Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR'S THESIS PROJECT

TITLE: Computer vision for bird strike prevention

DEGREE: Bachelor's degree in Aerospace Systems Engineering and Telecommunications Systems

AUTHOR: Adrià Ibáñez i Boix

ADVISORS: Alberto Burgos Plaza & Francisco Javier Mora Serrano

DATA: 12th July 2023

Overview

Collisions with birds cause damage to aircraft and in some cases can even cause air travel accidents. According to data from international organizations such as the Federal Aviation Administration (FAA), the radar-based tools currently used to address this problem do not solve it, as there is no indication of a decrease in the number of bird strikes. Early detection and notification to pilots of the presence of birds is key to trying to minimize the possibility that bird impacts can occur.

The objective of this project is to improve bird detection capacity in the airport environment. To achieve this goal, this work proposes that the solution could be the use of artificial intelligence based devices and computer vision. To test this hypothesis, a model based on convolutional neural networks (CNN) is selected, trained and deployed on a device for testing.

To do this, research is carried out on the different strategies used to solve problems with artificial intelligence and the performance of pre-trained classifier and detector models available. To select the computer board where the model will be deployed, a discussion of Raspberry Pi's market performance is made. A collection of bird images is made for training the model. The prototype will finally consist of deploying the model on a Raspberry Pi that through a script in Python programming language is able to automatically notice birds in the real world using a camera connected to the Raspberry Pi. If any detection occurs, the model is capable of making a notification that could serve to anticipate impacts and thus allow appropriate preventive measures to be taken beforehand.

In conclusion, this technology shows great potential to support existing solutions today. Theoretical results with validation images show accuracy and recall parameters above 90% but experimental tests with the prototype do not allow for a conclusive judgment due to limitations regarding the training data set.

Títol: Prevenció d'impacte amb ocells mitjançant visió per computador

Autor: Adrià Ibáñez i Boix

Directors: Alberto Burgos Plaza i Francisco Javier Mora Serrano

Data: 12 de juliol del 2023

Resum

La col·lisió amb aus provoquen danys a les aeronaus i en alguns casos fins i tot poden provocar accidents aeris. Segons les dades ofertes per organitzacions internacionals com la Federal Aviation Administration (FAA), les eines basades en radar que s'utilitzen actualment per abordar aquest problema no el resolen, ja que no hi ha cap indici de descens en el nombre de d'imapactes amb aus. La detecció i notificació precoç als pilots de la presència d'ocells és clau per mirar de minimitzar la possibilitat que els impactes amb ocells es puguin arribar a produir.

L'objectiu d'aquest projecte és millorar la capacitat de detecció d'aus en l'entorn aeroportuari. Per assolir aquest objectiu, es planteja en aquest treball que la solució podria passar per l´ús de dispositius basats en la intel·ligència artificial i visió per computador. Per posar a prova aquesta hipòtesis, es selecciona i s'entrena un model basat en xarxes neuronals convolucionals (en anglès, CNN) i es desplega en un dispositiu per posar-lo a prova.

Per fer-ho, es realitza una recerca sobre les diferents estratègies utilitzades per resoldre problemes amb intel·ligència artificial i les prestacions dels models classificadors i detectors pre-entrenats disponibles. Per seleccionar la placa computadora on es desplegarà el model es fa una discussió de les prestacions de Raspberry Pi al mercat. Es fa una recopilació d'imatges d'aus per fer l'entrenament del model. El prototip consistirà finalment en el desplegament del model en una Raspberry Pi que a través d'un script en llenguatge de programació Python sigui capaç de localitzar automàticament ocells al món real mitjançant una càmera connectada a la Raspberry Pi. Si es produeix alguna detecció el model és capaç de fer una notificació que podria servir per anticipar impactes i així permetre prendre mesures preventives adequades abans.

En conclusió, aquesta tecnologia mostra un gran potencial per a fer de suport a les solucions existents a avui dia. Els resultats teòrics amb imatges de validació mostren paràmetres d'accuracy i de recall per sobre del 90% però els tests experimentals amb el prototip no en permeten fer un judici concloent a causa de les limitacions pel que fa al conjunt de dades d'entrenament.

INDEX

CHAPTER 1. INTRODUCTION

Motivation

Computer vision is the subcategory of the emerging field of artificial intelligence (AI) that enables computers to obtain information from images and then taking action in accordance. Nowadays there are multiple applications for it, from inspecting crops in agriculture to recognizing tumors in medicine. The goal of this project is to enhance air transportation safety by utilizing computer vision technology in the field of aeronautics. A state-of-the-art device able to acknowledge the presence of birds in the vicinity will be developed and set up with the objective of preventing bird strike collisions.
An important consideration before beginning the project is to determine to what extent bird strikes are a threat for air transportation and therefore if such a project can be justified. The Federal Aviation Administration keeps an updated database of bird strikes reported by the pilots and provides some information about them.
The chart below shows the reported bird strikes in the U.S from the year 1990 when they started collecting data to this day. The data from years 2020 to 2022 are omitted as the COVID-19 pandemic had an impact on air transportation and therefore would not provide trustworthy information about the phenomenon.

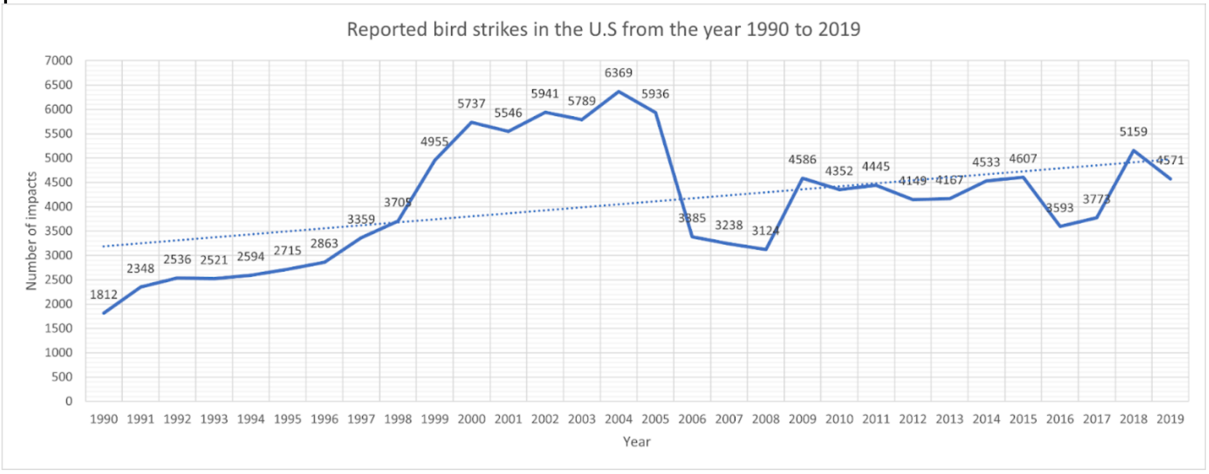


Fig. 1.1: Evolution of reported bird strike in the U.S from 1990 to 2019. [1]

Available data show that 57% of the pilots were not informed of the presence of birds prior to the bird strike, see Fig. 1.2.

The available information shows that most of the bird strikes occur during the day, when most of the flights are operated and there is light. This information is particularly relevant as the computer vision system has to be adapted to every condition so as to obtain the best possible accuracy. This means that during the night a camera would not work and during the evening would not be as reliable. This could be solved using a hybrid approach with a camera and an infra-red camera using two different computer vision models.

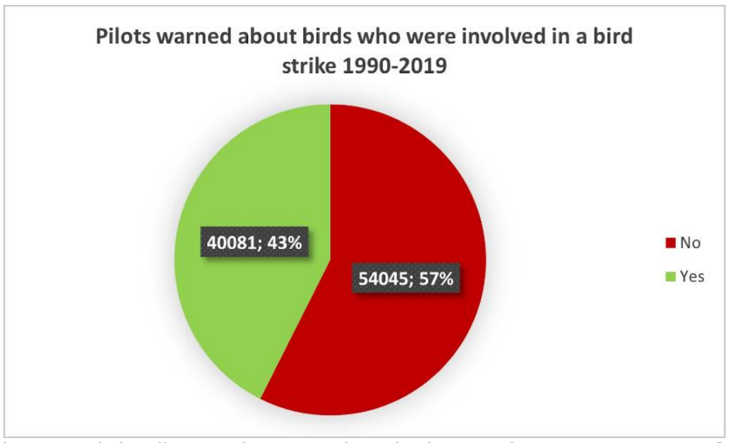


Fig. 1.2: Bird strike previous warnings in the U.S from 1990 to 2019. [1]

Another variable that is considered in the FAA database is the phase of the flight in which the strike took place. As the graph shows, most of the incidents (43%) take place during the approach phase, the landing roll follows up with an 18%, the same percentage of the take-off run and the next most important is the climb phase with the 16% of the reported strikes. It appears clear that most of the reported bird strikes take place in the vicinity of an airport during the most critical maneuvers.

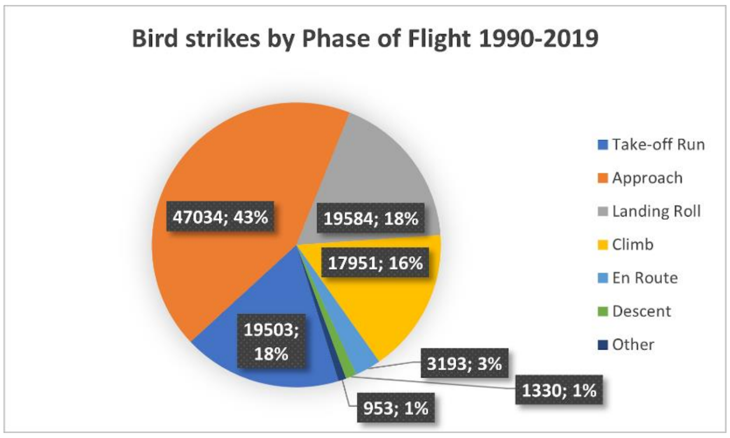


Fig. 1.3: Bird strike per phase in the U.S from 1990 to 2019. [1]

The profile of aircraft affected by bird strike is also heterogeneous. Most of the reported strikes correspond to large aircrafts. The distribution in the graph has been made according to the number of seats. Aircraft with capacity less than 50 seats are considered small, medium are those below 150, large are the ones who do not reach 220 seats and over that are the jumbo.

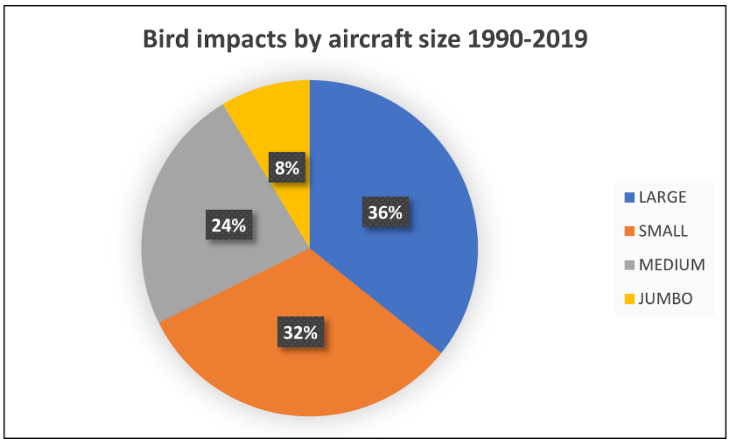


Fig. 1.4: Bird strike distribution by plane size in the U.S from 1990 to 2019. [1]

The International Civil Aviation Organization (ICAO) requires member States to collect data from aircraft and airport operators regarding bird strikes [2]. The effectiveness of avian radars currently used in airports was evaluated by the FAA in the early 2010's, results show that the majority of large single birds seen by field observers within 4 km of the radar were tracked by the radar about 30 percent of the time. Flocks of large birds, including those that were located several nautical miles away, were tracked by the radar 40 to 80 percent of the time. According to the researchers, radar can be a useful tool for monitoring bird flock activity at airports, but less so for monitoring large single birds [3].

Environmental impact

This project involves making a prototype device that could potentially prevent bird strikes by using electronic components, a battery and a cover box. These materials and components have environmental impacts at different stages of their life cycle, such as mining, manufacturing, transportation, energy use and waste disposal. Therefore, we need to justify our choices of materials and components to minimize the environmental damage and pollution caused by them. However, this project could also have a positive impact on the environment by protecting the birds and the aircraft passengers from collisions and reducing the need for repairs to damaged parts resulting from bird strikes.

Objective

The objective of this project is to enhance the detection rate of avian presence through the implementation of an artificial intelligence-based solution. This could potentially improve the quality and reliability of notifications of bird activity in the vicinity of aircraft transmitted to pilots, thereby addressing the current limitations of the radar-based technology. The following sub-objectives are defined:

1. Study of the background and the state of the art: This document presents a detailed analysis of the development, functionality and performance of computer vision systems and analyses the different deep learning strategies that can be used. So as to better understand what deep learning means in artificial intelligence applications, the basic concepts on image processing and coding will be explained.

2. Implementation of the model in a prototype: In order to assess the potential of this technology for practical applications in notifying bird presence, a prototype device for real-world evaluation will be designed and set-up.

Methodology

The project can be structured into five distinct stages: Identification of the observed problem, Definition of the objectives, Design and development, Demonstration, and Evaluation. Each stage encompasses a defined scenario and is accompanied by a specific set of tools and activities to facilitate the completion of the project. Fig. 1.5 summarizes all the information.

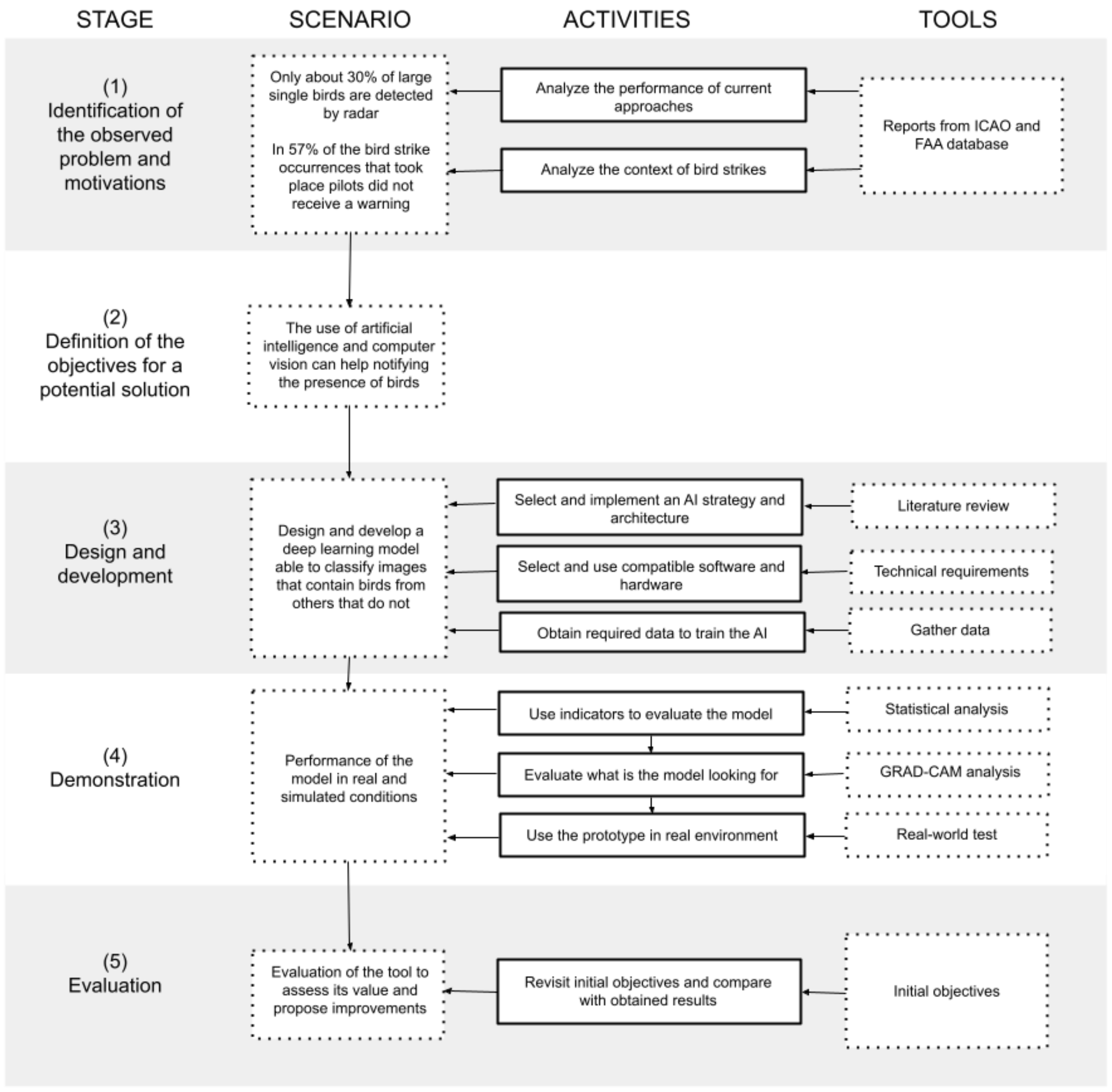

STAGE
SCENARIO
ACTIVITIES
TOOLS
(1) Identification of the observed problem and motivations
Only about 30% of large single birds are detected by radar
In 57% of the bird strike occurrences that took place pilots did not receive a warning
Analyze the performance of current approaches
Analyze the context of bird strikes
Reports from ICAO and FAA database
(2) Definition of the objectives for a potential solution
The use of artificial intelligence and computer vision can help notifying the presence of birds
(3) Design and development
Design and develop a deep learning model able to classify images that contain birds from others that do not
Select and implement an AI strategy and architecture
Select and use compatible software and hardware
Obtain required data to train the AI
Literature review
Technical requirements
Gather data
(4) Demonstration
Performance of the model in real and simulated conditions
Use indicators to evaluate the model
Evaluate what is the model looking for
Use the prototype in real environment
Statistical analysis
GRAD-CAM analysis
Real-world test
(5) Evaluation
Evaluation of the tool to assess its value and propose improvements
Revisit initial objectives and compare with obtained results
Initial objectives

Fig. 1.5: Methodology chart

Document layout

Correspondingly to the stages defined in the methodology this document can be subdivided into four parts in the following way:

- Introduction and definition of the objectives:

  Chapter 1: Introduction
  In the introductory chapter, the context and motives of the project are presented, the structure is established and the objective is outlined.

Chapter 2: Computer Vision, concepts and definitions
This chapter provides an overview of fundamental concepts related to computer vision and deep learning models to facilitate the reader's understanding of the document.

- Design and development

Chapter 3: Designing a computer vision system
In this part, the deep learning model is designed according to the best strategy for the issue to solve.

Chapter 4: Prototype
This chapter discusses the hardware to be used in order to deploy the model and being able to run it in a real-world environment.

Chapter 5: Dataset for computer vision applications
This section outlines the various considerations and decisions involved in constructing a dataset for the project. An explanation of the conditions and tools utilized in the dataset creation process is also provided.

- Demonstration

Chapter 6: Analyzing model performance and Real world application
In this chapter, different metrics are employed to evaluate the quality of the designed model, its performance is also assessed through real-world testing.

- Evaluation

Chapter 7: Conclusion, improvements and application
In the final chapter, conclusions are derived based on the results of the metrics and it can be established whether the objectives have been achieved or not. Additionally, recommendations for improvements and potential future applications are proposed.

CHAPTER 2. COMPUTER VISION, CONCEPTS AND DEFINITIONS

Artificial intelligence is a very broad area of information technology (IT). In this chapter some terminology will be explained in order to understand what Computer Vision is and how it works. Different concepts around this field will be outlined so as to ensure a better understanding of the project as a whole. Once the basics are covered it will be possible to make reasoned decisions to design and build the prototype.

2.1 Computer vision precedents: Image processing

Computer Vision is the field of Artificial Intelligence that uses algorithms to derive information from images. This concept was devised in the 1960's when scientists started processing images with computers in order to recognize edges and shapes and in the 1970's they developed the first Optical Character Recognition (OCR) technology [4].

Edge detection consists of applying an algorithm to an image in order to extract its features. It is a matter of correctly defining a threshold to be able to distinguish the shapes. The left and middle images on Fig. 2.1 represent image processing filters (also called kernels) which apply an operation on the grayscale pixel values of an image, 0 represents black and 255 means white. In the most basic example, to compute the gradient, the kernel slides over the pixel values and computes for the horizontal filter: $v'_{x,y}= v_{x-1,y}+0\cdot v_{x,y}- v_{x+1,y}$ and $v'_{x,y}= v_{x,y-1}+0\cdot v_{x,y}- v_{x,y+1}$ for the vertical one. v' correspond to the new pixel value and x and y are the positions on the grid. This way the vertical filter will find horizontal patterns and the horizontal will find the vertical.

Vertical
1
0
-1

Horizontal

1	0	-1

Fig. 2.1: A Vertical filter and a horizontal filter respectively.

By grouping those filters, it is possible to create bigger kernels and apply them to images, this is the case of the Canny filters, see Fig. 2.2. Fig. 2.3 shows an example of edge detection applied on some coins.

1	1	1
0	0	0
-1	-1	-1

1	0	-1
1	0	-1
1	0	-1

Fig. 2.2: A Canny vertical filter and a Canny horizontal filter respectively.

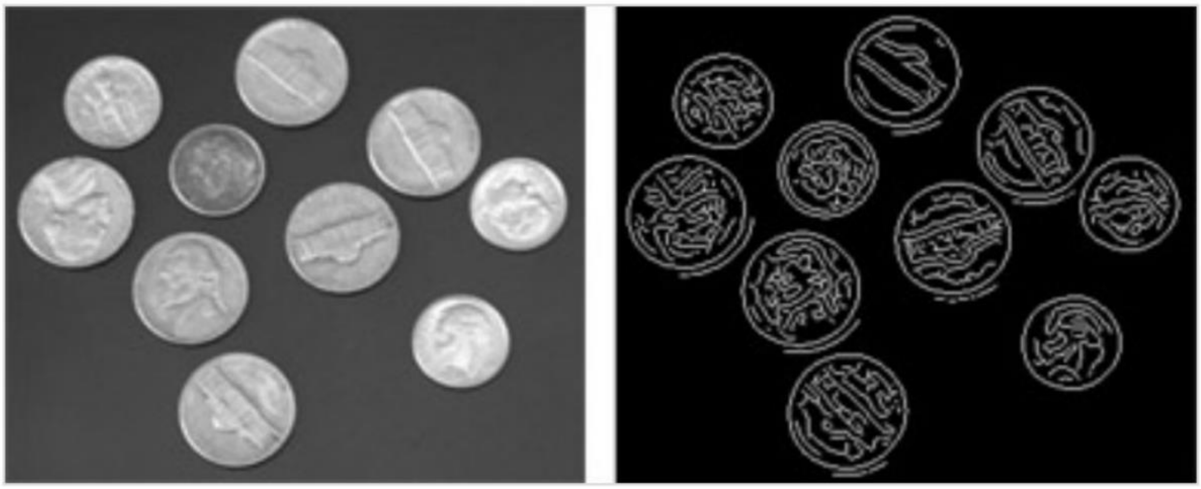
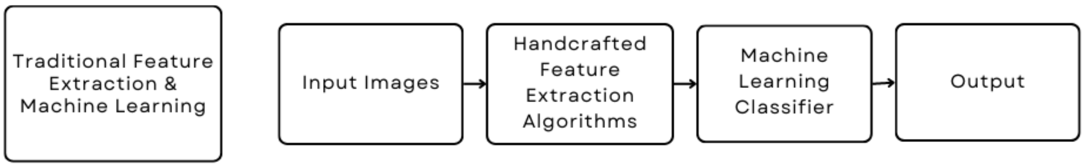


Fig. 2.3: A grayscale image of coins and the result after applying an edge detection mask (Canny) [5].

Since the 60's, in parallel to the development of OCR, scientists looked for ways in which computers would be able to perform certain tasks by providing them with an input of structured data, machine learning algorithms. Once all the instructions are provided, the algorithm should be able to take new data and classify, organize and sort it without any other intervention.

2.2 Computer vision in machine learning

As the technology evolved and the programming and image processing improved, machine learning solutions could be applied to computer vision problems. Computers are fed with an organized and structured input of images that contain the images to be evaluated. The algorithm runs the images through image processing filters especially selected to extract the useful features, finally a classifier algorithm produces an output.


Traditional Feature Extraction & Machine Learning
Input Images
Handcrafted Feature Extraction Algorithms
Machine Learning Classifier
Output
Fig. 2.4: Flowchart of the traditional feature extraction & machine learning algorithm.

Machine learning algorithms used in computer vision can be classified into three categories based on the availability of data, the nature of the task and the level of human intervention [5]:

- **Supervised learning:**
  Supervised learning is a machine learning strategy that uses labeled datasets to train algorithms that can classify data. A label is an associated value or category for each image. This type of machine learning algorithm is the most commonly used and studied.

- **Unsupervised learning:**
  Unsupervised learning is a type of machine learning that uses unlabeled data to discover patterns or structures in the information. The algorithm is self-thought to organize the input images in groups with little intervention (i.e number of groups). This type of machine learning algorithm is useful when image labels are missing and some kind of sorting and grouping is needed. It is not possible for unsupervised learning algorithms to assess the quality of the results through accuracy as a metric. However, similarity within the groups or dissimilarity between the groups can be evaluated.

- **Reinforced learning:**
  Reinforcement learning is a type of machine learning that uses a trial and error approach to learn from its own actions and feedback. The algorithm receives rewards or penalties based on the accuracy of its predictions. It learns to maximize its rewards by finding the optimal policy for the task. Reinforced learning is suitable for complex or dynamic problems where the data is large, flexible, and unpredictable. It usually requires a simulation environment to simulate a task in real life so that multiple situations can be simulated in a short period of time.

2.2.1 Deep learning fundamentals

A particular case of machine learning is deep learning (DL), in contrast to the human custom-made feature extraction techniques present in classical machine learning solutions, DL algorithms are automatic end-to-end iterative processes that autonomously learn to distinguish which part of the input data is important and which are the features that maximize the possibility to obtain the expected output.

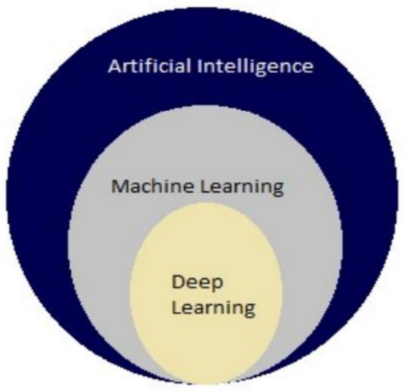


Fig. 2.5: Visual representation of artificial intelligence and its fields [6].

The most common algorithm in Deep Learning for Machine Learning purposes is the Artificial Neural Network (ANN). ANN algorithms are based on an analogy to the human brain and its complex system of neuron interconnections. The most basic structure in a machine learning model is an artificial neuron called a perceptron. Perceptrons are organized in layers to create ANNs. Each perceptron receives information from the previous layer, applies an activation function, and sends the output to the next layer. This information is associated with a weight relative to its relative importance and consists of the sum of all weights from inputs plus a bias and are updated with each iteration according to previous performance [7].

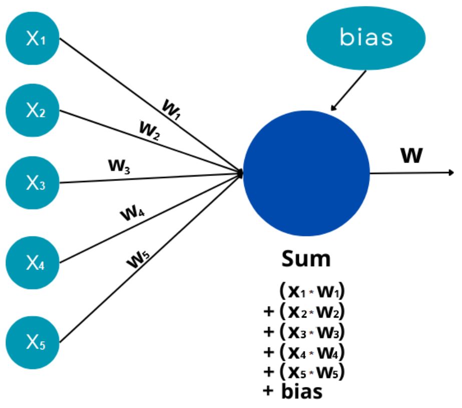


Fig. 2.6: Diagram showing of a perceptron

ANN can be divided into three groups of layers: the Input Layer, the Hidden Layers and the Output Layer. The features extracted by the filters are given to the Input Layer. The Input Layer neurons, after computing the weight pass on the information if a threshold is exceeded if they return a large positive value; otherwise, they return zero or a smaller value and therefore they may choose not to pass on information to the next layer [8]. The same principle applies to the Hidden Layer, the more layers in the Hidden Layer there are, the “deeper” the DL model is, finally the results are given to the neuron(s) in the Output Layer, there will be as many neurons as defined classes the model has.

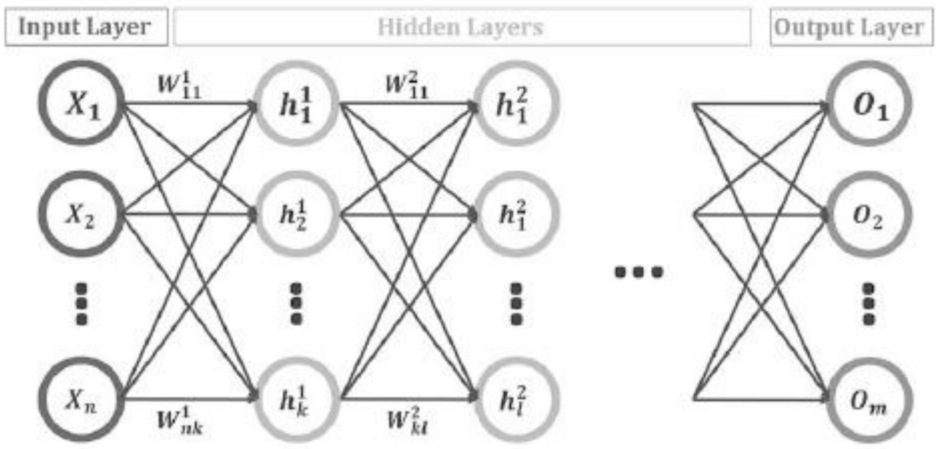


Fig. 2.7: Diagram showing the structure and the different layers in a Deep Learning algorithm [8].

The purpose of a ML algorithm is to minimize a cost function and binary cross entropy is the most commonly used cost function in binary classifications. Due to being very complex functions depending on multiple parameters, optimization algorithms are used. The two most commonly used optimization algorithms are Stochastic gradient descent (SGD) and Adaptive Moment Estimation (Adam). SGD is an iterative method used for optimizing the gradient descent during each search once the weights have been initialized, those are the weights that are adjusted (or learned) during the iterations performed during the training. Adaptive moment estimation or “Adam” is the most popular optimization algorithm as it has been proven more efficient than SGD as it tunes the moment hyperparameter that allows exiting a local minimum defined by the cost function. Fig. 2.8 shows a diagram of the full architecture of a Convolutional Neural Network (CNN).

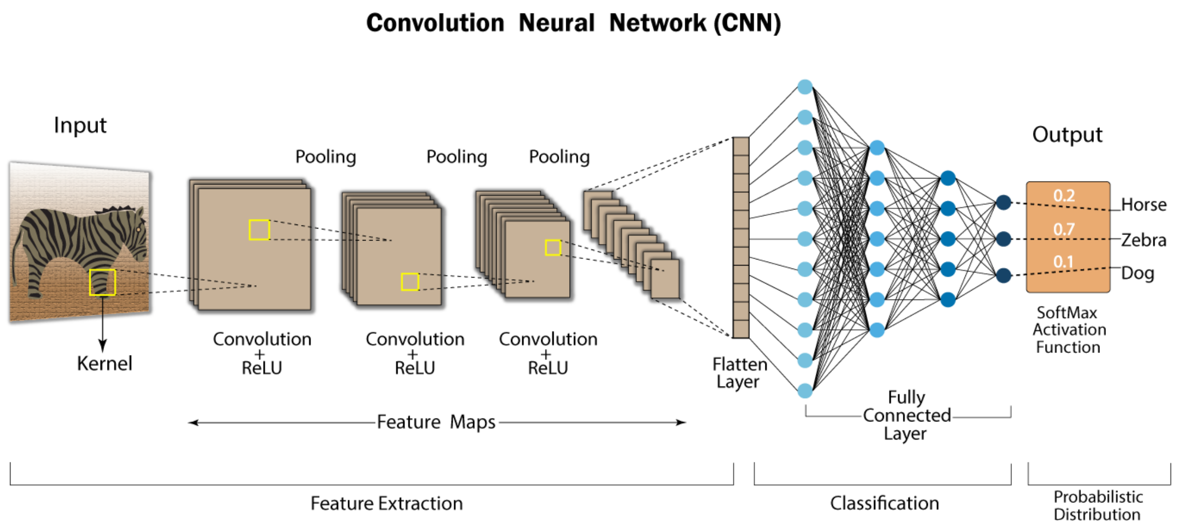


Fig. 2.8: Structure of a CNN in a Horse, Zebra, Dog computer vision classification program [9].

Generally, in DL applied to computer vision, convolutional filters are used for feature extraction which will be fed into the ANN. These convolutional filters are composed of various kernels (filters) that are used to perform convolutional operations by sliding through the whole image and generating various feature maps in the process. Through the usage of different convolutional filters it is possible to recognize patterns in the input images such as lines, gradients, circles, and more complex shapes. Convolutional filters are often paired with pooling layers. Pooling is a technique used to speed up and remove redundancy present in the input features extracted by convolutional filters as it helps the network to recognize features independent of their location in the image by taking the average or maximum value of all feature maps in an image [10]. This represents a difference with respect to the traditional hand-crafted architectures of filters in traditional machine learning algorithms. The extracted features are then “flattened” into an array to be fed to the ANN.

In the cases where results are poor or there is little data to train the data, in order to improve transfer learning may be used. Transfer learning is a design methodology used in deep learning where a pre-trained model is reused on a new task. It involves exploiting the knowledge gained from a previous task to improve generalization about another. For example, a classifier trained to predict whether an image contains cars could use the knowledge it gained during training to recognize people. Instead of starting the learning process from scratch, transfer learning starts with patterns learned from solving a similar task.

Fine-tuning is the technique generally combined with transfer learning where the pre-trained model is repurposed for a new task. With transfer learning, the initial layers of the model, which learn general features such as edges, shapes, and textures, are frozen while the final layers are retrained with fewer iterations of the training data and a lower learning rate. This allows the model to adapt to the new task by only changing the weights of the last few layers that came from the original model [11]. When fine-tuning, new images with variations are added to the original dataset and the model is trained for a few iterations.

To sum up, what defines deep learning is the quantity of layers the network has and the way the nodes that configure it interact with each other and the end-to-end automatization of the processes.

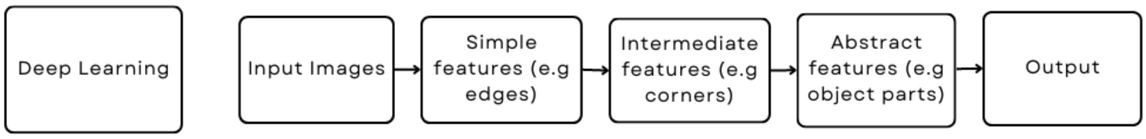

Deep Learning
Input Images
Simple features (e.g edges)
Intermediate features (e.g corners)
Abstract features (e.g object parts)
Output

Fig. 2.8: Flowchart of the deep learning algorithm with CNN.

2.2.2. Classical machine learning or deep learning?

Depending on the type of application that requires a computer vision solution a simple machine learning architecture will suffice in other cases, it will require a deep learning approach. Machine learning is good when problems are simple, there are few useful features that describe the class but it struggles when the problem is more complex and the background is very “noisy”. While deep learning uses convolutional filters to automate feature obtention, which requires lots of input images to obtain reliable results, on the other hand in machine learning, a relatively low quantity of data is enough to obtain good performance provided the tasks are simple [12]. As a result, deep learning models are often considered less interpretable (black box) while machine learning models are theoretically easier to understand and interpret.

Another factor to consider is that machine learning requires more ongoing human intervention to get results while deep learning is more complex to set up but requires minimal intervention thereafter. As per hardware requirements, machine learning programs tend to be less complex than deep learning algorithms and can often run on conventional computers, but deep learning systems require more powerful hardware and resources. In deep learning applications that require computer vision, because of the quantity of inputs, lots of images need to be processed at the same time, this is resolved by the usage of graphical processing units (GPUs) which represent an extra expense in resources with respect to the classical machine learning approach [13].

2.3 Computer vision strategies

Deep learning algorithm applied to computer vision mainly follow these strategies:

- **Image classification:** Image classification is a task in computer vision where a model is trained to classify images into predefined categories. The model determines whether an image contains an object belonging to any of the considered classes and, if so, assigns the image to the appropriate class. For example, a model may be trained to classify images of dogs and cats or to distinguish between different breeds of dogs [4].

- **Object detection:** Object detection also classifies the given images but additionally, after recognizing a certain object in an image belongs to a certain defined class, it is able to accurately locate it within the image by a rectangular box (bounding box). An example could be tracking the movements of people in a reserved area or counting objects [4].

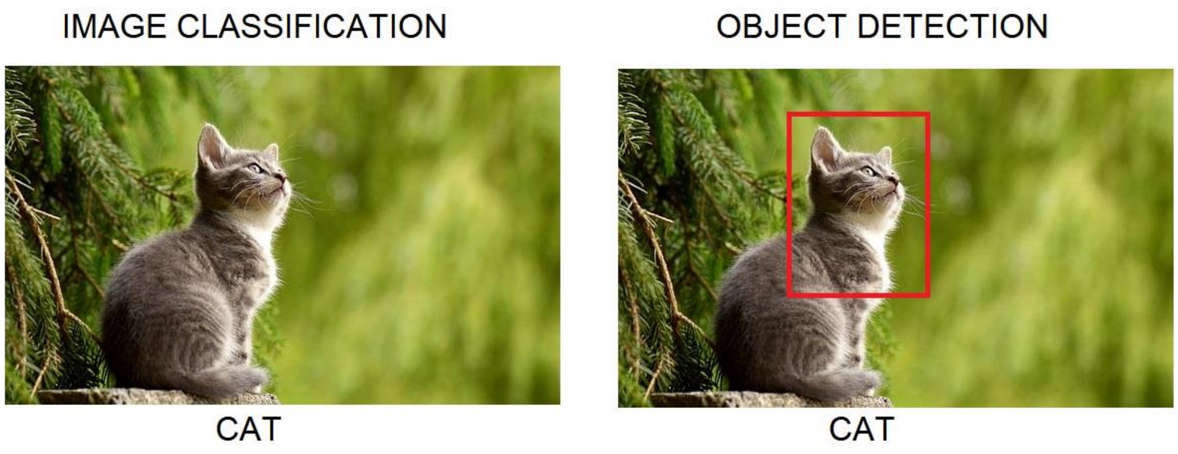


Fig. 2.9: Difference between image classification and object detection.

Image classification represents an advantage for applications where the position of the object is not relevant and what really matters is if the system is correctly inferring the classes. Classification eases the requirements in hardware and maximizes detections per frame as it requires less computations. According to the research [14] for tests conducted on a RaspberryPi 4 using a quantized INT8 tflite model, the most popular pre-trained models based on these two strategies have the following properties:

Strategy	Model	Parameters	Size	Execution Time (ms)
Classification	MobileNetV1	3.23M	13.2MB	83
Classification	MobileNetV2	2.26M	9.5MB	83
Classification	InceptionV3	21.81M	88.1MB	150
Detection	SSD-MobileNetV1	5.51M	22.7MB	230
Detection	SSD-MobileNetV2	3.87M	16.4MB	225
Detection	SSD-InceptionV2	13.3M	54.0MB	700

Table 2.1 With details of the three main classification and three detection pre-trained models

CHAPTER 3. DESIGNING A COMPUTER VISION SYSTEM

Having discussed the characteristics of the different available machine learning algorithms it is time to evaluate which characteristics a computer vision for bird strike prevention system has and which machine learning algorithm is the most convenient. It has to be noted that such a system has to be extraordinarily adaptative because of the variability of the conditions. Once set up, the processor will receive real time images from a camera pointed at the sky and will infer if there are any birds in sight.

Due to the wide range of bird species flying at various distances from the camera objective and the changing background (e.g time of the day, meteorological conditions…) there is a high casuistry. In order to solve this, the model will require a large dataset of images depicting as many situations as possible. All things considered, the deep learning approach would provide the best results and therefore it would be the appropriate type of machine learning algorithm for this problem.

3.1 Deep Learning planification

Deep learning models based on convolutional neural networks depend on multiple parameters. It is important to understand the impact of every metric involved in the network training and configuration in order for the model to perform well.

As explained in Chapter 2. Computer Vision, concepts and definitions, the CNN are based on layers of nodes (filters) that pass on the information to the next layer and each connection has a weight that is updated with each iteration. The images are fed into the network in batches of various images, generally 32 or 64. When the batch propagates forward in the network it computes the weights. Then the ANN receives an array of features obtained from the convolutional filters and adapts the weights on the network. When it reaches the output it tests its performance by checking the provided labels and calculating the loss, afterwards it performs a backward propagation adjusting the weights. The process will be repeated the number of epochs [15]. Both the CNN and ANN are modified with gradient descend. For a given number of images “N”, the number of iterations would be:

$$\#iterations = \frac{N}{Batch\ size} = 1\ epoch$$

The network will process all the N images for each epoch. If the batches are smaller, the network may make more errors in updating its weights and it will take longer to learn. If the batches are larger, the network will learn faster but it will not be able to handle different kinds of images well, it could overfit.

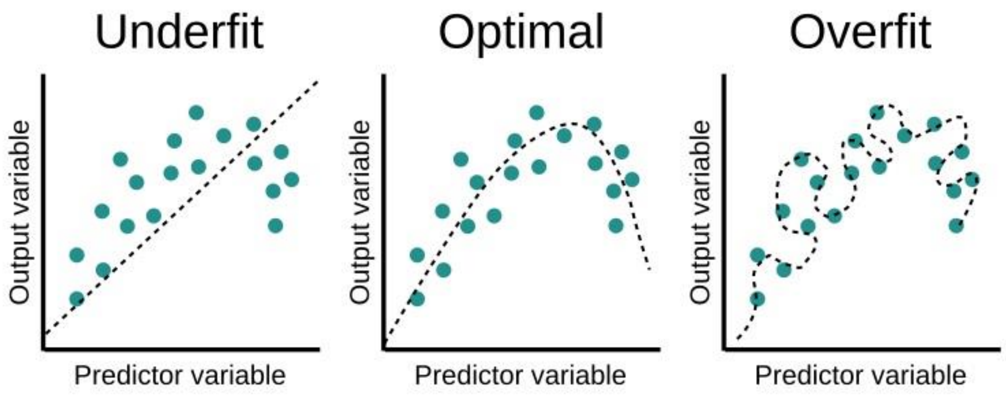


Fig. 3.1: Representation of the scenarios in deep learning models [16].

When the model is underfit, it cannot learn the patterns that explain the images and it makes wrong predictions. When the model is overfit, it only works well on images that are very similar to the ones it learned from. The best scenario is the optimal one, as shown in Fig. 3.1, where the model has learned the patterns and would be able to make good predictions even on images that are different from the ones it learned from. Dropout is a regularization technique that randomly sets a fraction of neurons to zero during training to reduce overfitting and enhance generalization.

What any deep learning model intends to do is to find the values that minimize its loss function. In deep learning, this function is nonlinear and depends on various features. The learning rate is the parameter that controls the magnitude of the gradient descent. In mathematical terms, it computes the partial derivative of the loss function with respect to the model parameters and updates them in the opposite direction of the gradient, until it reaches a global minimum. The challenge is that if the learning rate is too large, it might overshoot the global minimum, but if it is too small, it might converge to a local minimum [17].

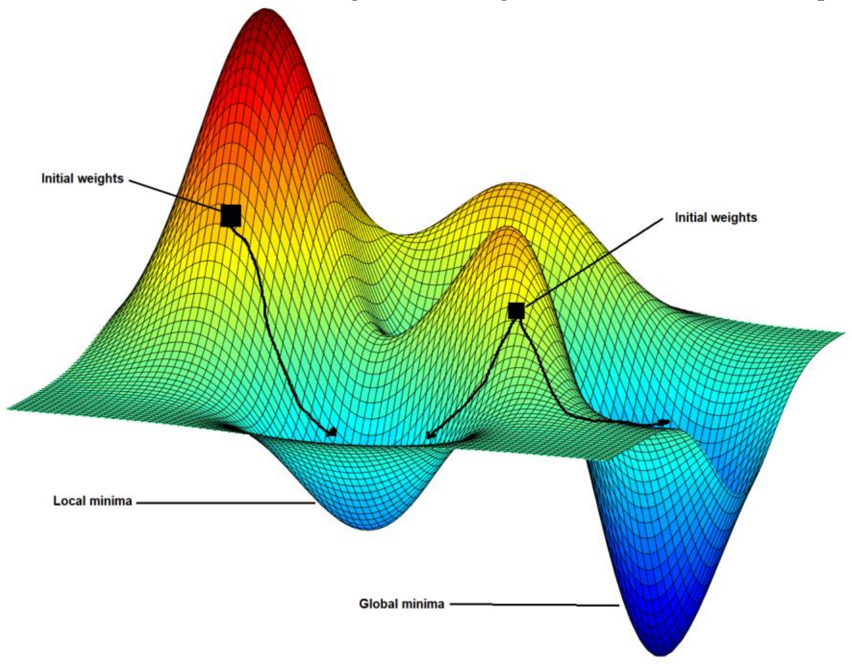


Fig. 3.2: Three-Dimensional equation representing gradient descent and its situations. Adapted image from Wikipedia [18]

Stochastic Gradient Descent (SGD) is a method to update the weights of a model after each training sample. It uses a small sample to estimate the gradient, which is the direction of steepest descent. This means that SGD may not always follow the smoothest path to the global minimum, but it will eventually reach it. Adam optimizer is another method that adapts the learning rate for each weight based on its past gradients and moments. Adam optimizer has some advantages over SGD, such as being faster, more memory-efficient, more suitable for large datasets and sparse problems, and easier to tune.

Pre-trained networks in deep learning are models that have been trained on a large dataset of images or other data before being used for a different task. They can help improve the performance and speed of the model on the new task by using some or all of the learned features from the pre-trained model. These types of networks are useful for transfer learning, feature extraction, and classification, as for the case of the present project.

3.2 PyTorch or TensorFlow?

Being a high-level programming language that supports modules and packages, Python has become the quintessential programming language for deep learning. As a high-level language the process of developing code is simple and more understandable and the use of packages encourages program modularity and code reuse. Among the various packages and libraries maintained by the community that are widely used to design deep learning algorithms and convolutional neural networks there are two that stand out: PyThorch and TensorFlow.

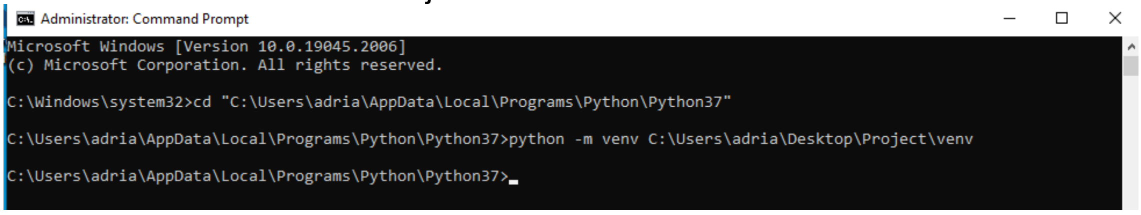
TensorFlow, as a deep learning framework was released in November 2015 by Google, by then it had already been tested by the company in Google's products such as Google Photos, Google Search, spam detection, speech recognition or Google Assistant among others [19]. Similarly, PyTorch developed by Facebook's artificial intelligence division was released in September 2017 to enable the processing of large-scale image analysis, including object detection, segmentation and classification.

Both the TensorFlow and PyTorch libraries are based on graph (node) based architectures and use tensors for computation. A tensor being a mathematical object and a generalization of scalars, vectors, and matrices, in sum it can be described as an n-dimensional array. Even though both libraries can be used interchangeably, in terms of data visualization and model deployment TensorFlow seems to be better. The early release of TensorFlow has made it possible for it to reach a stable state before its competitor PyTorch. As TensorFlow has been in the market for a longer period its adoption by the community has resulted in greater usage and support [19]. There are many more examples of TensorFlow based codes than in PyTorch, for this reason the computer vision for bird strike prevention system will be based in TensorFlow.

3.3 Setting up a virtual environment

The first step to develop a deep learning model in Python language is to create a virtual environment in which all the dependencies and libraries will be contained so that there are no interferences and cross-references with other libraries. As for this project, the operative system (O.S) used is Windows 11 and the Python version used is Python 3.7.8.

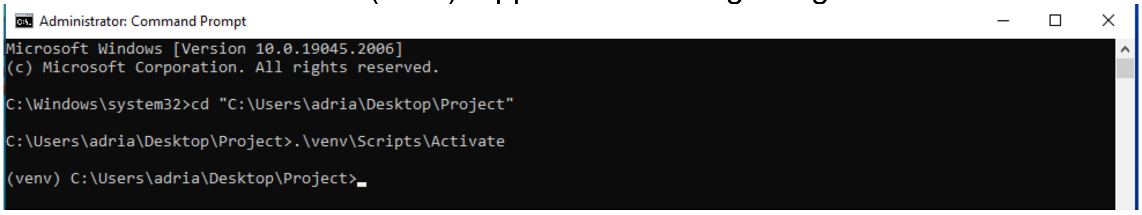
Using the command line from the path where python.exe is installed, a virtual environment can be created. In Fig. 3.3 a virtual environment called “venv” is created in a folder named Project.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19045.2006]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd "C:\Users\adria\AppData\Local\Programs\Python\Python37"

C:\Users\adria\AppData\Local\Programs\Python\Python37>python -m venv C:\Users\adria\Desktop\Project\venv

C:\Users\adria\AppData\Local\Programs\Python\Python37>
```

Fig. 3.3: Command instruction to create a virtual environment.

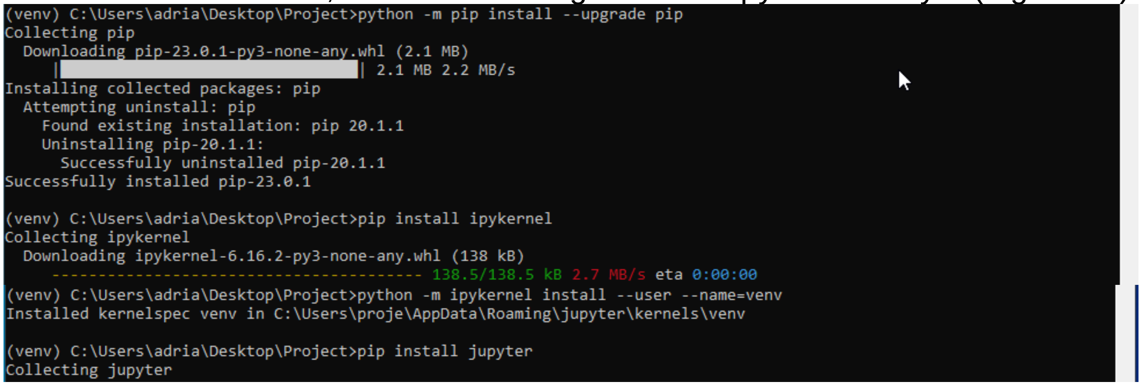
The way to access the environment is accessing the Activate script created in the “venv” environment in the folder Project, see in Fig. 3.4 that when the environment is activated “(venv)” appears at the beginning of the command line:



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19045.2006]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd "C:\Users\adria\Desktop\Project"

C:\Users\adria\Desktop\Project>.\venv\Scripts\Activate

(venv) C:\Users\adria\Desktop\Project>
```

Fig. 3.4: Accessing the virtual environment through the command line.

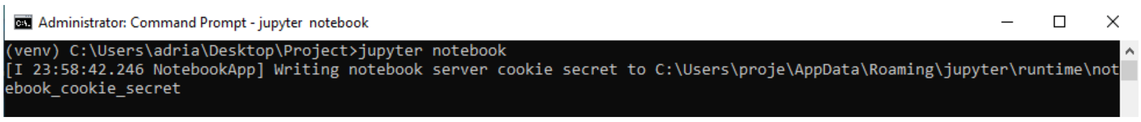
To finish setting up the environment, four things need to be done: upgrading the pip library that installs packages, installing ipykernel (Fig. 3.5), adding the name of the environment, and installing the Jupyter library (Fig. 3.6).



```
(venv) C:\Users\adria\Desktop\Project>python -m pip install --upgrade pip
Collecting pip
  Downloading pip-23.0.1-py3-none-any.whl (2.1 MB)
     |████████████████████████████████| 2.1 MB 2.2 MB/s
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 20.1.1
    Uninstalling pip-20.1.1:
      Successfully uninstalled pip-20.1.1
Successfully installed pip-23.0.1

(venv) C:\Users\adria\Desktop\Project>pip install ipykernel
Collecting ipykernel
  Downloading ipykernel-6.16.2-py3-none-any.whl (138 kB)
     ---------------------------------------- 138.5/138.5 kB 2.7 MB/s eta 0:00:00
(venv) C:\Users\adria\Desktop\Project>python -m ipykernel install --user --name=venv
Installed kernelspec venv in C:\Users\proje\AppData\Roaming\jupyter\kernels\venv

(venv) C:\Users\adria\Desktop\Project>pip install jupyter
Collecting jupyter
```

Figures 3.5 & 3.6: Accessing the virtual environment through the command line.

Jupyter is an open-source software that allows programming in multiple languages and it is structured in cells which can be run independently. Using the command instruction “Jupyter notebook” the Jupyter manager is opened in the default web browser.



```
(venv) C:\Users\adria\Desktop\Project>jupyter notebook
[I 23:58:42.246 NotebookApp] Writing notebook server cookie secret to C:\Users\proje\AppData\Roaming\jupyter\runtime\not
ebook_cookie_secret
```

Fig. 3.7: Opening Jupyter notebook from the command line.

The Jupyter manager should look like the one in Fig. 3.8. A new ipynb file can be created selecting a new notebook in “venv”.

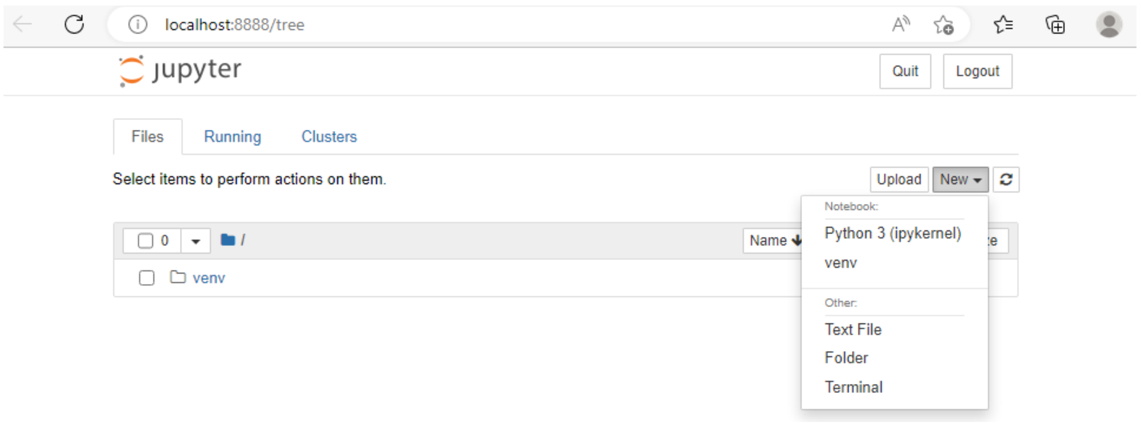
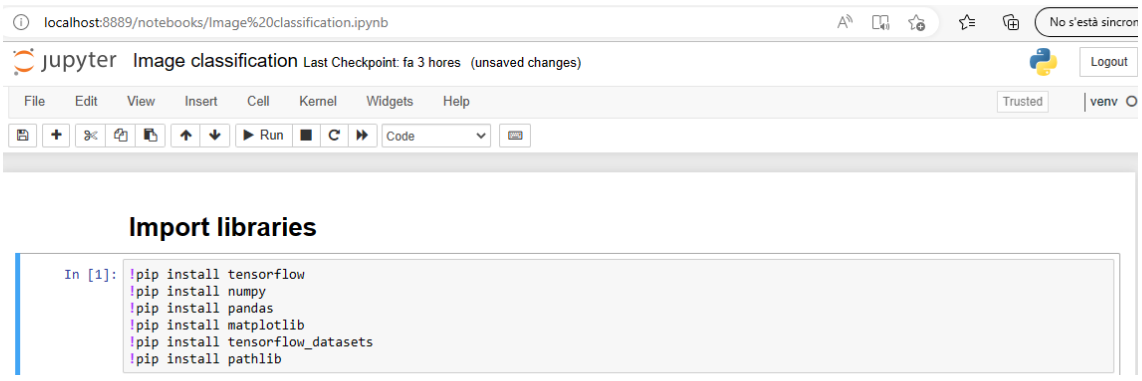


Fig. 3.8: Opening Jupyter notebook from the command line.

In order to create the deep learning model some libraries have to be imported, which can be done either by using the command line inside the environment or by using the notebook cell as shown in Fig. 3.9.


localhost:8889/notebooks/Image%20classification.ipynb
No s'està sincron
jupyter Image classification Last Checkpoint: fa 3 hores (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
venv
Run
Code
Import libraries

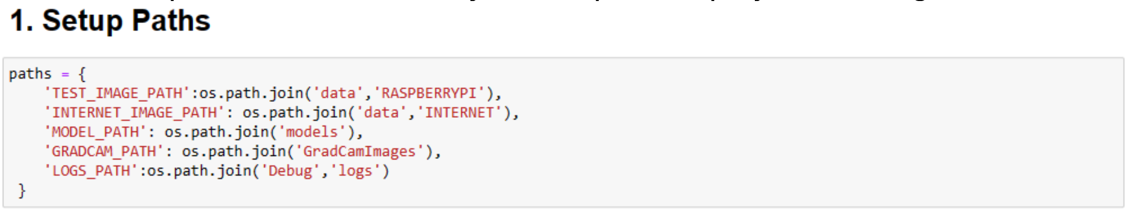
```
In [1]: !pip install tensorflow
        !pip install numpy
        !pip install pandas
        !pip install matplotlib
        !pip install tensorflow_datasets
        !pip install pathlib
```

Fig. 3.9: Instructions to install tensorflow, numpy, pandas, matplotlib, pathlib and tensorflow_datasets.

3.4 Managing datasets

Apart from tensorflow, numpy is a library that is useful when handling n-dimensional arrays, pandas is a data analysis and manipulation tool, matplotlib is a package that graphs information, tensorflow_datasets has functionalities suited for dataset handling and pathlib is convenient when managing paths.

Paths where the datasets are stored and where the outputs from the code are saved are specified in a dictionary for the present project, see Fig. 3.10.

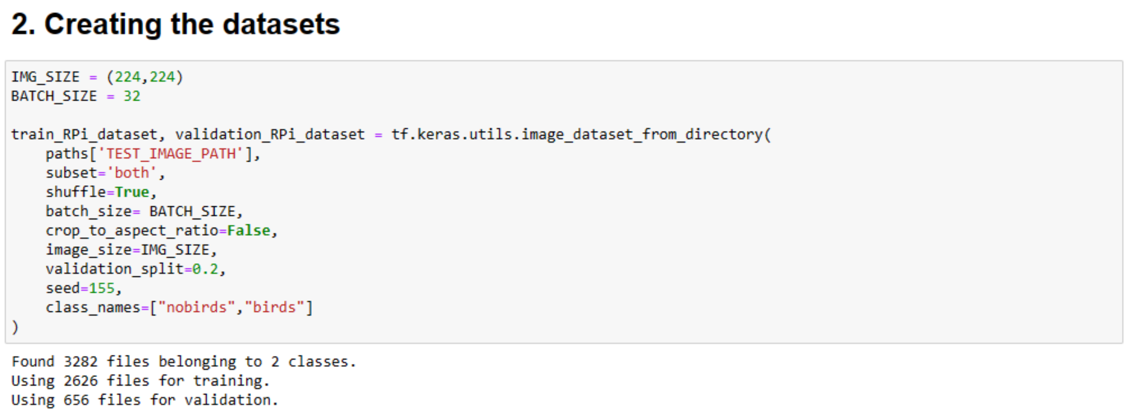


1. Setup Paths

```
paths = {
    'TEST_IMAGE_PATH':os.path.join('data','RASPBERRYPI'),
    'INTERNET_IMAGE_PATH': os.path.join('data','INTERNET'),
    'MODEL_PATH': os.path.join('models'),
    'GRADCAM_PATH': os.path.join('GradCamImages'),
    'LOGS_PATH':os.path.join('Debug','logs')
 }
```

Fig. 3.10: Dictionary showing the structure of the folders used in the projects.

The project contains four folders: data, models, GradCamImages and Debug. In the data folder, there are two folders, one containing images obtained through a RaspberryPi and the other with images downloaded from the internet. The models folder will store files containing the models, GradCamImages will contain the images resulting from a GRADCAM analysis and the Debug folder will contain logs from the code. Once defined the paths, it is possible to extract the input images from the folders, datasets are created as shown in Fig. 3.11.



2. Creating the datasets

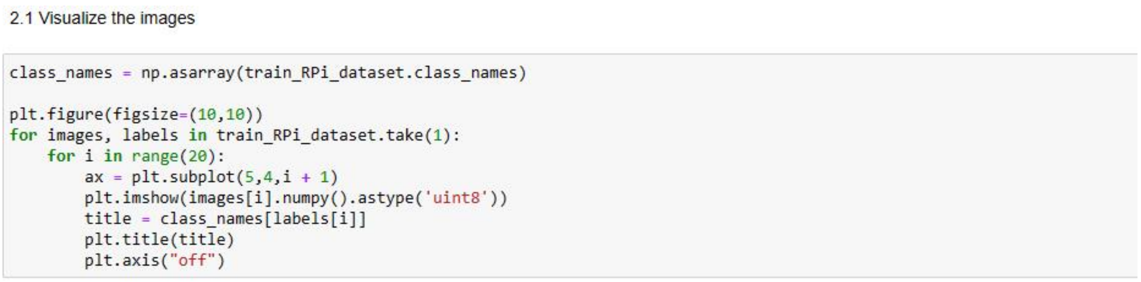
```
IMG_SIZE = (224,224)
BATCH_SIZE = 32

train_RPi_dataset, validation_RPi_dataset = tf.keras.utils.image_dataset_from_directory(
    paths['TEST_IMAGE_PATH'],
    subset='both',
    shuffle=True,
    batch_size= BATCH_SIZE,
    crop_to_aspect_ratio=False,
    image_size=IMG_SIZE,
    validation_split=0.2,
    seed=155,
    class_names=["nobirds","birds"]
)
```

```
Found 3282 files belonging to 2 classes.
Using 2626 files for training.
Using 656 files for validation.
```

Fig. 3.11: Declaration of the train and validation datasets.

Fig. 3.11 shows the creation of training and validation datasets from the RaspberryPi images. The validation_split parameter specifies a 20:80 split ratio for the validation and training datasets respectively. The batch size is 32 and no cropping is applied to the images. The image size is fixed at 224x224 pixels. Class names are derived from the input folders and each image is labeled with its corresponding category: bird or non-bird. The total number of samples is 3282, distributed across two classes, with 2626 for training and 656 for validation.

A small sample of the images in the dataset is shown through the instruction in Fig. 3.12.


2.1 Visualize the images

```
class_names = np.asarray(train_RPi_dataset.class_names)

plt.figure(figsize=(10,10))
for images, labels in train_RPi_dataset.take(1):
    for i in range(20):
        ax = plt.subplot(5,4,i + 1)
        plt.imshow(images[i].numpy().astype('uint8'))
        title = class_names[labels[i]]
        plt.title(title)
        plt.axis("off")
```

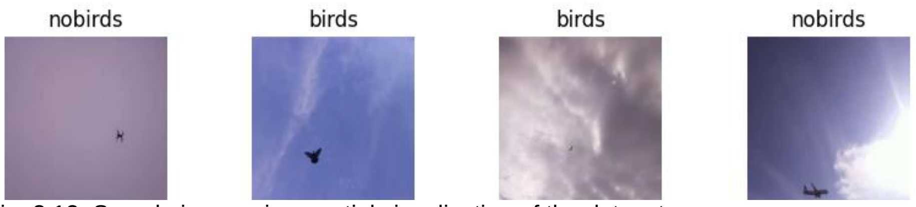
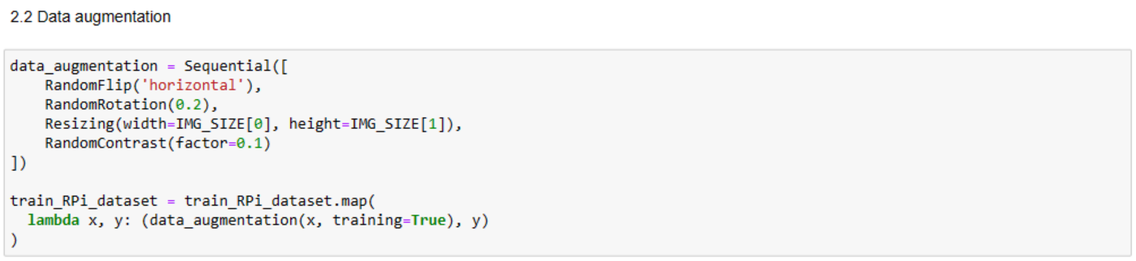


Fig. 3.12: Sample images in a partial visualization of the dataset.

The accuracy of a deep learning model depends on the amount and diversity of training data. As the available information is limited, we may use data augmentation to improve the results. Data augmentation can be used to transform (zoom, rotate etc) and artificially enhance and diversify existing training data. Nevertheless, data augmentation does not create new data, it generates variations of it. Fig. 3.13 shows the code used for the present project.


2.2 Data augmentation

```
data_augmentation = Sequential([
    RandomFlip('horizontal'),
    RandomRotation(0.2),
    Resizing(width=IMG_SIZE[0], height=IMG_SIZE[1]),
    RandomContrast(factor=0.1)
])

train_RPi_dataset = train_RPi_dataset.map(
  lambda x, y: (data_augmentation(x, training=True), y)
)
```

Fig. 3.13: Cell showing the instruction for data augmentation.

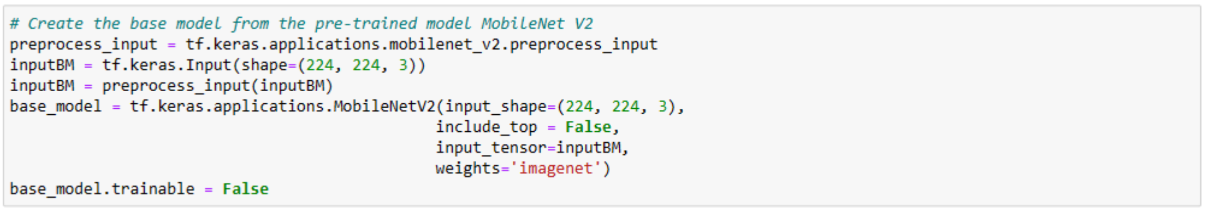
In this case, the images will be randomly flipped, rotated up to a 20% and its contrast will be randomly adjusted by a random factor.

3.5 Handling pre-trained models

The convolutional neural network (CNN) for this project is based on MobileNetV2 which is an architecture designed by Google for efficient object classification. This CNN is chosen for its fast execution time and its low memory consumption as seen in “Chapter 2. computer vision, concepts and definitions”, the weights in the pre-trained model will be modified by transfer learning and fine-tuning techniques to adapt it to the specific datasets for this project. Transfer learning involves using the pre-trained weights of MobileNetV2 as the initial values for the neurons in the CNN, while fine-tuning involves adjusting these weights during training to optimize its performance.

3.0 Deep Learning Model

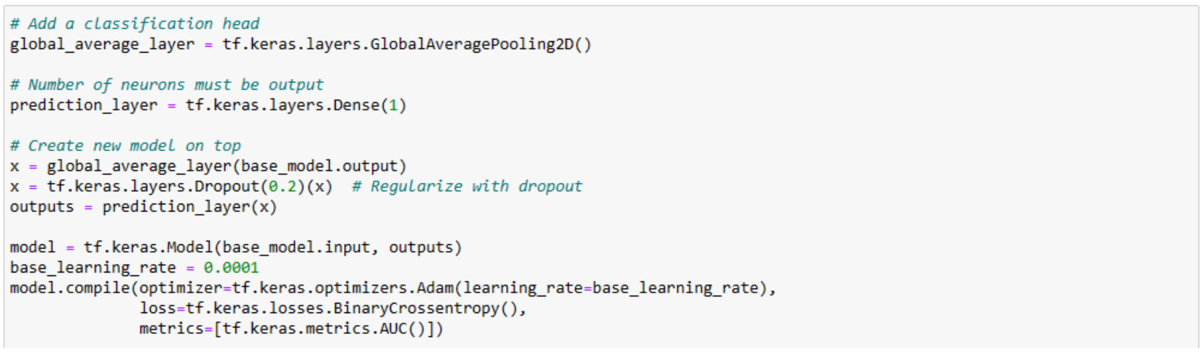
3.1 Build Deep Learning Model



```
# Create the base model from the pre-trained model MobileNet V2
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
inputBM = tf.keras.Input(shape=(224, 224, 3))
inputBM = preprocess_input(inputBM)
base_model = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3),
                                               include_top = False,
                                               input_tensor=inputBM,
                                               weights='imagenet')
base_model.trainable = False
```

Fig. 3.14: Setting up the pre-trained model based on MobileNetv2.

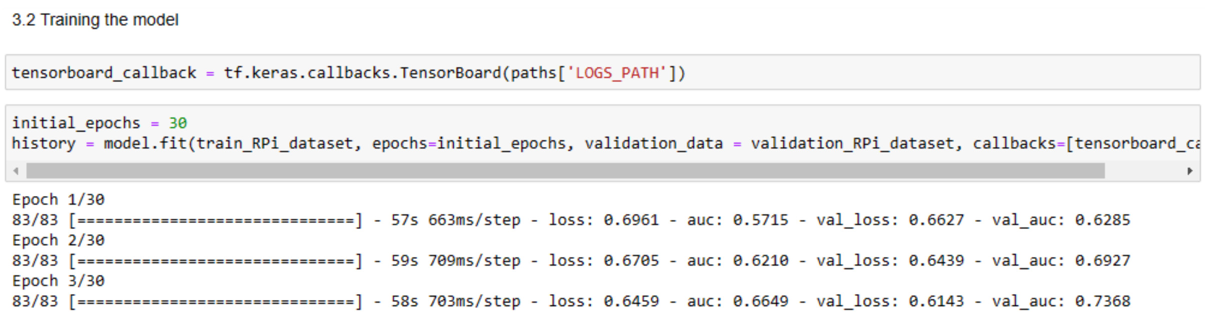
In Fig. 3.11, the input images are preprocessed to have a height and width of 124 pixels and three channels corresponding to the RGB channels. These input characteristics are fed into the MobileNet V2 model, which is pre-loaded with weights trained on ImageNet. The top layers of the model will be defined later. This constitutes the base model, which is then frozen by setting the trainable attribute to false, preventing the weights in the layers from being updated during training.



```
# Add a classification head
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()

# Number of neurons must be output
prediction_layer = tf.keras.layers.Dense(1)

# Create new model on top
x = global_average_layer(base_model.output)
x = tf.keras.layers.Dropout(0.2)(x)  # Regularize with dropout
outputs = prediction_layer(x)

model = tf.keras.Model(base_model.input, outputs)
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.AUC()])
```

Fig. 3.15: Compiling the model.

Once the pre-trained model is fixed, two new layers are added on top of it: a global average layer and a prediction layer. The global average layer reduces the exit of the pre-trained backbone from a 7x7x1280 block of features to a one dimensional array of 1x1x1280 to be fed to the ANN. Then, a dropout layer randomly removes 20% of the neurons to prevent overfitting. The prediction layer maps the remaining neurons to a single output value for each image. The learning rate is set to 0.0001 and Adam optimizer is used. The output value indicates a probability between 0 and 1 and after applying a threshold (which in a binary classification model trained with balanced datasets is generally 0.5) whether the image contains a bird (class 1) or not (class 0). No activation function is used for the output value because the model uses binary cross-entropy loss. The variable of control is the Area Under the Curve (AUC).



3.2 Training the model

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(paths['LOGS_PATH'])
```

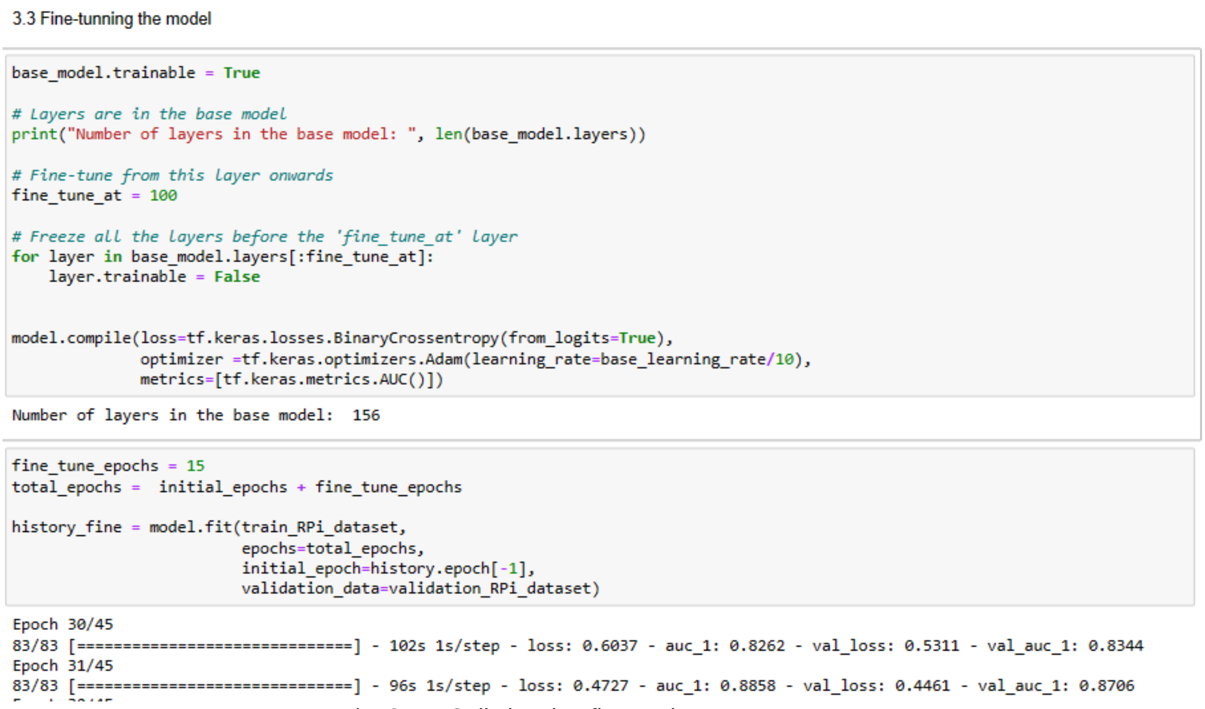
```
initial_epochs = 30
history = model.fit(train_RPi_dataset, epochs=initial_epochs, validation_data = validation_RPi_dataset, callbacks=[tensorboard_ca
```

```
Epoch 1/30
83/83 [==============================] - 57s 663ms/step - loss: 0.6961 - auc: 0.5715 - val_loss: 0.6627 - val_auc: 0.6285
Epoch 2/30
83/83 [==============================] - 59s 709ms/step - loss: 0.6705 - auc: 0.6210 - val_loss: 0.6439 - val_auc: 0.6927
Epoch 3/30
83/83 [==============================] - 58s 703ms/step - loss: 0.6459 - auc: 0.6649 - val_loss: 0.6143 - val_auc: 0.7368
```

Fig. 3.16: Declaring callback and fitting the model.

This part of the training only acts on the top layers of the MobileNetV2 base model. The weights of the pre-trained model are frozen and not updated during the training. Fig. 3.16 shows the training process for 30 epochs and the model is given validation_RPi_dataset for validation purposes. The figure displays the time it takes to complete the epoch, the step time, the loss and AUC metrics for both the training and validation datasets for the first three epochs.
In order to increase the performance the next step is to "fine-tune" the weights of the top layers of the pre-trained model alongside the training of the pre-trained classifier model added. The training process will adjust the weights from generic feature maps to the features specific to the dataset without erasing the generic learning [20].

This is achieved by un-freezing the top layers of the base_model and setting the bottom layers to be un-trainable. A larger part of the model is being trained now so as to readapt the pretrained weights. As a result, it is important to use a lower learning rate at this stage. Otherwise, the model could overfit very quickly. By doing so the accuracy should improve by a few percentage points.



3.3 Fine-tunning the model

```
base_model.trainable = True

# Layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the 'fine_tune_at' layer
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False


model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer =tf.keras.optimizers.Adam(learning_rate=base_learning_rate/10),
              metrics=[tf.keras.metrics.AUC()])
```

```
Number of layers in the base model:  156
```

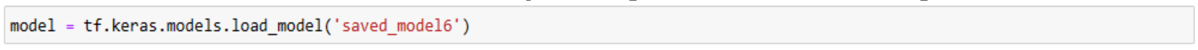
```
fine_tune_epochs = 15
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_RPi_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_RPi_dataset)
```

```
Epoch 30/45
83/83 [==============================] - 102s 1s/step - loss: 0.6037 - auc_1: 0.8262 - val_loss: 0.5311 - val_auc_1: 0.8344
Epoch 31/45
83/83 [==============================] - 96s 1s/step - loss: 0.4727 - auc_1: 0.8858 - val_loss: 0.4461 - val_auc_1: 0.8706
```

Fig. 3.17: Cell showing fine-tuning process.

Fig. 3.17 shows that the layers corresponding to the base_model are un-frozen and the fine tuning will be performed starting at the hundredth's layer out of 156 total layers. BinaryCrossentropy will be used and the learning rate will be divided by ten, the control metric will be AUC. The first two lines of the execution show the same information, it starts at the 30th epoch where the training stopped. In this case, since more layers are involved and the learning rate is lower, the epoch and step time have increased.

The model can be saved in a folder by writing the command in Fig. 3.18.

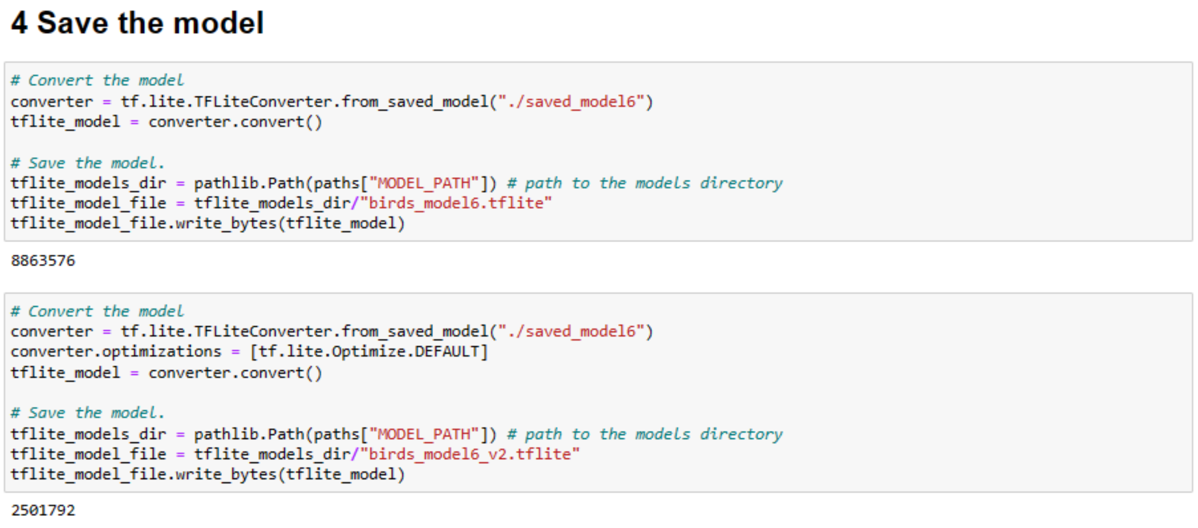


```
model = tf.keras.models.load_model('saved_model6')
```

Fig. 3.18: Saving the model.

After saving the model, it can be converted to a format that can be used in applications independently. This can be done by typing the commands in Fig. 3.18.

To reduce the memory usage of the model, a quantization and optimization process can be applied. This process maps continuous values to discrete values and uses lower precision numbers for computation instead of floating point values.



4 Save the model

```
# Convert the model
converter = tf.lite.TFLiteConverter.from_saved_model("./saved_model6")
tflite_model = converter.convert()

# Save the model.
tflite_models_dir = pathlib.Path(paths["MODEL_PATH"]) # path to the models directory
tflite_model_file = tflite_models_dir/"birds_model6.tflite"
tflite_model_file.write_bytes(tflite_model)
```

```
8863576
```

```
# Convert the model
converter = tf.lite.TFLiteConverter.from_saved_model("./saved_model6")
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

# Save the model.
tflite_models_dir = pathlib.Path(paths["MODEL_PATH"]) # path to the models directory
tflite_model_file = tflite_models_dir/"birds_model6_v2.tflite"
tflite_model_file.write_bytes(tflite_model)
```

```
2501792
```

Fig. 3.19: Saving the model in tflite extension.

Fig. 3.19 shows that the tflite model without optimizations and quantization takes up 8863576 bytes, while with them it only takes up 2501792 bytes.

CHAPTER 4. PROTOTYPE

After training and saving a Computer Vision model as a tflite file, the next step is to choose a processor to run it. A processor is an electronic device that executes instructions and manipulates data for a computer. A microprocessor is a specific type of processor that integrates the functions of a computer's CPU on a single chip. In the context of the prototype developed in this project, a Raspberry Pi will be used as a processor to support the deep learning model and other devices.

4.1 The Raspberry Pi

A Raspberry Pi is a small single-board computer (SBC) that can be used for various purposes such as learning programming and robotics, among others. It was released into the market the year 2012 by the Raspberry Pi Foundation based in the UK that gives its name. Its original purpose was to become a low cost, modular and open designed computer to teach basic computer science in developing countries. It quickly began popularizing among hobbyists around the world interested in fields such as robotics and computing. It contains a system-on-a-chip (SoC) from Broadcom that includes an ARM microprocessor. Raspberry Pi is a suitable choice for the prototype as it can run the model and write python code, while being compact and adaptable. Since its launching, different series and generations have been released. It is a matter of choosing the adequate candidate [21].

The following chart summarizes the characteristics taken into account for the project among the different Raspberry Pi series and generations:

Family	**Model**	**GPIO pins**	**Raspberry Pi Camera**	**USB**	**RAM**	**Wireless access**	**Price (€)**
Raspberry Pi	B+	40	Yes	4 USB 2.0	512MB	No	35
	A+			1 USB 2.0	512MB	No	27
Raspberry Pi 2	B	40	Yes	4 USB 2.0	1GB	No	32
Raspberry Pi 3	B+	40	Yes	4 USB 2.0	1GB	Yes	43
	A+			1 USB 2.0	512MB		32
Raspberry Pi 4	B	40	Yes	2 USB 2.0 2 USB 3.0	1/2/4/8 GB	Yes	46/50/67/81
Raspberry Pi Zero	Zero	40*	Yes	1 Micro USB	512MB	No	12
	W/WH	40*		1 Micro USB		Yes	19
	2W	40		1 Micro USB			17

Raspberry Pi Pico	Pico	26	No	1 USB 1.1	264kB	No	5
	W	26		1 USB 1.1		Yes	8

Table 4.1: Summary of characteristics of Raspberry Pi models [22] [23] [24] *The GPIO have to be soldered

This project's prototype requires at least the Raspberry to access a camera, a USB for power supply and General Purpose Input/Output pins (GPIO). Raspberry Pi Model A, B and 3B are not considered for the analysis as they were superseded by superior models, Raspberry Pi 4 400 is also excluded because it is assembled inside a keyboard and that makes its usage impracticable for computer vision purposes.

The number of USB ports and whether the device is compatible with the Raspberry Pi Camera is considered in the analysis because USB can be used to plug a third party camera. The election should take into account the budget and preferences as well as performance, quality and compatibility implications. The Raspberry Pi camera uses MIPI CSI-2 interface which connects directly to the GPU of the Raspberry Pi while USB cameras use standard USB ports that connect to its CPU. This means that Raspberry Pi cameras can achieve higher resolution, frame rate and video encoding than most USB cameras especially for high-definition video streaming or recording. They also have lower overhead and resource consumption than USB cameras since they don't rely on CPU or USB bus that may be shared with other devices. Additionally, they may have more mounting options and flexibility since they are smaller and lighter. The number of USB ports and whether the device is compatible with the Raspberry Pi Camera should also be considered since USB can be used to plug in third-party cameras.

RAM is a parameter which is relevant when it comes to executing the tflite deep learning model. While tflite models are optimized for low memory and power consumption, they still need some RAM to store the model parameters and intermediate activations during inference.

While the project did not consider it, wireless access could be used to make further improvements and additions such as giving remote instructions, sending images to databases, triggering notifications or real-time monitoring.

In order to develop this project, we will use a Raspberry Pi 4B, see Fig. 4.1. Since the aim of this project is to build a prototype, it is crucial not to fall short.

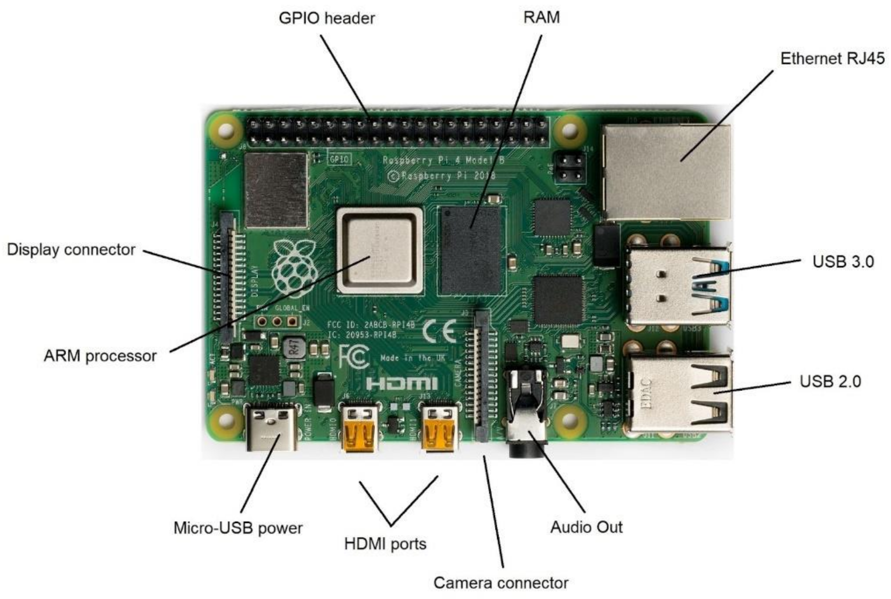


Fig. 4.1: Parts of a Raspberry Pi 4B

The full characteristics of the Raspberry Pi 4B can be found in the annex.

4.2 Camera

While the project does not have fixed requirements as per resolution of the camera and the number of pixels used in the images inside the model are 124x124, the greater the resolution is the easier it will be to prevent aliasing from distorting the images.

All things considered, given the importance of selecting a camera that is fully compatible with the Raspberry Pi, the Raspberry Pi camera module will be used for the prototype. This camera module connects to the Raspberry Pi’s Camera Serial Interface (CSI) bus connector via a flexible ribbon cable, ensuring optimal performance and seamless integration. Using a generic camera plugged through the USB would not provide the same level of compatibility and could result in suboptimal performance.

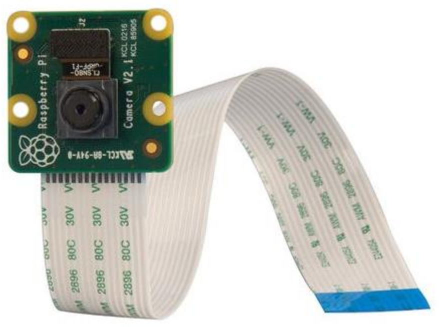


Fig. 4.2: Raspberry Pi Camera

The Sony IMX219 8 megapixels camera requires 5V and 1.8mA from the Raspberry Pi. From experimental testing, the camera has an aperture angle of approximately 65º. According to the provider, to avoid malfunction or damage to the camera, it should not be exposed to water, moisture, or be placed on a conductive surface. Additionally, it should not be exposed to heat sources as it is designed to work at normal ambient room temperatures [25]. See the datasheet in the annex for extra information.

4.3 LCD screen

For the prototype, an LCD screen was purchased to supervise the results of the execution of the code. Raspberry Pi 4B has a 15-pin DSI connector that can be used to connect to DSI displays, see Fig. 4.3.

The selected screen has a 5-inch touch screen with a resolution of 800x480 pixels and a refresh rate of 60Hz. The screen adds an additional weight of 191g to the prototype [26]. In the annex there is the datasheet with the instructions required to configure the display.

Fig. 4.3: Back (left) and frontal (right) view of the screen. Mounting not used.

4.4 Actuator

A button actuator is used in the prototype to control the execution of the code. Fig. 4.4 shows the electrical connection with the Raspberry Pi. The black wire is connected to the sixth pin, corresponding to the ground and the red wire is connected to the fifth pin which corresponds to GPIO 3.

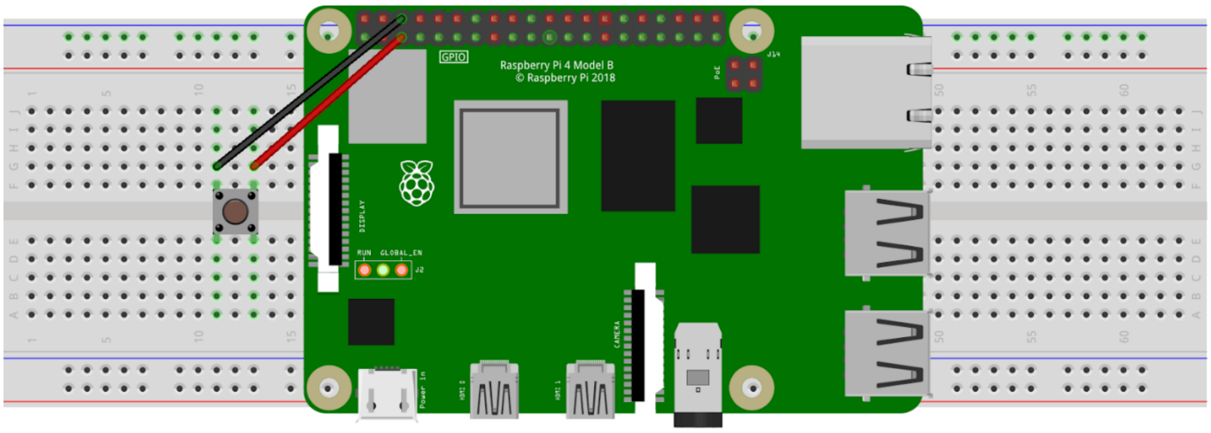


Fig. 4.4: Connections of the button with the Raspberry Pi.

4.5 Power Source

The P200 Posugear power bank is a Li-ion battery with a capacity of 20000mAh that can power the Raspberry Pi when there is no nearby power plug. It has 2 USB A outputs and 1 USB-C for fast charging, delivering 5V DC. With a total maximum output of 22.5W, it can provide up to 15 hours of autonomy at 400% CPU capacity. The power bank measures 10.8 x 6.9 x 2.75 cm and adds an additional weight of approximately 290g to the prototype [27].

Fig. 4.5: Posugear power battery.

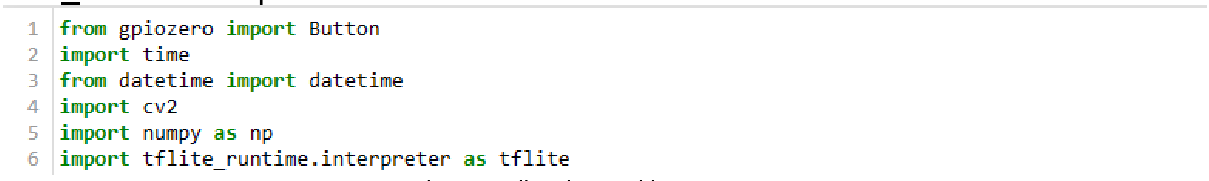
Additionally a USB-C to USB-A male charging cable with a switch is acquired so that the Raspberry Pi can be turned on and turned off without unplugging it from the power battery.

Fig. 4.6: USB-C to USB-A cable with switch

4.6 Code

A Python file is created in the Raspberry Pi to execute the model while the camera is running.

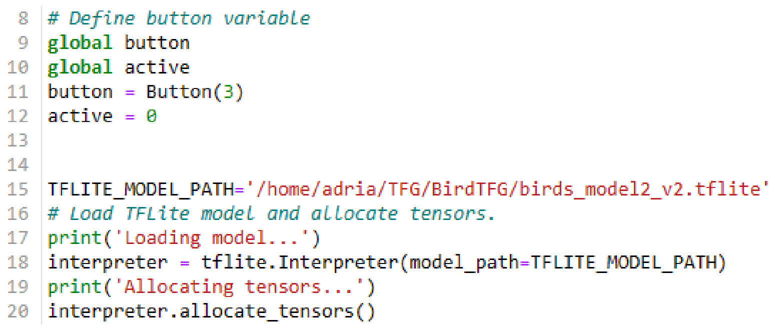
The following libraries are imported for the execution of the program: gpiozero pins Button, time, datetime, cv2, numpy and tflite_runtime.interpreter. The gpiozero pins Button library is used to control the button input. The time library is used to stop recording when a time limit is reached. The datetime library is used to keep track of the moment when the bird was detected. The cv2 library is used to run the camera. The numpy library is used to handle arrays. Finally, tflite_runtime.interpreter is used to execute the model.



```python
from gpiozero import Button
import time
from datetime import datetime
import cv2
import numpy as np
import tflite_runtime.interpreter as tflite
```

Fig. 4.7: Libraries and imports

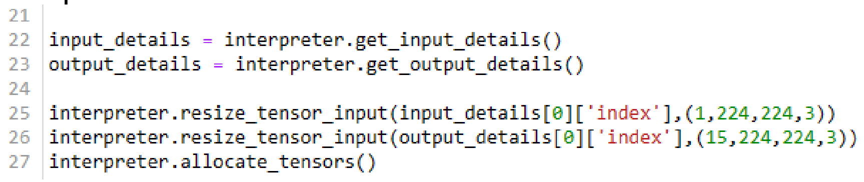
Two global variables are declared in the program. The first variable, button, is assigned to the GPIO 3 pin according to the schematics in Fig. 4.4. The second variable, active, is used to determine whether the camera is currently recording or not.
The saved model is located in the path saved in TFLITE_MODEL_PATH. After loading the interpreter using tflite.Interpreter, memory must be allocated for the input and output tensors of the model using the allocate_tensors() method.



```
# Define button variable
global button
global active
button = Button(3)
active = 0


TFLITE_MODEL_PATH='/home/adria/TFG/BirdTFG/birds_model2_v2.tflite'
# Load TFLite model and allocate tensors.
print('Loading model...')
interpreter = tflite.Interpreter(model_path=TFLITE_MODEL_PATH)
print('Allocating tensors...')
interpreter.allocate_tensors()
```

Fig. 4.8: Extract of the code where the main variables are declared

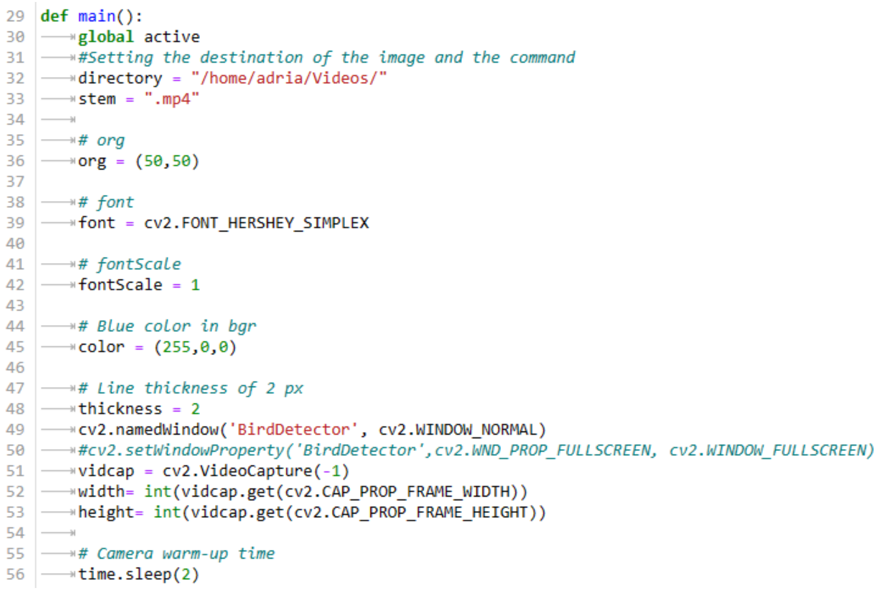
The get_input_details() method is used to get details about the input tensor of the model analogously the get_output_details() method gets details about the input tensor. In order to be used, the input tensor and output tensors are resized to (1,224,224,3) and (15,224,224,3) respectively and then are allocated in the interpreter.



```

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

interpreter.resize_tensor_input(input_details[0]['index'],(1,224,224,3))
interpreter.resize_tensor_input(output_details[0]['index'],(15,224,224,3))
interpreter.allocate_tensors()
```

Fig. 4.9: USB-C to USB-A cable with switch

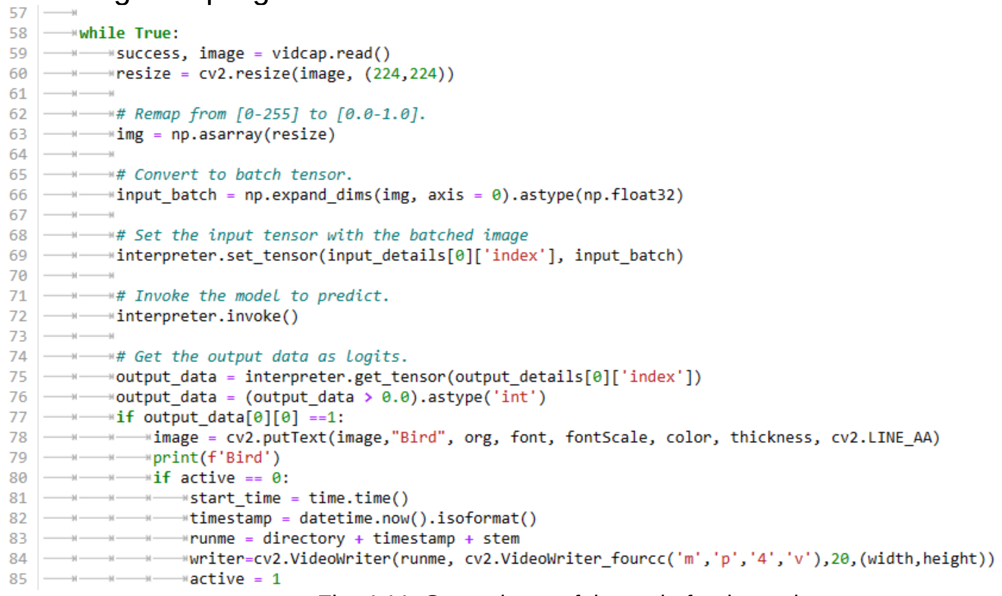
In the main program, which is the one that will run indefinitely, the directory where the videos from the detected birds will be stored is declared, the video extension will be mp4. Afterwards, the same is done with the parameters of the tag text displaying the word “bird” when it is detected (org, font, fontScale, color and thickness). In order to prompt a window displaying the images from the camera the VideoCapture(-1) method is called. The number -1 corresponds to the Raspberry Pi camera, if it does not work, the number must be changed. Said window will occupy the entirety of the display. To allow the camera to warm up, a wait time of two milliseconds is used.



```python
def main():
    global active
    #Setting the destination of the image and the command
    directory = "/home/adria/Videos/"
    stem = ".mp4"
    
    # org
    org = (50,50)

    # font
    font = cv2.FONT_HERSHEY_SIMPLEX

    # fontScale
    fontScale = 1

    # Blue color in bgr
    color = (255,0,0)

    # Line thickness of 2 px
    thickness = 2
    cv2.namedWindow('BirdDetector', cv2.WINDOW_NORMAL)
    #cv2.setWindowProperty('BirdDetector',cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
    vidcap = cv2.VideoCapture(-1)
    width= int(vidcap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height= int(vidcap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    
    # Camera warm-up time
    time.sleep(2)
```

Fig. 4.10: Part of the code for the main

Inside the loop, images from the camera are saved in a variable named image. They are resized to (224, 224), which is compatible with the ImageNet model. The image is then converted to a batch tensor and set to the interpreter. To obtain a prediction, the invoke() method is called. If the output data is positive (above 0), the prediction corresponds to a bird. If the prediction is positive, the tag is prepared. It sets the starting time, stores the date-time to use as the name for the file and starts recording the video. The active global variable is set to 1 as the recording is in progress.

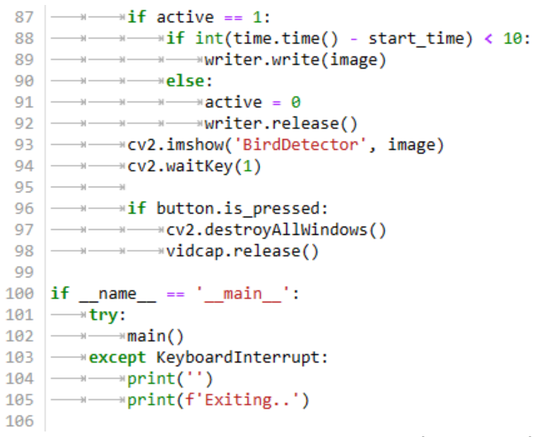


```python
    
    while True:
        success, image = vidcap.read()
        resize = cv2.resize(image, (224,224))
        
        # Remap from [0-255] to [0.0-1.0].
        img = np.asarray(resize)
        
        # Convert to batch tensor.
        input_batch = np.expand_dims(img, axis = 0).astype(np.float32)
        
        # Set the input tensor with the batched image
        interpreter.set_tensor(input_details[0]['index'], input_batch)
        
        # Invoke the model to predict.
        interpreter.invoke()
        
        # Get the output data as logits.
        output_data = interpreter.get_tensor(output_details[0]['index'])
        output_data = (output_data > 0.0).astype('int')
        if output_data[0][0] ==1:
            image = cv2.putText(image,"Bird", org, font, fontScale, color, thickness, cv2.LINE_AA)
            print(f'Bird')
            if active == 0:
                start_time = time.time()
                timestamp = datetime.now().isoformat()
                runme = directory + timestamp + stem
                writer=cv2.VideoWriter(runme, cv2.VideoWriter_fourcc('m','p','4','v'),20,(width,height))
                active = 1
```

Fig. 4.11: Second part of the code for the main

If the active global is set to 1 the time is checked to make sure it is below 10 seconds. If the current time minus the starting time is above the 10 seconds limit then the active global value is set to 0 and it stops recording.

If the button is pressed the execution of the program is stopped and the loop is broken.



```
        if active == 1:
            if int(time.time() - start_time) < 10:
                writer.write(image)
            else:
                active = 0
                writer.release()
        cv2.imshow('BirdDetector', image)
        cv2.waitKey(1)

        if button.is_pressed:
            cv2.destroyAllWindows()
            vidcap.release()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('')
        print(f'Exiting..')
```

Fig. 4.12: Final part of the code

4.7 Final concept

The prototype concept includes an electrical connection box that contains components. A slit was opened in the cover to introduce the camera, which is secured with four screws. The button can also be accessed from outside.

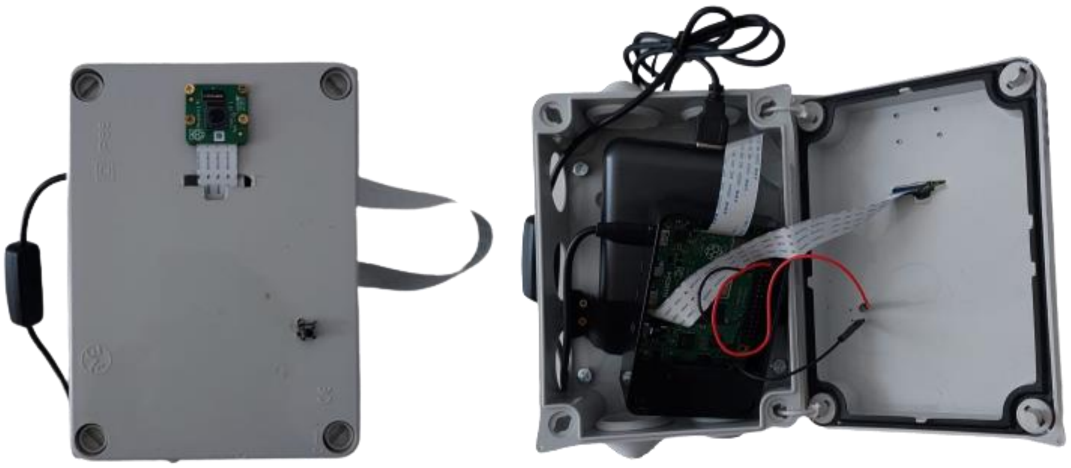


Fig. 4.13: Prototype concept from the outside (front) and from the inside

In the inside of the box the power source is plugged to the Raspberry Pi through the USB-C to USB-A male to male cable to control the switching on and off of the CPU.
So as to have more control over the prototype a display was attached to the back and screws to serve the purpose as legs were added so that the display would not be damaged when placed over a surface.

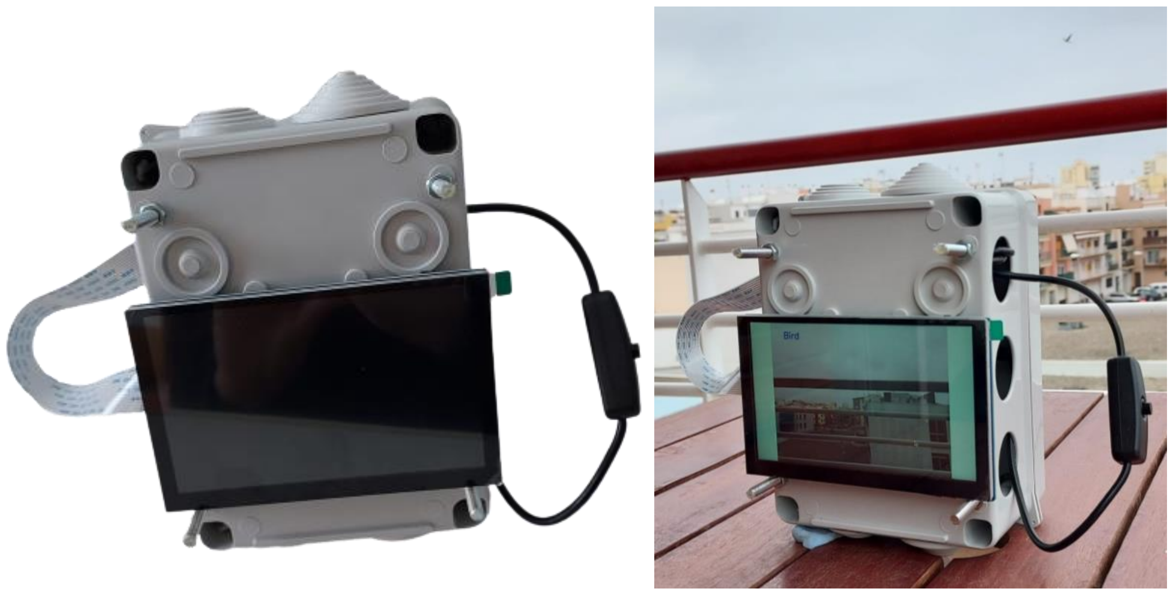


Fig. 4.14: Prototype concept from the back switched off and running

On Fig. 4.14 on the right, the program is running and a bird is detected which is visible in the top right of the image.

CHAPTER 5: Dataset for computer vision applications

A dataset in computer vision is a collection of images that are used to train and/or test a model. It consists of examples belonging to a particular class i.e objects, people, drawings or places. According to the type of computer vision model, a dataset may be labeled or unlabeled.

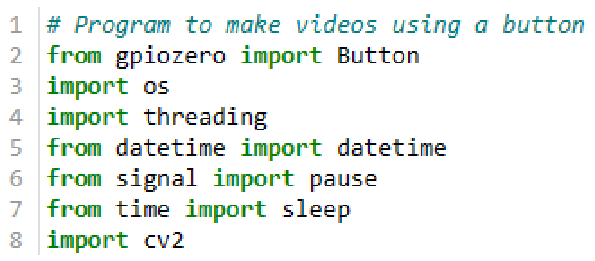
To create a dataset in computer vision, the object of study has to be analyzed in order to determine the kind of data needed to solve the problem. The size of the dataset, determined by the number of images required to properly describe the situation as well as the quality of the images should also be assessed. It is important to ensure that the dataset is diverse and representative of the real-world scenarios that the model will encounter. Another aspect to take into account is that the dataset should be as balanced as possible, meaning that each class has roughly the same number of samples.

5.1 Creating a dataset

The purpose of the present project is to create a computer vision for bird strike prevention. The model resulting from the project will be a supervised classification one, either the object in front of the camera is a bird or it is not. As such, the classification will be binary and images of birds and non-birds will be used to train and validate the models. As the camera will be pointed to the sky, the non-bird category will be composed of images of the sky with different cloud patterns, images of airplanes and images of drones.

The dataset will be created with as many images of birds and non-birds as possible while trying to maintain balance. The images will be obtained through the camera of the Raspberry Pi that will be used when executing the model in the real-time application. This will ensure that the pictures have the same quality expected when running the application.

A python script is designed to record videos from the sky. Once these videos are obtained, different frames depicting birds, the sky, planes or drones will be manually extracted and saved in the corresponding folder. Fig. 5.1 shows the libraries needed for the script.



```python
# Program to make videos using a button
from gpiozero import Button
import os
import threading
from datetime import datetime
from signal import pause
from time import sleep
import cv2
```

Fig. 5.1: Libraries used in the code

A button will be used to control the recording, Fig. 5.2 shows its declaration.

```
10 global button
11 button = Button(3)
```

Fig. 5.2: Global variable for the button in GPIO3

The changeState function receives the button parameter as an input. Inside this function, the camera is activated or deactivated and full-screen videos are saved inside the specified directory with the format given in line 22 in accordance with line 16.

```
13 def changeState(button):
14     #Setting the destination of the image and the command
15     directory = "/home/adria/Videos/"
16     stem = ".mp4"
17     timestamp = datetime.now().isoformat()
18     runme = directory + timestamp + stem
19     cap = cv2.VideoCapture(-1)
20     width= int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
21     height= int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
22     writer=cv2.VideoWriter(runme, cv2.VideoWriter_fourcc('m','p','4','v'),20,(width,height))
23     print("Starting recording")
```

Fig. 5.3:First part of changeState function

The camera records frames while the button is not pressed.

```
24     while True:
25         ret, frame= cap.read()
26 
27         writer.write(frame)
28 
29         cv2.imshow('frame',frame)
30 
31         if button.is_pressed:
32             print("bye")
33             sleep(1)
34             break
35 
36     cap.release()
37     writer.release()
38     cv2.destroyAllWindows()
```

Fig. 5.4: Second part of changeState function

The thread controls whether the button has been pressed or not at all times.

```
40 def Activate():
41     button.when_pressed = changeState
42     pause()
43 
44 tmp = threading.Thread(target = Activate)
45 tmp.start()
```

Fig. 5.5: Activate function and thread declaration

5.2 Resulting dataset

From the videos obtained with the script a total of 3282 images were saved distributed as follows:

- Birds: 1462 images
- No-birds: 1820 images, among which
    - 814 drone images
    - 537 plane images
    - 469 sky images

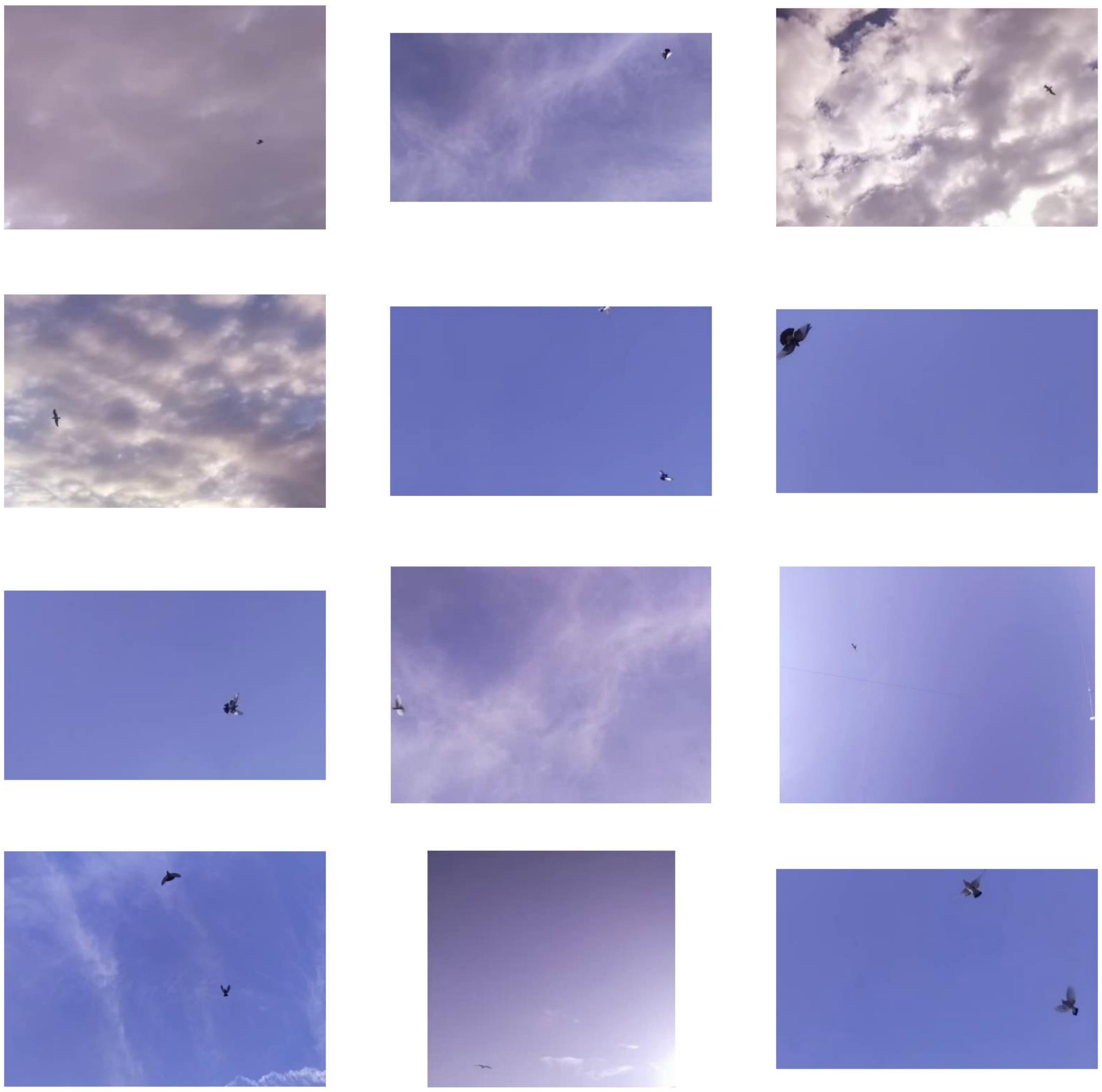


Fig. 5.6: Twelve randomly picked images from the bird dataset

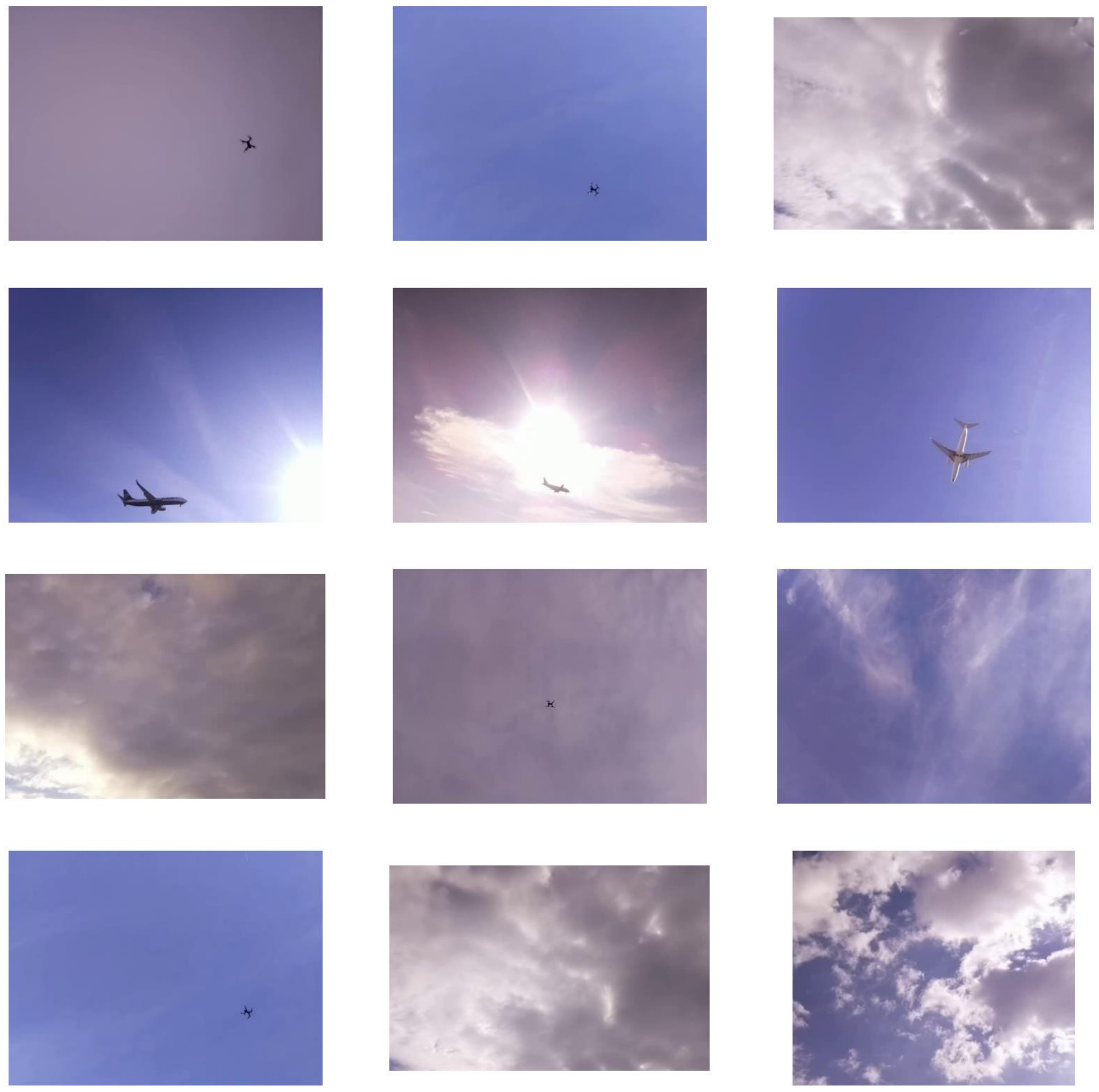


Fig. 5.7: Twelve randomly picked images from the nobird dataset

The distribution of images of different categories of no-birds is a result of trial and error in real-time application to improve model performance. From experimental results, the model appeared to have more difficulty distinguishing between birds and drones than between birds and planes. False positives with plain sky were limited and varied according to cloud patterns in the sky, so different sky images were considered. Regarding bird classification, most images corresponded to pigeons and seagulls at different altitudes, see Fig. 5.6.

While the dataset is not perfectly balanced, it has demonstrated high accuracy levels. The images of birds were saved in a folder named “birds” and the images of no-birds were saved in a folder named “nobirds” without regard to their subcategory. This serves the purpose of labeling the images in the code

5.3 Additional considerations

It is possible to create datasets from images obtained from the Internet. However, Tensorflow may encounter problems when fitting the model as images may be in a non-recognized format or be corrupt. Additionally, images found on the internet depict the object of study in full detail, centered in the frame which is not the case most of the time in real-life applications. The quality of the source image may also be different from the images taken by the prototype and thus resulting in poor accuracy.

It is desirable to have as many images as possible and have variations of them, for this reason data augmentation techniques may help in increasing the size and diversity of the dataset. Data augmentation combined with the usage of transfer learning improves the efficiency of the model when a limited number of images are available.

CHAPTER 6: ANALYZING MODEL PERFORMANCE AND REAL-WORLD APPLICATION

A supervised deep learning model based on a Convolutional Neural Network (CNN) is trained by updating the “weights” for different filters applied to images according to the results obtained by checking with training and validation datasets as explained in Chapter 3 of “Designing a computer vision system”. This chapter explains different metrics used to evaluate and validate computer vision models during and after its creation, with a focus on the model designed for this project.

6.1 Training metrics

During model training, the code aims to find the optimal value of a control parameter. In deep learning models, this variable is the loss function which measures how well the model is performing by evaluating the difference between predicted and expected outcomes. In this project, Binary Cross Entropy is used as a loss function.

Finding the minimum value of this function is challenging because it is determined by multiple factors that are related to each other in an unknown way. The model depends on several hyperparameters such as the number of images, the distribution between validation and test, the percentage of dropout of neurons, the number of epochs, the size of the batches, and the learning rate among others.

The training and validation curves can be used to describe the training process of a model. These curves show how well the model is performing by plotting the training and validation loss against the number of epochs. By trying different combinations of hyperparameters, their effects impact on the validation curves.

A phenomenon to be avoided when developing deep learning models is underfitting. Underfitting happens when the model is unable to converge. It can be produced for two main reasons, either the model is unable to learn the training data (see Fig. 6.1) or the training has stopped too early and the model could converge with additional epochs, see Fig. 6.2.

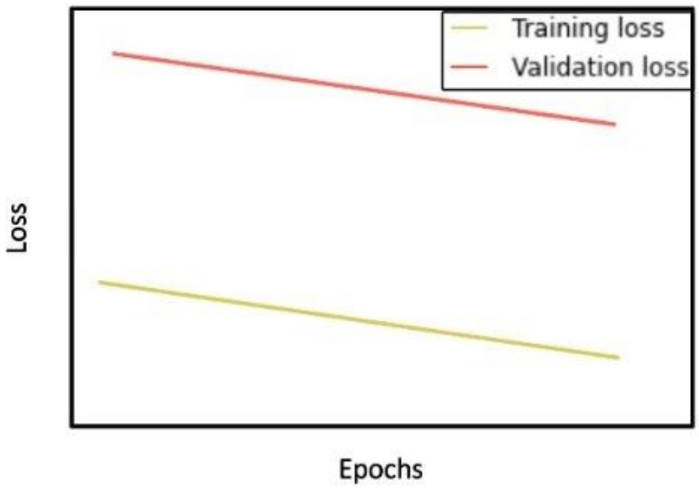


Fig. 6.1: Model unable to learn the data [28]

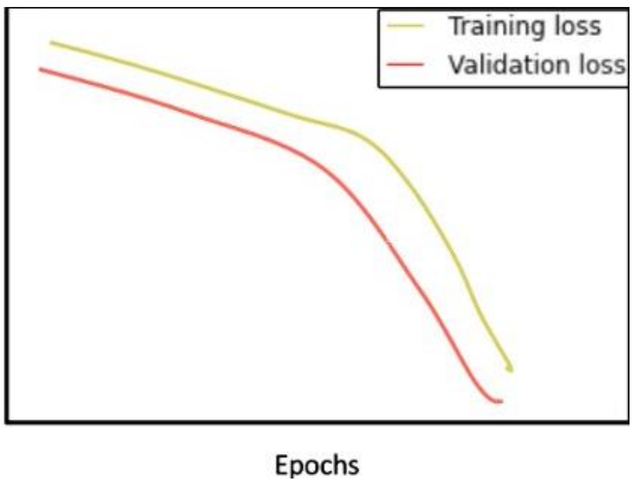


Fig. 6.2: Model about to converge [28]

If, on the other hand, the validation loss is much higher than the training loss, it indicates that the model is overfitting. Overfitting occurs when the model obtains good results for the training data but not for validation data, see Fig. 6.3. One way to solve overfitting is by modifying the probability of dropping out a neuron. As the dropout parameter is increased, the capacity of the network is reduced and the model becomes less likely to overfit. However, if the dropout parameter is set too high, then the model may underfit and have poor performance on both the training and test data. Other hyperparameters such as learning rate and batch size may be required to be edited to balance it.

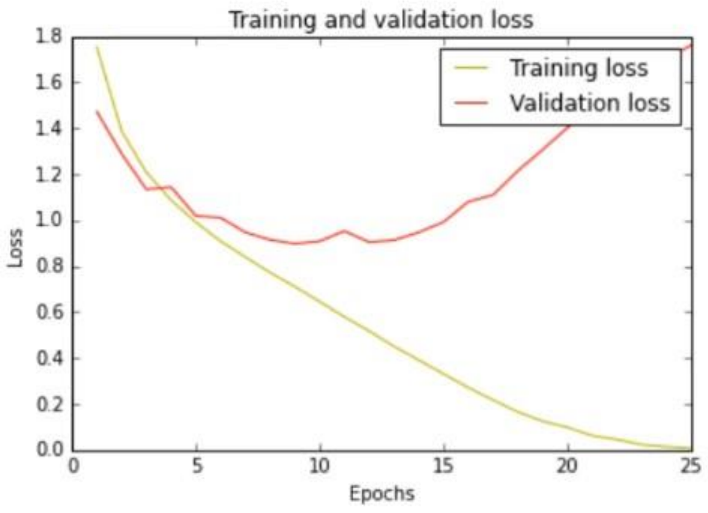


Fig. 6.3: Example of overfitting in a validation curve [28]

The resulting validation curves for the model trained for this project are the following:

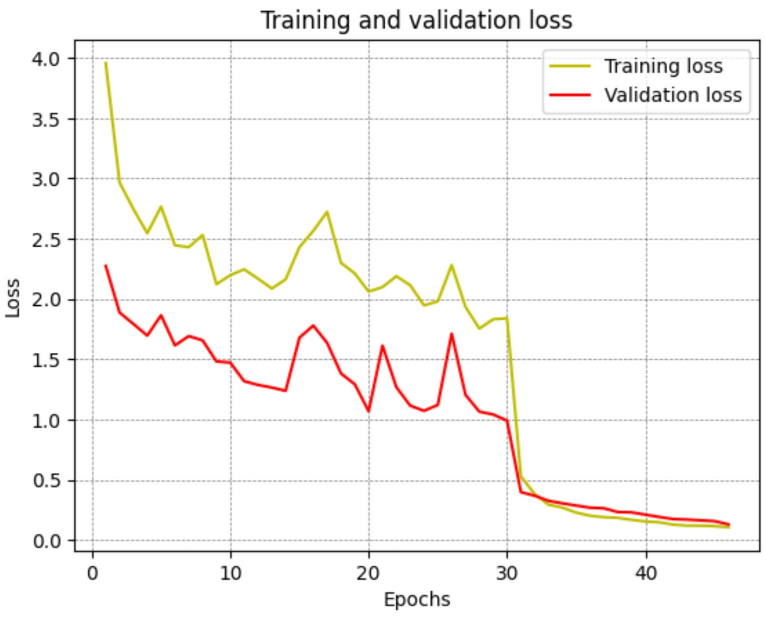


Fig. 6.4: Validation curve for the project's resulting model (own creation)

The validation curve shows that the layers of the base_model corresponding to the pre-trained model Imagenet_v2 are frozen during the first thirty epochs to perform transfer learning and fine-tuning. In the last fifteen epochs, the weights from the hundredth layer onwards are updated. From that point on, the model converges faster in terms of loss and the validation and training loss converge to values close to zero.

The metric used to evaluate the progression of the model with each epoch was the AUC (Area Under the Curve) is a metric used to evaluate the performance of a binary classifier model with each epoch. It is defined as the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate against the false positive rate. An AUC of 1 indicates that the model perfectly separates the two classes, see Fig. 6.5.

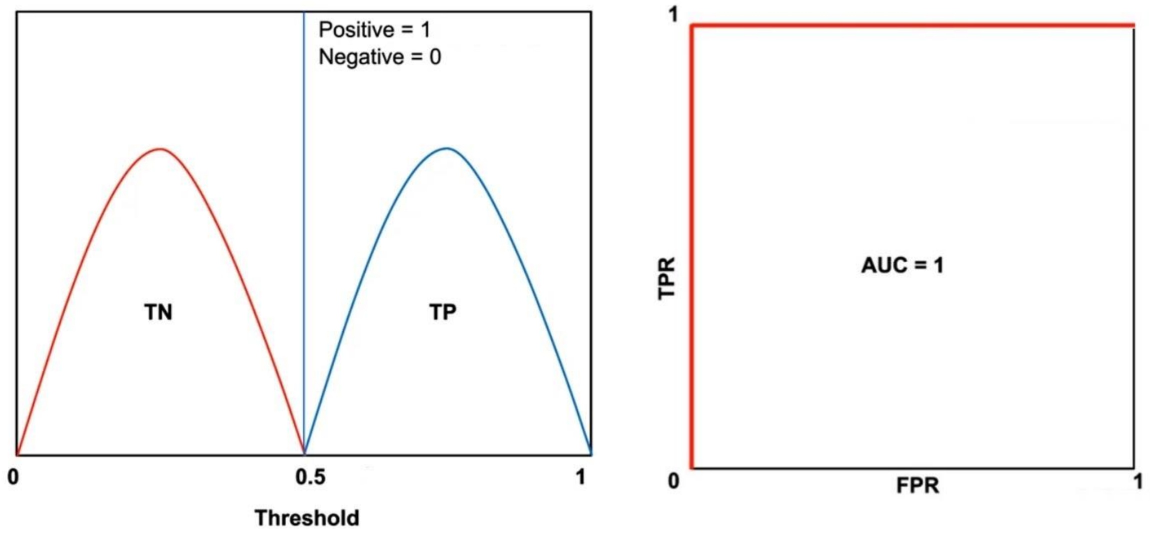


Fig. 6.5: Number of TP and TN according to the probability (left) and ROC curve (right) [29]

An AUC value less than 1 indicates that the model is not perfectly separating the two classes. This can occur when there are false positives and false negatives, meaning that the classification threshold is not able to completely separate the true positives from the true negatives. In the worst case scenario, AUC of 0.5 indicates that the model is no better than random chance, this could be due to an error in labeling.

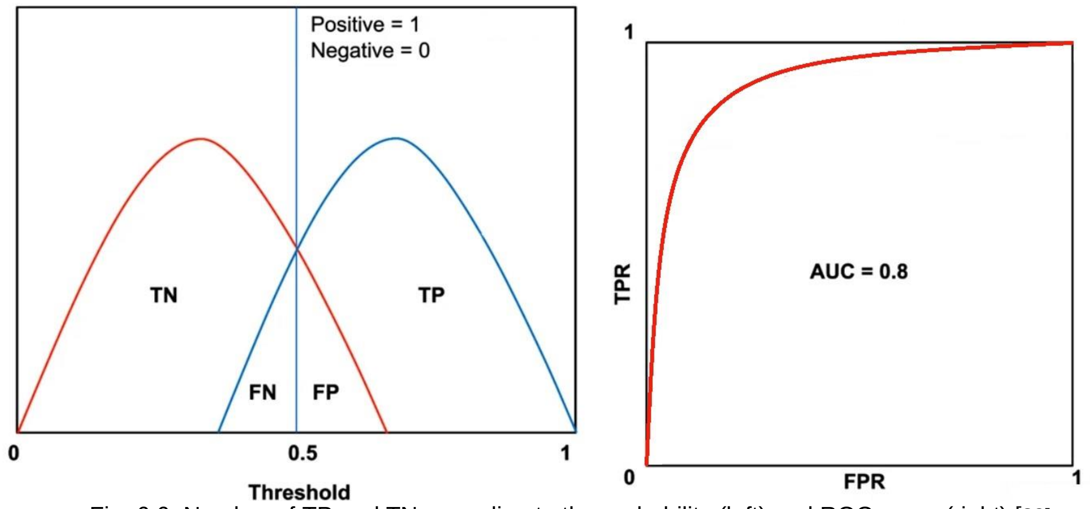


Fig. 6.6: Number of TP and TN according to the probability (left) and ROC curve (right) [29]

Fig. 6.7 displays the evolution of the training and validation AUC over the course of 45 epochs. The AUC values are higher than 0.95, indicating that the model is performing well, although not perfectly.

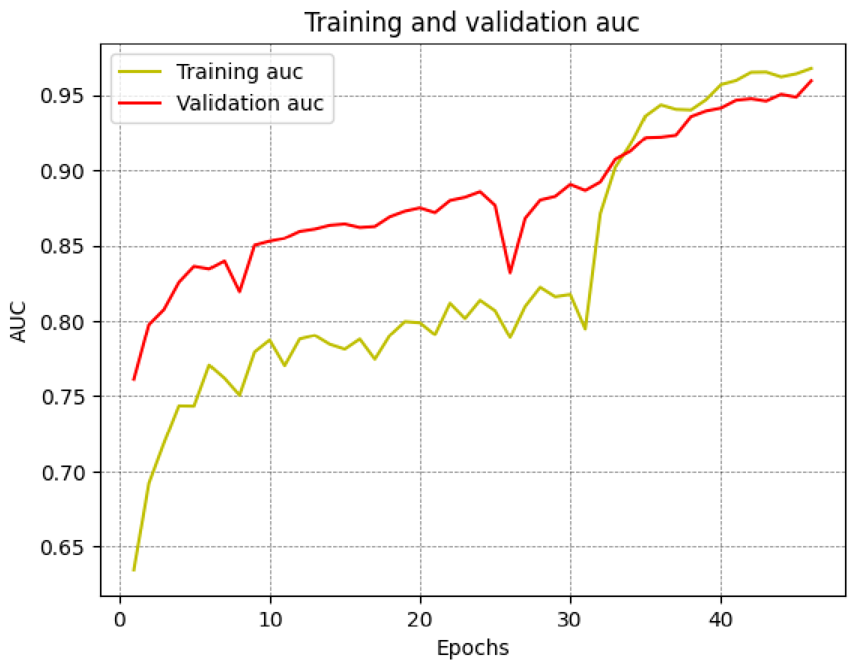


Fig. 6.7: Training and validation AUC curves

6.2 Model metrics

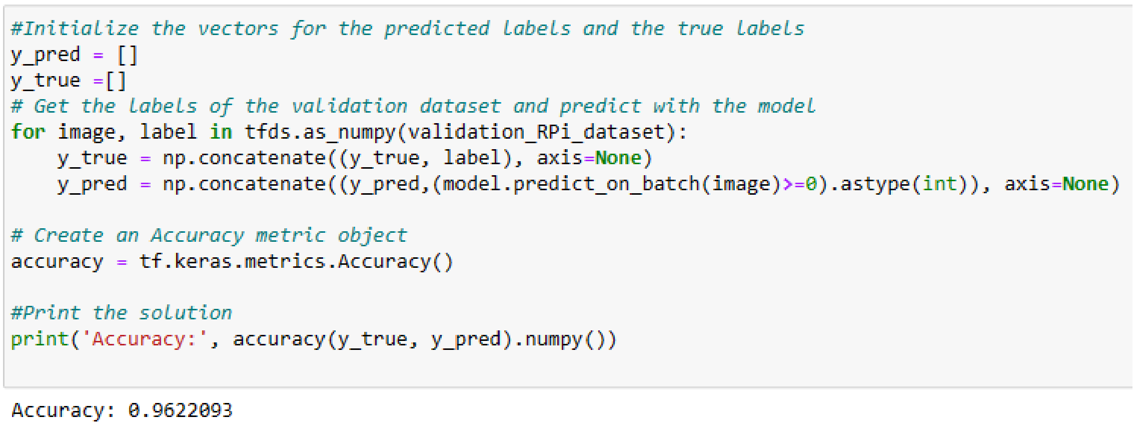
Once the model is finished there are some additional metrics that can be used to measure how it performs.

- ACCURACY

Accuracy is a commonly used performance metric for classification models. It is defined as the ratio of the number of correct predictions to the total number of predictions made by the model. Mathematically, accuracy is calculated as (TP + TN) / (TP + TN + FP + FN), where TP represents the number of true positives, TN represents the number of true negatives, FP represents the number of false positives, and FN represents the number of false negatives. The range of accuracy is from 0 to 1, with 0 in the worst case scenario indicating that TP and TN are both 0 and 1 in the best case scenario indicating that FN and FP are both 0. Accuracy provides a measure of how well the model can correctly classify instances into their respective classes [30].

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad (6.1)$$

The accuracy for the validation dataset can be computed using the tensorflow tools shown in Fig. 6.8.



```
#Initialize the vectors for the predicted labels and the true labels
y_pred = []
y_true =[]
# Get the labels of the validation dataset and predict with the model
for image, label in tfds.as_numpy(validation_RPi_dataset):
    y_true = np.concatenate((y_true, label), axis=None)
    y_pred = np.concatenate((y_pred,(model.predict_on_batch(image)>=0).astype(int)), axis=None)

# Create an Accuracy metric object
accuracy = tf.keras.metrics.Accuracy()

#Print the solution
print('Accuracy:', accuracy(y_true, y_pred).numpy())
```

```
Accuracy: 0.9622093
```

Fig. 6.8: Computing the accuracy for the validation dataset using Tensorflow

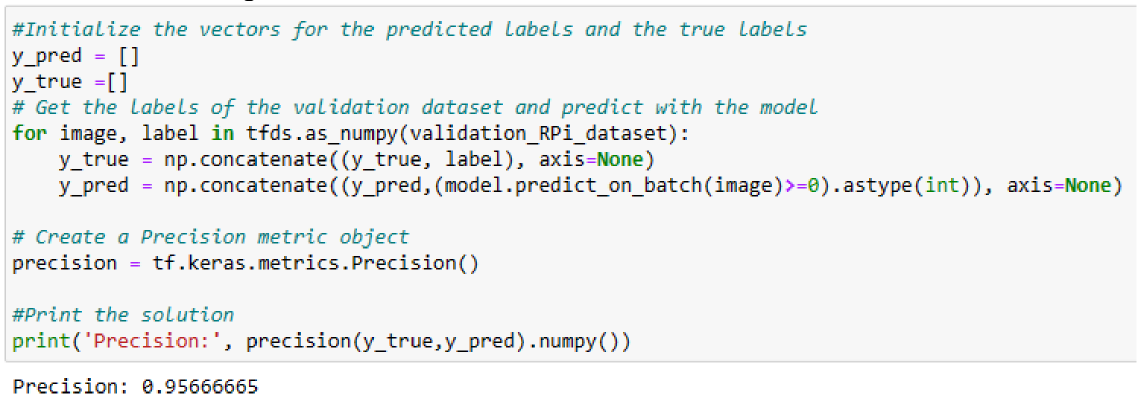
The accuracy for the model is 0.9622093 which is close to the optimal.

- PRECISION

Another commonly used performance metric for classification models is precision. It is calculated as the ratio of TP predictions to the total number of positive predictions made by the model (TP + FP). The range of accuracy is from 0 to 1, with 0 in the worst case scenario indicating that TP is 0 and 1 in the best case scenario indicating that FN is 0. Precision measures how well the model correctly identifies positive samples [31].

$$precision = \frac{TP}{TP+FP} \qquad (6.2)$$

The precision for the validation dataset can be computed using the tensorflow tools shown in Fig. 6.9.



```
#Initialize the vectors for the predicted labels and the true labels
y_pred = []
y_true =[]
# Get the labels of the validation dataset and predict with the model
for image, label in tfds.as_numpy(validation_RPi_dataset):
    y_true = np.concatenate((y_true, label), axis=None)
    y_pred = np.concatenate((y_pred,(model.predict_on_batch(image)>=0).astype(int)), axis=None)

# Create a Precision metric object
precision = tf.keras.metrics.Precision()

#Print the solution
print('Precision:', precision(y_true,y_pred).numpy())
```

```
Precision: 0.95666665
```

Fig. 6.9: Computing the precision for the validation dataset using Tensorflow

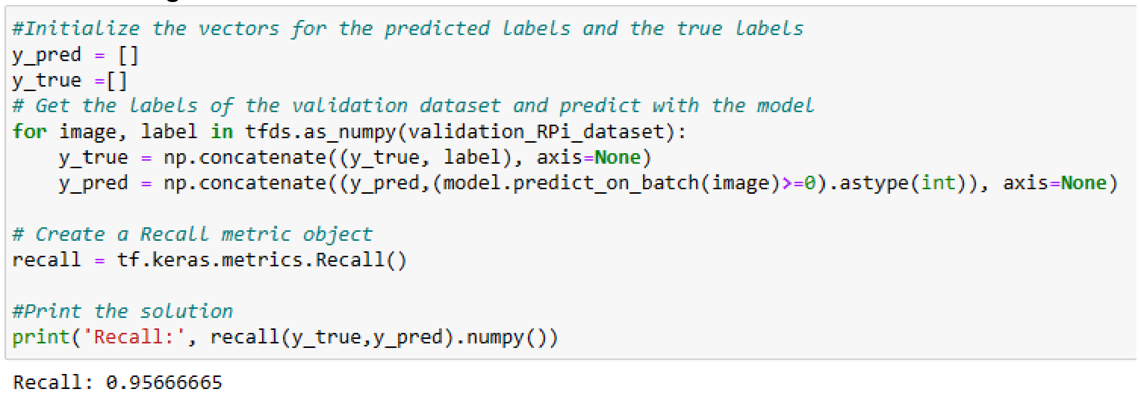
The precision for the model is 0.95666665 which is close to the optimal.

- RECALL

Recall measures the proportion of actual positive cases that were correctly identified by the model as positive cases. Recall is calculated as the ratio of TP predictions to the total number of actual positive instances (TP + FN). The range of recall is from 0 to 1, with 0 indicating that TP is 0 and 1 indicating that FN is 0. Recall measures how well the model correctly identifies all positive instances [31].

$$recall = \frac{TP}{TP+FN} \qquad (6.3)$$

The recall for the validation dataset can be computed using the tensorflow tools shown in Fig. 6.10.



```
#Initialize the vectors for the predicted labels and the true labels
y_pred = []
y_true =[]
# Get the labels of the validation dataset and predict with the model
for image, label in tfds.as_numpy(validation_RPi_dataset):
    y_true = np.concatenate((y_true, label), axis=None)
    y_pred = np.concatenate((y_pred,(model.predict_on_batch(image)>=0).astype(int)), axis=None)

# Create a Recall metric object
recall = tf.keras.metrics.Recall()

#Print the solution
print('Recall:', recall(y_true,y_pred).numpy())
```

```
Recall: 0.95666665
```

Fig. 6.10: Computing the recall for the validation dataset using Tensorflow

The recall for the model is 0.95666665 which is close to the optimal.

- F1-SCORE

The F1-score is a widely used performance metric for classification models. It is calculated as the harmonic mean of precision and recall, which are two other important evaluation metrics. The formula for F1-score is: F1 = 2 * (precision * recall) / (precision + recall), where precision is the ratio of true positive predictions to the total number of positive predictions made by the model (true positives + false positives) and recall is the ratio of true positive predictions to the total number of actual positive instances (true positives + false negatives). The range of F1-score is from 0 to 1, with 0 indicating the worst possible performance and 1 indicating the best possible performance. The F1-score provides a balance between precision and recall and is especially useful when dealing with imbalanced datasets. A high F1-score indicates that the model has both high precision and high recall [32].

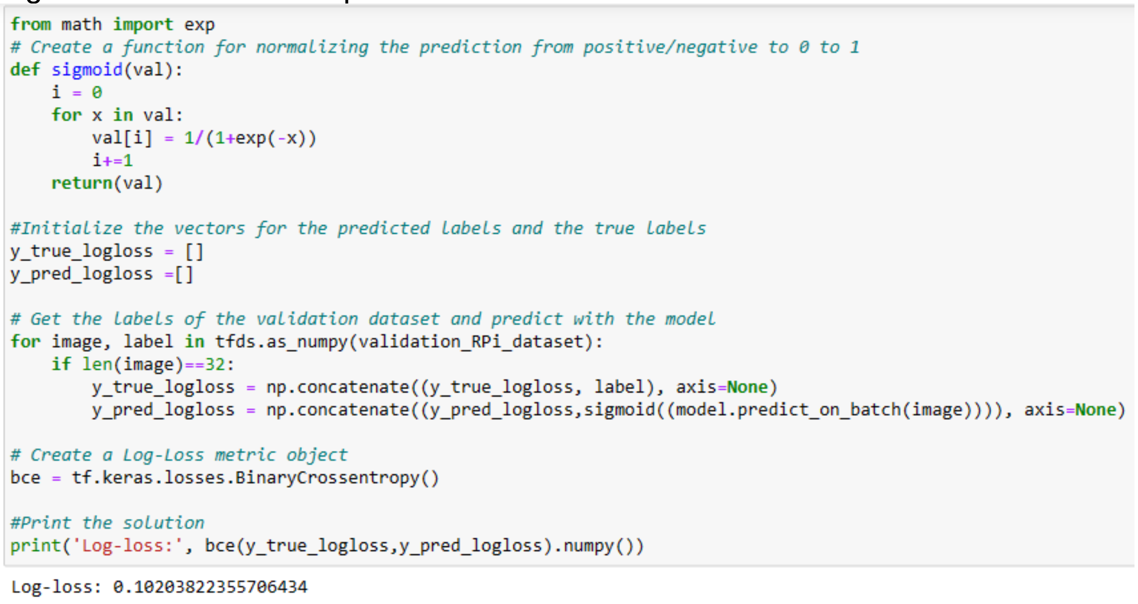
$$F1\ score = 2 * \frac{precision * recall}{precision + recall} = 0.95666665 \qquad (6.4)$$

- LOG-LOSS

Log-Loss, also known as logistic loss or cross-entropy loss, is a loss function used in binary classification problems. It measures the performance of a classification model where the prediction output is a probability value between 0 and 1. Log-Loss takes into account the uncertainty of the prediction based on how much it varies from the actual label. A perfect model would have a Log-Loss of 0. The goal of training a model is to find the best set of weights that minimize the Log-Loss. For a binary classification where the label “y” equals 0 or 1, defining “p” as the probability of the label being 1, the log-loss is computed according to the following formula [33]:

$$\log(y, p) = -(y * \log(p) + (1 - y)\log(1 - p)) \quad (6.5)$$

Using the tensorflow tools, the log-loss for the validation dataset can be computed as shown in Fig. 6.10. In this case, a sigmoid function was required to convert the prediction from positive and negative values to 0 to 1 values according to the probability, additionally it required to eliminate the last batch different than 32 as it gave errors in the compilation of the code.



```
from math import exp
# Create a function for normalizing the prediction from positive/negative to 0 to 1
def sigmoid(val):
    i = 0
    for x in val:
        val[i] = 1/(1+exp(-x))
        i+=1
    return(val)

#Initialize the vectors for the predicted labels and the true labels
y_true_logloss = []
y_pred_logloss =[]

# Get the labels of the validation dataset and predict with the model
for image, label in tfds.as_numpy(validation_RPi_dataset):
    if len(image)==32:
        y_true_logloss = np.concatenate((y_true_logloss, label), axis=None)
        y_pred_logloss = np.concatenate((y_pred_logloss,sigmoid((model.predict_on_batch(image)))), axis=None)

# Create a Log-Loss metric object
bce = tf.keras.losses.BinaryCrossentropy()

#Print the solution
print('Log-loss:', bce(y_true_logloss,y_pred_logloss).numpy())
```

```
Log-loss: 0.10203822355706434
```

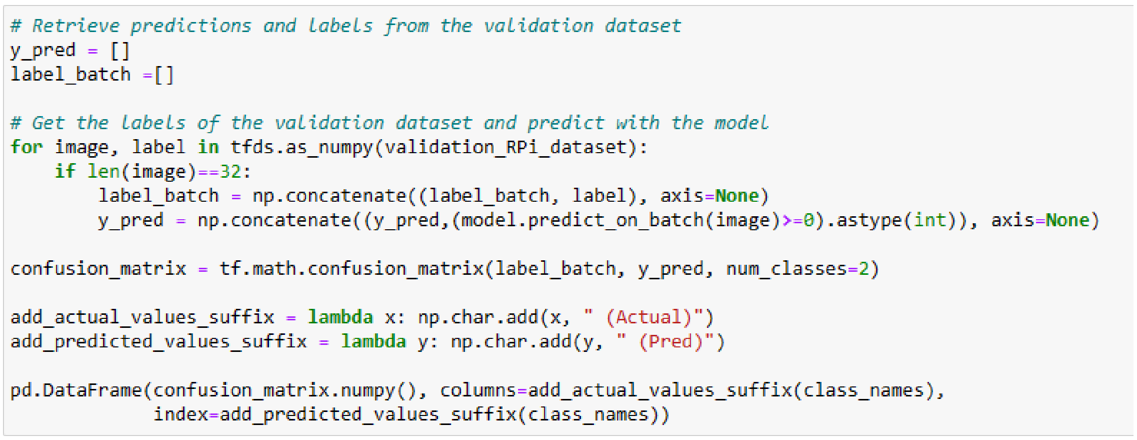
Fig. 6.11: Computing the log-loss for the validation dataset using Tensorflow

The log-loss for the model is 0.1020382 which is close to the optimal.

- CONFUSION MATRIX

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It summarizes the number of correct and incorrect predictions made by the model, broken down by each class. This allows for a more detailed analysis of the model's performance and can help identify where the model may be making errors.

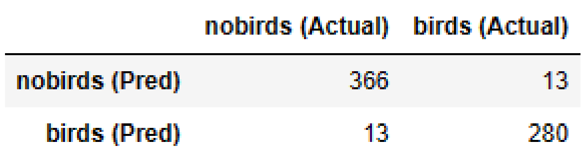
The confusion matrix can be obtained using the pandas library as shown in Fig. 6.12, as with the case of log-loss, the last batch which had less than 32 images is excluded to avoid compilation errors.



```
# Retrieve predictions and labels from the validation dataset
y_pred = []
label_batch =[]

# Get the labels of the validation dataset and predict with the model
for image, label in tfds.as_numpy(validation_RPi_dataset):
    if len(image)==32:
        label_batch = np.concatenate((label_batch, label), axis=None)
        y_pred = np.concatenate((y_pred,(model.predict_on_batch(image)>=0).astype(int)), axis=None)

confusion_matrix = tf.math.confusion_matrix(label_batch, y_pred, num_classes=2)

add_actual_values_suffix = lambda x: np.char.add(x, " (Actual)")
add_predicted_values_suffix = lambda y: np.char.add(y, " (Pred)")

pd.DataFrame(confusion_matrix.numpy(), columns=add_actual_values_suffix(class_names),
             index=add_predicted_values_suffix(class_names))
```

Fig. 6.12: Extracting the parameters from the validation dataset to create a confusion matrix using Tensorflow

In this particular project, the confusion matrix can be used to visually summarize the performance of a classification model in predicting whether an observation is a bird or not. The matrix displays the values for true positives (observations correctly predicted as birds), true negatives (observations correctly predicted as not birds), false positives (observations incorrectly predicted as birds), and false negatives (observations incorrectly predicted as not birds).



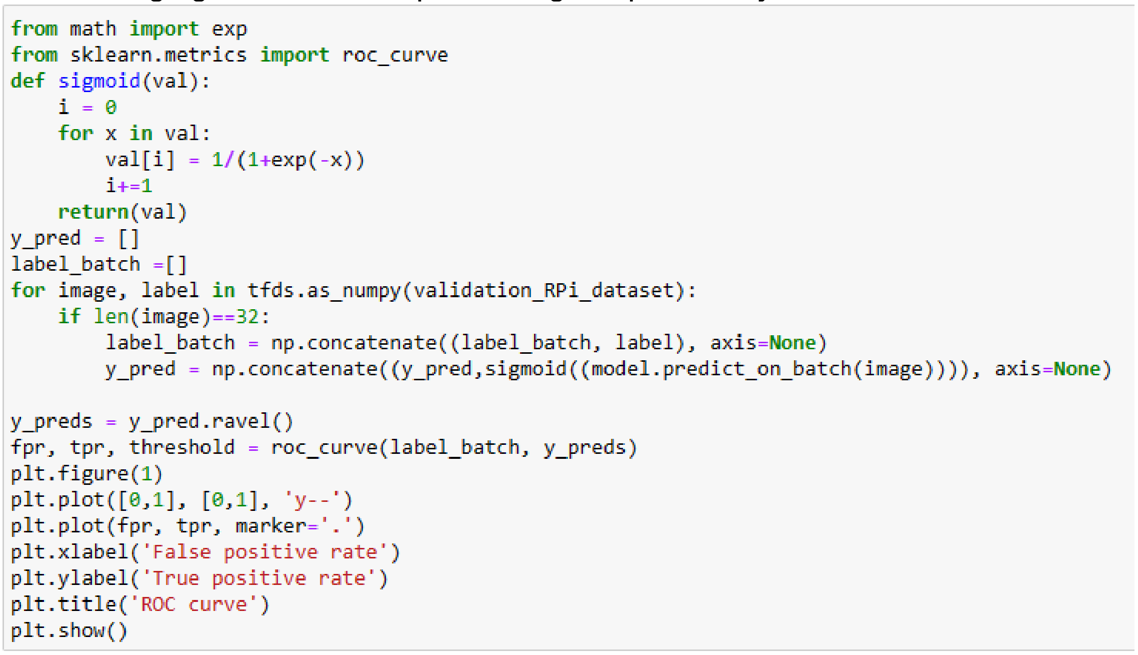
	nobirds (Actual)	birds (Actual)
nobirds (Pred)	366	13
birds (Pred)	13	280

Fig. 6.13: Confusion matrix (own creation)

- RECEIVER OPERATING CHARACTERISTIC

In the Receiver Operating Characteristic (ROC) curves the relation of true positives and false negatives is plotted for the validation images given to the model. Ideally, the false positive rate should be 0 and the true positive rate should be 1, therefore the perfect classifier should have a ROC classifier showing a vertical line in the 0 false positive rate until the value 1 for true positive rate and a horizontal line until the value 1 for false positive rate.

The ROC curve can be generated for the validation dataset using tools from the scikit-learn library. The generation of the ROC curve requires that the batches are complete with 32 images. Additionally, a sigmoid function must be applied to translate the positive/negative values output from the prediction into normalized values ranging from 0 to 1, representing the probability.



```
from math import exp
from sklearn.metrics import roc_curve
def sigmoid(val):
    i = 0
    for x in val:
        val[i] = 1/(1+exp(-x))
        i+=1
    return(val)
y_pred = []
label_batch =[]
for image, label in tfds.as_numpy(validation_RPi_dataset):
    if len(image)==32:
        label_batch = np.concatenate((label_batch, label), axis=None)
        y_pred = np.concatenate((y_pred,sigmoid((model.predict_on_batch(image)))), axis=None)

y_preds = y_pred.ravel()
fpr, tpr, threshold = roc_curve(label_batch, y_preds)
plt.figure(1)
plt.plot([0,1], [0,1], 'y--')
plt.plot(fpr, tpr, marker='.')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.show()
```

Fig. 6.14: Extracting the parameters from the validation dataset to create a ROC curve using sklearn

Fig. 6.15 shows the ROC curve for the validation dataset; the dots represent the individual values of AUC for every batch in the dataset. The blue line describes a shape close to the optimal. A random classifier, with AUC of 0.5, would follow the diagonal yellow dotted line.

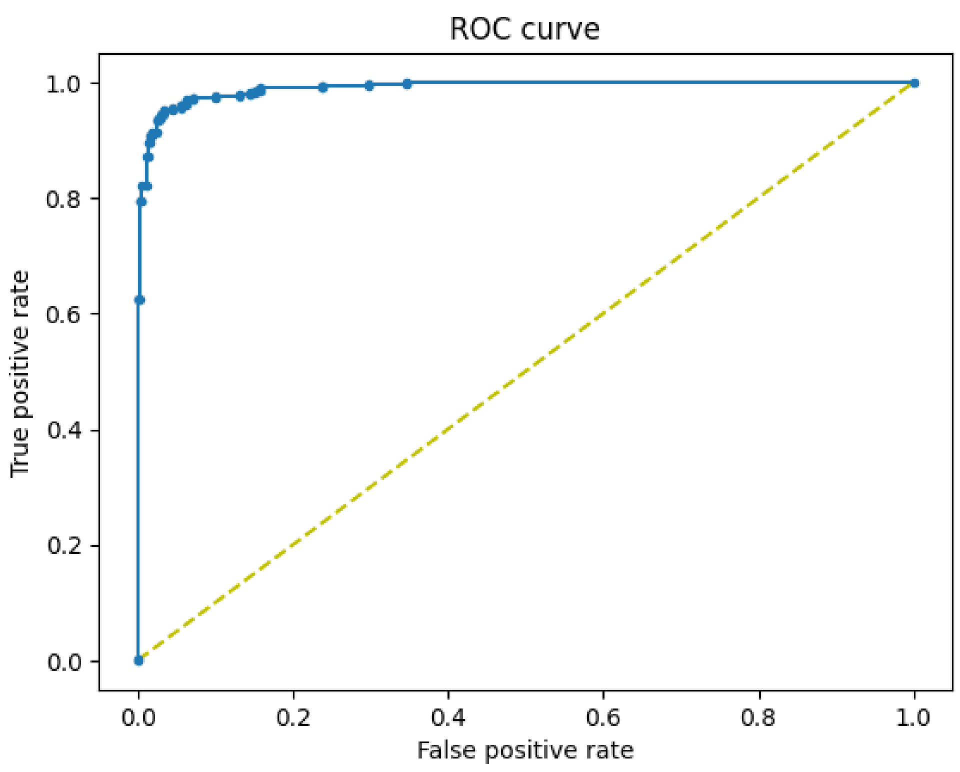


Fig. 6.15: ROC curve obtained from the validation dataset (own creation)

6.3 GRAD-CAM

Grad CAM stands for Gradient-weighted Class Activation Mapping. It is a technique for producing “visual explanations” for decisions from a large class of CNN-based models, making them more transparent. Grad-CAM uses the gradients of any target concept flowing into the final convolutional layer to produce a coarse localization map highlighting important regions in the image for predicting the concept [34]. Next, the code required to create is an adaptation of the one available in the keras reference webpage [35].

Apart from tensorflow, some functions from matplotlib library are necessary to plot and colorize the heatmap.

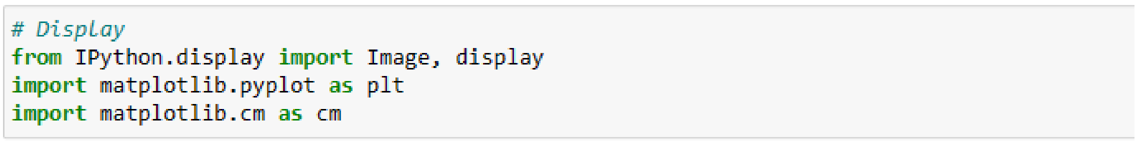

```
# Display
from IPython.display import Image, display
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

Fig. 6.16: Libraries necessary for executing the code

The “make_gradcam_heatmap” function accepts as input the image arrays, the model utilized for generating predictions, and the name of the last convolutional layer. These inputs are used to generate a heatmap visualization of the model's predictions and then return it.

```
def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.input], [model.get_layer(last_conv_layer_name).output, model.output]
    )

    # Then, we compute the gradient of the top predicted class for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    # This is the gradient of the output neuron (top predicted or chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
     # For visualization purpose, we will also normalize the heatmap between 0 & 1
     heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
     return heatmap.numpy()
```

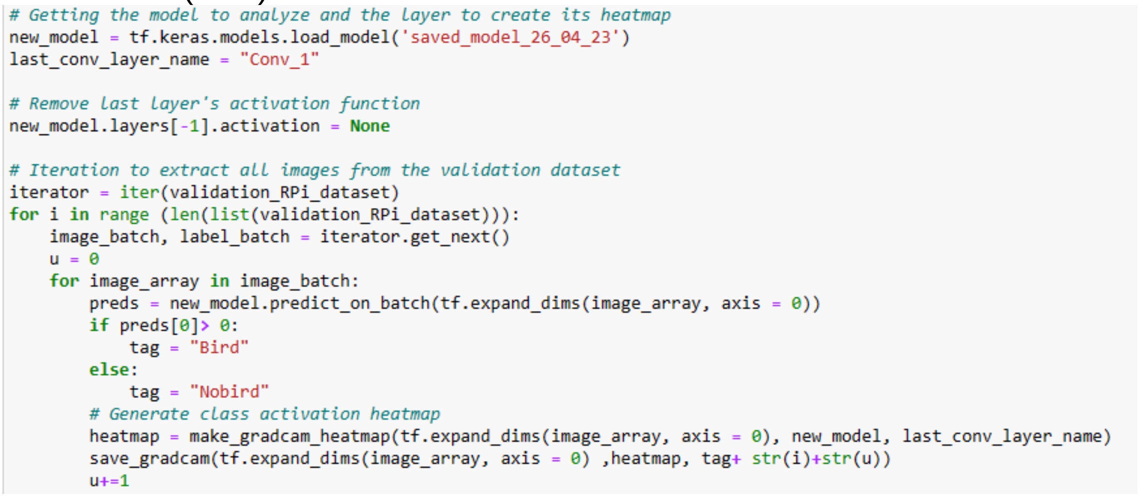
Fig. 6.17: Function that creates the heatmap

The save_gradcam function takes as input an image, the heatmap, the name that the resulting image should have and a transparency parameter alpha set to 0.4 to overlay the heatmap with the corresponding image. It then saves the resulting image in the GradCamImages folder in the root path of the project.

```
def save_gradcam(img_path, heatmap, cam_path, alpha=0.4):

    img = img_path[0,:,:]
    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # Use jet colormap to colorize heatmap
    jet = cm.get_cmap("jet")

    # Use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]

    # Create an image with RGB colorized heatmap
    jet_heatmap = tf.keras.preprocessing.image.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = tf.keras.preprocessing.image.img_to_array(jet_heatmap)

    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = tf.keras.preprocessing.image.array_to_img(superimposed_img)

    # Save the superimposed image
    name = ./GradCamImages/"+cam_path+".jpg
    superimposed_img.save(name)
```

Fig. 6.18: Function that overlaps and saves the heatmap and the image

All the necessary variables to create the Grad-Cam images are provided in this last block of code which gets the model, last convolution layer's name, image array and creates the name of the resulting overlaid image according to the class (bird or no-bird) and creates a unique identifier of the image according to its batch and number (0-31).



```
# Getting the model to analyze and the layer to create its heatmap
new_model = tf.keras.models.load_model('saved_model_26_04_23')
last_conv_layer_name = "Conv_1"

# Remove last layer's activation function
new_model.layers[-1].activation = None

# Iteration to extract all images from the validation dataset
iterator = iter(validation_RPi_dataset)
for i in range (len(list(validation_RPi_dataset))):
    image_batch, label_batch = iterator.get_next()
    u = 0
    for image_array in image_batch:
        preds = new_model.predict_on_batch(tf.expand_dims(image_array, axis = 0))
        if preds[0]> 0:
            tag = "Bird"
        else:
            tag = "Nobird"
        # Generate class activation heatmap
        heatmap = make_gradcam_heatmap(tf.expand_dims(image_array, axis = 0), new_model, last_conv_layer_name)
        save_gradcam(tf.expand_dims(image_array, axis = 0) ,heatmap, tag+ str(i)+str(u))
        u+=1
```

Fig. 6.19: Code to execute

These images will allow us to see what characteristics of an image the model takes into account when making a decision about the class it belongs to. This information may be used to improve the dataset if needed by identifying and addressing any biases or shortcomings in the data.

Fig. 6.20 displays a random sample of 12 images from the validation dataset, all of which are labeled as 'Bird'. The corresponding heatmaps, overlaid on the original images, highlight regions in red where birds are detected by the final convolutional layer 'Conv_1'.

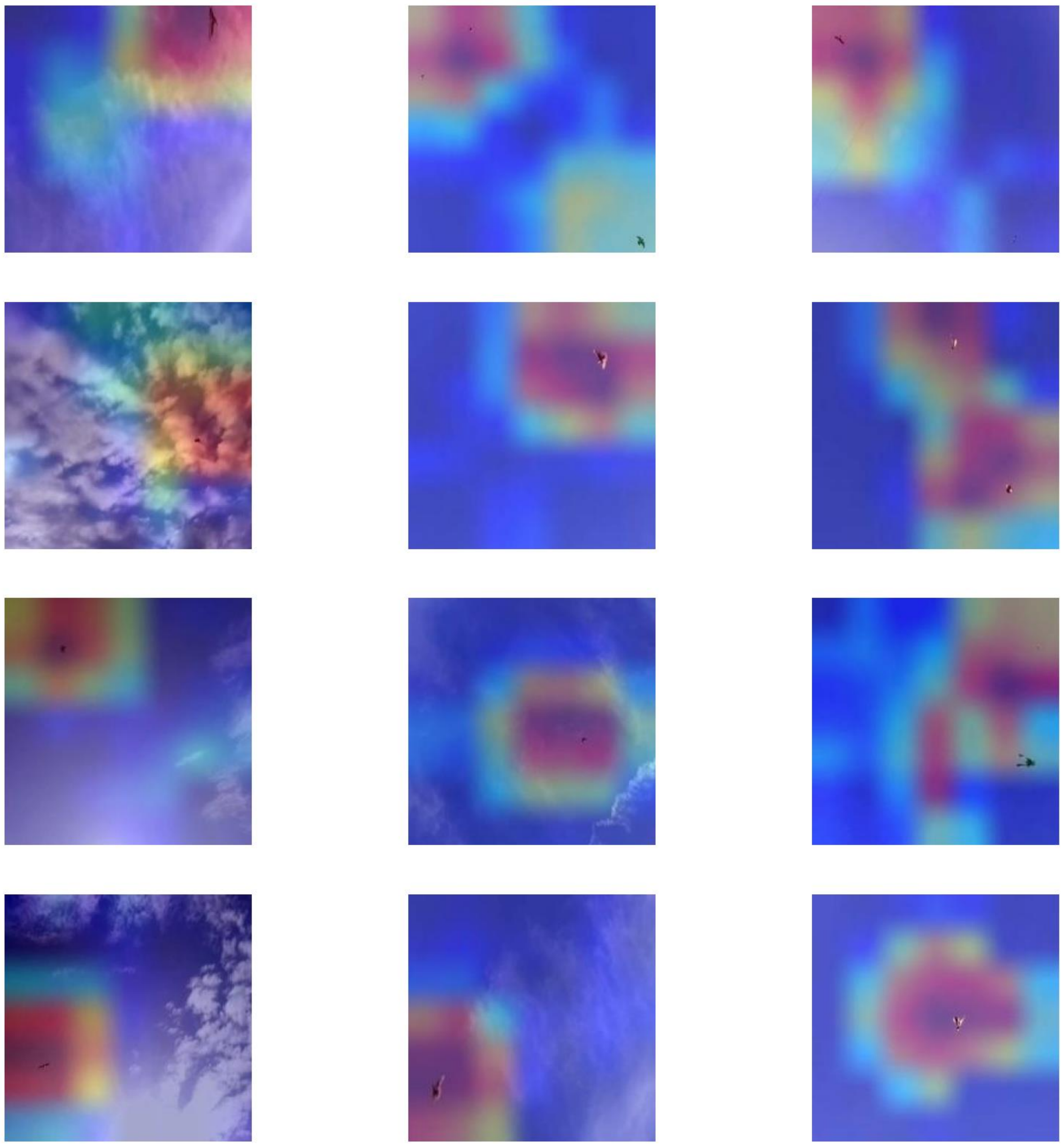


Fig. 6.20: Bird detections in validation dataset

Fig. 6.21 displays a random sample of 12 images from the validation dataset, all of which are labeled as 'No Bird' and may depict drones, aircraft, or plain sky. The corresponding heat maps generated by the Grad-CAM algorithm do not highlight any objects resembling birds.

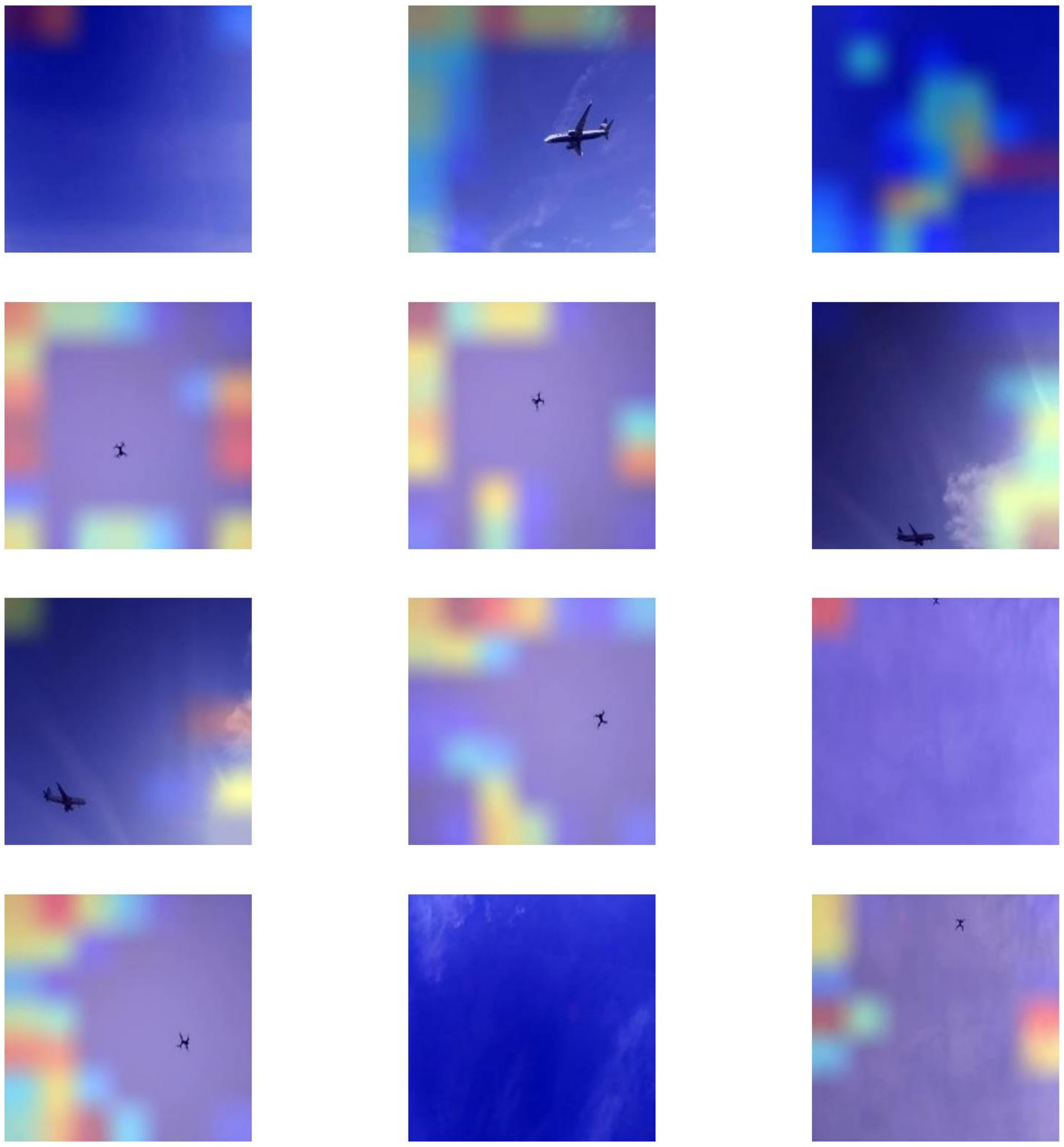
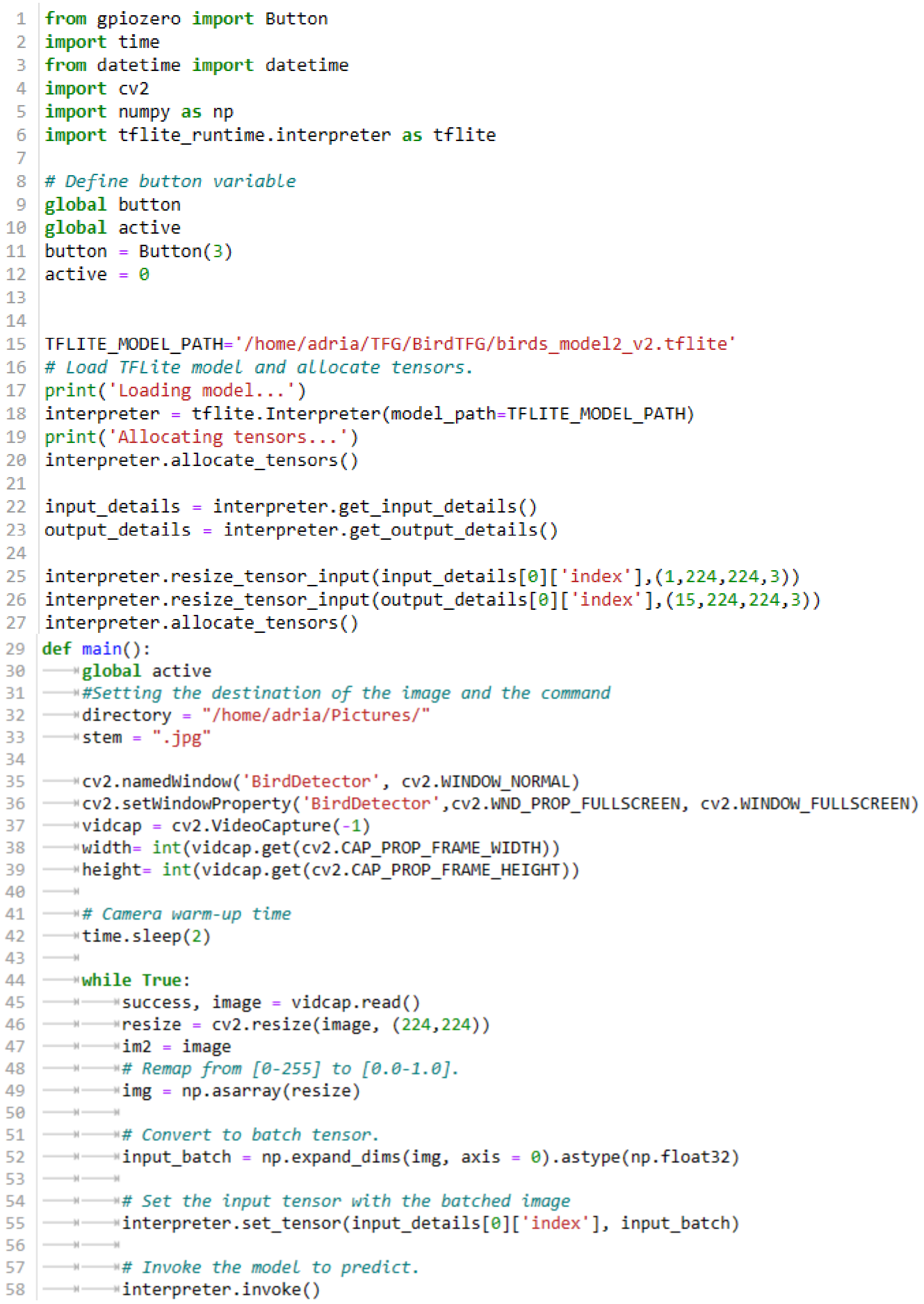


Fig. 6.21: No-Bird detections in validation dataset

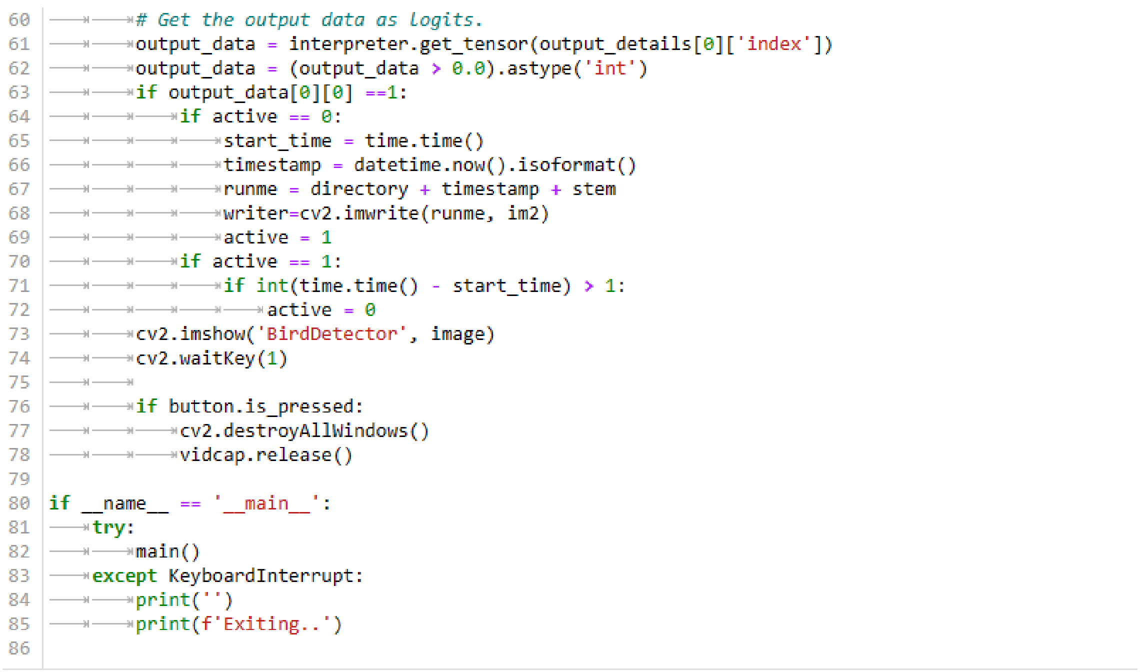
6.4 Real world results

To evaluate the model's performance under real-world conditions, a Python script was developed and integrated into the prototype. This script captures and stores an image whenever a bird is detected. The resulting analysis of these images enables the calculation of true positive and false positive detection rates, providing a quantitative measure of the model's performance in a test run.



```python
from gpiozero import Button
import time
from datetime import datetime
import cv2
import numpy as np
import tflite_runtime.interpreter as tflite

# Define button variable
global button
global active
button = Button(3)
active = 0


TFLITE_MODEL_PATH='/home/adria/TFG/BirdTFG/birds_model2_v2.tflite'
# Load TFLite model and allocate tensors.
print('Loading model...')
interpreter = tflite.Interpreter(model_path=TFLITE_MODEL_PATH)
print('Allocating tensors...')
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

interpreter.resize_tensor_input(input_details[0]['index'],(1,224,224,3))
interpreter.resize_tensor_input(output_details[0]['index'],(15,224,224,3))
interpreter.allocate_tensors()
def main():
    global active
    #Setting the destination of the image and the command
    directory = "/home/adria/Pictures/"
    stem = ".jpg"

    cv2.namedWindow('BirdDetector', cv2.WINDOW_NORMAL)
    cv2.setWindowProperty('BirdDetector',cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
    vidcap = cv2.VideoCapture(-1)
    width= int(vidcap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height= int(vidcap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    
    # Camera warm-up time
    time.sleep(2)
    
    while True:
        success, image = vidcap.read()
        resize = cv2.resize(image, (224,224))
        im2 = image
        # Remap from [0-255] to [0.0-1.0].
        img = np.asarray(resize)
        
        # Convert to batch tensor.
        input_batch = np.expand_dims(img, axis = 0).astype(np.float32)
        
        # Set the input tensor with the batched image
        interpreter.set_tensor(input_details[0]['index'], input_batch)
        
        # Invoke the model to predict.
        interpreter.invoke()
```

Fig. 6.22: Extract of the code to detect birds



```
        # Get the output data as logits.
        output_data = interpreter.get_tensor(output_details[0]['index'])
        output_data = (output_data > 0.0).astype('int')
        if output_data[0][0] ==1:
            if active == 0:
                start_time = time.time()
                timestamp = datetime.now().isoformat()
                runme = directory + timestamp + stem
                writer=cv2.imwrite(runme, im2)
                active = 1
            if active == 1:
                if int(time.time() - start_time) > 1:
                    active = 0
        cv2.imshow('BirdDetector', image)
        cv2.waitKey(1)
        
        if button.is_pressed:
            cv2.destroyAllWindows()
            vidcap.release()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('')
        print(f'Exiting..')

```

Fig. 6.23: Final part of the code to detect birds

The code architecture is analogous to that described in 'Chapter 5. Dataset for Computer Vision Applications', which details the process of saving recordings of birds, drones, and planes for dataset creation. Figures 6.22 and 6.23 illustrate a script that captures images from a camera feed and stores those containing detected birds. To prevent redundant detection of the same bird, the script incorporates a one-second delay between successive image captures. The program terminates upon user input via a button press.

This script facilitates the execution of real-world tests which were performed during optimal visibility conditions to delimit the casuistic for potential flaws. Images that were captured as a result of a trigger event were stored locally on the Raspberry Pi device. The analysis of these images enables the identification of the flaws and enable a first evaluation of the overall performance.

While the model demonstrated a high degree of accuracy in classifying birds, there were several instances of false positives among the images collected by the device. A selection of representative images is included below in this document to illustrate these cases, accompanied by a possible explanation (as deep learning models deal with probabilities) of the underlying factors contributing to the occurrence of false positives.

In a test performed during a windy day, several instances of plastic bags, shopping gloves and other light plastic items were found among the images that triggered the script. The most probable explanation for this is that the model does not understand what this type of objects are because the dataset does not contain images like these. See Fig. 6.24, Fig. 6.25, Fig. 6.26 and Fig. 6.27,

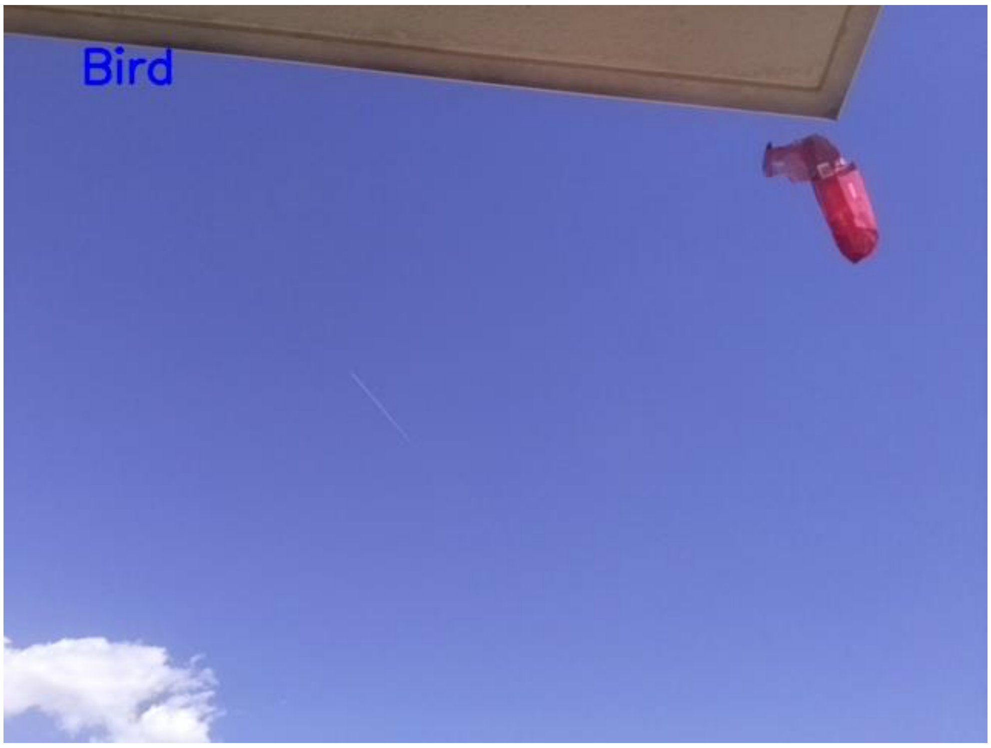


Fig. 6.24: Red paper bag classified as bird

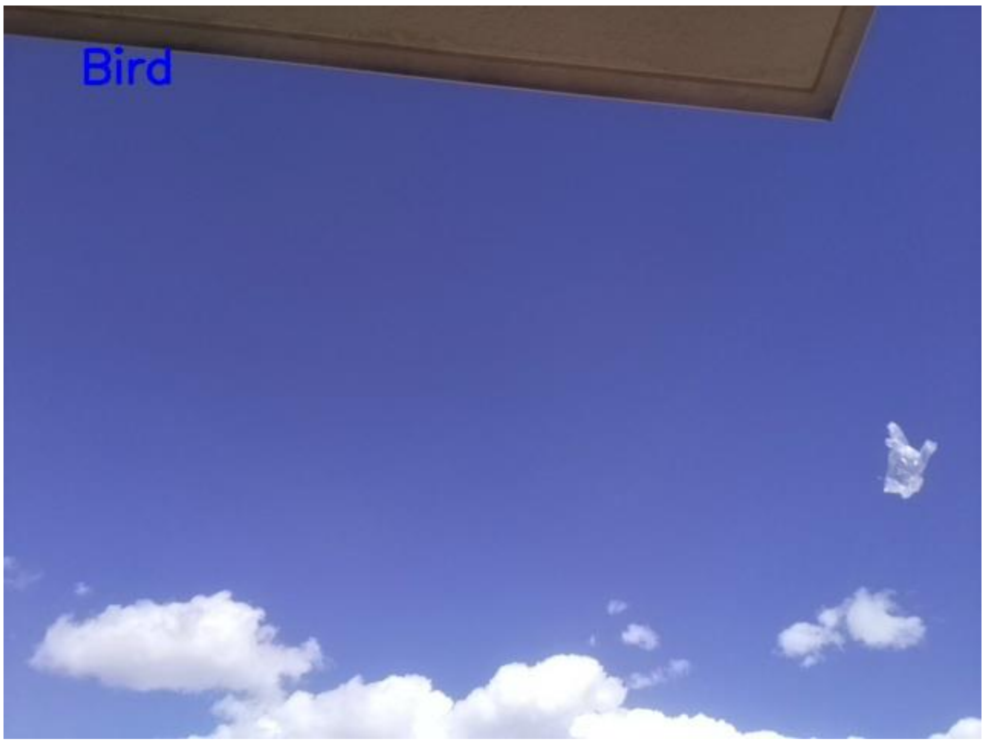


Fig. 6.25: Plastic glove classified as bird

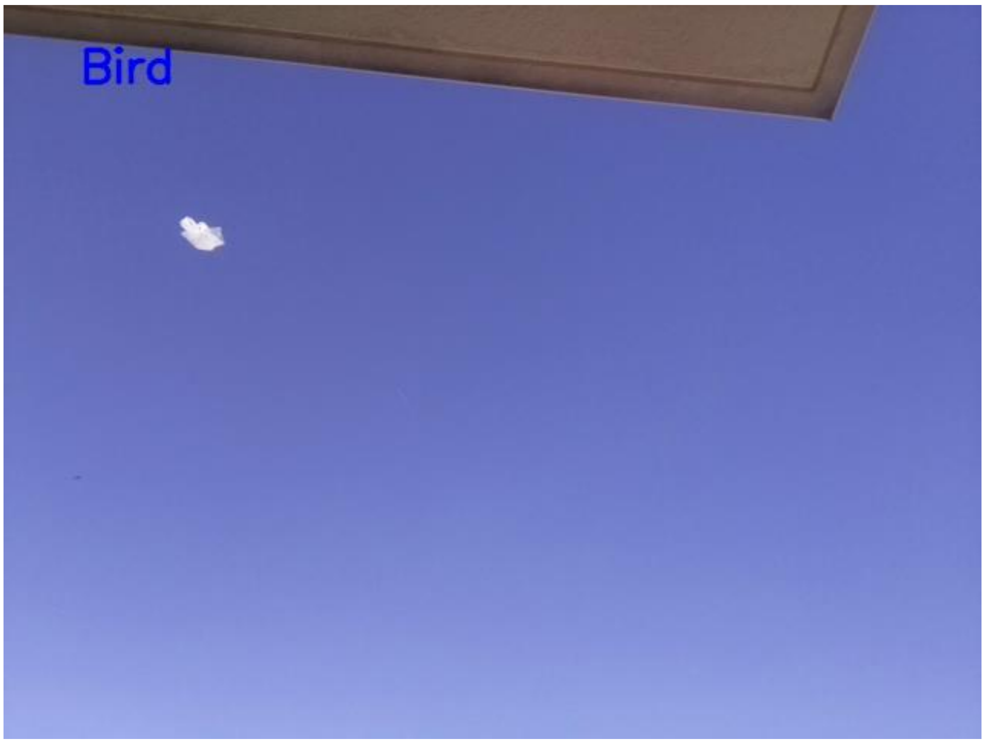


Fig. 6.26: Plastic bag classified as bird

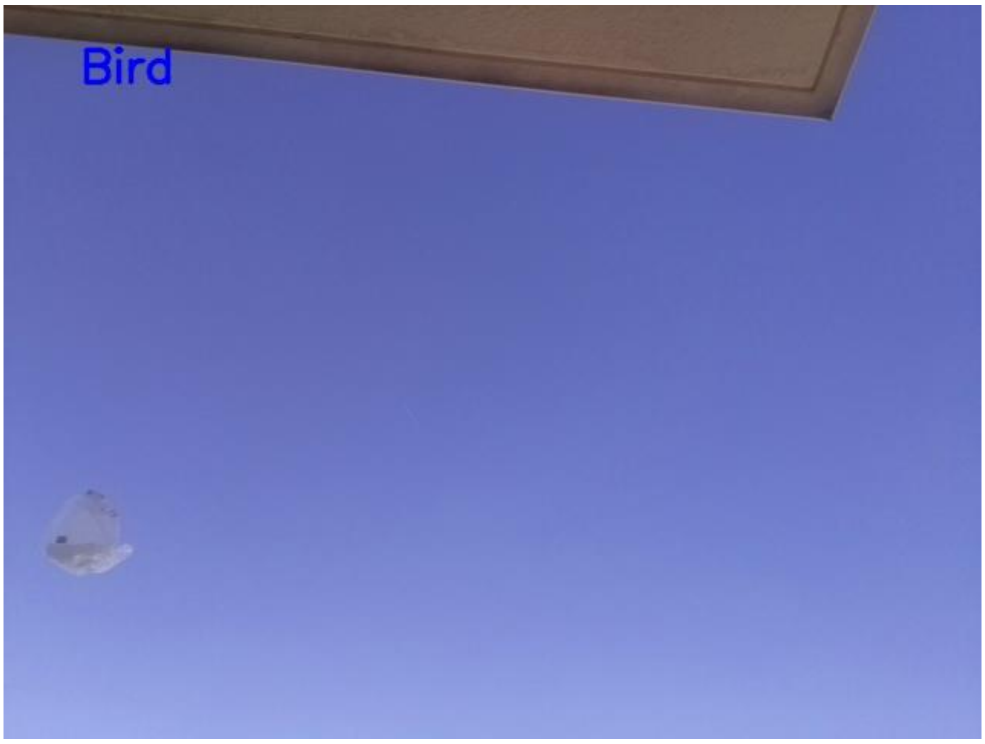


Fig. 6.27: Plastic bag classified as bird

The model erroneously identified several instances of contrails as birds, resulting in false positives. These false positives are particularly concerning because contrails remain visible in the camera's field of view for longer periods than birds, leading to consecutive triggers and an increased number of saved images containing contrails. Images containing contrails were added into the dataset but nevertheless some of them were still miss classified by the model. See Fig. 6.28 and Fig. 6.29.

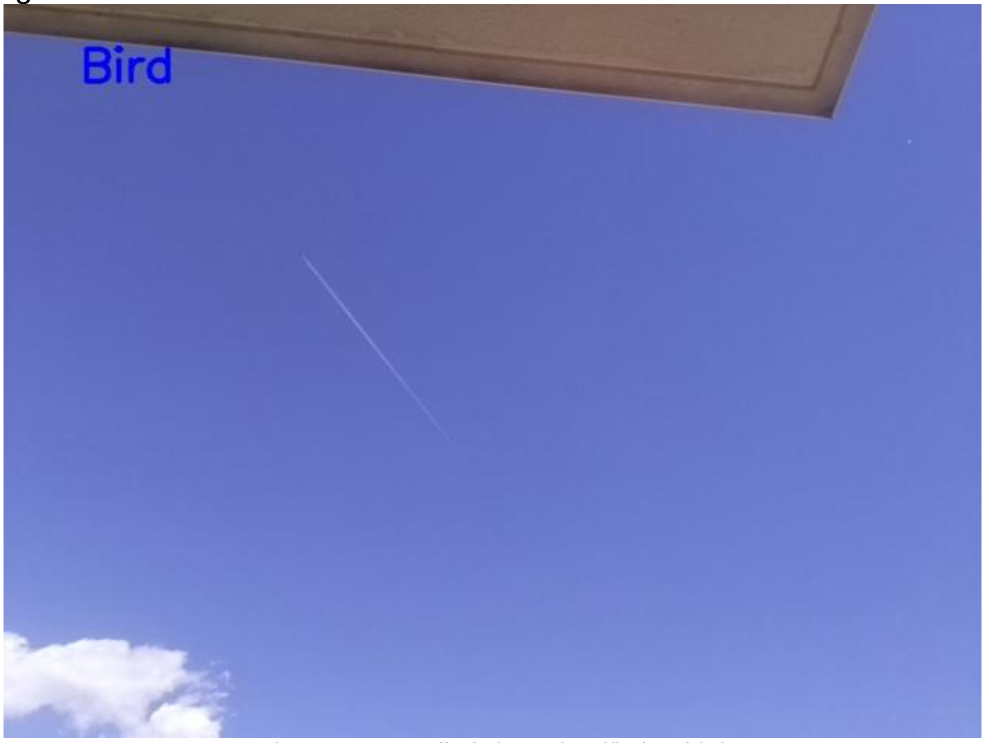


Fig. 6.28: Contrail of plane classified as bird

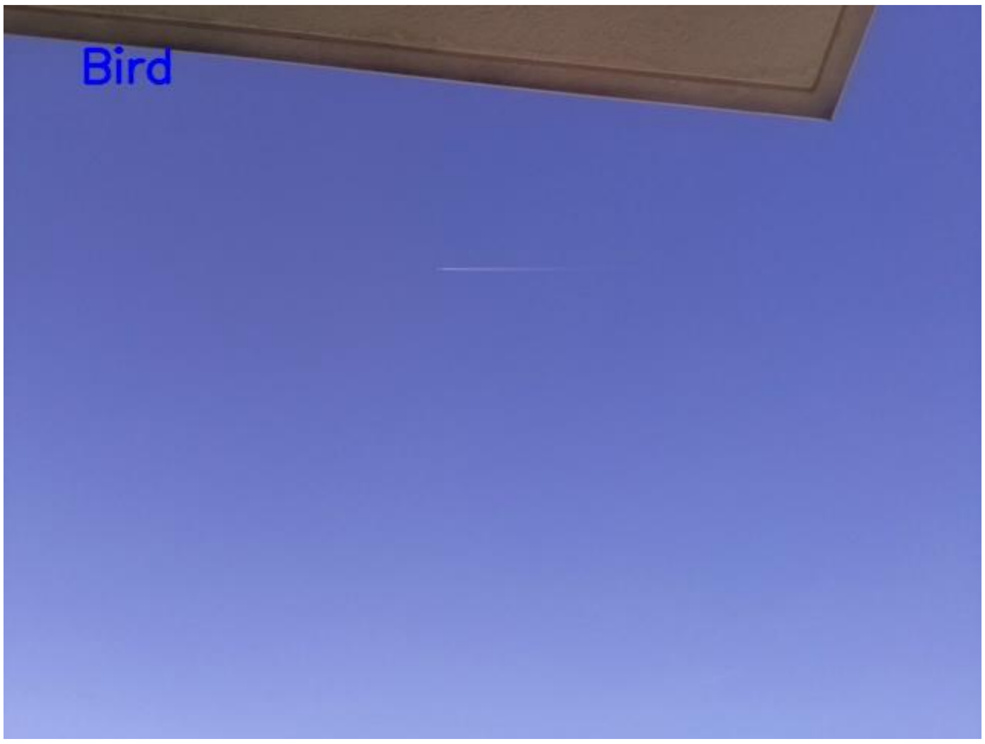


Fig. 6.29: Contrail of plane classified as bird

The model, in less frequency than plane contrails also miss-classified some small and scattered clouds as birds. The model has to be flexible enough to classify birds in species, sizes and positions different than those fed during the training phase as otherwise it will overfit and false negatives would increase. See Fig. 6.30 and Fig. 6.31.

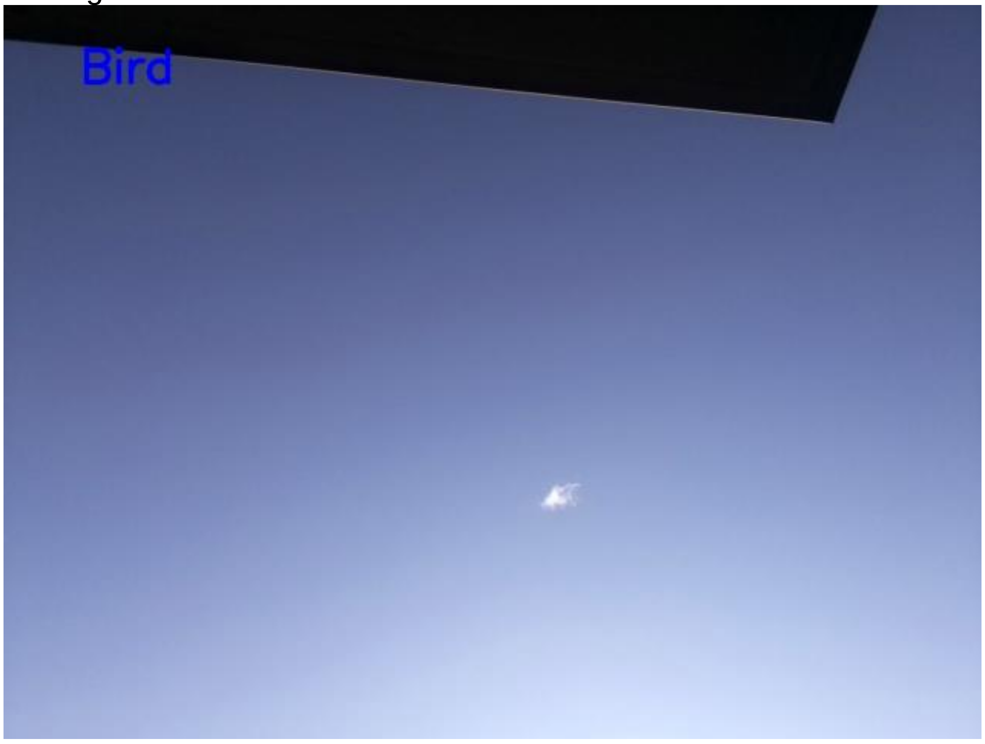


Fig. 6.30: Small cloud classified as bird

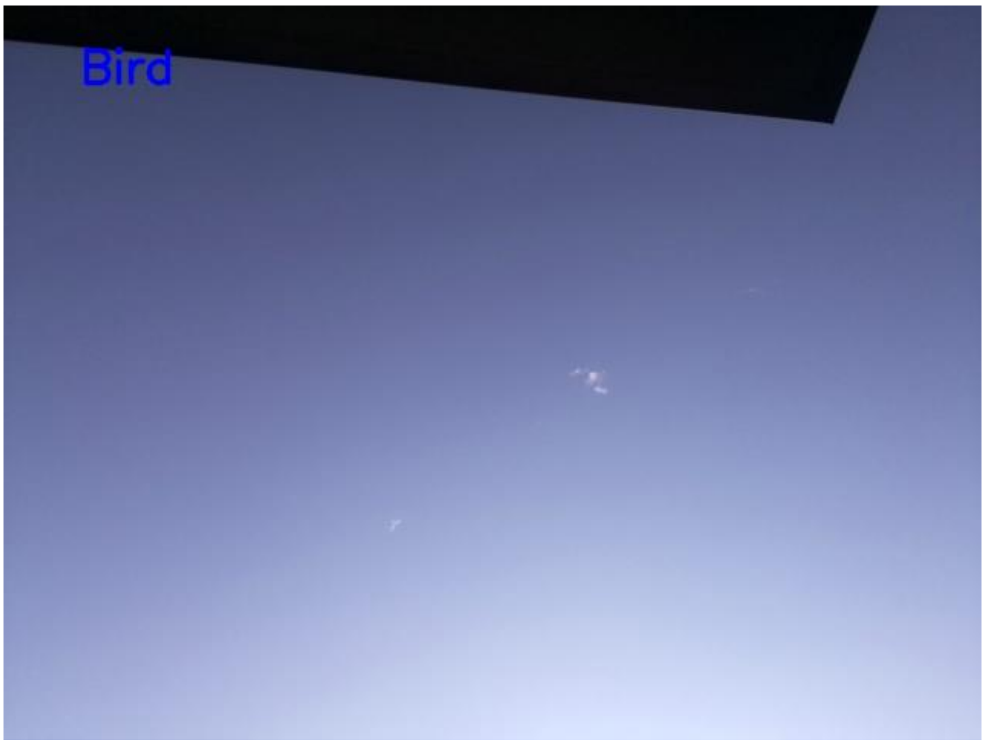


Fig. 6.31: Small cloud classified as bird

Most of the birds classified correctly by the model correspond to far away, small birds, some of them were barely visible to the naked eye. A few examples of these situations are presented below in Fig. 6.32, Fig. 6.33 and Fig. 6.34.

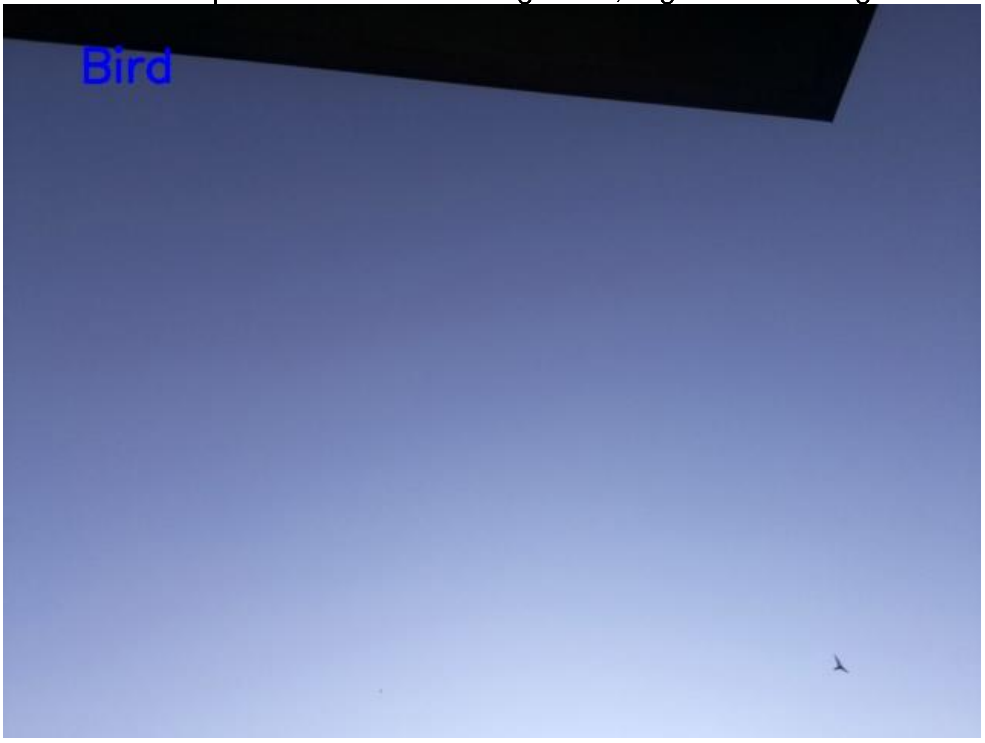


Fig. 6.32: Bird in the bottom right corner of the image

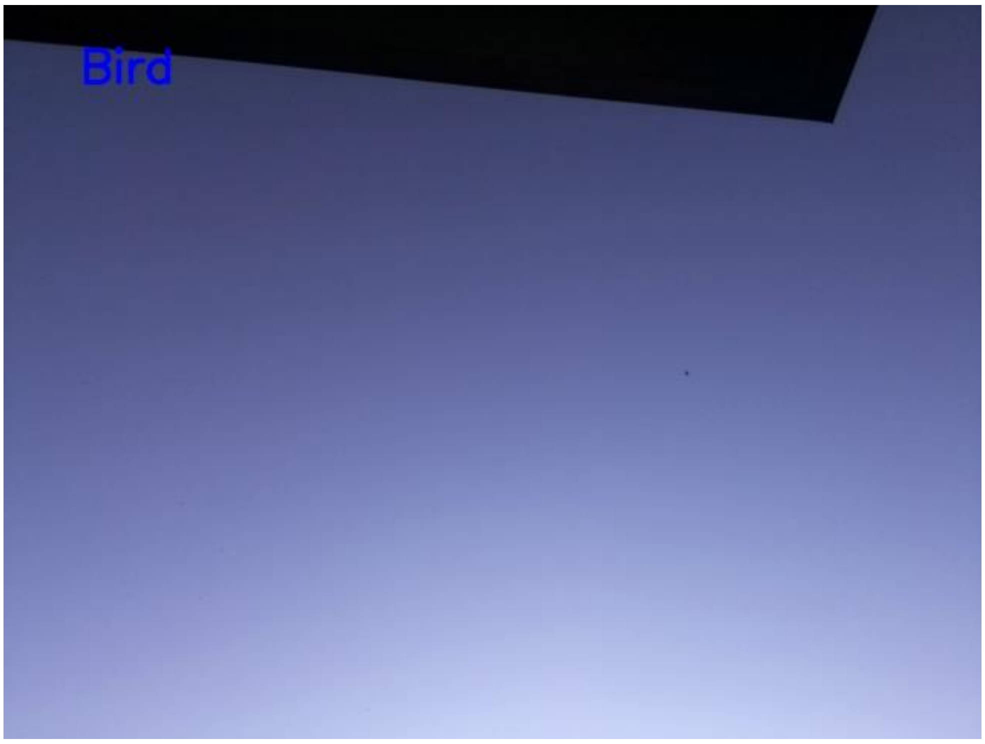


Fig. 6.33: Bird in the middle right of the image

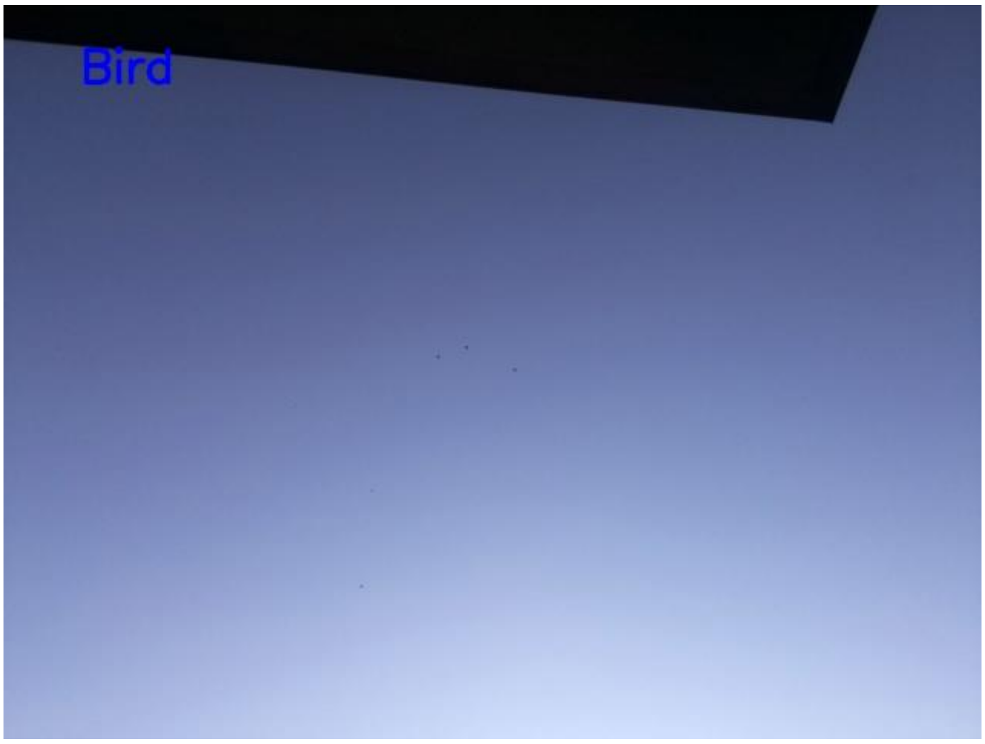


Fig. 6.34: Cluster of small birds in the middle of the image

Some instances of birds closer to the camera were also detected. It has to be noted that different sun positions and contrast did not have substantial effect on detection. See Fig. 6.35, Fig. 6.36 and Fig. 6.37.

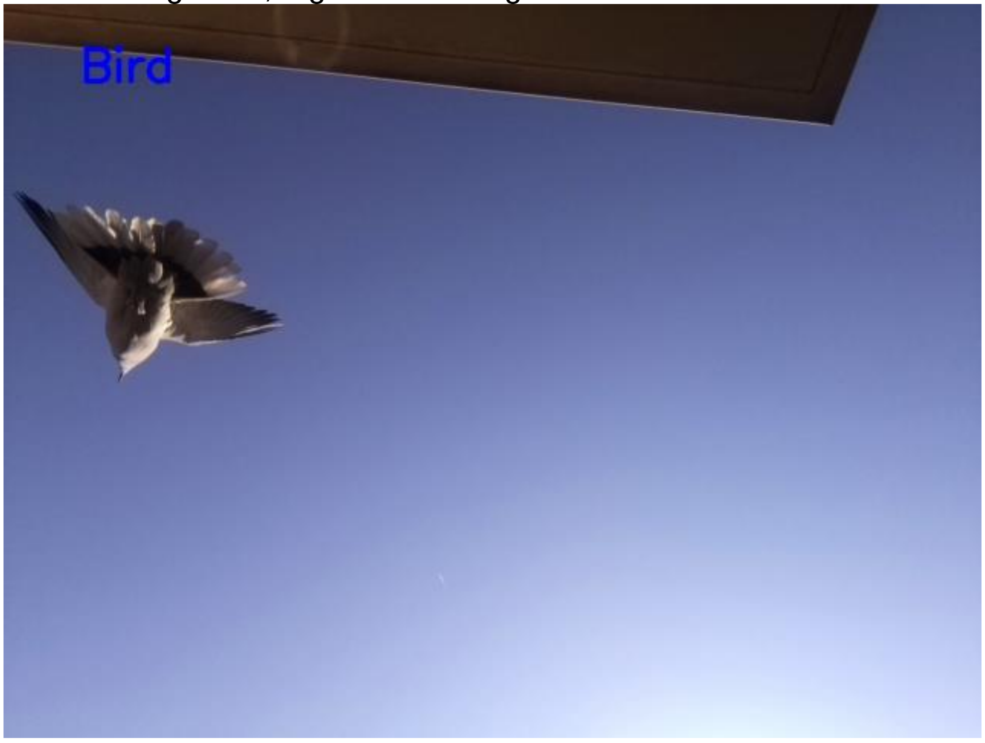


Fig. 6.35: Pigeon in mid-flight in the middle left part of the image

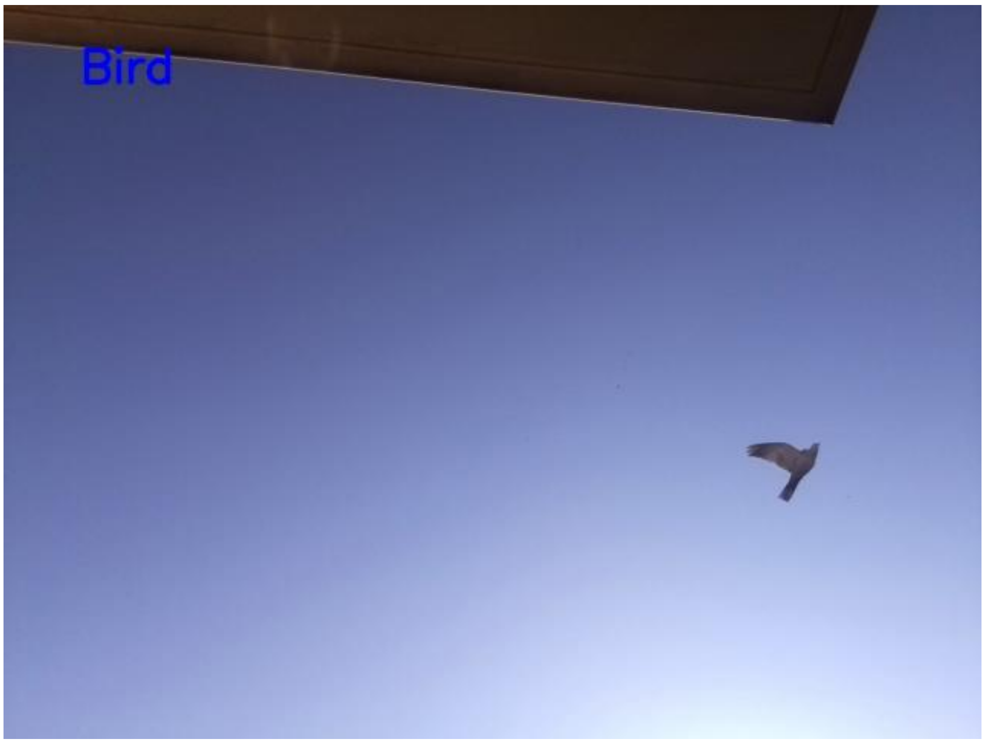


Fig. 6.36: Bird in mid-flight in the middle right part of the image

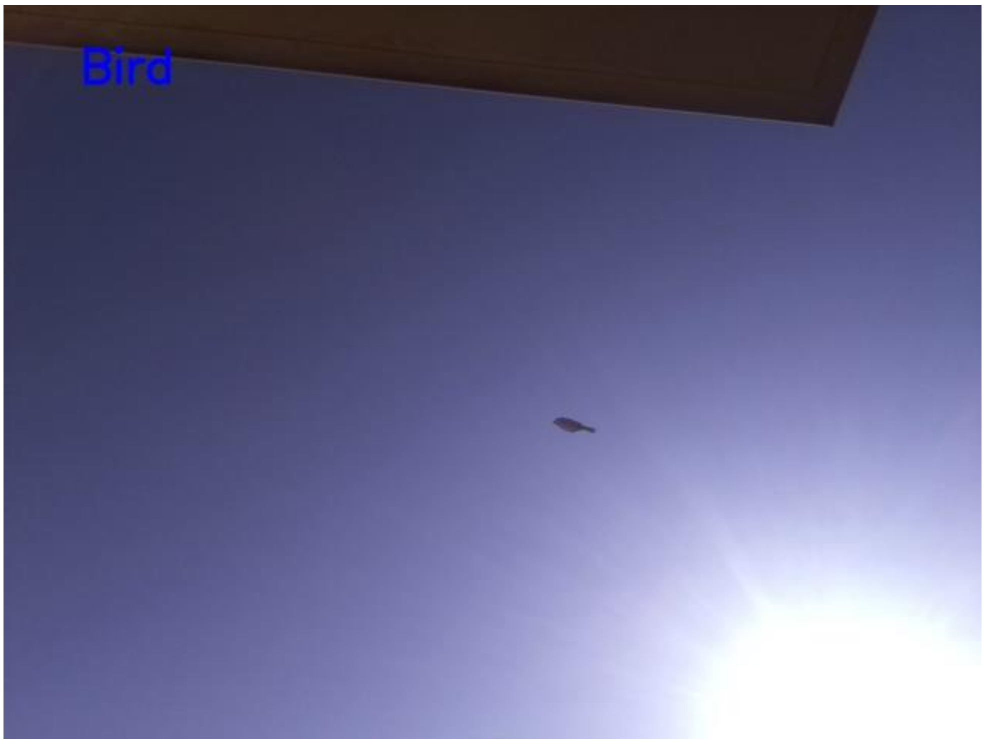


Fig. 6.37: Small bird in mid-flight in the middle of the image

A final test, in this case using a script capturing video showed that on some occasions, a bird was correctly classified in a frame but some frames later, the same bird could not be detected by the model permanently or for a few frames. As previously explained, deep learning models classified based on probability and it could happen that in certain frames the prediction fails to reach the threshold defined. Fortunately, in most instances this is not the case. See Fig. 6.38 and Fig. 6.39.

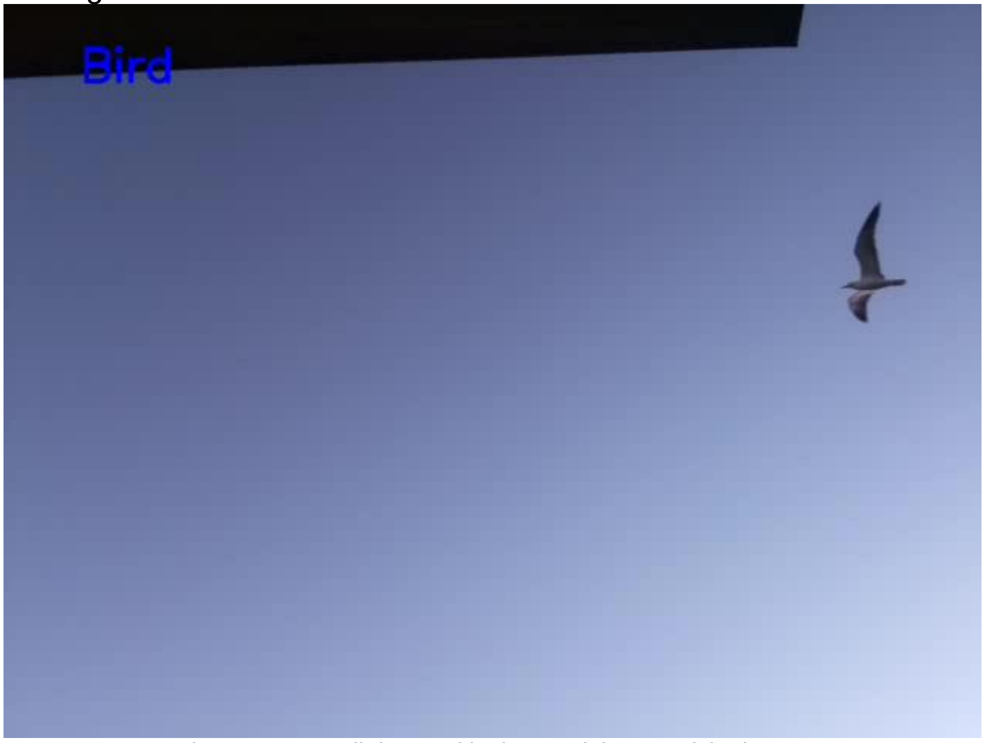


Fig. 6.38: Seagull detected in the top right part of the image

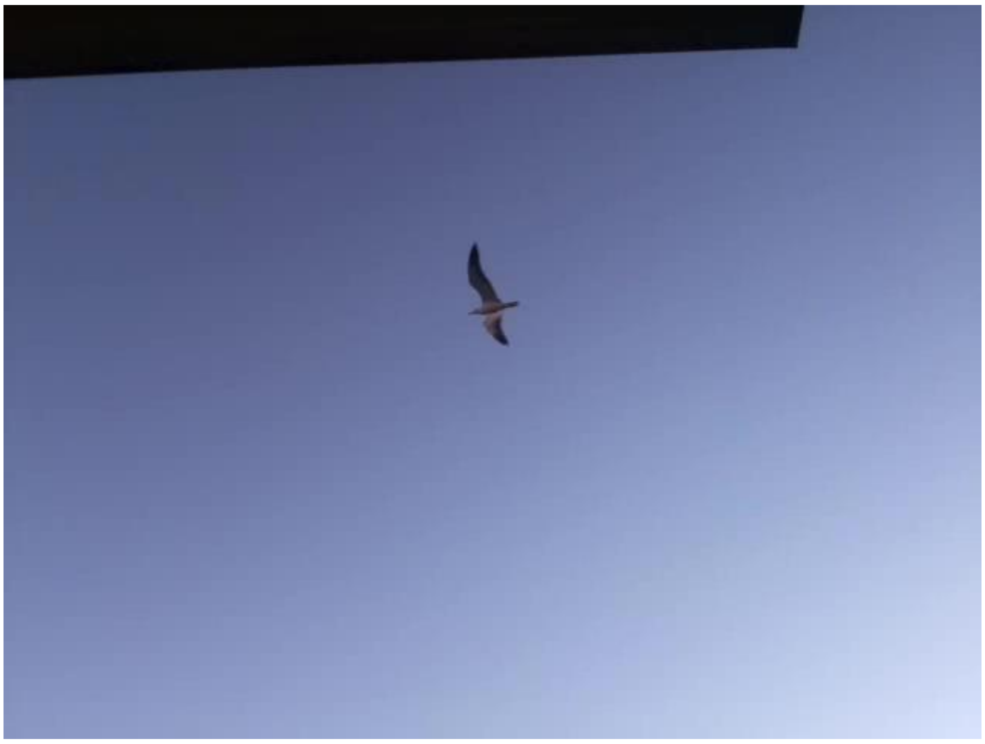


Fig. 6.39: Seagull not detected in the middle of the image

On April 27th, a test was conducted on the prototype from 7:00 AM to 5:00 PM. The camera was positioned to face the sky in an unobstructed location and was mounted on a table using plastic cable ties for stabilization, as depicted in Fig. 6.40.

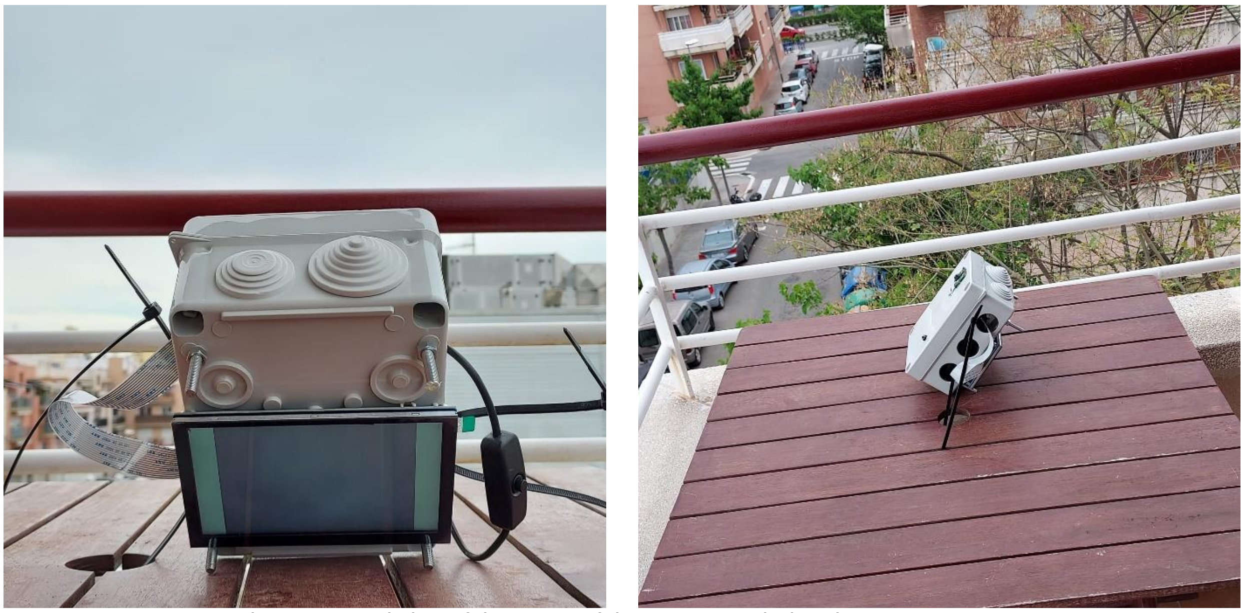


Fig. 6.40: Depiction of the setup of the prototype during the test

In that time the prototype stored 771 images that were later human reviewed, the resulting distribution is as follows:

- 3 False positives due to accidental triggering during the set up.
- 109 False positives.
- 659 True positives.

The images were then processed using the Grad-CAM algorithm to determine the features and characteristics that the model took into account to classify an image as containing a bird. This information is valuable for assessing the completeness of the dataset and identifying any additional image classes with distinct features that the dataset might be lacking in order to reduce the incidence of false positives. Figure 6.41 displays 12 randomly selected true positive classifications. The model demonstrates the ability to detect birds at significant distances, even when they are barely visible.

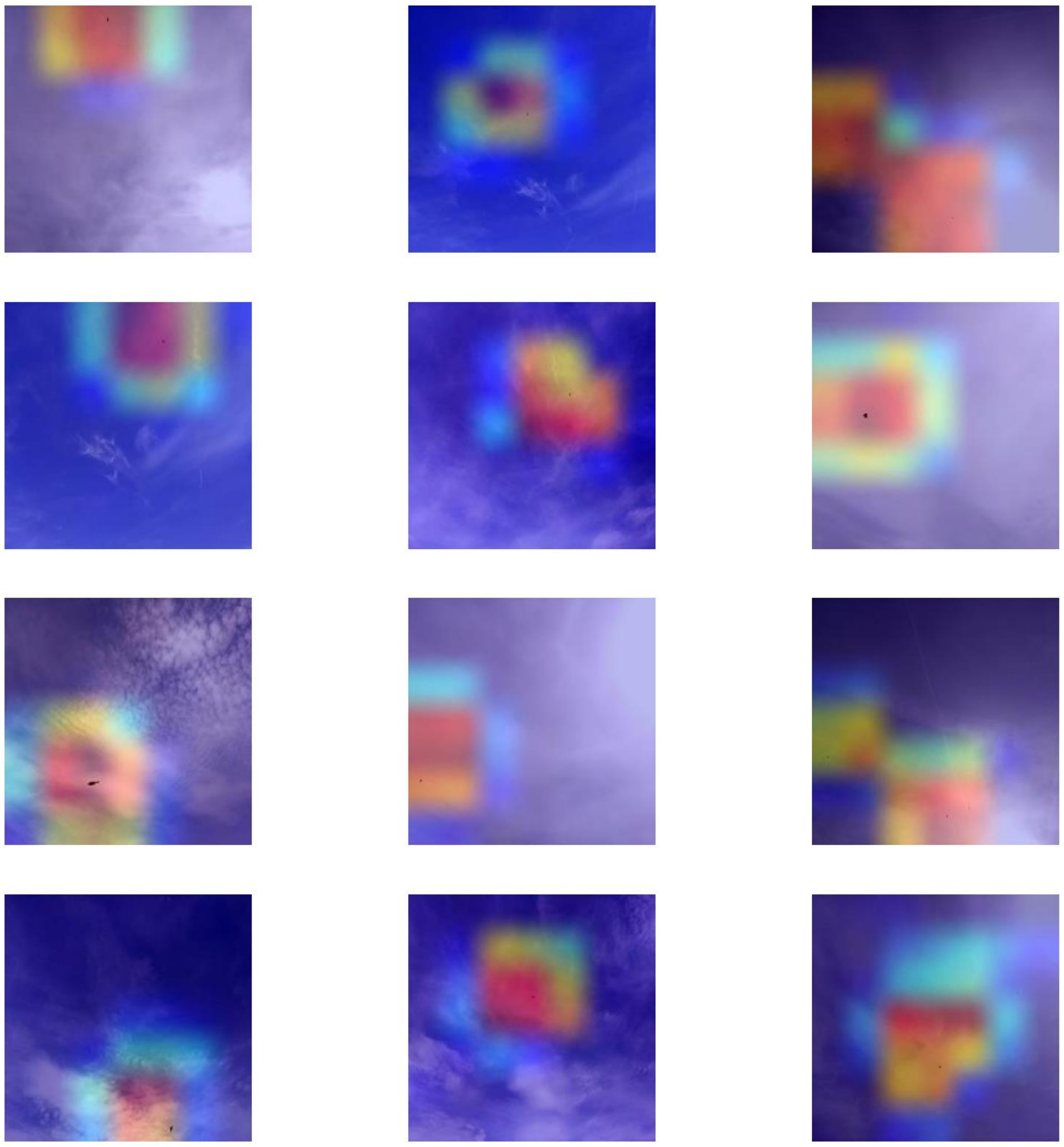


Fig. 6.41: True positives detected by the model

Fig. 6.42 illustrates 12 randomly selected images that demonstrate instances where the model incorrectly identifies aircraft contrails as birds, resulting in false positives. One potential solution to mitigate this issue could be to augment the training dataset with additional samples of skies depicting a wide range of contrail patterns.

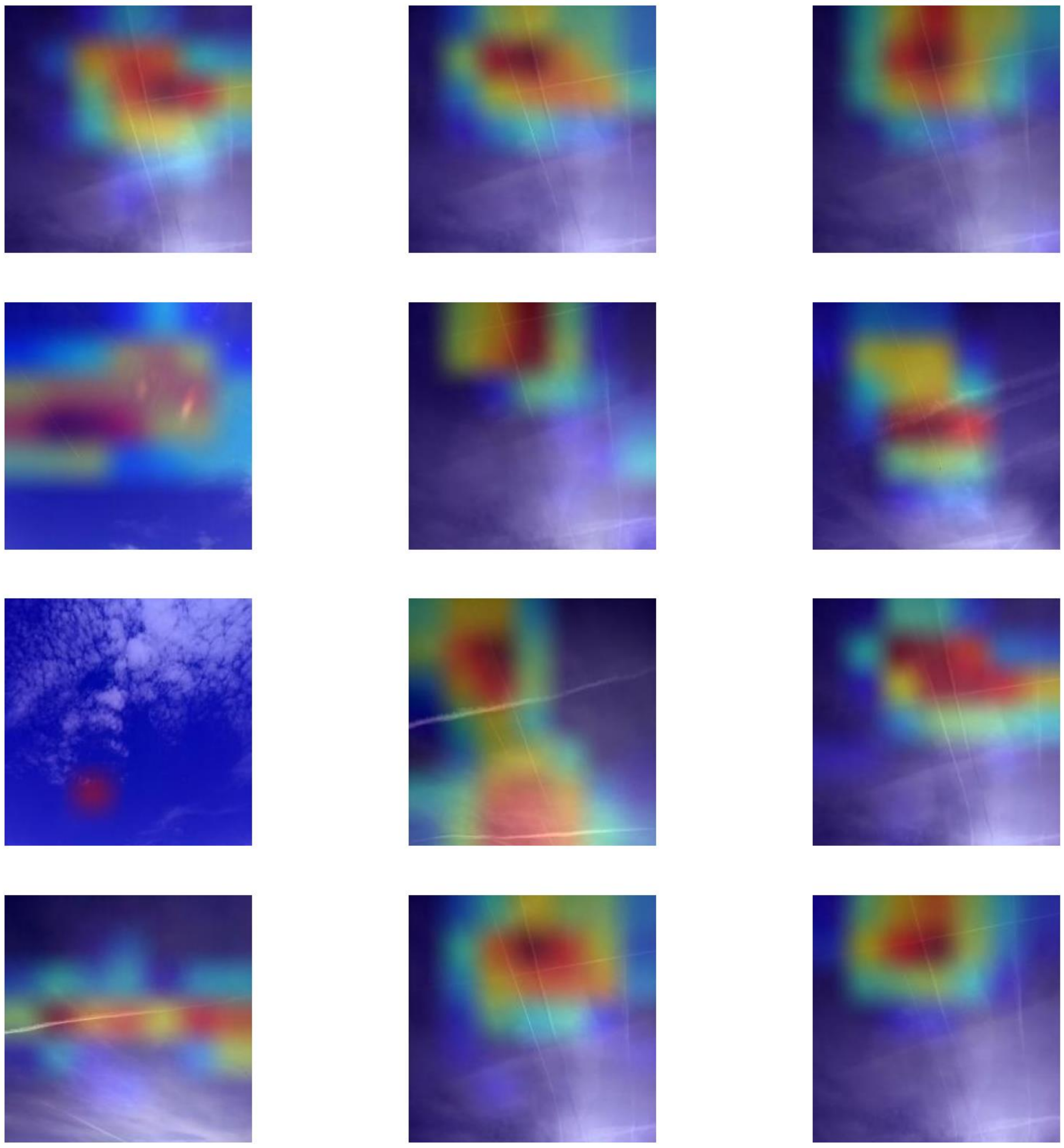


Fig. 6.42: False positives detected by the model

Per definition, excluding the detections during the setup, this means the program has a precision of:

$$precision = \frac{TP}{TP+FP} = \frac{659}{659+109} = 0.85807 \quad (6.5)$$

Unfortunately, this test does not take into account the number of false negatives and true negatives. False negatives refer to instances where birds enter the camera’s field of vision but fail to trigger the program. True negatives are frames that do not contain any birds. A significant number of false positives in the test were attributed to aircraft contrails. Although the one-second interval between saved images effectively reduces the likelihood of detecting the same bird twice, as birds typically exit the frame before this time elapses, contrails persist for longer periods and continue to trigger the program.

Chapter 7. Conclusion, improvements and application

Project conclusions

The aim of this project was to develop a computer vision system capable of acknowledging the presence of birds in order to prevent bird strikes in aviation. To achieve this goal, a deep learning model was developed using transfer learning with fine tuning. This model was designed to be flexible enough to detect birds in images that differed from those in the training dataset while also being restrictive enough to avoid triggering too easily.

In addition to the development of the deep learning model, a dataset was created and a microprocessor, specifically a RaspberryPi, was selected to execute the model and capture images using its legacy camera. The resulting prototype has shown promising results in both validation metrics and real-world testing, albeit with limitations. The effectiveness of the system depends on the optics used. A more complete dataset is required for the system to function optimally. The management of false positives is a challenge that must be taken into account when setting up a system based on this technology. An A/B test should be conducted to evaluate the performance of the system.

The potential impact of this prototype on the air transportation industry is significant. By providing pilots with more frequent and accurate notifications of the presence of birds in the vicinity of airports, particularly during landing and take-off maneuvers where the majority of bird strikes occur, the likelihood of such incidents can be reduced. Further study is needed to fully understand the effects of these notifications on pilot behavior and decision-making.

In contrast to radar-based solutions, the implementation of an artificial intelligence approach offers greater scalability and can be utilized to fortify critical points with increased precision due to its localized focus on specific regions. The cost-effectiveness of this solution is also noteworthy. Plugging the device to a power-source and using an inferior model of the Raspberry Pi could drop its cost under 100 euros (as per July 2023), these devices could be strategically placed around airports to provide real-time information on bird presence to a central server. The sensitivity of the alert trigger could be adjusted as needed through the use of appropriate training data. Additionally, this device could be used in recreational flights by pointing it out of the cockpit window and sounding an alarm if the presence of birds is detected.

Overall, this project has demonstrated the potential for computer vision technology to play a valuable role in improving aviation safety that could potentially reduce the probability of suffering bird strikes.

Improvements

- As per the prototype, enhancements could be made to both the design and safety of the casing. Prolonged exposure to weather conditions may result in damage to the electronic components due to humidity ingress, as the current design has unnecessary openings. A solution could be to design and fabricate a new casing using 3D printing technology. In order to progress from the prototyping stage, the device should be tested for pressure, temperature, fire resisting, electrical safety and humidity requirements. A good way to ensure that is to certify the product according to regulators like UL (in the United States) or CE (in the European Union).

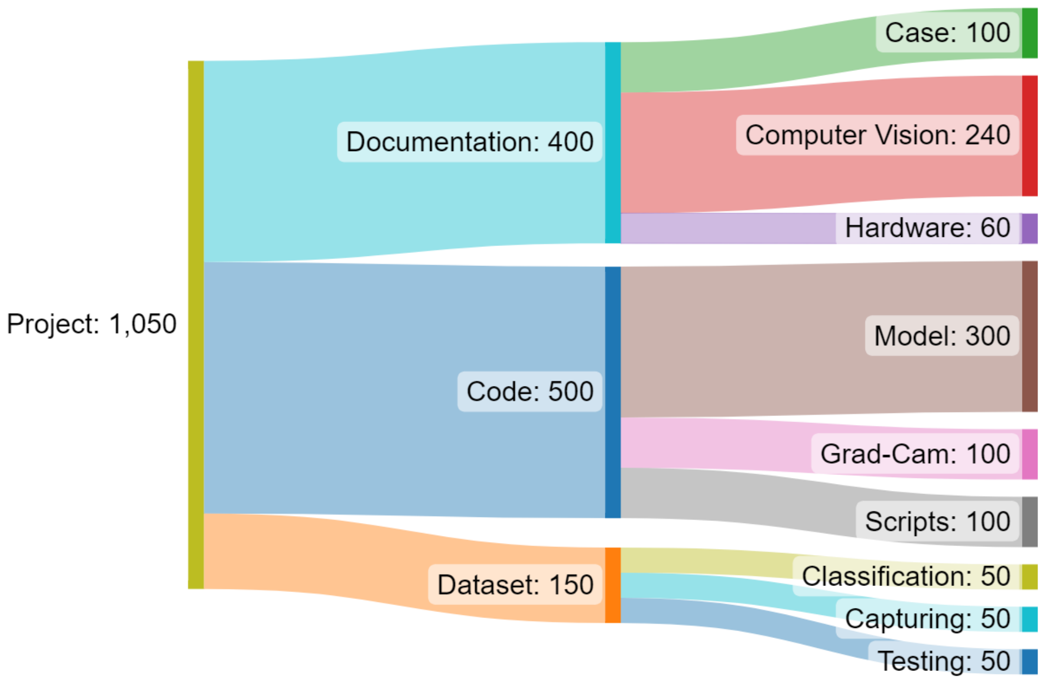
- The Raspberry Pi can be configured to connect to the Internet. If enabled, with a few changes in the script, the model could store images in an external database. Not only could the model be monitored for performance purposes but those images could be repurposed to add the birds and no-birds datasets with site-specific images.

- If installed on-board, an acoustic actuator could be used to trigger an alarm to the pilot.

Individual development

Despite having acquired basic concepts of Python programming and image processing in some subjects both for the Aerospace systems and Telecommunications' degree, along the development of this project I have learned a more in-depth knowledge of both the tooling and coding required to complete a project of these characteristics.

This project is the result of hundreds of extensive research about technologies that are rapidly evolving in the present and numerous hours of testing to ensure the device is able to identify birds with a reasonable precision. While the information available to the public on subjects such as deep learning and computer vision are plentiful and accessible to the public, it was involved a significant amount of trial and error when implementing different strategies. I had to deal with deprecated libraries and lots of debugging to adapt the source information and understand concepts like tensors, convolutional blocks, statistics and data representation. The development of a unique and original binary classification approach required the creation of numerous scripts for data representation, organization, editing, and classification of images for the dataset.



Flow chart representing the time distribution of hours per task and subtask

Overall, I am satisfied with the final result of this project. With its completion, the reader of this document will find a comprehensive guide that progresses from basic concepts to more technical material, culminating in the application of this knowledge to a specific use case: the development of a binary classifier capable of distinguishing birds. With this knowledge, the reader may approach other problems in other fields. Deep learning based computer vision solutions, with enough creativity for example can be used also to classify sounds as long as their characteristics can be represented with colors.

In the present and the near future, artificial intelligence will be playing a key role in the development of new tools and technology in both subjects studied at university. In my opinion my education would not be complete if I did not acquire the knowledge learned during the completion of the project as this technology will revolutionize both the telecommunications and aerospace fields among many others.

REFERENCES

[1] Federal Aviation Administration. Wildlife Hazard Mitigation. [Online] Available: https://wildlife.faa.gov/home [Accessed: 8 Jan. 2023].

[2] Skybrary. Bird Strike Reporting. [Online] Available: https://skybrary.aero/articles/bird-strike-reporting [Accessed: 21 May. 2023].

[3] Skybrary. Detection of Bird Activity Using Radar. [Online] Available: https://skybrary.aero/articles/detection-bird-activity-using-radar [Accessed: 21 May. 2023].

[4] IBM. Computer Vision. IBM Topics. [Online] Available: https://www.ibm.com/topics/computer-vision [Accessed 29 Mar. 2023]".

[5] Technological Institute of Education of Crete (TeiCrete). Edges. [PDF] Retrieved March 29, 2023, from https://eclass.teicrete.gr/modules/document/file.php/TP283/Lab/04.%20Lab/Edges.pdf".

[6] Sindhu V, Nivedha S, Prakash M (2020). "An Empirical Science Research on Bioinformatics in Machine Learning". *Journal of Mechanics of Continua and Mathematical Sciences*(7)

[7] DeepAI. Convolutional Neural Network. DeepAI Glossary. [Online] Available: https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network [Accessed 29 Mar. 2023].

[8] Gad, A. F. (2018). *Practical computer vision applications using deep learning with CNNs: With detailed examples in Python using TensorFlow and Kivy*. Apress, Berkeley, CA

[9] DevelopersBreach. CNN Banner. [Online] Available: https://i0.wp.com/developersbreach.com/wp-content/uploads/2020/08/cnn_banner.png [Accessed 17 Apr. 2023].

[10] Niall O' Mahony et al. Deep Learning vs. Traditional Computer Vision. arXiv. [Online] Available: https://arxiv.org/abs/1910.137962 [Accessed 03 Jun. 2023]

[11] DeepLizard. Neural Network Programming - Deep Learning with PyTorch. [Online]. Available: https://deeplizard.com/learn/video/5T-iXNNiwIs [Accessed: 7 May 2023]

[12] Apeer_micro. "What is deep learning and convolutional neural networks (CNN), Tutorial 89." [Online] Available: https://www.youtube.com/watch?v=2eQVKZFOHpI [Accessed: May 21, 2023]."

[13] Flatiron School. Deep Learning vs. Machine Learning. Flatiron School Blog. [Online] Available: https://flatironschool.com/blog/deep-learning-vs-machine-learning/ [Accessed 29 Mar. 2023]

[14] Kolosov, D., & Kelefouras, V. (2022). Anatomy of Deep Learning Image Classification and Object Detection on Commercial Edge Devices: A Case Study on Face Mask Detection. University of Patras.

[15] Kingma, Diederik P., and Jimmy Ba. Adam: A Method for Stochastic Optimization. [Online]. Available: https://arxiv.org/abs/1412.6980. [Accessed 29 Mar. 2023]

[16] FastAI. Overfitting. FastAI Reference. [Online] Available:https://www.fastaireference.com/overfitting [Accessed 29 Mar. 2023]

[17] Apeer_micro. "Introductory python tutorials for image processing, Tutorial 97." [Online]. Available: https://www.youtube.com/watch?v=OSY7hWADMZk. [Accessed 29 Mar. 2023]

[18] Wikipedia. File:Grafico con MATLAB di una superficie.png. Wikimedia Commons. [Online] Available:https://upload.wikimedia.org/wikipedia/commons/8/87/Grafico_con_MATLAB_di_una_superficie.png [Accessed 29 Mar. 2023]".

[19] Manaswi, N.K. (2018). Deep Learning with Applications Using Python. 1st ed. Apress.

[20] TensorFlow. Transfer learning. TensorFlow Tutorials. [Online] Available:https://www.tensorflow.org/tutorials/images/transfer_learning [Accessed 29 Mar. 2023]".

[21] Raspberry Pi. Processors. Raspberry Pi Documentation. [Online] Available: https://www.raspberrypi.com/documentation/computers/processors.html [Accessed 18 Mar. 2023]

[22] Farnell. SBC Raspberry Pi 2 Model B V1.2 [Online]. Available: https://es.farnell.com/raspberry-pi/rpi2-modb-v1-2/sbc-raspberry-pi-2-model-b-v1/dp/3772475?st=raspberry%20pi%202. [Accessed 29 Mar. 2023].

[23] Raspipc. Tienda Raspipc [Online]. Available: https://www.raspipc.es/index.php?ver=tienda&accion=verArticulosFamilia&idFamilia=3. [Accessed 29 Mar. 2023].

[24] Kubii. Raspberry Pi Zero v1.3 [Online]. Available: https://www.kubii.es/raspberry-pi-3-2-b/1401-raspberry-pi-cero-v13-kubii-3272496006973.html?src=raspberrypi. [Accessed 29 Mar. 2023].

[25] Raspberry Pi Camera Module V2-8 Megapixel,1080p [Online]. Available: https://www.amazon.es/Raspberry-Pi-Camera-Module-8MP/dp/B01ER2SKFS/. [Accessed 27 Mar. 2023].

[26] Freenove 5 inch Touchscreen with Case, 800x480 TFT LCD Display HDMI Monitor for Raspberry Pi 4B/3B+/3B/2B/B+/A+/Zero W, Capacitive Touch Screen with Touch Pen, Driver-Free [Online]. Available: https://www.amazon.es/Freenove-Touchscreen-Raspberry-Capacitive-Driver-Free/dp/B0B455LDKH. [Accessed 27 Mar. 2023].

[27] Amazon.es. POSUGEAR Batería Externa 20000mAh Mini Power Bank Carga Rapida 22.5W PD & QC 4.0 USB C Cargador Portatil con 3 Salidas, Compatible con iPhone13/12 Pro MAX, XS, Samsung, Xiaomi, Huawei, iPad, Airpods. [Online] Available: https://www.amazon.es/POSUGEAR-20000mah-Bateria-Portable-Compatible/dp/B09LM3RS2N [Accessed 28 Mar. 2023]

[28] Apeer_micro. “Deep Learning terminology explained – Learning curves, Tutorial 101.” [Online]. Available: https://www.youtube.com/watch?v=SNQEpOdPt5g. [Accessed 29 Mar. 2023]

[29] Apeer_micro. “Deep Learning terminology explained – ROC curves and AUC, Tutorial 106.” [Online]. Available: https://www.youtube.com/watch?v=jbeATQXKtzw. [Accessed 29 Mar. 2023]

[30] Google Developers, "Classification: Accuracy," [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy. [Accessed 29 Apr. 2023].

[31] Google Developers, "Classification: Precision and Recall," [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall. [Accessed 29 Apr. 2023].

[32] TensorFlow, "F1Score," [Online]. Available: https://www.tensorflow.org/addons/api_docs/python/tfa/metrics/F1Score. [Accessed 29 Apr. 2023].

[33] Google Developers, "Logistic Regression: Model Training," [Online]. Available: https://developers.google.com/machine-learning/crash-course/logistic-regression/model-training. [Accessed 29 Apr. 2023].

[34] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In 2017 IEEE International Conference on Computer Vision (ICCV) (pp. 618-626). IEEE.

[35] Keras, "Grad-CAM class activation visualization," [Online]. Available: https://keras.io/examples/vision/grad_cam/. [Accessed 29 Apr. 2023].

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

APPENDIX

TITLE: Computer vision for bird strike prevention

DEGREE: Bachelor's degree in Aerospace Systems Engineering and Telecommunications Systems

AUTHOR: Adrià Ibáñez i Boix

ADVISORS: Alberto Burgos Plaza & Francisco Javier Mora Serrano

DATA: 12th July 2023

DATASHEET

Raspberry Pi 4 Model B

Release 1

June 2019

Table 1: Release History

Release	Date	Description
1	21/06/2019	First release

The latest release of this document can be found at `https://www.raspberrypi.org`

Contents

List of Figures

List of Tables

1 Introduction

The Raspberry Pi 4 Model B (Pi4B) is the first of a new generation of Raspberry Pi computers supporting more RAM and with siginficantly enhanced CPU, GPU and I/O performance; all within a similar form factor, power envelope and cost as the previous generation Raspberry Pi 3B+.

The Pi4B is avaiable with either 1, 2 and 4 Gigabytes of LPDDR4 SDRAM.

2 Features

2.1 Hardware

- Quad core 64-bit ARM-Cortex A72 running at 1.5GHz
- 1, 2 and 4 Gigabyte LPDDR4 RAM options
- H.265 (HEVC) hardware decode (up to 4Kp60)
- H.264 hardware decode (up to 1080p60)
- VideoCore VI 3D Graphics
- Supports dual HDMI display output up to 4Kp60

2.2 Interfaces

- 802.11 b/g/n/ac Wireless LAN
- Bluetooth 5.0 with BLE
- 1x SD Card
- 2x micro-HDMI ports supporting dual displays up to 4Kp60 resolution
- 2x USB2 ports
- 2x USB3 ports
- 1x Gigabit Ethernet port (supports PoE with add-on PoE HAT)
- 1x Raspberry Pi camera port (2-lane MIPI CSI)
- 1x Raspberry Pi display port (2-lane MIPI DSI)
- 28x user GPIO supporting various interface options:
  - Up to 6x UART
  - Up to 6x I2C
  - Up to 5x SPI
  - 1x SDIO interface
  - 1x DPI (Parallel RGB Display)
  - 1x PCM
  - Up to 2x PWM channels
  - Up to 3x GPCLK outputs

2.3 Software

- ARMv8 Instruction Set
- Mature Linux software stack
- Actively developed and maintained
  - Recent Linux kernel support
  - Many drivers upstreamed
  - Stable and well supported userland
  - Availability of GPU functions using standard APIs

3 Mechanical Specification

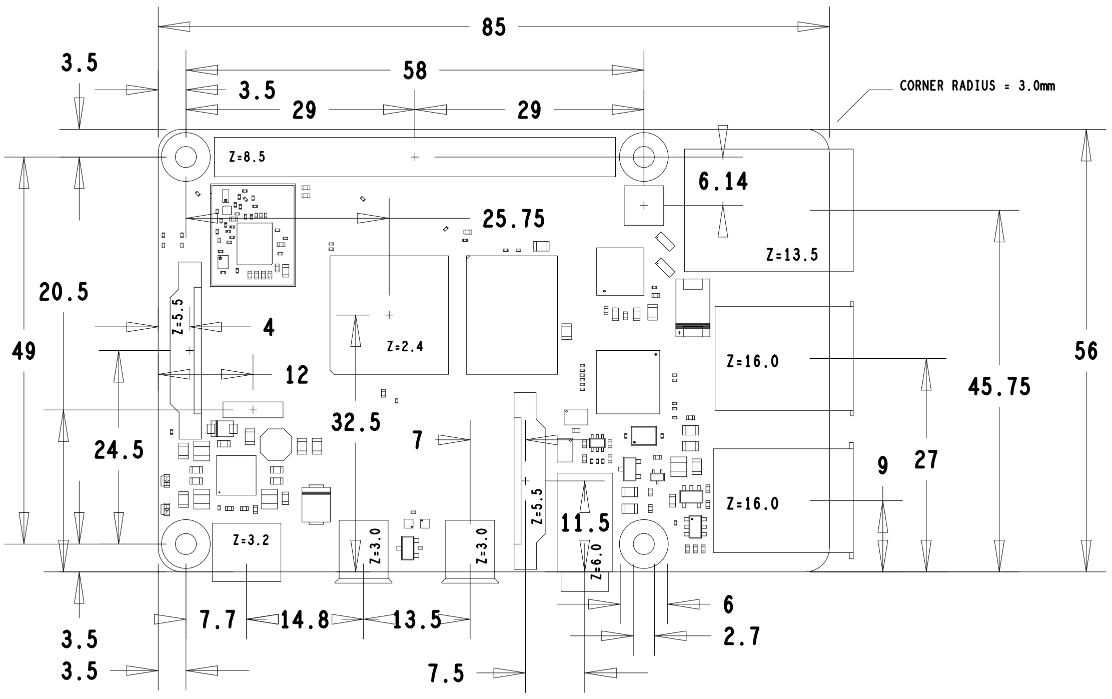


Figure 1: Mechanical Dimensions

4 Electrical Specification

Caution! Stresses above those listed in Table 2 may cause permanent damage to the device. This is a stress rating only; functional operation of the device under these or any other conditions above those listed in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Symbol	Parameter	Minimum	Maximum	Unit
VIN	5V Input Voltage	-0.5	6.0	V

Table 2: Absolute Maximum Ratings

Please note that VDD_IO is the GPIO bank voltage which is tied to the on-board 3.3V supply rail.

Symbol	Parameter	Conditions	Minimum	Typical	Maximum	Unit
V_{IL}	Input low voltage[a]	VDD_IO = 3.3V	-	-	TBD	V
V_{IH}	Input high voltage[a]	VDD_IO = 3.3V	TBD	-	-	V
I_{IL}	Input leakage current	TA = +85°C	-	-	TBD	μA
C_{IN}	Input capacitance	-	-	TBD	-	pF
V_{OL}	Output low voltage[b]	VDD_IO = 3.3V, IOL = -2mA	-	-	TBD	V
V_{OH}	Output high voltage[b]	VDD_IO = 3.3V, IOH = 2mA	TBD	-	-	V
I_{OL}	Output low current[c]	VDD_IO = 3.3V, VO = 0.4V	TBD	-	-	mA
I_{OH}	Output high current[c]	VDD_IO = 3.3V, VO = 2.3V	TBD	-	-	mA
R_{PU}	Pullup resistor	-	TBD	-	TBD	kΩ
R_{PD}	Pulldown resistor	-	TBD	-	TBD	kΩ

[a] Hysteresis enabled
[b] Default drive strength (8mA)
[c] Maximum drive strength (16mA)

Table 3: DC Characteristics

Pin Name	Symbol	Parameter	Minimum	Typical	Maximum	Unit
Digital outputs	t_{rise}	10-90% rise time[a]	-	TBD	-	ns
Digital outputs	t_{fall}	90-10% fall time[a]	-	TBD	-	ns

[a] Default drive strength, CL = 5pF, VDD_IO = 3.3V

Table 4: Digital I/O Pin AC Characteristics

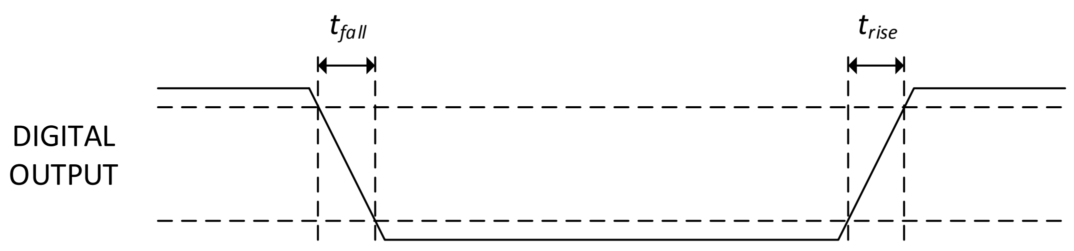


Figure 2: Digital IO Characteristics

4.1 Power Requirements

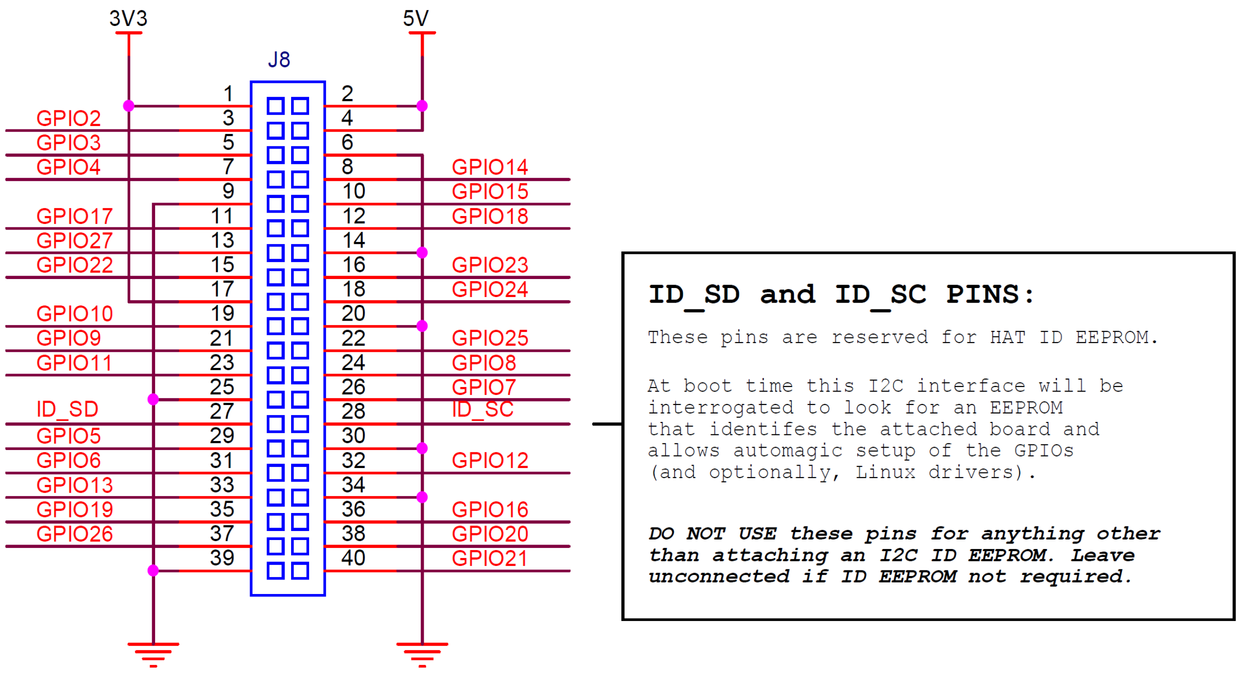
The Pi4B requires a good quality USB-C power supply capable of delivering 5V at 3A. If attached downstream USB devices consume less than 500mA, a 5V, 2.5A supply may be used.

5 Peripherals

5.1 GPIO Interface

The Pi4B makes 28 BCM2711 GPIOs available via a standard Raspberry Pi 40-pin header. This header is backwards compatible with all previous Raspberry Pi boards with a 40-way header.

5.1.1 GPIO Pin Assignments



ID_SD and ID_SC PINS:

These pins are reserved for HAT ID EEPROM.

At boot time this I2C interface will be interrogated to look for an EEPROM that identifes the attached board and allows automagic setup of the GPIOs (and optionally, Linux drivers).

DO NOT USE these pins for anything other than attaching an I2C ID EEPROM. Leave unconnected if ID EEPROM not required.

Figure 3: GPIO Connector Pinout

As well as being able to be used as straightforward software controlled input and output (with programmable pulls), GPIO pins can be switched (multiplexed) into various other modes backed by dedicated peripheral blocks such as I2C, UART and SPI.

In addition to the standard peripheral options found on legacy Pis, extra I2C, UART and SPI peripherals have been added to the BCM2711 chip and are available as further mux options on the Pi4. This gives users much more flexibility when attaching add-on hardware as compared to older models.

5.1.2 GPIO Alternate Functions

GPIO	Default Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
0	High	SDA0	SA5	PCLK	SPI3_CE0_N	TXD2	SDA6
1	High	SCL0	SA4	DE	SPI3_MISO	RXD2	SCL6
2	High	SDA1	SA3	LCD_VSYNC	SPI3_MOSI	CTS2	SDA3
3	High	SCL1	SA2	LCD_HSYNC	SPI3_SCLK	RTS2	SCL3
4	High	GPCLK0	SA1	DPI_D0	SPI4_CE0_N	TXD3	SDA3
5	High	GPCLK1	SA0	DPI_D1	SPI4_MISO	RXD3	SCL3
6	High	GPCLK2	SOE_N	DPI_D2	SPI4_MOSI	CTS3	SDA4
7	High	SPI0_CE1_N	SWE_N	DPI_D3	SPI4_SCLK	RTS3	SCL4
8	High	SPI0_CE0_N	SD0	DPI_D4	-	TXD4	SDA4
9	Low	SPI0_MISO	SD1	DPI_D5	-	RXD4	SCL4
10	Low	SPI0_MOSI	SD2	DPI_D6	-	CTS4	SDA5
11	Low	SPI0_SCLK	SD3	DPI_D7	-	RTS4	SCL5
12	Low	PWM0	SD4	DPI_D8	SPI5_CE0_N	TXD5	SDA5
13	Low	PWM1	SD5	DPI_D9	SPI5_MISO	RXD5	SCL5
14	Low	TXD0	SD6	DPI_D10	SPI5_MOSI	CTS5	TXD1
15	Low	RXD0	SD7	DPI_D11	SPI5_SCLK	RTS5	RXD1
16	Low	FL0	SD8	DPI_D12	CTS0	SPI1_CE2_N	CTS1
17	Low	FL1	SD9	DPI_D13	RTS0	SPI1_CE1_N	RTS1
18	Low	PCM_CLK	SD10	DPI_D14	SPI6_CE0_N	SPI1_CE0_N	PWM0
19	Low	PCM_FS	SD11	DPI_D15	SPI6_MISO	SPI1_MISO	PWM1
20	Low	PCM_DIN	SD12	DPI_D16	SPI6_MOSI	SPI1_MOSI	GPCLK0
21	Low	PCM_DOUT	SD13	DPI_D17	SPI6_SCLK	SPI1_SCLK	GPCLK1
22	Low	SD0_CLK	SD14	DPI_D18	SD1_CLK	ARM_TRST	SDA6
23	Low	SD0_CMD	SD15	DPI_D19	SD1_CMD	ARM_RTCK	SCL6
24	Low	SD0_DAT0	SD16	DPI_D20	SD1_DAT0	ARM_TDO	SPI3_CE1_N
25	Low	SD0_DAT1	SD17	DPI_D21	SD1_DAT1	ARM_TCK	SPI4_CE1_N
26	Low	SD0_DAT2	TE0	DPI_D22	SD1_DAT2	ARM_TDI	SPI5_CE1_N
27	Low	SD0_DAT3	TE1	DPI_D23	SD1_DAT3	ARM_TMS	SPI6_CE1_N

Table 5: Raspberry Pi 4 GPIO Alternate Functions

Table 5 details the default pin pull state and available alternate GPIO functions. Most of these alternate peripheral functions are described in detail in the BCM2711 Peripherals Specification document which can be downloaded from the hardware documentation section of the website.

5.1.3 Display Parallel Interface (DPI)

A standard parallel RGB (DPI) interface is available the GPIOs. This up-to-24-bit parallel interface can support a secondary display.

5.1.4 SD/SDIO Interface

The Pi4B has a dedicated SD card socket which suports 1.8V, DDR50 mode (at a peak bandwidth of 50 Megabytes / sec). In addition, a legacy SDIO interface is available on the GPIO pins.

5.2 Camera and Display Interfaces

The Pi4B has 1x Raspberry Pi 2-lane MIPI CSI Camera and 1x Raspberry Pi 2-lane MIPI DSI Display connector. These connectors are backwards compatible with legacy Raspberry Pi boards, and support all of the available Raspberry Pi camera and display peripherals.

5.3 USB

The Pi4B has 2x USB2 and 2x USB3 type-A sockets. Downstream USB current is limited to approximately 1.1A in aggregate over the four sockets.

5.4 HDMI

The Pi4B has 2x micro-HDMI ports, both of which support CEC and HDMI 2.0 with resolutions up to 4Kp60.

5.5 Audio and Composite (TV Out)

The Pi4B supports near-CD-quality analogue audio output and composite TV-output via a 4-ring TRS ’A/V’ jack.

The analog audio output can drive 32 Ohm headphones directly.

5.6 Temperature Range and Thermals

The recommended ambient operating temperature range is 0 to 50 degrees Celcius.

To reduce thermal output when idling or under light load, the Pi4B reduces the CPU clock speed and voltage. During heavier load the speed and voltage (and hence thermal output) are increased. The internal governor will throttle back both the CPU speed and voltage to make sure the CPU temperature never exceeds 85 degrees C.

The Pi4B will operate perfectly well without any extra cooling and is designed for sprint performance - expecting a light use case on average and ramping up the CPU speed when needed (e.g. when loading a webpage). If a user wishes to load the system continually or operate it at a high termperature at full performance, further cooling may be needed.

6 Availability

Raspberry Pi guarantee availability Pi4B until at least January 2026.

7 Support

For support please see the hardware documentation section of the Raspberry Pi website and post questions to the Raspberry Pi forum.

Raspberry Pi Camera v2

***Part number:** RPI 8MP CAMERA BOARD*

- 8 megapixel camera capable of taking photographs of 3280 x 2464 pixels
- Capture video at 1080p30, 720p60 and 640x480p90 resolutions
- All software is supported within the latest version of Raspbian Operating System

The Camera v2 is the new official camera board released by the Raspberry Pi foundation.

The Raspberry Pi Camera Module v2 is a high quality 8 megapixel Sony IMX219 image sensor custom designed add-on board for Raspberry Pi, featuring a fixed focus lens. It's capable of 3280 x 2464 pixel static images, and also supports 1080p30, 720p60 and 640x480p60/90 video. It attaches to Pi by way of one of the small sockets on the board upper surface and uses the dedicated CSi interface, designed especially for interfacing to cameras.

- 8 megapixel native resolution sensor-capable of 3280 x 2464 pixel static images
- Supports 1080p30, 720p60 and 640x480p90 video
- Camera is supported in the latest version of Raspbian, Raspberry Pi's preferred operating system

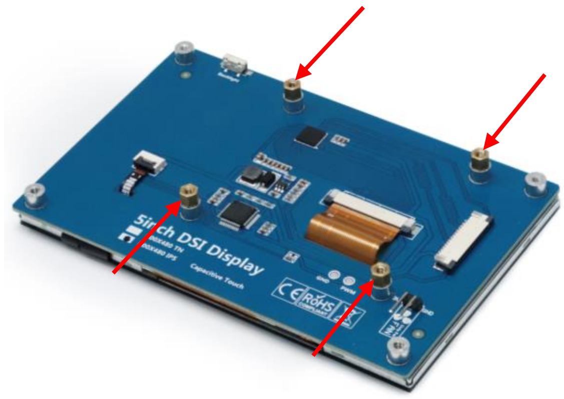
The board itself is tiny, at around 25mm x 23mm x 9mm. It also weighs just over 3g, making it perfect for mobile or other applications where size and weight are important. It connects to Raspberry Pi by way of a short ribbon cable.
The high quality Sony IMX219 image sensor itself has a native resolution of 8 megapixel, and has a fixed focus lens on-board. In terms of still images, the camera is capable of 3280 x 2464 pixel static images, and also supports 1080p30, 720p60 and 640x480p90 video.

Applications

- CCTV security camera
- motion detection
- time lapse photography

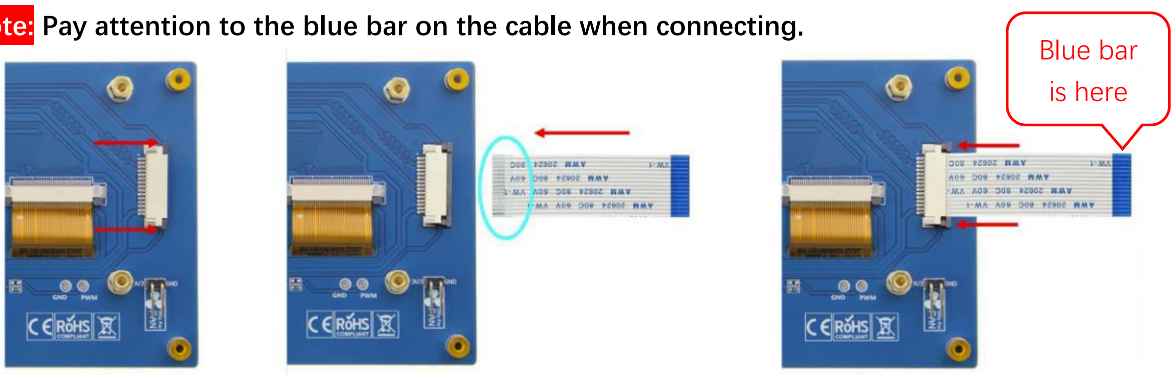
How to Connect?

1. After writing OS, insert the micro SD card to Raspberry Pi.
2. Install 4 brass standoffs.

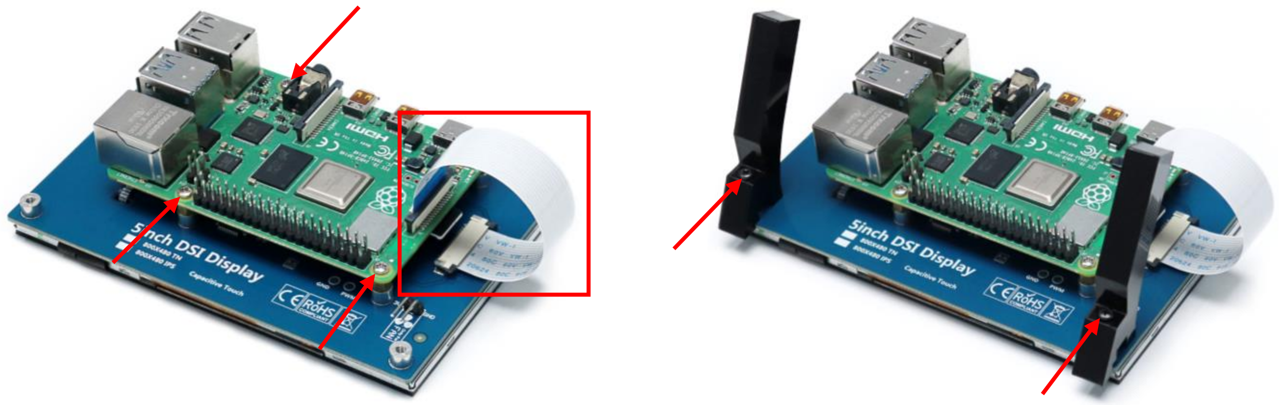


3. Connect the ribbon cable.

   Note: **Pay attention to the blue bar on the cable when connecting.**



4. Fix the Raspberry Pi with 4 screws and then connect the ribbon cable. Then install the stands with 2 screws.



5. Connect the power supply and wait a few seconds, the screen will display.

   If the screen doesn't display normally with Raspberry Pi OS

   Open **boot/config.txt** on Raspberry Pi or use a computer to open this file in the micro SD card.

   Modify **dtoverlay=vc4-kms-v3d** to **dtoverlay=vc4-fkms-v3d** or **#dtoverlay=vc4-kms-v3d**

   Save the file and then reboot the Raspberry Pi.

Having problems? Download the latest tutorial and troubleshooting: http://freenove.com/fnk0078

Need further help? Contact our technical support by email: support@freenove.com