In-Flight Bird Detection, Species Detection and Counting

By

Abdullah Ikram Ullah Tabassam

A thesis submitted in partial fulfilment of
the requirements of Liverpool John Moores
University for the degree of Master in
Artificial Intelligence (Machine Learning)

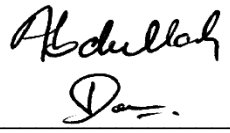
August 2023

Declaration

I hereby declare that this thesis titled "In-Flight Bird Detection, Species Detection and Counting" is my original work and no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning. All sources used in this thesis, including references and citations, have been duly acknowledged.

I further declare that any work previously submitted by me or done in collaboration with others for any academic or professional qualification has been properly acknowledged in this thesis.

The total word count of this thesis, including all chapters and sections, is approximately Eighteen thousand, two hundred and sixty eight (18,268) words.

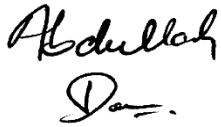
Signature: 

Abdullah Ikram Ullah Tabassam

Date: 31 August 2023

Acknowledgments

As I write this acknowledgments section, I am truly thankful and humbled to acknowledge the amazing support and encouragement I got during the process of completing my thesis. This accomplishment would not have been achieved without the unflinching support of various individuals and institutions that played key roles in developing this effort. First and foremost, I want to convey my heartfelt thanks to Dr Paul Fergus, my outstanding project supervisor, whose advice, knowledge, and mentorship have been vital in directing this study in the right direction. Their patience, valuable insights, and constructive feedback have served as the foundation for my thesis. I am grateful to the faculty members of Liverpool John Moores University for their dedicated contributions to my academic advancement through their teachings and inspiring speeches. Their commitment to developing young brains has been a source of inspiration and learning for me. I would want to express my profound gratitude to my friends and colleagues, whose support and encouragement have been a source of strength during this journey. Their engaging discussions and eagerness to assist have made this experience more rewarding. I would like to acknowledge LJMU for the essential resources and facilities that have helped me collect relevant data and perform the research. My heartfelt thanks go to my parents and family for their never-ending love, encouragement, and faith in my skills. Their unwavering encouragement has been the driving factor behind my efforts, making this achievement a joint victory. Finally, I extend my heartfelt thanks to the countless unnamed individuals who, in one way or another, have provided their support and encouragement throughout this journey. Thank you all for being an integral part of this remarkable journey.



Abdullah Ikram Ullah Tabassam

Abstract

This thesis investigates the use of object detection models for bird species categorization, which is an important issue in wildlife conservation and research. The primary goal is to create efficient models capable of reliably identifying and categorising various bird species from photos. The study employs two types of models: a specific bird species classifier that detects six different bird species and a generic bird classifier that classifies all birds under one class - Birds. The research includes data preprocessing, model training, evaluation, and inference. A large collection of images of various bird species is gathered and divided into training and testing sets. A pre-trained model called YOLOv8 is fine-tuned to identify the target successfully via transfer learning. For best model performance, hyperparameter tunning and optimizer selection are critical. The research compares models trained with various hyperparameters and optimizers, utilising assessment criteria such as precision, recall, and F1-score. Model performance is better understood by using visualisations such as confusion matrices and precision-recall curves. Results show that for specific species model, the model with the Adam optimizer and tunned hyperparameters performed the best overall, with excellent accuracy and recall values. Inference on unseen images further validates the model's ability to recognise and classify bird species. Similarly, for the generic birds classifier model, the auto optimiser with tunned hyperparameters performed the best. Using the PR curve helped choosing proper confidence levels ensuring that the required recall and precision levels are met. The object-detection models developed have major practical implications for conservation and wildlife monitoring activities. Researchers may collect data at scale by automating bird species identification, contributing to a better knowledge of bird populations and migration patterns.

[Page intentionally left blank]

Table of Contents

Table of Figures

List of Tables

List Of Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
AP	Average Precision
AR	Average Recall
CNN	Convolutional Neural Network
CSV	Comma Separated Values
DCNN	Deep Convolutional Neural Network
DNN	Deep Neural Network
DRCNN	Deep Residual Convolutional Neural Network
ESP-32	Wi-Fi Module
FAPG	Fast Average Peer Group
FCNN	Fully Connected Neural Networks
FPS	Frames Per Second
GPU	Graphical Processing Unit
HOG	Histogram Of Oriented Gradients
IEEE	Institute Of Electrical And Electronics Engineers
IOU	Intersection Over Union
ML	Machine Learning
MS-COCO	Microsoft Common Objects In Context
PIR	Passive Infrared
P-R	Precision Vs. Recall
RCNN	Region-Based Convolutional Neural Networks

ROC	Receiver Operating Characteristic
ROI	Region Of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machine
TP	True Positives
TSO	Threshold-Based Segmentation Operator
UAV	Unmanned Aerial Vehicle
US	United States
USA	United States Of America
VGG	Visual Geometry Group
YAML	Yet Another Markup Language
YOLO	You Only Look Once

Chapter 1

Introduction

1. Introduction:

Birds are awe-inspiring creatures that have long attracted people with their beauty, elegance, and distinct behaviours [1]. They are critical to the balance of our planet's ecosystems, offering crucial services including as pollination, seed dissemination, and pest control [1]. However, bird populations have been dropping at an alarming rate in recent years, owing to habitat degradation [2], climate change [3], and other causes [4]. According to North American Bird Conservation Initiative research, about one-third of all bird species in North America are suffering major population losses, with some species seeing up to 80% decreases [5].

Several reasons can be attributed for the fall in bird populations. Human-caused habitat loss and degradation, such as deforestation and urbanisation, have a considerable influence on bird populations [6]. Climate change, which includes changes in temperature and precipitation patterns, is also having an impact on bird populations by changing breeding and migrating patterns [7].

Understanding bird behaviour and mobility is essential for conservation efforts. Researchers have employed modern technologies such as artificial intelligence (AI) and machine learning (ML) to examine bird behaviour and movement patterns [8][9], revealing insights into their migratory routes and behaviour. AI-assisted in-flight bird detection and counting has enabled quick and precise monitoring of bird populations across broad regions.

1.1. Motivation:

The motivation behind the " In-Flight Bird Detection, Species Detection and Counting" project stems from the need for more efficient, cost-effective, and accurate monitoring methods for bird conservation. By harnessing the capabilities of AI and ML, we can overcome the limitations of traditional monitoring approaches. This project has the potential to revolutionize bird population studies, providing valuable insights that will guide conservation efforts and ultimately contribute to the preservation of these essential species and the ecosystems they inhabit. This project aims to address the need for efficient and accurate monitoring of bird populations. This motivation text will elaborate on the significance of this project and the reasons why it is essential to employ AI and ML techniques for bird conservation.

Birds have long served as indicators of environmental health due to their sensitivity to ecological changes [9][11][12]. Their presence, behaviour, and movements can provide valuable insights into the overall well-being of ecosystems [13]. By closely monitoring birds, we can gain a deeper understanding of how climate change, habitat loss, and other environmental factors impact their populations [14]. This knowledge is crucial for implementing effective conservation strategies that safeguard not only the bird species but also the entire ecosystem they inhabit.

Unfortunately, traditional methods of monitoring birds have proven to be both time-consuming and expensive [15][16]. Relying solely on human observers to record bird behaviour and count populations introduces limitations in terms of accuracy and efficiency. Human observers may not be able to cover large areas comprehensively, and their observations can be subject to bias and human error [17]. Consequently, there is a pressing need for innovative approaches that can overcome these challenges and provide more reliable data for bird conservation efforts.

In-Flight Bird Detection, Species Detection and Counting utilizing AI and ML technologies offers promising solutions to the aforementioned challenges [18][19]. By harnessing the power of artificial intelligence and machine learning algorithms, we can automate the process of detecting and identifying bird species in flight [20]. These algorithms can be trained to recognize specific bird species based on their unique characteristics such as flight patterns, wing shape, sounds, and coloration. This automation eliminates the need for extensive manual labour, reducing costs and significantly improving the efficiency of data collection [21][22][23].

Additionally, AI and ML techniques can be employed to accurately estimate bird populations. By analysing large datasets collected through in-flight bird detection, we can obtain reliable population counts in real-time [24]. This information is vital for assessing the status of bird populations, identifying any declines or fluctuations, and implementing timely conservation measures. Furthermore, AI algorithms can help identify and map migratory paths, allowing us to understand the routes and stopover locations crucial for birds during their long-distance journeys.

Behavioural analysis of birds is another aspect that can be greatly enhanced through AI and ML. These technologies enable us to study bird behaviour patterns and understand their responses to environmental changes more comprehensively. By analysing large volumes

of data, we can detect shifts in feeding patterns, breeding behaviours, or alterations in migration routes. Such insights are crucial for identifying the impact of human activities, climate change, or habitat loss on bird populations and informing targeted conservation strategies.

1.2. Importance:

Understanding the behaviour and movements of birds is crucial for several reasons. First, it can help us identify and protect critical habitats for these species. Birds rely on specific habitats for feeding, breeding, and nesting. By studying their behaviour and movements, researchers can gain insights into their habitat preferences and identify areas that are vital for their survival [25]. For example, research undertaken by [26] is critical in demonstrating the influence of habitat on bird species. The goal of this study was to look at how different environments differ in terms of species 'richness' and 'evenness'. They investigated whether habitat types and composition predict high bird diversity. According to the findings, forests reduced avian populations. Natural and farmland-related environments enhanced the number of bird species. Similarly, natural environment improved species diversity. The influence of forests on diversity and uniformity of species could not be replaced by urban greenspace.

Second, it can help us identify and mitigate potential threats to bird populations, such as collisions with wind turbines or other man-made structures. Bird crashes with structures can cause damage or death, especially during migration. A lot of birds, annually, either lose their habitats, or worse, their lives owing to the human need for energy, especially in the form of wind energy. It is difficult to estimate the number of casualties reported each year because most internet sources are not updated on a regular basis but quoting an estimate by the 'American Birds Conservancy', approximately 1,170,000 birds die each year in the US alone because of the collisions with wind turbine blades [27].

Finally, another reason to monitor the behaviour and movements of birds is to help us understand the impacts of climate change on bird populations, including changes in migratory patterns and breeding behaviours. Climate change may cause changes in various environmental variables leading to a significant impact on bird habitats and resources. The study in [28] looked at how migrating species reacted to climate change. Over time, information on reproductive dates and trends in populations were examined by the researchers. The study discovered that when spring phenology advanced owing to climate

change, there was a mismatch between the accessibility of food supplies and the timing of breeding, affecting reproductive success.

1.3. Current approaches:

Monitoring bird populations is an essential component of avian research and conservation efforts. Various approaches both traditional and non-traditional, have been employed to track and study bird populations, each with its own advantages and limitations. When discussing the traditional methods, the top techniques that are used are, visual surveys, acoustic monitoring, and radar.

Human observers count and identify birds in a specified region during visual surveys. This method is frequently used in point counts or transect surveys, in which observers document bird species and quantity at predefined points. Visual surveys give important information on bird activity and distribution, allowing experts to develop population estimates, evaluate habitat preferences, and track changes over time [17].

Acoustic monitoring entails recording and analysing bird vocalisations to learn about their existence, behaviour, and population dynamics. This approach uses specialised audio recording equipment to collect bird noises at particular time intervals. Acoustic data may be used to identify distinct species, evaluate diversity of species, and predict population densities [29].

Using radio waves, generated by radars, detection and quantification of bird migration cab be performed by measuring their altitudes. The radar technology, either weather radars or special bird radars, can identify and track bird flight patterns, velocity, direction, and a number of other parameters as a function of the calculated altitude. This method is very effective for analysing bird movements and migratory patterns, as well as estimating collision hazards near airports and wind farms [30].

The research carried out by a team of NVidia experts in on the topic of migratory route recognition is quite admirable which extended the radar technique to a next level. MistNet [31], a deep convolutional neural network (DNN), was developed to recognise precipitation from biological material in radar imagery. MistNet, unlike other machine learning approaches, can collect biological data from radar pictures containing precipitation and provide predictions on a finer scale. MistNet is created with neural networks for images and a variety of architecture components designed particularly for interacting with radar data. To avoid a lengthy human labelling effort, we train MistNet employing an extensive number

of noisy labels obtained using data from a dual polarisation radar. The researchers trained the deep learning model on a cluster of four GPUs and gave highly accurate results (97%).

1.4. Limitations:

Current approaches to monitoring bird populations have several limitations. Although visual surveys are commonly utilised, they do have significant disadvantages. They can be time-consuming, especially in vast research regions, and rely significantly on observer competence and training. Human error and observer bias can cause problems in identifying and counting species. Furthermore, certain bird species might be difficult to spot physically due to camouflage, isolated habitat sites, or nocturnal behaviour. In addition to this, differentiating between species merely based on vocalisations can be difficult, especially when dealing with complicated environments with overlapping vocalisations. Although automated software and machine learning approaches have been created to aid in species identification, their accuracy varies depending on the bird community and environmental variables. The use of acoustic monitoring in conjunction with other techniques for collecting data, such as visual surveys, can improve the accuracy and reliability of bird population estimations. Same is the case for radar monitoring, as it is costly and usually necessitates the use of specialised equipment and expertise. Interpreting radar data can be difficult since it involves separating birds from other aerial objects such as bugs, bats, or debris. Radar is best useful for large-scale investigations and tracking bird movements across huge geographic areas. Adding the concept discussed in MistNet [31] to radar detection technique, no doubt, makes it accurate but on the same time increases the computational complexity and the costs associated them.

1.5. Aims and Objectives:

The objective of this thesis is to create an AI and machine learning-based system for in-flight bird detection, species detection, and counting. Our goals are as follows:

- Create an artificial intelligence (AI) and machine learning (ML)-based system for detecting and counting birds in flight utilising photos and videos taken by unmanned aerial vehicles (UAVs).
- Create algorithms that identify and monitor individual birds in flight in order to establish migratory routes and behavioural patterns.
- Analyse the acquired data to identify essential habitats and potential threats to bird populations by analysing bird behaviour and movements.

- Compare the accuracy and efficacy of our solutions by training multiple models using different algorithms and AI technologies currently available for computer vision tasks.

In this thesis, we aim to address these issues by utilizing AI and ML techniques to analyse large datasets of bird behaviour and movement patterns. By understanding the factors that are contributing to their decline, we hope to develop targeted conservation strategies that can help protect bird populations and their habitats. Our approach builds on the advancements in technology and research in this field and has the potential to make significant contributions to bird conservation efforts.

Chapter 2

Background and Literature Review

2. Background and Literature Review:

The primary objective of this project revolves around the conservation endeavours dedicated to safeguarding wildlife, ecosystems, and biodiversity. As an essential part of the planet's tapestry of life, wildlife plays a crucial part in preserving the balance of ecological systems and fostering their resilience. In this chapter, we'll review the measures taken to preserve different animal species, their complex habitats, and, ultimately, the biodiversity of our planet. This academic analysis will include an insight into the approaches and strategies used prior to the development of computer vision techniques and machine learning, which assumed their crucial roles in the venerable endeavour of wildlife conservation. We will also explore cutting-edge techniques utilising deep learning for this cause, with a particular emphasis on the field of avian preservation. Finally, in order to further deepen our understanding of this vast and complex subject, we will explore the ethical, social, and political aspects connected to it.

2.1. Traditional methods:

Not long ago, the conservationists depended almost only on the traditional methods of data collection, and wildlife monitoring. Extensive research on techniques and methods used to count and preserve wildlife is provided in 'Ecological Census Techniques. A Handbook' [32]. The handbook gives details on collecting and understanding the samples using various techniques and discusses about the preservation of all kinds of living beings including plants, Invertebrates, Fish, Amphibians, Reptiles, Birds, and Mammals. One of the most common methods for animal conservation and counting, especially birds, include counting the nests. The techniques can vary depending on whether the nests are on some cliff, tree, plain ground, or burrows. In addition to nest counting the handbook also discusses other techniques like roost counting, flock counting, and migrant counting in detail. Another highly detailed manual on bird counting in the United Kingdom [33] is also available and mainly focuses on the seabirds in the Britain and Ireland regions. Although these details are highly insightful for our research but are based on field surveys and are highly dependent on human expertise and manual observation to detect and identify bird species. This requires enormous resources like trained manpower and time, still lacking behind in efficiency.

In addition to [32], [34] also discuss point counts. Point counts technique is used in a wide range of ecosystems and are especially well suited to densely forested areas for highly vocal or visible species. This technique requires the person counting the birds to be highly vigilant and stealthy as a sudden movement can scare off the birds and it is very easy to get

confused on whether a specific individual has been counted or not. All these techniques mentioned till now are performed by trained individuals and require manual tools like binocular to watch the birds, notebooks to keep the record of the counted birds, etc and are highly susceptible to human error.

2.2. Image Processing Techniques:

The impact of Paul Viola and Michael Jones' work on image processing undoubtedly demands recognition [35]. Their pioneering approach towards developing a framework for detecting faces called the Viola-Jones framework introduced in 2001 revolutionized facial feature identification methods used within images. The primary feature of this model is that it facilitated the quick and efficient detection of faces even on low computational devices through real-time video/image analysis. Its effectiveness specifically lies in how quickly it processes images by reducing background layers, thereby classifying minor characteristics resulting in precise outcomes.

To recognize faces in an image, this framework follows several progressive stages to filter out non-face areas. Initially, it scans the picture through varying sizes of sliding windows analysing various Haar-like features by utilizing sets of weak classifiers at every window position. With these weak classifiers coupled via boosting algorithms it eventually forms robust classifications enough to differentiate between faces and non-faces. Although the main purpose of this research was not strictly related to the wildlife conservation, it opened a lot of new doors towards new technologies, using image processing and edge computing, species detection and counting.

Navneet Dalal and Bill Triggs also revolutionized the field of image processing with their groundbreaking technique named "Histograms of Oriented Gradients" (HOG) [36]. Where many before them struggled with limitations surrounding detections in images; their ingenious idea changed everything significantly. The HOG approach proposed by these inventors recognises local gradient orientation when discerning crucial information from an image. Put simply, this equates to breaking down larger images into smaller cells with calculations performed on each to construct histograms that illustrate gradient orientations. Understanding gradient orientation distribution is critical for identifying unique patterns for detections in images. From there, the HOG algorithm can be utilized alongside these histograms to identify specific shapes and structures fundamental in complex visual feature detection.

Edge detection technique [37] is also an important topic to discuss for object detection in images. Its main purpose is to identify and highlight the boundaries or edges of objects within an image. It involves detecting significant changes in pixel intensity or colour values, which often correspond to object boundaries or transitions between different regions in an image. Detecting edges help extract the boundaries of the objects in the images which can be utilized for object segmentation and recognition. In addition to edge density, texture based analysis was also used to detect objects in images [38].

All these techniques discussed although were groundbreaking at the time of their introduction, required a lot of computational resources and power. These methods were also not much effective with images containing highly complex structures.

2.3. Deep Learning and Computer Vision:

The study and preservation of numerous species are now being approached by academics and conservationists in whole new ways thanks to the development of deep learning and computer vision as potent tools in the area of conservation. The use of these technologies has had a significant influence on many aspects of animal conservation, offering novel answers to age-old problems.

To detect wildlife from images and video frames, various machine learning techniques like supervised, semi-supervised, and unsupervised learning are used [39]. As claimed by [39] use of video data instead of images in domain of wildlife conservation is a new and emerging domain. The process discussed involves preprocessing using a FAPG (Fast Average Peer Group) filter after resizing the frames to some fixed size for ease. After the preprocessing, the images are fed to a DRCNN (Deep Residual Convolutional Neural Network) with the TSO (Threshold-based Segmentation Operator) algorithm to accurately classify different classes of wild animals using the Serengeti dataset and the evaluation is done on various performance metrics like F-1 Score, false alarm rate, false discovery rate, accuracy, precision, and sensitivity. In addition to this, [39] discusses the detection of wildlife from image data using various tree based classification algorithms like SVM, Random

Forests and Decision Trees. This technique is implemented on thermal images utilizing the HOG technique and Convolutional Neural Nets.

Animal accidents on highways is also a big reason for decline in their numbers. [41] discusses the same issue related to animal-vehicle interaction on roads. The paper mentions two models based on YOLOv4 for the animal detection on roads with high accuracy values around 80% and 85%. The models are trained using transfer learning utilizing the capabilities of already trained models on new datasets using images from different cameras and Lidar sensors to increase road safety for both humans and animals.

Jaswal et. al. in [42] used various region based networks like Fast and Faster RCNN to detect wildlife using custom dataset obtained from camera traps installed on various locations to collect short videos of animal to mitigate hunting, poaching and road accidents as well as saving human beings from predator attacks. Their methodology is based on IECG technique using Deep Convolutional Nets and integrating the model with a Raspberry Pi module to use it as an edge device.

A similar study for human safety from animal attacks is discussed in [43]. The author is of the view that object detection technique can be used to detect animals before they can attack humans to save their lives. In this approach they tend to use various feature like Local Binary Pattern, Statistical Moment Method, and Mean Amplitude Map to train a Deep Convolutional Neural Network. When tested on a dataset containing 1800 images of ten different classes of animals, the model gave an accuracy of 97%.

In order to monitor wildlife, particularly bird habitats and nesting places, deep learning models have shown to be incredibly successful in analysing satellite and aerial footage. These algorithms can precisely pinpoint crucial breeding grounds, roosting locations,

and nesting places by analysing enormous volumes of remote sensing data. This knowledge may be used by conservationists to carry out focused conservation initiatives and guarantee the preservation of vital habitats for vulnerable or endangered bird species.

To help ornithologists, researchers, and bird enthusiasts identify various bird species in a certain geographic region, researchers have developed an Arduino-based system for automated bird species recognition [44]. The system makes use of an ESP-32 camera, a PIR Motion Sensor, and an Arduino Uno. The camera records a picture when motion is detected, which is subsequently uploaded to Google Drive. A Kaggle dataset with 450 photos of different bird species (70,626 for training, 22,500 for testing, and 2,250 for validation) is used by trained deep learning models to analyse the photographs and predict the bird species name. Without depending on conventional bird books or manual identification techniques, this technology promises to increase user knowledge of diverse bird species and make birdwatching more accessible.

Studying birds offers important insights into nature and helps to preserve the balance of ecosystems. It can be difficult to distinguish between different bird species because of changes in form, appearance, and lighting. Rath et. al. [45] propose a method for precise bird categorization and species identification using deep learning. The classification procedure is not complete without the use of the unsupervised method. The computer learns to identify significant features in the bird image and ignore unimportant backgrounds. Large datasets must be divided into various categories as part of the photo categorization process, which presented difficulties in this study.

The researchers in [46] have used a similar approach to [44] for detecting birds species from image data sets. Using the Arduino and Keras deep learning frameworks, the "birdhouse" automated bird recognition system was created. An ESP-32 camera is activated by the system via an infrared sensor, and it takes pictures of birds for processing. The deep learning model, which uses MobileNetV2 framework and transfer learning, predicts bird species with a 95% test accuracy. Users receive the bird photograph and the determined species via the Telegram app. For devices with limited resources, such as ESP-32 cameras, MobileNetV2 [47] is perfect. Transfer learning offers fine-tuning for avian species recognition and training on a huge dataset. This technique assists with bird monitoring and conservation efforts to maintain ecological balance and biodiversity.

Given the scarcity of many bird species in the modern world, bird conservation and identifying them have emerged as vital issues. Effective bird classification is difficult because of the variety in sizes, colours, and angles that birds display, making hand recognition difficult [48]. With the introduction of deep convolutional neural networks (DCNN) used with the GoogLENet framework, a viable solution has arisen. The DCNN system successfully predicts bird species with a remarkable 88.33% accuracy by transforming bird photos into grayscale format and creating signatures [48]. The requirement for automated bird recognition and counting in aerial ocean photos has also been studied. Prior research frequently relied on intuitive custom features, but a new DCNN architecture changed the emphasis to systematic feature learning [49]. On the test set, the CNN-based classifier attained an astounding 95% accuracy [49], proving the efficacy of deep learning methods for bird species detection. With these developments, there is hope for better bird conservation and monitoring initiatives, which will help to protect biodiversity and preserve the delicate balance of our ecosystems.

The study [50] in describes a project that uses UAVs footage and deep convolutional neural networks for the automated detection and counting of birds along the coast of West Africa. Significant tern and gull breeding colonies, important bioindicators for the health of the aquatic environment, are located in the area. Traditional foot surveys are time-consuming, inaccurate, and harmful to birds. The research skilfully overcomes these constraints with the use of UAVs and CNNs, successfully localising several thousands of birds. The most common species, royal terns, are accurately predicted by the CNN model with high precision (90% precision at 90% recall), while the rarer Caspian tern and gull species, which make up around 7% of the population, are correctly predicted with fair precision (60% precision at 68% recall and 20% precision at 88% recall, respectively).

Chapter 3

Model Selection

3. Model Selection

As this project is based solely on the model's capabilities to detect the birds from UAV images, it is very important that we select the model that best suits our needs of accuracy and precision and gives the best performance in the real-world environment. The ability to identify and locate birds within aerial images and video frames makes object detection a crucial component of our avian monitoring research. A detailed analysis of various well-known models, including Faster R-CNN, YOLO (You Only Look Once), and SSD (Single Shot MultiBox Detector), is done to guarantee the best performance and accuracy.

Each model's advantages and disadvantages are systematically assessed as part of the evaluation process. Accurate detection, quick processing, flexibility to in-flight circumstances, and simplicity of implementation are important evaluation criteria. To assess if these models are appropriate for the goals of our project, their architectural components and optimisation techniques are also carefully analysed.

By carefully contrasting the aforementioned approaches, we aim to reach a well-informed judgment based on performance indicators and empirical facts. Our in-flight bird detection and counting system will be built on our chosen model, giving us dependable and effective avian monitoring. The criteria chosen for this approach are based on the coco metrics as all these models have been trained on the MS Coco dataset and these metrics give us all the information, we need to select the best suitable model for our project.

3.1. SSD – Single Shot MultiBox Detector

As the name suggests, it is a single-shot detector, which means that, unlike some other architectures that require multiple stages of processing, SSD [51] performs all the operations in one stage and is a very famous single-stage model. The main purpose of this model is to perform real-time detections, which makes it really suitable for our purpose as we require fast and accurate real-time, or at least near real-time detections in this project. It is also capable of detecting multiple objects in one image using the 6 additional layers after the VGG16 [52] backbone, but there are also a few drawbacks of this model. Although the SSD model is capable of creating multi-scale feature maps, which means that the model uses feature maps of various resolutions to detect objects, leading to multiscale object detection, ranging from

small to large, it does have a constraint on the annotation size during the training and requires pre-processing beforehand.

If we take a look at the numbers, the most common metrics that we can check for our purpose are the FPS and mAP values for the model. The mAP value will help us identify how precise the model can perform in real time, and FPS will be the measure of the speed of the model for real-time detection. These measures are taken from the focal loss paper [53] which is basically a comparison of various one-stage and two-stage object detection algorithms of its time and is considered as an authentic source. The paper shows the mAP of the SSD architecture at COCO training set to be 31.2% with the detection time to be at 125 ms, which comes out to be 8 FPS. This value looks a little less but on custom datasets it usually gives better performance. In [54] a comparison for various models is also provided by using a medical-based dataset containing pill images, the results that SSD show are promising standing at 32 FPS and a mAP value of 82.41%.

3.2. Faster RCNN

Faster RCNN is one of the state-of-art models for object detection and is considered a lot of times as a model of choice in cases where precise predictions are of more importance than the inference time. Unlike SSD, Faster RCNN [55] is a two-stage model, and this is the main reason that makes the predictions more accurate. The first stage consists of a region proposal network, RPN. The RPN recommends regions where the objectness is high, i.e., potential object-containing regions in the picture. The RPN is a fully connected CNN that predicts if an object is present within each window, as it moves a tiny window across the picture, or not. The results of these proposed regions can then be used by the ROI pooling architecture to extract fixed size features from these proposals, which then are fed to a classifier that makes predictions on the various classes. This classification mechanism is very sophisticated but takes a long time as compared to other models. In real-time applications, this model can prove to be a little slower. The FPS as calculated by the focal loss paper [53] for this architecture is 5 frames per second with an mAP value of 36.2%. On the pills dataset mentioned before [54], the FPS is 7 frames per second with 87.69% mAP.

3.3. YOLO – You Only Look Once

Although being a relatively newer models, YOLO has gained a lot of popularity in the recent years because of its fast and accurate prediction capabilities. The first version of

YOLO came in 2016 [56] and made a great change in the field of object detection and computer vision. Since then, more advanced versions of the model have been produced by the open source community to bring this technology to a new level. The basic working of the model as compared to the two stage models like Faster RCNN is pretty straight forward and is observable from the name. Unlike Faster RCNN which creates region proposals in one stage and then performs classification task based on the region proposals in the second stage, YOLO only looks once for the objects and hence is a single stage detector. The model architecture is more like an FCNN [57] where the images are given as an input to the FCNN, and the predictions are in the form of an array that contains the information about the position of the detected object in the image. Figure 1 shows a high level representation of this concept.

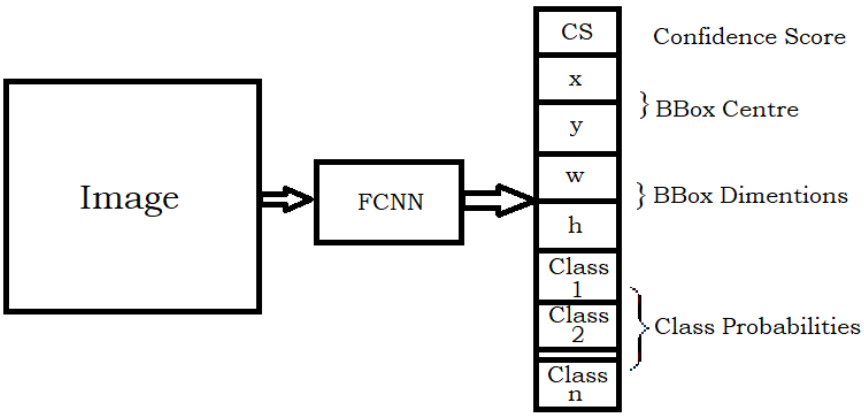


Figure 1: YOLO high level working

For this project we will be considering the latest version of YOLO because it supports our problem in multiple ways. Firstly, the mAP values for YOLOv8 when tested on COCO dataset are far higher (52.9% for large and 53.9% for extra-large model [58]) than the previously mentioned frameworks. The model is extremely fast in real time detections with high FPS during inferencing. On COCO dataset the speed is documented as 2.39 ms for large and 3.53ms for extra-large model [59]. This is around 42 FPS for the large model. The latest updates in the model also provide with a built-in tracking mechanism that can be helpful for our task. The tracker can be useful to track birds while we create a counter so that no bird is counted multiple times. This model is getting a lot of recognition and provides easy integration with most famous frameworks for deployment of the model in real world. This will be helpful in the future when we need to deploy the model for actual bird predictions.

To conclude this section, we can say that the YOLOv8 model is by far provides the most accurate and precise prediction with a relatively high speed. It is also easy to use and integrates easily with most of the other frameworks that we might need to use in the future. Hence, we will be using this architecture as a base model for our training. Table 1, Figure 2, and Figure 3 provide the summary of the mAP and FPS values we discussed in this section.

Table 1: Comparison of model performance metrics on COCO Val set

Model	**mAP**	**FPS**
SSD	31.2	8
Faster RCNN	36.2	5
YOLOv8 (large)	52.9	42

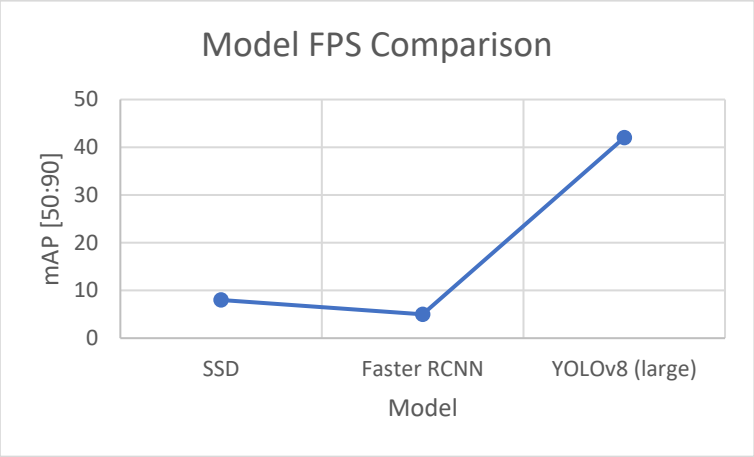


Figure 2: FPS Comparison of Models

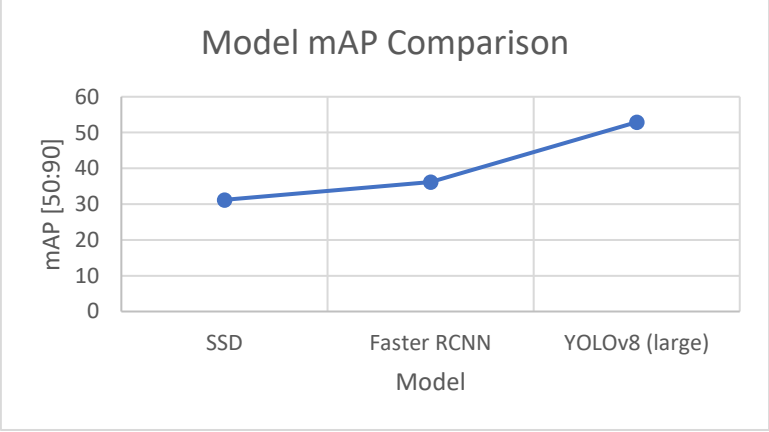


Figure 3: mAP Comparison of Models

Chapter 4

Methodology

4. Methodology

This chapter focuses deeply on the complex specifics of this project. The main goal is to give a detailed view of the work that is performed during this project. In order to advance avian research and conservation efforts, this study aims to create distinct models for generic bird detection, specific bird classification, and bird counting. In this chapter, we will be looking at the procedure conducted for data acquisition, data cleaning, and data pre-processing, followed by training various models, and conducting hyperparameter tuning.

The fundamental objective of this research is to leverage imagery from unmanned aerial vehicles (UAVs) to identify and count birds while they are in flight. UAVs give researchers a special vantage point from which they can take high-resolution images from aerial perspectives, creating an unprecedented opportunity for avian monitoring and analysis. To get the best results possible, a multi-stage approach is considered. The details of these stages will be discussed in the coming passages.

4.1. Data Acquisition

The first stage of this project is the acquisition of the UAV images. These images perform as the foundation for this project as the whole training process is dependent on these images. Ideally, the best approach is to plan and execute flights in the areas where bird populations are known to be present, but for the purpose of ease and due to the shortage of time, the complete dataset is acquired from various online resources.

For the generic bird classification model, UAV images of various bird species are collected from multiple geographic locations. The dataset thus created consist of a variety of birds in different backgrounds, like sand, rock, trees, etc. making it beneficial for the model to generalize on these images [60]. The combined dataset is made up 13 individual datasets comprising of around 279,000 images. These datasets are collected from locations like Everglades (Florida, USA), Palmyra (South Pacific), Antarctic Peninsula Guinea, South Shetland Islands (Antarctica), Falklands Islands, Saskatchewan (Canada), Cape Cod (USA), Indian Ocean, Utah (USA), New Mexico (USA), Lake Michigan (USA), and Poland. A detail of all these datasets is given in *Table 2*. As we aim to create a generic bird detection model, we will only be training the model under one label, ‘Birds’. Figure 4 shows a few of the images from the dataset.

Table 2: Combined Dataset Distribution

Location	**Type**	**Annotations**
Everglades (Florida, USA)	Wading Birds	56516
Palmyra (South Pacific)	Seabirds	1761
Antarctic Peninsula	Penguins	3037
Guinea	Terns	23091
South Shetland Islands (Antarctica)	Penguins and Shags	27865
Falklands Islands	Albatross and Penguin	9338
Saskatchewan (Canada)	Marsh Birds	1658
Cape Cod (USA)	Ducks	26643
Indian Ocean	Seabirds	10173
Utah (USA)	Pelicans	44742
New Mexico (USA)	Ducks, Geese and Cranes	2756
Lake Michigan (USA)	Gulls	64432
Poland	Water Birds	7584

Figure 4: Dataset Images

For the birds classification model, the dataset used is acquired from the Zoological Society of London [61]. The dataset is in the form of a large '.tiff' file that contains an orthomosaic image consisting of 21,066 instances of birds spread all over the landscape. The orthomosaic is based on Point Nord (Fatick Region, Senegal) and is shown in Figure 5.

Figure 5: Seabirds Dataset (Point Nord)

The dataset contains six named classes, African Royal terns, Caspian Terns, Slender-billed Gulls, Grey-Headed gulls, Great Cormorants, and Great White Pelicans. There are also some other species of birds also present which could not be identified and are termed as unknown in the dataset, making the total number of classes as seven. The dataset was created using a 20 Megapixels wide coverage camera on a 'DJI Phantom 4' drone at altitudes ranging from 20 to 50 m depending on the behaviour of the birds towards the drone to avoid any disturbance to their environment. The royal terns are the main focus of this dataset and contain a large number of instances as compared to the other six classes, causing an imbalance in the dataset. The exact numbers of the instances for all six classes are shown in *Table 3*.

Table 3: Seabirds Dataset Class Distribution

Class	*Instances*
Royal Terns	19742
Caspian Terns	450
Gray Headed Gulls	951
Slender Billed Gulls	24
Great Cormorant	72
Great White Pelican	43
Unknown	229

Figure 6 below helps visualize the difference in the number of instances in the seven said classes. We will discuss certain ways on how we can fix this imbalance in the dataset.

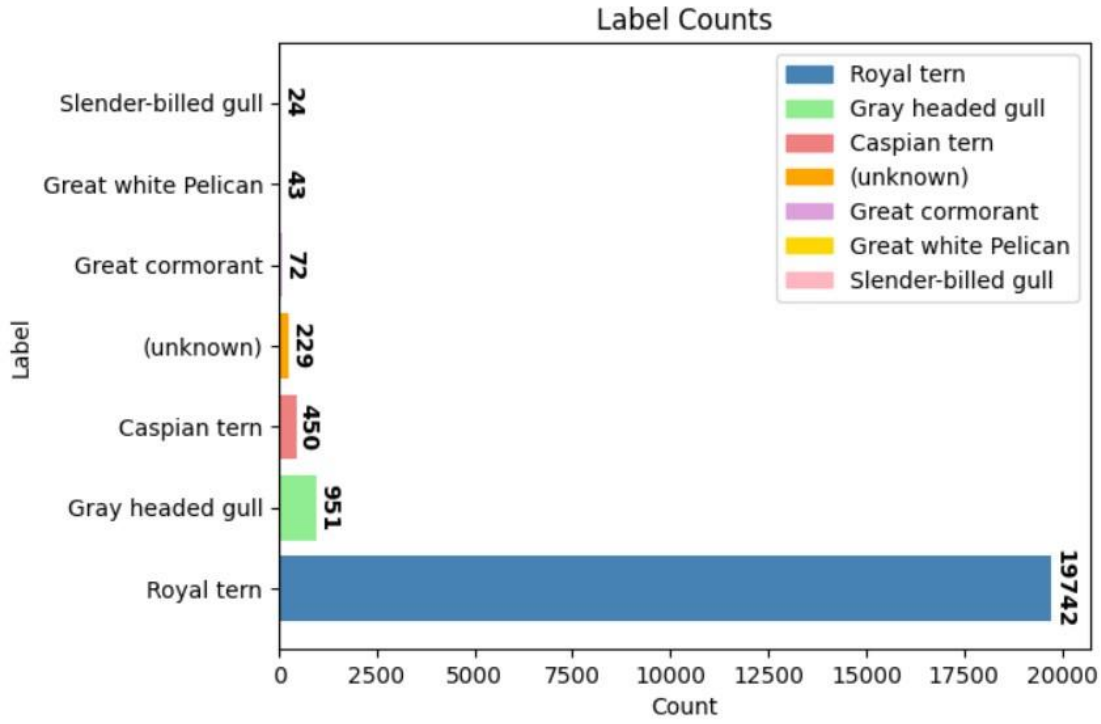


Figure 6: Label counts (Birds Species Classification)

The datasets also contain '.csv' files that consist of important information including the birds class names and the point coordinates of these birds on the orthomosaic. These coordinates will be helpful in determining the position of the birds in the dataset.

4.2. Data Cleaning and Pre-Processing

The combined dataset for the generic bird classification model contains CSV files for all 13 datasets. The CSV files contain bounding box coordinates of the birds in the images. Similarly, for the specific birds classification model, the coordinates present in the CSV file were used to find the exact location of the birds in the orthomosaic image. This was done by finding out the relation between the actual metric-coordinates and the pixels on the orthomosaic as the CSV file gives the actual metric-coordinates instead of the pixel coordinates of the birds. The dimensions of the location mentioned is 292.48 × 305.69 square meters and the image dimensions are 27,569 × 28,814 square pixels. Using this relationship, we were able to plot the points on the orthomosaic at the locations where the birds were present as shown in Figure 7 and Figure 8.

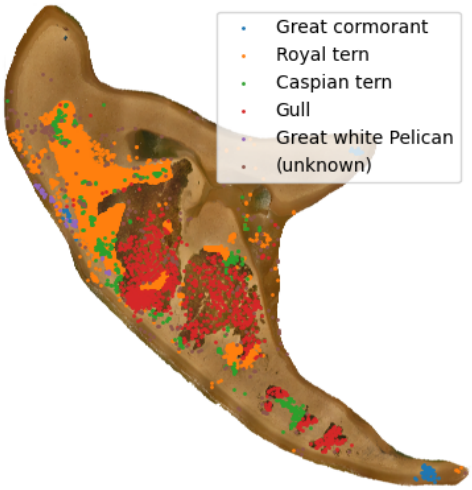


Figure 7: Coordinates on orthomosaic

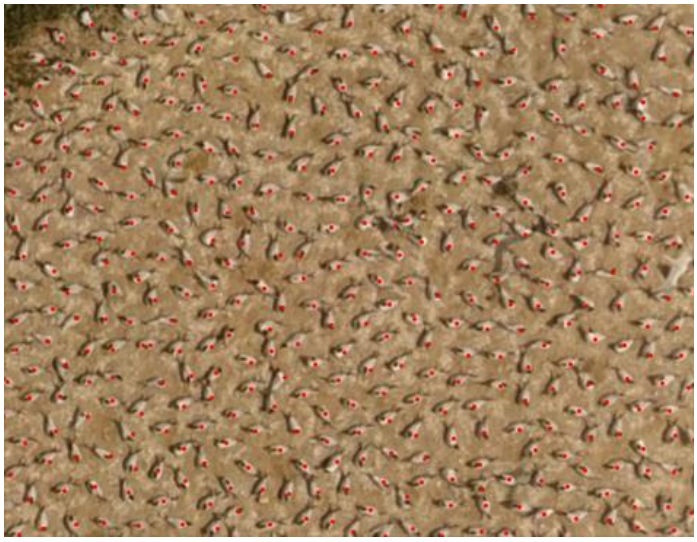


Figure 8: Coordinates of birds plotted

After finding the location of the birds, we divided the orthomosaic into smaller, non-overlapping, tiles of 800px by 600px along with the point coordinates to the birds in these tiles. We estimated the average bird size for all the species and to create bounding boxes and used the point coordinates as centre points. Figure 9 shows the bounding boxes for the class 'Caspian Tern'.

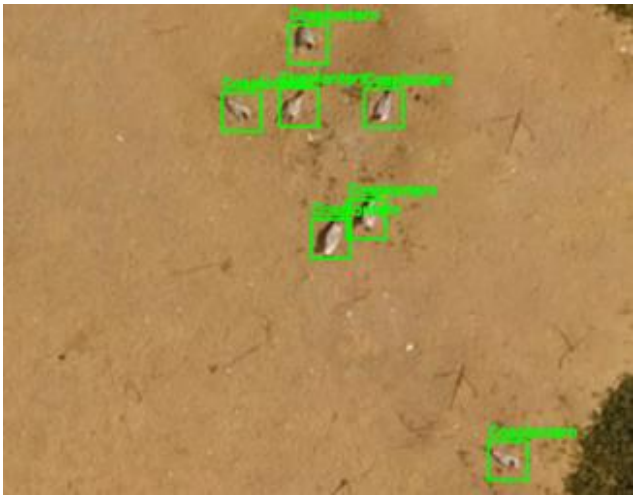


Figure 9: Bounding boxes on image

To overcome the imbalance in the dataset multiple techniques were used to create more image data for the models to train on. The classes 'Gray Headed Gulls' and 'Slender Billed Gulls' highly resemble to each other and the 'Slender Billed Gulls' are very few in number, so we decided to merge these classes to create a new class called 'gull'. We used image editing software like Photoshop [62] to create new instances of the classes 'Great

Cormorant' and 'Great White Pelican'. The AI powered tool in the software helped in copying the individual birds alone leaving the background. We than pasted these birds on different backgrounds, like grass water, sand, etc. in multiple different angles so that these appear to be new images to the model. We than hand labelled these new images to increase the efficiency. We also used image augmentation technique by leveraging the ImgAug library [63] that helped in creating more samples for the remaining two classes. The class 'Royal Tern' is highly oversampled. We reduced the samples for this class to be closer to the other classes. As the birds from multiple species are overlapping in many images, it is hard to get exactly the same number of instances for each class. Using these methods, we were able to create a dataset that is relatively balanced than before.

Now that the bounding box coordinates of all the labels are known, annotations files for our model can be created. For this a python script is used that reads all the box coordinates and convert them in to center point and dimensions relative to image dimensions as required by YOLO algorithm. After creating all the annotations files for each image, we are ready to use these images and annotations to train our model. To leverage the capabilities of YOLOv8 we will use PyTorch [64] framework as recommend by the model documentation.

4.3. Hyperparameter Tuning

Choosing correct hyperparameters [65] before training a deep learning model is very essential as the training process depends greatly on these hyperparameter values. For object detection task we need to configure these values in the config file keeping in view what outcome we require from the model. A few important hyperparameters that we configured for the model training are discussed below.

4.3.1. Learning Rate (lr0, lrf)

In YOLOv8 two hyperparameters, initial and final learning rate (lr0, and lrf) are used to set the start and end points for the learning rate scheduler. The learning rate scheduler sets a learning rate, and the learning rate determines the step size at which the model's weights are updated in each iteration to minimize the loss function and improve the model's performance.

The initial learning rate, lr0, is the starting value selected for the learning rate scheduler as the training starts. It defines the magnitude of the initial updates to the weights. A larger value of lr0 can lead to a fast convergence but it may cause overshooting in the optimization process. Similarly, Final learning rate, lrf, is the value of the learning rate at the end of the training process and is usually taken smaller than the initial value allows the optimization to fine-tune the model gradually as it approaches convergence, improving generalization.

For this experiment, We trained the model initially for ten epochs with varying initial and final learning rates to investigate how these hyperparameters affect the validation loss. We used an initial learning rate of 0.01, which is the default value, and five different final learning rates [66]: [1.0, 0.1, 0.01, 0.001, 0.0001]. The results showed that the best performance was achieved when using an initial learning rate of 0.01 and a final learning rate of 0.0001. This implies that a moderate initial learning rate, combined with a very small final learning rate for fine-tuning, allowed the model to achieve better results with respect to validation loss. Figure 10 presents the validation loss curves for the three best results from your experiment. It indicated that the initial learning rate was consistently set to 0.01, and the final learning rate was experimented with three different values: 0.01, 0.001, and 0.0001. Among these three, the best results were observed when the final learning rate was set to 0.0001.

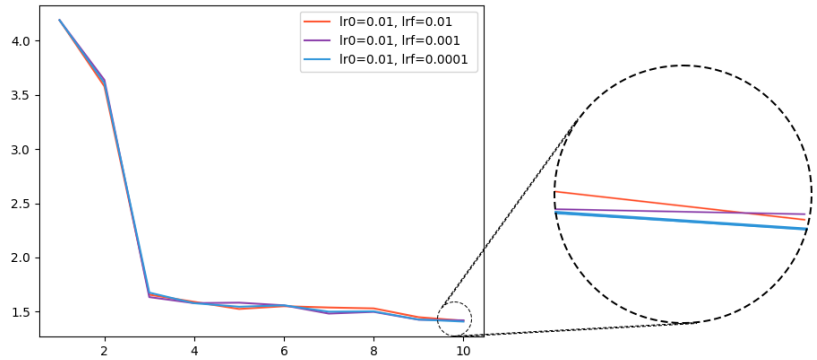


Figure 10: Tuning the learning rate

4.3.2. Warmup Epochs

Warmup epochs are essential to the training of YOLO and other deep learning models. Warmup Epochs are used to progressively raise the learning rate from a very low value to an appropriate starting learning rate. This slight rise aids in the model's early training stability and helps to prevent convergence problems. When employing high initial learning rates, warm-up epochs are especially crucial since they facilitate an easier transition to the main training phase.

Following the first experiment, We trained the model using various learning rates and looked into the effects of different warm-up epochs. Three final learning rate values were used: 0.01, 0.001, and 0.0001. The initial learning rate was set at 0.01. We experimented with warmup epoch settings of 2, 3, and 5 to see which arrangement produces the best outcomes [66]. As the default value for warmup epochs is 3, Figure 10 depicted the results for the default warmup epoch value. Figure 2 showcases the outcomes of the experiment when using different warmup epochs.

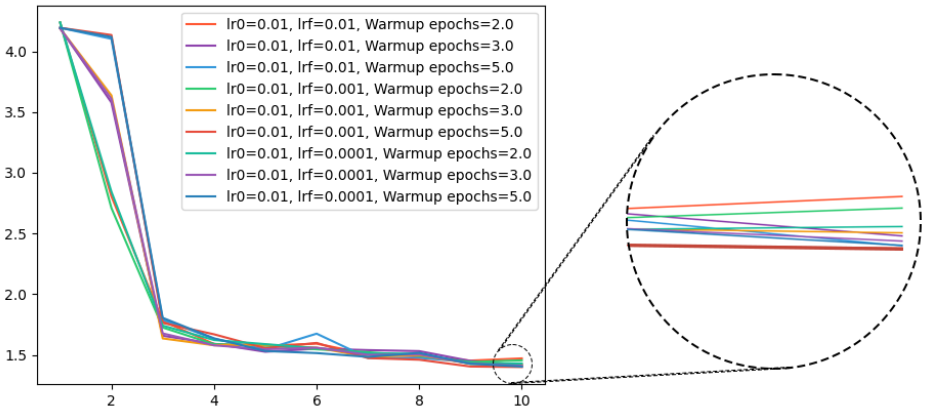


Figure 11: Tuning the Warmup Epochs

The results of this experiment showed that a warmup epoch value of 5 at initial and final learning rates of 0.01 and 0.001 respectively produced the best outcomes. This shows

that the model increased its stability and performance by progressively raising the learning rate across five epochs. But it's also noteworthy that the curve (with a warmup epoch of 3) that performed well in the previous plot is still an option. This implies that the default warmup epoch value may likewise produce acceptable outcomes but testing with a larger warmup epoch value — in this example, 5 — led to even greater performance.

4.3.3. Warmup Momentum

In the next experiment, We changed the warmup momentum values for the two top models we had previously described and retrained them for 10 epochs. For each training run, we chose values in range of 0.7 to 0.95 with a 0.05 increment to see how the warmup momentum impacts the YOLO model's performance [66]. We discovered that the model that produced the best results had a warmup momentum value of 0.95 and a final learning rate of 0.001. The model appears to have converged more successfully during the early training phases due to a larger warmup momentum, which also appears to have improved overall performance as measured by validation loss and other metrics. When employing models with a final learning rate value of 0.0001, We also observed a surprising result. Despite having good validation loss values, these models' accuracy, and recall scores for the "royal tern" class were entirely opposite. Since the precision was 100%, the model's predictions of the "royal tern" class were almost always accurate. The model, however, missed the majority of occurrences of the "royal tern" class, as seen by the recall of 0%, shown in Figure 12.

```
Ultralytics YOLOv8.0.145 🚀 Python-3.10.6 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l summary (fused): 268 layers, 43611234 parameters, 0 gradients
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:04<00:00,  1.62s/it]
                   all         93        606      0.565      0.467      0.594      0.242
       Greatcormorant         93        109      0.407      0.807      0.657      0.172
    GreatwhitePelican         93        109      0.497      0.963      0.948      0.454
          Caspiantern         93         95      0.796     0.0105      0.392      0.176
                 Gull         93         93      0.293      0.742      0.458      0.193
            Royaltern         93        111          1          0      0.805      0.334
            (unknown)         93         89      0.397      0.281      0.302      0.125
Speed: 3.7ms preprocess, 12.5ms inference, 0.0ms loss, 3.0ms postprocess per image
```

Figure 12: Opposite Recall and Precision for lrf = 0.0001

The model with a final learning rate of 0.0001 may have been excessively conservative in its predictions for the "royal tern" class based on the difference in accuracy and recall. It only generated predictions when it was very certain, which produced very few false positives but missed many true positives. In comparison, the model with a warmup momentum of 0.95 and a final learning rate of 0.001 managed to balance accuracy and recall better. It detected more true positives while still making certain predictions.

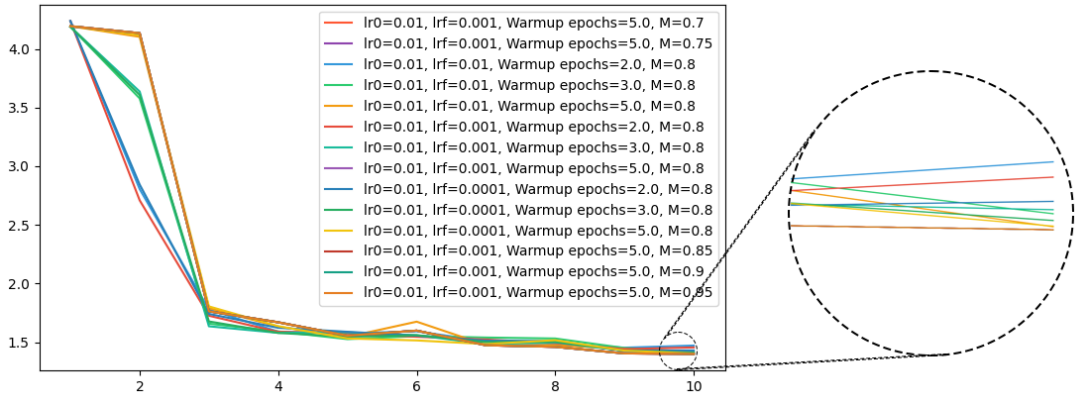


Figure 13: Tuning the Warmup Momentum

4.3.4. Optimizer

In the next experiment, we decided to employ the previously found optimal hyperparameters lr0 = 0.01, lrf = 0.001, warmup_momentum = 0.95, and warmup_epochs = 5.0. The objective was to figure out how the model's performance varies when different optimizers are used. To do this, we trained the model using three different optimizers [66]: SGD, Adam and RMSProp. By default, the optimizer used by the model was set to "auto". The implementation of the "auto" optimizer is particularly interesting as shown by the YOLOv8 trainer.py file code snippet in Figure 15. It alternates between two optimizers dynamically dependent on the number of iterations during training. When the number of iterations is less than 10,000, the AdamW optimizer is used initially. AdamW is a weight decay variation of the popular Adam optimizer that helps prevent overfitting and enhances

generalisation. However, If the number of iterations is more than 10,000, the "auto" optimizer switches to SGD (Stochastic Gradient Descent). SGD is a traditional optimisation approach that updates the parameters of the model based on the gradient of the loss function with respect to the parameters. It has a high computing efficiency and is commonly utilised in deep learning training. In our experiments, because we never go over 10,000 iterations, the "auto" optimizer performs basically the same as the AdamW optimizer. This signifies that the model uses the AdamW optimisation method to update its parameters throughout the training period and does not transition to SGD.

```
631         if name == 'auto':
632             nc = getattr(model, 'nc', 10)  # number of classes
633             lr_fit = round(0.002 * 5 / (4 + nc), 6)  # lr0 fit equation to 6 decimal places
634             name, lr, momentum = ('SGD', 0.01, 0.9) if iterations > 10000 else ('AdamW', lr_fit, 0.9)
635             self.args.warmup_bias_lr = 0.0  # no higher than 0.01 for Adam
```

Figure 14: trainer.py (Code Snippet)

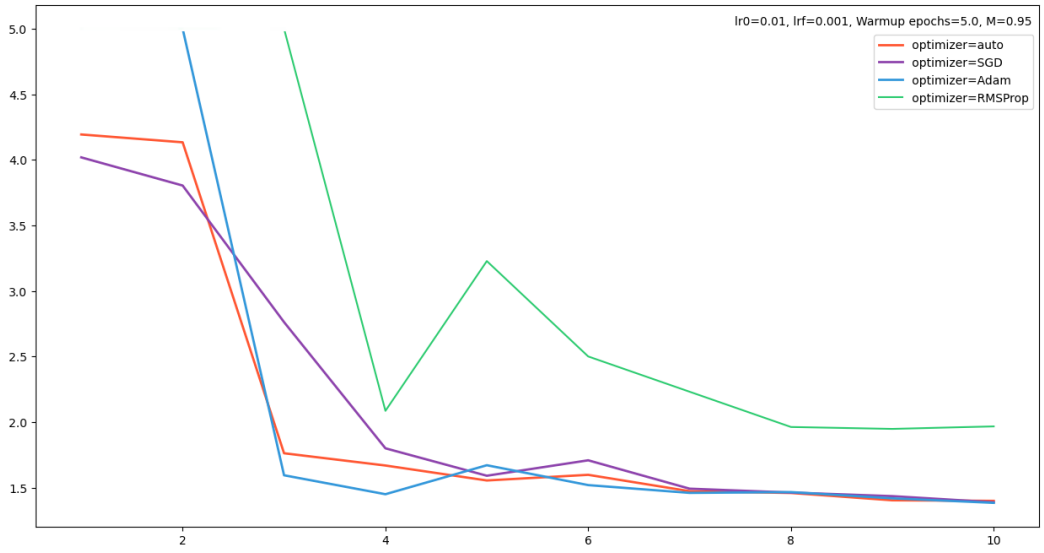


Figure 15: Tuning the Optimizer

The results show that the RMSProp optimizer does not perform as good as the other three optimizers. SGD, Adam, and auto optimizers start fairly different than each other but after some epochs the loss curves become fairly similar. To get more information on this, training till ten epochs is not enough. Till this point, it is decided that the learning rate

hyperparameters that will be used for training the model are lr0 = 0.01, lrf = 0.001, warmup_momentum = 0.95, and warmup_epochs = 5. But we will also train the model completely with the SGD and Adam optimizer to see the effect over greater number of epochs. The model will be trained initially with 300 epochs early stopping patience of 50 epochs. This means if the training starts to converge before 300 epochs it will stop training. Patience of 50 epochs mean that the model will train 50 epochs more than the best training checkpoint to see if the results get any better, and if they do not the training process will stop.

4.4. Training

After the hyperparameter tuning, the next and the most important step is to train the models completely to detect and track the birds to count them. In this section, we will start by training the YOLOv8 model initially on the default hyperparameters. We will observe the model training by checking the training and the validation loss over the epochs to see when the model converges. After training on default hyperparameters we train both the generic and the specific species classifier models on the hyperparameters decided in the previous section. As discussed earlier, we will also observe the effects of different optimizers on the model training as they slightly differ in initializing the weights and biases throughout the training process.

To train a yolo model we need to make sure that we have all the libraries and dependencies available before we start. We will be using PyTorch framework throughout this project for training and inferencing. The YOLOv8 model is available in a ‘pip’ package in the Ultralytics library. To train the model we also need to make sure that a directory structure is followed. The directory structure is shown in Figure 16. In the data folder, we need to put the images and the labels. We can create separate test and train folders in both images and labels folder to train and validate on different example images. The model also needs a configuration file in YAML format. This configuration file contains all the information

necessary to train the model, including the path to the data folder containing images and labels. The class names are also provided in this YAML file as shown in Figure 17.

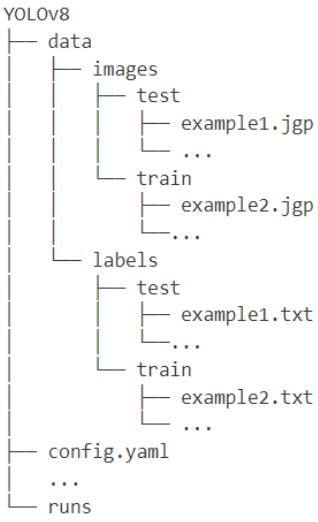


Figure 16: Directory Structure

```yaml
# dataset root dir
path: 'C:/Users/AI/YOLOv8/data'

# train images (relative to 'path')
train: 'images/train'

# val images (relative to 'path')
val: 'images/test'

#Classes
names:
  0:  Greatcormorant
  1:  GreatwhitePelican
  2:  Caspiantern
  3:  Gull
  4:  Royaltern
  5:  (unknown)
```

Figure 17: Configuration file

4.4.1. Species Classification Model – Default Hyperparameters

The species classification model was trained with the default hyperparameters, and its performance was constantly evaluated during the training process. We first saw a significant reduction in training loss, which is a promising sign. We performed regular validation after each epoch to assure the model's growth. We saw a huge decline in the loss curve during the early phases of training, suggesting the warmup phase of the learning process as shown by the learning rate curve in Figure 18.

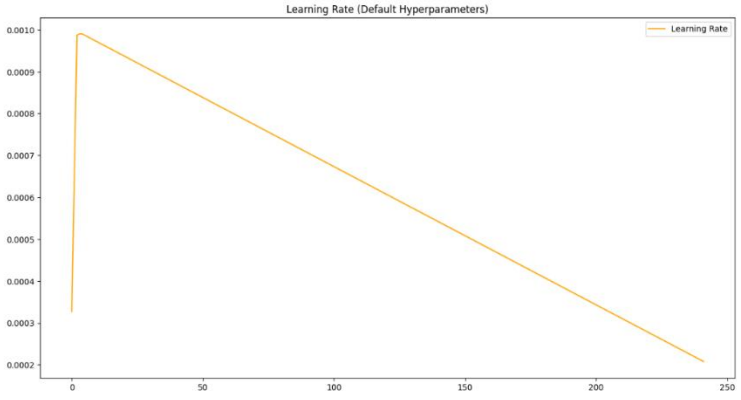


Figure 18: Learning rate (Default Hyperparameters)

As predicted, the learning rate steadily decreased, resulting in a more gradual reduction in training loss. This behaviour matches our expectations. However, after 147 epochs, we noticed a significant shift in the validation loss curve. It began to rise, indicating that the model's learning capability had hit a plateau and it had attained its top performance on the presented dataset. Any more training would result in overfitting - the model would become too specialised in the training data and would fail to generalise successfully on unknown data. This is observable in the plot in Figure 19.

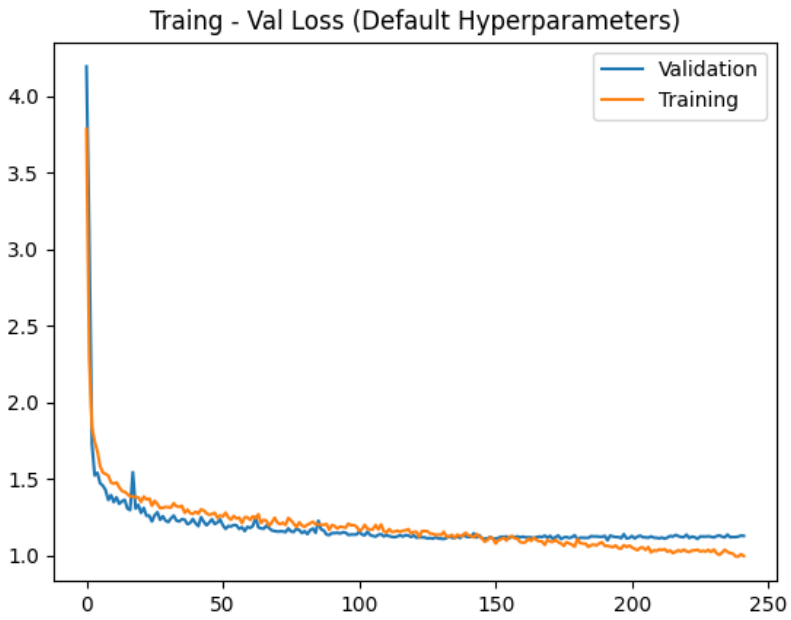


Figure 19: Training - Validation Loss (Default Hyperparameters)

This is also observable from the mAP curve at 50-95 IOU in Figure 20, as the mAP value stops increasing after the model is completely trained. Recognising this phenomenon, we determined that the model was completely trained at this point. Continuing the training after this point would provide no significant advantages and might potentially damage its performance on fresh, unknown data. As a result, we exercised care and halted the training process in order to maintain the model's potential to generalise successfully.

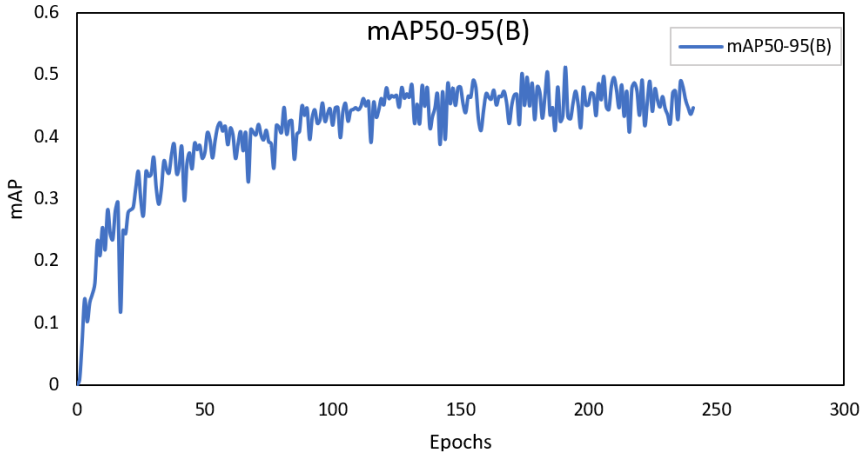


Figure 20: mAP 50 - 95 (Default Hyperparameters)

4.4.2. Species Classification Model – Tuned Hyperparameters (Auto Optimizer)

The after getting a base line for model, we trained the model with the tuned hyperparameters. The initial learning rate was set to 0.01 which is actually the default value. The final learning rate value was reduced from 0.01 to 0.01 create a steeper slope. The warmup epochs were set to 5 instead of 3 with a warmup momentum of 0.95. As discussed earlier, we will be using three different types of optimizers. In this experiment, the optimizers we used is the auto optimizer. The auto optimizer switches between AdamW and SGD optimizer depending on the number of iterations. As we do not go above 10000 iterations, the auto optimizer essentially behaves as AdamW optimizer. The loss value starts to drop drastically in the first 5 epochs because of the warmup, giving an initial boost to the learning process, then after the first 5 epochs, the loss reduces gradually as the learning rate reduces slowly. Both the training and the validation losses reduce till epochs. After 155 epochs, the training loss keeps on going down, but the validation loss stops this behaviour. It remains in the same range suggesting that the model has completed its learning and is not capable to generalize after this point. This is visible in Figure 21.This is also observable from the mAP 50-95 plot in Figure 22. After the model has completed its learning, the mAP does not go any up any further and tends to oscillate between a fixed range.

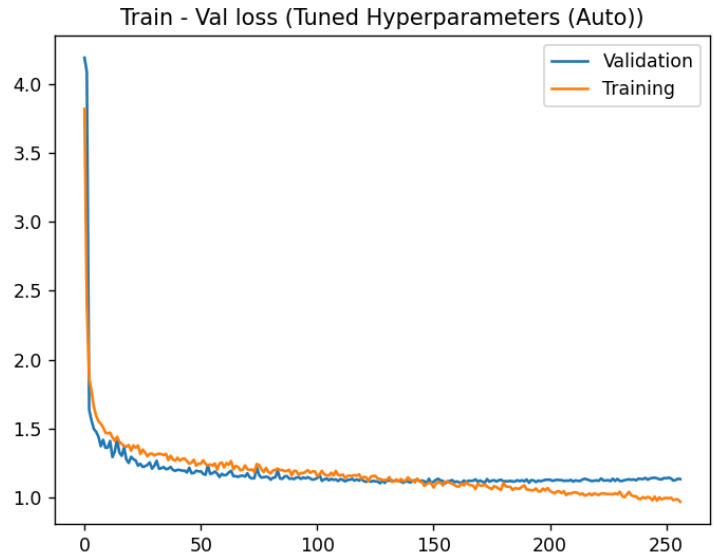


Figure 21: Training - Validation Loss (Auto)

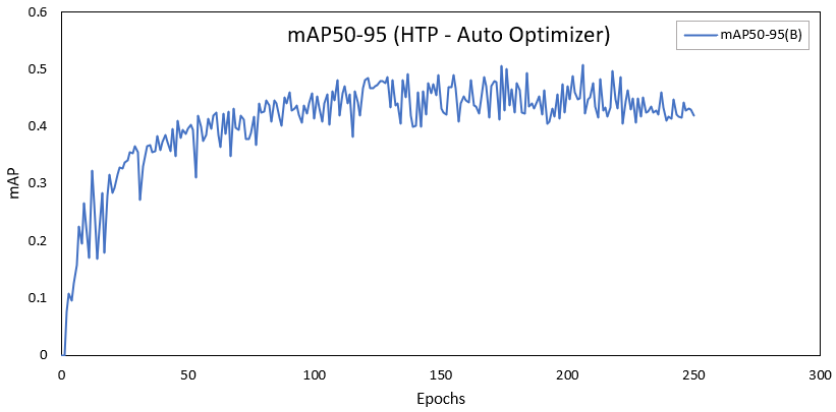


Figure 22: mAP 50 - 95 (Auto)

4.4.3. Species Classification Model – Tuned Hyperparameters (Adam Optimizer)

After using the auto optimizer, in the next experiment we trained the model on same hyperparameters but using the Adam optimizer. The training and validation plots show a similar trend, fast drop in the beginning due to warmup and a gradual reduce afterwards. A difference that we observed is that the model starts to converge sooner than in case of auto optimizer. At 138 epochs, the validation and training loss cross each other. After this point the model training only yields overfitting. The mAP curve also suggests that as there is no visible

progress after this point. Hence the training is stopped after this checkpoint to keep the model generalized. The loss and map curve are shown in the Figure 23 and Figure 24 respectively.

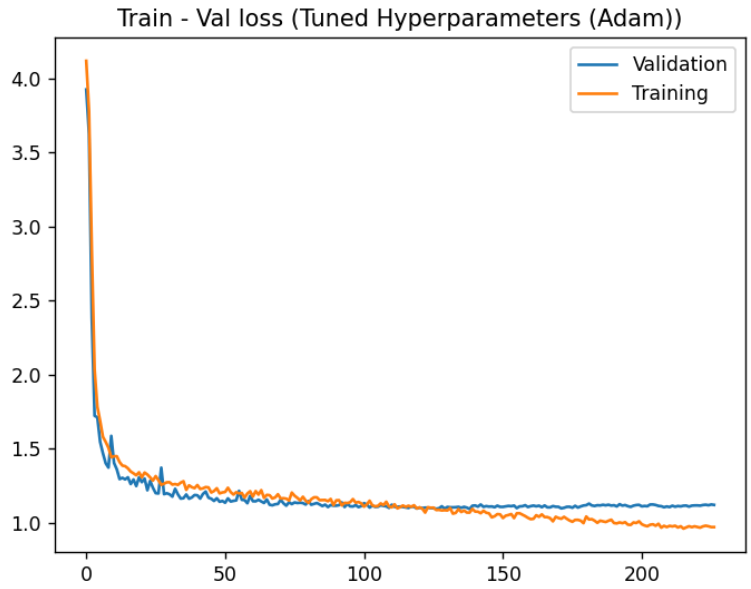


Figure 23: Training - Validation Loss (Adam)

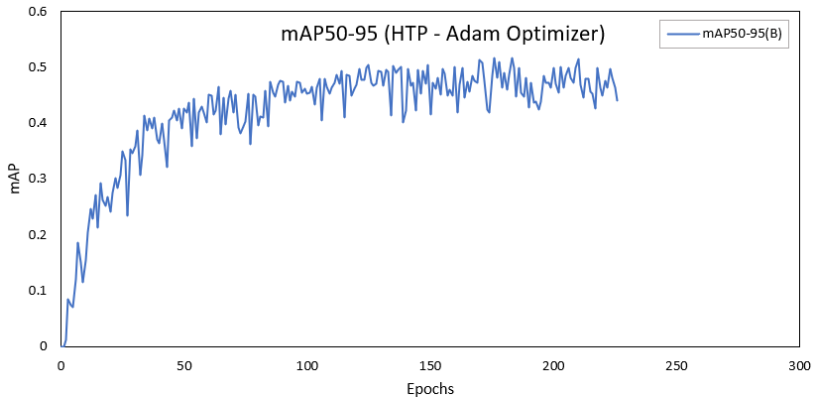


Figure 24: mAP 50 - 95 (Adam)

4.4.4. Species Classification Model – Tuned Hyperparameters (SGD Optimizer)

In the next experiment, we will be using SGD optimizer with the same hyperparameters and observe the training and the validation losses from this optimizer. The basic trend is the same as for the previous experiments. The Plot in Figure 25 shows that the model starts to converge almost similar to the Adam optimizer. On further investigation, we found out that for this

model also the validation curve cuts off the train curve at 138 epochs, same as the Adam optimizer. The mAP 50-95 curve show the same trend as well. The mAP curve is shown in Figure 26.

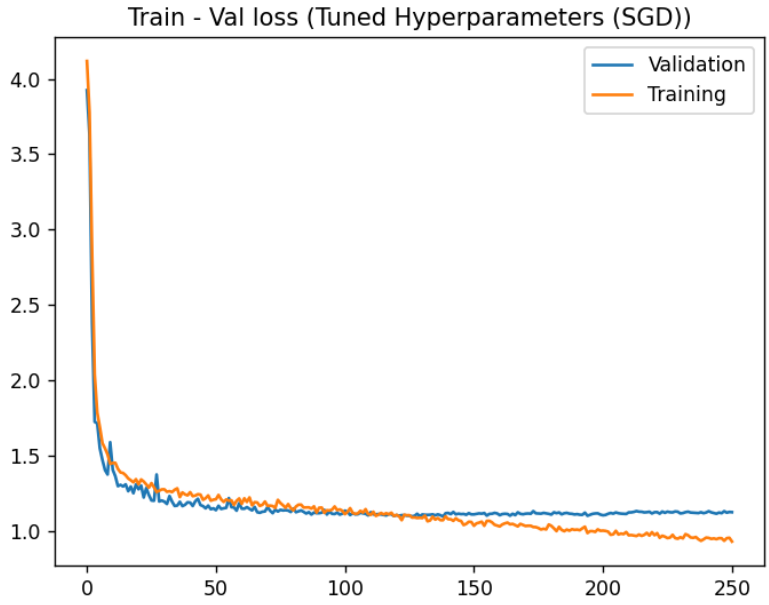


Figure 25: Training - Validation Loss (SGD)

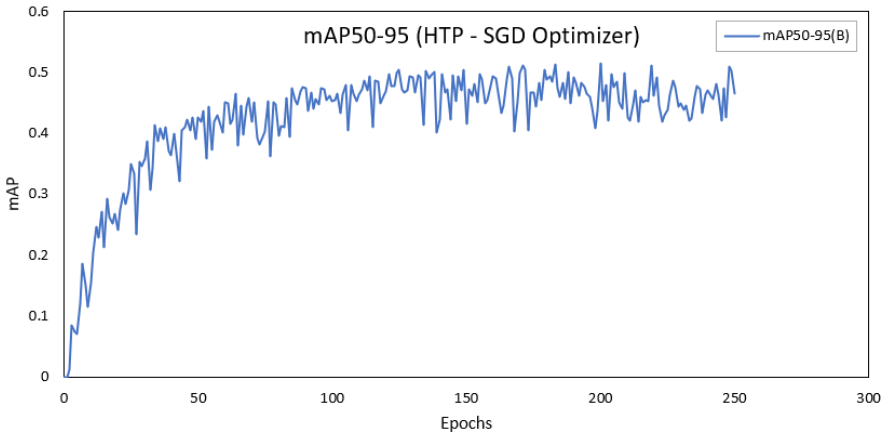


Figure 26: mAP 50 - 95 (SGD)

4.5. Generic Birds Detection Model

For the generic model, we decided to go with the same hyperparameters as calculated before, to see if they perform better than the default hyperparameters, and if we can use these

hyperparameters for training the general bird classification model. It is not much surprising that the model behaves similarly to the chosen hyperparameters as the dataset and the problems are similar with similar features to be worked on. We decide to see the effect of changing the optimizer while training the model with a larger dataset. We kept the model hyperparameter tuning process for generic bird detection model basic and simple as we had limited resources and time to perform a full scale hyperparameter tuning.

4.5.1. Generic Birds Detection Model – Tuned Hyperparameters (SGD Optimizer)

When tuned with the chosen hyperparameters and the SGD optimizer, the initial behaviour of the model was according to what we predicted and discussed before. A noticeable thing is that the loss curve for validation dataset is a little higher than the training curve, but the trend seems to be persistent. The loss reduces over the number of epochs and after around 170 epochs, the loss starts to go up in the validation curve. This means that the model has stopped generalizing after this point and is slowly going towards overfitting. This trend is visible in the training and validation loss curve shown in Figure 27.

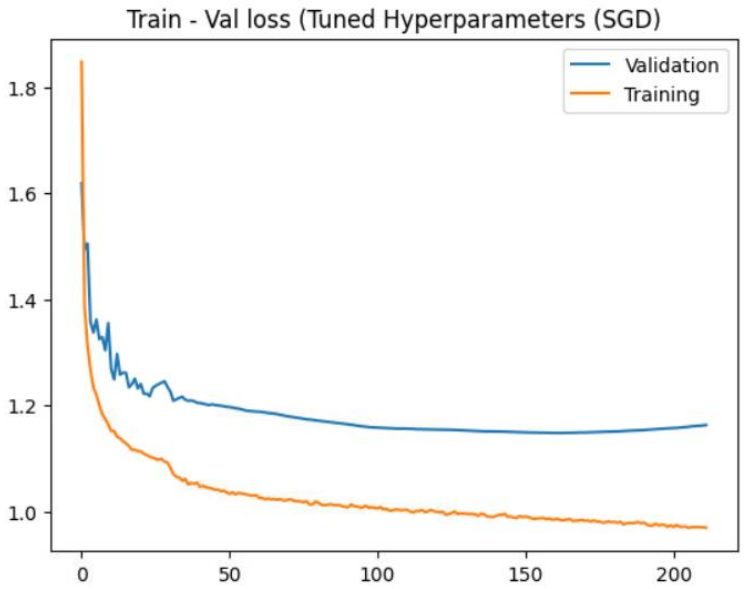


Figure 27: Generic model loss curve (SGD)

The same trend can be seen in the mAP at 50 – 95 IOU curve shown in Figure 28. The mAP value keeps increasing till the 170 epochs range, and after that we see a slight drop in the value. The model was set to train till 300 epochs with patience of 50 epochs. As around 170, the best value for the curve was achieved, the model kept training for another 50 to mitigate the chance of losing good results. After training for the next 50 epochs, when no better result is seen, the training process stops automatically.

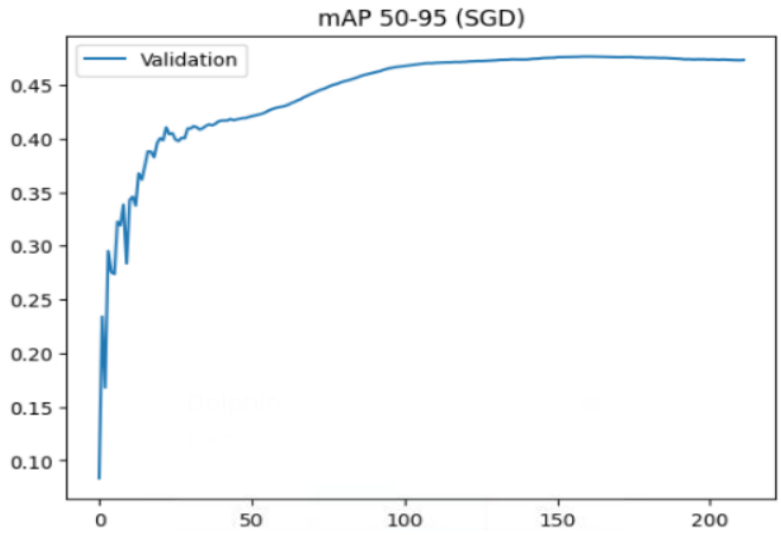


Figure 28: Generic model mAP 50 - 95 (SGD)

4.5.2. Generic Birds Detection Model – Tuned Hyperparameters (Adam Optimizer)

When trained with an Adam optimizer, the generic birds classification model shows very interesting results. The training and validation curves shown in Figure 29 are a good example of why we used a patience of 50 epochs for every training process we performed. We can see that the validation loss curve tends to go down till the first 50 epochs, but then it starts to go up suddenly. From our previous experience with this model, we know that 50epochs of training are not enough for complete training, but if were to monitor it manually we could have that this point as convergence point and stopped training. But interestingly enough, after around 30 more epochs the validation curve starts to go down again. The curve keeps going down till around 170 epochs and after this starts going up suggesting that the

model learning process is now complete. This is also visible from the mAP plot Figure 30. The mAP 50 – 95 value stops to decrease around the same time when the validation loss starts to increase. The curve for the Adam optimizer, although had a point where the loss started to increase, it came back on track and the loss value is also lower as compared to the pervious variant that used SGD optimizer.

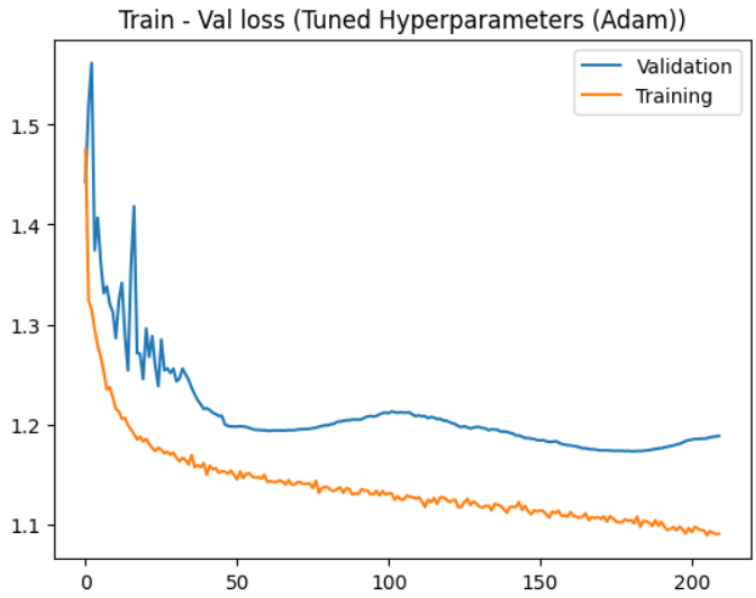


Figure 29: Generic model loss curve (Adam)

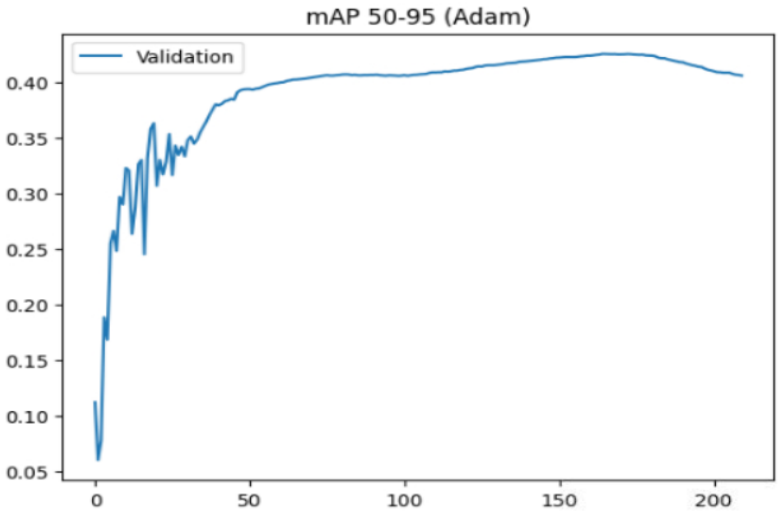


Figure 30: Generic model mAP 50 - 95 (Adam)

4.5.3. Generic Birds Detection Model – Tuned Hyperparameters (Auto Optimizer)

The loss curves for the third variant of our model, using the auto optimizer show good trend as the loss values are lower as compared to SGD and the reduction in the loss curve is also smoother as compared to the Adam optimizer. The curves shown in the Figure 31 also show that the training process end somewhat around 170 epochs. The same behaviour is shown in the previous two models. The mAP 50 - 95 for this model is also higher than the previous variant and goes around 47 percent. This is shown by the mAP curve displayed in Figure 32, where the mAP value keeps on increasing slowly till 170 epochs and after reaching the maximum height, it starts to reduce, hinting that the training is complete.

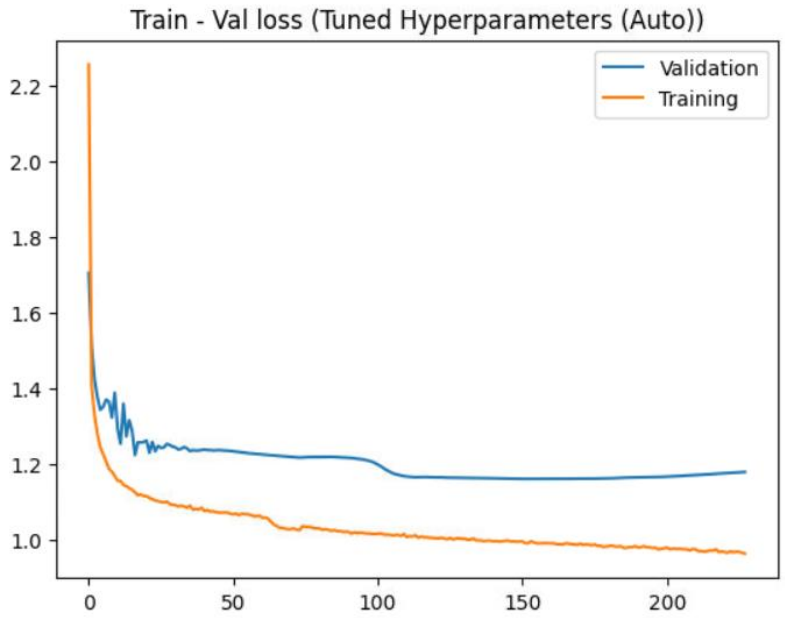


Figure 31: Generic model loss curve (Auto)

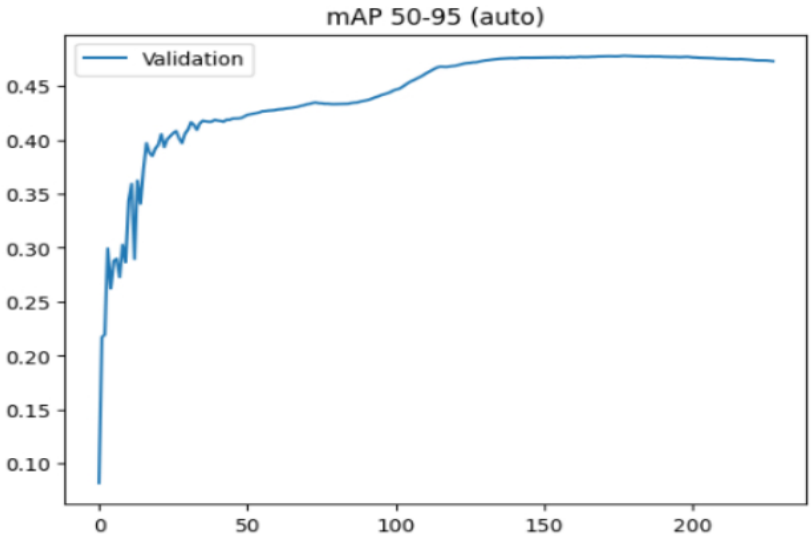


Figure 32: Generic model mAP 50 - 95 (Auto)

4.6. Birds Counting

After we have trained our object detection model for detecting the birds, our next task is to count the total number of birds in the area. The hard thing about this task is to keep a track of the birds that have already been counted. We do not want to make multiple counts for same birds again and again. For this we can use a technique called Kalman Filter. The Kalman Filter [67] is useful where we have an approximation of the current position of the object in the frame and can use it predict the next position of that object. For our task we will be using a technique called ByteTrack [68] that utilises Kalman filter to assign ids to the directions in each frame based on matching IoU value of previous and latest detection using appearance and motion prediction. A plus point here while using ByteTrack is that in the latest YOLOv8 update, we have ByteTrack feature built into the system which makes this task a bit easier.

After tracking and assigning Ids to the detections we can dedicate a region in which, if present, the object gets counted. In this way, if the object passes through that region of the frame, its Id is noted down and the counter is updated. This is depicted in Figure 33.

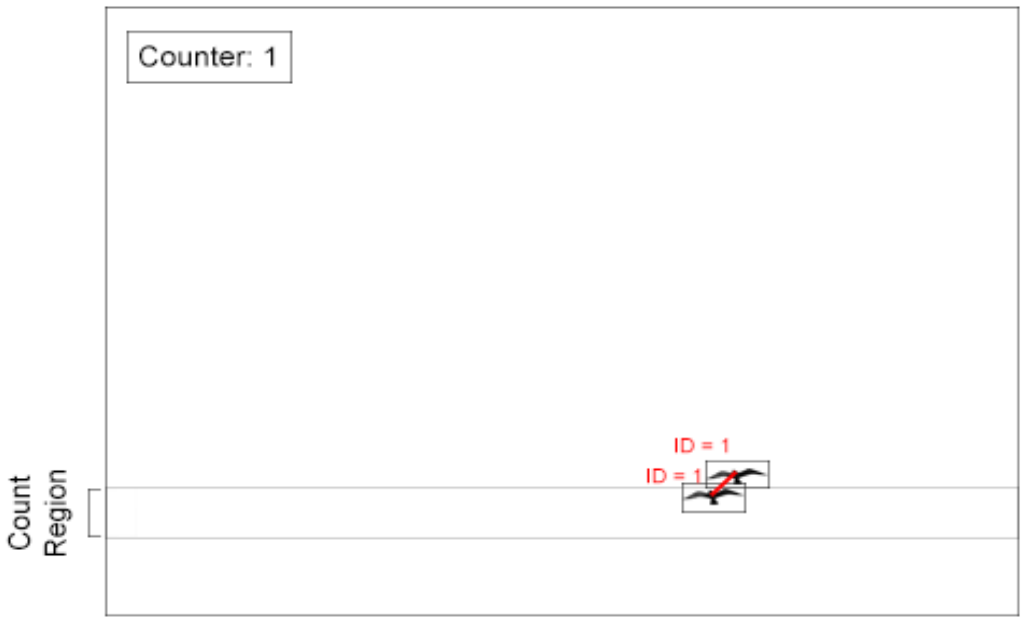


Figure 33: Bird Counting

Chapter 5

Performance Evaluation

5. Performance Evaluation

In this section, we will discuss the performance of both the generic and specific models using various performance evaluation metrics known for such problems. As this is a classification problem, the most important and widely used performance metrics for such problems are precision, recall, F-1 score and accuracy. We will discuss all these metrics in detail in this section and calculate them all to compare various variants of the model to find the best model for this task. In the coming sub-sections, we will discuss what each metric means and how it is useful to determine the performance of the model.

5.1. Confusion matrix and classification report

The confusion matrix [69] is a performance evaluation tool particularly used in classification problems in machine learning and is of utmost importance as it is the basis of almost all of the other important metrics. The rows in the confusion matrix usually represent the predicted classes, while the columns are the actual (true) classes. This gives a tabular representation to the outcomes of the model making it convenient to understand the model performance for a test set. A basic confusion matrix is shown in Figure 34.

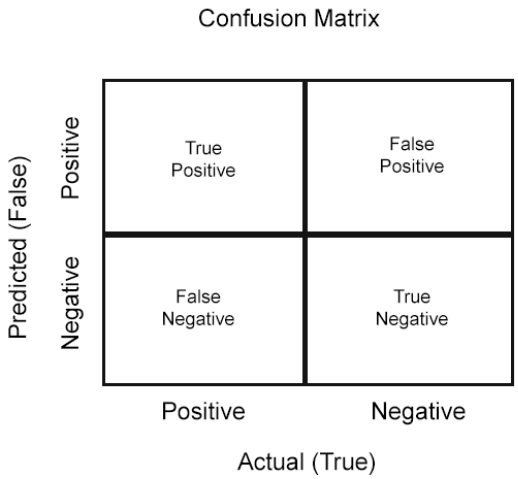


Figure 34: Basic confusion matrix

To understand the confusion matrix, we first need to know what the terms True Positive, False Positive, True Negative, and False Negative mean.

- **True Positive:** actual positive class predicted as positive class.
- **False Positive:** actual positive class predicted as positive class.
- **True Negative:** actual negative class predicted as negative class.
- **False Negative:** actual positive class predicted as negative class.

For multi-class problems, the matrix rows and columns are extended to account for all the class labels in the said problem. The class under consideration is considered as the positive class while all other classes are considered as background or negative classes. We will discuss the performance of all the models we created in the training section using confusion matrix as the values from the confusion matrix help us calculate other metrics. The Figure 35 shows the formulae to calculate these metrics.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Figure 35: Performance metrics formulae

A classification report [70] is a table that represents important classification metrics for individual classes. All these are calculated from the confusion matrix. The classification report that we tend to use for object detection models usually contain the most important metrics like Precision, Recall, Mean Average Precision at 50 IOU (mAP50) and, Mean Average Precision at 50-95 IOU (mAP50-95). We can also calculate metrics like F-score and accuracy and add them to the classification report. Although these metrics are not as important than the ones mentioned before, for the sake of completeness in our experiments we will be calculating the F-score and creating our own classification report.

F1-score is the arithmetic mean of the precision and recall and can prove as an important metric for model evaluation. Accuracy on the other hand, can lead to some confusing results, especially in case of imbalanced datasets. Higher accuracy can be achieved by predicting more results from high density class, but it provides no information on the low density class, hence the precision and recall are always preferred over accuracy.

5.1.1. Default Hyperparameter

For the specific bird species classifier, lets determine the model performance with the default hyperparameter model. We can see in confusion matrix in Figure 36, that all classes are performing fairly good in case of TP, but for royal terns the true positive value is a little

lower. We also see that the false negatives are only in the background class for the royal terns.

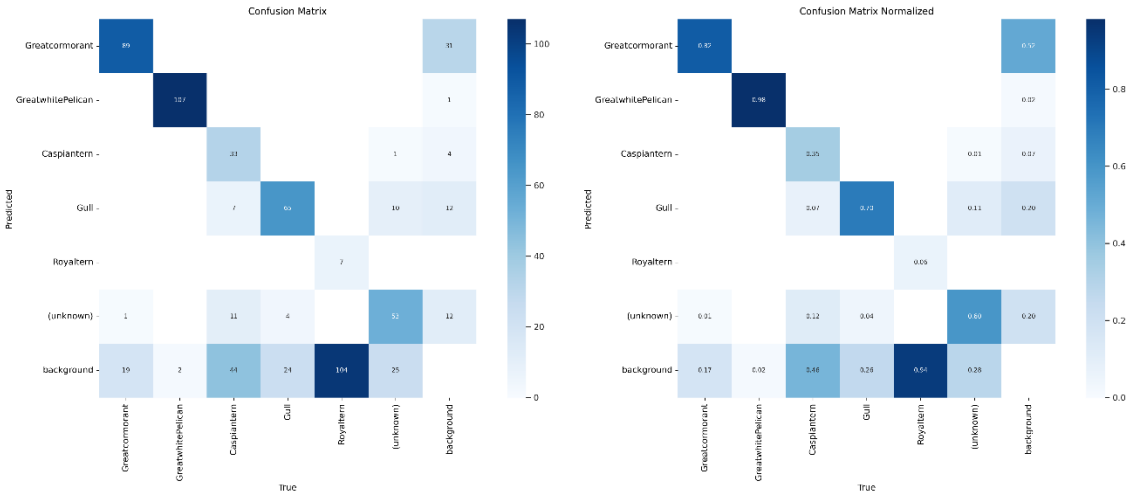


Figure 36: Conf. matrix specific model (Default Hyperparameters)

It means that the model has a hard time detecting this class, but it does not detect it as other classes as we also do not see any false positives for this class. For other five classes, we do see a few false positives and negatives in different places, but the numbers are very low. For this reason, we can say that the overall model performance will be in acceptable ranges, which is backed by the classification report shown in Figure 37.These values can be made better by optimizing the hyperparameters which will be discussed in the next sections.

Class	images	instances	P	R	F1-Score	mAP50	mAP50-95
all	93	606	0.83	0.83	0.83	0.87	0.51
Greatcormorant	93	109	0.83	0.92	0.87	0.92	0.47
GreatwhitePelican	93	109	0.99	1.0	1.0	1.0	0.81
Caspiantern	93	95	0.79	0.73	0.75	0.86	0.45
Gull	93	93	0.72	0.87	0.79	0.79	0.37
Royaltern	93	111	0.89	0.91	0.9	0.96	0.6
(unknown)	93	89	0.73	0.58	0.65	0.69	0.39

Figure 37: Classification report specific model (Default Hyperparameters)

For the generic model for bird classification, the confusion matrix and the classification report are shown in Figure 38, and Figure 39 respectively. It is observable from

the confusion matrix that the model has a high true positive rate and is detecting a lot of birds correctly. The false positive as compared to the true positives are lower in number. This is the reason that in the classification report we have higher value of precision, recall and F1-score. The model's mAP value at 50 IOU is 86.1 which is a good mAP value. We hope to get even better results from tuned models in coming sections.

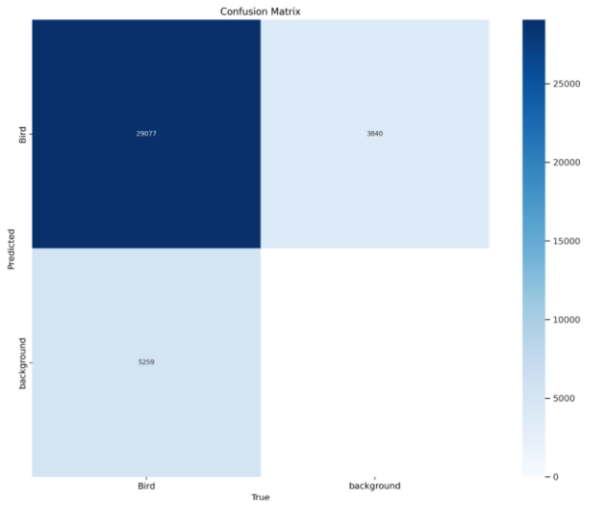


Figure 38: Conf. matrix generic model (Default Hyperparameters)

Class	images	instances	P	R	F1-Score	mAP50	mAP50-95
all	2209	34336	0.854	0.834	0.844	0.861	0.479

Figure 39: Classification report generic model (Default Hyperparameters)

5.1.2. Tuned Hyperparameters (SGD optimizer)

For the model trained on tunned hyperparameters and SGD optimizer, we see a lot of difference overall as compared to the previous confusion matrix. The model has improved for all classes other than the royal tern, which has become worse. The model is showing a conservative behaviour for the royal terns in this model as shown by the confusion matrix in Figure 40.

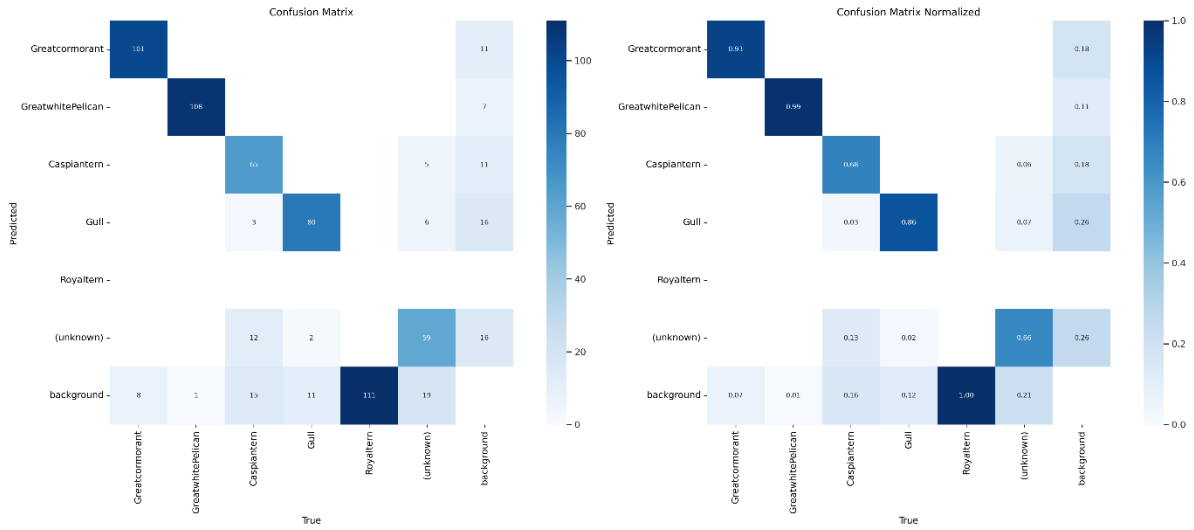


Figure 40: Conf. matrix specific model (SGD)

There are only false negatives detected as backgrounds. This means that the model has a 100% precision but 0% recall. It did not detect the royal terns in any instance. Overall, this model might show good performance for other classes, hence giving good values in the classification report, but it cannot generalize on the royal tern class. This is visible in the classification report shown in Figure 41 as well.

Class	images	instances	P	R	F1-Score	mAP50	mAP50-95
all	93	606	0.87	0.67	0.68	0.86	0.52
Greatcormorant	93	109	0.9	0.91	0.91	0.93	0.46
GreatwhitePelican	93	109	0.96	0.99	0.98	0.99	0.85
Caspiantern	93	95	0.83	0.65	0.73	0.81	0.43
Gull	93	93	0.81	0.8	0.81	0.84	0.39
Royaltern	93	111	1.0	0.0	0.0	0.82	0.53
(unknown)	93	89	0.71	0.65	0.68	0.78	0.46

Figure 41: Classification report specific model (SGD)

If we observe the confusion matrix for the specific model in Figure 42, we can see that the true positives have lowered in this model as compared to the model with default hyperparameters, but this difference is not very large. We can also see this effect in the classification report shown in Figure 43, as the precision and recall have reduced slightly, causing the f1-score to reduce too. Overall, the performance is not bad, but not what we expected it to be.

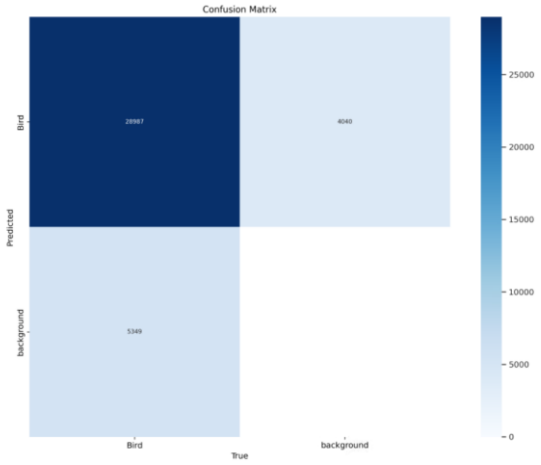


Figure 42: Conf. matrix generic model (SGD)

Class	images	instances	P	R	F1-Score	mAP50	mAP50-95
all	2209	34336	0.852	0.831	0.842	0.859	0.478

Figure 43: Classification report generic model (SGD)

5.1.3. Tuned Hyperparameters (Auto optimizer)

In the next experiment, we tried the same hyperparameters, (lr0 = 0.01, lrf = 0.001, warmup_epochs = 3, warmup_momentum = 0.95), but this time we changed the momentum to auto optimizer, which is actually AdamW optimizer as we trained the model till 300 epochs, with a patience of 50 epochs. The confusion matrices in Figure 44 show this model's performance on the test dataset.

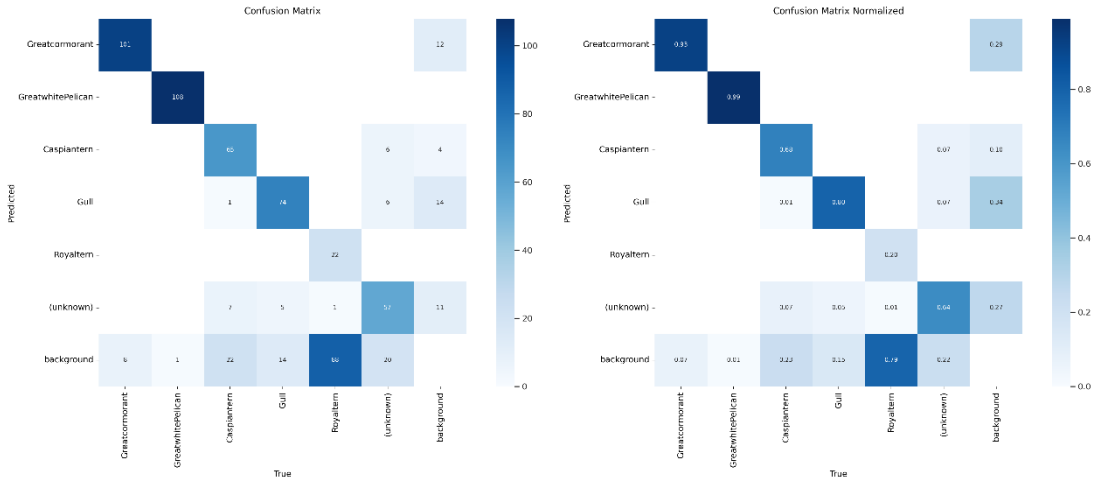


Figure 44: Conf. matrix specific model (Auto)

Comparing it to the previous models, we can see that for this model, we are getting a better result for the royal tern class as well. We can see that there are 22 instances where the model predicted the royal terns correctly. We also see that the model has become better on other class predictions as well. Overall, this is a good result we have seen so far. The classification report in Figure 45 also proves this claim to be true.

Class	images	instances	P	R	F1-Score	mAP50	mAP50-95
all	93	606	0.8	0.84	0.81	0.87	0.51
Greatcormorant	93	109	0.8	0.93	0.86	0.92	0.45
GreatwhitePelican	93	109	0.98	1.0	0.99	1.0	0.82
Caspiantern	93	95	0.8	0.8	0.8	0.87	0.44
Gull	93	93	0.72	0.87	0.79	0.84	0.37
Royaltern	93	111	0.96	0.74	0.83	0.94	0.61
(unknown)	93	89	0.55	0.71	0.62	0.66	0.37

Figure 45: Classification report specific model (Auto)

We have a good precision as well as recall value for this model for all classes standing at 80% and 84% respectively. The values are a little lower for the unknown class. This is because this class is very confusing for the model to learn on. This class contain different birds with different features that we do not know the name of. The F1-score values for all classes are in good ranges and the overall model mAP at 50-95 IOU is 51% which is a good value.

For the generic model with auto optimizer, the confusion matrix in Figure 46, shows that the model performance has increased as the true positive rate has gone higher. This is the best result that we are getting for the generic model as compared to the previous two. This is also observable in the classification report in Figure 47, where the model precision is shown to be 85.6% and the recall is 83.3%. The f1-score, being the mean of these two values is 84.4%. We also an increase in mAP value reaching to 86%.

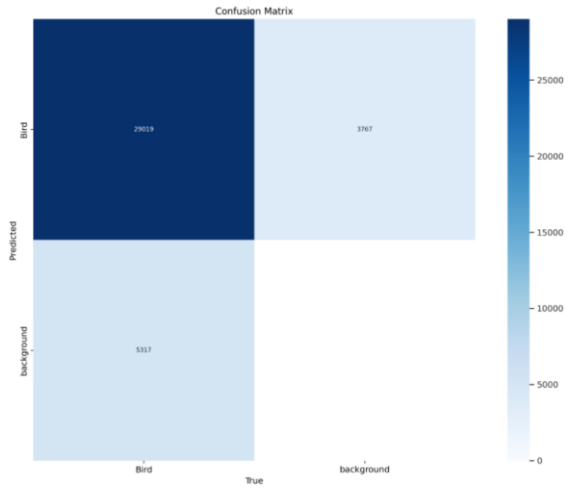


Figure 46: Conf. matrix generic model (auto)

Class	images	instances	P	R	F1-Score	mAP50	mAP50-95
all	2209	34336	0.856	0.833	0.844	0.86	0.478

Figure 47: Classification report generic model (Auto)

5.1.4. Tuned Hyperparameters (Adam optimizer)

The model when trained with an Adam optimizer and the tuned hyperparameters showed that the true positives for the royal terns class increased to 42. There are still false negatives, but the model performance has considerably increased. For other classes, the false negatives are reduced too and almost all detections are true positive. This is visible from the confusion matrices in Figure 48. It is noticeable that the mAP at 50-95 IOU has also increased to 52% for this variant of the model. All other metrics are above 80%. Although the

precision is slightly lower in this model as compared to the default hyperparameter model, the recall in the latest variant is considerably higher than before. The model with Adam optimizer is clearly better at generalizing on unseen data as it is more sensitive than the other models and also has a good precision and f1-score as well. The complete model performance can be observed from the Figure 49.

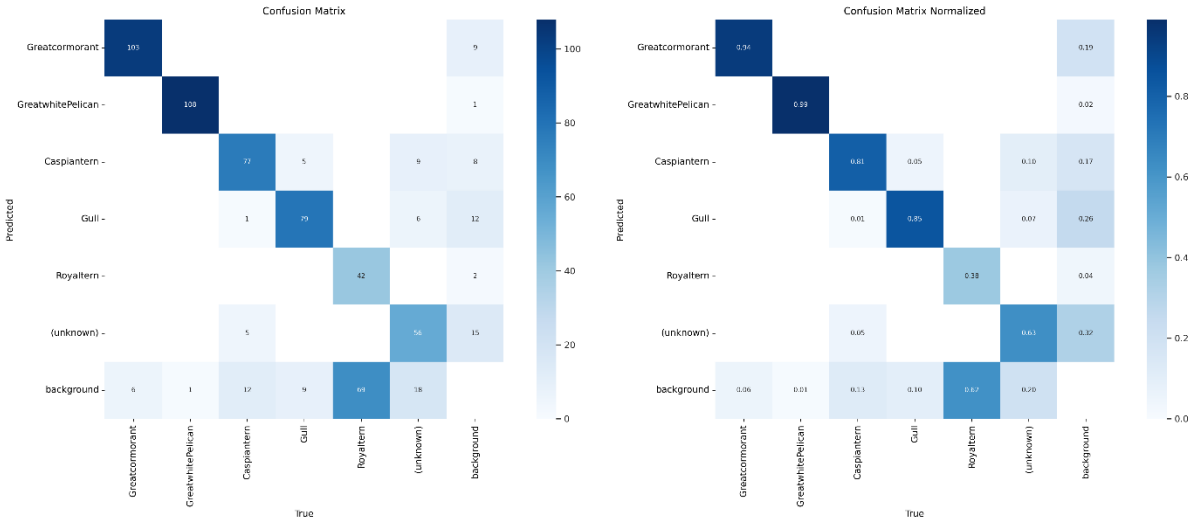


Figure 48: Conf. matrix specific model (Adam)

Class	images	instances	P	R	F1-Score	mAP50	mAP50-95
all	93	606	0.8	0.83	0.81	0.88	0.52
Greatcormorant	93	109	0.83	0.95	0.89	0.94	0.44
GreatwhitePelican	93	109	0.97	0.99	0.98	1.0	0.84
Caspiantern	93	95	0.69	0.83	0.75	0.85	0.44
Gull	93	93	0.7	0.88	0.78	0.84	0.39
Royaltern	93	111	0.92	0.65	0.76	0.9	0.59
(unknown)	93	89	0.67	0.67	0.67	0.75	0.43

Figure 49: Classification report specific model (Adam)

The confusion matrix in Figure 50, and the classification report in Figure 51, show the performance of the generic model with the Adam optimizer. The model's precision is in an acceptable range, but the recall has highly reduced causing the model to perform very bad. This model when compared to others has the worst performance and cannot be used for our task. After watch the performance of the three optimizers, the best model with these hyperparameters is with the auto optimizer.

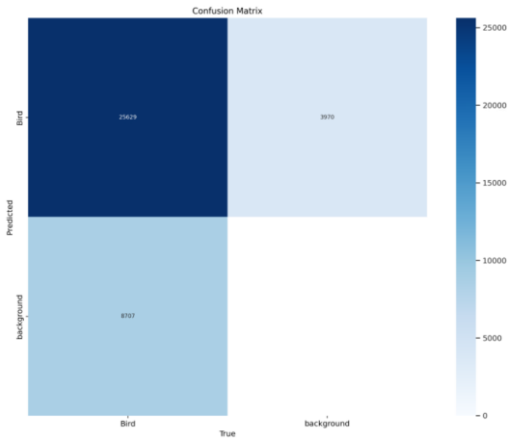


Figure 50: Conf. matrix generic model (Adam)

Class	images	instances	P	R	F1-Score	mAP50	mAP50-95
all	2209	34336	0.84	0.73	0.782	0.799	0.425

Figure 51: Classification report generic model (Adam)

5.2. PR – Curve

A Precision-Recall [71] plot is a graphical depiction of a machine learning model's performance, particularly used for classification problems. It is a tool for assessing the trade-off between recall and precision for various classification thresholds. The precision and recall are the most widely used metrics to test the performance of machine learning models for object detection the PR curve helps in understanding both values at the same time at a given threshold. The PR curve is generated by adjusting the model's classification threshold and computing precision and recall at each threshold. The classification threshold is the number above and below which the model predicts a positive class and a negative class. We can get different recall and precision level by adjusting this threshold. A PR curve that hugs the top-right corner of the figure indicates a model that achieves high precision as well as recall across varied thresholds. A model with poor performance, on the other hand, will have a curve that approaches the bottom-left corner.

The PR plot allows us to evaluate the performance of several models, choose an acceptable threshold based on the application needs, and determine the optimal trade-off between recall and precision for an individual scenario. It is particularly beneficial when dealing with unbalanced datasets in which one class dominates the other, since it gives a more balanced perspective of model performance that goes beyond accuracy alone.

5.2.1. Default Hyperparameter

The classification report and the confusion matrix give us a lot of insight on the model performance. The precision and recall which are the main performance metrics for object detection model are calculated using these, but it takes some understanding of the individual case and these metrics to understand the performance fully. The PR curve makes these things easier as we can estimate the model performance quiet easily just by looking at the PR curve and decide our desired precision and recall values, thus setting a suitable confidence threshold. Figure 52 shows the PR curve for the model we trained initially without any hyperparameter tuning and with just the default hyperparameters.

We can see that the curve for all classes is very close to the top-right corner. The area under the PR curve also helps us decide the model performance. The area under the blue curve is 0.797. The perfect model would have an area of 1. All other curves in the plot are the PR curves for the classes on which the model is trained. The PR curve is usually drawn for a binary classification problem. In multi-class classification problems like this, each class is treated as the main class and at that time, the rest of the classes are treated as background. Thus, the model creates separate PR curve for each class. Ideally all the curves should be the closest to the top right corner. This concept completely the same as the Receiver Operator Classification curve. An ROC curve is plotted in a similar fashion and is used in almost the similar way. The main difference in an ROC plot and PR plot is that the ROC is plotted

between sensitivity, and specificity, at different threshold values. while a PR plot is plotted between precision and recall value at different thresholds.

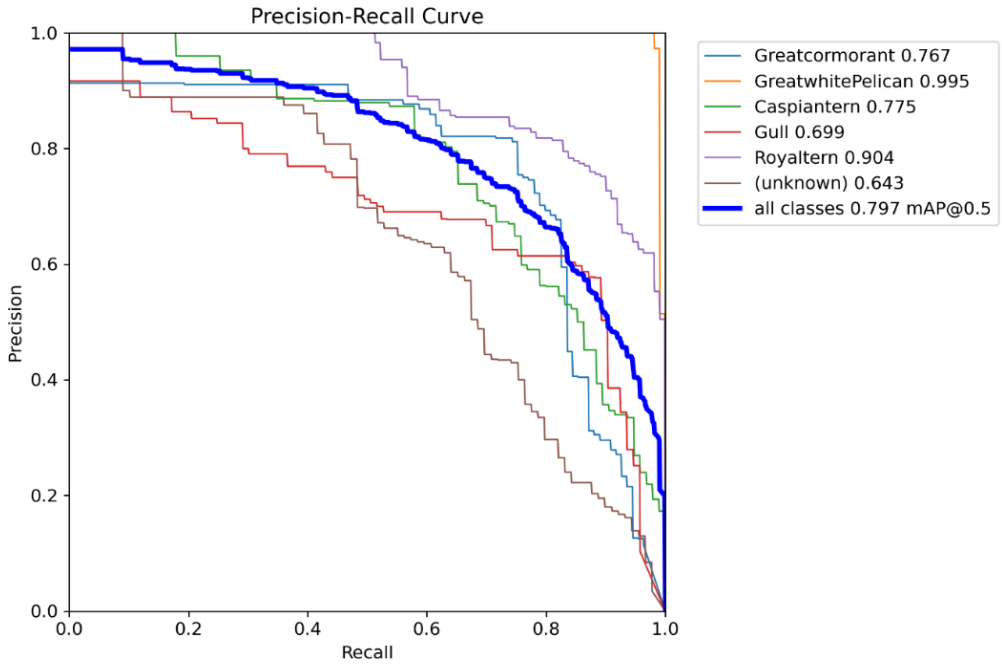


Figure 52: PR curve specific model (default)

5.2.2. Tuned Hyperparameters (SGD optimizer)

The PR plot in Figure 53, shows the performance of the model with tuned hyperparameters with an SGD optimizer. The curve shows at which recall rate, we can get what precision given a specific confidence threshold. This helps us choose between a range of performance metrics that we specifically require for our application. For example, if we need to have a model with high precision, but we don't care much about the recall, we can select a threshold at which the precision is high, in the top left corner for this model. Similarly, if we want a higher recall, but don't care about the precision, we can select a value from the lower left region. For our problem, we are looking for a model that shows a good value for both precision and recall at the same time. The best value for both that we get at the same time, with SGD model is around 80% precision and 80% recall. These values

correspond to a specific threshold, that can be found using the P-confidence, or the R-confidence plots. We will discuss about these and selecting a threshold in the Inference section. The overall area under the curve for this model is 0.863 which is higher than the previous curve (0.7) for the default hyperparameters. We can also observe that the individual class curves have risen in area as well which is another indication that this model is better performing than the previous.

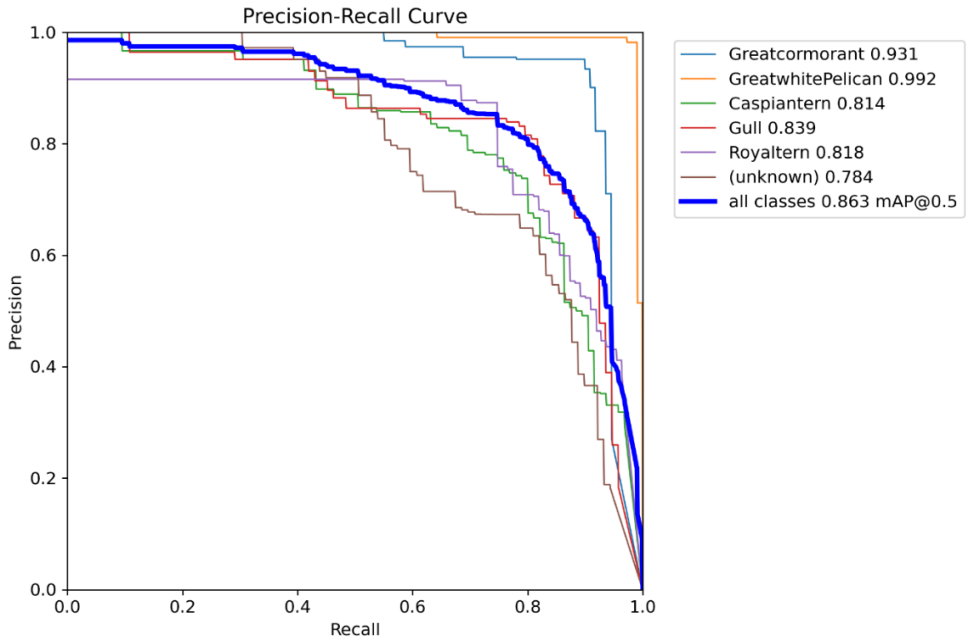


Figure 53: PR curve specific model (SGD)

5.2.3. Tuned Hyperparameters (Auto optimizer)

We case of auto optimizer with tuned hyperparameters, PR plot, represented by Figure 54, shows that the overall model performance has increased. All the curves in plot are closer to the top right corner. The area under the PR curve for all classes has slightly increased 0.863 to 0.869 as well. The auto optimizer has shown better performance than the SGD optimizer. This was also observed in the confusion matrix and the classification report too, where the precision of the SGD model was although better than auto optimizer, it showed significant degradation in recall. This is the reason that in the PR curve, the area of

the auto model is higher than the SGD model. If we look at the individual classes, the Great Cormorants and Great pelicans have improved in performance, but they were already performing good in the previous variant. The noticeable change is in the royal terns. The area for royal tern curve has significantly increased. This is because the recall value for this class is better in the auto optimizer model.

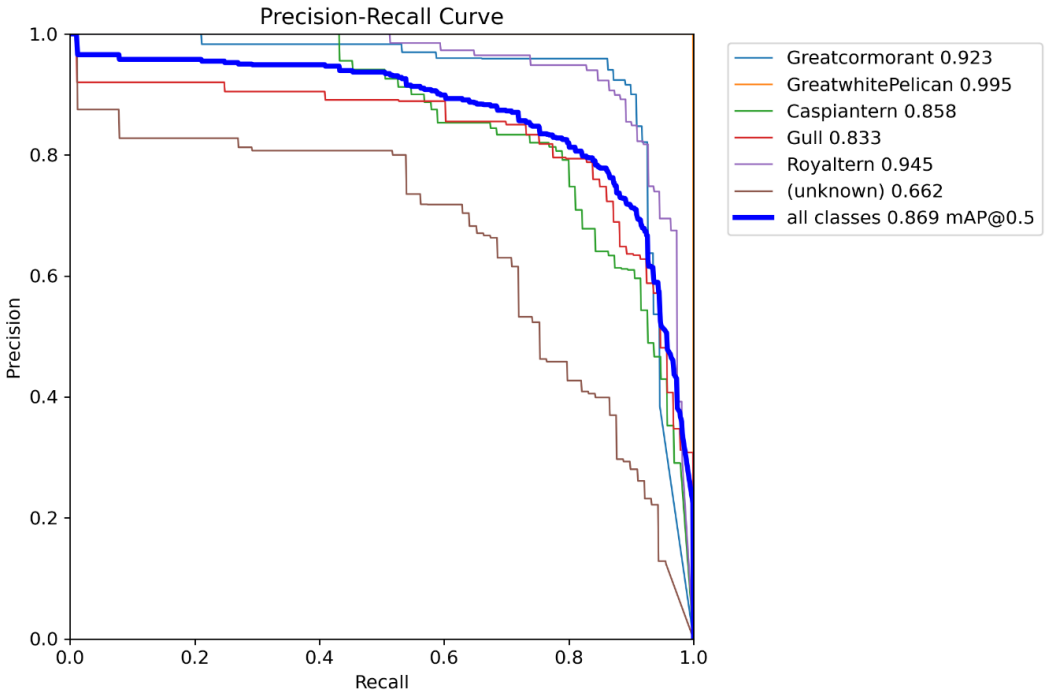


Figure 54: PR curve specific model (Auto)

5.2.4. Tuned Hyperparameters (Adam optimizer)

When we plot the PR curves for the Adam optimized model, we see that model performance of all the individual classes has improved. This is shown by the area under the PR curve for each class. In this variant, all the classes perform highly, including the (unknown) birds class which was showing consistency average results previously. The value has increased from 0.66 to 0.75 for this class which is a significant change. The overall performance, as shown by all classes curve has also increased from 0.869 to 0.878. This model is by far our best performing model which is a fact also backed the confusion matrix

and the classification report discussed earlier. The PR curve for this model is shown in the Figure 55 below.

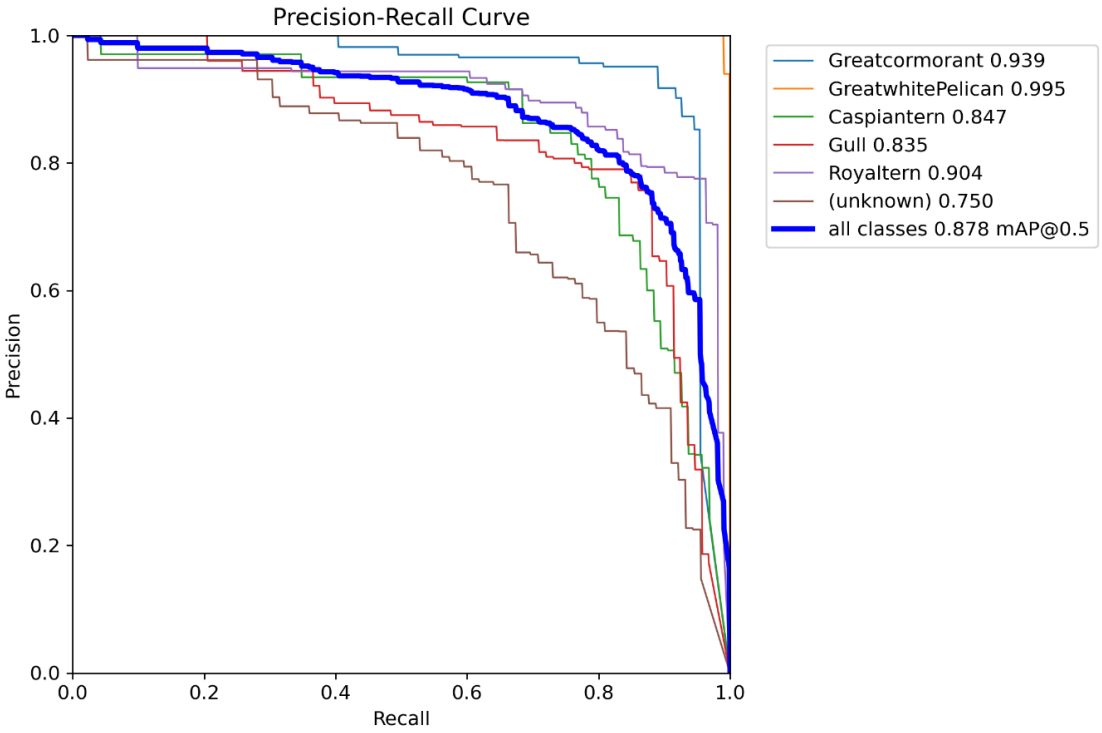


Figure 55: PR curve specific model (Adam)

5.3. Model Comparison

As discussed earlier, the PR curve can be used for three purposes.

- As performance metric of the model.
- To balance between precision and recall selecting a confidence threshold.
- To compare multiple models

In this section, we will be comparing the tree models we created by keeping the hyperparameters same but changing the optimizer. The models are shown by blue, red and purple colour in the plot shown in Figure 56. The Blue plot corresponds to Auto optimizer, Red colour to Adam and Purple colour to SGD optimizer. The dotted plot is the best possible result that can be ideally, as the area under the plot will be equal to 1. We compare our three graphs to the best possible result and decide which one is closer to it. All three models gave us similar results. But the maximum value that we got from our experiments for area under the PR curve is 0.878 obtained using the Adam optimizer, and the tuned hyper parameters, (lr0 = 0.01, lrf = 0.001, warmup_epochs = 3, warmup_momentum = 0.95).

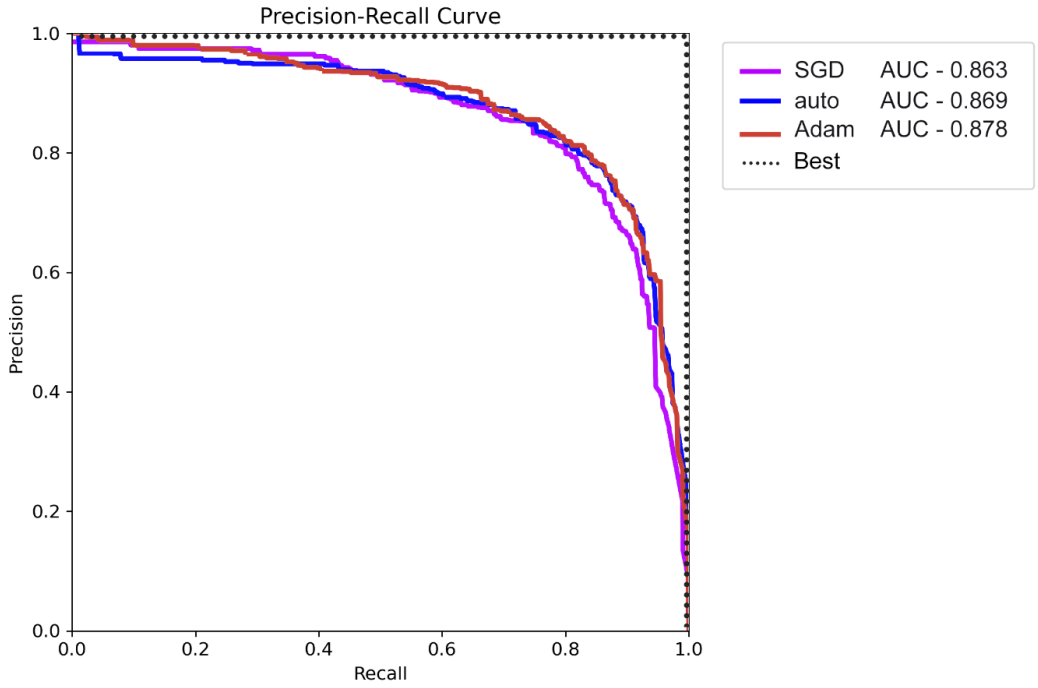


Figure 56: Model comparison

For the generic model the PR-curve when trained with an SGD optimizer as shown in Figure 57, has an area of 0.859, while for the model with Adam optimizer has an area of 0.799 as shown in Figure 58. The best model as found previously for generic bird detection was the one with auto optimizer and the area under the PR curve for this model as shown by Figure 59 is 0.861 which is higher than the others showing that our previous deduction was correct.

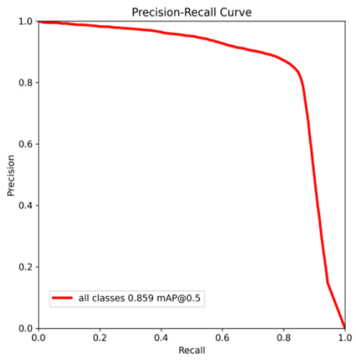


Figure 57: PR curve Generic model (SGD)

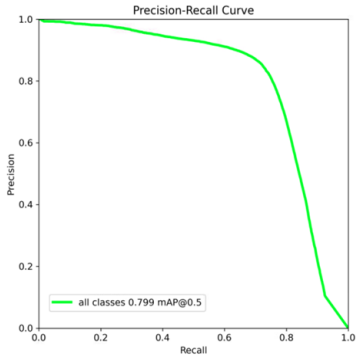


Figure 58: PR curve Generic model (Adam)

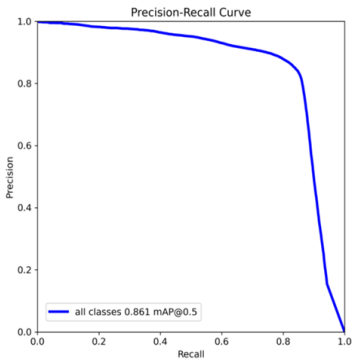


Figure 59: PR curve Generic model (Auto)

Chapter 6

Inference

6. Inference

Inference in machine learning, is the process of using a trained model to generate predictions on new, and unseen data. In the context of object detection models, inference is critical to the whole workflow. Once trained on a dataset, a model's primary goal is to be utilised in real-world applications to recognise objects in photos or videos. Inference is the stage in which the model exhibits its utility and makes predictions based on previously unknown data. Inference is also important because it influences how quickly a model can digest new data and make prediction so that we can decide if the model is capable of delivering a good value of Frames per second to be used in our task. After training and evaluating our model, we need to see the inference results as well both on images and video to visually see its performance on the real world data.

6.1. Inference on images

For specific bird species classification model, we tested the model on several images of the 6 classes of the birds it was trained on and to see how good it performs. An important factor to consider here is the confidence threshold. The confidence threshold is the confidence value above which the model considers a detection as the positive class while below this threshold it is considered as the background or the negative class. As it is a probability, the value is between 0.0 to 1.0. Depending on the precision and recall value we expect from our model, we can select a threshold. Fr this task, we want the model to have as high precision and recall possible. For the model with the Adam optimizer, the plot for precision-Recall is shown in Figure 60. In this plot, we can see that the highest precision and recall that we can get at the same time is roughly around 82% precision and 80% recall. If we consider the Precision - Confidence plot in Figure 61, we can see that 82% precision is at a confidence value of 0.20. If we see the value of recall from the Recall – Confidence plot in

Figure 62 at 0.20 confidence threshold value, we can confirm that our assumption was correct. At 0.20 threshold, we get 82% precise and 80% sensitive result for the specific bird classification model.

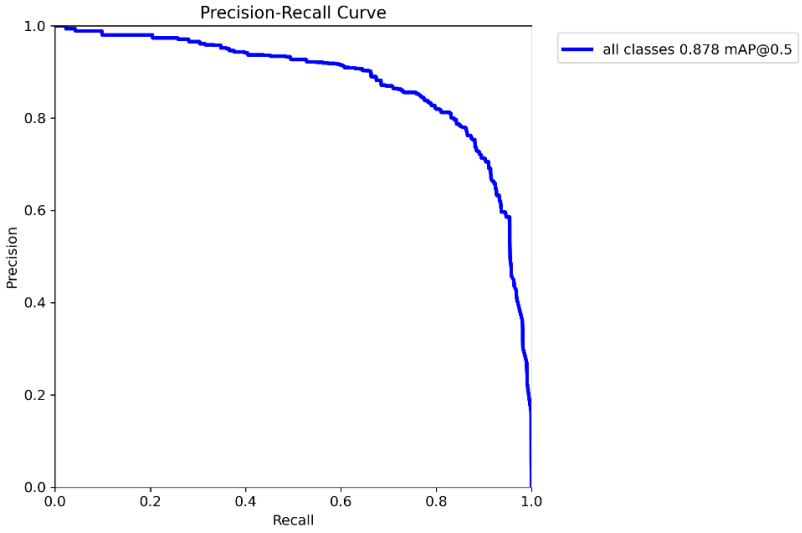


Figure 60: PR - Curve

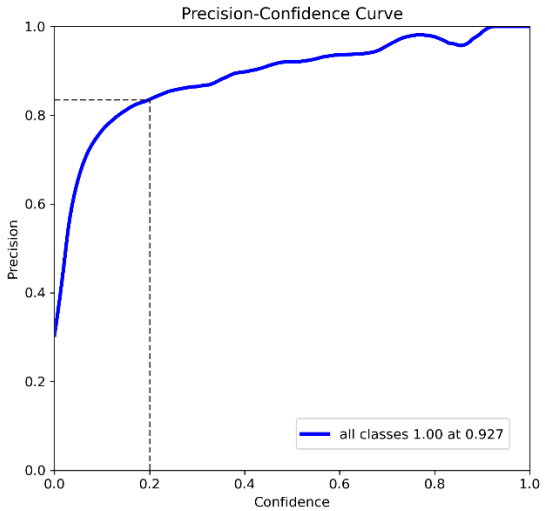


Figure 61: P - Curve

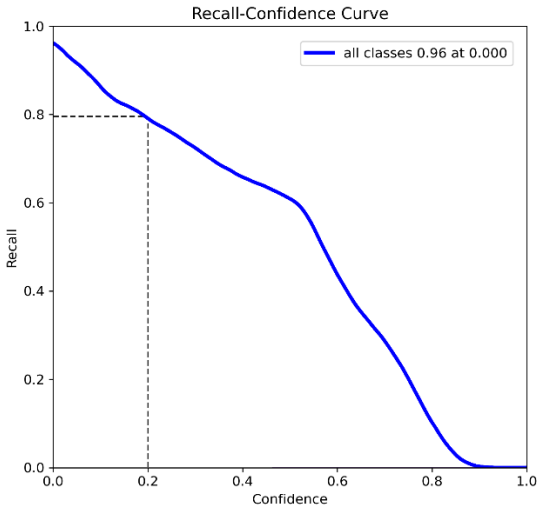


Figure 62: R - Curve

The model is used to inference on various images as shown in Figure 63 (and Figure 64), and the results are observed. The model is detecting almost all instances which complement the fact that the precision of the model is almost 82% at a threshold value of 0.20.

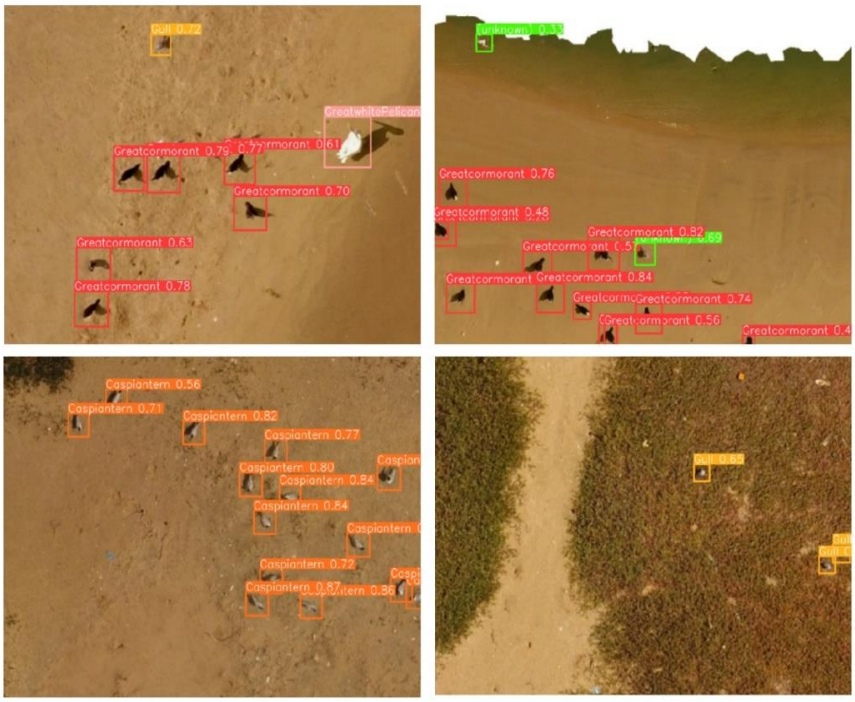


Figure 63:Inference on images using specific model

Figure 64: Inference on images using generic model

As mentioned previously, the inference speed is considered as an important parameter to judge the model performance that are supposed to be used in real time situations. The

inference time taken by the model helps in understanding if the model is capable of working in real-time applications or not. On testing, the models show an average inference time of 8.4 ms, which is a good speed.

6.2. Inference on video

The inference on the images showed good result for the model. In this section, we will try the model on a video to inference on each frame. We will also utilize the models capability to track each object in individual frames. This is done by assigning ids to each detection so that the model can track it. Using these ids, we count the birds in a specified region of the frame. This way we can show the total count of birds passing through this region. Figure 65, shows the inference done on a frame of video made from the orthomosaic of seabirds dataset.

Figure 65: Inference on video

Chapter 7

Conclusion

7. Conclusion

In this research, we attempted to address the challenge of bird classification using object detection models. The main purpose of this study is to create and test several object detection models for precise and efficient bird recognition. To do this, we investigated alternative architectures, hyperparameters, and optimizers in order to determine the best-performing model for this particular job. We used a systematic approach throughout the research, which included data preparation, hyperparameter tuning, model training, evaluation, and inference.

The initial stage of our research entailed developing two types of models: one for specific bird species classification and the second for generic birds classification. The specific birds classification model is intended to recognise six distinct bird species, whereas the generic model is intended to recognise only one class, Birds, from UAV images.

For the specific bird species detection model, we collected a diverse dataset containing images of six different bird species taken using UAVs and split it into training and testing sets. Leveraging transfer learning, we fine-tuned pre-trained models to recognize these bird species effectively. Similarly, for the generic bird classification model, we collected 13 different birds image datasets taken from different heights, geographic locations and with different backgrounds. This was done to increase the diversity of the data so that the trained model would be able to generalize better on the unseen data.

For both models, we experimented with different hyperparameters and optimizers. We compared the performance of the models using various evaluation metrics such as precision, recall, and F1-score. We did not spend time on metrics such as accuracy as these metrices do not provide a good representation of the model performance in such applications. This is the

reason why mAP and AR are used widely for model evaluation in object detection. Additionally, we visualized the model performance using confusion matrices and PR curves.

Our findings revealed that the model with Adam optimizer and tuned hyperparameters achieved the best overall performance for the specific birds species classification. This model exhibited high precision and recall values, making it well-suited for real-world applications. It outperformed the models trained with SGD and auto optimizers in terms of precision-recall trade-offs. However, the generic birds classification model showed best results with the tuned hyperparameters and the auto optimizer.

The model's skills were further confirmed through inference on unseen images. We confirmed that the model produced the appropriate recall and precision levels by defining a confidence threshold using the PR and individual P - confidence and R - confidence curves. The results revealed the ability of the model to detect and classify bird and various species in a variety of images, even when using previously unseen data.

The development of efficient object detection models has far-reaching implications for daily use. These models can be used in a variety of conservation and wildlife monitoring projects. Researchers may collect data at scale by automating the process of bird species identification, adding to a better knowledge of bird populations and migration patterns. However, there are certain areas that might be improved and should be looked at further. The model's performance on the "royal terns" class was rather weak, indicating that identifying this species may be difficult. By addressing these shortcomings, we may be able to develop even more robust and trustworthy object detection models for bird categorization.

Overall, this study gives useful insights into the use of object detection models for bird and species categorization and sets the groundwork for future advances in this field. We

can make great achievements in animal conservation and contribute to a more sustainable and biodiverse planet by using the endless possibilities of artificial intelligence

Chapter 8

Future Recommendations

8. Future Recommendations

In this research, we created models that can detect different types of birds, both general and particular species, as well as count the number of birds in a given region. We discovered that the models worked effectively after evaluating them. However, our effort does not end here. We live in a constantly evolving era of computer vision and AI, where new algorithms and architectures frequently surpass their predecessors. As a result, we propose investigating the usage of newer and more sophisticated architectures to potentially increase the models' performance. Additionally, adopting ensemble approaches, which combine the characteristics of numerous models for increased accuracy, is worth considering.

One approach to improving model performance is by increasing the amount of training data. We can anticipate higher performance if we train the models on a bigger dataset, concentrating on classes such as royal terns and the unknown class. To do this, conducting surveys in regions with known bird populations and using drones to take additional photos for augmentation could prove effective. Using generative adversarial neural networks is another method for expanding the dataset. Using current photos and the capabilities of generative AI, we can use this method to produce new images that the model hasn't seen before.

The process of AI model development, however, does not end with training. To effectively evaluate a model's capabilities, it must be used in realistic circumstances. We advocate installing the model on edge computing devices to do this. These gadgets may be used with drones to deliver predictions in real time while simultaneously filming video feeds. This configuration also allows for long-term monitoring studies in which the model may constantly calculate bird counts over time. Conservationists will benefit greatly from this

innovation since it reduces the need for repetitive surveys and adds automation to their conservation efforts.

Thus, the work described in this thesis provides a solid foundation for the development of object detection models in the categorization of bird species. Researchers may build on this basis to produce more accurate, efficient, and adaptable models that contribute to a better knowledge of bird ecosystems and help global conservation efforts by addressing the future suggestions indicated above.

References

[1]. R. Roslan, N. A. Nazery, N. Jamil, and R. Hamzah, 'Color-based bird image classification using Support Vector Machine', in *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, 2017, pp. 1–5.

[2]. P. J. Negret, M. Maron, R. A. Fuller, H. P. Possingham, J. E. M. Watson, and J. S. Simmonds, "Deforestation and bird habitat loss in Colombia," *Biological Conservation*, p. 109044, Mar. 2021, doi: https://doi.org/10.1016/j.biocon.2021.109044.

[3]. H. Q. P. Crick, "The impact of climate change on birds," *Ibis*, vol. 146, no. 1, pp. 48–56, Sep. 2004, doi: https://doi.org/10.1111/j.1474-919x.2004.00327.x.

[4]. D. Yang, A. Yang, J. Yang, R. Xu, and H. Qiu, "Unprecedented Migratory Bird Die-Off: A Citizen-Based Analysis on the Spatiotemporal Patterns of Mass Mortality Events in the Western United States," *GeoHealth*, vol. 5, no. 4, Apr. 2021, doi: https://doi.org/10.1029/2021gh000395.

[5]. *State of the Birds 2019*, Sep. 20, 2019. [online] https://www.stateofthebirds.org/2019/download-pdf-report/

[6]. J. M. Hoekstra, T. M. Boucher, T. H. Ricketts, and C. Roberts, "Confronting a biome crisis: global disparities of habitat loss and protection," *Ecology Letters*, vol. 8, no. 1, pp. 23–29, Dec. 2004, doi: https://doi.org/10.1111/j.1461-0248.2004.00686.x.

[7]. J. Serrano, J. E. Richardson, T. D. Pennington, R. Cortes-B, D. Cardenas, A. Elliot, I. Jimenez, "Biotic homogeneity of putative biogeographic units in the Neotropics: A test with Sapotaceae," vol. 24, no. 8, pp. 1121–1135, Aug. 2018, doi: https://doi.org/10.1111/ddi.12752.

[8]. S. Zhao, 'Bird Movement Recognition Research Based on YOLOv4 Model', in 2022 4th International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM), 2022, pp. 441–444.

[9]. S. Rath, S. Kumar, V. S. K. Guntupalli, S. M. Sourabh, and S. Riyaz, 'Analysis of Deep Learning Methods for Detection of Bird Species', in *2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, 2022, pp. 234–239.

[10]. C. Sekercioglu, 'Ecological Significance of Bird Populations', vol. 11, 01 2006, pp. 15–51.

[11]. S. Naeem, J. E. Duffy, and E. Zavaleta, "The Functions of Biological Diversity in an Age of Extinction," *Science*, vol. 336, no. 6087, pp. 1401–1406, Jun. 2012, doi: https://doi.org/10.1126/science.1215855.

[12]. R. W. Furness and Jeremy, *Birds as Monitors of Environmental Change*. Springer Nature, 1993. doi: https://doi.org/10.1007/978-94-015-1322-7.

[13]. R. D. Gregory *et al.*, "Developing indicators for European birds," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 360, no. 1454, pp. 269–288, Feb. 2005, doi: https://doi.org/10.1098/rstb.2004.1602.

[14]. J. W. Pearce-Higgins and R. E. Green, *Birds and Climate Change: Impacts and Conservation Responses*. Cambridge: Cambridge University Press, 2014. Accessed: May 31, 2023.

[15]. E. K. Haas, F. A. La, H. M. McCaslin, M. C. Belotti, and K. G. Horton, "The correlation between eBird community science and weather surveillance radar-based estimates of migration phenology," vol. 31, no. 11, pp. 2219–2230, Jul. 2022, doi: https://doi.org/10.1111/geb.13567.

[16]. A. O. Panjabi, P. J. Blancher, W.E. Easton, J.C. Stanton D. W. Demarest, R. Dettmers, and K. V. Rosenberg, *The Partners in Flight Handbook on Species Assessment*. Partners in Flight Technical Series No. 3. Version 2017. Bird Conservancy of the Rockies.

[17]. C. J. Bibby, N. D. Burgess, D. M. Hillis, D. A. Hill, and S. Mustoe, *Bird census techniques*. Elsevier, 2000.

[18]. R. L. Boring and A. Bye, "Bridging Human Factors and Human Reliability Analysis," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 52, no. 11, pp. 733–737, Sep. 2008, doi: https://doi.org/10.1177/154193120805201108.

[19]. M. Sethu, N. A. Titu, M. Madadi, J. Coble, R. Boring, K. Blache, V. Agarwal, V. Yadav, A. Khojandi, *Using Artificial Intelligence To Mitigate Human Factor Errors In Nuclear Power Plants: A Review*. Apr. 2021, doi: https://doi.org/10.13140/RG.2.2.17556.83849.

[20]. X. Zhang, V. Mehta, M. Bolic, and I. Mantegh, 'Hybrid AI-enabled Method for UAS and Bird Detection and Classification', in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020, pp. 2803–2807.

[21]. R. Mohanty, B. K. Mallik, and S. S. Solanki, "Automatic bird species recognition system using neural network based on spike," *Applied Acoustics*, vol. 161, p. 107177, Apr. 2020, doi: https://doi.org/10.1016/j.apacoust.2019.107177.

[22]. P. Eichinski, C. Alexander, P. Roe, S. Parsons, and S. Fuller, "A convolutional neural network bird species recognizer built from little data by iteratively training, detecting, and

labeling," Frontiers, [online] https://www.frontiersin.org/articles/10.3389/fevo.2022.810330/full

[23]. D. Stowell, M. D. Wood, H. Pamuła, Y. Stylianou, and H. Glotin, "Automatic acoustic detection of birds through deep learning: The first Bird Audio Detection challenge," *Methods in Ecology and Evolution*, vol. 10, no. 3, pp. 368–380, Nov. 2018, doi: https://doi.org/10.1111/2041-210x.13103.

[24]. H. G. Akçay, B. Kabasakal, D. Aksu, N. Demir, M. Öz, and A. Erdoğan, "Automated Bird Counting with Deep Learning for Regional Bird Distribution Mapping," *Animals*, vol. 10, no. 7, p. 1207, Jul. 2020, doi: https://doi.org/10.3390/ani10071207.

[25]. K. J. Spiller and R. Dettmers, "Evidence for multiple drivers of aerial insectivore declines in North America," *The Condor*, vol. 121, no. 2, May 2019, doi: https://doi.org/10.1093/condor/duz010.

[26]. H.-M. Tu, M.-W. Fan, and J. C.-J. Ko, "Different Habitat Types Affect Bird Richness and Evenness," *Scientific Reports*, vol. 10, no. 1, pp. 1–10, Jan. 2020, doi: https://doi.org/10.1038/s41598-020-58202-4.

[27]. J. Lowe, "How Many Birds Are Killed by Wind Turbines?" American Bird Conservancy, Jan. 26, 2021. https://abcbirds.org/blog21/wind-turbine-mortality.

[28]. C. Both, S. Bouwhuis, C. M. Lessells, and M. E. Visser, "Climate change and population declines in a long-distance migratory bird," *Nature*, vol. 441, no. 7089, pp. 81–83, May 2006, doi: https://doi.org/10.1038/nature04539.

[29]. D. T. Blumstein *et al.*, "Acoustic monitoring in terrestrial environments using microphone arrays: applications, technological considerations and prospectus," *Journal of Applied Ecology*, vol. 48, no. 3, pp. 758–767, Apr. 2011, doi: https://doi.org/10.1111/j.1365-2664.2011.01993.x.

[30]. A. M. Dokter, F. Liechti, H. H. Stark, Laurent Delobbe, P. Tabary, and I. Holleman, "Bird migration flight altitudes studied by a network of operational weather radars," vol. 8, no. 54, pp. 30–43, Jan. 2011, doi: https://doi.org/10.1098/rsif.2010.0116.

[31]. T. Y. Lin *et al.*, "MistNet: Measuring historical bird migration in the US using archived weather radar data and convolutional neural networks," *Methods Ecol Evol*, vol. 10, no. 11, pp. 1908–1922, Nov. 2019, doi: 10.1111/2041-210X.13280.

[32]. F. Goldsmith and W. Sutherland, 'Ecological Census Techniques. A Handbook', *The Journal of Ecology*, vol. 85, p. 107, 02 1997.

[33]. P. Walsh, D. Halley, M. Harris, A. Nevo, I. Sim, and M. Tasker, “Seabird monitoring handbook for Britain and Ireland A compilation of methods for survey and monitoring of breeding seabirds.”

[34]. C. J. Bibby, Ecoscope Applied Ecologists, British Trust For Ornithology, R. Society, and Birdlife International, *Bird census techniques*. London; San Diego: Academic Press, 2000.

[35]. P. Viola Michael Jones, “SECOND INTERNATIONAL WORKSHOP ON STATISTICAL AND COMPUTATIONAL THEORIES OF Robust Real-time Object Detection,” 2001.

[36]. N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection.”

[37]. Y. Ramadevi, T. Sridevi, B. Poornima, and B. Kalyani, “Segmentation And Object Recognition Using Edge Detection Techniques,” *International Journal of Computer Science and Information Technology*, vol. 2, no. 6, pp. 153–161, Dec. 2010, doi: 10.5121/ijcsit.2010.2614.

[38]. M. Saraswat, A. K. Goswami, and A. Tiwari, “Object Recognition Using Texture Based Analysis.”

[39]. V. K. Vatsavayi and N. Andavarapu, “Identification and classification of wild animals from video sequences using hybrid deep residual convolutional neural network,” *Multimedia Tools and Applications*, Apr. 2022, doi: https://doi.org/10.1007/s11042-022-12852-w.

[40]. Y. Munian, A. Martinez-Molina, D. Miserlis, H. Hernandez, and M. Alamaniotis, “Intelligent System Utilizing HOG and CNN for Thermal Image-Based Detection of Wild Animals in Nocturnal Periods for Vehicle Safety,” *Applied Artificial Intelligence*, pp. 1–29, Feb. 2022, doi: https://doi.org/10.1080/08839514.2022.2031825.

[41]. D. Sato, A. J. Zanella, and E. X. Costa, “Computational classification of animals for a highway detection system,” *Brazilian Journal of Veterinary Research and Animal Science*, vol. 58, pp. e174951–e174951, May 2021, doi: https://doi.org/10.11606/issn.1678-4456.bjvras.2021.174951.

[42]. D. Jaswal, S. V, and K. P. Soman, “Image Classification Using Convolutional Neural Networks,” Int. J. Sci. Eng. Res., vol. 5, no. 6, pp. 1661–1668, 2014, doi: 10.14299/ijser.2014.06.002

[43]. H. Shetty, H. Singh, and F. Shaikh, “Animal Detection using Deep Learning,” vol. 11, no. 06, pp. 28059–28061, 2021

[44]. H. K. Kondaveeti, P. Nithiyasri, B. S. L. Sri, K. H. Jessica, S. V. S. Kumar, and S. C. Gopi, “Bird Species Recognition using Deep Learning,” *IEEE Xplore*, Mar. 01, 2023. https://ieeexplore.ieee.org/document/10134804

[45]. S. Rath, S. Kumar, V. S. K. Guntupalli, S. M. Sourabh, and S. Riyaz, “Analysis of Deep Learning Methods for Detection of Bird Species,” *IEEE Xplore*, Feb. 01, 2022. https://ieeexplore.ieee.org/document/9742798

[46]. H. K. Kondaveeti, K. S. Sanjay, K. Shyam, R. Aniruth, S. C. Gopi, and S. V. S. Kumar, “Transfer Learning for Bird Species Identification,” *IEEE Xplore*, Mar. 01, 2023. https://ieeexplore.ieee.org/document/10142979 (accessed Jul. 28, 2023).

[47]. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” *arXiv.org*, 2018. https://arxiv.org/abs/1801.04381

[48]. P. Gavali and J. Saira. Banu, “Bird Species Identification using Deep Learning on GPU platform,” *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, Feb. 2020, doi: https://doi.org/10.1109/ic-etite47903.2020.85.

[49]. L. B. Boudaoud, F. Maussang, R. Garello, and A. Chevallier, “Marine Bird Detection Based on Deep Learning using High-Resolution Aerial Images,” *OCEANS 2019 - Marseille*, Jun. 2019, doi: https://doi.org/10.1109/oceanse.2019.8867242.

[50]. B. Kellenberger, T. Veen, E. Folmer, and D. Tuia, “21 000 birds in 4.5 h: efficient large-scale seabird detection with machine learning,” *Remote Sensing in Ecology and Conservation*, vol. 7, no. 3, pp. 445–460, Mar. 2021, doi: https://doi.org/10.1002/rse2.200.

[51]. W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” *Computer Vision – ECCV 2016*, pp. 21–37, 2016, doi: https://doi.org/10.1007/978-3-319-46448-0_2.

[52]. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv.org*, Apr. 10, 2015. https://arxiv.org/abs/1409.1556

[53]. T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” *arXiv:1708.02002 [cs]*, Feb. 2018, Available: https://arxiv.org/abs/1708.02002

[54]. L. Tan, T. Huangfu, L. Wu, and W. Chen, “Comparison of YOLO v3, Faster R-CNN, and SSD for Real-Time Pill Identification,” *Comparison of YOLO v3, Faster R-CNN, and SSD for Real-Time Pill Identication*, Jul. 2021, doi: https://doi.org/10.21203/rs.3.rs-668895/v1.

[55]. S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *arXiv.org*, 2015. https://arxiv.org/abs/1506.01497

[56]. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” 2016. Available: https://arxiv.org/pdf/1506.02640v5.pdf

[57]. M. Chablani, “YOLO — You only look once, real time object detection explained,” *Medium*, Aug. 31, 2017. https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006

[58]. Ultralytics, “YOLOv8,” *docs.ultralytics.com*. https://docs.ultralytics.com/models/yolov8/#supported-modes (accessed Jul. 28, 2023).

[59]. Ultralytics, “YOLOv8,” *docs.ultralytics.com*. https://docs.ultralytics.com/models/yolov8/#supported-modes (accessed Jul. 28, 2023).

[60]. Weinstein, B.G., Garner, L., Saccomanno, V.R., Steinkraus, A., Ortega, A., Brush, K., Yenni, G., McKellar, A.E., Converse, R., Lippitt, C.D., Wegmann, A., Holmes, N.D., Edney, A.J., Hart, T., Jessopp, M.J., Clarke, R.H., Marchowski, D., Senyondo, H., Dotson, R. and White, E.P. (2022). A general deep learning model for bird detection in high resolution airborne imagery. *bioRxiv*, [online] p.2021.08.05.455311. doi:https://doi.org/10.1101/2021.08.05.455311.

[61]. Kellenberger, B., Veen, T., Folmer, E. and Tuia, D. (2021). 21 000 birds in 4.5 h: efficient large-scale seabird detection with machine learning. *Remote Sensing in Ecology and Conservation*, 7(3), pp.445–460. doi:https://doi.org/10.1002/rse2.200.

[62]. Adobe (n.d.). *Buy Adobe Photoshop | Best photo, image, and design editing software*. [online] www.adobe.com. Available at: https://www.adobe.com/uk/products/photoshop.html.

[63]. Jung, A. (2020). *aleju/imgaug*. [online] GitHub. Available at: https://github.com/aleju/imgaug.

[64]. PyTorch (2023). *PyTorch*. [online] Pytorch.org. Available at: https://pytorch.org/.

[65]. Ultralytics (n.d.). *Configuration*. [online] docs.ultralytics.com. Available at: https://docs.ultralytics.com/usage/cfg/ [Accessed 17 Aug. 2023].

[66]. Ultralytics (n.d.). *Ray Tune*. [online] docs.ultralytics.com. Available at: https://docs.ultralytics.com/integrations/ray-tune/#default-search-space-description [Accessed 17 Aug. 2023].

[67]. C. Montella, 'The Kalman Filter and Related Algorithms: A Literature Review', 05 2011.

[68]. Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W. and Wang, X. (2022). ByteTrack: Multi-Object Tracking by Associating Every Detection Box. *arXiv:2110.06864 [cs]*. [online] Available at: https://arxiv.org/abs/2110.06864.

[69]. scikit-learn (2019). *sklearn.metrics.confusion_matrix — scikit-learn 0.21.3 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

[70]. scikit-learn.org. (n.d.). *3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 0.23.2 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/model_evaluation.html#classification-report.

[71]. scikit-learn. (n.d.). *sklearn.metrics.precision_recall_curve*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_curve.html#sklearn.metrics.precision_recall_curve [Accessed 17 Aug. 2023].

Appendices

Appendix A: Data Acquisition Notebook

In-flight Birds detection, species detection, and counting

Data Acquisition

In [1]:
```
# Import libraries
import cv2
import matplotlib.pyplot as plt
import csv
import os
import pandas as pd
import xml.etree.ElementTree as ET
```

Fixing coordinates:

In [2]:
```
# Ortho mosaic size check
def get_image_size(file_path):
    image = cv2.imread(file_path, cv2.IMREAD_UNCHANGED)
    height, width = image.shape[:2]
    return width, height

tif_file_path = "sea_birds.tif"
image_width, image_height = get_image_size(tif_file_path)

print("Image size:")
print("Width:", image_width, "pixels")
print("Height:", image_height, "pixels")
```

```
Image size:
Width: 27569 pixels
Height: 28814 pixels
```

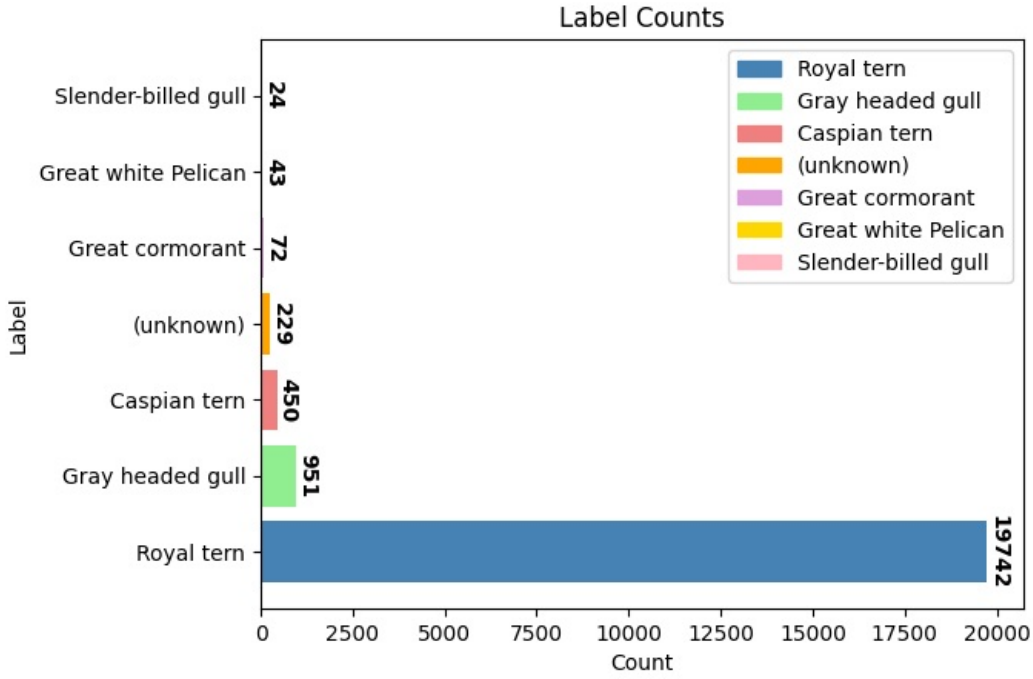
In [3]:
```
# Convert x and y coordinates to pixel values
data = pd.read_csv('labels.csv')
data['x'] = data.X*(27569/292.48)
data['y'] = data.Y*(28814/305.69)
data = data.drop(['X','Y','unsure'],axis = 1)
```

Class Distribution

In [4]:
```
label_counts = data['label'].value_counts().to_dict()

# Print label counts
for label, count in label_counts.items():
    print(f'{label}: {count}')

# Plot horizontal bar graph
labels = list(label_counts.keys())
counts = list(label_counts.values())

colors = ['steelblue', 'lightgreen', 'lightcoral', 'orange', 'plum', 'gold', 'lightpink']

plt.barh(labels, counts, color=colors)
plt.xlabel('Count')
plt.ylabel('Label')
plt.title('Label Counts')
```

```
# Add count labels on each bar
for i, count in enumerate(counts):
    plt.text(count, i, str(count), ha='left', va='center', color='black', weight='bold',
rotation=270)

# Create legend
legend_elements = [plt.Rectangle((0, 0), 1, 1, color=color) for color in colors[:len(labels)]]
plt.legend(legend_elements, labels, loc='upper right')

plt.show()
```

```
Royal tern: 19742
Gray headed gull: 951
Caspian tern: 450
(unknown): 229
Great cormorant: 72
Great white Pelican: 43
Slender-billed gull: 24
```

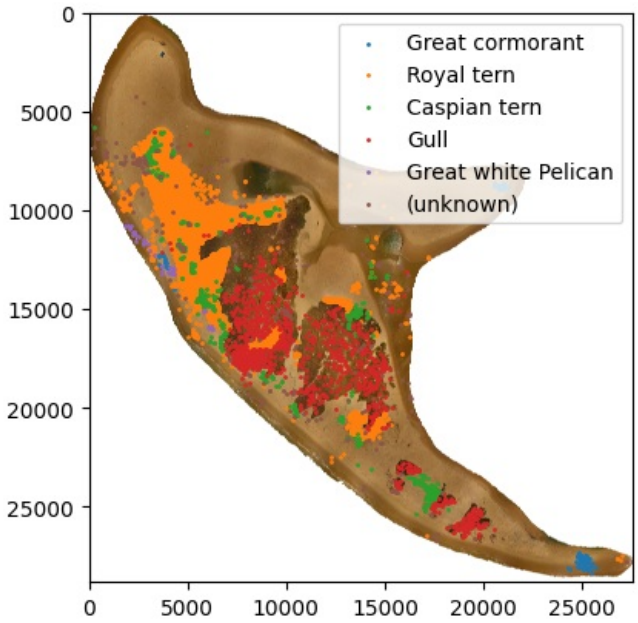


Visualzing points on orthomosaic:

In [5]:
```
data['label'] = data['label'].replace(['Slender-billed gull', 'Gray headed gull'], 'Gull')
data.to_csv('labels_new.csv',index=False)
```

In [6]:
```
tif_image_path = "sea_birds.tif"
csv_file_path = "labels_new.csv"

image = cv2.flip(cv2.imread(tif_image_path),0)

x_coords = []
y_coords = []
labels = []
with open(csv_file_path, "r") as file:
    reader = csv.reader(file)
    next(reader)  # Skip the header row if it exists

    for row in reader:
        x_coords.append(float(row[2]))
        y_coords.append(float(row[3]))
        labels.append(row[1])
```

```
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

unique_labels = list(set(labels))  # Get unique labels

for label in unique_labels:
    label_x = [x for x, l in zip(x_coords, labels) if l == label]
    label_y = [y for y, l in zip(y_coords, labels) if l == label]
    plt.scatter(label_x, label_y, marker='o', label=label, s=1)

plt.legend()
# Save the figure with plotted points as a new TIF file
plt.savefig('output.tif', dpi='figure', format='tif')

plt.show()
```



Creating non-overalping tiles:

In [7]:

```
# Open the TIFF file
image_bgr = cv2.flip(cv2.imread("sea_birds.tif"),0)
image = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)

# Set the tile size (width, height)
tile_size = (800, 600)  # Adjust the values accordingly

# Get the dimensions of the image
image_width, image_height =  (27569, 28814)

# Calculate the number of tiles horizontally and vertically
num_tiles_horizontal = image_width // tile_size[0]
num_tiles_vertical = image_height // tile_size[1]

# Divide the image into tiles
tiles = []
for y in range(num_tiles_vertical):
    for x in range(num_tiles_horizontal):
        left = x * tile_size[0]
        upper = y * tile_size[1]
        right = left + tile_size[0]
        lower = upper + tile_size[1]
        tile = image[upper:lower, left:right]
        tiles.append(tile)
```

```
        tile_objects = data[
        ((data["x"] >= int(left)) & (data["x"] < int(right))) &
        ((data["y"] >= int(upper)) & (data["y"] < int(lower)))
        ]
        tile_objects['x']=tile_objects['x']-left
        tile_objects['y']=tile_objects['y']-upper
        tile_objects.to_csv(f"dataset/tile_{x}_{y}.csv", index=False)

        # you can also save the tile image
        plt.imshow(tile)
        plt.axis("off")  # Optional: Remove axis
        plt.savefig(f"dataset/tile_{x}_{y}.jpg", bbox_inches="tight", pad_inches=0)
        plt.close()
```

```
<ipython-input-7-f99bdc9c23ee>:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  tile_objects['x']=tile_objects['x']-left
<ipython-input-7-f99bdc9c23ee>:30: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  tile_objects['y']=tile_objects['y']-upper
```

In [8]:

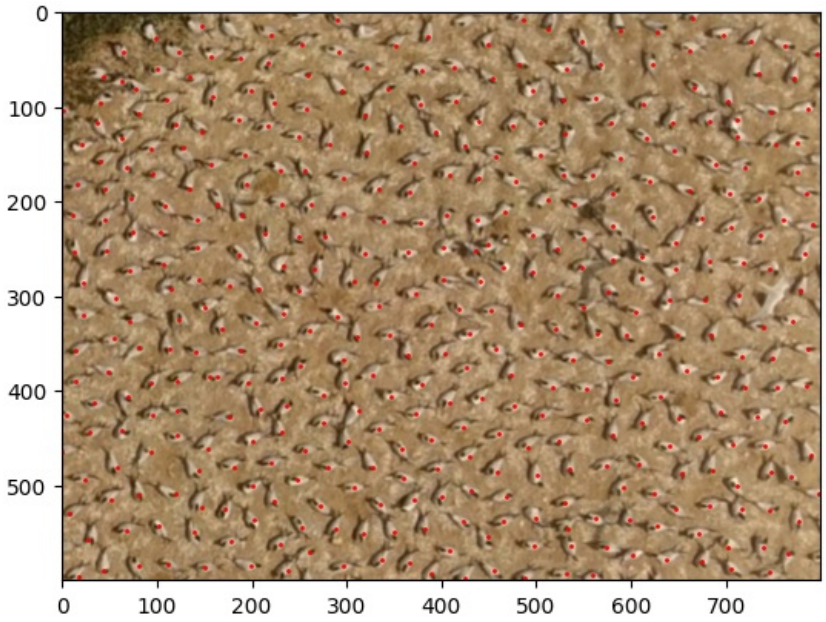
```
# Resize the tiles to make sure everything worked fine
def resize_png_files(library_path, target_size=(800, 600)):
    # Loop through all files in the library path
    for filename in os.listdir(library_path):
        file_path = os.path.join(library_path, filename)

        # Check if the file is a PNG
        if os.path.isfile(file_path) and filename.lower().endswith('.jpg'):
            # Read the image
            image = cv2.imread(file_path, cv2.IMREAD_UNCHANGED)

            # Resize the image
            resized_image = cv2.resize(image, target_size)

            # Write the resized image to the same file path
            cv2.imwrite(file_path, resized_image)

            print(f"Resized {filename} and saved at {file_path}")

# Example usage
library_path = 'dataset'
resize_png_files(library_path)
```

Visualize tile with bird positions

In [9]:

```
i,j = 7,21  # Replace with the desired tile index
tile_path = f"dataset/tile_{i}_{j}.jpg"  # Update with your tile image path
image = cv2.imread(tile_path)

# Load the object coordinates from the CSV file
csv_path = f"dataset/tile_{i}_{j}.csv"  # Update with your CSV file path
objects_df = pd.read_csv(csv_path)  # Assuming the CSV file has columns "X" and "Y" for object
coordinates
```

```
x_coords = []
y_coords = []
import numpy as np

for _, row in objects_df.iterrows():
#     x_coords.append(int(np.interp(row[5], [0, 27569], [0, 800])))
#     y_coords.append(int(np.interp(row[6], [0, 28814], [0, 600])))
    x_coords.append(float(row[2]))
    y_coords.append(float(row[3]))

# with open(tile_path, "r") as file:
#     reader = csv.reader(file)
#     next(reader)  # Skip the header row if it exists

#     for row in reader:
#         x_coords.append(float(row[5]))
#         y_coords.append(float(row[6]))
#         labels.append(row[1])

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

plt.scatter(x_coords, y_coords, color='red', marker='o', s=1)

plt.show()
```



Delete CSV files that have no birds:

In [10]:

```
def delete_csv_files_with_empty_label(directory):
    # Iterate through files in the directory
    for filename in os.listdir(directory):
        if filename.endswith('.csv'):
            csv_file_path = os.path.join(directory, filename)
            image_file_path = os.path.join(directory, filename.replace('.csv', '.jpg'))

            # Read the CSV file
            with open(csv_file_path, 'r') as csv_file:
                csv_reader = csv.DictReader(csv_file)

                # Check if 'label' column is empty
                if 'label' in csv_reader.fieldnames and all(row['label'] == '' for row in
```

```
csv_reader):
                    # Delete the CSV file
                    os.remove(csv_file_path)
                    print(f"Deleted CSV: {filename}")

                    # Delete the corresponding image file
                    if os.path.exists(image_file_path):
                        os.remove(image_file_path)
                        print(f"Deleted Image: {filename.replace('.csv', '.jpg')}")

    print("Deletion process completed.")

# Specify the directory containing the CSV files and images
directory_path = 'dataset'

# Call the function to delete CSV files with an empty 'label' column and their corresponding
images
delete_csv_files_with_empty_label(directory_path)
```

```
Deletion process completed.
```

Create coco XML files

In [15]:

```
import os
import pandas as pd
import xml.etree.ElementTree as ET

def create_xml_coco(file_path, objects):
    root = ET.Element("annotation")

    path = ET.SubElement(root, "path")
    path.text = os.path.basename(file_path)

    source = ET.SubElement(root, "source")
    database = ET.SubElement(source, "database")
    database.text = "Unknown"
    reviewresult = ET.SubElement(source, "reviewresult")
    comment = ET.SubElement(source, "comment")
    admin = ET.SubElement(comment, "admin")
    subord = ET.SubElement(comment, "subord")

    size = ET.SubElement(root, "size")
    width_elem = ET.SubElement(size, "width")
    width_elem.text = "800"
    height_elem = ET.SubElement(size, "height")
    height_elem.text = "600"
    depth = ET.SubElement(size, "depth")
    depth.text = "3"

    segments = ET.SubElement(root, "segments")
    segments.text = "0"

    for obj in objects:
        object_elem = ET.SubElement(root, "object")
        name = ET.SubElement(object_elem, "name")
        name.text = obj["label"]

        pose = ET.SubElement(object_elem, "pose")
        pose.text = "Unspecified"
```

```
        truncated = ET.SubElement(object_elem, "truncated")
        truncated.text = "0"

        difficult = ET.SubElement(object_elem, "difficult")
        difficult.text = "0"

        bndbox = ET.SubElement(object_elem, "bndbox")
        xmin = ET.SubElement(bndbox, "xmin")
        xmin.text = str(obj["xmin"])
        xmax = ET.SubElement(bndbox, "xmax")
        xmax.text = str(obj["xmax"])
        ymin = ET.SubElement(bndbox, "ymin")
        ymin.text = str(obj["ymin"])
        ymax = ET.SubElement(bndbox, "ymax")
        ymax.text = str(obj["ymax"])

    folder = ET.SubElement(root, "folder")
    folder.text = "dataset/"

    filename = ET.SubElement(root, "filename")
    filename.text = os.path.basename(file_path)

    tree = ET.ElementTree(root)
    xml_file_path = os.path.splitext(file_path)[0] + ".xml"
    tree.write(xml_file_path)

    print(f"Generated XML file: {xml_file_path}")



# Define the dataset path where the CSV files are stored
dataset_path = "dataset"

# Label sizes dictionary
label_sizes = {
    "Great white Pelican": {"width": 80, "height": 80}, #done
    "Caspian tern": {"width": 40, "height": 40}, #done
    "Gull": {"width": 30, "height": 30}, #done
    "(unknown)": {"width": 40, "height": 40}, #done
    "Great cormorant": {"width": 60, "height": 60}, #done
    "Royal tern": {"width": 40, "height": 40} #done
}

# Iterate over each CSV file in the dataset
for file_name in os.listdir(dataset_path):
    if file_name.endswith(".csv"):

        file_path = os.path.join(dataset_path, file_name)

        # Read the CSV file
        df = pd.read_csv(file_path)

        # Initialize a list to store the objects
        objects = []

        # Iterate over each row
        for index, row in df.iterrows():
            label = row["label"]
            center_x = row["x"]
```

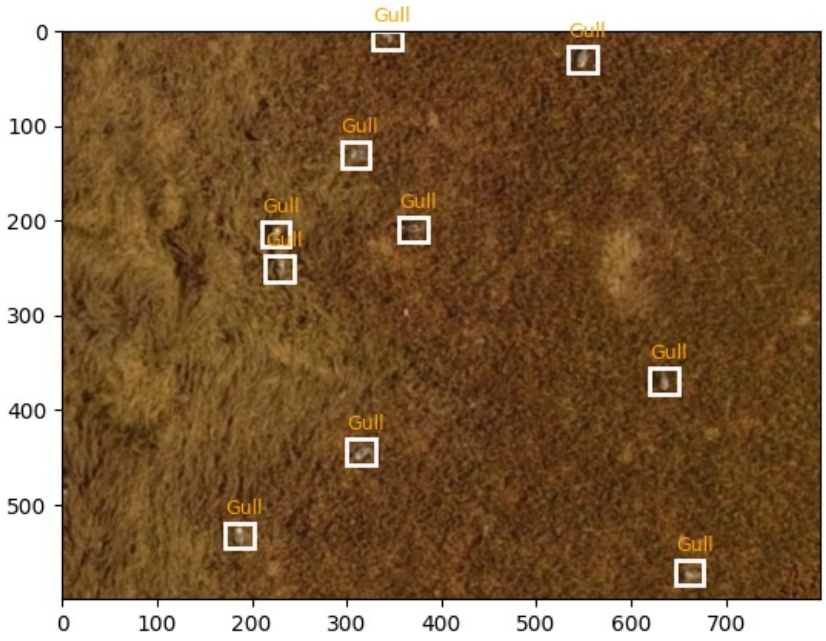
```
            center_y = row["y"]

            # Check if the label is one of the desired labels
            if label in label_sizes:
                size = label_sizes[label]
                width = size["width"]
                height = size["height"]

                # Calculate the bounding box coordinates
                xmin = max(center_x - width / 2, 0)
                ymin = max(center_y - height / 2, 0)
                xmax = min(center_x + width / 2, 800)
                ymax = min(center_y + height / 2, 600)

                # Create an object dictionary
                obj = {
                    "label": label,
                    "xmin": xmin,
                    "ymin": ymin,
                    "xmax": xmax,
                    "ymax": ymax
                }

                # Append the object to the list
                objects.append(obj)
        file_path = os.path.splitext(file_path+'.jpg'
        # Create XML in COCO format
        create_xml_coco(file_path, objects)
```

In [13]:
```
# delete all csv files
!rm dataset/*.csv
```

visualizing object positions from XML files

In [17]:
```
import matplotlib.patches as patches
# Path to the image and annotation files
i,j=9,23
image_path = f'dataset/tile_{i}_{j}.jpg'
annotation_path = f'dataset/tile_{i}_{j}.xml'

# Load the image
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Create figure and axes
fig, ax = plt.subplots()

# Display the image
ax.imshow(image)

# Read the XML file
tree = ET.parse(annotation_path)
root = tree.getroot()

# Iterate through all object annotations
for obj in root.findall('object'):
    # Extract class label
    class_label = obj.find('name').text

    # Extract bounding box coordinates
```

```
    bbox = obj.find('bndbox')
    xmin = int(round(float(bbox.find('xmin').text)))
    ymin = int(round(float(bbox.find('ymin').text)))
    xmax = int(round(float(bbox.find('xmax').text)))
    ymax = int(round(float(bbox.find('ymax').text)))-3

    # Draw the bounding box on the image
    # Create a rectangle patch
    rect = patches.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin, linewidth=2,
edgecolor='w', facecolor='none')

    # Add the rectangle patch to the axes
    ax.add_patch(rect)
    ax.text(xmin, ymin - 10, class_label, fontsize=9, color='orange')

# Show the plot
plt.show()
```



Note:

We created COCO XML files because we tried using SSD and Faster RCNN with this data initially, but for YOLOv8 we had to convert the XML files to YOLO format

Image Agumentation

In [78]:

```
import imgaug.augmenters as iaa
import xml.etree.ElementTree as ET
import os
import numpy as np
from PIL import Image

# Directory paths
image_dir = ''
annotation_dir = ''

# Augmentation settings
augmentations = {
#     'rotate_90': iaa.Affine(rotate=90),
#     'rotate_180': iaa.Affine(rotate=180),
#    'rotate_270': iaa.Affine(rotate=270),
```

```
#    'flip': iaa.Fliplr(1.0)
}

# Iterate through the images
for image_file in os.listdir(image_dir):
    if image_file.endswith('.jpg'):
        image_path = os.path.join(image_dir, image_file)
        image = Image.open(image_path)
        image_name = os.path.splitext(image_file)[0]

        # Convert PIL image to NumPy array
        image_array = np.array(image)

        # Iterate through the augmentations
        for aug_name, augmentation in augmentations.items():
            augmented_image_array = augmentation(image=image_array)
            augmented_image = Image.fromarray(augmented_image_array)

            augmented_image_path = os.path.join(image_dir, f'{image_name}_{aug_name}.jpg')
            augmented_image.save(augmented_image_path)

            # Update XML annotations
            annotation_file = os.path.join(annotation_dir, f'{image_name}.xml')
            augmented_annotation_file = os.path.join(annotation_dir,
f'{image_name}_{aug_name}.xml')

            tree = ET.parse(annotation_file)
            root = tree.getroot()

            # Update image filename in the annotation XML
            root.find('filename').text = f'{image_name}_{aug_name}.jpg'

            # Apply necessary transformations to bounding boxes in the annotation XML
            for obj in root.findall('object'):
                bbox = obj.find('bndbox')
                xmin = float(bbox.find('xmin').text)
                ymin = float(bbox.find('ymin').text)
                xmax = float(bbox.find('xmax').text)
                ymax = float(bbox.find('ymax').text)

                if aug_name == 'rotate_90':
                    xmin, ymin, xmax, ymax = image.width-ymin-100, xmin - 100, image.width-
ymax-100, xmax-100
                elif aug_name == 'rotate_180':
                    xmin, ymin, xmax, ymax = image.width - xmax, image.height - ymax,
image.width - xmin, image.height - ymin
                elif aug_name == 'rotate_270':
                    xmin, ymin, xmax, ymax = ymax+100, image.height-xmin+100 , ymin+100,
image.height-xmax+100
                elif aug_name == 'flip':
                    xmin, xmax = image.width - xmax, image.width - xmin

                bbox.find('xmin').text = str(xmin)
                bbox.find('ymin').text = str(ymin)
                bbox.find('xmax').text = str(xmax)
                bbox.find('ymax').text = str(ymax)

            # Save the augmented annotation XML
            tree.write(augmented_annotation_file)
```

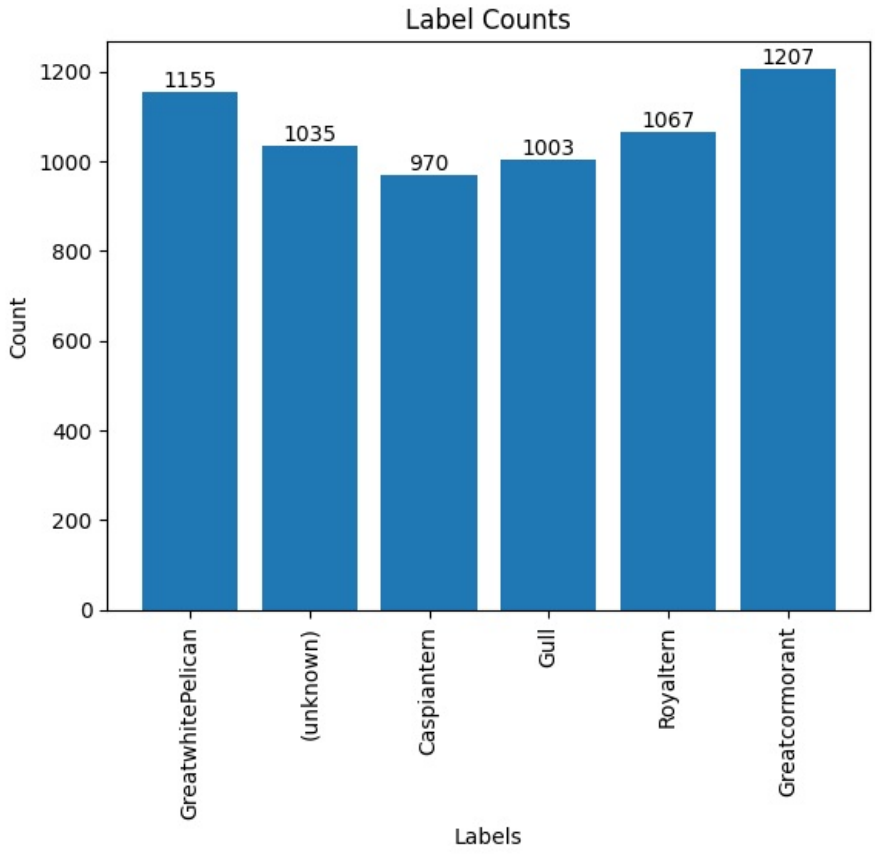
Remove the Royal tern tag to down sample

In [72]:
```
import os
import xml.etree.ElementTree as ET

def remove_object_from_xml(xml_path, object_name):
    tree = ET.parse(xml_path)
    root = tree.getroot()

    # Find all object elements with the specified name
    objects = root.findall("object[name='{}']".format(object_name))

    # Remove the found objects from the XML tree
    for obj in objects:
        root.remove(obj)

    # Save the modified XML back to the file
    tree.write(xml_path)

# Set the directory path containing the COCO XML files
folder_path = ''

# Iterate over all XML files in the folder
for filename in os.listdir(folder_path):
    if filename.endswith(".xml"):
        xml_file_path = os.path.join(folder_path, filename)
        remove_object_from_xml(xml_file_path, 'Royal tern')
```

Balanced Dataset

In [8]:
```
import os
import xml.etree.ElementTree as ET
from collections import Counter
import matplotlib.pyplot as plt

def parse_xml_file(xml_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    labels = []
    for obj in root.findall('object'):
        label = obj.find('name').text
        labels.append(label)

    return labels

def count_labels(folder_path):
    all_labels = []

    for file in os.listdir(folder_path):
        if file.endswith('.xml'):
            xml_file = os.path.join(folder_path, file)
            labels = parse_xml_file(xml_file)
            all_labels.extend(labels)

    return Counter(all_labels)

def plot_label_counts(label_counts):
```

```
    labels = label_counts.keys()
    counts = label_counts.values()

    plt.bar(labels, counts)
    plt.xlabel('Labels')
    plt.ylabel('Count')
    plt.title('Label Counts')
    plt.xticks(rotation=90)

    for i, count in enumerate(counts):
        plt.text(i, count, str(count), ha='center', va='bottom')

    plt.show()

folder_path = 'Desktop/dataset/train'

label_counts = count_labels(folder_path)
plot_label_counts(label_counts)
```



In [10]:
```
label_counts
```

Out[10]:
```
Counter({'GreatwhitePelican': 1155,
         '(unknown)': 1035,
         'Caspiantern': 970,
         'Gull': 1003,
         'Royaltern': 1067,
         'Greatcormorant': 1207})
```

Appendix B: Hyperparameter Tunning Notebook

In-flight Birds detection, species detection, and counting

Hyperparameter Tunning

In [ ]:
```
from ultralytics import YOLO

lr0=0.01
lrf_set=[0.01, 0.001, 0.0001]
warmup_epochs_set=[2.0, 3.0, 5.0]
warmup_momentum_set=[0.70, 0.75, 0.80, 0.85, 0.90, 0.95]
optimizer = 'auto'
# Load the model
model = YOLO("yolov8l.yaml")  # fine tune YOLO v8 large model
# Train the model with above parameter values
for lrf in lrf_set:
    for warmup_epochs_set in warmup_epochs_set:
        for warmup_momentum_set in warmup_momentum_set:
            results = model.train(data="config.yaml",
                                  lr0 = lr0,
                                  lrf = lrf,
                                  warmup_epochs = warmup_epochs,
                                  warmup_momentum = warmup_momentum,
                                  optimizer = optimizer,
                                  project =
f'TUNED_lr0_{lr0}_lrf_{lrf}_warmup_epochs_{warmup_epochs}_warmup_momentum_{warmup_momentum}_opti

                                  epochs = 10)
```

Note:

- The code kept crashing because of out of memory error. To cater this, the tunning was done in parts.
- The plots were observed using wandb on weights and biases server.

In [ ]:
```
# Using the chosen parameters:
lr0 = 0.01
lrf = 0.001
warmup_epochs = 5.0
warmup_momentum = 0.95
optimizer_set = ['SGD', 'Adam', 'RMSProp']
# Load the model
model = YOLO("yolov8l.yaml")  # fine tune YOLO v8 large model
# Train the model with above parameter values

fpr optimizer in optimizer_set:
    results = model.train(data="config.yaml",
                                    lr0 = lr0,
                                    lrf = lrf,
                                    warmup_epochs = warmup_epochs,
                                    warmup_momentum = warmup_momentum,
                                    optimizer = optimizer,
                                    project =
f'TUNED_lr0_{lr0}_lrf_{lrf}_warmup_epochs_{warmup_epochs}_warmup_momentum_{warmup_momentum}_opti
```

```
                                        epochs = 10)
```

If you want to resume any training pass only the argument resume = True

In [ ]:

```
from ultralytics import YOLO
model =
YOLO("TUNED_lr0_0.01_lrf_0.001_warmup_epochs_5.0_warmup_momentum_0.95_optimizer_auto/train/weigh
  # build a new model from scratch
# Train the model with current parameter values
results = model.train(resume=True)
```

Appendix C: Training Notebook

In-flight Birds detection, species detection, and counting

Training YOLO v8

In [4]:
```
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8l.yaml")  # build a new model from scratch

# Use the model
results = model.train(data="config.yaml", epochs=300)  # train the model
```

```
New https://pypi.org/project/ultralytics/8.0.137 available 😃 Update with 'pip install -U ultralytics'
Ultralytics YOLOv8.0.136 🚀 Python-3.8.17 torch-2.0.1+cu117 CUDA:0 (NVIDIA GeForce RTX 3090, 24268MiB)
engine/trainer: task=detect, mode=train, model=runs/detect/train2/weights/last.pt, data=config.yaml, epochs=300
, patience=50, batch=16, imgsz=640, save=True, save_period=1, cache=False, device=None, workers=8, project=None
, name=None, exist_ok=False, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True, single_
cls=False, rect=False, cos_lr=False, close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, over
lap_mask=True, mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False, save_hybrid=False, conf=None, i
ou=0.7, max_det=300, half=False, dnn=False, plots=True, source=None, show=False, save_txt=False, save_conf=Fals
e, save_crop=False, show_labels=True, show_conf=True, vid_stride=1, line_width=None, visualize=False, augment=F
alse, agnostic_nms=False, classes=None, retina_masks=False, boxes=True, format=torchscript, keras=False, optimi
ze=False, int8=False, dynamic=False, simplify=False, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, mo
mentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=7.5, cls=0.5
, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, tr
anslate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0
.0, cfg=None, tracker=botsort.yaml, save_dir=runs/detect/train4
/home/msc1/anaconda3/envs/PT/lib/python3.8/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Ple
ase update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm

                   from  n    params  module                                       arguments
  0                  -1  1      1856  ultralytics.nn.modules.conv.Conv             [3, 64, 3, 2]
  1                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  2                  -1  3    279808  ultralytics.nn.modules.block.C2f             [128, 128, 3, True]
  3                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  4                  -1  6   2101248  ultralytics.nn.modules.block.C2f             [256, 256, 6, True]
  5                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  6                  -1  6   8396800  ultralytics.nn.modules.block.C2f             [512, 512, 6, True]
  7                  -1  1   2360320  ultralytics.nn.modules.conv.Conv             [512, 512, 3, 2]
  8                  -1  3   4461568  ultralytics.nn.modules.block.C2f             [512, 512, 3, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  3   4723712  ultralytics.nn.modules.block.C2f             [1024, 512, 3]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  3   1247744  ultralytics.nn.modules.block.C2f             [768, 256, 3]
 16                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  3   4592640  ultralytics.nn.modules.block.C2f             [768, 512, 3]
 19                  -1  1   2360320  ultralytics.nn.modules.conv.Conv             [512, 512, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  3   4723712  ultralytics.nn.modules.block.C2f             [1024, 512, 3]
 22        [15, 18, 21]  1   5583571  ultralytics.nn.modules.head.Detect           [1, [256, 512, 512]]
YOLOv8l summary: 365 layers, 43630611 parameters, 43630595 gradients

Transferred 595/595 items from pretrained weights
```

Note:

If you want to resume the training, change the model to last.pt file path and add 'resume =True' in train command.

In [1]:
```
from ultralytics import YOLO
```

```python
# Load model
model = YOLO("runs/detect/train/weights/last.pt")
results = model.train(resume=True)
```

```
New https://pypi.org/project/ultralytics/8.0.137 available 😃 Update with 'pip install -U ultralytics'
Ultralytics YOLOv8.0.136 🚀 Python-3.8.17 torch-2.0.1+cu117 CUDA:0 (NVIDIA GeForce RTX 3090, 24268MiB)
engine/trainer: task=detect, mode=train, model=runs/detect/train/weights/last.pt, data=config.yaml, epochs=300, patience=50, batch=16, imgsz=640, save=True, save_period=-1, cache=False, device=None, workers=8, project=None, name=None, exist_ok=False, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=False, close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, overlap_mask=True, mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False, save_hybrid=False, conf=None, iou=0.7, max_det=300, half=False, dnn=False, plots=True, source=None, show=False, save_txt=False, save_conf=False, save_crop=False, show_labels=True, show_conf=True, vid_stride=1, line_width=None, visualize=False, augment=False, agnostic_nms=False, classes=None, retina_masks=False, boxes=True, format=torchscript, keras=False, optimize=False, int8=False, dynamic=False, simplify=False, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.0, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0.0, cfg=None, tracker=botsort.yaml, save_dir=runs/detect/train2
/home/msc1/anaconda3/envs/PT/lib/python3.8/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
wandb: Currently logged in as: abdullahdar2017. Use `wandb login --relogin` to force relogin
```

Tracking run with wandb version 0.15.5

Run data is saved locally in /home/msc1/Desktop/YOLO/wandb/run-20230719_104545-c8gn8x21

Syncing run scarlet-voice-8 to Weights & Biases (docs)

View project at https://wandb.ai/abdullahdar2017/YOLOv8

View run at https://wandb.ai/abdullahdar2017/YOLOv8/runs/c8gn8x21

```
                   from  n    params  module                                       arguments
  0                  -1  1      1856  ultralytics.nn.modules.conv.Conv             [3, 64, 3, 2]
  1                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  2                  -1  3    279808  ultralytics.nn.modules.block.C2f             [128, 128, 3, True]
  3                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  4                  -1  6   2101248  ultralytics.nn.modules.block.C2f             [256, 256, 6, True]
  5                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  6                  -1  6   8396800  ultralytics.nn.modules.block.C2f             [512, 512, 6, True]
  7                  -1  1   2360320  ultralytics.nn.modules.conv.Conv             [512, 512, 3, 2]
  8                  -1  3   4461568  ultralytics.nn.modules.block.C2f             [512, 512, 3, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  3   4723712  ultralytics.nn.modules.block.C2f             [1024, 512, 3]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  3   1247744  ultralytics.nn.modules.block.C2f             [768, 256, 3]
 16                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  3   4592640  ultralytics.nn.modules.block.C2f             [768, 512, 3]
 19                  -1  1   2360320  ultralytics.nn.modules.conv.Conv             [512, 512, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  3   4723712  ultralytics.nn.modules.block.C2f             [1024, 512, 3]
 22        [15, 18, 21]  1   5583571  ultralytics.nn.modules.head.Detect           [1, [256, 512, 512]]
YOLOv8l summary: 365 layers, 43630611 parameters, 43630595 gradients

Transferred 595/595 items from pretrained weights
optimizer: SGD(lr=0.01, momentum=0.9) with parameter groups 97 weight(decay=0.0), 104 weight(decay=0.0005), 103 bias(decay=0.0)
Resuming training from runs/detect/train/weights/last.pt from epoch 170 to 300 total epochs
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to runs/detect/train2
Starting training for 300 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    170/300      14.8G      1.293     0.7924       1.01        111        640: 100%|██████████| 1351/1351 [05:29<00:00,  4.10it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/70 [00:18<00:00,  3.81it/s]
                 all       2209      34336      0.853      0.833      0.862      0.478

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    171/300        17G      1.301        0.8      1.015         50        640: 100%|██████████| 1351/1351 [05:30<00:00,  4.08it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/70 [00:18<00:00,  3.84it/s]
```

```
                 all       2209      34336      0.854      0.832      0.861      0.478

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    172/300      15.5G       1.31     0.8015      1.016         83        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.82it/s]
                 all       2209      34336      0.854      0.832      0.862      0.477

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    173/300        14G       1.31     0.8025      1.017         26        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.81it/s]
                 all       2209      34336      0.853      0.832      0.861      0.477

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    174/300      18.7G      1.308     0.8027      1.016        118        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.83it/s]
                 all       2209      34336      0.852      0.832      0.861      0.477

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    175/300      15.5G      1.305     0.7959      1.017        118        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.81it/s]
                 all       2209      34336      0.852      0.832      0.861      0.476

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    176/300      18.2G      1.307     0.8043      1.016         29        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.83it/s]
                 all       2209      34336      0.852      0.832      0.861      0.476

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    177/300      17.8G      1.304     0.8013      1.015         84        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.07it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.81it/s]
                 all       2209      34336      0.851      0.832       0.86      0.476

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    178/300      14.2G      1.305     0.7966      1.016         94        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.84it/s]
                 all       2209      34336      0.851      0.832       0.86      0.476

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    179/300      14.6G      1.302     0.7912      1.015         61        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.81it/s]
                 all       2209      34336      0.852      0.832      0.861      0.475

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    180/300      16.1G      1.302     0.7959      1.013         63        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.80it/s]
                 all       2209      34336      0.851      0.831      0.861      0.475

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    181/300      17.2G      1.307     0.7957      1.016         32        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.83it/s]
                 all       2209      34336      0.852       0.83       0.86      0.475

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    182/300      13.6G      1.304     0.7951      1.013         27        640: 100%|██████████| 1351/1351 [05:3
1<00:00,  4.08it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 70/7
0 [00:18<00:00,  3.81it/s]
                 all       2209      34336      0.852       0.83       0.86      0.475
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   183/300      15.8G      1.298     0.7916      1.013        244        640:  61%|██████    | 826/1351 [03:23
<02:09,  4.07it/s]
```

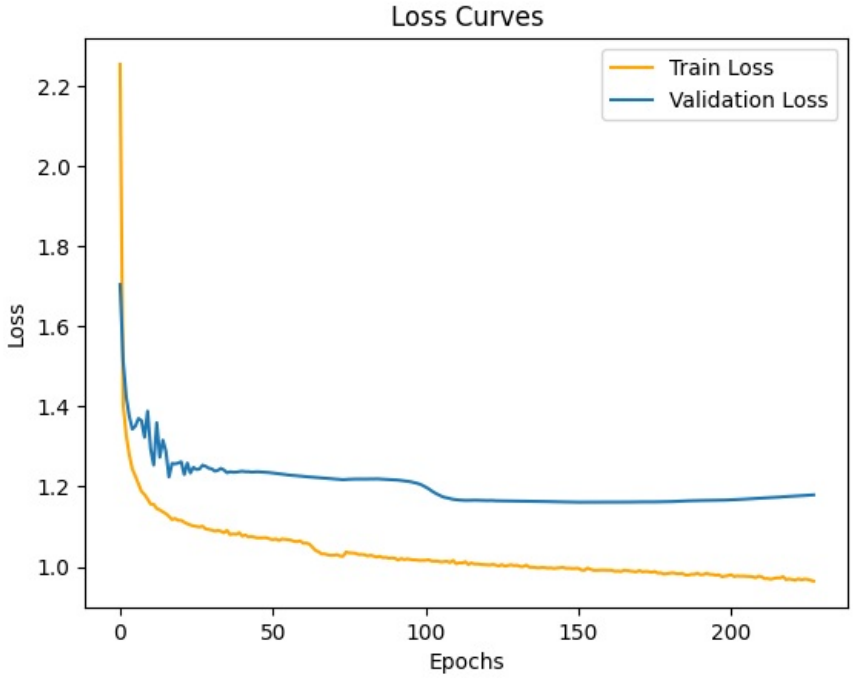
Appendix D: Traing Plots Notebook

In-flight Birds detection, species detection, and counting

Plot training results

Training loss vs Validation loss

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
# Read the CSV file into a DataFrame
df =
pd.read_csv('TUNED_lr0_0.01_lrf_0.001_warmup_epochs_5.0_warmup_momentum_0.95_optimizer_auto/trai
 header=0)

# Extract the required columns
epochs = df.iloc[:,[0]]
train_loss = df.iloc[:,[3]]
val_loss = df.iloc[:,[10]]

# Plot the curves with different colors
plt.plot(epochs, train_loss, color='orange', label='Train Loss')
plt.plot(epochs, val_loss, color='tab:blue', label='Validation Loss')

# Set plot title and labels
plt.title('Loss Curves')
plt.xlabel('Epochs')
plt.ylabel('Loss')

# Add legend
plt.legend()

# Show the plot
plt.show()
```

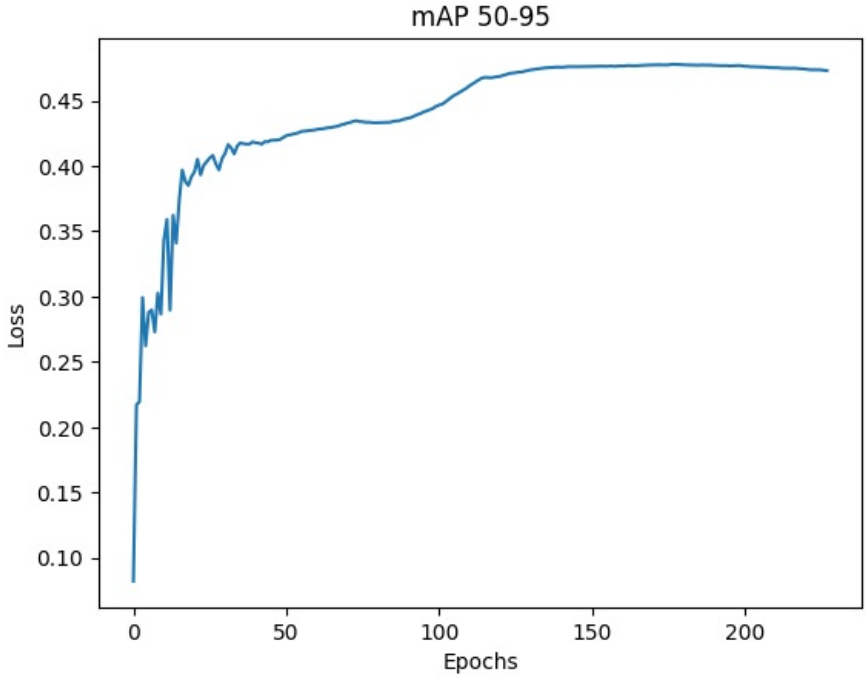


mAP @ 0.5:0.05:0.95 IOU

mAP @ 0.5:0.05:0.95 IOU:

In [2]:

```
map_50_95 = df.iloc[:,[7]]
plt.plot(epochs, map_50_95)

plt.title('mAP 50-95')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



In [ ]:

Appendix E: Evaluation Notebook

In-flight Birds detection, species detection, and counting

Evaluation

In [1]:
```
from ultralytics import YOLO
# Load the model
model = YOLO("runs/train_save/weights/best.pt")
# Validation metrics
metrics = model.val()  # no arguments needed, dataset and settings remembered
```

```
/home/msc1/anaconda3/envs/PT/lib/python3.8/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Ple
ase update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
Ultralytics YOLOv8.0.136 🚀 Python-3.8.17 torch-2.0.1+cu117 CUDA:0 (NVIDIA GeForce RTX 3090, 24268MiB)
YOLOv8l summary (fused): 268 layers, 43607379 parameters, 0 gradients
val: Scanning /home/msc1/Desktop/YOLO/data/labels/test.cache... 2209 images, 0 backgrounds, 0 corrupt: 100%|███
███████| 2209/2209 [00:00<?, ?it/s]
val: WARNING ⚠ /home/msc1/Desktop/YOLO/data/images/test/seabirds_rgb_808.png: 1 duplicate labels removed
val: WARNING ⚠ /home/msc1/Desktop/YOLO/data/images/test/seabirds_rgb_810.png: 1 duplicate labels removed
val: WARNING ⚠ /home/msc1/Desktop/YOLO/data/images/test/seabirds_rgb_850.png: 1 duplicate labels removed
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):   1%|          | 2/13
9 [00:04<05:40,  2.48s/it]WARNING ⚠ NMS time limit 1.300s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):   2%|          | 3/13
9 [00:06<05:16,  2.33s/it]WARNING ⚠ NMS time limit 1.300s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 139/
139 [00:35<00:00,  3.96it/s]
                   all       2209      34336      0.854      0.834      0.861      0.479
Speed: 0.2ms preprocess, 8.4ms inference, 0.0ms loss, 1.9ms postprocess per image
Results saved to runs/detect/val12
```

Create a custom classification report

(to show the F1 Score as well)

In [2]:
```
# Create custom Classification report
col = ['Class', 'images', 'instances', 'P', 'R', 'F1-Score', 'mAP50', 'mAP50-95']
f1_avg = sum(metrics.box.f1)
all = 'all'
images= '2209'
instances = '34336'
print(f"{col[0]: <20} {col[1]: <10} {col[2]: <10} {col[3]: <10} {col[4]: <10} {col[5]: <10}
{col[6]: <10} {col[7]: <10} ")

print(f"{all: <20} {images: <10} {instances: <10} {round(metrics.box.mp,3): <10}
{round(metrics.box.mr,3): <10} {round(f1_avg,3): <10} {round(metrics.box.map50,3): <10}
{round(metrics.box.map,3): <10}")
```

```
Class               images     instances  P          R          F1-Score   mAP50      mAP50-95
all                 2209       34336      0.854      0.834      0.844      0.861      0.479
```

Note:

The plots for the evaluation are available in the results folder.

Appendix F: Inference Notebook

In-flight Birds detection, species detection, and counting

Inference

In [1]:
```
# Import all functions from the inf.py file
from inf import *
```

```
Functions:
    1- img_inf(img_path = None, species = False, resize = True)
    2- vid_inf(video_path = None, output_path = None)

Working:
    1- img_inf(img_path = None, species = False, resize = True):
        * If img_path is provided, inference on provided image.
        * If img_path isn't provided, inference on test image.
        * If species is set to True, does inference on species, otherwise, classify as 'Bird'.
        * If resize is set to False, it will not resize the output image.

    2- vid_inf(video_path = None, output_path = None):
        * If video_path is provided, does inference on provided video.
        * If output_path is provided, saves the output video on provided path.
        * If no argument is given, does inference on test video.

    3- If python file is run separately, it will do inference on test video first and than the test image
```

Inference on image

In [5]:
```
img_inf(img_path = 'tile_12_28_rotate_180.jpg', species = True, resize = False)
```

```
/home/msc1/Desktop/YOLO/species_model/train/weights/best.pt
Speed: 1.8ms preprocess, 8.8ms inference, 0.8ms postprocess per image at shape (1, 3, 480, 640)
```

Inference on video

In [2]:
```
vid_inf()
```

```
Speed: 3.3ms preprocess, 9.0ms inference, 0.6ms postprocess per image at shape (1, 3, 480, 640)
End of video.
```

Apendix G: Python File (inf.py)

```
import os
import math
from ultralytics import YOLO
import cv2
import cvzone
import time


print('''
Functions:
    1- img_inf(img_path = None, species = False, resize = True)
    2- vid_inf(video_path = None, output_path = None)

Working:
    1- img_inf(img_path = None, species = False, resize = True):
        * If img_path is provided, inference on provided image.
        * If img_path isn't provided, inference on test image.
        * If species is set to True, does inference on species, otherwise, classify as 'Bird'.
        * If resize is set to False, it will not resize the output image.

    2- vid_inf(video_path = None, output_path = None):
        * If video_path is provided, does inference on provided video.
        * If output_path is provided, saves the output video on provided path.
        * If no argument is given, does inference on test video.

    3- If python file is run separately, it will do inference on test video first and than the test image
''')


def img_inf(img_path = None, species = False, resize = True):
    if species == True:
        model_path = '/home/msc1/Desktop/YOLO/species_model/train/weights/best.pt'
        cls_name = ['Greatcormorant','GreatwhitePelican','Caspiantern','Gull','Royaltern','Unknown']
    elif species == False:
        model_path = "TUNED_lr0_0.01_lrf_0.001_warmup_epochs_5.0_warmup_momentum_0.95_optimizer_auto/train/weights/best.p
        cls_name = ['Bird']
    if img_path == None:
        img_path = '/home/msc1/Downloads/newmexico/BDA_12C_20181127_1_21.png'
    img = cv2.imread(img_path)
    print(model_path)
    model = YOLO(model_path)  # load model
    threshold = 0.2 # Score threshold

    results = model(img ,show = True, stream = True)

    while True:
        try:
            for result in results:
                for box in result.boxes:
                    x1, y1, x2, y2 = box.xyxy[0]
                    x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                    score = math.ceil((box.conf[0]*100)) / 100
                    if score > threshold:
                        #Draw B-Box on object:
                        cv2.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)), (0, 0, 100), 2)

                        #Write name of class on top:

                        text_size, _ = cv2.getTextSize(f'{cls_name[int(box.cls)]}', cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
                        text_w, text_h = text_size
                        cv2.rectangle(img, (int(x1), int(y1 - 30)), (int(x1) + text_w, int(y1)), (0, 0, 100), -1)

                        cv2.putText(img, cls_name[int(box.cls)], (int(x1), int(y1 - 10)),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (1, 1, 1), 1, cv2.LINE_AA)
            if resize == True:
                cv2.imshow("Image Inference",cv2.resize(img,(600,int((600 / img.shape[0]) * img.shape[1]))))  # Display i
            elif resize == False:
                cv2.imshow("Image Inference",img)
            if cv2.waitKey(1) == 27:  # Press escape key to close the window
                break
        except:
            print('No detections')
    cv2.destroyAllWindows()

def vid_inf(video_path = None, output_path = None):

    if video_path == None:
        video_path = 'test/test.mp4'

    cap = cv2.VideoCapture(video_path)
    # Get video frame dimensions and create a VideoWriter object for saving the output
    frame_width = int(cap.get(3))
    frame_height = int(cap.get(4))
```

```
    if output_path != None:
        out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'), 30, (frame_width, frame_height))
    if not cap.isOpened():
        print("Error: Could not open video file.")
        exit()

    model_path = "TUNED_lr0_0.01_lrf_0.001_warmup_epochs_5.0_warmup_momentum_0.95_optimizer_auto/train/weights/best.pt"

    # Load a model
    model = YOLO(model_path)  # load a custom model

    threshold = 0.2
    total_counts = 0

    while True:
        success, img = cap.read()

        if not success:
            print("End of video.")
            break

        results = model.track(img, stream=True, show=False, persist = True)

        for result in results:

            boxes = result.boxes
            for box in boxes:
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                w, h = x2 - x1, y2 - y1
                cx,cy = x1+ w//2, y1+ h//2
                if box.id != None:
                    ID = box.id.cpu().numpy().astype(int)[0]

                    score = math.ceil((box.conf[0]*100)) / 100
                    if score > threshold:
                        if cy <=502:
                            cv2.circle(img,(cx, cy), 5, (0,0,255),-1)
                        else:
                            cv2.circle(img,(cx, cy), 5, (0,255,0),-1)

                        if 0 < cx < img.shape[1] and 515 > cy > 485:
                            cv2.circle(img,(cx, cy), 5, (0,0,0),-1)
                            total_counts+= 1

            imgGraphic = cv2.imread('graphic1.png', cv2.IMREAD_UNCHANGED)
            img = cvzone.overlayPNG(img, imgGraphic, (0,0))
            cv2.putText(img, f' {total_counts}', (35, 80),cv2.FONT_HERSHEY_SIMPLEX, 3, (255, 255, 255), 3, cv2.LINE_AA)
        cv2.line(img, (0, 500), (img.shape[1], 500), (0, 0, 100), 2)
        if output_path != None:
            out.write(img)
        cv2.imshow("Video Detection", img)
        if cv2.waitKey(0) == 27:                ## Press escape key to come out of the loop (close the camera window)
            break

    cap.release()
    if output_path != None:
        out.release()
    cv2.destroyAllWindows()

def main():
    vid_inf()
    img_inf()

if __name__ == "__main__":
    main()
```

Appendix H: config.yaml file

```
path: /home/msc1/Desktop/YOLO/data  # dataset root dir
train: images/train  # train images (relative to 'path')
val: images/test  # val images (relative to 'path')

# Classes
names:
  0: Bird
```