-stage (region proposal based) object detection and one-stage (regression/classification based) object detection. Multi-stage frameworks consist of mainly two steps, firstly the generation of region proposals and then attempting to classify objects in each of the given region proposal. In contrast, a one-stage framework does the object classification and bounding-box regression directly without the use of region proposals [12]. Generally, a one stage framework may be faster than a multi-stage detector, but this is at the cost of reduced performance. The differences between one stage and multi stage algorithms can be understood more clearly using Figure 2.2.

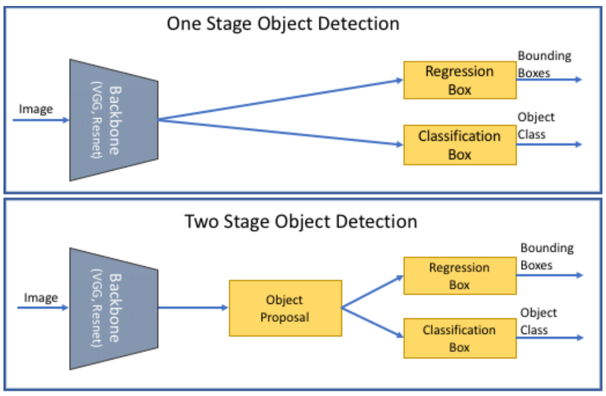


Figure 2.2: One-stage vs multi-stage (two-stage) object detection [40]

In the following sections, an in-depth discussion of several examples of one-stage and multi-stage object detection frameworks using deep learning is made.

2.3.1 Multi-Stage Frameworks

To generate region proposals for the task of object localization, there are two methods. Firstly, the sliding window algorithm or exhaustive search, where a "window" is used to locate regions by sliding the window over and checking every pixel of the image which is rather time consuming and computationally inefficient. Second, the selective search algorithm [31], which makes use of the graph-based segmentation technique described by Felzenszwalb et al [14]. The algorithm represents an image as a graph where each pixel is a vertex and the affinity between the vertices which is based on similarity between them is considered as the weight of the edge. Regions are segmented by finding cuts in the graph and separating different subsets of vertices into disjoint cut-sets [14]. Selective search then adds the bounding boxes corresponding to each segmented region to the list of region proposals, after this it tries to group similar adjacent segments iteratively, this process can be visualized using Figure 2.3. The factors contributing to similarity between segments are colour, size, texture, and shape. Hence larger segments are formed in each step by grouping smaller segments [31].

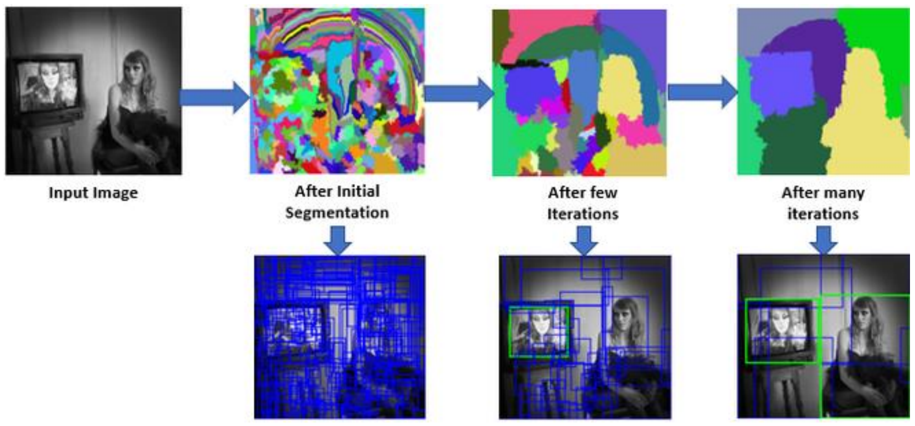


Figure 2.3: Generation of region proposals using Selective search algorithm [42]

2.3.1.1 Region-based Convolutional Neural Networks (RCNN)

One of the early predominant models to use CNNs for object detection was proposed by Girshick et al. [15]. It was a simple and scalable model which had a great impact in the field. Selective search [31] was used to create bounding boxes or region proposals (hence the name) for an input image. Around 2000 region proposals were generated for each image and passed through the feature extractor which is a CNN that has 5 convolutional layers and two fully connected layers at the end of which, was a Support Vector Machine (SVM) [30], that classified whether the given region contained an object, and what the object was [16]. However, even though this algorithm was fast during its time, it could not be used for real-time applications due to its high computational time [13]. The RCNN pipeline is described in Figure 2.4.

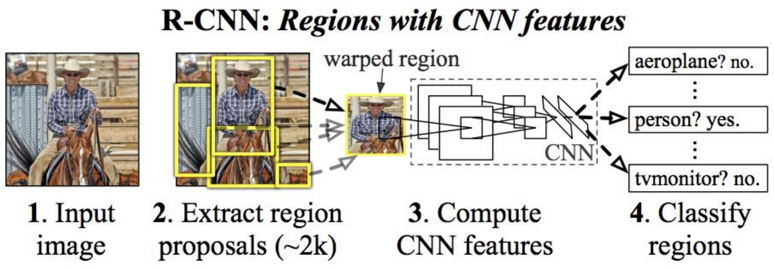


Figure 2.4: RCNN pipeline [15]

2.3.1.2 Fast RCNN

Girshick realized that a lot of the 2000 proposed regions overlapped with each other which resulted in unnecessary repeated runs of the same CNN. Moreover, inferior region proposal generation could be attributed to the selective search being a fixed algorithm which prevented any *learning* from taking place in the algorithm. Consequently, he improved his previous model by changing his algorithm. Instead of the CNN taking the region proposals as input, it was now fed the input image directly from which it produced convolutional feature maps [18]. Since the region proposals can be of different sizes, an RoI (Region of Interest) pooling layer is used to reshape them to a uniform size so it can be fed to the fully connected layers. This process is described in Figure 2.5.

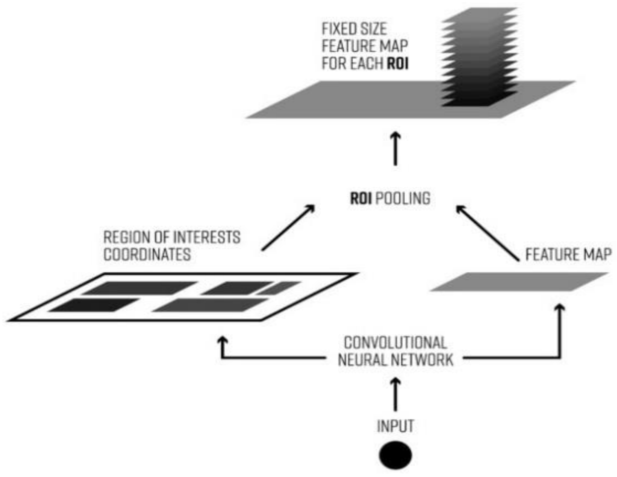


Figure 2.5: Object detection pipeline with RoI pooling [51]

The fully connected layers ultimately diverge to two output layers, namely the softmax layer, and a linear regression layer. The softmax layer outputs a classification, and the linear regression layer outputs bounding box locations for the detection. The Fast RCNN pipeline is given in Figure 2.6.

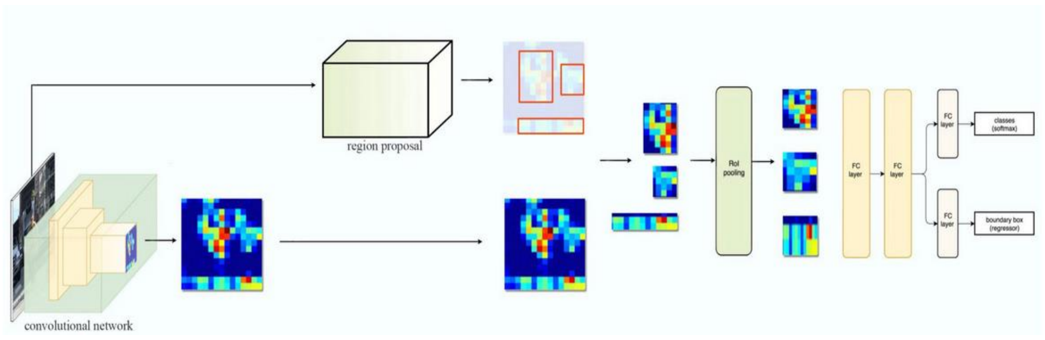


Figure 2.6: Fast RCNN architecture [43]

As the name suggests Fast RCNN was faster than RCNN because it eliminated the requirement to feed 2000 proposals to the CNN for every image, but the selective search was still quite slow and acted as a bottle neck for performance, yet another limitation was the requirement of object proposals with the input [16].

2.3.1.3 Faster RCNN

Ren et al. [19], proposed an algorithm that replaced selective search with Region Proposal Networks (RPNs) that uses a Fully Convolutional Network (FCN, different from CNNs in that it only contains convolutional layers) to generate region proposals. They additionally also had the capability of being trained on specific classes. Thus, by replacing the use of selective search, which was slow and inefficient, they greatly improved the performance of RCNNs [12]. The Faster RCNN pipeline is described in Figure 2.7.

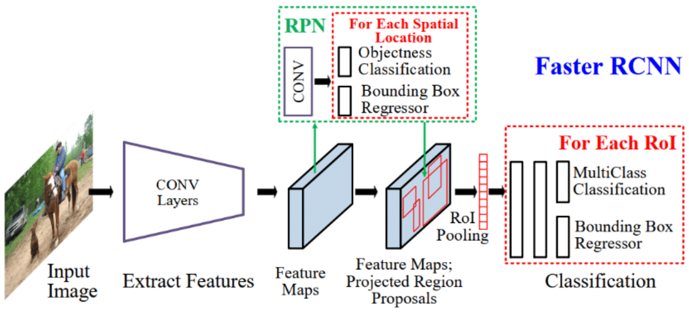


Figure 2.7: Faster RCNN architecture [44]

2.3.1.4 Region-based Fully Convolutional Networks (R-FCN)

This method proposes to solve two common issues faced in object detection namely, translation invariance for image classification and translation variance for object detection. Simply put, translation of an object in an image should not affect classification, but the translation of an object in a bounding box should be significant to object detection. This is achieved by making use of position-sensitive score maps. In R-FCN, the last convolutional layer is used to generate position sensitive score maps that divide the RoIs into k*k bins, and each bin is used for class identification [12]. This process is visualized for different scenarios in Figure 2.8.

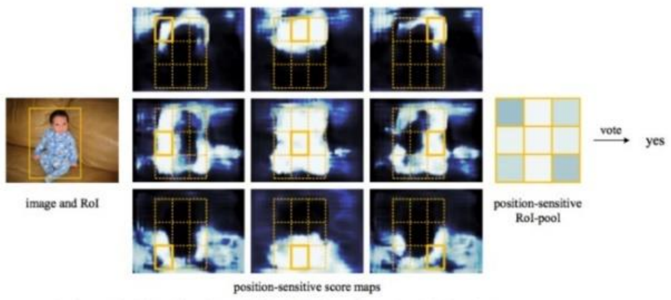


Figure 3: Visualization of R-FCN ($k \times k = 3 \times 3$) for the *person* category.

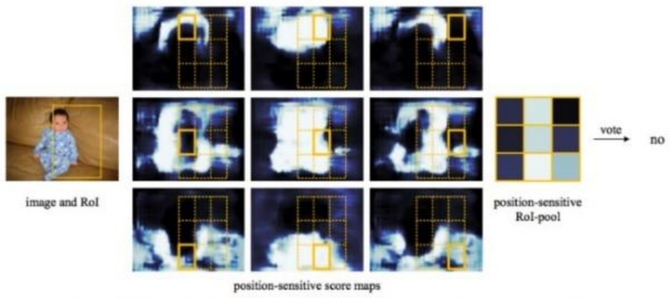


Figure 4: Visualization when an RoI does not correctly overlap the object.

3.

Figure 2.8: Examples of when an RoI aligns with an object and when an RoI is misaligned [22]

It is faster than previous region-based algorithms as RFCN is fully convolutional, and the entire computation is split on the whole image. Also, the feature maps are independent of RoIs and as a result they can be computed outside of each RoI [22]. The R-FCN pipeline is shown in Figure 2.9.

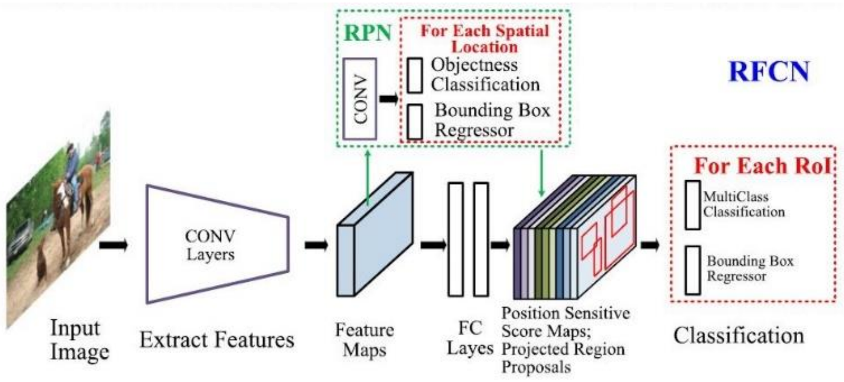


Figure 2.9: R-FCN architecture [45]

2.3.1.5 Mask R-CNN

Mask RCNN was worked towards the problem of object instance segmentation. Instance segmentation is the process of detecting and demarcating each distinct object in an image. Hence, it is essentially the mixture of two sub-problems: object detection and semantic segmentation. Semantic segmentation (otherwise known as background segmentation) is the process of assigning each pixel in an image to an object class. The differences, as well as the relation between these problems, can be better understood using Figure 2.10.

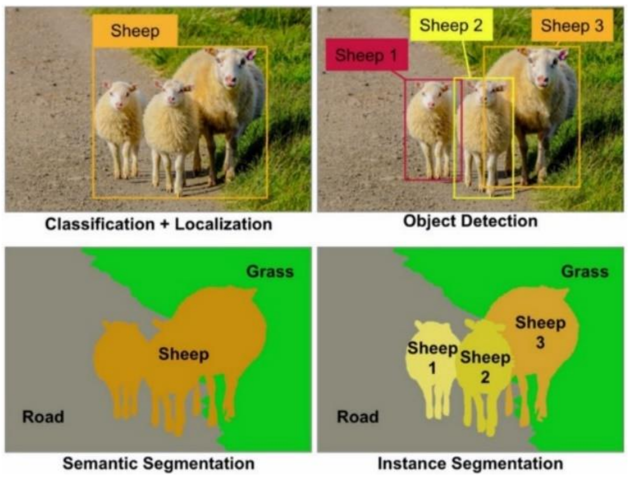


Figure 2.10: Object detection vs Instance segmentation vs Semantic segmentation [32]

Mask RCNN is built on top of Faster RCNN. To the class label and bounding box offset outputs of the Faster RCNN, it adds a third output branch for the object mask, which is different compared to the other outputs as it requires the extraction of finer spatial layouts of objects. It encodes these mask representations of objects' spatial layouts using a fully convoluted network. The other modification in Mask RCNN was the replacement of RoI Pooling, that was used in Faster RCNN, with an RoI Align mechanism. The main difference between them is that RoI Align does not use quantization for data pooling which helped to prevent misalignment and loss of pixel data [21]. The Mask RCNN architecture is provided in Figure 2.11.

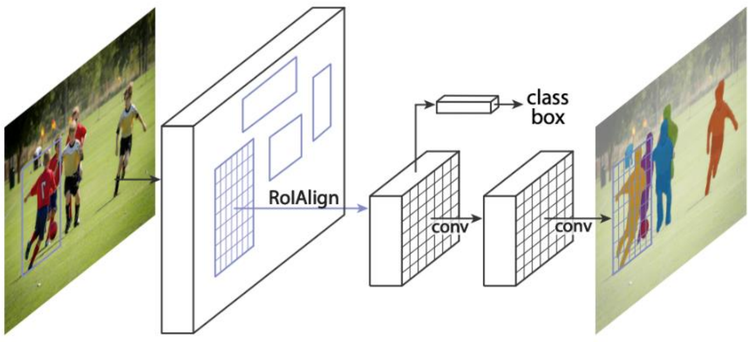


Figure 2.11: Mask RCNN framework [21]

The additional mask branch added only a small overhead but nonetheless outperformed all existing single model entries in every task in the COCO suite of challenges including instance segmentation ad bounding box object detection [20, 21]. The Microsoft COCO (Common Objects in Context) dataset is used as the benchmark dataset to compare performance of real time object detection algorithms [20].

2.3.2 One-Stage Frameworks

These are one-step frameworks that map directly from the pixels in an image to the coordinates of bounding boxes and the class probabilities [12] and hence are structurally simpler and less time consuming as they skip the region proposal step, in contrast to the multi-step frameworks based on region proposals.

2.3.2.1 Single Shot Multibox Detector (SSD)

This method uses a single deep neural network which takes the feature map from its base network and divides the output space of bounding boxes into boxes of distinct aspect ratios [23]. The model then produces the scores regarding the presence of the object for each default box for each object, and then scales it to fit the shape of the object by producing offset values for the locations [16]. The SSD architecture consists of several convolutional layers that progressively decrease in size which allows for predictions of numerous different scales. The final step is non-maximum suppression (NMS) which basically filters out bounding boxes regulated by a certain threshold value based on the overlap between bounding boxes to output final predictions. The SSD architecture is described in Figure 2.12.

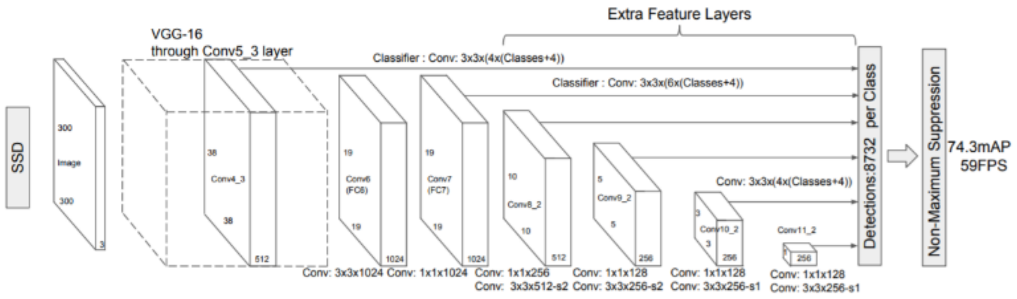


Figure 2.12: SSD architecture [46]

SSD outperforms Faster RCNN significantly in terms of accuracy and speed, and it outperforms YOLO in classification as it can detect objects better in multiple scales.

2.3.2.2 You Only Look Once (YOLO)

YOLO is one of the most recent and the most popular real-time object detection algorithm. This can be attributed to the fact that it is a very fast model and can process images in real time at 45 fps [24]. The model is different from others in that it uses the whole image during training, and it is a one-step detection model. It works by dividing the input image into a grid where each cell predicts respective bounding boxes and confidence scores. Each bounding box predicts coordinates and confidence scores, and each grid cell also predicts class probabilities depending on whether an object is present in the cell, this is done with the help of the sole CNN. This procedure is visualized in Figure 2.13. Finally, NMS is used to filter out the final detections.

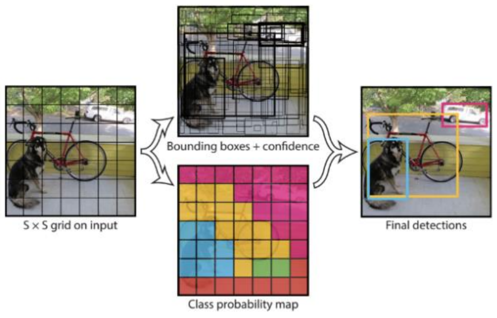


Figure 2.13: Object detection using YOLO [24]

Compared to the latest models YOLO makes less false positives but it does make more localization errors, which consequently leads to problems like difficulty in prediction of small objects appearing in groups [24]. Nevertheless, the model has lower computational demand and higher accuracy than any other real time detector, this can partly be accredited to the use of the entire image during training which enables the learning and use of more contextual information compared other models. The YOLO architecture is provided in Figure 2.14.

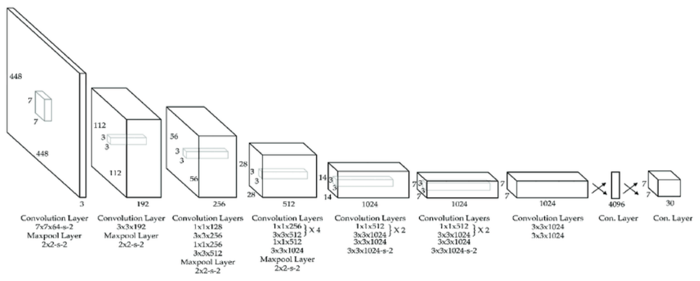


Figure 2.14: YOLO architecture [47]

2.3.3 State-of-the-Art Object Detection

The present-day models that achieve outstanding object detection performance are discussed briefly in this section.

2.3.3.1 InternImage

InternImage is a very recent, large-scale CNN-based foundation object detection model. An adaptation of which, InternImage-H, achieved a record accuracy of 65.4 mAP on the COCO test-dev dataset [52], outperforming all other leading CNN-based models. This exceptional performance was a result of effectively increasing parameters and training data used by the model compared to recent CNNs. As a result, the model is able to learn stronger and more robust patterns. Further, a large effective receptive field owing to use of deformable convolutional networks as the core operator helped to enhance the task of segmentation and detection [53].

2.3.3.2 YOLOv7

The YOLOv7 model is the latest iteration of the previously discussed YOLO object detection model and has considerably surpassed its predecessors in the YOLO series as well as all other object detection models in speed and accuracy with a 58.6% AP on the MS COCO dataset making it the fastest real-time object detection model thus far [54]. This was accomplished by means of optimization of the training process, in addition to the architecture optimization from previous versions. The optimization methods proposed were referred to as trainable bag-of-freebies because they reinforced the training cost for boosting accuracy without increasing inference cost [55].

2.4 Context in Object Detection

How is context used in object detection? Useful context can be thought of as "*any and all information that may influence the way a scene and the objects within it are perceived*" [27].

Many objects are often found together in a particular environment, for instance, animals in their natural habitat such as deer in the forest, other objects are likely to be found in particular geographic locations. The scene along with other objects in the image provides much semantic context. How an image is illuminated because of aspects such as the weather or camera parameters, affect the colour, shadows, and brightness of objects in an image. Information derived from three dimensional physical constraints such as a fire hydrant requiring a ground plane are all examples of useful contextual information [27].

The rather small number of extracted region proposals from regular algorithms in complex image scenes are less capable to detect smaller objects that may be suppressed or occluded and can lead to lower accuracy or misdetection [35]. This is because scores are calculated using only the content of the few selected proposals which cover only a minor part of the input image and as a result generally have lower confidence scores than larger regions and may get disregarded during NMS. The modification of these bottom-up processes to use context aware modules in a top-down processing fashion which summarize global context features at the image level to complement and assist the features in the pooled region proposals have shown to increase accuracy with little extra overhead. The overhead comes from a side task of determining whether specific object classes appear in the whole image at different scales and ignoring its location, these image level cues are integrated with the regional features for better classification. The strive to exploit global and local contextual information using different mechanisms [35, 36, 37] are implemented in the form of connective subnetworks that can be easily combined with existing models to aid the detection process. Global context consists of the visual information from the entirety of the image and local context consists of the visual information near the subject, this distinction is shown in Figure 2.15.

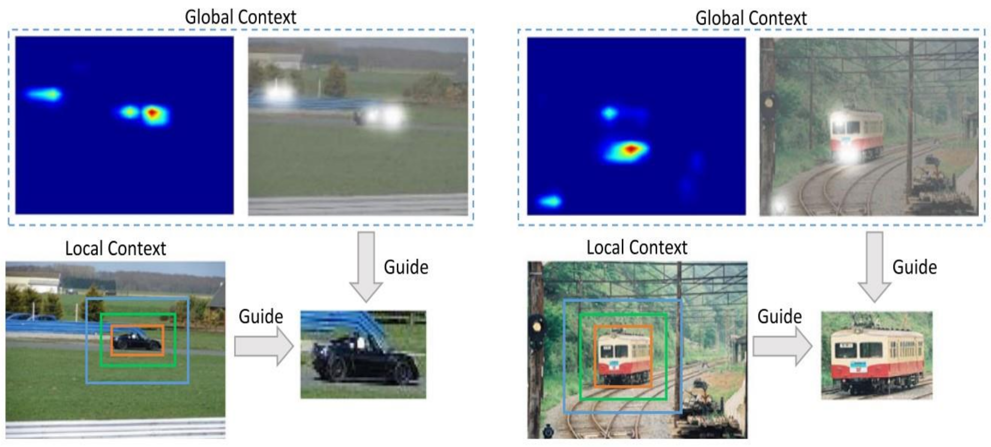


Figure 2.15: Global and local context [36]

However, it should be mentioned that not all background information is useful in developing performance, as the incorporation of irrelevant contextual noise can actually hinder performance, therefore it should be seen that the identification of helpful contextual information is the first step to creating a coherent model. Nonetheless, the ways in which contextual information can effectively be consolidated into object detection pipelines is still an area in the field that is relatively underexplored.

Context can be used in a multitude of approaches. Algorithms have made use of object relationships by implementing a spatial-context-based approach that can basically group co-occurring objects in the scene [33]. These objects are clustered into groups based on frequency of co-occurrence and, at a negligible drop in accuracy, have increased efficiency of the detection models as it could facilitate the prediction of the presence of other objects that are known to occur together based on these clustered groups. The object layouts in relation with each other in an image is useful information that can be captured with the help of graph networks because of their flexibility to describe pair-wise relations in space. Graph nodes are therefore object proposal regions which can now aid in the detection process. These graph networks leverage spatial and semantic information to produce interpretable graph structures that model spatial relations between objects to improve classification and localization [34].

An example of a context aware algorithm that is applicable to existing detection models uses context rescoring in post processing which assigns a new confidence score to each bounding box which considers confidence scores of all other bounding boxes thus taking into consideration location and sizes of boxes, and co-occurrences of objects [28]. Strong co-occurring relationships between objects is an important form of contextual cue that can be used to recognize classes as having a low *context score* could be attributed to low probability in co-occurrence, among other things. From Figure 2.16, it can be discerned that related objects co-occur frequently for example dining table and chair, and bat and ball. Infrequent co-occurrences are also clear for example tennis racket and forks etc.

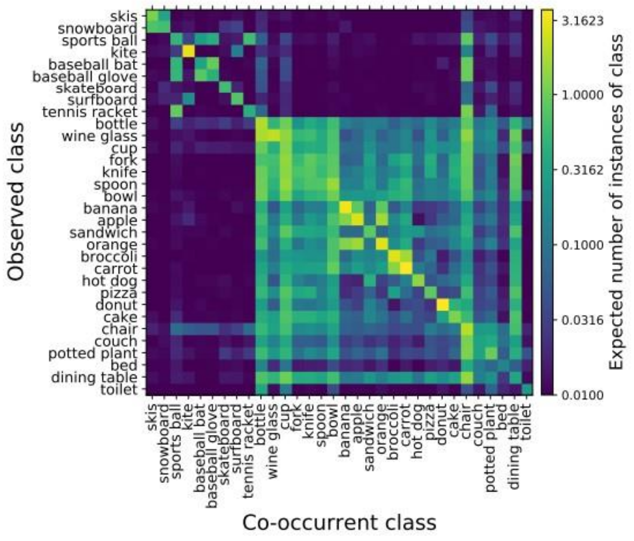


Figure 2.16: Object co-occurrences in a subset of classes in the COCO train2017 dataset [29]

Rescoring works to improves performance by assigning a new higher confidence score to true positives rather than false negatives so that those detections with correct class and better bounding box coordinates predictions survive longer in the algorithm than detections with relatively poor predictions. The rescoring algorithm is trained with a loss that seeks average precision maximization based on the idea that better localizations should be scored higher. The discussed model displayed an average precision improvement of 0.5 to 1 over strong region-based detectors on MS COCO 2017. Though a modest improvement, the model showed consistent improvements in accuracy by maintaining confidence of correct detections and decreasing it for out of context and duplicate detections [29].

2.5 Related Work & Critical Analysis

This chapter discusses related works and offer a critical analysis of the current state of the field.

2.5.2 Related Works

Optimizing the trade-off between the different types of object detection frameworks

State of the art object detection algorithms can be reasonably divided into two main types. On one hand, two stage detectors attain high detection accuracy rates despite being computationally slower. On the other hand, one stage detectors achieve lower accuracy rates but are much faster than two stage detectors. For example, among models tested on the PASCAL VOC (Visual Object Classes) 2007 testing set, R-FCN had a mean average precision (mAP, discussed in Chapter 4) of 76.4 while YOLO v1 had a mAP of 63.4. However, YOLO performed at 45 frames per second (FPS) whereas R-FCN ran at 5 FPS [56]. This raises a question about the possibility of an efficient strategy to optimize the trade-off between accuracy and speed in object detection. Attempts made at solving this problem have seen the testing of different configurations of frameworks to optimize detection in different scenarios by changing various parameters and components. An alternative method [49] proposed the use of an image-difficulty predictor to split the dataset into easy and hard images, wherein the objects in easy images could be straightforwardly detected by the faster single stage detector and the harder images would be sent to the more precise two-stage detector for accurate predictions. They concluded that the simple approach of using image difficulty as the primary factor to split the dataset compared favourably to a random set of images.

Vision Transformers

In 2022, Vision Transformers (ViTs) have emerged as a strong competitor to the current state of the art CNNs in image recognition tasks. ViT based models have been shown to outperform CNN based models by almost 4x in computational speed and accuracy [57]. The transformer architecture is currently the go-to for Natural Language Processing

(NLP) tasks and is only recently gaining attention in the field of Computer Vison. The first model using ViTs in computer vision was introduced in 2021 by the Google Research Brain Team. It is a deep learning model that works by differentially weighing the significance of each portion of the input data to get an understanding of the local and global features of the image and hence achieve higher precision rates on large datasets with shorter training time. However, the optimization of the parameters of ViTs is more difficult compared to CNNs. Additionally, ViT models outperform CNNs only if they are trained on substantially large datasets, if not, the CNN based models are the superior choice when trained on mid-sized datasets such as ImageNet training set which contains 1.2 million images [58].

2.5.3 Critical Analysis of Literature

A critical analysis of the background in the field of context aware object detection reveals that object detection is a well-researched field as the subject of computer vision has been explored extensively since the 1960s. Concepts such as extraction of image features seen initially in traditional object detection using Histograms of Oriented Gradients (HOGs) and Deformable Part based Models (DPM) are used to this day in modern object detection. The fusion of deep learning in computer vision through the incorporation of CNNs was pivotal in the rebirth of the field after it had seen a plateau in its development as the performance of handcrafted features were saturated. A major turning point was the proposal of region based CNNs as they significantly improved classification performance. The incremental progress of two stage detectors brought about the rise of one stage detectors which greatly improved the speed of operation of the models to a point that they could now facilitate the use of object detection algorithms in real time applications such as in security systems and autonomous vehicles.

Chapter 3

Project Implementation

This chapter discusses the entire development process of the project including the software and hardware environment, project methodology and workflow, and the implementation pipeline of each model in great detail.

3.1 Development Environment

The project is developed on a hardware configuration consisting of an i7-8565U CPU with 16GB RAM. The code is written in Python 3 and executed on Jupyter Notebooks within an Anaconda environment. TensorFlow 2 [3], the widely known open-source machine learning framework developed by Google, in conjunction with the Keras API [4], which provides a high-level user-friendly interface for building and training artificial neural networks in Python were used extensively in the project to build, train, and deploy the deep learning models. To annotate, pre-process, augment, and manipulate data, several libraries such as OpenCV [5], labelme [6], NumPy [74], Albumentations [38], and Scikit-learn [73] were utilized.

3.2 Source Code

The source code implemented in 4 Jupyter Notebooks and the datasets created to train and test the models are available in a GitHub repository accessible with this link. https://github.com/syedayman/Context-Aware-Object-Detection. The implementation of the scene classification and object detection models are inspired by code from GitHub repositories ‘Transfer-Learning-ResNet-Keras’ [75] and ‘FaceDetection’ [76].

3.3 Project Methodology

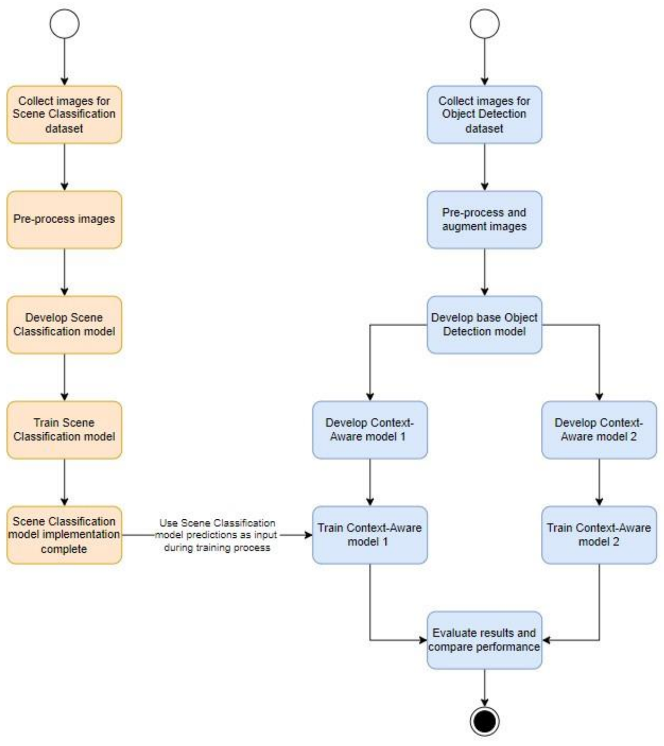
The essence of the project is to develop a contextually aware system for object detection. The way this problem was approached was to first develop an object detection algorithm, then choose a source from the vast variety of sources of contextual information in images, to finally, figure out a way to consolidate that information into the training process of the model in a way that would ideally improve its performance.

Consequently, the choice of machine learning framework used to develop the model had to be one which allowed for greater flexibility and control over the architecture and training process. This is precisely why the Keras API is used to develop the models despite the availability of other options such as the TensorFlow Object Detection API [59], which does not facilitate customization of deep learning models at a lower level. However, unlike for image classification tasks, Keras applications does not provide modular pre-trained models trained on object detection tasks, nor does it provide usage examples and documentation for it. This is why an object detection model had to be developed from scratch which was fortunately possible using the Keras Functional API [60].

As for the choice of contextual information used, ‘scene information’ from images was the selection made. By analysing the scene or background of an image, a model can gain a better understanding of the objects present in an image and their spatial relationships with one another as well as and with their background. Additionally, scene information can provide semantic cues about objects present in the scene and improve the robustness of object detection models to changes in lighting, background, and other environmental factors because it can help the model to differentiate between objects and their surroundings [61].

To use this information in the training process, predictions on image scenes made were used as one of the components contributing to the computation of the total loss during the training of the model which should effectively help the model “learn” the additional information. This process is discussed in greater detail in Section 3.6.

Figure 3.1 below describes the implementation workflow of the project’s methodology.



Collect images for Scene Classification dataset
Pre-process images
Develop Scene Classification model
Train Scene Classification model
Scene Classification model implementation complete
Use Scene Classification model predictions as input during training process
Collect images for Object Detection dataset
Pre-process and augment images
Develop base Object Detection model
Develop Context-Aware model 1
Develop Context-Aware model 2
Train Context-Aware model 1
Train Context-Aware model 2
Evaluate results and compare performance

Figure 3.1: Project Workflow diagram

Finally, results obtained from the evaluation of two context-aware models, each developed to use context in a different way could help provide a conclusive answer regarding the effectiveness of administering context-aware properties to aid object detection.

The following sections discuss the development process of each individual model and the respective datasets.

3.4 Scene Classification Model Implementation

This section discusses the implementation of a scene classification model that is subsequently used in the training of a context-aware object detection model. It covers the data collection process, pre-processing techniques applied to the data, and development of the model.

3.4.1 Data Collection and Pre-processing

Images of various scenes used in the dataset were collected from Google Images using a Chrome extension called 'Download All Images' [62] that allows for downloading images in bulk. The images were then manually filtered to remove undesirable or disparate images and incompatible file types. The dataset consists of 770 images divided into four classes, namely 'beach', 'forest', 'sky', and 'street'. The scenes were chosen based on where the object classes from the object detection dataset were most likely to appear in images. For example, the scene classes of 'beach' and 'street' correspond to the scenes where the object class of 'football' from the object detection dataset (Section 3.5.1) is likely to appear. This was done to help the model learn scene information in images that could be used to improve the performance of the context-aware object detection model. Each class consists of approximately 200 images to ensure the class distribution is fairly balanced in the dataset, to prevent the model from becoming biased towards any one class. Examples of images in the dataset for each class is given below in Figure 3.2.

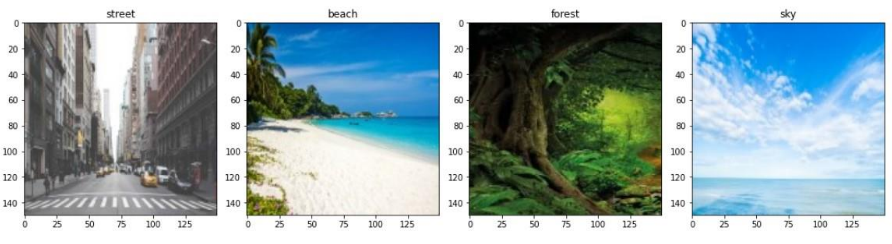
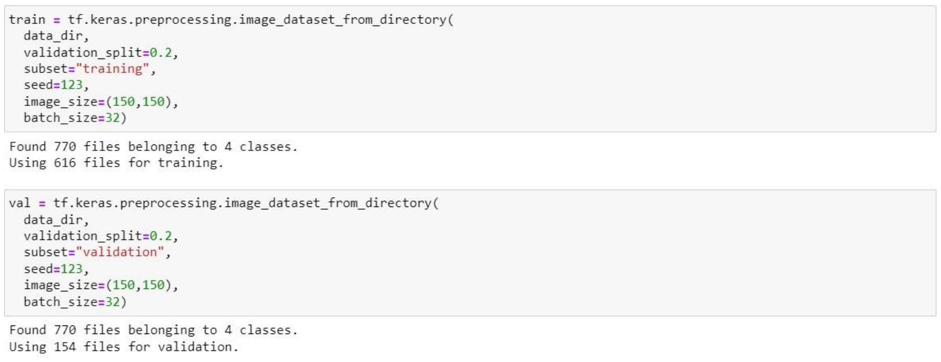


Figure 3.2: Examples of images in the Scene Classification dataset

The TensorFlow *tf.keras.preprocessing.image_dataset_from_directory()* utility was used to split the data by an 80:20 ratio into batches of 32, which resulted in a training set of 616 images and a validation set of 154 images that were resized to 150 x 150 pixels each as shown in Figure 3.3. This was done as the images collected were of varying sizes and since the model only accepts input of equal size, it also has additional benefits such as improving memory efficiency and generalization of the model during training.



```
train = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(150,150),
  batch_size=32)
```

```
Found 770 files belonging to 4 classes.
Using 616 files for training.
```

```
val = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(150,150),
  batch_size=32)
```

```
Found 770 files belonging to 4 classes.
Using 154 files for validation.
```

Figure 3.3: Code snippet showing the splitting and resizing of the dataset

3.4.2 Model Development

The scene classification model was built using the Keras *Sequential()* class [63] which allows for building deep learning models by stacking desirable layers in a configurable sequence.

To achieve a higher accuracy and better generalization with less data, Transfer learning was employed to build the model. Transfer learning is a machine learning method where a model trained on one task is used as a starting point for another related task. It involves taking a pre-trained model that has already learned features from a large dataset and using the weights it has learnt to perform a new task with a smaller dataset and lesser training time than with a new model from built from scratch [64]. The pre-trained ResNet50 model [68] was trained on the large ImageNet dataset [65] which consists of 1000 classes, a few of which are similar to the classes in the scene dataset. For example, ‘seashore’ is a class is comparable to the ‘beach’ scene in the constructed dataset. Thus, it was used as the base for the model as a feature extractor.

A *Flatten()* layer was added to flatten the input to one-dimension, and two subsequent *Dense()* layers were added as well which are essentially fully-connected layers that feed all of the previous layer’s outputs to all of its output neurons for classification. The final Dense layer had four output nodes, corresponding to the four classes in the dataset, and it was defined with the softmax activation function to convert the model's raw output into probability scores for each class. The model was compiled using ‘sparse categorical cross-entropy’ as the loss function since the target labels were integer encoded, ‘adam’ optimizer for its fast computation property, and ‘accuracy’ as the evaluation metric.

The values of the hyperparameters were deemed appropriate as the EarlyStopping callback was used on the validation accuracy. Keras EarlyStopping [66] is a callback that allows you to stop the training of a neural network model based on a specified condition. The callback was used with the ‘patience’ parameter set to 5, so it would wait 5 epochs without an increase in validation accuracy to stop training. It is useful to prevent overfitting and help save time by stopping the training early if the model has already converged and is unlikely to improve further.

The model was trained with a batch size of 32 for 13 epochs and obtained an **accuracy of 93.51%** on the constructed scenes dataset.

3.5 Object Detection Model Implementation

This section discusses the implementation of the object detection model built to detect 8 distinct objects. It covers all aspects of the development pipeline including image collection for the dataset, annotation of the images, data augmentation techniques, pre-processing techniques, loading the data, and finally, the development, compilation, various parameter settings, and training of the model. The structure of this model was used as the foundation on top of which the functionalities of the subsequent context-aware object detection models were built.

3.5.1 Data Collection and Annotation

Finding an appropriate pre-built object detection dataset with the all the necessary requirements for the task proved to be challenging. Existing datasets were either too small with only one primary class, or too large with too many classes in an unbalanced distribution like in many popular datasets such as COCO. Further, combining multiple small datasets would be cumbersome as they may have different image sizes or annotation formats, or both. Therefore, the decision was made to manually construct a dataset. Although creating a dataset from scratch is a time-consuming and laborious process, it provides complete control over the data and ultimately leads to a more precise model.

The images for the object detection dataset were carefully collected from Google Images to ensure a good quality dataset was constructed. Images were chosen in such a way that objects from the object classes may appear in one or two scenes from the scene detection dataset. The object detection dataset comprises of 8 classes, namely ‘car’, ‘football’, ‘apple’, ‘umbrella’, ‘bird’, ‘moon’, ‘sandcastle’, and ‘airplane’. The datasets were designed with an overlap between objects and scenes to prevent a one-to-one relationship connecting them. Hence when the object detection dataset is used to train the context-aware models, the scene input to the model could provide extra information about what objects could be recognized. Additionally, having a many-to-many relationship between the objects and scenes inhibited reducing the problem to merely scene recognition, which would have already been provided as input. This configuration should help the model learn object-scene relationships and thereby improve its performance. Using Figure 3.4 below, one can gain an understanding of the object-scene relationship present in the two datasets.

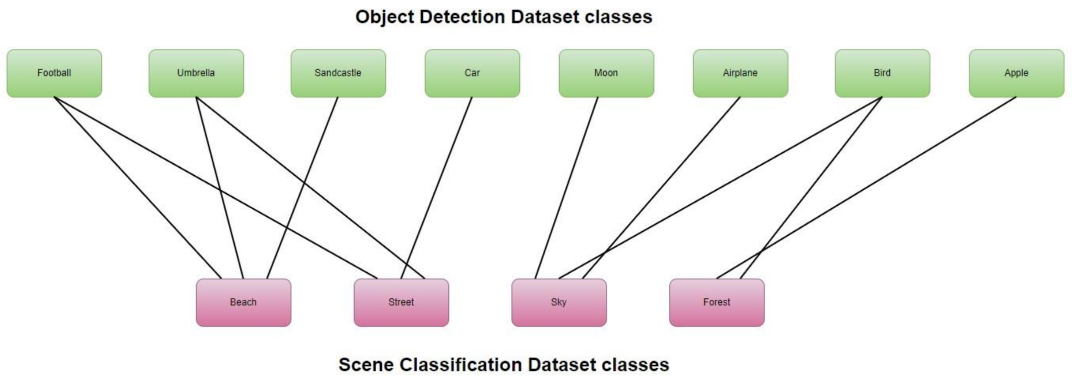


Figure 3.4: Relationship between the objects and scenes in their respective datasets

Examples of images in the object detection dataset are given below in Figure 3.5 to help visualize this. For instance, the 'umbrella' object class may appear in the 'beach' scene or the 'street' scene in the dataset.

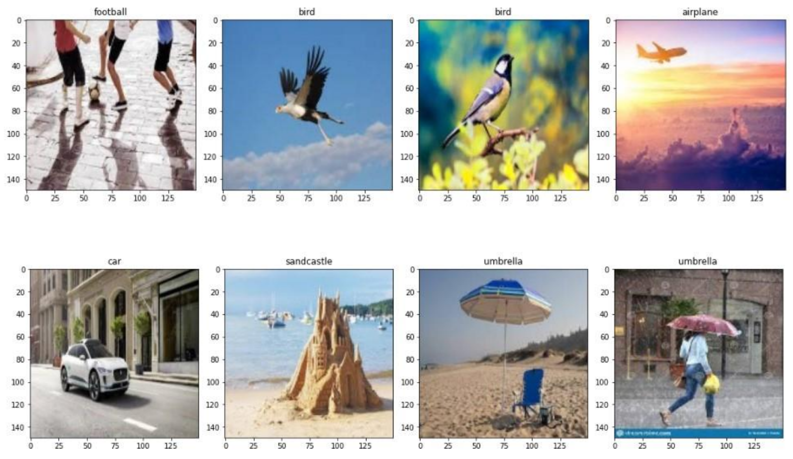
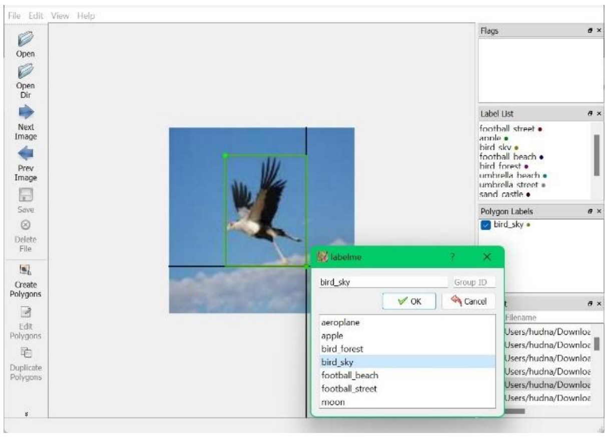
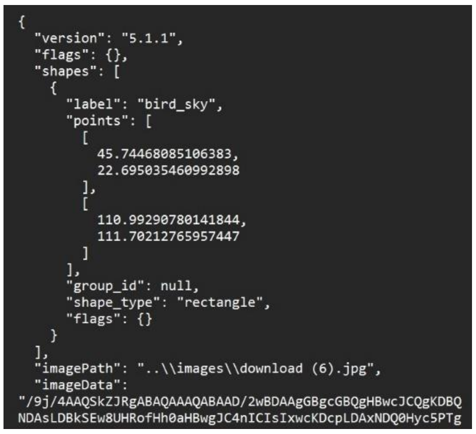


Figure 3.5: Examples of images in the Object Detection dataset

The label and bounding box annotations were done using a polygonal annotation library called Labelme [6] (not to be confused with LabelMe which is another an open-source graphical annotation tool developed by MIT on which it is based). It has various capabilities like semantic and instance segmentation, and bounding box annotations that can be saved in COCO or VOC format. The image class label and bounding box coordinates for each image in this dataset were exported in the VOC format and saved as JSON files. A total of 160 images were annotated manually in the dataset containing 2528 images, the rest of the 2368 were done using an augmentation software (discussed in Section 3.5.3). Figure 3.6 (a) below shows an example of an image being annotated using labelme and Figure 3.6 (b) is the resulting JSON annotation file output for the same image.



(a) creating bbox annotations in labelme



```
{
  "version": "5.1.1",
  "flags": {},
  "shapes": [
    {
      "label": "bird_sky",
      "points": [
        [
          45.74468085106383,
          22.695035460992898
        ],
        [
          110.99290780141844,
          111.70212765957447
        ]
      ],
      "group_id": null,
      "shape_type": "rectangle",
      "flags": {}
    }
  ],
  "imagePath": "..\\images\\download (6).jpg",
  "imageData":
"/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQ
NDAsLDBkSEw8UHRofHh0aHBwgJC4nICIsIxwcKDcpLDAxNDQ0Hyc5PTg
```

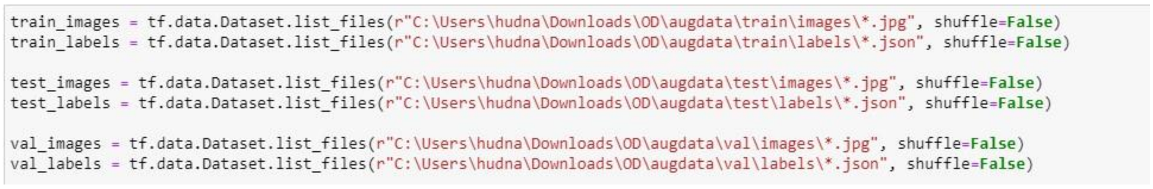
(b) resulting annotation file

Figure 3.6: Annotation creation process

3.5.2 Data Pre-processing

To prepare data for training certain pre-processing steps were taken to refine the quality of the data, and ergo the performance of the model. The dataset was constrained to only include images of JPG format to make the process of loading and handling the images easier.

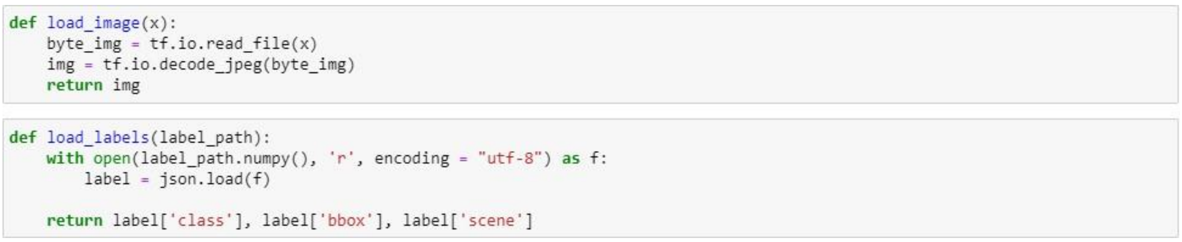
To create a descriptive input pipeline of images the TensorFlow *tf.data.Dataset* API [67] was used, which allowed for easy element-wise application of transformations using lambda functions and the ability to iterate and process its elements efficiently. The *list_files()* function creates a 'Dataset' object of all files matching one or more glob patterns which in this case is '.jpg'. The shuffle argument is set to 'False', so that the file paths are returned in the order they are found in the directory as shown in Figure 3.7



```
train_images = tf.data.Dataset.list_files(r"C:\Users\hudna\Downloads\OD\augdata\train\images\*.jpg", shuffle=False)
train_labels = tf.data.Dataset.list_files(r"C:\Users\hudna\Downloads\OD\augdata\train\labels\*.json", shuffle=False)

test_images = tf.data.Dataset.list_files(r"C:\Users\hudna\Downloads\OD\augdata\test\images\*.jpg", shuffle=False)
test_labels = tf.data.Dataset.list_files(r"C:\Users\hudna\Downloads\OD\augdata\test\labels\*.json", shuffle=False)

val_images = tf.data.Dataset.list_files(r"C:\Users\hudna\Downloads\OD\augdata\val\images\*.jpg", shuffle=False)
val_labels = tf.data.Dataset.list_files(r"C:\Users\hudna\Downloads\OD\augdata\val\labels\*.json", shuffle=False)
```

Figure 3.7: Code snippet showing the creation of images and labels Dataset objects for each dataset split

Helper function *load_JPG()* (depicted in Figure 3.8) reads an image file in binary format and then decodes the JPG image to return a Tensor of type uint8 (unsigned integer 8-bit). Tensors can be understood as multidimensional arrays of a uniform type. Another helper function *load_labels()* (depicted in Figure 3.8) loads the labels for an image from a JSON file and returns the list of class labels, the list of bounding box coordinates, and the list of scene labels (the scene labels were not annotated here but during the during augmentation process, section 3.5.3)

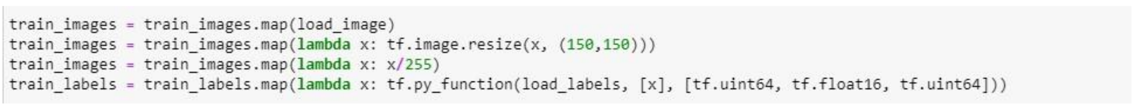


```
def load_image(x):
    byte_img = tf.io.read_file(x)
    img = tf.io.decode_jpeg(byte_img)
    return img
```

```
def load_labels(label_path):
    with open(label_path.numpy(), 'r', encoding = "utf-8") as f:
        label = json.load(f)

    return label['class'], label['bbox'], label['scene']
```

Figure 3.8: Code snippet showing the helper functions for loading data

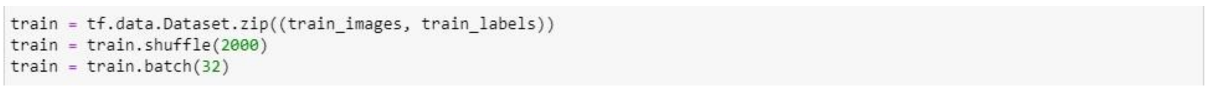
The images were resized to a uniform size of 150x150 pixels using the *tf.image.resize()* function which stretches images to fit the dimensions so that no part of the image is cropped out. Then the pixel values of the images were normalized by dividing each pixel value by 255, these processes could be completed effortlessly by virtue of lambda functions. Normalizing the pixel values of images can help in faster convergence during training, this is because normalization helps in reducing the variance in pixel values between different images and between the pixels within each image in the dataset. The pre-processed image data is then stored in the *train_images* Dataset object. To load the labels, the *tf.py_function()* function is used because the *load_labels()* function does not operate on TensorFlow tensors, but rather on Python objects. By wrapping the *load_labels()* function in *tf.py_function()*, it could be used to process the label file paths within the TensorFlow pipeline. The resulting training labels are returned as a tuple of a NumPy array of uint64 class labels, a NumPy array of NumPy arrays of 4 float16 bounding box coordinates, and a Numpy array of uint64 scene labels. This annotation data is stored in the *train_labels* Dataset object as shown in Figure 3.9.



```
train_images = train_images.map(load_image)
train_images = train_images.map(lambda x: tf.image.resize(x, (150,150)))
train_images = train_images.map(lambda x: x/255)
train_labels = train_labels.map(lambda x: tf.py_function(load_labels, [x], [tf.uint64, tf.float16, tf.uint64]))
```

Figure 3.9: Code snippet showing data loading and pre-processing for the training set

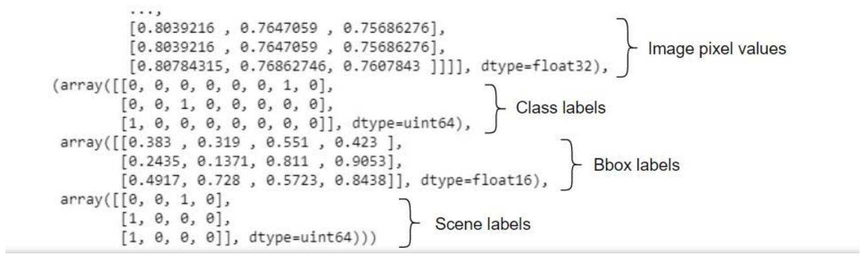
The Dataset objects were created for the validation and training sets in a similar manner. Lastly, the images and labels were combined to a single Dataset using the *tf.data.Dataset.zip()* module and assigned to variable *train* (shown in Figure 3.10). The data could now be shuffled and assorted into tuples of images and labels into batches of 32 which could be passed to the model for training, validation, and testing.



```
train = tf.data.Dataset.zip((train_images, train_labels))
train = train.shuffle(2000)
train = train.batch(32)
```

Figure 3.10: Code snippet showing the creation of the final 'train' Dataset object

The Dataset objects for the validation set (*val*) and test set (*test*) were created in the same manner. To observe the contents of the *train, val,* or *test* objects, the *as_numpy_iterator().next()* functions can be used with the appropriate indexing. The output is shown in Figure 3.11.



```
         ...,
         [0.8039216 , 0.7647059 , 0.75686276],
         [0.8039216 , 0.7647059 , 0.75686276],
         [0.80784315, 0.76862746, 0.7607843 ]]]], dtype=float32),
(array([[0, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0]], dtype=uint64),
 array([[0.383 , 0.319 , 0.551 , 0.423 ],
        [0.2435, 0.1371, 0.811 , 0.9053],
        [0.4917, 0.728 , 0.5723, 0.8438]], dtype=float16),
 array([[0, 0, 1, 0],
        [1, 0, 0, 0],
        [1, 0, 0, 0]], dtype=uint64)))
```

Figure 3.11: Example of the content inside each Dataset object

Moreover, to get a better understanding of the structure of the three Dataset objects a flowchart is provided in Figure 3.12 below.

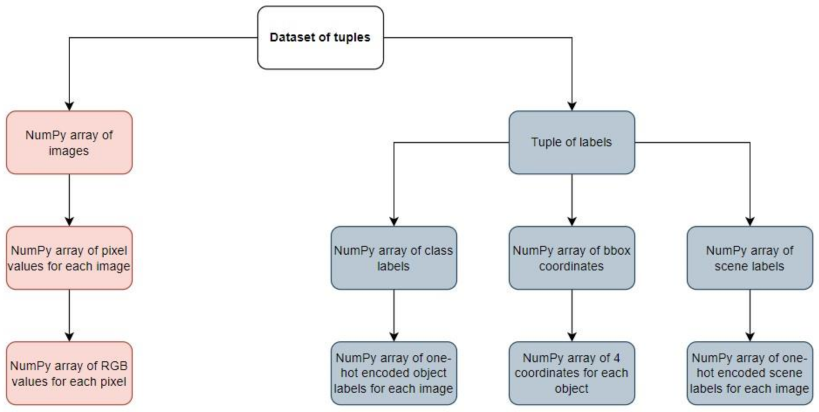

Dataset of tuples
NumPy array of images
NumPy array of pixel values for each image
NumPy array of RGB values for each pixel
Tuple of labels
NumPy array of class labels
NumPy array of bbox coordinates
NumPy array of scene labels
NumPy array of one-hot encoded object labels for each image
NumPy array of 4 coordinates for each object
NumPy array of one-hot encoded scene labels for each image

Figure 3.12: Structure of each Dataset object

3.5.3 Data Augmentation

Augmentation of images in the dataset was necessary for several reasons. Firstly, it helped to increase the size of the dataset, which was especially important since the original dataset of manually collected and annotated images was very small as it contained only 20 images per class which made for a total of 160 images. Secondly, it helped reduce overfitting by exposing the model to different variations of the same image, this enabled the model to learn the underlying patterns of the image, rather than just memorizing the training data. Finally, it helped improve the robustness and flexibility of the model, as it learnt to recognize objects under different lighting conditions, angles, and perspectives.

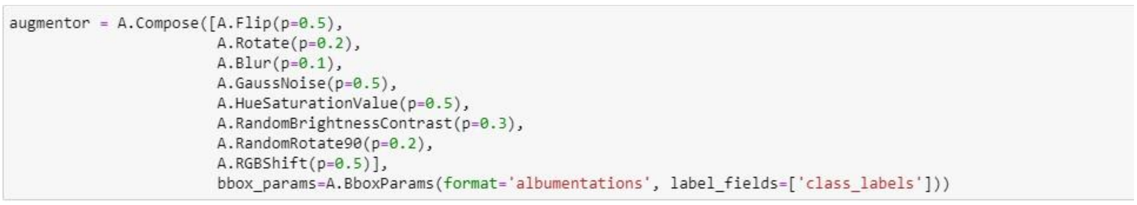
Albumentations [38] is a python library that can be used to efficiently implement a rich variety of image transform operations in a fast and flexible fashion. Additionally, it has the functionality to create new and accurate bounding boxes for the objects in the augmented images which was ideal as it obviated the need to annotate the augmented images manually which would have been a very time-consuming and tedious process. Therefore, the albumentations library was chosen for data augmentation and the various image transforms used are described below.

- **Flip:** Flipping an image horizontally or vertically is helpful to increase the number of samples in the dataset while maintaining the same class label. It is particularly useful when the orientation of the object in the image is not significant.
- **Rotate:** Rotating an image at different angles can help the model generalize better to images with objects in different orientations which is important when the object can appear in different orientations in real-world scenarios.
- **Blur:** Adding blur to an image can help the model generalize better to images that may have motion blur or may not be in focus
- **Gaussian Noise:** Adding Gaussian noise to an image can help the model become more robust to noisy images which is helpful when the input images are not of high quality.
- **Hue/Saturation/Value:** Adjusting the HSV values of an image can help the model become more robust to changes in lighting conditions. Since lighting

conditions can vary significantly in real-world scenarios the model should be able to detect objects well in all types of lighting.

- **Brightness and Contrast:** Adjusting the brightness and contrast of an image can further help the model become more robust to changes in lighting conditions.
- **RGB Shift:** Shifting the RGB channels of an image is useful since objects can appear in all types of different colours.

Each transform function had a parameter adjusting the probability of the transformation being applied on an image which are also required to be specified as shown in Figure 3.13.



```
augmentor = A.Compose([A.Flip(p=0.5),
                       A.Rotate(p=0.2),
                       A.Blur(p=0.1),
                       A.GaussNoise(p=0.5),
                       A.HueSaturationValue(p=0.5),
                       A.RandomBrightnessContrast(p=0.3),
                       A.RandomRotate90(p=0.2),
                       A.RGBShift(p=0.5)],
                       bbox_params=A.BboxParams(format='albumentations', label_fields=['class_labels']))
```

Figure 3.13: Code snippet showing the transformation function used to apply augmentations

Examples of the augmented images in the dataset are shown below in Figure 3.14 with the automatically constructed bounding box drawn over it.

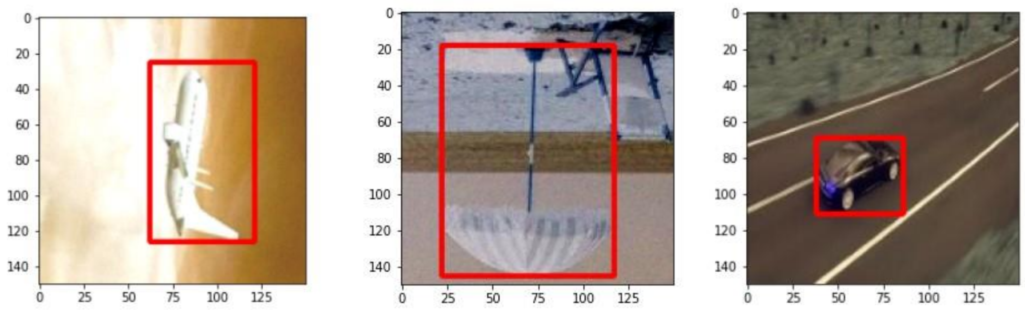
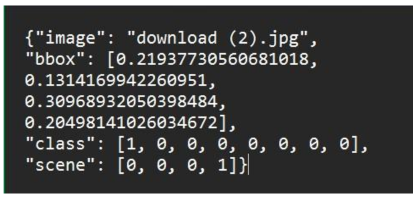


Figure 3.14: Examples of augmented images and bbox annotations in the dataset

A function was written to create augmented versions from a set of original images and associated annotations. It looped through folders containing the original data and reads in each image and the associated annotation file if, it existed. It reads the annotation information from VOC format and converts it to a format used by albumentations. Both formats are very similar in that they both use the top left and bottom right coordinates of each bounding box; the difference is that albumentations uses normalized coordinate values. It then uses the *Compose()* object 'augmentor' (shown in Figure) to create 16 new, augmented versions of each image, each with a different filename, and writes each new image to a new folder. It also creates new annotation files for each augmented image, using the same naming scheme as the images. The new annotations were modified to use one-hot encoding for the class labels and the new format of bbox coordinates. An extra label was used to specify the 'scene' of each image which is also one-hot encoded, this can be seen in Figure 3.15. This is the attribute that annotates the scene in which the object in the image is found. As discussed in Section 3.5.1 the images were collected for this dataset in such a way that the object is found in its most common natural scene, and it is this scene data which is the contextual information that is leveraged when training the model.



```
{"image": "download (2).jpg",
"bbox": [0.21937730560681018,
0.1314169942260951,
0.30968932050398484,
0.20498141026034672],
"class": [1, 0, 0, 0, 0, 0, 0, 0],
"scene": [0, 0, 0, 1]}
```

Figure 3.15: Example of the final annotation file for each image

Consequently, the size of the dataset had now been increased to 2528 images (180x16=2560, but a few images were removed to account for even divisibility of the total size of the dataset with the batch size of 32). The ratios for the train, validation, and test splits are described in the given flowchart in Figure 3.16.

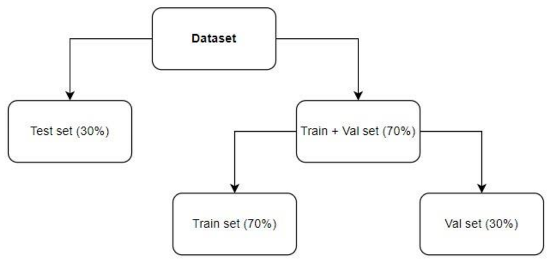


Figure 3.16: Flowchart depicting the train-val-test split in the dataset

The total number of images in each set and the actual distribution of each class in each split is described in the Tables 1 and 2 below. Please note the distribution is not exactly a 70-30 split between sets as the split sets had to be slightly adjusted to account for even divisibility of each set with the batch size while training.

Number of training images	1280
Number of validation images	544
Number of testing images	704
Total	2528

Table 1: Number of images in each set

Set\Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Total**
Train	160	128	160	128	160	160	200	204	1280
Validation	64	64	64	96	64	64	64	64	544
Test	96	80	96	64	88	88	96	96	704

Table 2: Class distribution in each set

The evaluation metrics were adapted to take the minor class imbalances into account.

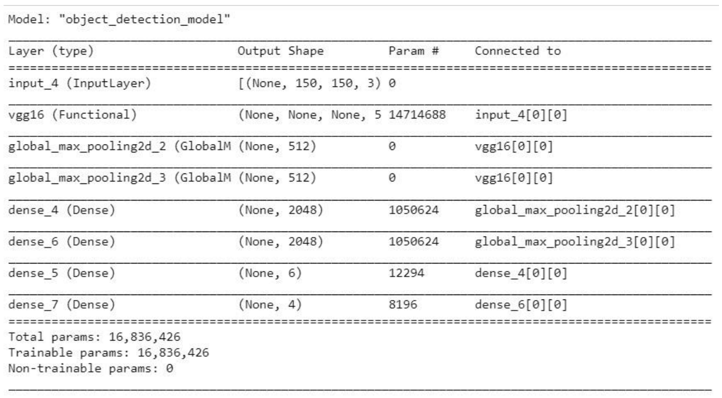
3.5.4 Model Development

As opposed to the scene classification model developed using the Sequential API, the object detection model was developed using the Keras Functional API. This is because the Functional API allows for more flexibility in designing and training complex deep learning models. Unlike the Sequential API which only supports a linear stack of layers, the Functional API can handle models with non-linear topology, shared layers, and even multiple inputs or outputs which is precisely what was required for the object detection task. The model to be developed required one input layer for the images and two outputs: one for object classification and one for object bounding box regression.

Transfer learning was leveraged once again for the completing the task with less data and computational expense. The VGG16 model [69] was used as the backbone for the object detection model and is in fact a common practice in the field as VGG16 has shown to be effective in feature extraction for a various number of computer vision tasks since its release. VGG16 is a deep convolutional neural network architecture developed by the Visual Geometry Group at the University of Oxford and is known for its simplicity and its ability to achieve high accuracy on image recognition tasks. The VGG16 architecture was trained on the ImageNet dataset and consists of 13 convolutional layers and 3 fully connected layers, that obtained a classification accuracy of 92.7 percent in the 2014 ImageNet Classification Challenge [65] which is why it is regarded as one of the best vision model architectures available. By fine-tuning the VGG16 base, the pre-trained model's knowledge of features could be taken advantage of to help train it to recognize and localize objects in the constructed dataset.

The model takes in images of size 150 x 150 pixels as input and uses the VGG16 as the base while specifying *include_top=False* to remove the final layers of VGG16 so the two prediction heads of the object detection model can be branched out from that layer. The two output branches of the model have a *GlobalMaxPooling()* layer which takes the output of the VGG16 base and reduces it to a 1D tensor. The tensor is then passed to a Dense (fully connected) layer with 2048 neurons and ReLU activation function. The classification head of the model has a final Dense layer with 8 output neurons corresponding to the 8 object classes in the dataset, and the bounding box regression head of the model has a Dense layer with 4 output neurons corresponding to the 4 bounding box coordinates of the prediction. Finally, the *Model()* function is used to combine the inputs and outputs layers into a single model.

A detailed overview of the model can be obtained as a summary of the created model shown in Figure 3.17 below.



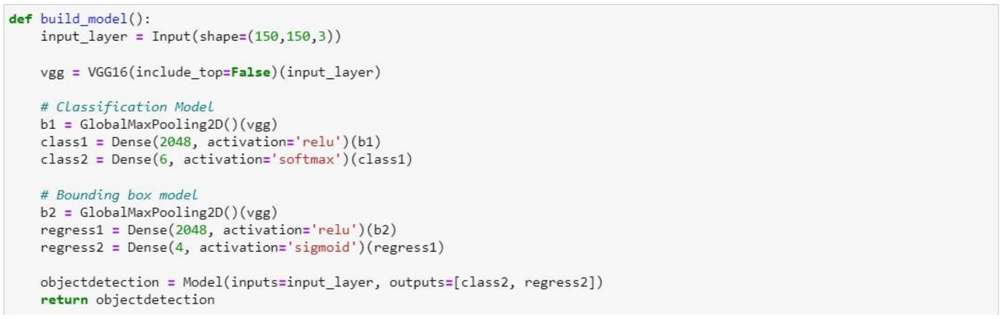
```
Model: "object_detection_model"
__________________________________________________________________________________________________
Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
input_4 (InputLayer)           [(None, 150, 150, 3) 0

__________________________________________________________________________________________________
vgg16 (Functional)             (None, None, None, 5 14714688    input_4[0][0]
__________________________________________________________________________________________________
global_max_pooling2d_2 (GlobalM (None, 512)         0           vgg16[0][0]
__________________________________________________________________________________________________
global_max_pooling2d_3 (GlobalM (None, 512)         0           vgg16[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                (None, 2048)         1050624     global_max_pooling2d_2[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                (None, 2048)         1050624     global_max_pooling2d_3[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                (None, 6)            12294       dense_4[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                (None, 4)            8196        dense_6[0][0]
==================================================================================================
Total params: 16,836,426
Trainable params: 16,836,426
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 3.17: Summary of Object Detection model

The choice of activation functions used in the final layers of the classification and bounding box heads is based on the nature of the tasks they are designed to perform.

The softmax activation function is commonly used in multi-class classification tasks as is seen from how it was also used as the activation function in the output of the scene classification model. In this case, the model is trained to predict the probability distribution over the 8 mutually exclusive classes. It ensures that the sum of probabilities over all classes is equal to one which ensures the model assigns a high probability to the correct class and low probabilities to the other classes.

On the other hand, the task to predict the four coordinates of the bounding box coordinates which are continuous values is a regression problem. The sigmoid function maps these values to the range of [0, 1]. The use of the rectified linear unit (ReLU) activation function in the penultimate layers of both branches is also a common choice in deep learning. ReLU is a function that helps the neural network to learn better by preventing the gradients from becoming too small and allowing it to capture more complex patterns in the data. It is a simple and efficient activation function that has proved to work well in a wide range of applications. The structure of the model is built in the *build_model()* function depicted in Figure 3.18.



```
def build_model():
    input_layer = Input(shape=(150,150,3))

    vgg = VGG16(include_top=False)(input_layer)

    # Classification Model
    b1 = GlobalMaxPooling2D()(vgg)
    class1 = Dense(2048, activation='relu')(b1)
    class2 = Dense(6, activation='softmax')(class1)

    # Bounding box model
    b2 = GlobalMaxPooling2D()(vgg)
    regress1 = Dense(2048, activation='relu')(b2)
    regress2 = Dense(4, activation='sigmoid')(regress1)

    objectdetection = Model(inputs=input_layer, outputs=[class2, regress2])
    return objectdetection
```

Figure 3.18: Code snippet showing the creation of the model architecture

The TensorFlow documentation states that "A core principle of Keras is progressive disclosure of complexity. You should be able to gain more control over the small details while retaining a commensurate amount of high-level convenience." Accordingly, the Keras Model class provides the functionality to do just this by being able to customize what happens in the *model.fit()* function [70]. The *fit()* function is what is used to train the model for the required number of epochs. Hence being able to override the training process at lower level is a distinguishing feature.

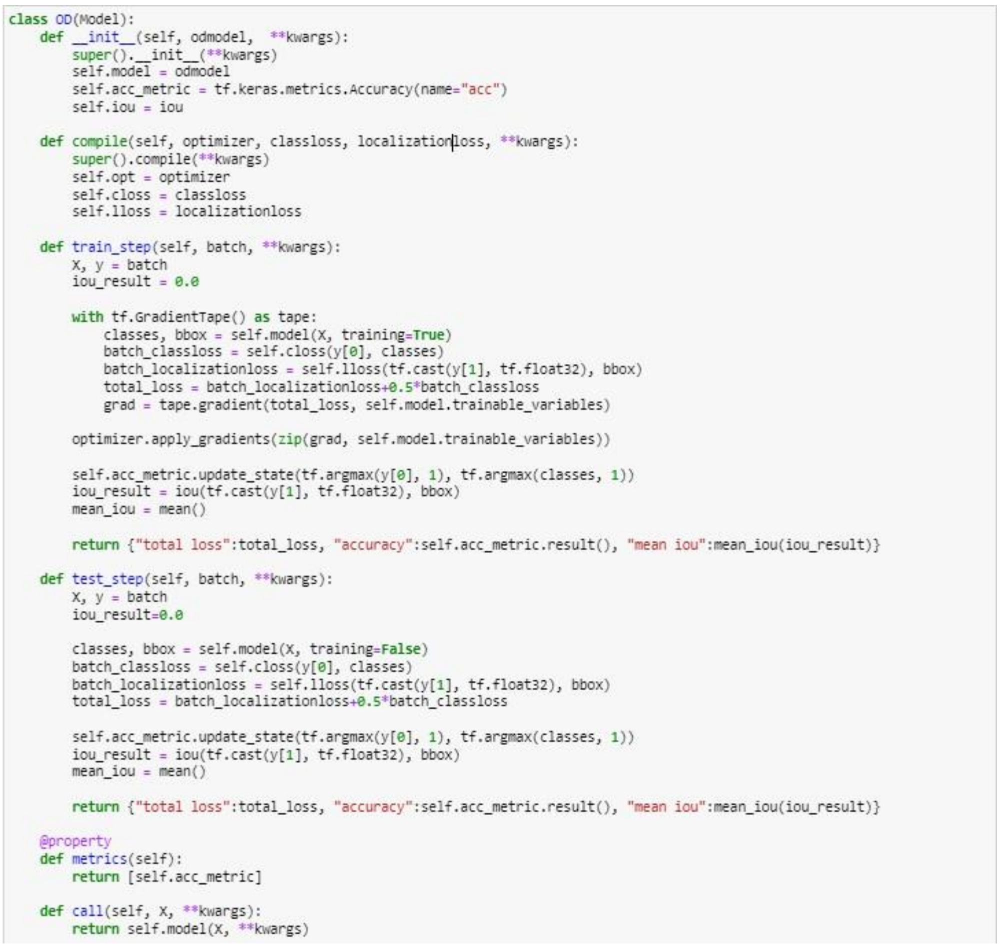
Subclassing the Model class from Keras [71] requires 6 methods to be explicitly defined namely *__init__(), compile(), train_step(), test_step(), metrics(),* and *call().* A custom class

(Figure 3.19) that is inherited from the Model class takes the object detection model instance as input and implements the following functions that are explained below.

- **__init__:** this is where the initial parameters are passed through and in this case, it is the pre-built neural network instance created using the *build_model* function and the metrics that the model track while training. The model is defined to track and calculate two metrics namely accuracy and the mean IoU over bbox predictions. The accuracy of the model is calculated using the *tf.keras.metrics.Accuracy()* function whereas for the mean IoU, a custom function [72] had to be defined.
- **compile:** this is where the optimizer and the loss functions to be used are passed through. In this case there are two loss function for the two output heads of the model. The classification output uses the *tf.keras.losses.CategoricalCrossentropy()* loss function which is the most suitable loss function for one-hot encoded multi-class labels, and the bbox output uses the *tf.keras.losses.Huber()* loss function which is essentially the same as Smooth L1 loss, the most common loss used for bounding box regression problems. Huber loss is a combination of the mean squared error (MSE) loss and the mean absolute error (MAE) loss. It is less sensitive to outliers than the MSE loss and more robust than the MAE loss, making it a good choice for bounding box regression, where the predicted values may contain some noise. The optimizer used is the *tf.keras.optimizers.Adam()* optimizer with a low learning rate of 0.0001 since fine-tuning of a pre-trained model is required. This is necessary because the pre-trained model already has good initial weights, and only small adjustments to the weights are needed rather than completely changing them. A larger learning rate may cause the model to overfit or diverge from the optimal weights.

- **train_step:** this is the function that actually trains the neural network batch by batch. It takes a batch of training data along with the true labels and then the model makes predictions on images in the batch to return the predicted values of the class labels and bbox coordinates. The respective loss functions are applied to the true and predicted values of both the class label and bbox coordinates and the total loss for that step is calculated as a weighted linear addition of the two losses. After this the gradients are calculated with respect to the loss function and applied using the optimizer to minimize the total loss. Another key step taking place is the state of the metric values being updated after each prediction. Finally, the loss and average metrics values are returned which can be used to monitor the training process.
- **test_step:** it is triggered when the validation or test set is passed through to the model and is almost identical to train_step. The only difference being that there are no gradient values being calculated and hence no backpropagation is taking place.
- **metrics:** it is used to reset the state of the metrics after each epoch. Otherwise, the returned metrics values would be an average since the start of training, whereas per-epoch averages are preferred as they give a better sense of the gradual improvement in the quality of predictions made by the model.
- **call:** it is used to make predictions using the trained model.

A class instance is created using the model object returned by *build_model(),* this instance was then used for training and testing the object detection model.



```python
class OD(Model):
    def __init__(self, odmodel,  **kwargs):
        super().__init__(**kwargs)
        self.model = odmodel
        self.acc_metric = tf.keras.metrics.Accuracy(name="acc")
        self.iou = iou

    def compile(self, optimizer, classloss, localizationloss, **kwargs):
        super().compile(**kwargs)
        self.opt = optimizer
        self.closs = classloss
        self.lloss = localizationloss

    def train_step(self, batch, **kwargs):
        X, y = batch
        iou_result = 0.0

        with tf.GradientTape() as tape:
            classes, bbox = self.model(X, training=True)
            batch_classloss = self.closs(y[0], classes)
            batch_localizationloss = self.lloss(tf.cast(y[1], tf.float32), bbox)
            total_loss = batch_localizationloss+0.5*batch_classloss
            grad = tape.gradient(total_loss, self.model.trainable_variables)

        optimizer.apply_gradients(zip(grad, self.model.trainable_variables))

        self.acc_metric.update_state(tf.argmax(y[0], 1), tf.argmax(classes, 1))
        iou_result = iou(tf.cast(y[1], tf.float32), bbox)
        mean_iou = mean()

        return {"total loss":total_loss, "accuracy":self.acc_metric.result(), "mean iou":mean_iou(iou_result)}

    def test_step(self, batch, **kwargs):
        X, y = batch
        iou_result=0.0

        classes, bbox = self.model(X, training=False)
        batch_classloss = self.closs(y[0], classes)
        batch_localizationloss = self.lloss(tf.cast(y[1], tf.float32), bbox)
        total_loss = batch_localizationloss+0.5*batch_classloss

        self.acc_metric.update_state(tf.argmax(y[0], 1), tf.argmax(classes, 1))
        iou_result = iou(tf.cast(y[1], tf.float32), bbox)
        mean_iou = mean()

        return {"total loss":total_loss, "accuracy":self.acc_metric.result(), "mean iou":mean_iou(iou_result)}

    @property
    def metrics(self):
        return [self.acc_metric]

    def call(self, X, **kwargs):
        return self.model(X, **kwargs)
```

Figure 3.19: Subclassed Keras *Model()* class for the Object Detection process

The model was now ready to compile and train. The EarlyStopping callback was used to track the validation accuracy with a patience of 5 epochs since it was observed with trial and error that the mean IoU continued to improve despite the validation accuracy having plateaued. However, it was important to choose a number that was not too high in order to prevent overfitting of the model while trying to obtain the best performance.

3.6 Context-Aware Object Detection Models Implementation

This section details the implementation of the two context-aware object detection models and explains the modifications made to the dataset and models from the previous section to attempt to implement the capability of scene comprehension. The performance of the models are then evaluated with respect to each other on the same dataset to understand the efficacy of the experiment. The models use two different approaches to incorporate context-awareness by adjusting the loss functions of the object detection models based on scene predictions; Model 1 is trained on scene classification on top of object detection in the usual manner using its own predictions, while Model 2 used the predictions from the supplementary scene classification model for scene classification.

The rationale of this experiment is that by altering the loss function values, there is a change in the gradients that are computed during backpropagation. And since the gradients computed during backpropagation are used to update the weights of the model during training, if a term is included in the loss function that penalizes the model for making incorrect predictions about image scene information, the model can take into account the scene prediction error and adjust the weights of the model to improve scene interpretation accuracy. By adjusting the loss function in this way, the model is encouraged to learn useful features for both image context/scene comprehension and object detection thus incorporating contextual information from the scene to theoretically result in a more robust and accurate model overall.

3.6.1 Context-Aware Model 1

The first implementation of the context aware model was structured similarly to the standard object detection model in all respects except that it had 3 outputs instead of 2. In addition to the class label and bbox coordinates outputs, this model had a scene label output as well. Its architecture is given below in Figure 3.20

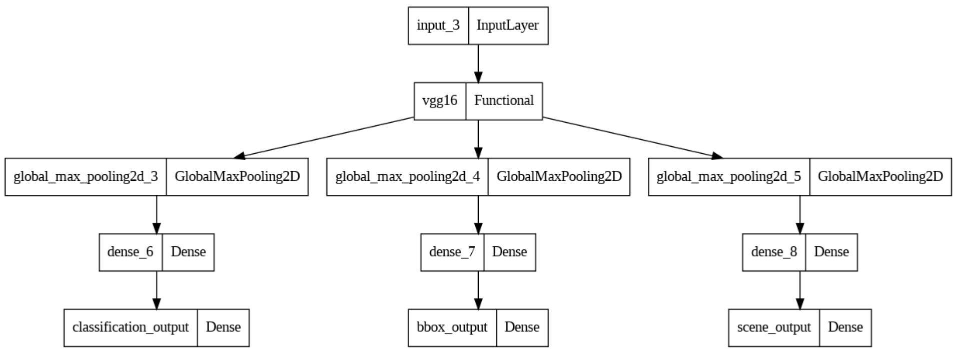
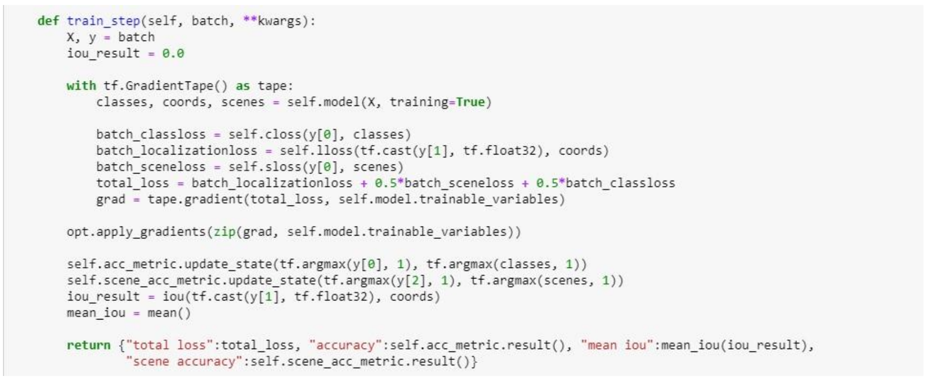

input_3 | InputLayer
vgg16 | Functional
global_max_pooling2d_3 | GlobalMaxPooling2D
global_max_pooling2d_4 | GlobalMaxPooling2D
global_max_pooling2d_5 | GlobalMaxPooling2D
dense_6 | Dense
dense_7 | Dense
dense_8 | Dense
classification_output | Dense
bbox_output | Dense
scene_output | Dense

Figure 3.20: Architecture of Context-Aware Model 1

This was done in order to allow the model to train itself to make accurate predictions on the scene of an image with the aim of learning extra information that could be useful for classifying objects. In this way the model could learn not only object classification and localization but also object-scene relationships which would help it make more accurate predictions. The model is initialized with an additional metric for the scene classification accuracy as well as an additional loss function for the scene predictions. The *tf.keras.losses.CategoricalCrossentropy()* loss is used to calculate the loss between the one-hot encoded true labels and the models predictions of the scene. This 'scene_loss' value is then used as the final constituent of the 'total_loss' value that is calculated by a weighted addition of all the 3 loss functions (class and scene classification loss, bbox regression loss) which is used to calculate the gradients and perform gradient descent. Following, the object classification accuracy, bbox IoU, and scene classification accuracy values are updated and returned while training. This process is then repeated for each batch until the training is complete. The modified train_step() function for Context-Aware Model 1 is described in Figure 3.21



```
def train_step(self, batch, **kwargs):
    X, y = batch
    iou_result = 0.0

    with tf.GradientTape() as tape:
        classes, coords, scenes = self.model(X, training=True)

        batch_classloss = self.closs(y[0], classes)
        batch_localizationloss = self.lloss(tf.cast(y[1], tf.float32), coords)
        batch_sceneloss = self.sloss(y[0], scenes)
        total_loss = batch_localizationloss + 0.5*batch_sceneloss + 0.5*batch_classloss
        grad = tape.gradient(total_loss, self.model.trainable_variables)

    opt.apply_gradients(zip(grad, self.model.trainable_variables))

    self.acc_metric.update_state(tf.argmax(y[0], 1), tf.argmax(classes, 1))
    self.scene_acc_metric.update_state(tf.argmax(y[2], 1), tf.argmax(scenes, 1))
    iou_result = iou(tf.cast(y[1], tf.float32), coords)
    mean_iou = mean()

    return {"total loss":total_loss, "accuracy":self.acc_metric.result(), "mean iou":mean_iou(iou_result),
            "scene accuracy":self.scene_acc_metric.result()}
```

Figure 3.21: Code snippet showing the modified training algorithm for Model 1

3.6.2 Context-Aware Model 2

The second context aware object detection model also uses the same architecture as the standard object detection model. However, as opposed to context aware model 1 which uses the base model to train itself on scene classification, this model uses the scene classification model (discussed in Section 3.4, obtained an accuracy of 93.5%) to make predictions on image scenes. The architecture of the model (which is unchanged from the standard object detection model) is given in Figure 3.22.

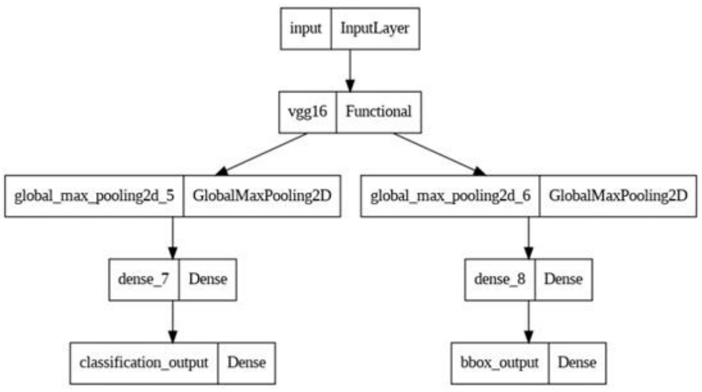
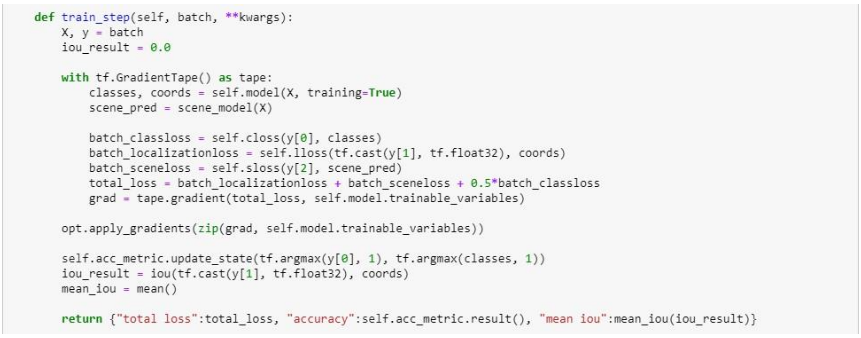

input | InputLayer
vgg16 | Functional
global_max_pooling2d_5 | GlobalMaxPooling2D
global_max_pooling2d_6 | GlobalMaxPooling2D
dense_7 | Dense
dense_8 | Dense
classification_output | Dense
bbox_output | Dense

Figure 3.22: Architecture of Context-Aware Model 2

The scene classification model was developed in a separate Jupyter Notebook as Model 2 but it was possible to save its configuration, weights and optimizer's state using the *tf.keras.saving.save_model()* function, which saves the implemented model as a HDF5 file. This file can be saved to a location on the local machine and loaded up inside any different environment using the *tf.keras.saving.load_model()* function. It can then be used to make predictions inside the *train_step()* on batches of images as shown in Figure 3.23. The model was also compiled with additional parameter called scene_loss to calculate the *tf.keras.losses.CategoricalCrossentropy()* loss between the true one-hot encoded labels of the scene classes and the floating point predictions made by the scene classification model.

The model works in the same way as the object detection model expect that while training, for each batch of images the model makes predictions for the object class, bbox and scene labels which are passed to the respective loss functions. However, the scene classification model is what is used to predict the scene of the image, and not the base model itself. As a result, since the additional scene information is already available to the model, there is no need for it to learn it anew. After this the loss values and gradients are calculated, gradient descent is performed, and the metrics are updated and returned.



```python
def train_step(self, batch, **kwargs):
    X, y = batch
    iou_result = 0.0

    with tf.GradientTape() as tape:
        classes, coords = self.model(X, training=True)
        scene_pred = scene_model(X)

        batch_classloss = self.closs(y[0], classes)
        batch_localizationloss = self.lloss(tf.cast(y[1], tf.float32), coords)
        batch_sceneloss = self.sloss(y[2], scene_pred)
        total_loss = batch_localizationloss + batch_sceneloss + 0.5*batch_classloss
        grad = tape.gradient(total_loss, self.model.trainable_variables)

    opt.apply_gradients(zip(grad, self.model.trainable_variables))

    self.acc_metric.update_state(tf.argmax(y[0], 1), tf.argmax(classes, 1))
    iou_result = iou(tf.cast(y[1], tf.float32), coords)
    mean_iou = mean()

    return {"total loss":total_loss, "accuracy":self.acc_metric.result(), "mean iou":mean_iou(iou_result)}
```

Figure 3.23: Code snippet showing the modified training algorithm for Model 2

Chapter 4

Evaluation and Results Analysis

In this chapter, the performance of the object detection models is examined, and a comparative analysis of the results are discussed. The performance of the models is evaluated using the evaluation strategies introduced in Appendix B.

After obtaining the models that performed best during fitting, each model makes predictions on unseen images in the test set of 704 images and the output obtained from the function *predict()* is used for the performance evaluation of the model. The confusion matrix, accuracy, and all other metrics are calculated using this output.

4.1 Context-Aware Model 1 Results

Model 1 was trained for 7 epochs which took 63 minutes and obtained **an accuracy of 71.31%** and a **mean IoU of 0.66**. Figure 4.1 displays the model's loss, accuracy and mean IoU performance trends with the best performance over the training period. Figure 4.2 shows the confusion matrix and Table 3 gives the classification metrics report. An additional metric defined for Model 1 was 'scene accuracy' which calculated the accuracy of scene predictions made by the model. It obtained a score of 85.22% for this metric.

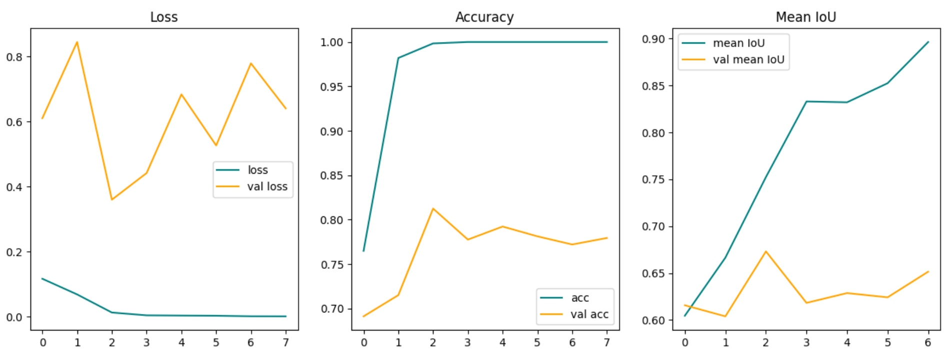
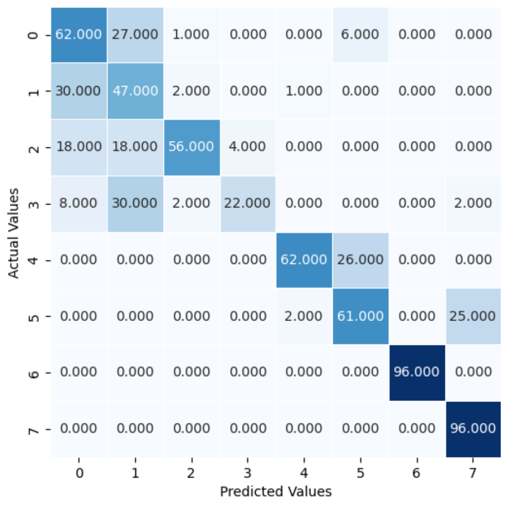


Figure 4.1: Loss and Metrics performance trends during training of Model 1



	0	1	2	3	4	5	6	7
0	62.000	27.000	1.000	0.000	0.000	6.000	0.000	0.000
1	30.000	47.000	2.000	0.000	1.000	0.000	0.000	0.000
2	18.000	18.000	56.000	4.000	0.000	0.000	0.000	0.000
3	8.000	30.000	2.000	22.000	0.000	0.000	0.000	2.000
4	0.000	0.000	0.000	0.000	62.000	26.000	0.000	0.000
5	0.000	0.000	0.000	0.000	2.000	61.000	0.000	25.000
6	0.000	0.000	0.000	0.000	0.000	0.000	96.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	96.000

Class	**Precision**	**Recall**	**F1-score**	**Support**
0	0.53	0.65	0.58	96
1	0.39	0.59	0.47	80
2	0.92	0.58	0.71	96
3	0.85	0.34	0.49	64
4	0.95	0.70	0.81	88
5	0.66	0.69	0.67	88
6	1.00	1.00	1.00	96
7	0.78	1.00	0.88	96
Accuracy			**0.71**	704
Macro avg.	**0.76**	**0.69**	**0.72**	704

Figure 4.2: Confusion Matrix for Model 1

Table 3: Classification Report for Model 1

Class and bounding box predications made by Model 1 are displayed below.

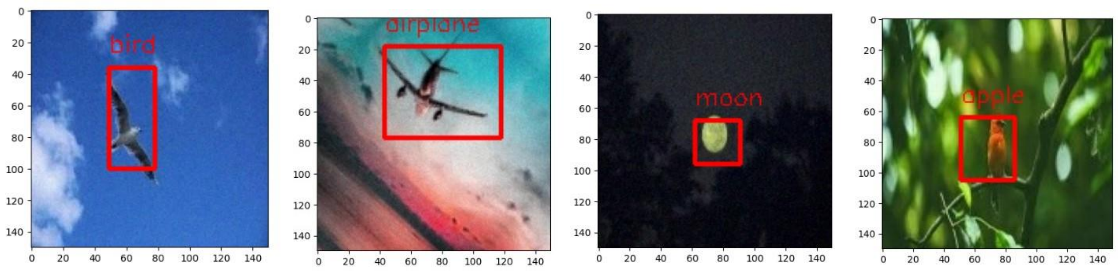


Figure 4.3: Class and bounding box predictions made by Model 1

4.2 Context-Aware Model 2 Results

Model 2 was also trained for 7 epochs taking 67 minutes and obtained an **accuracy of 75.14%** and a **mean IoU of 0.69**. Figure 4.4 displays the model's loss, accuracy and mean IoU performance trends with the best performance over the training period. Figure 4.5 shows the confusion matrix and Table 4 gives the classification metrics report.

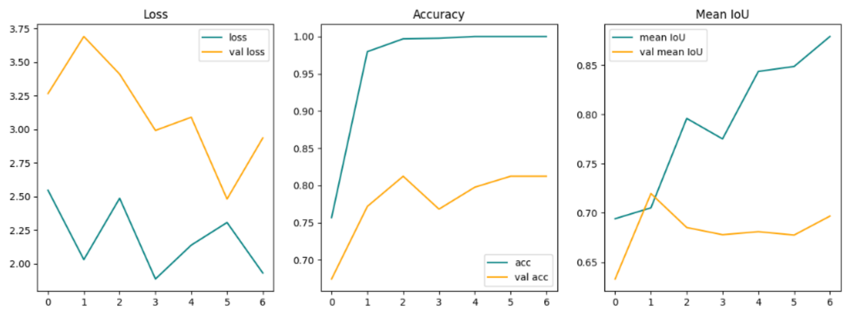


Figure 4.4: Loss and Metrics performance trends during training of Model 2

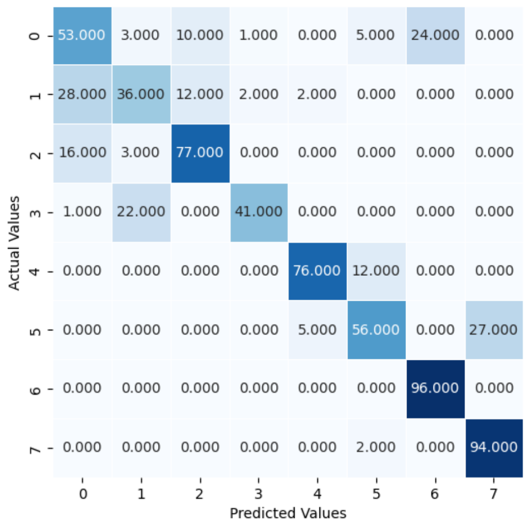


Figure 4.5: Confusion Matrix for Model 2

Class	Precision	Recall	F1-score	Support
0	0.54	0.55	0.55	96
1	0.56	0.45	0.50	80
2	0.78	0.80	0.79	96
3	0.93	0.64	0.76	64
4	0.92	0.86	0.89	88
5	0.75	0.64	0.69	88
6	0.80	1.00	0.89	96
7	0.78	0.98	0.87	96
Accuracy			**0.75**	704
Macro avg.	**0.76**	**0.74**	**0.75**	704

Table 4: Classification Report for Model 2

Class and bounding box predictions made by Model 2 are displayed below.

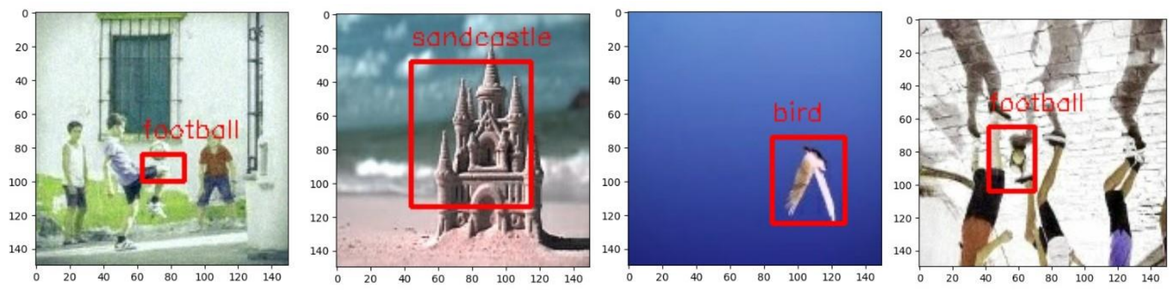


Figure 4.6: Class and bounding box predictions made by Model 2

4.3 Comparative Analysis of Object Detection Models

A comparative study of the implemented models is performed in this section to understand which variant of the context-aware object detection model performs the best. The performance of the standard object detection model without any context-aware capabilities is described as well in order to gain a comprehensive understanding of how well the context-aware modifications work and how beneficial or counterproductive they are. Table 5 below denotes the values of each of the metrics for all models.

Models	**Accuracy**	**Mean IoU**	**Precision**	**Recall**	**F1-score**
Context-Aware Model 1	71.30%	0.66	0.76	0.69	0.72
Context-Aware Model 2	**75.14%**	**0.69**	0.76	**0.74**	**0.75**
Standard Object Detection Model	73.43%	0.67	**0.78**	0.72	**0.75**

Table 5: Comparison of metrics obtained by all models

From the table it can be observed that Context-Aware model 2 has the best performance in most of the metrics with the highest accuracy of 75.14% and highest mean IoU of 0.69. It outperformed the standard object detection model by a slight margin regarding the accuracy and mean IoU, but the standard object detection model obtained a better precision and the same F1-score. However, Context-Aware model 2 performed worse than both the other models in all respects and thus performed the worst overall.

4.4 Critical Analysis of Results

Both models performed decently for the object classification tasks although the IoU metrics suggest that there is room for improvement in the object localization aspect. The loss trends for both models are irregular which could be attributed to the large variation in images in the relatively small dataset. However, the loss trend for Model 2 was ultimately falling which is ideal, as opposed to the loss trend of Model 1 which is rather erratic.

Model 1 obtained an accuracy and mean IoU lower than both Model 2 and the standard object detection model. This result demonstrated that the method used to incorporate contextual information in Model 1 by training the model to learn scene information in addition to object classification and localization actually worsened its performance. Potential reasons for this could be that the scene classification task led to a more complex optimization problem due to the additional noise in the training process, which may have caused the model to struggle to perform multiple tasks leading to subpar predictions. It is also possible that the scene information was not useful in this specific approach and information provided it may be immaterial.

However, Model 2 obtained an accuracy improvement of 5.39% and a mean IoU increase of 4.55% from Model 1 and outperformed the standard object detection model as well. This result was rather interesting since the technique used in Model 2 had the opposite effect from Model 1 wherein the context-awareness hindered its performance. It showed that the approach used to actually integrate the contextual information is very significant in determining the performance of a model. The use of a pre-trained external classification model to make intelligent scene predictions proved to be more constructive than training the base model itself for multiple tasks simultaneously. The two models had the same precision, but Model 2 had a greater recall, which meant that it correctly identified a larger proportion of true positives which is desirable since it is better at identifying positive instances and has a lower rate of false negatives. Furthermore, the evaluation of Model 2 compared to the standard object detection model proved that the appropriate implementation of context-aware functionality can in fact be effective in increasing the performance of object detection models.

Chapter 5

Conclusion

The aim of this project was to develop a context-aware object detection model. The process involved constructing a suitable dataset, incorporating contextual information in the training process in different ways, and evaluating the developed models. With this approach, one of the implemented context-aware models obtained an **accuracy of 75.14%** and **mean IoU of 0.69** on the constructed dataset which are the highest values obtained for those metrics, making it the best performing model overall. The positive outcome of this experiment led to the conclusion that contextual knowledge from images is a valuable source of information that can be used to improve object detection.

This chapter concludes the report by reflecting on the challenges faced during the timeline of this project, its limitations, future works to pursue, validation of project requirements, and finally a reflection of thoughts and experience working on this year-long project.

5.1 Challenges Faced during Implementation

The main difficulty faced during the implementation phase of this dissertation was the requirement to process large volumes of image data, which is computationally intensive. Unfortunately, the available hardware was not equipped with the required specifications to perform efficiently during the training phase. Consequently, training the CNN models was time-consuming, which slowed down the development process due to this absence of GPU acceleration capabilities. The training process was exceptionally slow, taking over 10 minutes for every epoch, which made hyperparameter tuning a tedious trial-and-error process. Although Google Colab was considered as an option for an IDE, its limitations in runtime usage required discarding it as an option. Despite the challenges faced during the training phase, all the high-priority functional requirements outlined by the requirements analysis (Appendix A) were met.

Another challenge to overcome was the limited ground up support for object detection tasks using CNNs from Keras (for example the unavailability of a built-in IoU metric for bbox localization) and minimal information for developing any context-aware implementations found online.

5.2 Limitations & Future Work

The first part of this section briefly talks about the limitations in the project implementation. The second part of this section suggests ideas for future work to further improve the system.

5.2.1 Limitations

The main limitation of each object detection models implementation is that the models were developed and trained to detect only one object per image which meant that if an image contained multiple objects of interest, the model could only identify one of them. The reason for this limitation is that since the object detection models used in this project were developed from scratch, they had to be designed and trained specifically for the task at hand. Implementing multi-label functionality would require significant additional effort and complexity beyond the scope of the allotted timeframe.

5.2.2 Future Work

Testing different CNN architectures and base networks for transfer learning would be critical in determining the best network for optimal performance. Hence, additional research could be conducted on the benefits and drawbacks of specific architectures for object detection.
The implementation of multi-object classification and localization functionality is necessary to ensure the developed model is of a good standard comparable with the state-of-the-art object detection models of today.

Further research into the various sources of contextual information in images and the appropriate methods to implement context-awareness is vital to make sure the best techniques are used to bring out the highest caliber of results.

5.3 Validation of Requirements

To verify that all the requirements presented in Appendix A are completed, Table 6 shows the requirements and their completion status.

Functional Requirements			
ID	**Description**	**Priority**	**Status**
FR1	Find (or construct) a suitable dataset to train the model to detect objects by making use of context information	M	Done
FR2	Assess previous machine learning and deep learning frameworks and libraries to make the appropriate choices for the implementation of the model	M	Done
FR3	Perform proper pre-processing and feature extraction of the dataset for training	M	Done
FR4	Suitably split the dataset into training and testing sets	M	Done

FR5	Choose a suitable object detection algorithm that makes use of contextual information	M	Done
FR6	Train and test the implemented model	M	Done
FR7	The model must be able to detect objects and place the corresponding bounding boxes over them	M	Done
FR8	The model must be able to classify detected objects with great accuracy	M	Done
FR9	Perform an evaluation of the performance and accuracy of the model	M	Done
FR10	Compare and contrast the performance of the model with other state of the art object detection models	C	Done
FR11	The model could be implemented as a mobile application to work in real time	W	Not done

Table 6: Validation of requirements analysis

5.4 Reflection

This project provided a great opportunity to gain more insight into the world of computer vision and explore the nuances of object detection. The obstacles faced throughout the course of this project were an insightful teacher that provided the opportunity to hone my skills and forced me to think creatively and problem-solve in new ways. Through trial and error, experimentation, and hard work, I was able to overcome all challenges and produce models that succeeded in achieving the aim of this project. I have developed new skills and gained a deeper understanding of the field, and I am excited to apply these insights to future projects and challenges.

References

[1] P. Ongsulee, "Artificial intelligence, machine learning and deep learning," *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*, 2017, pp. 1, doi: 10.1109/ICTKE.2017.8259629.

[2] R.S. Michalski, J.G. Carbonell, T.M. Mitchell, *Machine Learning: An Artificial Intelligence Approach.* USA: Springer-Verlag, 1983, pp. 3. Accessed: Oct. 16, 2022. [Online]. Available: https://doi.org/10.1007/978-3-662-12405-5

[3] tensorflow.org https://www.tensorflow.org/resources/models-datasets

[4] keras.io https://keras.io/api/

[5] opencv.org https://docs.opencv.org/4.x/

[6] pypi.org https://github.com/wkentaro/labelme

[7] I. Mihajlovic, *"Everything You Ever Wanted to Know About Computer Vision."* towardsdatascience.com. https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e (accessed Oct. 19, 2022).

[8] T.J. Thomson, D. Angus and P. Dootson, *"3.2 billion images and 720,000 hours of video are shared online daily. Can you sort real from fake?"* theconverstaion.com. https://theconversation.com/3-2-billion-images-and-720-000-hours-of-video-are-shared-online-daily-can-you-sort-real-from-fake-148630 (accessed Oct. 19, 2022).

[9] S.B. Maind and P. Wankar, "Research Paper on Basic of Artificial Neural Network," *International Journal IJRITCC,* vol. 2, pp. 96-97, Jan. 2014. Accessed: Oct. 19, 2022. [Online]. Available: https://ijritcc.org/index.php/ijritcc

[10] K. O’Shea and R. Nash, “An Introduction to Convolutional Neural Networks”, pp. 4-8, Dec. 2015. Accessed: Oct. 20, 2022. doi: 1511.08458v2 [Online] Available: https://arxiv.org/pdf/1511.08458.pdf

[11] D. Bhatt et al., “CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope,” *Electronics*, vol. 10, no. 20, p. 2470, Oct. 2021. Accessed: Oct. 20, 2022. doi: 10.3390/electronics10202470. [Online] Available: https://www.mdpi.com/2079-9292/10/20/2470/htm#B1-electronics-10-02470

[12] Z. Zhao, P. Zheng, S. Xu and X. Wu, “Object Detection with Deep Learning: A Review*,” IEEE transactions on neural networks and learning systems,* pp. 3212-3232, Apr. 2019. Accessed: Oct. 20, 2022. doi: 1807.05511. [Online] Available: https://arxiv.org/pdf/1807.05511.pdf&usg=ALkJrhhpApwNJOmg83O8p2Ua76PNh6tR8A

[13] S.R. Prakash and P.N. Singh, “Object detection through region proposal based techniques,” *Materials Today: Proceedings,* vol. 46, no. 9, pp. 3997-4002, Feb. 2021. Accessed: Oct. 20, 2022. doi: https://doi.org/10.1016/j.matpr.2021.02.533. [Online] Available: https://www.sciencedirect.com/science/article/pii/S2214785321016746#

[14] P.F. Felzenszwalb and D.P. Huttenlocher, “Efficient Graph-Based Image Segmentation,” *International Journal of Computer Vision,* vol. 59, pp. 167-181, Sep. 2004. Accessed: Oct. 20, 2022. doi: https://doi.org/10.1023/B:VISI.0000022288.19776.77. [Online] Available: https://link.springer.com/article/10.1023/B:VISI.0000022288.19776.77#citeas

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 580–587, Nov. 2013. Accessed: Oct. 21, 2022. doi: 1311.2524v5 [Online]. Available: https://arxiv.org/abs/1311.2524v5

[16] N. Yadav and U. Binay, “Comparative Study of Object Detection Algorithms,” *International Research Journal of Engineering and Technology (IRJET),* vol. 4, no. 11, pp. 586-591, Nov. 2017. Accessed: Oct. 21, 2022. [Online]. Available: https://academia.edu

[17] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," *2017 International Conference on Engineering and Technology (ICET)*, pp. 1-6, 2017, Accessed: Oct. 21, 2022. doi: 10.1109/ICEngTechnol.2017.8308186. [Online]. Available: https://ieeexplore.ieee.org/document/8308186

[18] R. Girshick, “Fast r-cnn,” *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440-1448, Accessed: Oct. 21, 2022. [Online]. Available: https://openaccess.thecvf.com/content_iccv_2015/papers/Girshick_Fast_R-CNN_ICCV_2015_paper.pdf

[19] S. ren, K. He, R. Girshick and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* pp. 1136-1149, Jan. 2017. Accessed: Oct. 21, 2022. doi: https://doi.org/10.48550/arXiv.1506.01497 [Online] Available: https://arxiv.org/pdf/1506.01497.pdf

[20] www.cocodataset.org https://cocodataset.org/#home

[21] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2980-2988, Oct. 2017. Accessed: Nov. 24, 2022. doi: 10.1109/ICCV.2017.322. [Online] Available: https://ieeexplore.ieee.org/document/8237584

[22] J. Dai, Y. Li, K. He and J. Sun, “R-FCN: Object Detection via Region-based Fully Convolutional Networks,” *Advances in neural information processing systems,* vol. 29, Jun. 2016. Accessed: Oct. 23, 2022. doi: https://doi.org/10.48550/arXiv.1605.06409. [Online] Available: https://proceedings.neurips.cc/paper/2016/hash/577ef1154f3240ad5b9b413aa7346a1e-Abstract.html

[23] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, Dec. 2015. Accessed: Oct. 23, 2022. doi: https://doi.org/10.1007/978-3-319-46448-0_2 [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-46448-0_2

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December, pp. 779–788, Jun. 2015. Accessed: Oct. 23, 2022. doi: https://doi.org/10.48550/arXiv.1506.02640 [Online]. Available: https://arxiv.org/abs/1506.02640v5

[25] A.K. Dey, “Understanding and Using Context,” *Personal and Ubiquitous Computing*, vol. 5, pp. 4-7, Feb. 2001. Accessed: Oct. 25, 2022. doi: https://doi.org/10.1007/s007790170019 [Online]. Available: https://link.springer.com/article/10.1007/s007790170019

[26] H. Hock, G. P. Gordon, and R. Whitehurst. “Contextual relations: The influence of familiarity, physical plausibility, and belongingness,” *Perception & Psychophysics*, vol. 16, pp. 4-8, Feb. 1974. Accessed: Oct. 25, 2022. doi: https://doi.org/10.3758/BF03203242 [Online]. Available: https://link.springer.com/article/10.3758/BF03203242

[27] S.K. Divvala, D. Hoiem, J.H. Hays, A.A Efros and M. Hebert “An empirical study of context in object detection,” *IEEE Conference on computer vision and Pattern Recognition*, pp. 1271-1278, Jun. 2009. Accessed: Oct. 25, 2022. doi: 10.1109/CVPR.2009.5206532. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5206532

[28] P.F. Felzenszwalb, R.B. Girshick, D. McAllester and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 32, no. 9, pp. 1627-1645, Sept. 2010. Accessed: Oct. 25, 2022. doi: 10.1109/TPAMI.2009.167. [Online]. Available: https://ieeexplore.ieee.org/document/5255236

[29] L. V. Pato, R. Negrinho and P. M. Q. Aguiar, "Seeing without Looking: Contextual Rescoring of Object Detections for AP Maximization," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 14598-14606, Mar. 2020. Accessed: Nov. 25, 2022. doi: 10.1109/CVPR42600.2020.01462. [Online]. Available: https://arxiv.org/pdf/1912.12290.pdf

[30] R. Gandhi, "*Support Vector Machine — Introduction to Machine Learning Algorithms.*" https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (accessed Nov. 17, 2022)

[31] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, A.W.M, Smeulders, "Selective Search for Object Recognition." *Int J Comput Vis,* vol. 104, pp. 154–171, Apr. 2013. Accessed: Nov. 17, 2022. Available: https://doi.org/10.1007/s11263-013-0620-5

[32] N. Murali, "Image Classification vs Semantic Segmentation vs Instance Segmentation" medium.com. https://nirmalamurali.medium.com/image-classification-vs-semantic-segmentation-vs-instance-segmentation-625c33a08d50 (accessed Nov. 24, 2022)

[33] M. Naseem, S. Reda, "AdaCon: Adaptive Context-Aware Object Detection for Resource-Constrained Embedded Devices*." 2021 IEEE/ACM International Conference on Computer-Aided Design,* Aug. 2021. Accessed: Nov 20, 2022. Available: https://arxiv.org/pdf/2108.06850.pdf

[34] H. Xu, C. Jiang, X. Liang, Z. Li, "Spatial-aware Graph Relation Network for Large-scale Object Detection." *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* Jun. 2019, Accessed: Nov 20, 2022. doi: 10.1109/CVPR.2019.00952. Available: https://ieeexplore.ieee.org/document/8954369

[35] J. Peng, H. Wang, S. Yue, Z. Zhang, "Context-aware co-supervision for accurate object detection," *Pattern Recognition*, vol. 121, Jan. 2022, Accessed: Nov. 20,2022. Available: https://www.sciencedirect.com/science/article/pii/S0031320321003812

[36] J. Li et al., "Attentive Contexts for Object Detection," *IEEE Transactions on Multimedia*, vol. 19, no. 5, pp. 944-954, May 2017, Accessed: Nov. 20, 2022. doi: 10.1109/TMM.2016.2642789. Available: https://ieeexplore.ieee.org/document/7792742

[37] R. Mottaghi et al., "The Role of Context for Object Detection and Semantic Segmentation in the Wild," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 891-898, Jun. 2014. Accessed: Nov. 20, 2022. doi: 10.1109/CVPR.2014.119. Available: https://ieeexplore.ieee.org/document/6909514

[38] albumentation.ai https://albumentations.ai/docs/

[39] “Introduction to Convolutional Neural Networks” theclickreader.com. https://www.theclickreader.com/introduction-to-convolutional-neural-networks/

[40] “Two stage vs one stage object detection models” researchgate.net. https://www.researchgate.net/figure/Two-stage-vs-one-stage-object-detection-models_fig3_353284602

[41] K. Mittal, “A Gentle Introduction Into The Histogram Of Oriented Gradients.” medium.com. https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa

[42] “Selective Search for Object Detection | R-CNN” geeksforgeeks.org. https://www.geeksforgeeks.org/selective-search-for-object-detection-r-cnn/

[43] A. Mohan, “A Review On Fast RCNN.” medium.com https://medium.datadriveninvestor.com/review-on-fast-rcnn-202c9eadd23b

[44] A. Makone, “Faster RCNN.” medium.com. https://ashutoshmakone.medium.com/faster-rcnn-502e4a2e1ec6

[45] L. Liu, "Deep Learning for Generic Object Detection: A Survey." researchgate.net. https://www.researchgate.net/figure/Taxonomy-of-challenges-in-generic-object-detection_fig6_327550187

[46] E. Forson, "Understanding SSD MultiBox — Real-Time Object Detection In Deep Learning." towardsdatascience.com. https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

[47] M. Chablani," YOLO — You only look once, real time object detection explained." towardsdatascience.com. https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006

[48] "Confusion matrix – an overview" sciencedirect.com. https://www.sciencedirect.com/topics/engineering/confusion-matrix

[49] P. Soviany and R. T. Ionescu, "Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction," *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 209-214, Aug. 2018. Accessed: Nov. 25, 2022. doi: 10.1109/SYNASC.2018.00041. Available: https://arxiv.org/pdf/1803.08707.pdf

[50] www.researchgate.net https://www.researchgate.net/figure/Illustration-of-intersection-over-union-IOU_fig5_346512249

[51] T. Grel, "Region of interest pooling explained." deepsense.ai https://deepsense.ai/region-of-interest-pooling-explained/ (accessed Nov. 24, 2022)

[52] Object Detection on COCO test-dev paperswithcode.com https://paperswithcode.com/sota/object-detection-on-coco

[53] W. Wang et al. "InternImage: Exploring Large-Scale Vision Foundation Models with Deformable Convolutions" Nov. 2022. Accessed: Nov. 29, 2022. Available: https://arxiv.org/pdf/2211.05778v2.pdf

[54] Real-Time Object Detection on COCO paperswithcode.com https://paperswithcode.com/sota/real-time-object-detection-on-coco

[55] C. Wang, A. Bochkovskiy, H.M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors" Jul. 2022. Accessed: Nov. 29, 2022. Available: https://arxiv.org/pdf/2207.02696v1.pdf

[56] J. Hui, "Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)" medium.com https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359 (accessed Nov. 29, 2022)

[57] S. Paul, P. Chen, "Vision Transformers are Robust Learners" May 2021. Accessed: Nov. 29, 2022. Available: https://arxiv.org/abs/2105.07581

[58] G. Boesch, "Vision Transformers (ViT) in Image Recognition" viso.ai https://viso.ai/deep-learning/vision-transformer-vit/ (accessed Nov. 29, 2022)

[59] tensorflow.org https://www.tensorflow.org/hub/tutorials/object_detection

[60] tensorflow.org https://www.tensorflow.org/guide/keras/functional

[61] H. Sun, Z. Meng, P. Y. Tao and M. H. Ang, "Scene Recognition and Object Detection in a Unified Convolutional Neural Network on a Mobile Manipulator," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Sep. 2018, pp. 5875-5881. Accessed: Feb. 15, 2023. doi: 10.1109/ICRA.2018.8460535.

[62] chrome.google.com https://chrome.google.com/webstore/detail/download-all-images/ifipmflagepipjokmbdecpmjbibjnakm?hl=en

[63] tensforflow.org https://www.tensorflow.org/api_docs/python/tf/keras/Sequential?version=nightly

[64] “Understanding Transfer Learning for Deep Learning,” analyticsvidhya.com https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/

[65] image-net.org https://www.image-net.org/challenges/LSVRC/2014/index.php

[66] keras.io https://keras.io/api/callbacks/early_stopping/

[67] tensorflow.org https://www.tensorflow.org/api_docs/python/tf/data/Dataset

[68] keras.io https://keras.io/api/applications/resnet/#resnet50-function

[69] keras.io https://keras.io/api/applications/vgg/#vgg16-function

[70] tensorflow.org https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit

[71] tensorflow.org https://www.tensorflow.org/api_docs/python/tf/keras/Model?version=nightly

[72] github.com https://github.com/Balupurohit23/IOU-for-bounding-box-regression-in-Keras/blob/master/iou_metric.py

[73] scikit-learn.org https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

[74] numpy.org https://numpy.org/doc/stable/

[75] github.com https://github.com/nachi-hebbar/Transfer-Learning-ResNet-Keras

[76] github.com https://github.com/nicknochnack/FaceDetection

Appendices

Appendix A Requirements Analysis

The content of this chapter outlines the various functional and non-functional requirements in line with the objectives of the project previously stated. It also contains detailed use case scenarios along with suitable descriptions. The user requirements are prioritized using the MoSCoW analysis method given below:

- **Must (M):** These are mandatory requirements to achieve the goal of the project.
- **Should (S):** These requirements are important as they add great merit to the project but do not affect main functionality.
- **Could (C):** These requirements are not necessary to the project but if added would improve certain features and can be implemented depending based on time availability.
- **Wont (W):** These requirements are optional and are most likely not to be implemented.

A.1 Functional Requirements

Functional Requirements		
ID	**Description**	**Priority**
FR1	Find a suitable dataset to train the model to detect objects by making use of context information	M
FR2	Assess previous machine learning and deep learning frameworks and libraries to make the appropriate choices for the implementation of the model	M
FR3	Perform proper pre-processing and feature extraction of the dataset for training	M

FR4	Suitably split the dataset into training and testing sets	M
FR5	Choose a suitable object detection algorithm that makes use of contextual information	M
FR6	Train and test the implemented model	M
FR7	The model must be able to detect objects and place the corresponding bounding boxes over them	M
FR8	The model must be able to classify detected objects with great accuracy	M
FR9	Perform an evaluation of the performance and accuracy of the model	M
FR10	Compare and contrast the performance of the model with other state of the art object detection models	C
FR11	The model could be implemented as a mobile application to work in real time	W

A.2 Non-Functional Requirements

Non-functional requirements		
ID	**Description**	**Priority**
NFR1	Ensure selected dataset complies with GDPR	M
NFR2	The code is well structured, documented and commented	M
NFR3	The code is open-source and backed up with version control	M
NFR4	The code is updated regularly to facilitate work with new upcoming hardware	S

Appendix B Evaluation Strategy

This chapter describes the evaluation strategies that are used to assess the performance of the proposed model. The performance metrics that determine and quantify the efficiency of the model is discussed below. A general structure that is used to help assess the performance of an object detection model such as the proposed model for this project is a confusion matrix, that can be used to measure performance of any classification algorithm. An illustration is given below in Figure 4.1:

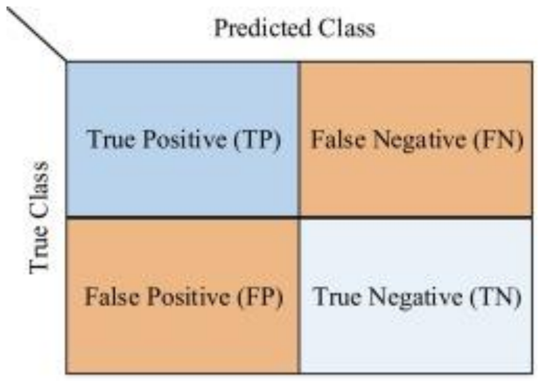


Figure B.1: Confusion Matrix

The two aspects of object detection namely, object classification and object localization can both be evaluated based on the following metrics:

- **Intersection over Union (IoU):** defined as the ratio between the area of overlap between the ground truth (annotated bounding box in training dataset) and the predicted truth (bounding box output of the model) and the area of union between them. The IoU vizualization is given in Figure 4.2.

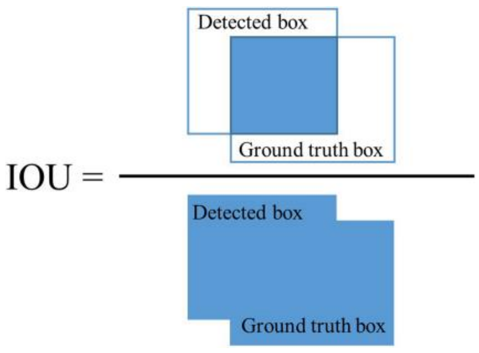


Figure B.2: IoU visualization [50]

- **Average Precision (AP):** precision is calculated as

$$Precision = \frac{True\ Postives}{True\ Positives + False\ Positives}$$

the ratio of positive predictions to total positive instances is called recall and it is calculated as

$$Recall = \frac{True\ Postives}{True\ Positives + False\ Negatives}$$

The average precision is defined as the area under the graph made by the precision vs recall line curve.

- **F1-score:** a weighted average or harmonic mean of precision and recall values. Since it considers both recall and precision, it is considered as an apt metric for the evaluation of a model's performance.

$$F1\ score = \frac{2 * Recall * Precision}{Recall + Precision}$$

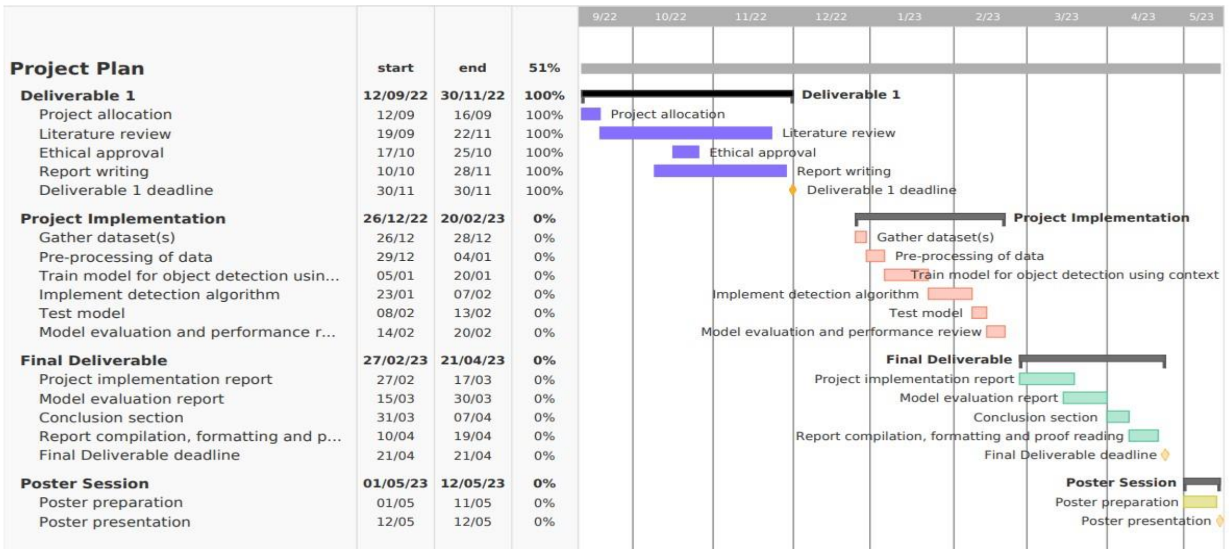
- **Accuracy:** accuracy of the classifications can be calculated as

$$Accuracy = \frac{True\ Postives + True\ Negatvies}{Total\ Population}$$

Appendix C Project Management

C.1 Project Plan

The timetable for the development of the project is described using the following Gantt chart in Figure 5.1. It specifies the project plan for the whole year and includes 4 sections: Deliverable 1, Project implementation, Final deliverable, and Poster session. Each section specifies the associated activities, deliverable and deadline for that section.



Project Plan	start	end	51%
Deliverable 1	**12/09/22**	**30/11/22**	**100%**
Project allocation	12/09	16/09	100%
Literature review	19/09	22/11	100%
Ethical approval	17/10	25/10	100%
Report writing	10/10	28/11	100%
Deliverable 1 deadline	30/11	30/11	100%
Project Implementation	**26/12/22**	**20/02/23**	**0%**
Gather dataset(s)	26/12	28/12	0%
Pre-processing of data	29/12	04/01	0%
Train model for object detection usin...	05/01	20/01	0%
Implement detection algorithm	23/01	07/02	0%
Test model	08/02	13/02	0%
Model evaluation and performance r...	14/02	20/02	0%
Final Deliverable	**27/02/23**	**21/04/23**	**0%**
Project implementation report	27/02	17/03	0%
Model evaluation report	15/03	30/03	0%
Conclusion section	31/03	07/04	0%
Report compilation, formatting and p...	10/04	19/04	0%
Final Deliverable deadline	21/04	21/04	0%
Poster Session	**01/05/23**	**12/05/23**	**0%**
Poster preparation	01/05	11/05	0%
Poster presentation	12/05	12/05	0%

Figure C.1: Gantt chart for project plan

C.2 Risk Analysis

Early identification and analysis of risks associated with the fulfilment of project requirements are key to stay on track with the project plan. Potential risks to the development of the project and their appropriate mitigation strategies are described in the table below.

Risk Analysis			
Risk description	**Likelihood**	**Impact**	**Mitigation Strategy**
Insufficient or incomplete data in the dataset to train and test model	Low	High	Maintain one or a few backup datasets to change if necessary and pre-process data properly.
Health complications of author	Medium	Medium	Take necessary health precautions and reduce time spent on work. Revise project plan to make up for lost time during recovery
Corruption or loss of code/ report drafts	Medium	High	Always store backups on the cloud and save completed work frequently. Use version control software when developing code
Hardware malfunction	Low	Medium	Use backups to work on a different device. Make use of keyserver.
Unable to meet deadlines	Medium	High	Plan out work schedule in advance and follow project plan.

C.3 Professional, Ethical, Legal and Social Issues

Any research paper, articles, websites, or figures used in this project have been properly cited. Further, any code, libraries, and datasets that may be used for the development of this project are appropriately referenced at the time of its use. The implementation of the project has used only open-source resources that comply with GDPR rules and regulations. This project involves only the use of publicly available datasets that are frequently used in the field of computer vision. It does not involve the use or collection of any personal or confidential data as it does not make use of human subjects. This is in accordance with the ethical approval granted to this project by the Ethics Coordinator for the project.