UNIVERSITI MALAYSIA SARAWAK • UNIMAS •

Faculty of Engineering

A MOBILE APPLICATION FOR BIRD SPECIES RECOGNITION USING DEEP LEARNING TECHNIQUES

ALVAREZ BERAI ANAK UCHONG

Bachelor of Engineering

Electrical and Electronics Engineering with Honours

2023

UNIVERSITI MALAYSIA SARAWAK

Grade: ____________

Please tick (√)

Final Year Project Report	√
Masters	
PhD	

DECLARATION OF ORIGINAL WORK

This declaration is made on the 25th day of JULY 2023.

Student's Declaration:

I ALVAREZ BERAI ANAK UCHONG, 69024, FACULTY OF ENGINEERING hereby declare that the work entitled A MOBILE APPLICATION FOR BIRD SPECIES RECOGNITION USING DEEP LEARNING TECHNIQUES is my original work. I have not copied from any other students' work or from any other sources except where due reference or acknowledgement is made explicitly in the text, nor has any part been written for me by another person.

27/07/2023 — Date submitted

Alvarez Berai Anak Uchong (69024) — Name of the student (Matric No.)

Supervisor's Declaration:

I ASSOCIATE PROFESSOR TS DR ROHANA SAPAWI hereby certifies that the work entitled A MOBILE APPLICATION FOR BIRD SPECIES RECOGNITION USING DEEP LEARNING TECHNIQUES prepared by the above named student and was submitted to the "FACULTY" as a * partial/full fulfillment for the conferment of BACHELOR OF ENGINEERING, ELECTRICAL AND ELECTRONIC ENGINEERING WITH HONOURS, and the aforementioned work, to the best of my knowledge, is the said student's work.

Received for examination by: AP Ts Dr Rohana Sapawi (Name of the supervisor) Date: 27/07/2023.

A MOBILE APPLICATION FOR BIRD SPECIES RECOGNITION USING DEEP LEARNING TECHNIQUES

ALVAREZ BERAI ANAK UCHONG

A dissertation submitted in partial fulfilment
of the requirement for the degree of
Bachelor of Engineering
Electrical and Electronics Engineering with Honours

Faculty of Engineering

Universiti Malaysia Sarawak

2023

ACKNOWLEDGEMENT

First and foremost, I would like to thank our Almighty God for His grace, wisdom and protection throughout the research and writing process. Without His guidance, this project would not have been feasible.

I would also like to extend my deepest gratitude to Associate Professor Ts Dr Rohana Sapawi, my Final Year Project supervisor, for her unwavering guidance and support, as well as for generously sharing her skills from the beginning to the completion of this academic journey.

Additionally, I must give my most thanks to Universiti Malaysia Sarawak for providing access to research facilities and resources. Especially, I am very grateful to the Department of Electrical and Electronics Engineering at University Malaysia Sarawak for their support and resources throughout this academic endeavour.

Last but not least, I would like to express my heartfelt thanks to my family for their love, support and encouragement over the course of conducting research and writing the paper.

ABSTRACT

The development of a mobile application for bird species identification has become a significant area of research due to its potential to engage bird enthusiasts, facilitate citizen science initiatives, and contribute to conservation efforts. In this project, we present a comprehensive approach to building a mobile application for bird species identification, encompassing the development of a machine learning model in Jupyter Lab, model optimization using Google Cloud Platform with Vertex AI, then building the API to process the request then send prediction and the creation of an application through Android Studio. The system utilizeses a convolutional neural network (CNN) architecture through transfer learning trained on a dataset of local Sarawak birds. The images were preprocessed to enhance relevant features and remove noise. The CNN transfer learning model chosen was EfficientNet-B4. To validate the robustness of the model, 5 famous transfer learning model (EfficientNet-B4, ResNet-50, InceptionV3, MobileNetV2 and VGG16) were compared through parameters such as training time, training loss, validation loss, training accuracy, validation accuracy, test loss, and test accuracy. The chosen EfficientNet-B4 model achieved an accuracy of 93.06%, test loss of 0.3156, training accuracy of 96.81%, training loss of 0.1081, validation accuracy of 93.87%, validation loss of 0.2853 and average training time of 285 seconds for 1 epoch with a total of 20 epochs, demonstrating its effectiveness in accurately classifying the bird species. The successful completion of this project makes a valuable contribution to the field of bird species identification and conservation. The developed mobile application provides an accessible and user-friendly tool for bird enthusiasts, researchers, and citizen scientists, promoting active engagement in bird conservation efforts and facilitating data collection for monitoring bird populations and habitats. Future directions for this project include expanding the model's dataset to cover a broader range of bird species and environmental conditions, as well as integrating real-time updates and environmental sensor data to enhance the application's functionality and provide users with dynamic and contextually rich bird identification information.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN	-	Convulutional Neural Network
AI	-	Artificial Inteligence
TV	-	Television
CUB	-	Caltech UCSD Bird
UCSD	-	University of California San Diego
ResNet	-	Residual Network
ProtoPNet	-	Prototypical Part Network
UAV	-	Unmanned Aerial Vehicle
R-CNN	-	Region-based Convolutional Neural Network
YOLO	-	You Only Look Once
IoT	-	Internet of Things
SVM	-	Support Vector Machine
VggNet	-	Visual Geometry Group Network
API	-	Application Programming Interface
DeCaf	-	Deep Convulutional Neural Network Activation Features
RS-DeCaf	-	Reducing and Stretching
MobileNet	-	Mobile Network
CWBF	-	Chinese Wild Bird Federation
ID	-	Identification
POOF	-	Part-based One-vs-One Features
IOS	-	IPhone Operating System
UI	-	User Interface
ONNX	-	Open Neural Network Exchange
CPU	-	Central Processing Unit

GPU	-	Graphics Processing Unit
RAM	-	Random Access Memory
OS	-	Operating System
GDDR	-	Graphics Double Data Rate
HDD	-	Hard Disk Drive
SSD	-	Solid State Drive

CHAPTER 1

INTRODUCTION

1.1 Background

Sarawak, known for its historical association with head-hunters and abundant hornbill population, is the largest state in Malaysia and covers the north western region of the island of Borneo. The rainforests in this area are incredibly diverse and teeming with a wide variety of tropical wildlife [1]. Sarawak is home to a significant number of Borneo's 650 bird species, including many of the island's unique, endemic species such as the Bornean Bristlehead. The state boasts the highest number of national parks and nature reserves in Malaysia, covering over 600,000 hectares of protected areas, providing excellent birdwatching opportunities [2].

National parks in Sarawak still rely on old method using the expertise of ornithologists to identify bird's species. By old methods means Sarawak government still hired ornithologist or birds' experts to help them identify and keep count of birds manually in Sarawak every year to obtain their statistics for conservations purposes. Recent advancement of deep learning, provides state of the art solutions to help bird conservation.

According to Sarawak Forestry, as of 2019 statistics, there are 47,349 visitors 1/3 are foreigners came to Sarawak for birdwatching and as of 2021 annual shorebird count by ornithologist detected that an increase of 6.7% bird in Sarawak from the previous year and 2.4% among detected were endangered bird species [3]. Figure 1.1 shows some of the rarest birds that can only be found in Sarawak are Pink-necked Green Pigeon, Ashy Tailorbird, Mangrove Blue Flycatcher and Common Flameback [4].

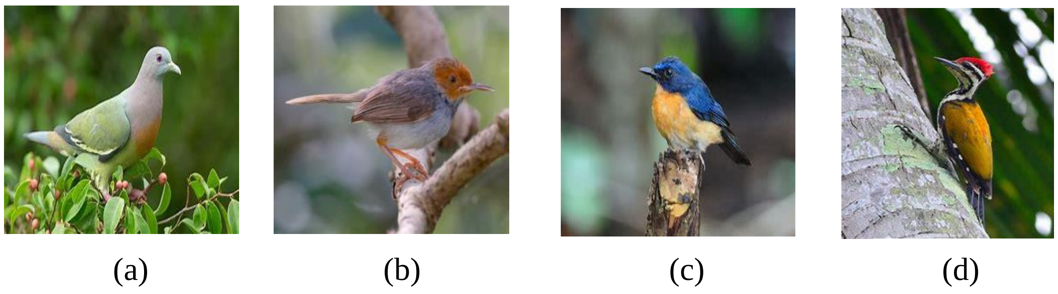


Figure 1.1 Pink-necked Green Pigeon (a), Ashy Tailorbird (b), Mangrove Blue Flycatcher (c) and Common Flameback (d) [6].

Past research in using deep learning technique such as [5], [6], [7], [8] and [9] stated that deep learning technique achieved a high accuracy and faster time in identifying and classifying bird species. This shows that deep learning is proven to be the ideal and effectives way to help identifies and classifying the bird's species. Therefore, it is a need to create a mobile application that integrate deep learning model for identifying and classifying bird species to help Sarawak Government as well as Sarawak Tourism for conservations and tourists.

1.2 Problem Statement

National parks in Sarawak still rely on old method using the expertists of ornithologists to identify bird species where current methods for bird species identification rely heavily on expert knowledge and manual observation, which can be time-consuming and subject to observer bias. This can limit the accuracy and reliability of bird species identification, and can also make it difficult to collect and analyse data on a large scale. In addition, current methods may not be practical or feasible in remote and difficult-to-access areas, where expert knowledge may not be readily available and manual observation may be challenging.

Birdwatching is a popular activity for tourists visiting Sarawak, however, many visitors have a hard time identifying the different bird species in real-time. This is due to a lack of resources available for tourists to reference, such as local guidebooks on birds found in Sarawak. With over 600 species of birds found in Sarawak, many of which are endemic to Borneo, identifying the birds can be a challenging task for visitors who are not familiar with the local avifauna. Even for experienced birders, it can be difficult to find detailed and accurate information about the birds found in the area. As a result, tourists often have to rely on their eyes and the knowledge of local guides to identify the birds they encounter in the national parks. This can make the experience of birdwatching less enjoyable and informative for tourists, as it can be difficult to learn about the different bird species and their behaviours without proper resources. In addition, it also limits the potential for tourists to contribute to citizen science or bird monitoring programs.

Deep learning has been used in recent years as a method for identifying birds, however, it is not without its limitations. One of the main limitations is that it is not a real-time process. These methods typically involve the use of pre-trained models that require a large amount of data to be fed into them in order to accurately identify birds. This process of feeding the data and running the models can take a significant amount of time, making it difficult to identify birds in real-time while observing them in the wild.

1.3 Objectives

The main objective of this research is to develop a mobile application for bird species recognition using image processing techniques and deep learning, with the goal of providing an accurate and convenient tool for birdwatchers and ornithologists to identify and classify different species of birds in real time.

The specific research objectives are to:

1. Develop a deep learning model that can classify bird species based on images.
2. Design a user-friendly mobile application that can utilised the trained deep learning model to classify birds in real time.
3. Evaluate the accuracy of the mobile application in term of its accuracy.

1.4 Summary

In summary, this research aims to develop a mobile application that uses image processing techniques and deep learning to accurately recognize and identify different bird species from photographs, and provide identification and information about the species. The application is intended to make it easier for non-experts to identify and learn about birds, and has the potential to be a valuable tool for birdwatchers, conservationists, and anyone interested in birds. The development of this application is motivated by the importance of birds in biodiversity and the need for a tool that facilitates the identification and monitoring of bird species, and has the potential to make a significant contribution to the field of ornithology and conservation.

Chapter 2

LITERATURE REVIEW

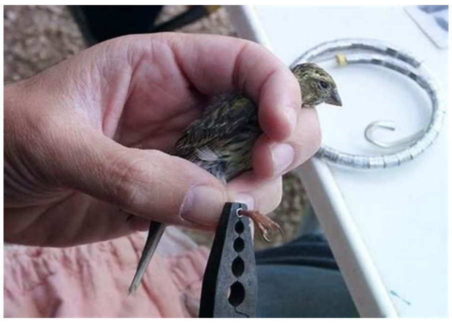
2.1 Overview

This chapter will discuss on old or traditional method in identifying bird and compare it with existing researches on bird identification using deep learning technique (CNN or Transfer Learning) and also the existing mobile application in Appstore and Google Play store. This research also discusses the research gap of the current deep learning technique as well as the existing mobile application for bird identification.

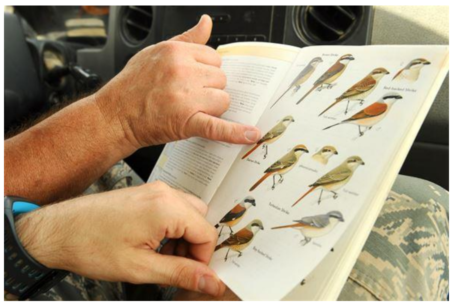
2.2 Related Studies

Many researches have been worked on deep learning specifically using transfer learning but few has applied the deep learning model into mobile applications with local Sarawak's bird dataset. Many researchers have worked on only bird recognition with many traditional diverse solutions. Table 2.1 will discuss on some of the related works on bird identification by researchers.

2.2.1 Traditional Bird Identification Method



(a)



(b)

Figure 2.1 (a) Bird tagging [10] and (b) Local Guidebook for Bird [11].

Traditional bird identification methods have been used for centuries by ornithologists and birdwatchers to identify and study bird species. These methods include using field guides, plumage, behavior, and habitat to identify birds [12].

Figure 2.1 shows bird tagging that involves attaching a small tag to a bird, typically on the leg, that contains identifying information such as a unique number or code. This allows researchers to track the movements and migrations of individual birds, which can provide valuable information about the ecology and behaviour of different species. Local book guides, on the other hand, are physical or digital guidebooks that contain detailed information about different bird species, including illustrations, descriptions, and information about their distribution, habitat, and behavior. These guides are typically focused on a specific region and can be used by birdwatchers and researchers to identify different bird species in the field.

Both of these methods have their own advantages and disadvantages. Bird tagging provides a way to track individual birds over time and can provide detailed information about their movements and habits, but it requires capturing and handling the birds. Local book guides are an efficient and easy way to identify birds in the wild, but it requires an expert to use them, and the information provided may not be as detailed as the information provided by tagging.

One of also the most common methods for identifying birds is to rely on memory and prior knowledge. Birdwatchers or birders must familiarize themselves with different species by studying birding books, magazines, and guides in order to recognize a species when they see it in the wild. This can require a significant amount of time and effort to review and learn about different birds. In the field, birds may not always be clearly visible and may only be seen briefly, which can make identification difficult.

One way for a birdwatcher to determine the identity of a bird is to ask someone who is knowledgeable about birds and is nearby or easily accessible. Another option is to take a picture of the bird and then refer to resources such as books or online guides, or seek out an expert for assistance. Alternatively, the birdwatcher can make notes and possibly draw the bird and then seek expert help at a later time.

Field guides are books that contain illustrations, photographs, and descriptions of different bird species. These guides are often organized by region, and they provide information on the size, shape, color, and markings of different bird species, as well as information on their habitat, behavior, and distribution. Field guides are a popular and widely used tool for bird identification, and they are considered to be the foundation of traditional bird identification methods.

Plumage, or the feathers of a bird, is another important aspect of traditional bird identification method. The color, pattern, and shape of a bird's feathers can provide important clues as to its identity. Birds also have different feather patterns and colors depending on their age and sex, so ornithologists and birdwatchers must take these factors into account when identifying birds by their plumage.

Behavior and habitat are also important factors in traditional bird identification methods. Birds have different behaviors and habitats depending on their species, and these can provide important clues as to their identity. For example, some birds are more active during the day, while others are more active at night. Some birds live in forests, while others live in grasslands or wetlands.

2.2.2 Deep Learning for Bird Species Identification

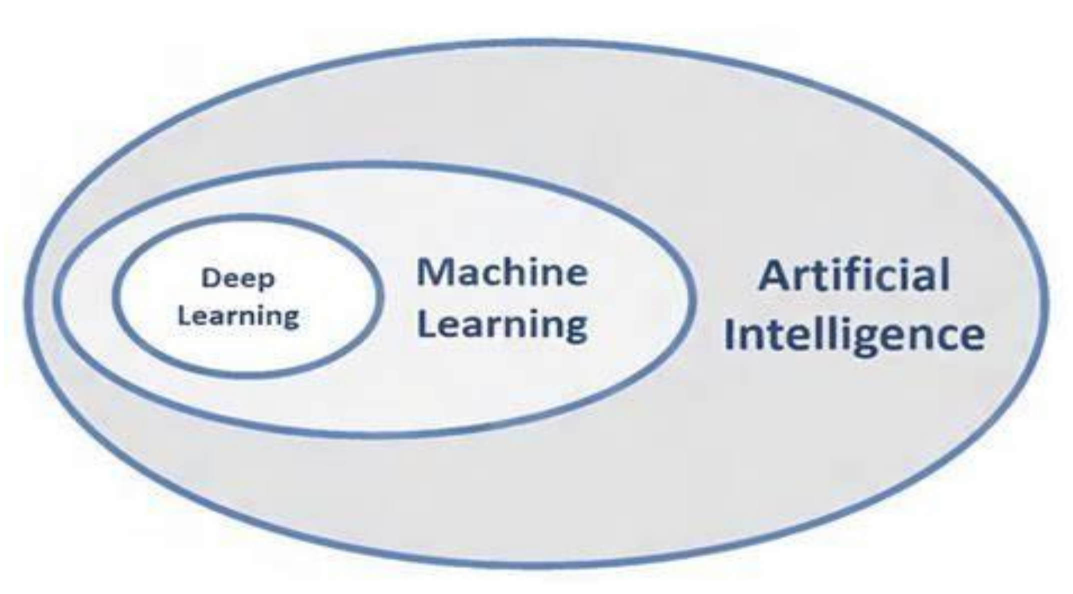


Figure 2.2 Deep Learning Subsets [13].

Based on Figure 2.2, deep learning is a type of machine learning that utilizes neural networks with three or more layers, which aim to mimic the functioning of the human brain, allowing it to learn from large amounts of data. While a neural network with a single layer can still make rough predictions, extra hidden layers can improve and refine accuracy. Deep learning is the driving force behind many AI applications and services that increase automation, enabling the execution of analytical and physical tasks without human intervention. It is the underlying technology for everyday products and services like digital assistants, voice-enabled TV remotes, and credit card fraud detection, as well as emerging technologies such as self-driving cars.

Deep learning is a branch of machine learning that stands out due to the type of data it can handle and the methods it employs for learning. Machine learning algorithms generally use structured, labelled data to make predictions, while deep learning skips pre-processing steps and can handle unstructured data such as text and images, automatically extracting features [13]. For instance, deep learning algorithms can identify which features, such as ears, are crucial to distinguish different animals, whereas in machine learning, this hierarchy of features is established manually by human experts. Through the process of gradient descent and backpropagation, the deep learning algorithm adapts and improves itself for greater accuracy, allowing it to make predictions with more precision.

2.2.2.1 Convolutional Neural Network

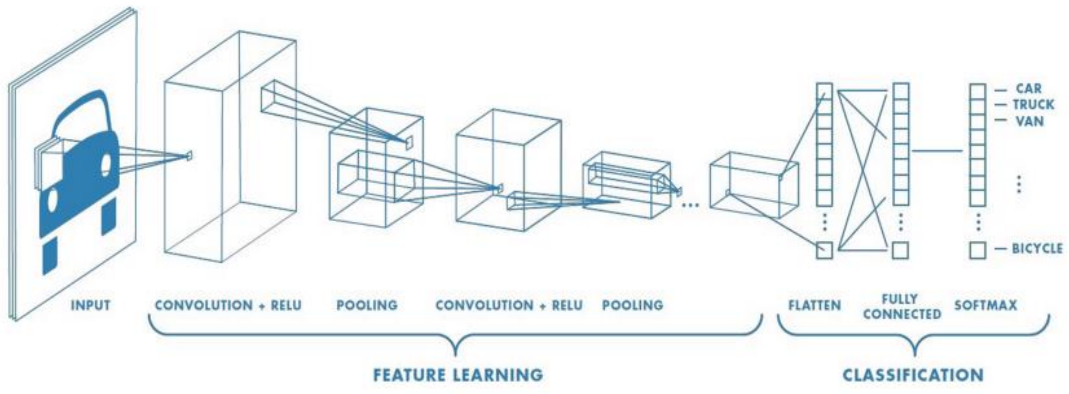


Figure 2.3 Typical CNN Architecture [14].

A standard CNN architecture comprises several layers that work in tandem to extract features from an input image. Figure 2.3 shows that these layers can be broadly divided into two categories: convolutional and fully connected layers.

The convolutional layers extract features from the input image by applying a set of filters, which are small matrices that scan the image to identify specific patterns. The output of the convolutional layer is a feature map, which is a condensed representation of the input image that highlights the features detected by the filters. The pooling layers are used to reduce the number of pixels in the image by taking the maximum or average value of a small region of the feature map, this reduces computational burden and improves the robustness of the network.

The fully connected layers are used to analyze the features extracted by the convolutional layers. There are a large number of neurons composed together that are connected to all the neurons in the previous layer. The output of the fully connected layers is a set of class scores, which are used to classify the input image into one of several predefined classes. The normalization layers are used to improve the stability of the network during training. The most common normalization technique used in CNNs is batch normalization, which normalizes the activations of the neurons across a batch of images to reduce the internal covariate shift.

Additionally, more complex models may use additional layers such as dropout, which is a technique used to prevent overfitting by randomly dropping out neurons during training, and residual layers, which aims to make the optimization process easier by allowing the network to learn the residual mapping between the input and output.

Based on Table 2.1, related studies on bird identification such as Wei et al [15] that published a paper on a method for identifying specific species of birds in images, a difficult task in computer vision known as fine-grained bird species categorization. The research used the Caltech-UCSD Birds (CUB) dataset, which includes over 5,994 images of birds from 200 different species. The authors developed the Mask-CNN method, which combines a CNN called ResNet-50 with a masking module to localize parts of the bird in the image, such as the head and wings, and select relevant descriptors for classification. The method achieved an accuracy of 80.2% on the CUB dataset, a notable result in this field where distinctions between species can be subtle and variations within a species can be large. Mask-CNN addresses these challenges and performs well on the CUB dataset.

Chen et al [16] published paper in 2019 on deep learning for interpretable image recognition, a task that involves accurately classifying images and explaining the reasoning behind the classification decisions. The research used the CUB-200-2011 dataset, which includes images of 200 different bird species, and developed the ProtoPNet method based on a combination of three neural network architectures. ProtoPNet achieved an accuracy of 84.8% on the CUB-200-2011 dataset and provided interpretable explanations for its classification decisions.

In 2019, Hong et al [17] published a paper on using deep learning methods for bird detection in images captured by unmanned aerial vehicles (UAVs). Accurate detection of birds in UAV imagery is important for a variety of applications, such as wildlife monitoring and conservation. The research used a dataset of 113,466 images self-captured by the authors using UAVs. The authors compared three different deep learning models for the task, Faster R-CNN, YOLO, and a combination of Resnet 101 and Inception v.2.

The combination of Resnet 101 and Inception v.2 achieved the best performance, with an accuracy of 95.44% on the dataset.

In 2021, Jacob et al [18] published a paper on a deep learning algorithm for image-based recognition in IoT applications. The algorithm was developed using the Caltech-UCSD Birds-200-2011 dataset, which includes over 11,000 images of 200 bird species. The authors used a pre-trained SVM in the development of the algorithm, which achieved an accuracy of 92.5% on the dataset while in 2020, Satyam Raj [19] published a paper on using CNNs for identifying bird species in images. The research used a dataset of 8218 images of 60 different bird species obtained using Microsoft's Bing Image Search API v7 and developed the VGGNet CNN for the task. The model achieved an accuracy of 93.19% on the dataset, showing the effectiveness of CNNs in image-based bird species identification.

In 2021, Huang et al [5] published a paper on using deep learning models for recognizing endemic bird species. The research used a dataset of 31,320 images from the China Wildlife and Nature Federation and BirdLife International, and developed the Inception-ResNet-v2 model for the task. The model achieved an accuracy of 97.90% on the dataset compared to in 2018, Niemi et al [9] published a paper on deep learning for automatic bird identification. The research used a dataset of 9,312 images collected in Finland and developed a deep learning model based on an SVM classifier. The model achieved an accuracy of 94.63% on the dataset.

2.2.2.2 Transfer learning

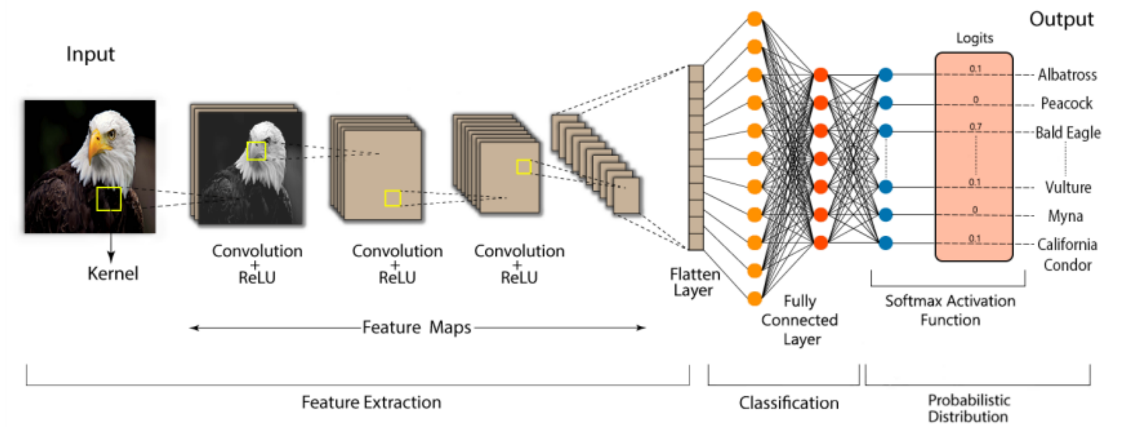


Figure 2.4 Architecture of Transfer Learning [20].

Figure 2.4 shows that transfer learning involves using a model that has already been trained on a task as a starting point for a new, related task [20]. This allows the new model to take advantage of the knowledge gained from the original training, potentially leading to better performance. Transfer learning can be particularly useful when there is a limited amount of data available for the new task. In deep learning, transfer learning has been successful in achieving top results for a range of tasks, such as image recognition, speech recognition, and natural language processing. All the researchers here used transfer learning method which in 2018, Zhong et al [21] published a paper on using DeCAFs for image classification. The research used the Caltech-UCSD birds' dataset and developed the RS-DeCAF method for reducing and stretching DeCAFs. The RS-DeCAF method was used with transfer learning and an AlexNet model and achieved an accuracy of 35.31% on the dataset. While this result demonstrates the potential of DeCAFs for image classification, the performance of the RS-DeCAF method was lower than other methods.

Srijan et al (2021) [20] conduct research focused on mobile application for bird species identification using transfer learning. The application was trained on a dataset from Kaggle that includes 40,000 images of 260 different bird species. The authors used the efficientNet-Lite architecture and transfer learning to train the model. The results of their study showed that the mobile application achieved an accuracy of 98.61% in identifying bird species. The use of transfer learning in this study allowed the authors to take advantage of pre-trained models, which reduced the amount of data and computational resources needed to train the model, making it feasible to deploy the model on a mobile device.

In 2020, Rahman et al [7] published a paper on using deep learning to recognize local birds in Bangladesh. The research used a dataset of 3500 images of local birds in Bangladesh and developed the Inception-v3 and MobileNet models for the task. Transfer learning was used to improve the performance of the models, which achieved an accuracy of 91.00% on the dataset while in 2020, Ragib et al [22] published a paper on using deep learning to automatically identify bird species. The research used a dataset of 14,000 images and developed the ResNet-101 model for the task. Transfer learning was used to improve the performance of the model, which achieved an accuracy of 97.98% on the dataset.

In 2021, Al-Showarah et al [6] published a paper on using deep learning to identify birds. The research used a dataset of 4340 images of 434 bird species from the Jordanian

Bird Watching Association and developed the VGG19 model for the task. Transfer learning was used to improve the performance of the model, which achieved an accuracy of 71% on the dataset compare to in 2019, Boudaoud et al [23] published a paper on using deep learning to detect marine birds in high-resolution aerial images. The research used a dataset of 12,352 images and developed a CNN-based model for the task. Transfer learning was used with a pre-trained model to improve the performance of the model, which achieved an accuracy of 95.18% on the dataset. In 2021, Wu et al [24] published a paper on using deep transfer learning for bird classification. The research used the CUB-200-2011 dataset of 11,788 images and developed the ResNeXt model for the task. Transfer learning was used to improve the performance of the model, which achieved an accuracy of 84.43% on the dataset.

2.2.2.3 Dataset

Datasets in deep learning are groups of data utilized to teach, assess, and examine a machine learning model, such as a deep neural network. The data in a dataset generally includes input and output pairs, where the input is a set of features or attributes and the output is the related label or category [25].

The significance of a dataset in deep learning is that it enables training a model that can generalize well to new, unseen data. In simpler terms, a good dataset will have a wide variety of examples that cover the full spectrum of possible inputs and outputs, allowing the model to make accurate predictions on new, unseen data. Additionally, a large and diverse dataset can also aid in preventing overfitting, a common issue in deep learning where a model becomes too specific to the training data and performs inadequately on new data [25].

Most of the researcher and journals reviewed such as [16], [15], [21], [18] and [24] use dataset obtains from CUB-200-2011 dataset, developed by Caltech and UCSD. The dataset is a collection of images of birds, comprising of more than 11,000 images of 200 distinct bird species. The dataset is intended for use in fine-grained visual classification tasks, such as determining the particular breed of a bird in an image. The images in the dataset have been gathered from the internet, and are varied in terms of perspective, scale, and background. The images have also been manually labeled with bounding boxes around the birds and labels indicating the species.

Additionally, each image has a set of attributes, such as the presence of specific body parts of the bird (e.g., crest, tail, etc.) which can be utilized as supplementary features for training models. It is a challenging dataset due to the high variability of the images, which makes it a good test-bed for developing and evaluating new image recognition algorithms. The dataset is also widely used as a benchmark for comparing the performance of different deep learning models.

Researcher in [19] uses Microsoft's Bing Image Search API v7 that contain 60 species with 8218 images are used in combination with deep learning techniques to create a bird identification system. The API are used to search the internet for images of birds, and to retrieve the search results in the form of image URLs, metadata, and thumbnails. While researcher [26] used dataset from JONATHAN specifically for fine-grained bird classification. It is a large-scale dataset of annotated images of birds, which can be used to train deep learning models for the task of fine-grained bird classification. The dataset includes over 200,000 images of birds belonging to 600 different species. Both the dataset is better in terms of large number of images and variety of bird images but since the dataset is from the web the quality of some images may be poor.

Some researcher [17], [7], and [9], capture and collect their own dataset of images of birds with 113,466 images, 3500 images and 9,312 images respectively. The bird’s images collected were from the researcher’s local birds to cater for specific birds in their own regions respectively. So, from the journal self-collected images the researchers actually have control over data collection, types of images that are included, as well as the annotation process. This allows researchers to ensure that the dataset is tailored to the specific task of bird identification and that the data is of high quality but this process is time-consuming as collecting and annotating a dataset can be a time-consuming process, which can delay the development of a deep learning model.

Table 2.1 Related Studies on Bird Identification using Deep Learning Technique

Ref	**Research Focus**	**Dataset/ Database**	**Size**	**Learning**	**Architecture/ Model**	**Accuracy**
Srijan et al, 2021 [20]	Mobile Application for Bird Species Identification Using Transfer Learning	Kaggle	260 bird species, 40,000 images	Transfer Learning	EfficientNet-Lite	98.61%
Wei et al, 2017 [15]	Localizing Parts and Selecting Descriptors for Fine-Grained Bird Species Categorization	Caltech-UCSD Birds (CUB)	5,994 images	Deep Learning	Mask-CNN ResNet-50	80.2%
Chen et al, 2019 [16]	Deep Learning for Interpretable Image Recognition	CUB-200-2011	200 species	Deep Learning	ProtoPNet (VGG19 +ResNet34 +DenseNet121-based)	84.8%

Hong et al, 2019 [17]	Application of Deep-Learning Methods to Bird Detection Using Unmanned Aerial Vehicle Imagery	Self-captured	113,466 images	Deep Learning	Faster R-CNN YOLO Resnet 101 Inception v.2	95.44%
Zhong et al, 2018 [21]	Reducing and Stretching Deep Convolutional Activation Features for Accurate Image Classification	Caltech-UCSD birds	5,994 images	Transfer Learning	RS-DeCAF AlexNet	35.31%
Rahman et al, 2020 [7]	Recognition of Local Birds of Bangladesh using MobileNet and Inception-v3	Bangladesh local birds	3500 images,	Transfer Learning	Inception-v3, MobileNet	91.00%
Jacob et al, 2021 [18]	Design of Deep Learning Algorithm for IoT Application by Image based Recognition	Caltech-UCSD Birds-200-2011	11,000 images	Deep Learning	Pre-Trained SVM	92.5%
Satyam Raj, 2020 [19]	Image based Bird Species Identification using Convolutional Neural Network	Microsoft's Bing Image Search API v7	60 species, 8218 images	Deep Learning	VGGNet	93.19%

Ragib et al, 2020 [27]	Automatic Bird Species Identification Using Deep Learning		14000 images	Transfer Learning	ResNet-101	97.98%
Al-Showarah et al, 2021 [6]	Birds Identification System using Deep Learning	Jordanian Bird Watching Association	4340 images, 434 bird species	Transfer Learning	VGG19	71%
Huang et al, 2021 [5]	Recognition of Endemic Bird Species Using Deep Learning Models	CWBF, BirdLife International	31,320 images	Deep Learning	Inception-ResNet-v2	97.90%
Boudaoud et al, 2019 [23]	Marine Bird Detection Based on Deep Learning using High-Resolution Aerial Images	JONATHAN Training	12,352 images	Transfer Learning	CNN Based, Pre-trained model	95,18%
Wu et al, 2021 [24]	Birds Classification Based on Deep Transfer Learning	CUB-200-2011	11,788 images	Transfer Learning	ResNeXt	84.43%
Niemi et al, 2018 [9]	Deep Learning Case Study for Automatic Bird Identification	Manually collected data at site (Finland)	9,312 images	Deep Learning	SVM classifier	94.63%

2.3 Existing Bird Identification Mobile Applications

This research study examined 5 widely available bird identification mobile apps, Merlin Bird ID, National Geographic Birds, Peterson Birds, iBird Pro and Audobon Birds as shown in Table 2.2. The development of Merlin Bird ID involved the contribution of experts, Amazon Mechanical Turk workers, and over 500 citizen scientists, who helped to annotate the dataset used by the app [28]. The dataset included partial annotations and bounding boxes. The app also uses a "human-in-the-loop" approach, in which it provides users with an identification result based on input from human experts.

Partial annotation is a costly process that can be time-consuming and labour-intensive, and it does not necessarily result in a dataset that is completely error-free. Novice annotators may make mistakes, but if the annotators are knowledgeable in the subject matter, the dataset may be more accurate. "Human-in-the-loop" methods, which involve input from human experts, can reduce the user experience because they require users to take additional steps to obtain a result .

National Geographic Birds uses a different type of classifier than the commonly used "one-vs-all" classifier [29]. The "one-vs-most" classifier used by National Geographic Birds compares the input image to the species that is most likely to be a match, rather than all species. National Geographic Birds employs Part-based one-vs-one Features (POOFs) and Support Vector Machines (SVMs) for image classification. However, Convolutional Neural Networks (CNNs) have been found to be more effective for image classification than SVMs. Like, Merlin Bird ID, National Geographic Birds also uses a dataset with bounding boxes and part annotations.

Peterson Birds is a widely used mobile application for bird identification created by ornithologist Roger Tory Peterson [30] while Audubon Birds is based on the Audubon field guide, a widely recognized and respected guide to bird identification [31]. Both the app, includes illustrations, descriptions, and information about the distribution, habitat, and behaviour of different bird species, and is organized by colour, shape, and size, making it easy for users to identify birds in the field. One of the benefits of the apps are their recognition and long history of helping people identify birds. The app is also easy to use and navigate, making it easy for users to find the information they need. Furthermore, its portable and accessible nature makes it convenient for users who are in the field. However, both the app may require internet connection to access its information which might not be accessible in some areas, and it may not be updated regularly, lacking the most recent information about bird species.

iBird Pro is a mobile application that provides a comprehensive guide for identifying over 940 bird species found in North America [32]. The app features detailed illustrations, descriptions, and information on the distribution, habitat, and behaviour of different bird species. It also includes additional features such as bird calls, search filters, and quizzes, which assist users in identifying birds in the field. One of the benefits of using iBird Pro is the large number of bird species included in the guide, making it a valuable resource for bird watchers, naturalists, and researchers. Furthermore, the app includes features such as bird calls and quizzes that can help users improve their bird identification skills. However, a disadvantage of the app is that it focuses only on North American bird species, making it not useful for users in other regions. Additionally, certain features of the app may require internet connection which may not be available in some areas.

All the bird app mentioned uses only the dataset for their own respective region which is mainly in the Europe and North-America region. In this research, it focuses specifically on the local bird here in Sarawak which will uses a local bird dataset and specifically identify and classes the birds accordingly based on the local birds from Sarawak.

Table 2.2 Existing Mobile Applications for Bird Identification

Ref.	**Application**	**Dataset Size**	**Ease of use**	**Platform availability**	**Accuracy**
[28]	Merlin Bird ID	400 common species in North America	Search Time - Approximately 7 seconds	IOS only	Approximately 95.00%

[29]	National Geographic Birds	860 species	Search Time - Approximately 5 seconds	IOS & Android	Approximately 90.00%
[30]	Peterson Birds	800+ species	Search Time - Approximately 5 seconds	IOS & Android	Approximately 93.00%
[32]	iBird Pro	946 species	Search Time - Instant search	IOS only	Approximately 95.00%
[31]	Audobon Birds	760+ species	Search Time - Approximately 3 seconds	IOS & Android	Approximately 98.00%

2.4 Research gap

The existing literature on bird image classification and deep learning is extensive, with deep learning techniques achieving impressive results on a variety of image classification tasks. However, most of the existing research has focused on traditional computer-based approaches, with relatively little attention given to mobile-based solutions. This is a significant gap, as mobile devices are becoming increasingly powerful and prevalent, and there is a growing demand for accessible and user-friendly tools that can be used in the field.

The current existing researches also contain mostly dataset from the Europe and North America region which is not suitable since this research focus is on local birds from Sarawak. Thus, the existing deep learning models are not suitable to train local bird dataset on. Besides, the existing study are mostly using a computer-based software to evaluate the accuracy and identified the bird species on. This is not a real-time approach for birder and tourist to rely on when identifying the bird species on the field.

2.5 Summary

The literature review chapter in this report on bird species recognition utilizing image processing techniques and deep learning give a summary of the current research in this field. This chapter start by addressing related studies, including any prior work on mobile applications for bird species recognition, and any related studies on image processing techniques and deep learning. The chapter then explore traditional bird identification methods and their shortcomings. Additionally, it also discusses the benefits that image processing techniques and deep learning bring to the field of bird species recognition. This literature review also delves into the specific methods employed in your research, such as convolutional neural networks and transfer learning. Related studies also provide a summary of the dataset used in the research, including how it was obtained and preprocessed. Last but not least, the chapter summarize the key findings of the literature review and how they relate to the research question or problem statement of the project that include a discussion of the gaps in existing research that this study aims to fill.

Chapter 3

METHODOLOGY

3.1 Overview

This chapter discusses the methods that need to be done to achieve the objectives stated in chapter 1. The chapter delves into the conceptual framework of the mobile app, outlining how it will work. Additionally, it covers the methodology, which includes the dataset used and how it was obtained, as well as the size of the data related to the local birds of Sarawak.

Furthermore, the chapter explores the data pre-processing techniques employed to process the dataset effectively. It also describes the deep learning model used, incorporating transfer learning, and outlines the process of designing and training the model. An essential part of this chapter involves the steps taken to convert the model into a lighter version suitable for the mobile app, utilizing TensorFlow and TensorFlow-Lite.

Moreover, the chapter delves into the design process of the User Interface (UI) of the mobile app, ensuring an optimal user experience. It also addresses the process of integrating and implementing the model into the mobile app. The evaluation of the mobile app is a significant aspect, where its accuracy is assessed.

Last but not least, this chapter provides insights into the hardware and software involved in the entire process, along with their specifications. Understanding these aspects is crucial for comprehending the comprehensive setup that underlies the development of the mobile app and its deep learning model.

3.2 Conceptual Framework

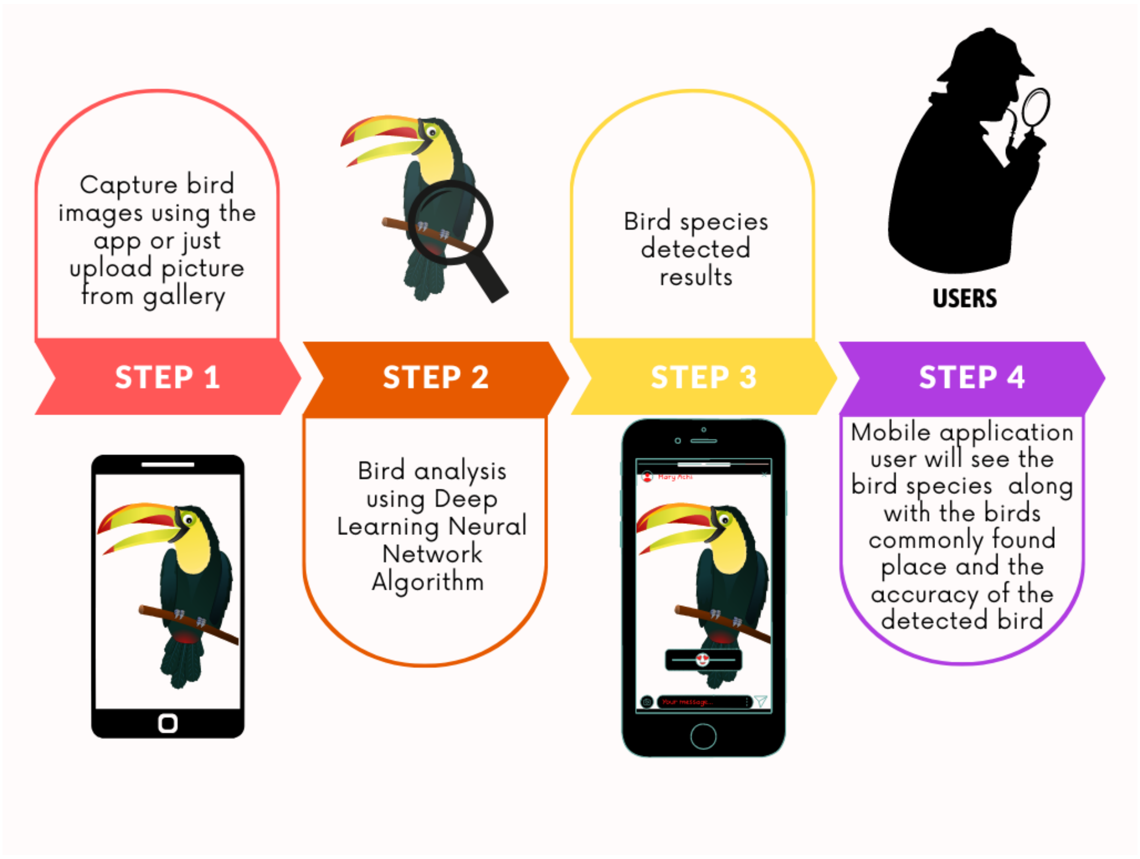

Capture bird images using the app or just upload picture from gallery
STEP 1
STEP 2
Bird analysis using Deep Learning Neural Network Algorithm
Bird species detected results
STEP 3
USERS
STEP 4
Mobile application user will see the bird species along with the birds commonly found place and the accuracy of the detected bird

Figure 3.1 Conceptual Framework of the study

Based on Figure 3.1, a conceptual framework was used to develop a mobile application for bird species recognition. The mobile application allows users to capture bird images using their phone's camera or upload images from their phone's library. The captured images are then processed and analysed using a deep learning neural network algorithm implemented in the mobile application. The algorithm determines the species of the bird based on the cross entropy, train accuracy, and validation accuracy of the pixels in the image. The output of the analysis, based on the trained model, displays the identified bird species, its scientific name, and a recommended location where the bird is commonly found.

3.3 Methodology

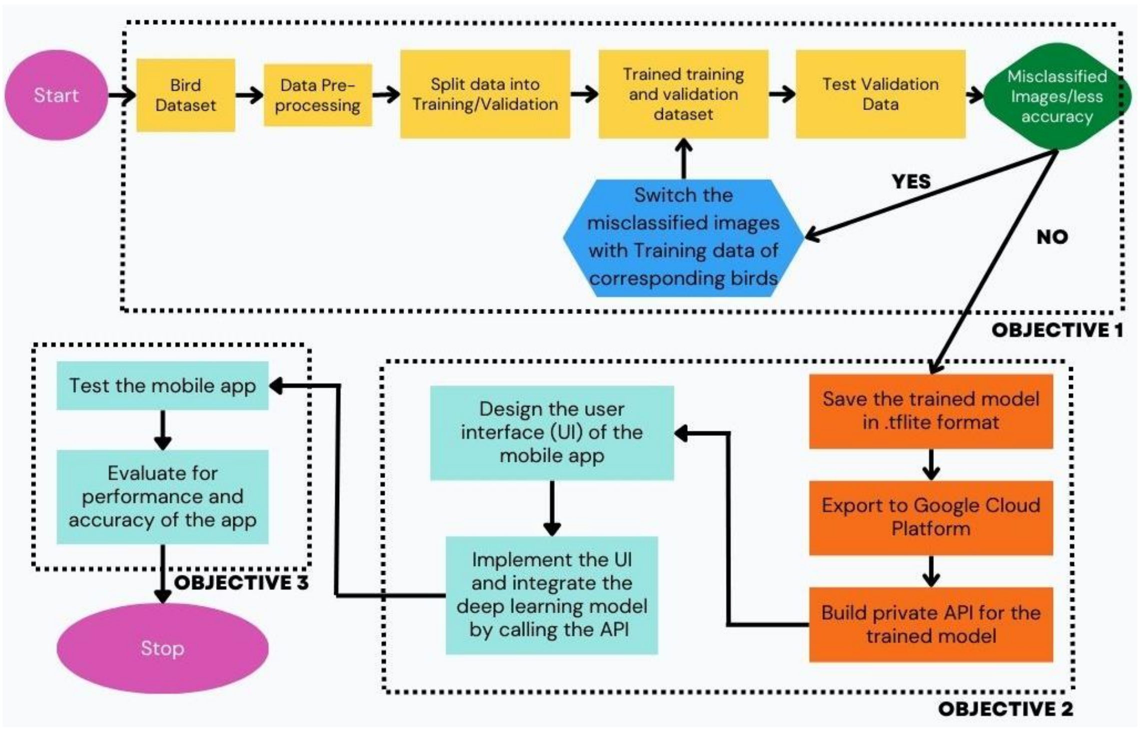

Start
Bird Dataset
Data Pre-processing
Split data into Training/Validation
Trained training and validation dataset
Test Validation Data
Misclassified Images/less accuracy
YES
NO
Switch the misclassified images with Training data of corresponding birds
OBJECTIVE 1
Save the trained model in .tflite format
Export to Google Cloud Platform
Build private API for the trained model
Design the user interface (UI) of the mobile app
Implement the UI and integrate the deep learning model by calling the API
OBJECTIVE 2
Test the mobile app
Evaluate for performance and accuracy of the app
OBJECTIVE 3
Stop

Figure 3.2 Overall Flowchart of System

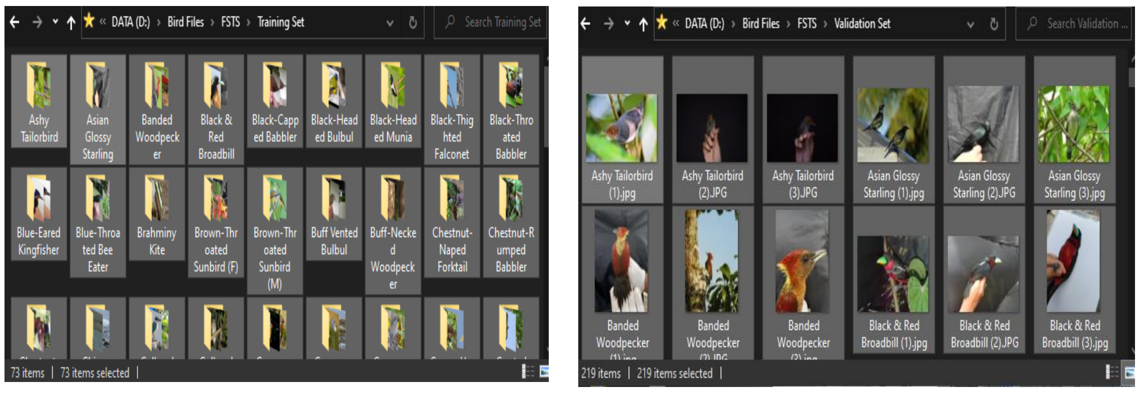
Figure 3.2 shows the flowchart and the process in development of a mobile app for bird identification that starts with acquiring a dataset of bird pictures. This dataset requires cleaning and organizing through a pre-processing phase. After pre-processing, the data is divided into training and validation datasets. The training dataset is then used to train the deep learning model. It is crucial to evaluate the model's performance by checking for any incorrectly classified images or low accuracy. In case of misclassified images or low accuracy, the solution is to replace those misclassified images with the training data of the corresponding birds to enhance the model's performance.

When the model performs well on the validation dataset, the next step is to design the user interface (UI) of the mobile app which includes creating an easy to use and user-friendly layout for the app, as well as designing various elements such as buttons and text fields. After the UI design, it must be implemented and integrated with the deep learning model by connecting the UI elements with the functionality of the model.

Next, the model that has been trained in Jupyter Lab is exported to Google Cloud Platform using Vertex AI for further optimization of the model where the model will undergo optimisation process such as quantisation, pruning, weight sharing and model distillation. Since this model will be used for real time application which is the mobile application, optimisation is important to improved performance, reduced model size, and efficient deployment. After that, Application Programming Interface (API) is created to be the bridge between the model in the cloud and the mobile app in Android Studio. API is used to receive request from the mobile app and send it to the cloud where the model is stored to receive and process the prediction from the app.

Finally, the mobile app must be tested for accuracy and bugs. This includes testing the app with various bird images and ensuring that it correctly identifies the birds in the images. It also includes checking for any bugs or crashes in the app.

3.3.1 Bird Dataset



(a) (b)

Figure 3.3 Training sets (a) and Validation set (b) of local bird species.

In this project, the dataset used are obtained from a lecturer, Dr Mohamad Fizl Sidq bin Ramji from Faculty of Resource Science and Technology [33]. From Figure 3.3 the dataset provided contain a total of 1878 images of birds from 30 species of local birds in Sarawak. All the images are in .jpg format and categorise in separate folders for each species. Each species of birds contains at least 60 to 80 images in each separate folder. The dataset then separated into training and validation set with 80:20 split which 1503 files for training and 375 files for validation.

3.3.2 Data Pre-Processing

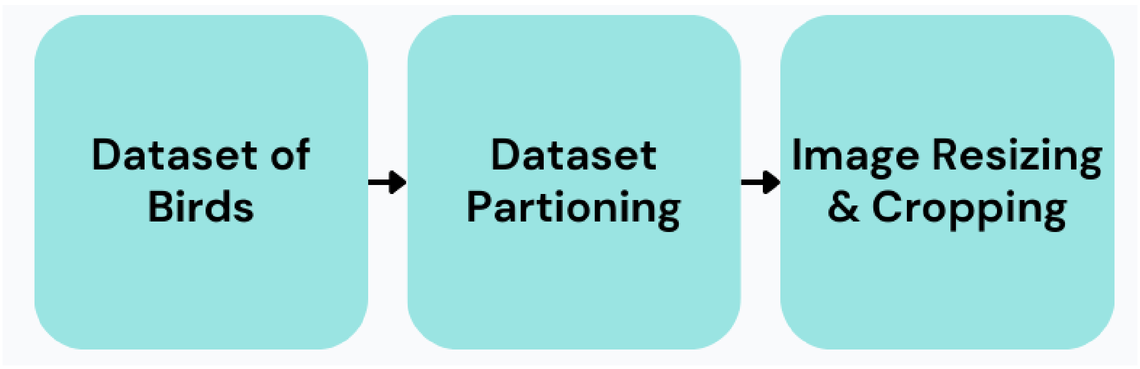


Figure 3.4 Dataset Pre-Processing Process

Figure 3.4 shows the process involved in dataset pre-processing process such as data partitioning where it is a method of dividing a dataset into three distinct groups, training, validation, and testing sets. The main goal of this technique is to prevent overfitting, which happens when a model becomes too intricate and is unable to generalize to new data. By utilizing a separate validation set, the model's performance can be examined on unseen data, which provides a more precise evaluation of its capabilities. The standard split is 80:20, where 80% of the data is used for training, 20% for validation and testing. This technique also allows for evaluating the deep learning model's performance, as the test set is used to measure its ability to generalize to new data. In conclusion, data partitioning is an essential step in the development of deep learning models and helps to ensure robustness and reliability.

Adjusting the dimensions of an image is known as image resizing. It can be used to make an image bigger or smaller. In the context of deep learning, image resizing is frequently used to make the size of images consistent before it can be used in the model. For instance, in this research images are converted to 256x256 pixels, all images used for training and testing must be adjusted to this size. This is significant as deep learning models usually require a fixed input size and adjusting images to a consistent size allows the model to process the images more effectively. The process of image resizing is accomplished using image editing software OpenCV.

3.3.3 Deep Learning Model Design

Transfer learning is a method in which a pre-trained model is used as a starting point for a new, but related, task. The pre-trained model provides a foundation of knowledge that can be utilized in the new domain. There are different options for implementing transfer learning, such as feature transfer and fine-tuning, depending on the relationship between the tasks. Another option is to keep certain layers of the network unchanged and retrain others [34].

The process of designing a deep learning model utilizing transfer learning encompasses several essential stages. Initially, the required libraries and dependencies, such as TensorFlow or PyTorch, are imported into the Python environment, commonly utilizing Jupyter Lab as an interactive development platform. These libraries provide the necessary tools and functions for constructing and training deep neural networks.

Subsequently, a pre-trained model is loaded into the framework. Renowned pre-trained models such as VGG16, ResNet, MobileNet, EffifientNet and Inception are readily available in deep learning frameworks like TensorFlow and PyTorch. These models have undergone training on extensive datasets, often comprising millions of images, enabling them to capture intricate and meaningful representations of visual patterns.

To ensure that the pre-trained layers remain unaltered during the training process, the model are frozen. By freezing the pre-trained layers, the learned weights and patterns are preserved and not modified throughout the model's training. This step is critical in transfer learning as it enables the newly added layers to concentrate on learning task-specific features without interference from the pre-trained layers.

Once the pre-trained layers are frozen, custom layers are incorporated atop the pre-trained model. These layers are responsible for learning the task-specific features that are relevant to the objective at hand. For instance, one may add a Global Average Pooling layer followed by a Dense layer. The Global Average Pooling layer consolidates spatial information across the output feature maps, thereby reducing the dimensionality of the data. Subsequently, the Dense layer offers the flexibility to learn non-linear relationships and capture intricate patterns within the data.

Following the addition of custom layers, a new model is constructed by connecting the input and output tensors of the pre-trained model to the input and output tensors of the custom layers. This resulting model combines the pre-trained layers' capability for feature extraction with the task-specific layers' aptitude for learning and categorizing the data [35].

3.3.3.1 Designing Deep Learning using Transfer Learning

The first step in designing a deep learning model with transfer learning in python on Jupyter Lab is to load the pre-trained model using the "import" function from the TensorFlow library. This function will return a pre-trained CNN model that can use as a starting point for the new model.

The next step is to replace the final fully connected layers of the pre-trained model with new layers that are suitable for the dataset. In this case, since the dataset contain 30 species with 30 classes, the final layer needs to be replaced with a new fully connected layer with 30 outputs. This step allows the model to adapt to the new task and dataset.

Once the layers are replaced, the weights of the pre-trained layers need to be freeze so it will not be updated during training. This step is important because the pre-trained layers already contain useful information and so that the model will not overwrite it.

After that, the model be train on the new dataset using the "trainNetwork" function. In this research, data portioning and image resizing will be performed to improve the performance of the model. Once the model is trained, the model will be fine-tune by unfreezing some of the pre-trained layers and training them on the new dataset. This step allows for further adaption of the model to the new task.

Finally, the model will be tested on new data to evaluate its performance. Various metrics such as training time, training loss, validation loss, training accuracy, validation accuracy, test loss, and test accuracy.

3.3.3.2 TensorFlow and TensorFlow-lite

After the deep learning model is trained and fine-tuned in MATLAB, the model file is converted to TensorFlow or TensorFlow Lite for deployment on Android platform.

In converting the model to TensorFlow, "exportONNXNetwork" function is used from the Deep Learning Toolbox to export the model to the ONNX format. ONNX (Open Neural Network Exchange) is an open format for representing deep learning models that allows for interoperability between different frameworks. Once the model is in ONNX format, the TensorFlow's ONNX converter is used to import the model into TensorFlow. The ONNX converter provides a python script that can be used to convert the ONNX model to TensorFlow's protobuf format.

After the model is in TensorFlow format, TensorFlow Lite is used to convert it to a format that is optimized for mobile and embedded devices. TensorFlow Lite is a lightweight version of TensorFlow that is designed for mobile and embedded devices. It allows TensorFlow models to run on devices with limited resources such as smartphones and IoT devices. So, to convert the model to TensorFlow Lite, TensorFlow Lite converter need to used. The TensorFlow Lite converter provides a python script that can be used to convert the TensorFlow model to TensorFlow Lite format. TensorFlow Lite's command-line tool also can be used to convert the model directly from the command line.

3.3.4 Integration and Implementation of Deep Learning Model

The first step to implement and integrate the TensorFlow Lite model into a mobile application is to add the TensorFlow Lite for mobile library into the mobile application. This can be done by including the TensorFlow Lite library in the application's build.gradle file for Android or Podfile for iOS. This will allow the TensorFlow Lite APIs to be used in the mobile application, which provides a set of APIs that allow the TensorFlow Lite models to run on mobile and embedded devices.

The next step is to load the TensorFlow Lite model into the application. This can be done by reading the model file and creating a TensorFlow Lite interpreter object that can run the model. The interpreter object provides an easy-to-use interface to run the model on input data. Once the model is loaded, the input data need to be prepared for the model. This includes converting the data into the format that the model expects and scaling the data if necessary. The input data should match the input tensor shape and type that the model is expecting.

After the input data is prepared, the TensorFlow Lite interpreter object can be used to run the model on the input data. This will produce the model's output, which is a set of tensors. The next step is to process the output tensors to make them usable in the mobile application. This may include converting the output to a format that is easy to use, or mapping the output to a specific action or result in the application. The output tensors should be processed based on the expected output of the model.

Finally, after the above steps have been implemented, the model can be integrated into the mobile application by triggering the model's execution in response to user input or other events. It's important to consider the performance and energy consumption of the model when integrating it into the mobile application.

3.3.5 Model Optimisation in Google Cloud Platform using Vertex AI

The Model Optimization Toolkit provided by Vertex AI on Google Cloud Platform offers a comprehensive range of techniques and tools to optimize machine learning models. The goal is to improve model performance, reduce model size, and enable efficient deployment. The toolkit consists of various components that address different aspects of model optimization.

One important technique available in the toolkit is quantization. Quantization involves converting the high-precision floating-point numbers in a model into lower-precision fixed-point or integer representations. This reduction in the number of bits used to represent numbers significantly decreases the memory requirements of the model and enables faster inference on hardware accelerators like TPUs.

Pruning is another essential optimization technique. It involves identifying and removing redundant connections, weights, or neurons from a trained model. During the training process, neural networks often contain redundant components that have minimal impact on overall performance. By eliminating these unnecessary components, pruning produces a sparser model that requires fewer computations during inference, resulting in improved efficiency.

Weight sharing is a technique included in the Model Optimization Toolkit that reduces the number of unique weights in a model. Similar weights are grouped together and stored as shared parameters, thereby reducing the memory footprint of the model. This technique is particularly effective for convolutional neural networks (CNNs) due to the spatial redundancies present in their weight tensors.

Model distillation is an optimization approach available in the toolkit, which involves transferring knowledge from a larger, more complex model (teacher model) to a smaller, more efficient model (student model). The student model is trained to mimic the behaviour and predictions of the teacher model, allowing it to achieve comparable performance while being more lightweight and suitable for deployment on devices with limited resources.

To support the deployment of optimized models on mobile and edge devices, the Model Optimization Toolkit utilizes TensorFlow Lite. TensorFlow Lite is a framework specifically designed for deploying machine learning models on mobile and embedded platforms. It incorporates optimizations such as quantization, weight quantization, and hardware acceleration, further enhancing the efficiency and performance of models in these deployment scenarios.

3.3.6 User Interface (UI) for Mobile Application

Recently, mobile technology has been increasingly utilized in wildlife conservation, with a variety of mobile apps being developed to help identify and track different bird species. One such app is a bird identification tool that uses deep learning to aid users in recognizing birds by their visual features. The application's user interface (UI) is crucial to its success, as it must be easy to use and understand in order to effectively guide users in identifying birds.

Designing the User Interface (UI) for a mobile application that integrates a machine learning model in Android Studio encompasses several essential steps. Initially, the Android Studio project is established by creating a new project and specifying details like the application name, package name, and minimum Android version. Depending on the requirements, one can opt for a project template or start from scratch with a blank activity.

Subsequently, the main UI of the application is designed, taking into account the necessary components for user interaction and displaying the machine learning model's output. These components can include buttons, text views, image views, and input fields. The layout and organization of these elements are carefully planned to ensure an intuitive and user-friendly interface.

Once the UI design is finalized, the machine learning model is integrated into the application. This involves importing the required libraries and dependencies specific to the machine learning model into the Android project. The model is loaded and initialized within the application code. Strategies are determined for processing and feeding user input into the model for inference or prediction. The logic for invoking the model and obtaining the results or predictions is established.

Handling user input is a critical aspect of UI design. Event handlers for UI components are implemented to capture user input effectively. The UI is designed to collect the necessary information for the model, such as text input, image capture, or file upload. Preprocessing of user input data is conducted to ensure compatibility with the model's input requirements. It is crucial to validate and format user input correctly before passing it to the model for processing.

The presentation of the machine learning model's output to the user is another significant consideration in UI design. The UI is designed to display the output in a user-friendly format, which can include text, images, graphs, or visualizations. The formatting and presentation of the output data are tailored to the specific requirements of the application. Suitable UI components are employed to enhance the visual representation of the results.

Optimizing the UI and performance of the application is vital for delivering a smooth user experience. The performance impact of running the machine learning model on a mobile device is taken into account. Strategies are implemented to optimize the UI and code to ensure responsiveness and minimize processing time. Techniques such as asynchronous processing or background threads may be employed to prevent UI freezing during model execution. Thorough testing is performed on various Android devices and screen sizes to ensure compatibility and optimal performance of the application.

3.3.7 Evaluate Mobile Application Accuracy

After implementing and integrating the trained deep learning model into the mobile application, the mobile app is then move into testing phase. To evaluate the accuracy of the mobile application designed for bird species identification, a comprehensive assessment was carried out using a specific collection of 30 images showcasing various local bird species. The primary objective of this evaluation was to determine the application's ability to correctly recognize and identify the bird species depicted in the provided images.

The evaluation procedure involved in putting the 30 test images into the mobile application and recording the species identifications generated by the application for each respective image. Subsequently, a manual assessment was conducted by comparing the application's identifications with the known bird species present in the images.

In order to quantify the accuracy of the test, the number of correct identifications made by the application was divided by the total number of test images. To express the accuracy as a percentage, the resulting ratio was then multiplied by 100. For example, if the application accurately identified 25 out of the 30 test images, the accuracy would be calculated as (25/30) * 100 = 83.33%.

Through this evaluation process, a quantitative measure of the mobile application's accuracy in identifying local bird species was obtained. The average accuracy rate derived from this evaluation serves as a significant metric for assessing the application's performance and reliability in real-world scenarios.

It is important to acknowledge that this evaluation was conducted using a specific set of test images, and the accuracy of the application may vary when presented with different images or bird species. To gain a more comprehensive understanding of the application's accuracy and its potential for broader use, further evaluations employing diverse datasets and a larger sample size would be beneficial.

3.4 System Hardware & Software

As for the system hardware, all simulations and the training of model is on a laptop ASUS ROG GL552VW with specifications of Central Processing Unit (CPU) is Intel Core i7-6700HQ, Graphics Processing Unit (GPU) is NVIDIA GeForce GTX 960M (2GB GDDR5), Random Access Memory (RAM) is 16GB DDR4, with a storage of 128GB SSD + 1000GB HDD. For the mobile hardware where the app will be deployed on Huawei nova 5T with specifications of Operating System (OS) is Android 9.0 (Pie), upgradable to Android 10, EMUI 10, chipset is Kirin 980 (7 nm), Central Processing Unit (CPU) is Octa-core (2x2.6 GHz Cortex-A76 & 2x1.92 GHz Cortex-A76 & 4x1.8 GHz Cortex-A55), Graphics Processing Unit (GPU) of Mali-G76 MP10, memory of 128GB 8GB RAM and battery capacity of Li-Po 3750 mAh.

As for the system software, Jupyter Lab that is used for obtaining the pre-trained model, Python software for training the deep learning model, TensorFlow & TensorFlow-lite for converting the model into a lighter version to deployed the model into mobile app, Vertex AI in Google Cloud Platform to optimise the model and Android Studio that is used to design the UI for the mobile app.

3.5 Summary

The methodology chapter of this research on a mobile application for bird species recognition using image processing techniques and deep learning provide an overview of the methods used in this research. This chapter begin by describing the conceptual framework of the mobile app, including the overall design. Next, this chapter detailed the methods used in the research, including any image processing techniques used to pre-process the bird dataset. This section also describes the deep learning model design process and the steps taken to convert the model to TensorFlow-Lite for use in a mobile app. Then this chapter also discuss the design of the UI and the steps taken to integrate and implement the model into the mobile application. This chapter also include a discussion of how the mobile app was evaluated in terms of accuracy and hardware and software specifications. Finally, this chapter should summarize the key findings of the methodology and how they relate to the research problem statement.

CHAPTER 4

RESULT AND DISCUSSION

4.1 Overview

In this chapter, an overview is provided of the outcomes and analysis derived from various parameters for five image classification models employed in classifying local bird species. The models under examination include ResNet-50, InceptionV3, VGG16, MobileNetV2, and EfficientNet-B4. The parameters assessed encompass a range of aspects such as training time, training loss, validation loss, training accuracy, validation accuracy, test loss, and test accuracy.

Through the comparison of these parameters, the most effective model is selected for further evaluation. Subsequently, the chosen model is deployed on Google Cloud Platform using Vertex AI, where it undergoes optimization to enhance its performance. The optimization process involves scrutinizing the precision-recall graph and precision-recall threshold to gauge the model's precision and recall rates.

Additionally, the chapter delves into assessing the performance of the mobile application associated with the model. This evaluation entails subjecting the mobile app to testing with a test set, thereby facilitating the calculation of its average accuracy in identifying bird species.

In summary, this chapter presents a comprehensive overview of the results and analysis obtained from the image classification models, including the selection of the best model, its deployment on Google Cloud Platform for optimization, and the evaluation of the mobile app's accuracy.

4.2 Comparison of Five (5) Transfer Learning Image Classification Model

Transfer learning is a widely used technique in deep learning for image classification tasks. In this project, five popular transfer learning models EfficientNet-B4, MobileNetV2, InceptionV3, ResNet-50, and VGG16 were compared to help choose which model best perform for the bird's images dataset. These models are evaluated on seven parameters which are their training time, training loss, validation loss, training accuracy, validation accuracy, test loss, and test accuracy.

Training time refers to the duration for a neural network model to learn from the training data and update its parameters. Additionally, training loss measures the disagreement between the model's predicted outputs and actual labels in the training data. Similarly, validation loss assesses the disagreement on a separate validation dataset, guiding model improvements. Moreover, training accuracy measures the percentage of correctly predicted outputs on the training data, while validation accuracy does the same on the validation data. Furthermore, test loss evaluates the disagreement on unseen test data, and test accuracy measures the percentage of correct predictions on the test data.

In order to assess the reliability and accuracy of the model, the local bird image datasets will be utilized to train various image classification models. The purpose of employing these datasets is to ensure that the image classification process can accurately identify the species of each bird, with the intention of implementing it in a mobile app platform later on. This project emphasizes the importance of evaluating the accuracy of each trained model and how effectively it performs on the provided dataset.

4.2.1 ResNet-50

ResNet-50 is a member of the Res-Net family, which is a convolutional neural network architecture widely utilized in computer vision tasks like image classification and object detection.ResNet-50 is characterized by its 50-layer deep structure, enabling it to effectively capture intricate patterns and features in images.

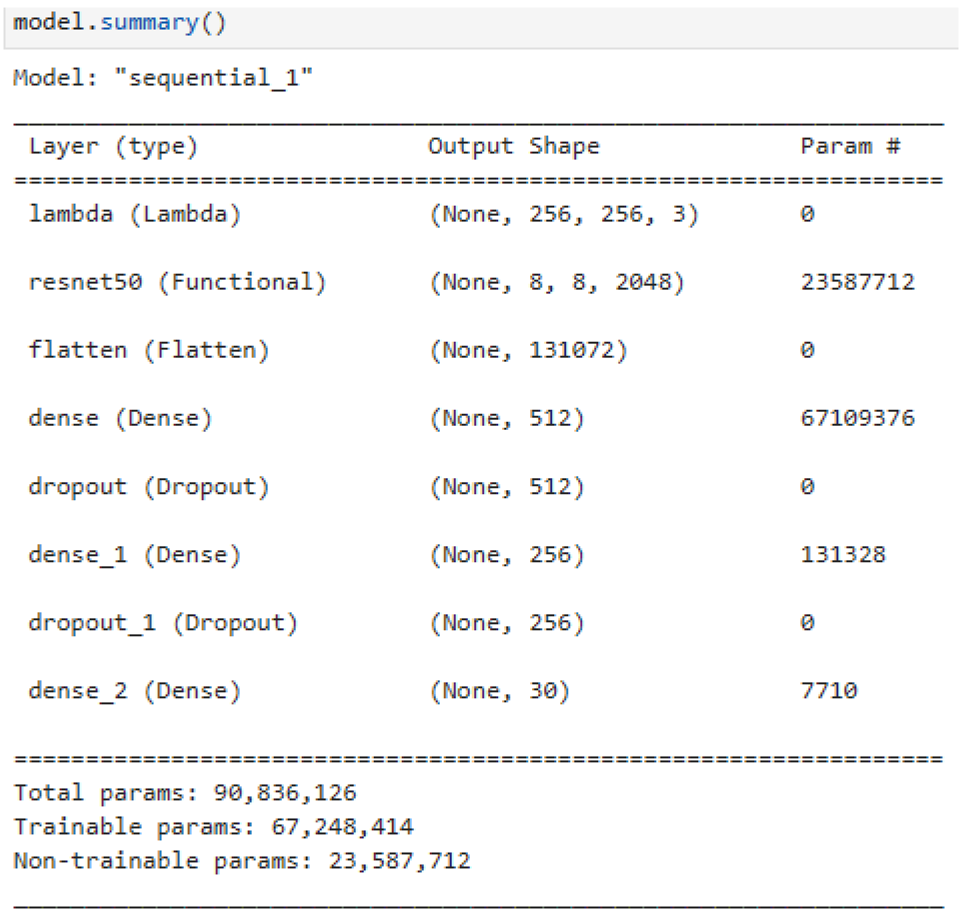
An important innovation of ResNet-50 is the incorporation of residual connections, allowing the network to learn residual mappings and directly focus on the difference between input and output at each layer. This resolves the issue of vanishing gradients and enables successful training of deep networks.

ResNet-50 follows a modular design, comprising blocks that consist of convolutional layers, batch normalization, and activation functions. These blocks facilitate feature extraction at different levels of abstraction, enabling the model to learn representations of varying complexities.

The ResNet-50 model has a total of 90,836,126 parameters shown in Figure 4.1. Parameters are the variables learned by the model during training, and they are crucial for its ability to make predictions on new data. A higher number of parameters often indicates a more complex and potentially powerful model.

Out of the total parameters, 67,248,414 are trainable. Trainable parameters are the ones that get updated during the training process to optimize the model's performance on the training data. These trainable parameters are what the model adjusts and fine-tunes to improve its ability to recognize patterns and features in the data.

The remaining 23,587,712 parameters are non-trainable. Non-trainable parameters are those that are fixed and do not get updated during training. These parameters are typically pre-defined or come from pre-trained models that serve as a starting point for further training or transfer learning.



```
model.summary()
```

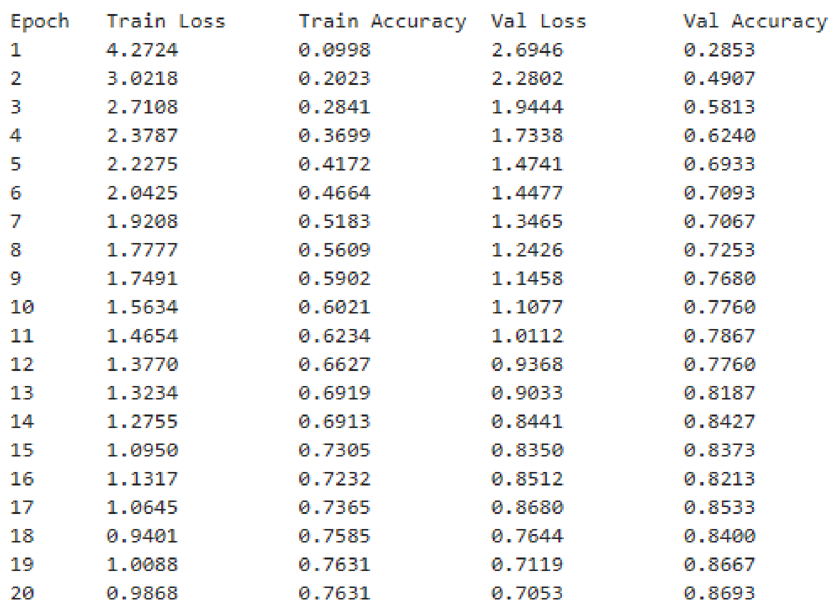
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 256, 256, 3)       0

 resnet50 (Functional)       (None, 8, 8, 2048)        23587712

 flatten (Flatten)           (None, 131072)            0

 dense (Dense)               (None, 512)               67109376

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 30)                7710

=================================================================
Total params: 90,836,126
Trainable params: 67,248,414
Non-trainable params: 23,587,712
_________________________________________________________________
```

Figure 4.1: ResNet-50 Model Summary

From figure 4.2, 4.3, 4.4 and 4.5 during the first epoch, the model had a relatively high training loss of 4.2724 and a low training accuracy of 0.0998 , indicating poor performance on the training data. The validation loss was 2.6946, and the validation accuracy was 0.2853, showing that the model was also struggling with unseen data during validation.

As the training continued, the model's performance improved significantly. The training loss decreased, and the training accuracy increased with each epoch. By the 20th epoch, the training loss dropped to 0.9868, and the training accuracy reached 0.7631, demonstrating the model's ability to learn and improve its predictions on the training data.

The validation performance also showed steady improvement throughout the training process. The validation loss decreased from 2.6946 in the first epoch to 0.7053 in the 20th epoch, indicating that the model became better at generalizing to new data. The validation accuracy increased from 0.2853 to 0.8693, showing the model's ability to make accurate predictions on previously unseen data.



```
Epoch    Train Loss       Train Accuracy  Val Loss         Val Accuracy
1        4.2724           0.0998          2.6946           0.2853
2        3.0218           0.2023          2.2802           0.4907
3        2.7108           0.2841          1.9444           0.5813
4        2.3787           0.3699          1.7338           0.6240
5        2.2275           0.4172          1.4741           0.6933
6        2.0425           0.4664          1.4477           0.7093
7        1.9208           0.5183          1.3465           0.7067
8        1.7777           0.5609          1.2426           0.7253
9        1.7491           0.5902          1.1458           0.7680
10       1.5634           0.6021          1.1077           0.7760
11       1.4654           0.6234          1.0112           0.7867
12       1.3770           0.6627          0.9368           0.7760
13       1.3234           0.6919          0.9033           0.8187
14       1.2755           0.6913          0.8441           0.8427
15       1.0950           0.7305          0.8350           0.8373
16       1.1317           0.7232          0.8512           0.8213
17       1.0645           0.7365          0.8680           0.8533
18       0.9401           0.7585          0.7644           0.8400
19       1.0088           0.7631          0.7119           0.8667
20       0.9868           0.7631          0.7053           0.8693
```

Figure 4.2: Data of ResNet-50 train loss, validation loss, train accuracy and validation accuracy for 20 epochs.

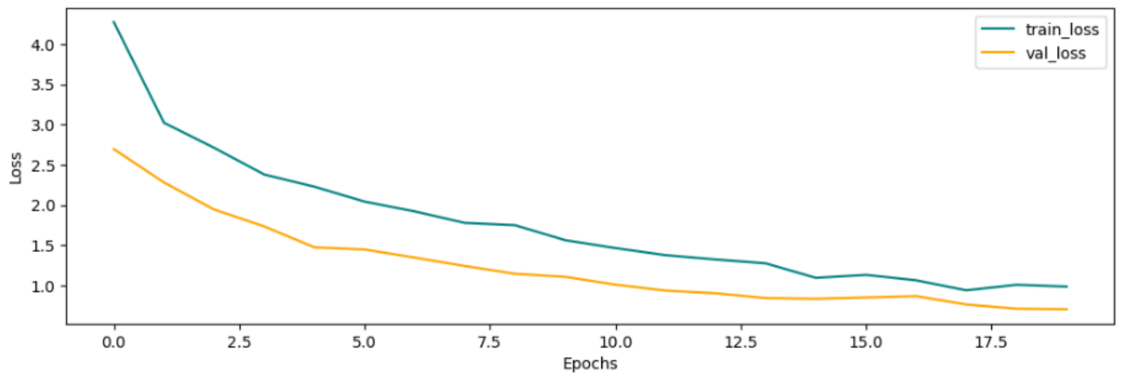


Figure 4.3 : Graph of training and validation loss for ResNet-50 model.

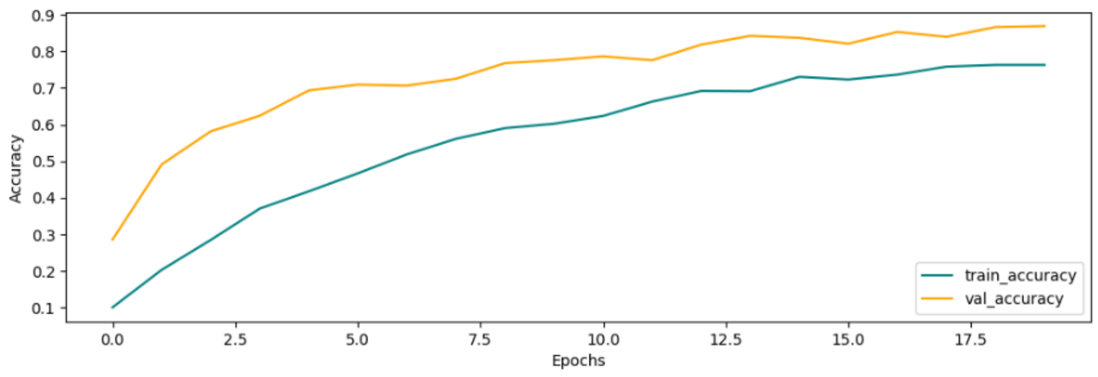
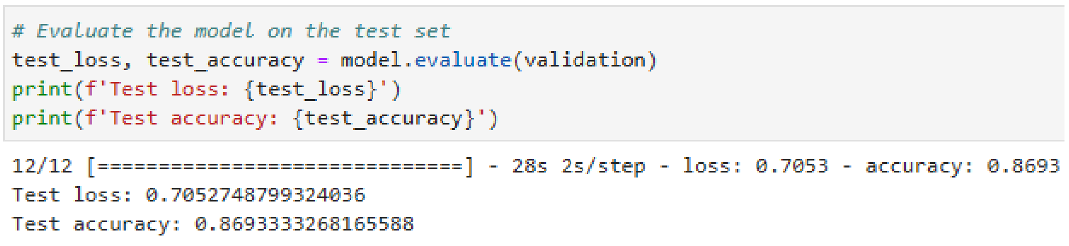


Figure 4.4: Graph of training and validation accuracy for ResNet-50 model.



```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(validation)
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
12/12 [==============================] - 28s 2s/step - loss: 0.7053 - accuracy: 0.8693
Test loss: 0.7052748799324036
Test accuracy: 0.8693333268165588
```

Figure 4.5: Testing loss and accuracy of the ResNet-50 model.

The Table 4.1, provides a detailed evaluation of the performance of ResNet-50 throughout the training and validation processes, as well as the final test results. The average training time for one epoch is 215 seconds, indicating how long it takes for the model to process the entire training dataset once during training.

During the training phase, the model achieves an average training loss of 0.9401, reflecting the discrepancy between its predicted outputs and the actual target labels. The average training accuracy reaches 0.7631, indicating the model's proficiency in classifying examples from the training set.

In the validation phase, the model's performance is assessed on a separate dataset not used during training. The average validation loss is 0.7053, indicating good generalization performance and learning meaningful representations from the data. The average validation accuracy is 0.8693, demonstrating the model's ability to generalize well to new, unseen data.Finally, the model's performance on completely unseen data is measured using the test dataset. The test loss is 0.7052, confirming that the model performs well on new examples, while the test accuracy reaches 0.8693, further indicating its good generalization capability.

Table 4.1 : Results for each parameter train for 20 epochs for ResNet-50 model.

Average Training Time for 1 Epochs (s)	**Average Training Loss**	**Average Training Accuracy**	**Average Validation Loss**	**Average Validation Accuracy**	**Test Loss**	**Test Accuracy**
215	0.9401	0.7631	0.7053	0.8693	0.7052	0.8693

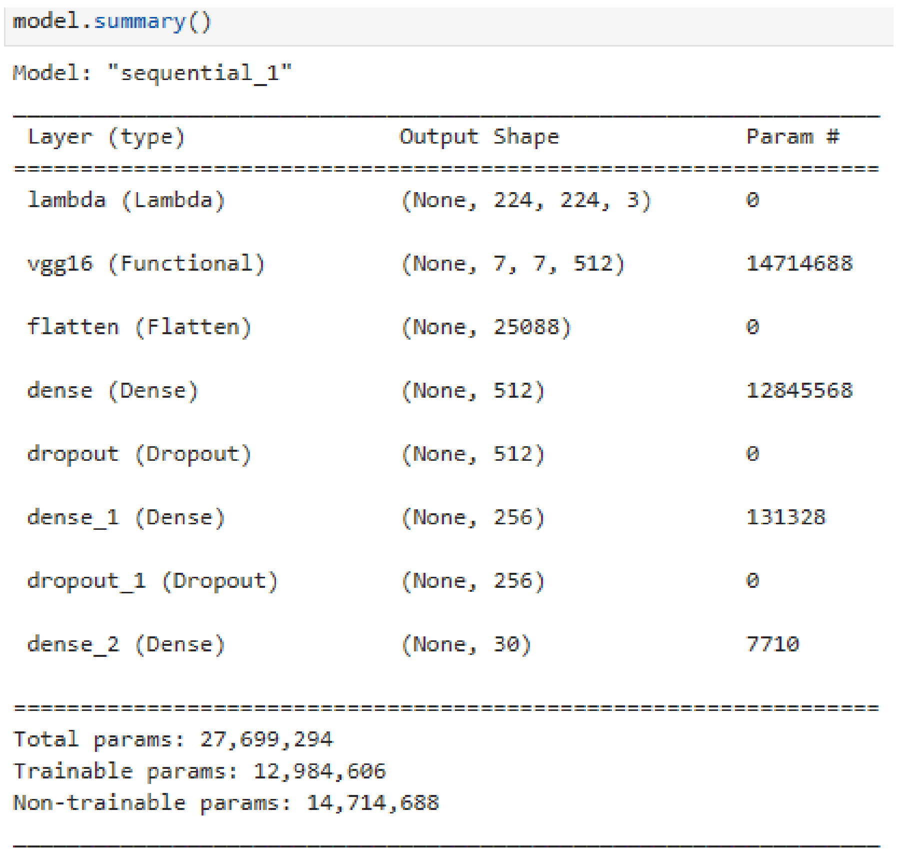
4.2.2 VGG16

VGG16 is a well-known convolutional neural network architecture recognized for its effectiveness and simplicity in image classification tasks. Developed by the Visual Geometry Group at the University of Oxford, VGG16 is characterized by its 16 layers, consisting of 13 convolutional layers and 3 fully connected layers. Its architecture follows a uniform structure with stacked 3x3 convolutional filters.

The deep convolutional layers in VGG16 allow the model to learn intricate patterns and hierarchical representations in images. Through the use of multiple stacked convolutional layers, VGG16 can capture both low-level features like edges and textures, as well as high-level features such as shapes and objects.

Unlike more recent architectures, VGG16 does not incorporate specialized layers like residual connections or inception modules. It prioritizes simplicity and employs a straightforward design approach.

The model summary in Figure 4.6 shows that VGG16 consists of a total of 27,699,294 parameters. Among these, 12,984,606 parameters are designated as trainable, and they are adapted during the training phase to gather insights from the data. Conversely, the remaining 14,714,688 parameters are classified as non-trainable and remain fixed throughout the training process. The abundance of trainable parameters enables the model to grasp intricate patterns and features from the data, enhancing its efficacy in image classification tasks. On the other hand, the non-trainable parameters offer valuable pre-defined features or representations that support the model in its classification endeavors.

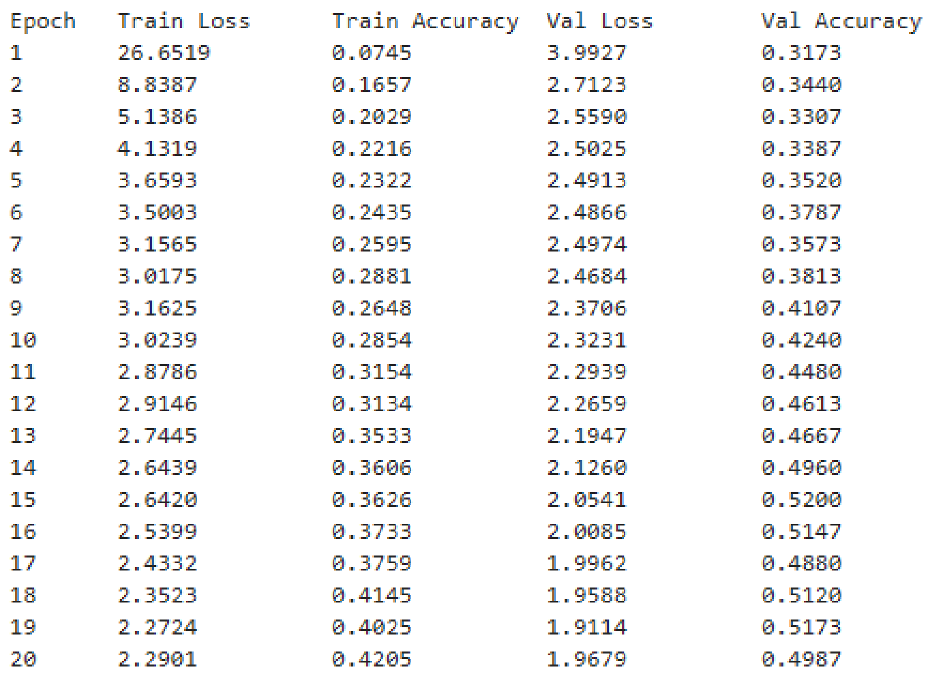


```
model.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 224, 224, 3)       0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 30)                7710

=================================================================
Total params: 27,699,294
Trainable params: 12,984,606
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 4.6: Model Summary for VGG16 model.

The data summary and graphs in Figure 4.7, 4.8, 4.9 and 4.10 presents the performance of VGG16 during training and validation across multiple epochs. It tracks key metrics that reflect the model's learning and generalization capabilities. As the training progresses over the epochs, the "Train Loss" decreases, indicating an enhancement in the model's ability to make accurate predictions on the training data. For instance, in the first epoch, the train loss is 26.6519, which significantly reduces to 2.2901 in the twentieth epoch. Concurrently, the "Train Accuracy" increases as the model gets better at classifying examples from the training set. It rises from 0.0745 in the first epoch to 0.4205 in the twentieth epoch.

In the validation phase, the model's performance on unseen data is measured through the "Val Loss" and "Val Accuracy" metrics. The validation loss, which starts at 3.9927 in the first epoch, decreases to 1.9679 in the twentieth epoch, showcasing the model's improvement in generalization. Similarly, the validation accuracy increases from 0.3173 in the first epoch to 0.4987 in the twentieth epoch, highlighting the model's enhanced ability to generalize to new, unseen examples.



```
Epoch   Train Loss      Train Accuracy  Val Loss        Val Accuracy
1       26.6519         0.0745          3.9927          0.3173
2       8.8387          0.1657          2.7123          0.3440
3       5.1386          0.2029          2.5590          0.3307
4       4.1319          0.2216          2.5025          0.3387
5       3.6593          0.2322          2.4913          0.3520
6       3.5003          0.2435          2.4866          0.3787
7       3.1565          0.2595          2.4974          0.3573
8       3.0175          0.2881          2.4684          0.3813
9       3.1625          0.2648          2.3706          0.4107
10      3.0239          0.2854          2.3231          0.4240
11      2.8786          0.3154          2.2939          0.4480
12      2.9146          0.3134          2.2659          0.4613
13      2.7445          0.3533          2.1947          0.4667
14      2.6439          0.3606          2.1260          0.4960
15      2.6420          0.3626          2.0541          0.5200
16      2.5399          0.3733          2.0085          0.5147
17      2.4332          0.3759          1.9962          0.4880
18      2.3523          0.4145          1.9588          0.5120
19      2.2724          0.4025          1.9114          0.5173
20      2.2901          0.4205          1.9679          0.4987
```

Figure 4.7: Data of VGG16 train loss, validation loss, train accuracy and validation accuracy for 10 epochs.

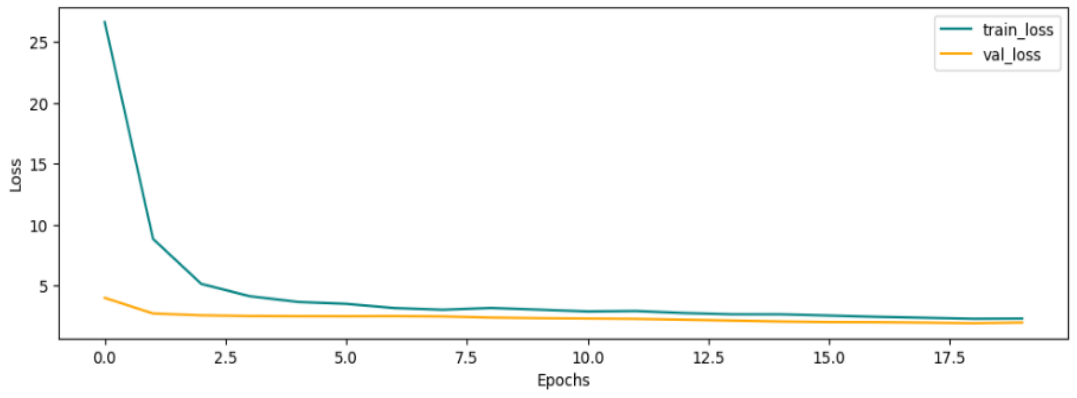


Figure 4.8: Graph of training and validation loss for VGG16 model.

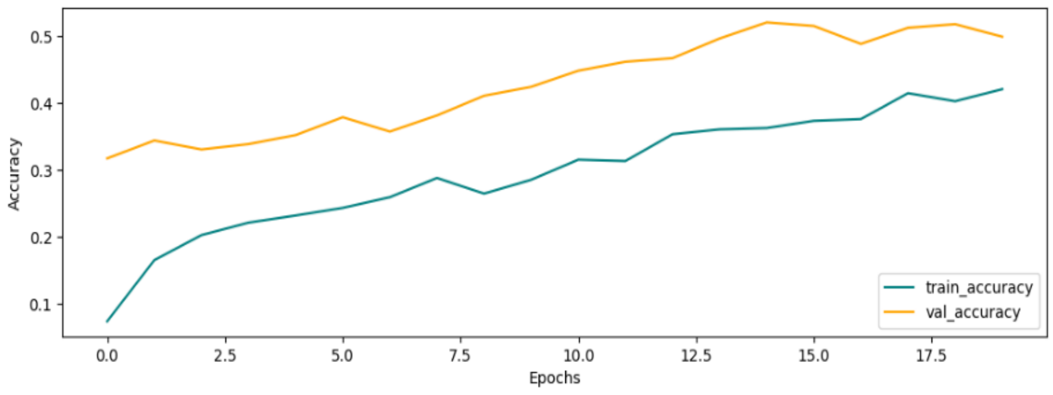
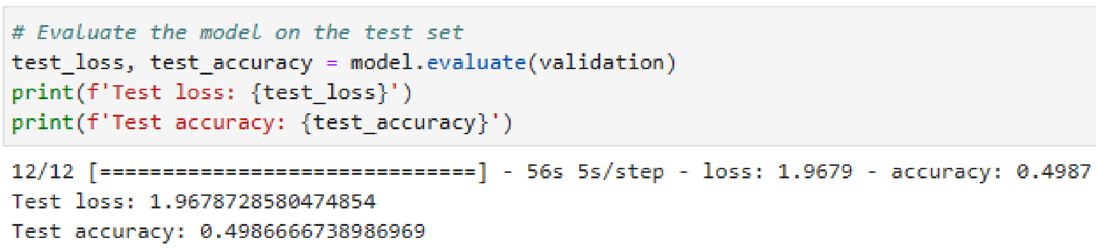


Figure 4.9: Graph of training and validation accuracy for VGG16 model.



```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(validation)
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
12/12 [==============================] - 56s 5s/step - loss: 1.9679 - accuracy: 0.4987
Test loss: 1.9678728580474854
Test accuracy: 0.4986666738986969
```

Figure 4.10: Testing loss and accuracy of the VGG16 model.

The table 4.2 summary provides a comprehensive evaluation of VGG16's performance across different phases of the model's training and testing.

During training, the model demonstrates an average training loss of 2.2724, indicating that its predictions are improving and getting closer to the true labels as it undergoes more epochs. The average training accuracy increases from 0.0745 in the first epoch to 0.4205 in the twentieth epoch, showcasing the model's ability to classify examples from the training set more accurately over time.

In the validation phase, the model's performance is assessed on a separate dataset, and the average validation loss decreases from 3.9927 in the first epoch to 1.9679 in the twentieth epoch, demonstrating its ability to generalize well to unseen data. Similarly, the average validation accuracy rises from 0.3173 to 0.5173 during the same period, indicating better generalization performance.

The final evaluation on the test dataset reveals a test loss of 1.9678 and a test accuracy of 0.4986, demonstrating that the model performs relatively well on completely new, unseen examples.

Table 4.2: Results for each parameter train for 20 epochs for VGG16 model.

Average Training Time for 1 Epochs (s)	**Average Training Loss**	**Average Training Accuracy**	**Average Validation Loss**	**Average Validation Accuracy**	**Test Loss**	**Test Accuracy**
398	2.2724	0.4205	1.9679	0.5173	1.9678	0.4986

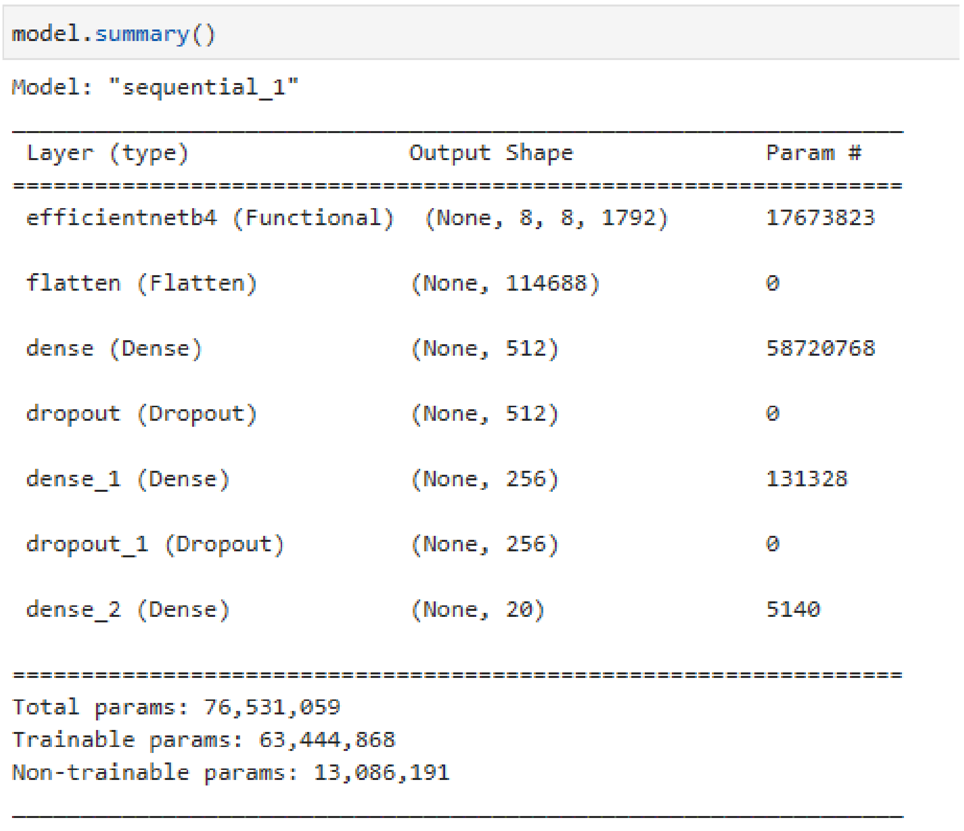
4.2.3 EfficientNet-B4

EfficientNet-B4 is a variation within the Efficient-Net series of convolutional neural network models. Its purpose is to strike a balance between model size and performance across a broad range of computer vision tasks, including image classification and object detection.

EfficientNet-B4 extends the base architecture of the Efficient-Net model by incorporating additional layers and parameters, making it larger and potentially more robust. The "B4" designation denotes the specific scaling factor applied to the base architecture, indicating its relative size in comparison to the original Efficient-Net model.

EfficientNet-B4 takes advantage of advanced techniques like compound scaling, which simultaneously scales the model's depth, width, and resolution. This approach enhances both efficiency and accuracy. By employing compound scaling, EfficientNet-B4 achieves superior performance compared to earlier models while maintaining computational and parameter efficiency.

The EfficientNet-B4 model in Figure 4.11 has a total of 76,533,629 parameters. During training, 63,447,438 of these parameters are updated and optimized, while the remaining 13,086,191 parameters are fixed and not modified during the training process. Having a large number of trainable parameters allows the model to learn complex patterns and features from the data, which contributes to its high performance in image classification tasks.



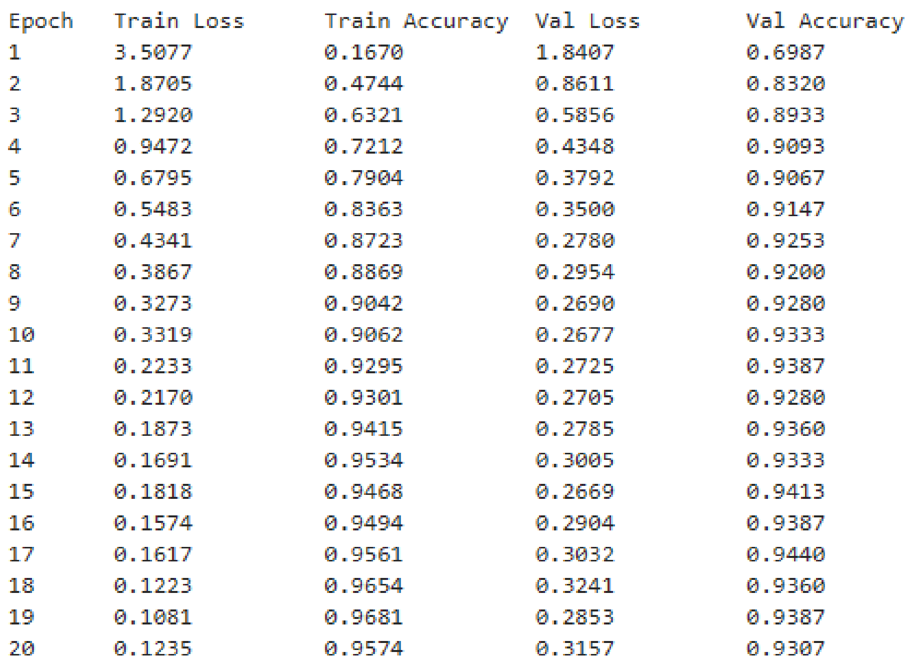
```
model.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb4 (Functional)  (None, 8, 8, 1792)       17673823

 flatten (Flatten)           (None, 114688)            0

 dense (Dense)               (None, 512)               58720768

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 20)                5140

=================================================================
Total params: 76,531,059
Trainable params: 63,444,868
Non-trainable params: 13,086,191
_________________________________________________________________
```

Figure 4.11: EfficientNet-B4 Model Summary

The data summary in Figure 4.12, 4.13, 4.14 and 4.15 are the performance of EfficientNet-B4 during training and validation across multiple epochs. During the training process, the model experiences a significant decrease in the "Train Loss" values, starting from 3.5077 in the first epoch and dropping to 0.1235 in the twentieth epoch. This decline indicates the model's improvement in making accurate predictions on the training data. Concurrently, the "Train Accuracy" increases steadily from 0.1670 to 0.9574, demonstrating the model's growing proficiency in classifying examples from the training dataset.

In the validation phase, the model's performance is evaluated on a separate dataset not used for training. The "Val Loss" diminishes progressively from 1.8407 to 0.3157, suggesting that the model is effectively learning meaningful patterns and features from the data, enabling it to make accurate predictions on unseen examples. Simultaneously, the "Val Accuracy" rises from 0.6987 to 0.9307, affirming the model's ability to generalize its knowledge to new data and perform well on unseen examples.

Overall, the data summary reveals that EfficientNet-B4 exhibits a decreasing trend in both training and validation loss, along with an increasing trend in both training and validation accuracy. These trends indicate that the model is effectively learning from the data and generalizing well to new, unseen examples, making it a suitable and efficient choice for image classification tasks.



```
Epoch    Train Loss      Train Accuracy  Val Loss        Val Accuracy
1        3.5077          0.1670          1.8407          0.6987
2        1.8705          0.4744          0.8611          0.8320
3        1.2920          0.6321          0.5856          0.8933
4        0.9472          0.7212          0.4348          0.9093
5        0.6795          0.7904          0.3792          0.9067
6        0.5483          0.8363          0.3500          0.9147
7        0.4341          0.8723          0.2780          0.9253
8        0.3867          0.8869          0.2954          0.9200
9        0.3273          0.9042          0.2690          0.9280
10       0.3319          0.9062          0.2677          0.9333
11       0.2233          0.9295          0.2725          0.9387
12       0.2170          0.9301          0.2705          0.9280
13       0.1873          0.9415          0.2785          0.9360
14       0.1691          0.9534          0.3005          0.9333
15       0.1818          0.9468          0.2669          0.9413
16       0.1574          0.9494          0.2904          0.9387
17       0.1617          0.9561          0.3032          0.9440
18       0.1223          0.9654          0.3241          0.9360
19       0.1081          0.9681          0.2853          0.9387
20       0.1235          0.9574          0.3157          0.9307
```

Figure 4.12: Data of EfficientNet-B4 train loss, validation loss, train accuracy and validation accuracy for 20 epochs.

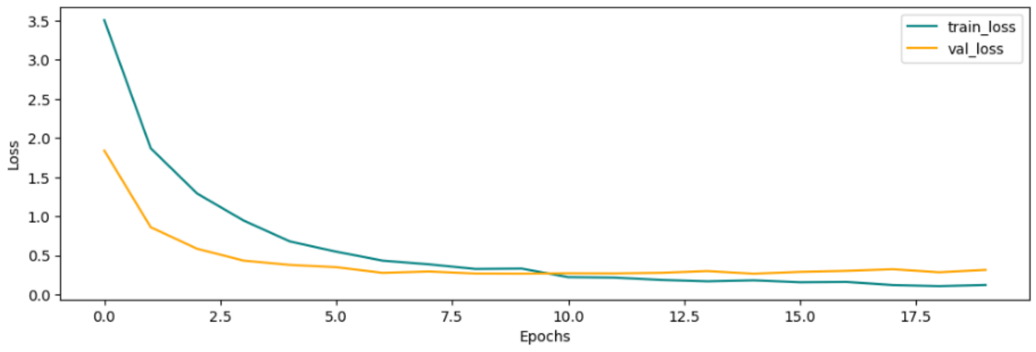


Figure 4.13: Graph of training and validation loss for EfficientNet-B4 model.

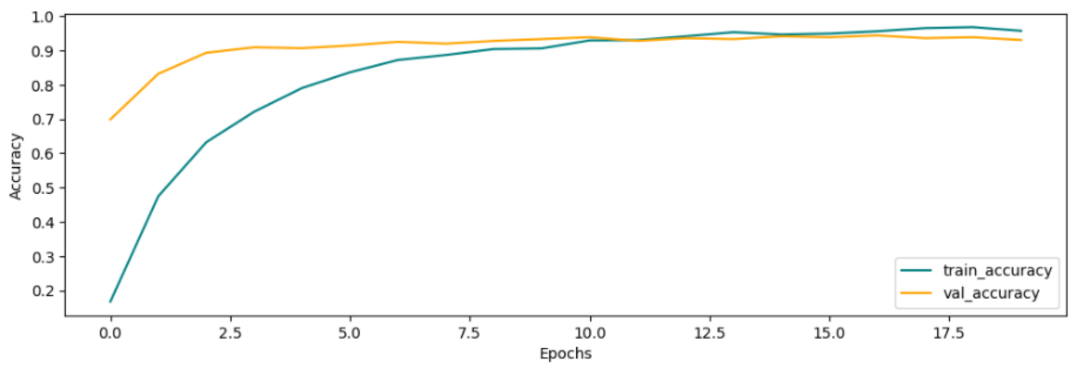
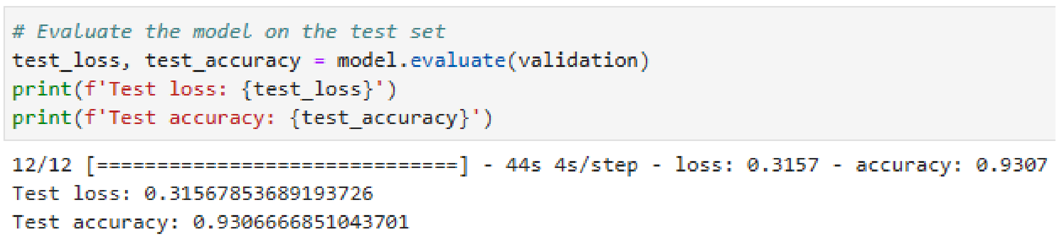


Figure 4.14: Graph of training and validation accuracy for EfficientNet-B4 model.



```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(validation)
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
12/12 [==============================] - 44s 4s/step - loss: 0.3157 - accuracy: 0.9307
Test loss: 0.31567853689193726
Test accuracy: 0.9306666851043701
```

Figure 4.15: Testing loss and accuracy of the EfficientNet-B4 model.

The table 4.3 data represents the performance metrics of the EfficientNet-B4 model. During training, each epoch took an average of 285 seconds (approximately 4 minutes and 45 seconds) to complete. The model achieved an average training loss of 0.1081, indicating how well it performed on the training data, with lower values being better. The average training accuracy reached an impressive 96.81%, reflecting the model's ability to correctly predict and classify images in the training set.

For validation, the model achieved an average validation loss of 0.2853, which suggests that it performed well on unseen data during validation. The average validation accuracy was 93.87%, demonstrating the model's ability to generalize and make accurate predictions on new, unseen data.

After training and validation, the model was tested on a separate test dataset. The test loss was measured at 0.3156, which is consistent with the validation loss and indicates good generalization. The test accuracy was recorded at 93.06%, which shows the model's strong ability to accurately classify images in the test set.

Table 4.3: Results for each parameter train for 20 epochs for EfficientNet-B4 model.

Average Training Time for 1 Epochs (s)	**Average Training Loss**	**Average Training Accuracy**	**Average Validation Loss**	**Average Validation Accuracy**	**Test Loss**	**Test Accuracy**
285	0.1081	0.9681	0.2853	0.9387	0.3156	0.9306

4.2.4 MobileNetV2

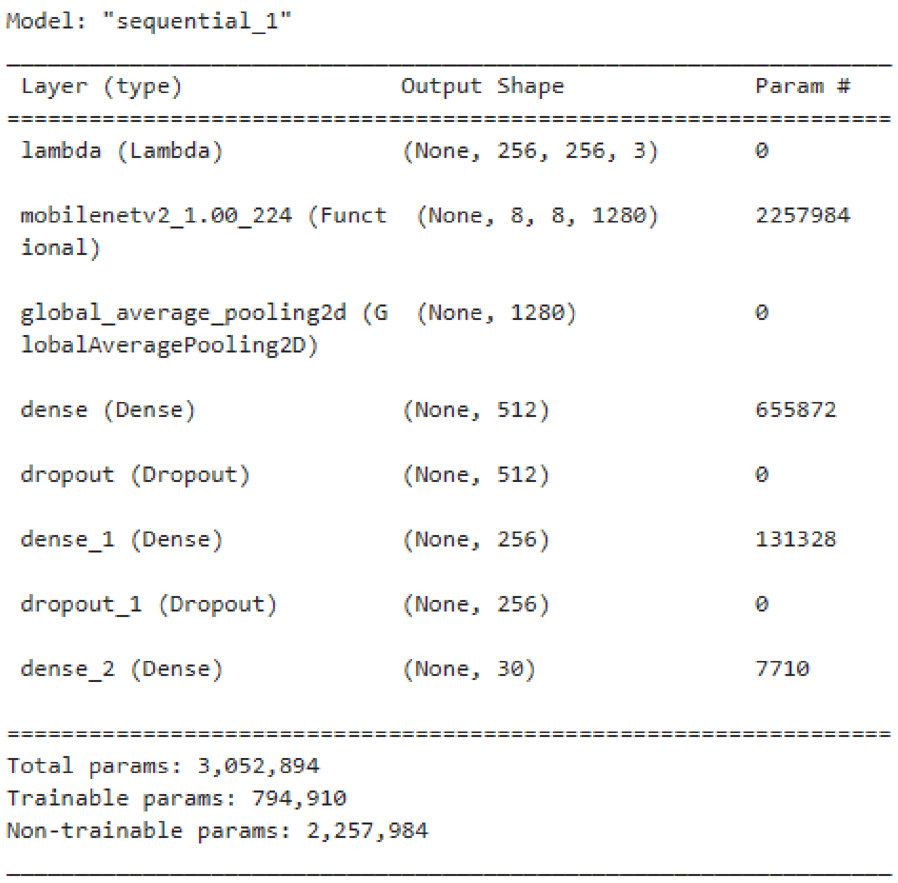
MobileNetV2 is a convolutional neural network architecture developed by Google to process and classify images efficiently on devices with limited resources, such as mobile phones and embedded systems. It is an enhanced version of the original MobileNet.

The primary goal of MobileNetV2 is to strike a balance between model size and accuracy. It incorporates various techniques to optimize its performance, including depth wise separable convolutions, which separate spatial and channel-wise convolutions to reduce computational complexity.

MobileNetV2 also features inverted residual blocks with linear bottlenecks, which enable efficient information flow by using bottleneck layers with fewer channels. This design captures complex patterns while minimizing the number of parameters. Additionally, MobileNetV2 utilizes linear scaling of depth and width, allowing customization of the model's size and complexity based on available resources and performance requirements.

The MobileNetV2 model has a total of 3,052,894 parameters as shown in figure 4.16, which are the variables learned during training and are crucial for making predictions on new data. With more than three million parameters, the MobileNetV2 is of moderate size, making it efficient for resource-constrained devices.

Out of the total parameters, 794,910 are trainable. These trainable parameters are updated during training to optimize the model's performance on the training data, allowing the model to learn and improve its ability to recognize patterns and features. The remaining 2,257,984 parameters are non-trainable. These parameters are fixed and do not change during training. They are often predefined or come from pre-trained models, serving as a starting point for further training or transfer learning processes.



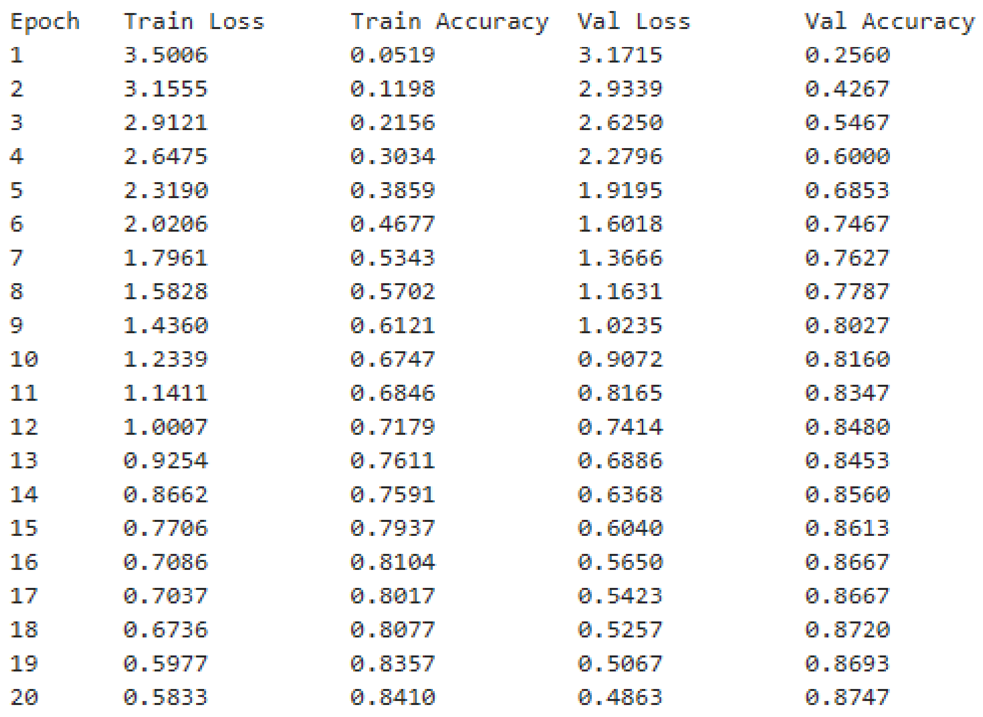
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 256, 256, 3)       0

 mobilenetv2_1.00_224 (Funct  (None, 8, 8, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 512)               655872

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 30)                7710

=================================================================
Total params: 3,052,894
Trainable params: 794,910
Non-trainable params: 2,257,984
_________________________________________________________________
```

Figure 4.16: MobileNetV2 Model Summary

From figure 4.17, 4.18, 4.19, and 4.20, in the beginning, the model's performance was not good, with high training loss (3.5006) and low training accuracy (0.0519). The validation results were also unsatisfactory, with a validation loss of 3.1715 and validation accuracy of 0.2560, indicating difficulty in handling new, unseen data during validation.

However, as the training continued, the model significantly improved its performance. The training loss decreased, reaching 0.5833, and the training accuracy increased to 0.8410 by the 20th epoch, showing the model's ability to learn and make more accurate predictions on the training data.

The validation performance also steadily improved during training. The validation loss decreased from 3.1715 in the first epoch to 0.4863 in the 20th epoch, indicating the model's better generalization to new data. The validation accuracy increased from 0.2560 to 0.8747, demonstrating the model's improved ability to make accurate predictions on previously unseen data.



Epoch	Train Loss	Train Accuracy	Val Loss	Val Accuracy
1	3.5006	0.0519	3.1715	0.2560
2	3.1555	0.1198	2.9339	0.4267
3	2.9121	0.2156	2.6250	0.5467
4	2.6475	0.3034	2.2796	0.6000
5	2.3190	0.3859	1.9195	0.6853
6	2.0206	0.4677	1.6018	0.7467
7	1.7961	0.5343	1.3666	0.7627
8	1.5828	0.5702	1.1631	0.7787
9	1.4360	0.6121	1.0235	0.8027
10	1.2339	0.6747	0.9072	0.8160
11	1.1411	0.6846	0.8165	0.8347
12	1.0007	0.7179	0.7414	0.8480
13	0.9254	0.7611	0.6886	0.8453
14	0.8662	0.7591	0.6368	0.8560
15	0.7706	0.7937	0.6040	0.8613
16	0.7086	0.8104	0.5650	0.8667
17	0.7037	0.8017	0.5423	0.8667
18	0.6736	0.8077	0.5257	0.8720
19	0.5977	0.8357	0.5067	0.8693
20	0.5833	0.8410	0.4863	0.8747

Figure 4.17: Data of MobileNetV2's train loss, validation loss, train accuracy and validation accuracy for 20 epochs.

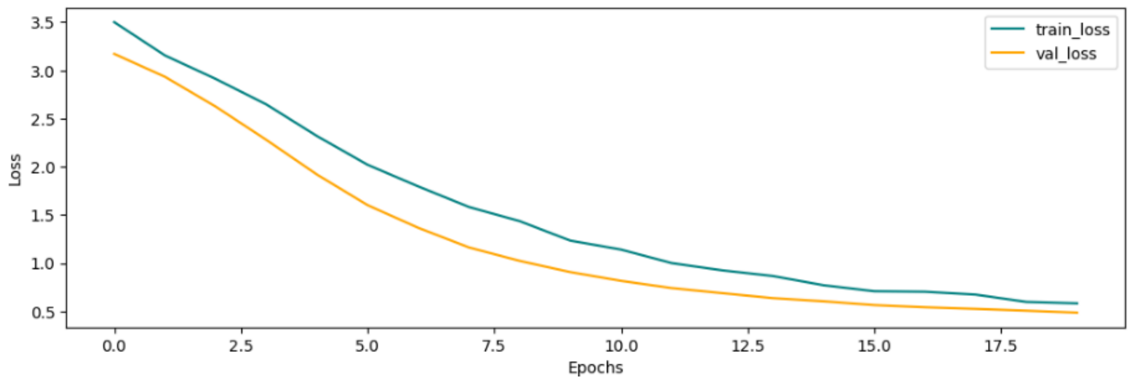


Figure 4.18: Graph of training and validation loss for MobileNetV2 model.

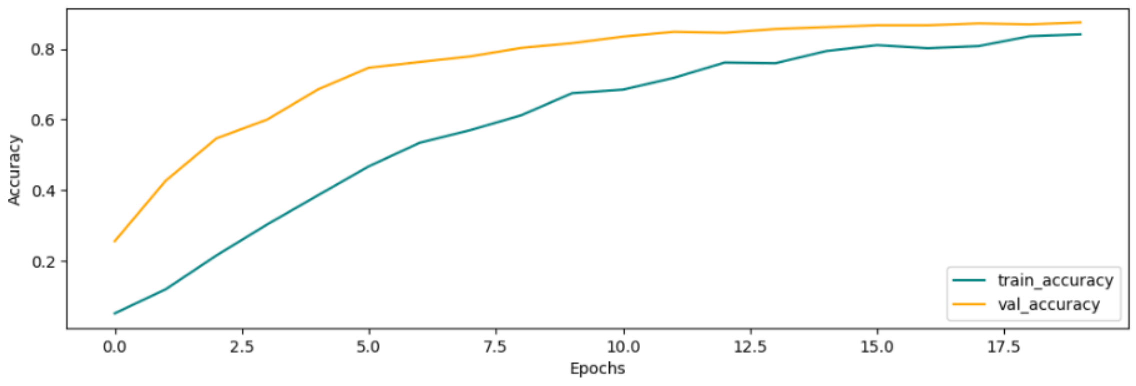
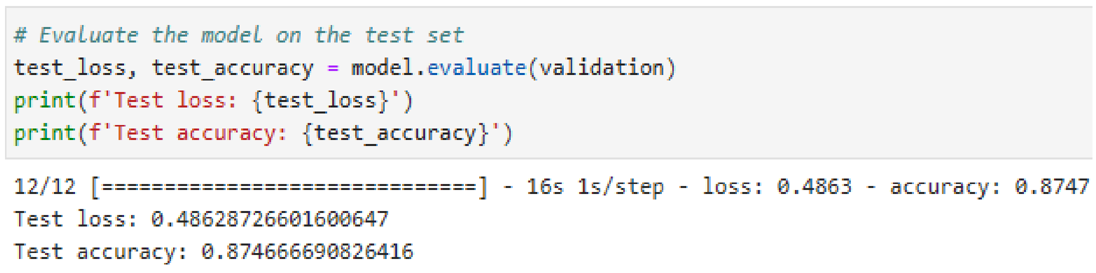


Figure 4.19: Graph of training and validation accuracy for MobileNetV2 model.



```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(validation)
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
12/12 [==============================] - 16s 1s/step - loss: 0.4863 - accuracy: 0.8747
Test loss: 0.48628726601600647
Test accuracy: 0.874666690826416
```

Figure 4.20: Testing loss and accuracy of the MobileNetV2 model.

The table 4.4 presents performance details for the MobileNetV2 model in terms of training, validation, and test metrics. On average, each training epoch took 174 seconds, representing the time it took for the model to complete one full round of training on the entire dataset. The model achieved an average training loss of 0.5833, indicating how well it performed on the training data. A lower training loss suggests improved accuracy in making predictions on the training set.

The average training accuracy reached 0.8410, showing the percentage of correct predictions the model made on the training data. A higher training accuracy indicates the model's increased ability to recognize patterns and features in the training set. During validation, the model showed an average validation loss of 0.4863, demonstrating its performance on unseen data during validation. A lower validation loss suggests that the model is generalizing well to new data.

The average validation accuracy was an impressive 0.8747, indicating the model's capability to make accurate predictions on new, unseen data during validation. Lastly, when tested on a separate test dataset, the model achieved an average test loss of 0.4862 and an average test accuracy of 0.8746. These results highlight the model's strong performance in accurately classifying data it has never encountered before.

Table 4.4: Results for each parameter train for 20 epochs for MobileNetV2 model.

Average Training Time for 1 Epochs (s)	**Average Training Loss**	**Average Training Accuracy**	**Average Validation Loss**	**Average Validation Accuracy**	**Test Loss**	**Test Accuracy**
174	0.5833	0.8410	0.4863	0.8747	0.4862	0.8746

4.2.5 InceptionV3

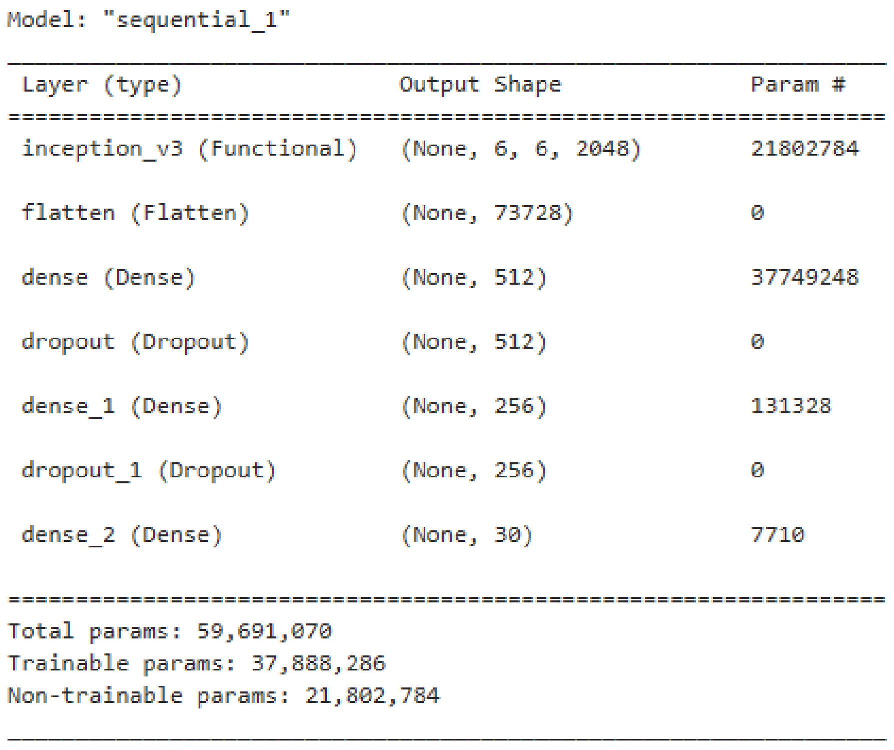
InceptionV3, developed by Google's DeepMind team, is a convolutional neural network architecture created for computer vision tasks like image classification and object detection.

The main innovation in InceptionV3 is the utilization of "Inception modules," which are blocks that perform parallel convolutions at various scales and concatenate the results. This enables the network to capture information at different levels of abstraction, extracting both detailed and high-level features from images effectively. InceptionV3 also integrates techniques like factorized convolutions and pooling to reduce computational costs and memory requirements without sacrificing expressive capabilities.

Additionally, InceptionV3 employs "auxiliary classifiers" at intermediate layers, which provide extra supervision during training and address the issue of vanishing gradients, leading to more stable and efficient training. With its deep architecture and advanced design, InceptionV3 has demonstrated remarkable performance on image classification benchmarks, surpassing earlier models. It is widely adopted in research and practical applications due to its ability to capture comprehensive visual representations and accurately classify images.

The InceptionV3 model possesses a grand total of 59,691,070 parameters as shown in figure 4.21. Parameters are the learned variables during training that are crucial for the model's predictive capabilities on new data. With approximately 60 million parameters, the InceptionV3 model is relatively large and intricate.

Within the total parameters, 37,888,286 are trainable. These trainable parameters are the ones that undergo updates during the training process to optimize the model's performance on the training data. The model fine-tunes and adjusts these trainable parameters to enhance its capacity for recognizing patterns and features in the data. On the other hand, the remaining 21,802,784 parameters are non-trainable. These parameters remain fixed and do not experience updates during training. They are typically pre-defined or sourced from pre-trained models, serving as an initial foundation for further training or transfer learning endeavors.

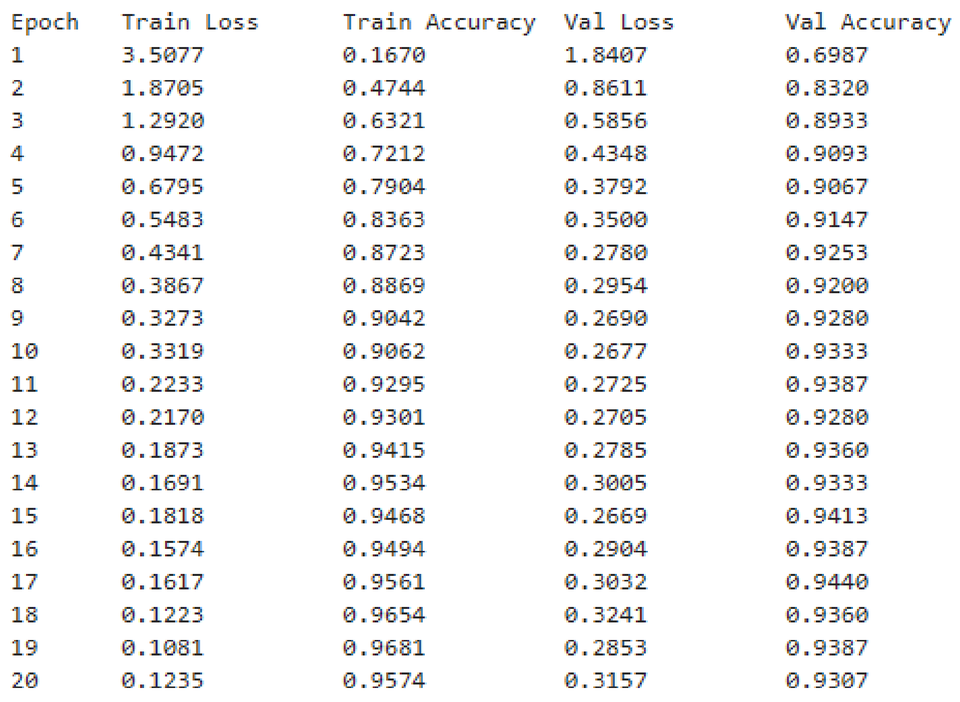


```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784

 flatten (Flatten)           (None, 73728)             0

 dense (Dense)               (None, 512)               37749248

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 30)                7710

=================================================================
Total params: 59,691,070
Trainable params: 37,888,286
Non-trainable params: 21,802,784
_________________________________________________________________
```

Figure 4.21: InceptionV3 Model Summary.

During the training as shown in 4.22, 4.23, 4.24 and 4.25 the model's performance was measured using training loss and training accuracy. The training loss decreased gradually from 33.43 in the initial epoch to 3.40 in the 12th epoch, indicating that the model improved in making predictions on the training data. However, the training accuracy remained relatively low, varying around 0.03 to 0.05, suggesting the model struggled to correctly classify the training data.

For validation, the model's performance was measured using validation loss and validation accuracy. The validation loss fluctuated between 3.40 and 3.41 throughout the training epochs. The validation accuracy remained consistently low, at around 0.03 to 0.04, indicating that the model faced difficulties in generalizing to new, unseen data during validation.



Epoch	Train Loss	Train Accuracy	Val Loss	Val Accuracy
1	3.5077	0.1670	1.8407	0.6987
2	1.8705	0.4744	0.8611	0.8320
3	1.2920	0.6321	0.5856	0.8933
4	0.9472	0.7212	0.4348	0.9093
5	0.6795	0.7904	0.3792	0.9067
6	0.5483	0.8363	0.3500	0.9147
7	0.4341	0.8723	0.2780	0.9253
8	0.3867	0.8869	0.2954	0.9200
9	0.3273	0.9042	0.2690	0.9280
10	0.3319	0.9062	0.2677	0.9333
11	0.2233	0.9295	0.2725	0.9387
12	0.2170	0.9301	0.2705	0.9280
13	0.1873	0.9415	0.2785	0.9360
14	0.1691	0.9534	0.3005	0.9333
15	0.1818	0.9468	0.2669	0.9413
16	0.1574	0.9494	0.2904	0.9387
17	0.1617	0.9561	0.3032	0.9440
18	0.1223	0.9654	0.3241	0.9360
19	0.1081	0.9681	0.2853	0.9387
20	0.1235	0.9574	0.3157	0.9307

Figure 4.22: Data of InceptionV3 train loss, validation loss, train accuracy and validation accuracy for 20 epochs.

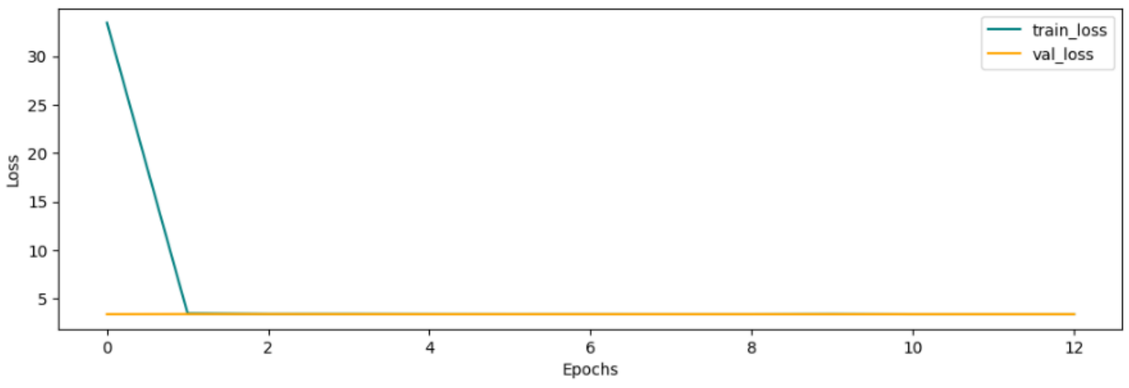


Figure 4.23: Graph of training and validation loss for InceptionV3 model.

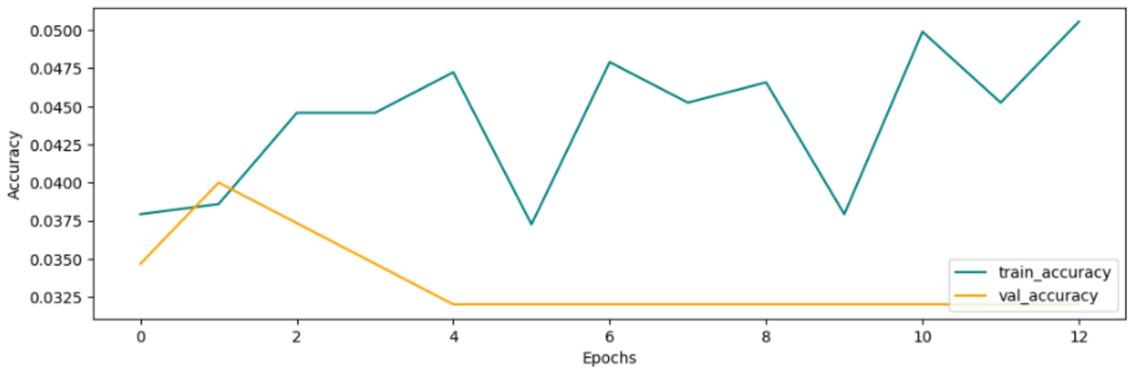
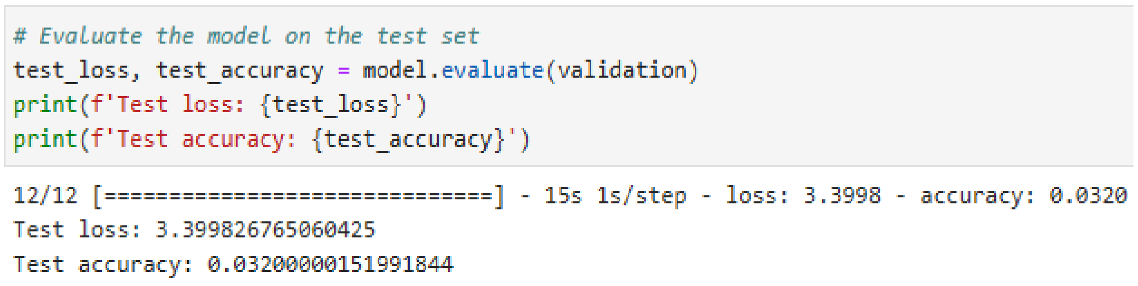


Figure 4.24: Graph of training and validation accuracy for InceptionV3 model.



```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(validation)
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
12/12 [==============================] - 15s 1s/step - loss: 3.3998 - accuracy: 0.0320
Test loss: 3.399826765060425
Test accuracy: 0.03200000151991844
```

Figure 4.25: Testing loss and accuracy of the InceptionV3 model.

Table 4.5 shows the InceptionV3 model took an average of 125 seconds for each training epoch, representing the time it needed to complete one round of training using the entire dataset. During training, the model achieved an average training loss of 3.3953, indicating its performance on the training data. A lower training loss implies improved accuracy in predicting the training set. The average training accuracy reached 0.0505, showing the percentage of correct predictions made by the model on the training data. Higher training accuracy indicates the model's improved ability to recognize patterns and features in the training set.

For validation, the model's average validation loss was 3.3998, representing its performance on new, unseen data during validation. A lower validation loss suggests that the model is generalizing well to new data. The average validation accuracy was 0.0320, indicating the model's capability to make accurate predictions on new, unseen data during validation. In separate testing on an independent dataset, the model obtained an average test loss of 3.3998 and an average test accuracy of 0.0320.

Table 4.5: Results for each parameter train for 20 epochs for InceptionV3 model.

Average Training Time for 1 Epochs (s)	**Average Training Loss**	**Average Training Accuracy**	**Average Validation Loss**	**Average Validation Accuracy**	**Test Loss**	**Test Accuracy**
125	3.3953	0.0505	3.3998	0.0320	3.3998	0.0320

4.2.6 Comparison Analysis Based on Parameters for 5 Image Classification Model

Comparing the models based on Table 4.6 below with the given parameters which are their training time, training loss, validation loss, training accuracy, validation accuracy, test loss, and test accuracy, it becomes apparent that each model possesses its own strengths and weaknesses. In terms of training time, InceptionV3 stands out as the swiftest model, averaging a mere 125 seconds. MobileNetV2 closely follows with a training time of approximately 174 seconds. Conversely, EfficientNetB4, ResNet-50, and VGG16 necessitate more time for training, averaging at 285, 215, and 398 seconds, respectively.

In the realm of training loss, EfficientNetB4 exhibits the most favourable performance, achieving an average training loss of 0.1081. MobileNetV2 also demonstrates commendable results in this regard, with a training loss of 0.5833. On the other hand, ResNet-50, VGG16, and InceptionV3 present higher training losses, with values of 0.9401, 2.2724, and 3.3953, respectively.

EfficientNetB4 emerges as the frontrunner in training accuracy, boasting an average accuracy of 0.9681. MobileNetV2 closely trails behind with a training accuracy of 0.8410, while ResNet-50, VGG16, and InceptionV3 attain lower training accuracies of 0.7631, 0.4205, and 0.0505, respectively.

Shifting focus to the validation phase, EfficientNetB4 exhibits superior performance by attaining the lowest validation loss of 0.2853. MobileNetV2 also fares well in this aspect, with a validation loss of 0.4863. Conversely, ResNet-50, VGG16, and InceptionV3 present higher validation losses, recording values of 0.7053, 1.9679, and 3.3998, respectively.

When considering validation accuracy, EfficientNetB4 takes the lead with an average accuracy of 0.9387, closely pursued by MobileNetV2 at 0.8747. ResNet-50 achieves a validation accuracy of 0.8693, whereas VGG16 and InceptionV3 lag behind with values of 0.5173 and 0.0320, respectively.

In the testing phase, EfficientNetB4 continues to shine with the lowest test loss of 0.3156. MobileNetV2 closely follows with a test loss of 0.4862. Conversely, ResNet-50, VGG16, and InceptionV3 exhibit higher test losses, recording values of 0.7052, 1.9678, and 3.3998, respectively.

Lastly, in terms of test accuracy, EfficientNetB4 proves to be the top performer with an accuracy of 0.9306. MobileNetV2 closely trails behind with an accuracy of 0.8746, while ResNet-50 achieves an accuracy of 0.8693. VGG16 and InceptionV3 record lower test accuracies of 0.4986 and 0.0320, respectively.

Taking into account the overall performance across all parameters, EfficientNetB4 emerges as the optimal model for image classification. It consistently delivers excellent results in terms of training loss, training accuracy, validation loss, validation accuracy, test loss, and test accuracy.

Table 4.6: Comparison of parameters for each of the image classification model.

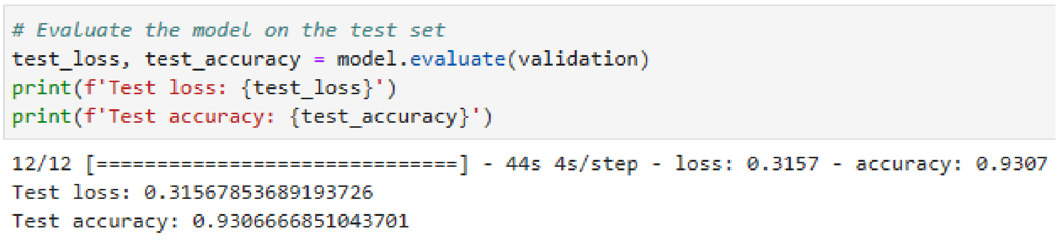
Image Classification Model	**Average Training Time (s)**	**Average Training Loss**	**Average Training Accuracy**	**Average Validation Loss**	**Average Validation Accuracy**	**Test Loss**	**Test Accuracy**
InceptionV3	125	3.3953	0.0505	3.3998	0.0320	3.3998	0.0320
ResNet-50	215	0.9401	0.7631	0.7053	0.8693	0.7052	0.8693
VGG16	398	2.2724	0.4205	1.9679	0.5173	1.9678	0.4986
EfficientNetB4	285	0.1081	0.9681	0.2853	0.9387	0.3156	0.9306
MobileNetV2	174	0.5833	0.8410	0.4863	0.8747	0.4862	0.8746

4.3 Analysis of EfficientNet-B4 in Jupyter Lab Platform

JupyterLab is a popular interactive development environment extensively utilized in the fields of machine learning and data science. It offers users a web-based interface to generate and exchange documents known as notebooks. These notebooks are capable of containing real-time code, equations, visual representations, and descriptive text.

After comparison from Table 4.6, EfficientNet-B4 model is the best model performing on our dataset on the local bird images where it consistently delivers excellent results in terms of training loss, training accuracy, validation loss, validation accuracy, test loss, and test accuracy.

From Figure 4.26, EfficientNet-B4 model perform the best against the test dataset where the model achieves an accuracy of 93.06% with the least test losses of 0.3156. This meets the minimum requirement for this project where the model must perform minimum 90% for the dataset of the local birds in Sarawak. This shows that the model performs well against new data and with little loss means the model generalise well against new input data that will be test in the mobile app.



```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(validation)
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
12/12 [==============================] - 44s 4s/step - loss: 0.3157 - accuracy: 0.9307
Test loss: 0.31567853689193726
Test accuracy: 0.9306666851043701
```

Figure 4.26: EfficientNet-B4 model test loss and accuracy performance.

From Figure 4.27, it shows the output of the prediction which were tested against the test dataset for the local birds in Sarawak. Above each bird picture shows what class of birds the model predicted and at the bottom of each image shows the real class of the image. From the figure it can be seen that the model accurately predicts the bird images. This means the model perform well against the test dataset of the local Sarawak birds used.

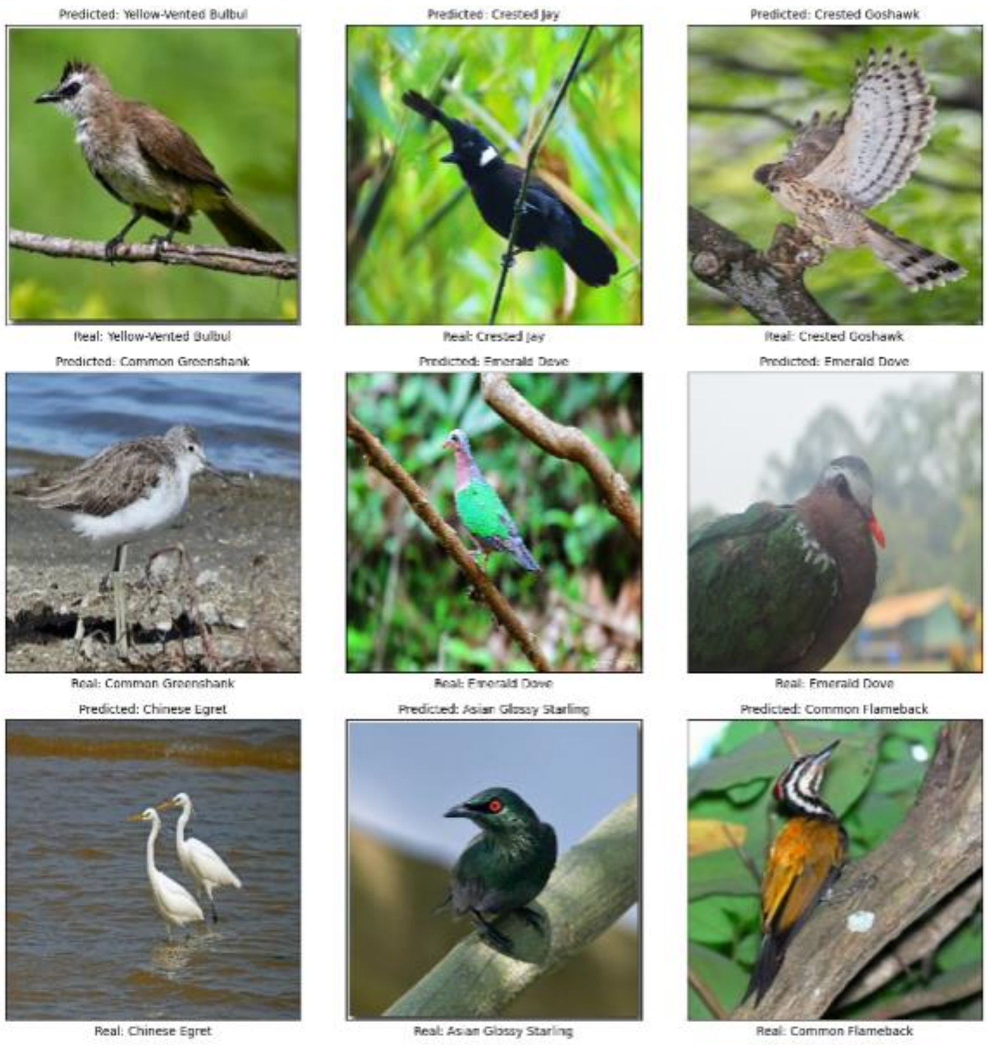


Figure 4.27: Result of prediction of the model using the test set.

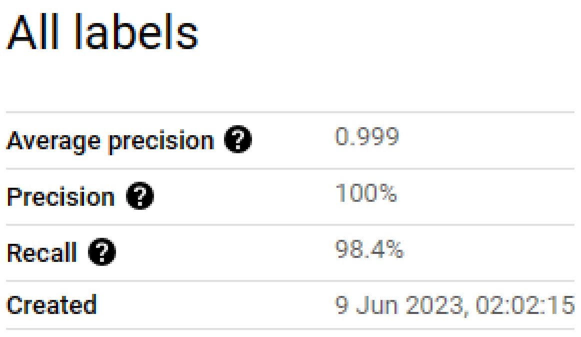
4.4 Analysis of Model After Optimisation in Google Cloud Platform using Vertex AI

Google Cloud Platform (GCP) is a comprehensive collection of cloud computing services offered by Google, providing a diverse array of tools and services to create, deploy, and manage applications and services in the cloud. GCP furnishes a scalable and dependable infrastructure, enabling businesses to utilize different computing resources, storage options, and networking capabilities.

While Vertex AI is a specialized machine learning platform available within Google Cloud Platform, designed to streamline and expedite the development and deployment of machine learning models. It presents a unified and user-friendly interface for tasks like data preparation, model training, evaluation, and deployment. Vertex AI is compatible with popular machine learning frameworks like TensorFlow and PyTorch, making it effortless to utilize existing models and code.

In this result, the model was actually trained with Python through Jupyter Lab platform and then exported to Google Cloud Platform to be further optimised. During optimisation process, the model undergoes processes such as quantisation, pruning, weight sharing and model distillation with default settings in Vertex AI.

From Figure 4.28 and 4.29, provided data findings lead to the conclusion that the classification model being examined demonstrates exceptional precision and recall performance. The average precision score of 0.999 indicates an extremely high level of accuracy in the model's positive predictions. This score, which evaluates the trade-off between precision and recall across various thresholds, signifies a remarkable precision level in the model's predictions.



All labels

Average precision	0.999
Precision	100%
Recall	98.4%
Created	9 Jun 2023, 02:02:15

Figure 4.28: Average precision, precision and recall precision from Vertex AI.

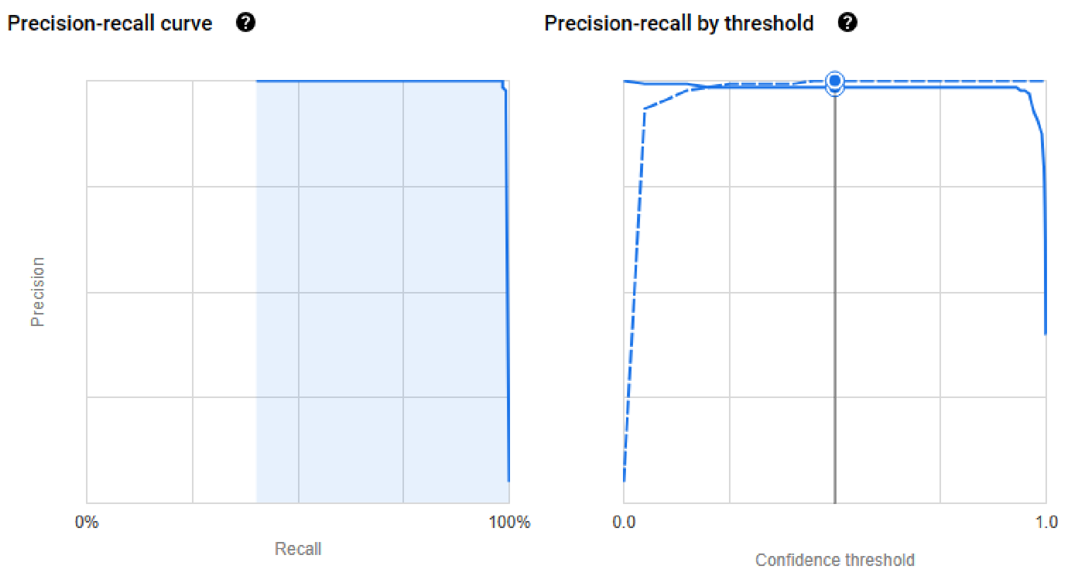


Figure 4.29: Graph of Precision-recall and Precision-recall by threshold.

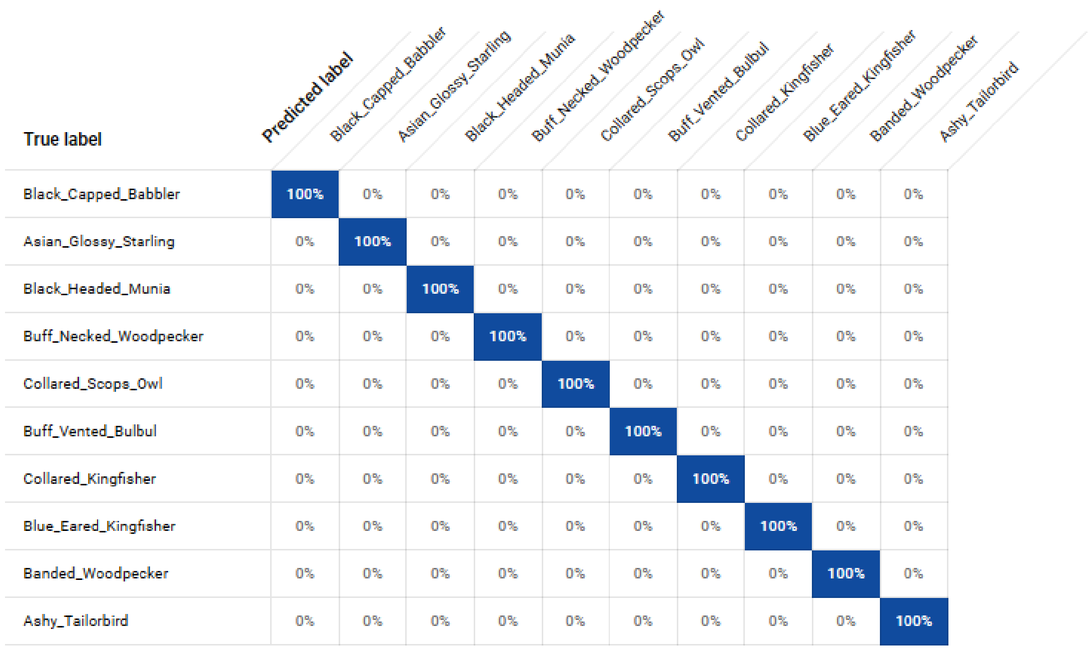
Furthermore, the precision value of 100% further emphasizes the exceptional accuracy of the model's positive predictions. Precision represents the ratio of correctly predicted positive instances to all positive predictions made by the model. Achieving a precision score of 100% suggests that the model has successfully identified all positive instances with no false positives. This highlights a strong confidence in the model's ability to accurately classify positive instances.

Additionally, the recall value of 98.4% indicates that the model has a robust ability to capture a significant proportion of the actual positive instances in the dataset. Recall value, also referred to as sensitivity, measures the model's capacity to correctly identify all positive instances. A recall score of 98.4% suggests that the model effectively identifies 98.4% of the actual positive instances, thereby minimizing false negatives.

In summary, the data findings provide a highly positive evaluation of the classification model's performance. With an average precision of 0.999 and a precision of 100%, the model's positive predictions exhibit exceptional accuracy, ensuring no false positives. Additionally, the recall rate of 98.4% indicates that the model successfully identifies a substantial portion of the actual positive instances, resulting in minimal false negatives. Collectively, these findings indicate that the model is highly reliable and proficient in accurately identifying positive instances while maintaining a high level of precision and recall.

While as for Table 4.7, where it represents the confusion matrix of the model shown in Vertex AI after optimisation. The confusion matrix table presented provides valuable insights into the performance of a multi-class classification model by illustrating the correspondence between predicted and true labels. Each row signifies a true label, while each column represents a predicted label. The values in the table indicate the proportion or percentage of instances falling into various combinations of true and predicted labels.

Table 4.7: Model Confusion Matrix.



True label \ Predicted label	Black_Capped_Babbler	Asian_Glossy_Starling	Black_Headed_Munia	Buff_Necked_Woodpecker	Collared_Scops_Owl	Buff_Vented_Bulbul	Collared_Kingfisher	Blue_Eared_Kingfisher	Banded_Woodpecker	Ashy_Tailorbird
Black_Capped_Babbler	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Asian_Glossy_Starling	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
Black_Headed_Munia	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%
Buff_Necked_Woodpecker	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%
Collared_Scops_Owl	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%
Buff_Vented_Bulbul	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%
Collared_Kingfisher	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%
Blue_Eared_Kingfisher	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%
Banded_Woodpecker	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%
Ashy_Tailorbird	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%

Upon analysing the confusion matrix, the following observations can be made where for the first row, which corresponds to the true label "Black_Capped_Babbler," the model exhibits 100% accuracy in predicting this label, as indicated by the value in the first column. Additionally, the remaining columns in this row have values of 0%, indicating that the model did not misclassify any other labels as "Black_Capped_Babbler."

Moving on to the second row, which represents the true label "Asian_Glossy_Starling," we find that the model accurately predicts this label with 100% precision, as denoted by the value in the second column. Similarly, the other columns in this row have values of 0%, indicating the absence of misclassifications for this label.

Continuing to the third row, corresponding to the true label "Black_Headed_Munia," the model achieves a perfect prediction rate of 100% for this particular label, as demonstrated by the value in the third column. Consistently, the remaining columns in this row hold values of 0%, signifying accurate classification without any false positives.

This pattern persists for the subsequent rows, with each true label being accurately predicted with 100% precision. In each case, all other columns in the respective rows have values of 0%, indicating the absence of misclassifications.

In conclusion, based on the provided confusion matrix table, the classification model demonstrates remarkable accuracy across all classes present in the dataset. It successfully predicts each true label without generating any false positives or false negatives. This exceptional performance serves as evidence of the model's effectiveness for this particular multi-class classification task.

4.5 Analysis of Mobile Application Testing Accuracy Result

A user-friendly interface is essential for a bird image classification mobile application, guiding users throughout the identification process. From Figure 4.30, the overview of how the mobile application was shown. It starts with the initial screen, acting as a welcome page, provides users with a concise overview of the app's functionality. It engages users right from the start by showcasing captivating bird images.

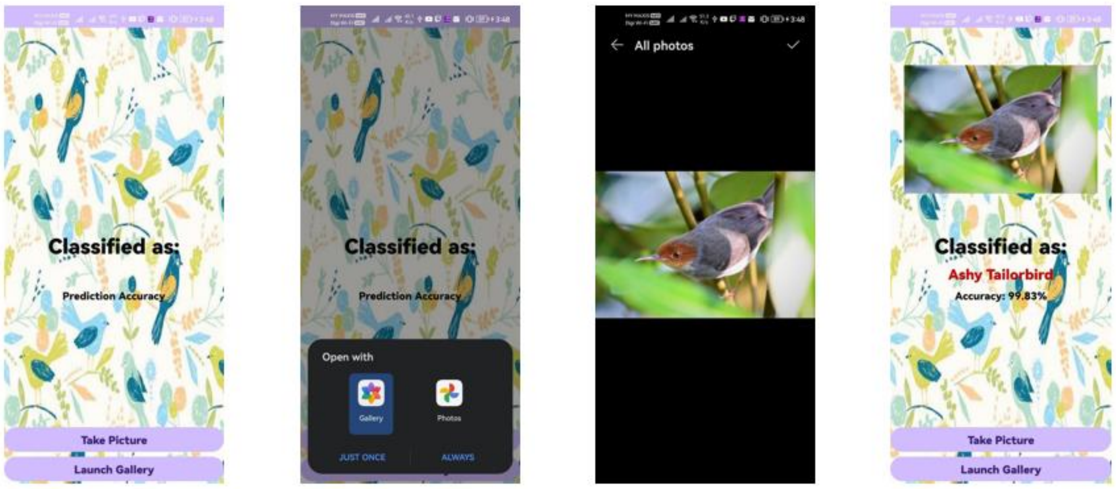


Figure 4.30: Mobile Application Overview.

The application allows users to capture bird images directly using device's camera, or alternatively, users can upload images from their gallery or storage. Clear instructions or prompts are provided to help users capture or select suitable images, ensuring optimal quality for accurate identification and classification.

Once an image is captured or uploaded, the application employs computer vision techniques where it sends a request to the model in Google Cloud Platform through API to process it. The image goes through feature extraction, where relevant visual characteristics are extracted to represent the bird. The backend processing is designed to be seamless and efficient, providing users with swift identification results.

The identification results are presented in a clear and comprehensible manner to the user. The app displays the predicted bird species name, accompanied by additional information such as its prediction confidence accuracy. Users have the option to explore multiple images and detailed descriptions of the bird species, facilitating further learning and exploration.

Next to test how well the mobile application performs on the new dataset, the mobile application was tested with new dataset called test dataset containing 30 images randomly obtain from the internet to get the average accuracy. This average accuracy will show how well the model performs and altogether to achieve the third objective where the need to evaluate how well the mobile applications perform especially in terms of its accuracy.

The provided Table 4.8 contains data about a collection of images and their respective predictions for the bird species depicted, which are then compared to the actual bird species (referred to as the ground truth). Each row in the table represents an image and includes the Image ID, the Bird Species (Ground Truth), the Predicted Bird Species, and the Accuracy percentage.

To calculate the real average accuracy of the model confidence towards the test dataset containing 30 images obtain randomly from the internet:

$$Average\ Accuracy\ (\%) = \frac{\sum Accuracy(\%)}{n} \tag{4.1}$$

$$= \frac{2644.42}{30}$$

$$= 88.15\%$$

Upon analysing the accuracy values, it becomes evident that the average accuracy across all the images is approximately 88.15%. This indicates that, on average, the predicted bird species align quite well with the actual bird species (ground truth).

Upon further examination of the accuracy values for each image, it is notable that a considerable number of predictions exhibit high levels of accuracy, surpassing 90%. For example, images 2, 10, 22, 24, 26, and 27 achieved accuracy percentages of 99.97%, 99.99%, 99.83%, 99.91%, 99.42%, and 99.99%, respectively. These predictions demonstrate a close correspondence between the predicted bird species and the actual bird species, indicating the effectiveness of the employed model or system.

Additionally, there are several predictions with accuracies ranging from 80% to 90%. Images 6, 7, 12, 13, 17, 18, 20, and 21 achieved accuracy percentages of 87.78%, 91.12%, 83.60%, 84.99%, 77.63%, 84.91%, 91.63%, and 89.91%, respectively. Although these accuracies are relatively high, there is still room for improvement, as some variation exists between the predicted bird species and the actual bird species.

Nevertheless, it is important to note that a few images received lower accuracy scores. For instance, image 5 had an accuracy of 52.87%, image 14 obtained an accuracy of 29.38%, and image 15 achieved an accuracy of 37.73%. These results indicate a significant mismatch between the predicted bird species and the actual bird species for these particular images.

Considering the overall average accuracy of approximately 88.15%, it can be concluded that the model or system utilized for predicting bird species performs reasonably well. However, there is potential for improvement, especially in terms of the accuracy of certain predictions. Addressing the cases with lower accuracy and striving for further enhancements would contribute to an increase in the reliability of the predictions.

Table 4.8: Result of Mobile App Testing with Test Dataset.

Image ID	**Bird Species (Ground Truth)**	**Predicted Bird Species**	**Accuracy (%)**
1	Asian Glossy Starling	Asian Glossy Starling	97.65
2	Banded Woodpecker	Banded Woodpecker	99.97
3	Black & Red Broadbill	Black & Red Broadbill	89.98
4	Ashy Tailorbird	Ashy Tailorbird	93.26
5	Black-Capped Babbler	Black-Throated Babbler	52.87
6	Black-Headed Bulbul	Black-Headed Bulbul	87.78
7	Black-Headed Munia	Black-Headed Munia	91.12
8	Black-Thighted Falconet	Black-Thighted Falconet	79.95
9	Black-Throated Babbler	Black-Throated Babbler	77.35
10	Blue-Eared Kingfisher	Blue-Eared Kingfisher	99.99
11	Blue-Throated Bee Eater	Blue-Throated Bee Eater	85.67
12	Brahminy Kite	Brahminy Kite	83.60
13	Brown-Throated Sunbird	Brown-Throated Sunbird	84.99

14	Buff Vented Bulbul	Black-Headed Bulbul	29.38
15	Buff-Necked Woodpecker	Banded Woodpecker	37.73
16	Chestnut-Naped Forktail	Chestnut-Naped Forktail	96.92
17	Chestnut-Rumped Babbler	Chestnut-Rumped Babbler	77.63
18	Chestnut-Winged Babbler	Chestnut-Winged Babbler	84.91
19	Chinese Egret	Chinese Egret	96.01
20	Collared Kingfisher	Collared Kingfisher	91.63
21	Collared Scops Owl	Collared Scops Owl	89.91
22	Common Flameback	Common Flameback	99.83
23	Common Greenshank	Common Greenshank	93.44
24	Common Iora	Common Iora	99.91
25	Cream-Vented Bulbul	Cream-Vented Bulbul	97.19
26	Crested Goshawk	Crested Goshawk	99.42
27	Crested Jay	Crested Jay	99.99
28	Emerald Dove	Spotted Dove	63.45
29	Spotted Dove	Spotted Dove	80.19
30	Yellow-Vented Bulbul	Yellow-Vented Bulbul	98.65

4.6 Comparison of research model with another researcher's model.

The table 4.9 compares different research models and their accuracy in classifying bird species using diverse datasets. The accuracy percentage reflects how well the models correctly identified the bird species.

This research utilized a self-captured dataset with 1878 images and 30 bird species. The research used the EfficientNet-B4 model, achieving an 88.15% accuracy. Transfer learning was applied, allowing the model to leverage pre-trained knowledge from other tasks. Al-Showarah et al (2021) used the Jordanian Bird Watching Association dataset with 4340 images and 434 bird species. Their research employed the VGG19 model with transfer learning, resulting in a 71% accuracy.

Wu et al (2021) worked with the CUB-200-2011 dataset comprising 11,788 images. They used the ResNeXt model with transfer learning and achieved an 84.43% accuracy. Rahman et al (2020) worked with the Bangladesh local bird's dataset containing 3500 images of various bird species. They used transfer learning with Inception-v3 and MobileNet models, achieving an impressive accuracy of 91.00%.

Wei et al (2017) used the Caltech-UCSD Birds (CUB) dataset with 5,994 images. Their deep learning model, Mask-CNN, achieved an accuracy of 80.20%. In summary, researchers employed different datasets and models to classify bird species. Transfer learning was a common technique used to improve model performance by leveraging pre-trained knowledge. Among the presented models, Rahman et al achieved the highest accuracy of 91.00% on the Bangladesh local bird's dataset using Inception-v3 and MobileNet models but this model does not apply for Sarawak local birds as this research focuses on the varieties of dataset specifically for Sarawak local birds.

Table 4.9:Comparison for the research model with another existing research.

Author	**Dataset/ Database**	**Size**	**Learning**	**Architecture/ Model**	**Accuracy %**
Myself(Alvarez)	Self-captured by Dr Mohamad Fizl Sidq bin Ramji	1878 images 30 species	Transfer Learning	EfficientNet-B4	88.15
Al-Showarah et al, 2021	Jordanian Bird Watching Association	4340 images, 434 bird species	Transfer Learning	VGG19	71
Wu et al, 2021	CUB-200-2011	11,788 images	Transfer Learning	ResNeXt	84.43
Rahman et al, 2020	Bangladesh local birds	3500 images,	Transfer Learning	Inception-v3, MobileNet	91.00
Wei et al, 2017	Caltech-UCSD Birds (CUB)	5,994 images	Deep Learning	Mask-CNN	80.20

4.7 Limitations of the Project

To begin with, despite efforts to optimize the bird species identification model using GCP's Vertex AI, there are inherent limitations in its accuracy. Factors like variations in lighting conditions, image quality, and the visual similarities among different bird species can impact the identification accuracy. Continuous refinement and updates to the model are necessary for ongoing improvements in accuracy.

Another aspect to consider is the limited diversity of the dataset used for training the model. In this case the dataset primarily represents specific regions or common bird species, the model's ability to accurately identify bird species from other regions or less common species may be restricted. To enhance the application's performance, it is important to expand the dataset by including a broader range of bird species and geographical regions.

Additionally, the project face limitations in terms of hardware computing power. The speed of processing and the latency of sending image data to the API for identification and receiving results is influenced by the computational capabilities of the hardware. Limited computing power led to increased processing time and potential bottlenecks. Optimization of the application and infrastructure is crucial to minimize latency and ensure efficient performance.

Moreover, compatibility with various platforms and devices is an important consideration. The mobile application developed on Android Studio does have compatibility constraints with specific Android versions or devices. Extensive testing and potential modifications are required to ensure compatibility across a wide range of Android versions and devices with varying hardware configurations and screen resolutions.

In addition, the user interface (UI) and user experience (UX) aspects of the mobile application play a crucial role. While the focus is often on the model and API backend, providing an intuitive and user-friendly interface significantly enhances the overall usability and appeal of the application.

Ethical considerations should also be addressed, including privacy, data usage, and potential biases in the dataset or model. Obtaining proper consent, ensuring transparent data practices, and mitigating biases that may arise during the training process are vital for responsible AI development.

Lastly, the limitations of hardware computing power need to be taken into account. The computational demands of running the model and processing image data is substantial, and the available hardware have limitations in processing power and memory capacity. It is crucial to ensure that the hardware used for hosting the model and running the application can adequately handle the computational requirements to ensure smooth operation.

4.8 Summary

This chapter offers a comprehensive overview of the outcomes and analysis derived from the utilization of five image classification models for the identification of local bird species. The models investigated include ResNet-50, InceptionV3, VGG16, MobileNetV2, and EfficientNet-B4. Performance evaluation involved assessing various parameters such as training time, training loss, validation loss, training accuracy, validation accuracy, test loss, and test accuracy.

Based on a comparison of these parameters, the most effective model was determined which is the EfficientNet-B4 for further examination. Subsequently, the chosen model underwent optimization by deploying it on Google Cloud Platform using Vertex AI. Optimization strategies included an analysis of the precision-recall graph and precision-recall threshold to ascertain the precision and recall rates of the model.

The chapter also encompasses an evaluation of the mobile application's performance in association with the model. Testing the mobile app with a fresh dataset known as the test set facilitated the calculation of its average accuracy in identifying bird species. The average accuracy obtained was 88.15% which indicates the mobile application perform well in identifying the species of local Sarawak birds.

In summary, the chapter provides an extensive summary of the results and analysis obtained from the image classification models, including the identification of the best-performing model, its optimization on Google Cloud Platform, and the evaluation of the mobile app's accuracy. The chapter's contributions are significant in terms of advancing the identification of bird species through image classification.

CHAPTER 5

CONCLUSION

5.1 Future Works and Improvements

The bird species identification project has demonstrated its ability to accurately identify bird species and engage users in bird conservation. However, there are several areas where further work and improvements can be explored to enhance the project's capabilities and maximize its impact.

One key area for improvement is expanding the project's compatibility to other mobile platforms, such as iOS. Currently, the project may be limited to a specific platform, which restricts its reach and potential user base. By extending compatibility to other popular mobile operating systems, the project would become accessible to a wider audience, allowing more individuals to participate in bird identification and conservation efforts.

Another important aspect to consider is the incorporation of real-time updates and notifications. Providing users with timely information about bird sightings, conservation initiatives, and relevant news would greatly enhance their engagement with the application. Real-time updates could notify users about rare bird sightings in their area or inform them about ongoing conservation projects, creating a sense of urgency and motivating users to actively contribute to the preservation of bird species.

Furthermore, integrating crowdsourcing and citizen science into the project can bring significant benefits. Allowing users to contribute their bird species observations through the application would not only expand the dataset but also help validate the accuracy of the identification model. By engaging users in citizen science initiatives, the project can tap into the collective knowledge and efforts of bird enthusiasts worldwide, ultimately improving the precision and reliability of species identification.

Localization and language support are also critical considerations for wider participation. Adapting the application to support multiple languages and localizing the user interface would overcome language barriers and make the project accessible to users from diverse regions and cultures. This inclusiveness would foster greater engagement and involvement from individuals around the world, facilitating global collaboration in bird conservation efforts.

To encourage collaboration and knowledge sharing among users, integrating social sharing and community building features into the application is crucial. Enabling users to share their bird sightings, photos, and identification successes on social media platforms would create a virtual community of bird enthusiasts. This community-driven approach would promote collaborative learning, facilitate mentorship opportunities, and allow users to engage in discussions and exchanges about bird species, conservation strategies, and environmental challenges.

Another avenue for improvement lies in exploring the integration of environmental sensors and leveraging existing data sources. By incorporating environmental sensors or utilizing available data on weather patterns, bird migration, and habitat conditions, the identification model could be enriched with additional contextual information. This integration would enhance the accuracy of species identification and provide users with a more comprehensive understanding of the bird species they encounter. Moreover, incorporating such data sources would contribute to scientific research on bird behavior, migration patterns, and ecological changes.

Augmented reality (AR) enhancements present an exciting opportunity to make the bird species identification project more immersive and interactive. Introducing AR features that overlay digital information about identified bird species onto real-time camera views would offer users a captivating and educational experience. For instance, users could point their phone's camera at a bird and instantly receive information about its habitat, diet, and unique characteristics. This interactive approach would foster a deeper appreciation for birds and their natural environment.

Additionally, implementing offline functionality would provide valuable features for users. Allowing individuals to access the identification model and relevant information even without an internet connection would enhance the usability and practicality of the application. This feature would be particularly useful for bird enthusiasts exploring remote areas with limited connectivity or embarking on outdoor expeditions where internet access may be scarce. Offline functionality ensures that users can continue to identify bird species and access relevant information, regardless of their location or internet availability.

By considering and implementing these future works and improvements, the bird species identification project has the potential to evolve into a comprehensive tool that not only assists users in accurately identifying bird species but also actively engages them in bird conservation efforts. The project can become a valuable resource for bird enthusiasts, researchers, and conservationists, contributing to the preservation of avian biodiversity and promoting a deeper understanding of the natural world.

5.2 Conclusion

In summary, this project has successfully developed a mobile application for bird species identification, employing an integrated approach that combines Jupyter Lab for model development, Google Cloud Platform with Vertex AI for model optimization, and Android Studio for application creation. Through this comprehensive methodology, it has achieved a highly efficient and accurate bird species identification system that is accessible to bird enthusiasts and researchers.

The project began by creating a robust bird species identification model in Jupyter Lab, utilizing a diverse dataset containing various bird species. Leveraging machine learning algorithms and deep learning techniques, the trained model to recognize different bird species based on their unique characteristics. This initial step laid the foundation for the subsequent stages of the project.

In the comparison analysis of transfer learning models, including VGG16, MobileNetV2, InceptionV3, ResNet-50, and EfficientNet-B4, EfficientNet-B4 stood out as the most suitable model for this research. Its superior performance in terms of accuracy, efficiency, and computational resource utilization made it the ideal choice for achieving the project's goals. EfficientNet-B4 demonstrated exceptional capability in accurately identifying bird species, while also optimizing computational resources, enabling faster processing times and ensuring a smooth user experience.

To optimize the model's performance and efficiency, the model was seamlessly transitioned to the Google Cloud Platform, capitalizing on the advanced capabilities of

Vertex AI. Leveraging the platform's powerful tools and resources facilitated model optimization, resulting in improved accuracy and speed. Utilizing the cloud infrastructure ensured that the model could handle a substantial number of requests from the mobile application without compromising responsiveness.

Subsequently, the integrated model was integrated into the mobile application developed in Android Studio. The application features a user-friendly interface and intuitive design, enabling users to capture bird images through their smartphone cameras and receive instant and accurate species identifications. The API, which connects the application to the cloud-based model, efficiently processes user requests, providing swift responses and ensuring a seamless user experience.

The successful implementation of this mobile application holds significant implications for bird enthusiasts, researchers, and conservationists. The application serves as a powerful and accessible tool for identifying bird species in the wild, enabling users to deepen their knowledge and appreciation of avian biodiversity. Furthermore, the application's crowdsourcing potential allows users to contribute to ongoing bird conservation efforts, fostering a global community of citizen scientists and assisting in the monitoring of bird populations and habitats.

While the project has achieved commendable results, there are several areas for future exploration and improvement. Expanding the model's dataset to include a broader range of bird species and environmental conditions could further enhance its accuracy and generalizability. Additionally, incorporating real-time updates and integrating environmental sensor data into the model could provide users with dynamic and contextually rich bird identification information.

In conclusion, the creation of this mobile application for bird species identification showcases the potential of integrating cutting-edge technologies in the fields of machine learning, cloud computing, and mobile app development. The seamless collaboration between Jupyter Lab, Google Cloud Platform with Vertex AI, and Android Studio has resulted in a user-friendly, efficient, and accurate tool that contributes to bird conservation efforts and fosters a deeper understanding and appreciation of avian biodiversity. As technology continues to advance, this project lays the groundwork for further innovation and exploration in the realm of species identification and wildlife conservation.

REFERENCES

[1] 'Birding in Sarawak - The most number of Important Bird Areas in Malaysia'. https://chinese.sarawaktourism.com/attraction/birding-sarawak-malaysia/ (accessed Jan. 12, 2023).

[2] 'Bird Watching in Sarawak - Bird Watching in Sarawak'. https://birdwatching.sarawaktourism.com/ (accessed Jan. 12, 2023).

[3] 'Visitor Statistics - Sarawak Forestry Corporation'. https://sarawakforestry.com/visitor-statistics/ (accessed Jan. 12, 2023).

[4] 'Sarawak'. https://birdsmalaysia.my/sarawak/ (accessed Jan. 12, 2023).

[5] Y. P. Huang and H. Basanta, 'Recognition of Endemic Bird Species Using Deep Learning Models', *IEEE Access*, vol. 9, pp. 102975–102984, 2021, doi: 10.1109/ACCESS.2021.3098532.

[6] S. A. Al-Showarah and S. T. Al-Qbailat, 'Birds Identification System using Deep Learning'. [Online]. Available: www.ijacsa.thesai.org

[7] M. M. Rahman, A. Amin Biswas, A. Rajbongshi, A. Majumder, and C. Realty, 'Recognition of Local Birds of Bangladesh using MobileNet and Inception-v3', 2020. [Online]. Available: www.ijacsa.thesai.org

[8] P. Wu, G. Wu, X. Wu, X. Yi, and N. Xiong, 'Birds Classification Based on Deep Transfer Learning', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Science and Business Media Deutschland GmbH, 2021, pp. 173–183. doi: 10.1007/978-3-030-74717-6_19.

[9] J. Niemi and J. T. Tanttu, 'Deep learning case study for automatic bird identification', *Applied Sciences (Switzerland)*, vol. 8, no. 11, Oct. 2018, doi: 10.3390/app8112089.

[10] 'COTERC Partnership - Environmental & Wildlife ManagementEnvironmental & Wildlife Management | Cégep Vanier College'. https://www.vaniercollege.qc.ca/ewm/coterc-partnership/ (accessed Jan. 16, 2023).

[11] 'Ten Tips to Help Avoid Mis-Identifying Birds - Birding World'. http://birding-world.com/ten-tips-help-avoid-mis-identifying-birds/ (accessed Jan. 16, 2023).

[12] 'How to Identify Birds | Audubon'. https://www.audubon.org/content/how-identify-birds (accessed Jan. 14, 2023).

[13] 'What is Deep Learning? | IBM'. https://www.ibm.com/topics/deep-learning (accessed Jan. 16, 2023).

[14] M. Hussain, J. J. Bird, and D. R. Faria, 'A study on CNN transfer learning for image classification', in *Advances in Intelligent Systems and Computing*, Springer Verlag, 2019, pp. 191–202. doi: 10.1007/978-3-319-97982-3_16.

[15] X. S. Wei, C. W. Xie, J. Wu, and C. Shen, 'Mask-CNN: Localizing parts and selecting descriptors for fine-grained bird species categorization', *Pattern Recognit*, vol. 76, pp. 704–714, Apr. 2018, doi: 10.1016/j.patcog.2017.10.002.

[16] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C. Rudin, 'This Looks Like That: Deep Learning for Interpretable Image Recognition'.

[17] S. J. Hong, Y. Han, S. Y. Kim, A. Y. Lee, and G. Kim, 'Application of deep-learning methods to bird detection using unmanned aerial vehicle imagery', *Sensors (Switzerland)*, vol. 19, no. 7, Apr. 2019, doi: 10.3390/s19071651.

[18] I. J. Jacob and P. E. Darney, 'Design of Deep Learning Algorithm for IoT Application by Image based Recognition', *Journal of ISMAC*, vol. 3, no. 3, pp. 276–290, Aug. 2021, doi: 10.36548/jismac.2021.3.008.

[19] S. Raj, S. Garyali, S. Kumar, B. E. Scholar, and S. Shidnal, 'Image based Bird Species Identification using Convolutional Neural Network'. [Online]. Available: www.ijert.org

[20] Srijan, Samriddhi, and D. Gupta, 'Mobile Application for Bird Species Identification Using Transfer Learning', in *3rd IEEE International Conference on Artificial Intelligence in Engineering and Technology, IICAIET 2021*, Institute of Electrical and Electronics Engineers Inc., Sep. 2021. doi: 10.1109/IICAIET51634.2021.9573796.

[21] G. Zhong, S. Yan, K. Huang, Y. Cai, and J. Dong, 'Reducing and Stretching Deep Convolutional Activation Features for Accurate Image Classification', *Cognit Comput*, vol. 10, no. 1, pp. 179–186, Feb. 2018, doi: 10.1007/s12559-017-9515-z.

[22] X.-S. Yang and Institute of Electrical and Electronics Engineers, *Proceedings of the World Conference on Smart Trends in Systems, Security and Sustainability (WS4 2020) : July 27-28, 2020, virtual conference*.

[23] Institute of Electrical and Electronics Engineers, *OCEANS 2019 MTS/IEEE SEATTLE*.

[24] P. Wu, G. Wu, X. Wu, X. Yi, and N. Xiong, 'Birds Classification Based on Deep Transfer Learning', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Science and Business Media Deutschland GmbH, 2021, pp. 173–183. doi: 10.1007/978-3-030-74717-6_19.

[25] N. E. Khalifa, M. Loey, and S. Mirjalili, 'A comprehensive survey of recent trends in deep learning for digital images augmentation', *Artif Intell Rev*, vol. 55, no. 3, pp. 2351–2377, Mar. 2022, doi: 10.1007/s10462-021-10066-4.

[26] F. Z. el Bouni, T. el Hariri, C. Zouitni, I. ben Bahva, H. el Aboui, and A. el Ouaazizi, 'Bird image recognition and classification using Watson visual recognition services from IBMCloud and Conventional Neural Network (CNN)', in *3rd International Conference on Electrical, Communication and Computer Engineering, ICECCE 2021*, Institute of Electrical and Electronics Engineers Inc., Jun. 2021. doi: 10.1109/ICECCE52056.2021.9514269.

[27] X.-S. Yang and Institute of Electrical and Electronics Engineers, *Proceedings of the World Conference on Smart Trends in Systems, Security and Sustainability (WS4 2020) : July 27-28, 2020, virtual conference*.

[28] 'Merlin Bird ID – Free, instant bird identification help and guide for thousands of birds – Identify the birds you see'. https://merlin.allaboutbirds.org/ (accessed Jan. 17, 2023).

[29] 'National Geographic Birds: Field Guide to North America by National Geographic Society'. https://appadvice.com/app/national-geographic-birds-field-guide-to-north-america/315268465 (accessed Jan. 17, 2023).

[30] 'Peterson Birds North America - Free download and software reviews - CNET Download'. https://download.cnet.com/Peterson-Birds-North-America/3000-18495_4-78373956.html (accessed Jan. 16, 2023).

[31] ‘Birds | Audubon’. https://www.audubon.org/birds (accessed Jan. 17, 2023).

[32] ‘iBird Pro^{TM} - Why the Switch?’ https://www.ibird.com/why-switch/why-switch.html (accessed Jan. 16, 2023).

[33] ‘Expert’. https://expert.unimas.my/profile/2729 (accessed Jan. 14, 2023).

[34] A. S. Qureshi, A. Khan, A. Zameer, and A. Usman, ‘Wind power prediction using deep neural network based meta regression and transfer learning’, *Applied Soft Computing Journal*, vol. 58, 2017, doi: 10.1016/j.asoc.2017.05.031.

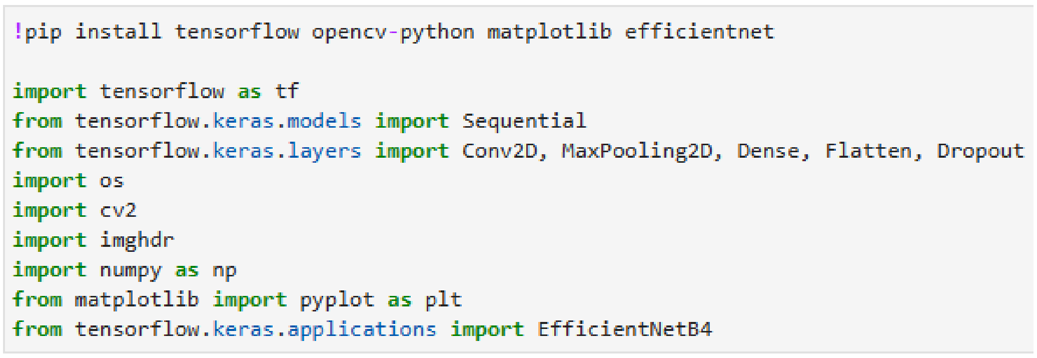
[35] J. Gorospe, R. Mulero, O. Arbelaitz, J. Muguerza, and M. Á. Antón, ‘A generalization performance study using deep learning networks in embedded systems’, *Sensors (Switzerland)*, vol. 21, no. 4, 2021, doi: 10.3390/s21041031.

APPENDIX

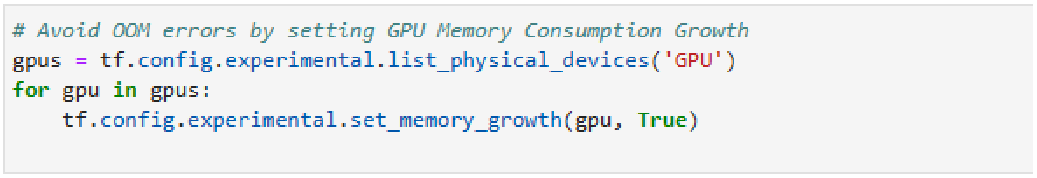
All of the datasets, power point slides, codes for the model in Jupyter Lab platform, the XML files and code for the Android Studio files is in this google drive link below:

https://drive.google.com/drive/folders/13jR_8s3PsCngp0XVt8Vr5mywq5lKvlAF?usp=sharing

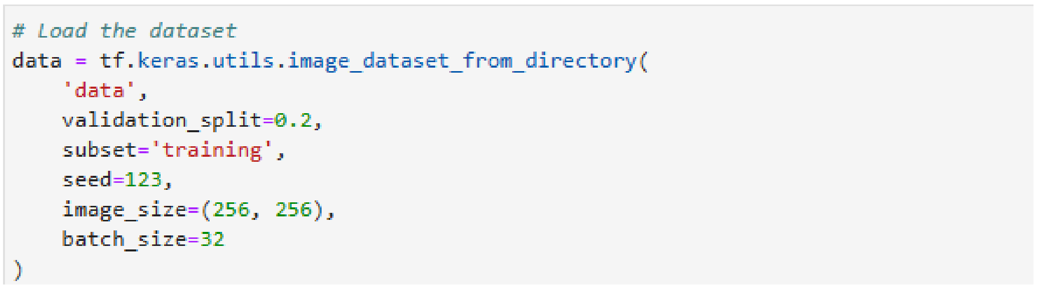
Code for EfficientNet-B4 model in Jupyter Lab platform.



```
!pip install tensorflow opencv-python matplotlib efficientnet

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
import os
import cv2
import imghdr
import numpy as np
from matplotlib import pyplot as plt
from tensorflow.keras.applications import EfficientNetB4
```



```
# Avoid OOM errors by setting GPU Memory Consumption Growth
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```



```
# Load the dataset
data = tf.keras.utils.image_dataset_from_directory(
    'data',
    validation_split=0.2,
    subset='training',
    seed=123,
    image_size=(256, 256),
    batch_size=32
)
```

```
# Split the dataset into train and validation sets
train = data
validation = tf.keras.utils.image_dataset_from_directory(
    'data',
    validation_split=0.2,
    subset='validation',
    seed=123,
    image_size=(256, 256),
    batch_size=32
)
```

```
# Retrieve class names
class_names = data.class_names
num_classes = len(class_names)
```

```
# Configure dataset for performance
AUTOTUNE = tf.data.experimental.AUTOTUNE
train = train.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
validation = validation.cache().prefetch(buffer_size=AUTOTUNE)
```

```
# Data augmentation
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.RandomZoom(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./255)  # Normalize pixel values
])
```

```
# Load the EfficientNet-B4 model without the top layers
base_model = EfficientNetB4(weights='imagenet', include_top=False, input_shape=(256, 256, 3))

# Freeze only the first few layers and unfreeze the rest
for layer in base_model.layers[:-20]:
    layer.trainable = False
for layer in base_model.layers[-20:]:
    layer.trainable = True

# Create a new model by adding the EfficientNet-B4 base model and a custom classifier on top
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```
# Build the model
model.build((None, 256, 256, 3))
```

```
# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
model.summary()
```

```
# Early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
```

```
# Train the model
epochs = 20
hist = model.fit(train, epochs=epochs, validation_data=validation, callbacks=[early_stopping])
```

```
# Print the training data in a tabular format
print("Epoch\tTrain Loss\tTrain Accuracy\tVal Loss\tVal Accuracy")
for epoch in range(epochs):
    train_loss = hist.history['loss'][epoch]
    train_accuracy = hist.history['accuracy'][epoch]
    val_loss = hist.history['val_loss'][epoch]
    val_accuracy = hist.history['val_accuracy'][epoch]
    print(f"{epoch + 1}\t{train_loss:.4f}\t\t{train_accuracy:.4f}\t\t{val_loss:.4f}\t\t{val_accuracy:.4f}")
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(validation)
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
# Visualize training history
fig, ax = plt.subplots(2, 1, figsize=(12, 8))
ax[0].plot(hist.history['loss'], color='teal', label='train_loss')
ax[0].plot(hist.history['val_loss'], color='orange', label='val_loss')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Loss')
ax[0].legend(loc='upper right')

ax[1].plot(hist.history['accuracy'], color='teal', label='train_accuracy')
ax[1].plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Accuracy')
ax[1].legend(loc='lower right')

plt.show()
```

```
plt.figure(figsize=(12, 12))  # Increase the figure size for better visibility

for i, (images, labels) in enumerate(validation.take(1)):  # Add an index variable to enumerate the batches
    predictions = model.predict(images)
    predicted_labels = np.argmax(predictions, axis=1)

    for j in range(9):
        ax = plt.subplot(3, 3, j + 1)
        plt.imshow(images[j].numpy().astype("uint8"))
        plt.title("Predicted: " + class_names[predicted_labels[j]], fontsize=10)  # Use a smaller font size for the title
        plt.xlabel("Real: " + class_names[labels[j]], fontsize=10)  # Use a smaller font size for the x-label
        plt.tick_params(axis='both', which='both', bottom=False, left=False)  # Remove the ticks on both axes
        plt.xticks([])  # Remove the x-axis tick labels
        plt.yticks([])  # Remove the y-axis tick labels

plt.tight_layout()  # Adjust the spacing between subplots
plt.show()
```