Roman Wolf 09/04/2024

Data Science Nanodegree

RooVision – Detect and Classify Kangaroos

Table of Contents

1 Project Definition 2

1. Project Overview 2

1.1 Problem statement 2

1.2 Metrics 3

2 Analysis 3

2.1 Data Exploration 3

2.2 Image sizes and aspect ratios 4

2.3 Object sizes and dimensions 5

2.4 Balance of the Species Classes 6

3 Methodology 7

3.1 Data Preprocessing 7

3.2 Data Augmentation 8

3.3 Implementation 8

3.3.1 Complications 10

3.3.2 Models 10

3.3.3 Metrics and Model Performance 10

3.3.4 Training Data and Training 11

3.4 Refinement 11

3.4.1 Dataset 12

3.4.2 Models 12

3.4.3 Hardware 12

3.4.4 Training Settings 12

4 Results 14

4.1 Model Evaluation and Validation 14

4.1.1 Selection of parameters 14

4.1.2 Model training 300 Epochs 15

4.1.3 Per-Class Validation 15

4.1.4 Model performance 17

4.1.5 Confusion Matrix 18

4.2 Justification 19

4.2.1 Resources 19
4.2.2 Data 20
4.2.3 Model 20
4.2.4 Biology 21
4.2.5 Improvements 21
5 Conclusion 21
5.1 Reflection 21
5.2 Improvement 22
6 References 22
7 Appendix 23

1 Project Definition

1. Project Overview

The project focuses on the detection and classification of species within the family Macropodidae native to Victoria, Australia. This family includes a diverse range of marsupials, in particular kangaroos, wallabies, and wallaroos. The initiative is driven by the need to understand and conserve the region's biodiversity and to support wildlife management and conservation efforts.

The project was born out of the critical need to accurately identify and monitor various Macropodidae species. This requires the use of advanced technologies, including machine learning and computer vision, to automate species detection and classification processes. Through data-driven methods, the project aims to streamline wildlife monitoring while providing insights into population dynamics and ecological well-being.

To achieve its goals, the project relies on the following data:

1. Images of *Macropodidae* in their natural habitats.
2. Annotated datasets containing labelled examples of different species for training machine learning algorithms.
3. Domain knowledge about the abundance of a species and major physical properties.

By integrating these data and information and applying advanced data analysis and machine learning techniques, the project aims at developing a robust model capable of accurately identifying and classifying *Macropodidae* species.

Ultimately, the project aims to contribute to the education of wildlife enthusiasts by helping them to better understand the species they observe in Victorian nature.

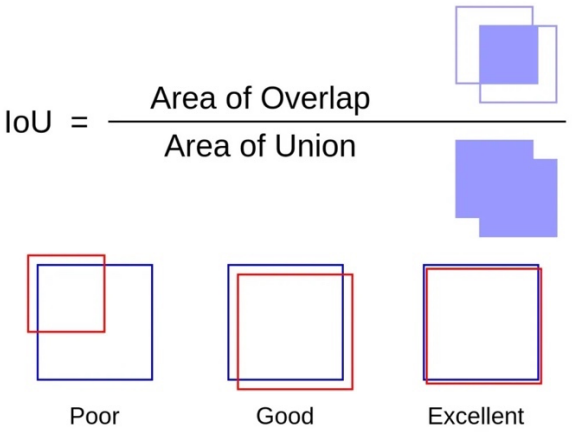
1.1 Problem statement

When enjoying Victoria's wildlife, it is common to encounter species from the Macropodidae family. However, it is relatively difficult for the untrained eye to tell which species you

are looking at. The aim of the project is to develop an app that allows the user to upload images that can be used to locate and classify Macropodidae species. In other words, the app will help nature lovers understand what they are seeing in the wild.

1.2 Metrics

To measure the performance of the model the mean average precision (mAP) is used. In object detection, the employed evaluation metrics measure how close the detected bounding boxes are to the ground-truth bounding boxes. This measurement is done independently for each object class, by assessing the amount of overlap of the predicted and ground-truth areas (Padilla et al. 2021). Ideally, the area and location of the predicted and ground-truth boxes are the same which can be measured by the intersection over union (IOU).



Source: *https://idiotdeveloper.com/what-is-intersection-over-union-iou/*

It is common to use different mean average precisions that refer to the distinct IoUs. Using the YOLO algorithm by default mAP50 and mAP50-95 are reported. The term mAP50 represents the mean Average Precision at an IoU threshold of 0.5. This means that only detections with an IoU of 0.5 or higher with ground truth boxes are considered correct when calculating mAP50.

The term mAP50-95 extends this evaluation to a range of IoU thresholds from 0.5 to 0.95. mAP50-95 calculates the mean Average Precision across a range of IoU thresholds, providing a more comprehensive assessment of detection performance.

Typically, mAP50 is lower than mAP50-95 because the latter considers a wider range of IoU thresholds, including those higher than 0.5. The use of higher IoU thresholds necessitates that detections have greater overlap with the ground truth boxes, making it more difficult for detections to be deemed correct. As a result, mAP50-95 is typically higher than mAP50 due to its more stringent evaluation criteria.

2 Analysis

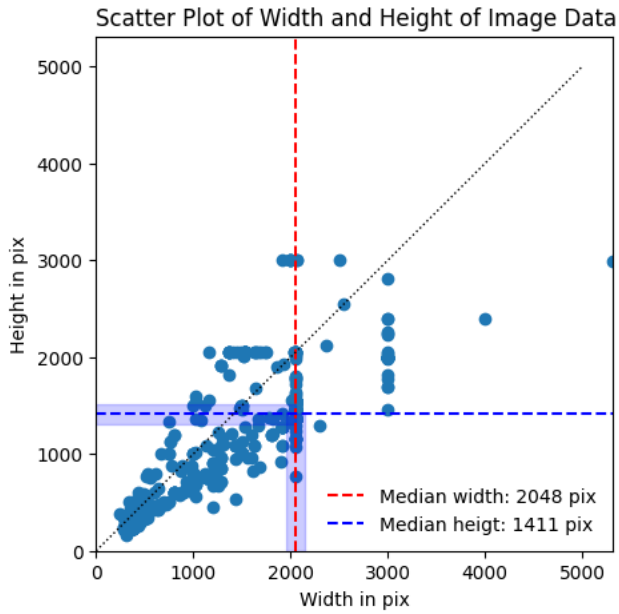
2.1 Data Exploration

An exploratory data analysis was carried out within a Jupyter notebook (referenced). This analysis differs from other areas of data science due to the unique nature of images, where metadata plays a crucial role in pre-processing procedures.

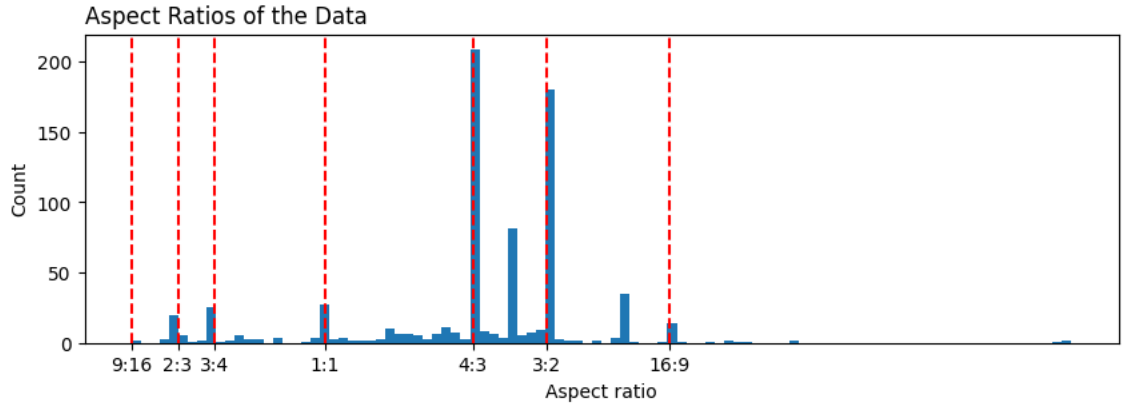
2.2 Image sizes and aspect ratios

In real-world scenarios, datasets often contain images of varying sizes and aspect ratios. Analysing the basic statistics of datasets, such as aspect ratios, image widths and heights, is critical to making informed decisions:

- Considering whether to perform destructive resizing, which changes aspect ratios, and whether it is necessary.
- For non-destructive resizing, determining the preferred output resolution and amount of padding is essential. This decision affects the hyperparameters of the deep learning model, such as anchor size and ratios, and may even involve meeting minimum input image size requirements.



The visualisation shows the median values for both the width and height of the dataset, providing insight into the distribution of sizes and approximate aspect ratios. The diagonal line represents a 1:1 aspect ratio, indicating regions where width exceeds height and vice versa.



According to Wikipedia, the most common aspect ratios in still photography are 4:3, 3:2 (equivalent to 1.5:1) and, increasingly seen in consumer cameras, 16:9, which corresponds to ratios of 1.33, 1.5 or 1.77. Most of the images in the data were taken in these formats, or their inverse in the case of the upright format, but with some variations, the 3rd and 4th most common ratios are uncommon. In general, most images are wider than they are tall. This must be taken into account when pre-processing the images.

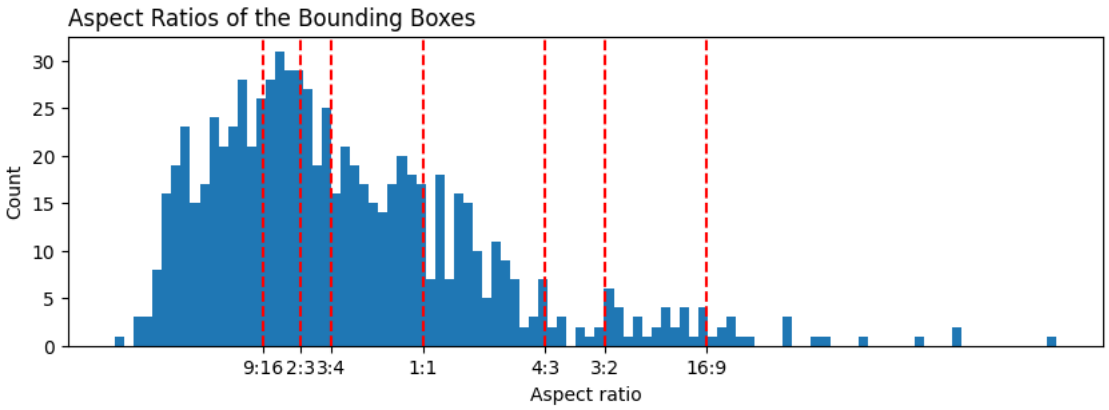
2.3 Object sizes and dimensions

This section focuses on the analysis of target labels, specifically to understand the distribution of sizes and aspect ratios. The importance of this analysis lies in the design constraints inherent in different modelling approaches. Models are typically optimised to perform well on standardised benchmark datasets. However, deviations from these norms in your data can make effective training of these models impossible.

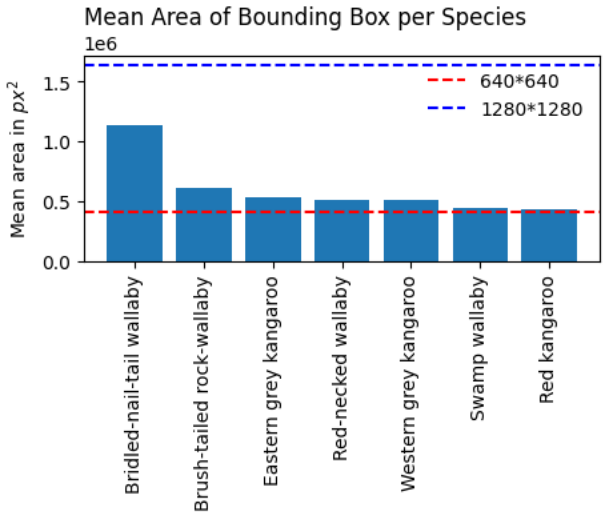
For example, if a dataset consists primarily of large objects, there may be an opportunity to significantly simplify the model. Conversely, if a dataset contains small images with small objects (e.g. 10x10px), training the model may prove challenging.

Key considerations for box or mask dimensions include:

- Aspect ratio
- Size (area)



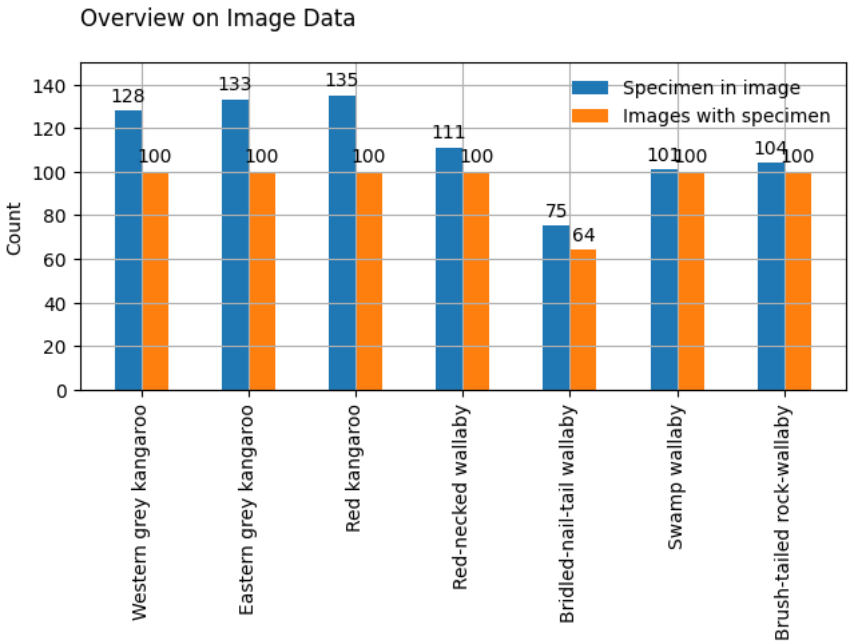
The histogram shows that the majority of the bounding boxes have an aspect ratio of less than 1, suggesting that they are higher than they are wide. This mismatch in image format is indicative of the pose of the specimen in the images. Evaluating model training based on these ratios could provide valuable insights. It would be worth investigating whether the trained model has difficulties with certain poses, and whether examining the associated training data could improve model performance.



The analysis shows a relatively low occurrence of species instances per image, with an average of slightly more than one animal per image. This observation is important when evaluating model performance for images containing multiple animals, as the effectiveness of the model may decrease in such scenarios. In addition, the dataset includes 100 images per species and 100 background images, with the exception of the bridled nail-tail wallaby due to limited image availability in the database. While it was possible to increase the number of images for other species to reach the recommended threshold of approximately 1,500 per species, this approach was avoided to mitigate the effects of class imbalance.

2.4 Balance of the Species Classes

The following section examines the balance of species classes within the dataset. In general, care should be taken to balance the number of images per class to reduce the possibility of introducing error due to over- or under-representation in the data.



The analysis shows a relatively low occurrence of species instances per image, with an average of slightly more than one animal per image. This observation is important when evaluating model performance for images containing multiple animals, as the effectiveness of the model may decrease in such scenarios. In addition, the dataset includes 100 images per species and 100 background images, with the exception of the bridled nail-tail wallaby due to limited image availability in the database. While it was possible to increase the number of images for other species to reach the recommended threshold of approximately 1,500 per species, this approach was avoided to mitigate the effects of class imbalance.

3 Methodology

3.1 Data Preprocessing

Two different models were tested in the analysis, YOLO and Faster R-CNN. Although both are based on PyTorch, the preprocessing is organised differently:

- In YOLO, data preprocessing and augmentation is controlled by parameter settings in a YAML file. Under the hood, additional libraries such as Albumentations are used to manipulate the data,
- while the preprocessing for Faster R-CNN is done purely in PyTorch.

For the sake of comparison, the preprocessing has been assimilated for both models in order to obtain meaningful results.

The following preprocessing steps transform the input data in place. They modify the appearance of the images to introduce variations that can help the model learn robust features without increasing the size of the dataset.

- **hsv_h** (value: 0.015): Image HSV hue augmentation adjusts the hue component of the image in the HSV (hue, saturation, value) colour space. The fraction parameter determines the range of adjustment applied to the hue values.
- **hsv_s** (value: 0.7): Image HSV saturation augmentation modifies the saturation component of the image in the HSV colour space. The fraction parameter controls the amount of saturation adjustment.
- **hsv_v** (value: 0.4): Image HSV valence augmentation modifies the valence component of the image in the HSV colour space. The fraction parameter controls the amount of adjustment applied to the valence channel.
- **Degrees** (value: 0.0): Image rotation rotates the image by a specified number of degrees. The degree parameter determines the range of rotation applied, typically clockwise or counterclockwise.
- **Translate** (value: 0.1): Image translation shifts the image horizontally and/or vertically by a fraction of its size. The translate parameter controls the amount of translation along each axis.
- **Scale** (value: 0.9): Image Scale adjusts the size of the image by a gain factor. The scale parameter specifies the amount of scaling, with a value greater than 1 increasing the size of the image and a value less than 1 decreasing the size of the image.
- **Shear** (value: 0.0): Image Shear applies a shear transformation to the image, distorting its shape along a specified axis. The shear parameter determines the amount of shear, typically in degrees.

- **Perspective** (value: 0.0): The Perspective transformation changes the perspective of the image, simulating the effect of viewing the image from different angles. The Perspective parameter controls the amount of perspective adjustment, typically specified as a fraction.

Additionally, input images are cropped to the defined image size, typically 640 or 320 pixels, by resizing to --img-size on the longer side, with the shorter side automatically adjusted to preserve the aspect ratio. Letterboxing, which involves adding grey padding, is used to meet stride constraints (i.e. 32 or 64 pixel multiples).

3.2 Data Augmentation

Of the complete set of parameters of YOLO the following are considered data augmentation techniques as they transform the input data to create more samples. They are used to generate additional variations of the input data, which helps prevent overfitting by providing the model with a more diverse training set. The indicated values represent the probability of applying the transformation.

- **Flipud** (value: 0.0): This augmentation flips images vertically, effectively reversing their orientation along the vertical axis. With a value of 0.0, indicating a probability of 0%, this augmentation is not applied during training.
- **Fliplr** (value: 0.5): This augmentation flips images horizontally, mirroring them along the vertical axis. With a value of 0.5, indicating a probability of 50%, this enhancement will be applied to half of the images during training.
- **Mosaic** (value: 1.0): This augmentation combines multiple images into a single mosaic, creating a synthetic dataset with increased diversity. With a value of 1.0, indicating a probability of 100%, this enhancement is applied to all images during training.
- **Mixup** (value: 0.1): This augmentation mixes pairs of images and their corresponding labels to create new training samples, improving model generalisation. With a value of 0.1, indicating a probability of 10%, this augmentation is applied to a small fraction of images during training.
- **Copy_paste** (value: 0.1): This augmentation pastes segments from one image onto another, introducing variability and complexity into the dataset. With a value of 0.1, indicating a probability of 10%, this augmentation is applied to a small fraction of the images during training.

3.3 Implementation

The implementation section describes the steps and technical procedures of the project, detailing the execution of algorithms, models and system components.

1) **Database exploration and acquisition**: The Atlas of Living Australia database was used to locate images relevant to the project species of interest.
2) **Data Download**: Implemented a script to download images corresponding to the identified species from the database.
3) Performed manual inspection of the downloaded data to identify and resolve any apparent anomalies or problems.
4) **Exploratory Data Analysis** (EDA): Performed extensive EDA on the image data to gain insight into their characteristics and distributions.

5) **Image Annotation Selection**: Selection of appropriate images for annotation based on the results of the EDA process.
6) **Data Annotation**: Uploaded selected images to Roboflow for efficient annotation using additional features provided by the platform.
7) **Automated model training**: Trained an automated machine learning algorithm to speed up the annotation process.
8) **Annotation**: Manually annotated the data to facilitate model training and validation.
9) **Data splitting**: Performed a stratified training validation test split (70-20-10) to ensure representative distributions across subsets.
10) **Annotation Scheme**: Downloaded data in the desired annotation scheme for further processing.
11) **Local model training setup**: Set up the local environment for model training, including preprocessing and augmentation configurations.
12) **Local model training**: Trained Faster R-CNN and YOLOv9 architectures locally for 25 epochs each.
13) **Training Comparison**: Compared training results in terms of Mean Average Precision (mAP) to determine the more effective architecture.
14) **Model Selection**: Decided on the superior architecture based on training performance metrics.
15) **Cloud model training**: Trained the selected model on Google Colab for 300 epochs.
16) **Training Monitoring**: Regularly monitored training progress to ensure adherence to the intended trajectory and to identify potential problems such as overfitting or underfitting.
17) Validated the trained model to assess its performance on unseen data.
18) **Result Extraction**: Download the training and validation results along with the trained model weights for further analysis.
19) **Application Skeleton Coding**: Implemented the Flask application skeleton following the "application factory" pattern.
20) **Front-end integration**: Added front-end functionality using JavaScript to improve user interaction.
21) **Model integration**: Integrated the trained model into the Flask application to enable inference capabilities.
22) **Local Testing**: Performed thorough local testing of the Flask application to ensure proper fun factor.
23) **Create a Docker image** of the application along with the necessary dependencies for streamlined deployment.
24) **Local Container Execution**: Run the container locally, incorporating the Google secrets configuration through Docker Compose.
25) **Artifact Registry Push**: Uploaded the Docker image to the Google Artifact Registry for centralised access.
26) **IAM Setup**: Configured Identity and Access Management (IAM) to allow services to securely access secrets stored in Google Secret Manager.
27) **Cloud deployment**: Deployed the containerised application to Google Cloud Run for scalable and efficient execution. The application can be accessed via https://roovision-6bnwkndfqa-km.a.run.app/.
28) Updated **GitHub repository**: Published the final state of the project, including code, documentation and configuration files.

29) **Reporting**: Wrote a detailed report documenting the entire project lifecycle, including methodology, challenges encountered and lessons learned.

3.3.1 Complications

While PyTorch provides initial support for Apple's M1 series, not all features are fully compatible, which may result in local operational issues. For example, configuring the training device to use the M1 chip by specifying the device as Metal Performance Shaders (MPS) can result in significantly slower training speeds compared to using the CPU. This observation has been widely reported by users on PyTorch's GitHub platform.

In the context of CPU-based training, the use of multithreading can effectively reduce training times. This optimisation is typically achieved by specifying the number of workers. However, it is advisable to set the number of workers to 0 to avoid potential code execution errors.

3.3.2 Models

Two different models were tested:

- YOLO v9 (Wang et al. 2023) and
- Faster R-CNN (Ren et al. 2015).

YOLO (You Only Look Once) is a popular object detection algorithm known for its speed and accuracy. It divides an image into a grid and predicts bounding boxes and probabilities for each cell simultaneously. The latest release has reduced parameters and calculations by 49% and 43%, respectively, while still achieving an average precision of 0.6% on the MS COCO dataset. The YOLO architecture enables real-time detection and classification of objects (Jocher et al., 2024).

YOLOv9 seeks to overcome the challenges of information loss inherent in deep neural networks by integrating PGI and the versatile GELAN architecture. This enhances YOLOv9's ability to learn and retain crucial information throughout the detection process, resulting in exceptional accuracy and performance.

To address the challenge of information loss in deep learning, YOLOv9 implements the Programmable Gradient Information (PGI). PGI helps preserve essential data across the network's depth, resulting in more reliable gradient generation, better model convergence, and improved performance.

YOLOv9 introduces the Generalized Efficient Layer Aggregation Network (GELAN). GELAN is a strategic architectural advancement that enables YOLOv9 to achieve superior parameter utilization and computational efficiency.

Faster R-CNN (Region-based Convolutional Neural Network) stands as a prominent object detection algorithm celebrated for its remarkable precision and efficiency. Unlike *YOLO*, which predicts bounding boxes and probabilities directly from grid cells, Faster R-CNN follows a two-stage approach. It first proposes regions of interest (RoIs) through a region proposal network (RPN) and then classifies those proposals.

3.3.3 Metrics and Model Performance

To provide a general overview of the models' performance, the average precision is measured, which is widely used in object detection and measures the area under the precision-

recall curve. This metric calculates the area under the precision-recall curve and quantifies how well a model can correctly classify and localize an object. It takes into account both precision and recall, which are the true positives over all positives and the true positives over all actual positives, respectively. The better the overall performance of the model, the closer the average precision is to 1.0.

Despite the overall performance of a model, it is likely that not all classes are detected equally well. Therefore, confusion matrices can help to better understand where the model has problems detecting objects. The initial exploratory data analysis helps to understand how to assess the model's performance. For instance, identifying problems and improving training data can be aided by considering the aspect ratio, object size, number of annotations, and their overlap.

3.3.4 Training Data and Training

The training data was obtained from the Atlas of Living Australia (see script), an open-source databank that provides detailed information on the species of interest. It is unclear whether the images were verified by a professional to ensure that the depicted species is accurate. Images with obvious classification errors were removed from the dataset. To reduce the influence of class imbalance 100 images of each Victorian species were used, except for the bridled nail-tail wallaby for which there are only very few images on the databank:

- 64 Bridled nail-tail wallabies (*Onychogalea fraenata*)
- 100 Brush-tailed rock-wallabies (*Petrogale penicillata*)
- 100 Eastern grey kangaroos (*Macropus giganteus*)
- 100 Red kangaroos (*Ospranter rufus*)
- 100 Red-necked wallabies (*Notamacropus rufogriseus*)
- 100 Swamp wallabies (*Wallabia bicolor*)
- 100 Western grey kangaroos (*Macropus fuliginosus*)

The images were uploaded to Roboflow, where they were annotated and downloaded as a complete dataset. Image modifications or augmentations were performed during training. The models were trained in different environments.

- YOLO was trained partly locally and partly on Google Colab and
- Faster R-CNN was trained locally on Mac M1 Chip.

The models were trained for 25 epochs with a batch size of 16 and then compared. The better untuned model was then tuned by varying the model parameters and re-running an initial training period of 25 epochs. The best performing parameters were then used to train the model for 300 epochs with a batch size of 64 on Google Colab.

3.4 Refinement

The refinement section provides the documentation of the iterative process of tuning the algorithms and techniques employed.

3.4.1 Dataset

After downloading the annotated images from Roboflow some of the images were without annotations despite containing objects. These images were deleted from the dataset (see script).

After conducting the initial exploratory data analysis, it became evident that there was a significant class imbalance. To address this issue, a total of 100 images of each class were used. However, the availability of images for the bridled nail-tail wallaby is limited, with only 64 images currently available on the Atlas of Living Australia. This scarcity of images per class is likely to negatively impact the model's performance.

Additionally, it is ideal for the number of instances in an image to be similarly distributed across all classes. To enhance model performance, images with more than 5 instances were removed from the dataset after the initial EDA.

It is worth noting that some preliminarily downloaded images were captured at night using animal observation cameras, while the app is primarily intended for use during the day. Therefore, images with a dark background that were taken at night were removed from the dataset.

Another issue that needed to be addressed was the accuracy of the labels. Many images in the database come from independent sources such as hobby biologists or zoologists, which can lead to misinterpretation of images classified as a certain species. Images with obvious errors were deleted.

Additionally, it is recommended to use 0-10% background images to reduce false positives. However, the databank does not contain any images without specimens. It is assumed that adding background images to the dataset would improve model performance.

3.4.2 Models

In almost all cases, larger models will give better results. However, they have more parameters, require more memory to train and run slower. Smaller models are recommended for mobile and cloud deployments. In general, it is possible to improve model performance by using model versions with more parameters. However, due to limited resources, this step is not taken further.

3.4.3 Hardware

Effective computer vision model training relies heavily on hardware. Training on an Apple M1 Chip can significantly increase the time required to train a model. While libraries like PyTorch generally support the M1 architecture, certain parts of the training pipeline are not fully integrated. Comparing training on a Google Colab TPU4 environment (which is free) to local training on the M1 chip, the former leads to significantly faster training times. Therefore, the training was in part conducted on Google Colab and Roboflow to avoid disrupting the workflow due to training times exceeding a day.

3.4.4 Training Settings

Once YOLO was selected as the most promising model architecture, its model parameters were tuned. The first training of this phase was run with default settings to establish a performance baseline. Initially the training was set to 300 epochs. If early overfitting occurs, the

number of epochs is reduced, and if no overfitting occurs, the number of epochs is increased.

A pre-trained COCO model with a native resolution of 640 was used. To ensure optimal training, the resolution was kept at 640, as images with very small objects were removed from the dataset.

The batch size used depended on the available system memory, with a range of 8-64 being used for training. It is important to avoid small batch sizes as they produce poor batch-norm statistics, while large batch sizes can improve model generalisation.

The hyperparameters of a YOLO model are controlled by a YAML file (see file). In general, increasing the augmentation hyperparameters will reduce and delay overfitting, allowing longer training sessions and higher final mAP. These are the adjusted hyperparameters:

- **Learning rate** - initial (lr0) and final (lrf): controls how fast the model takes in information during training. If it is too large, the optimisation will diverge, if it is too small, it will take too long to train or we will end up with a suboptimal result. The combination of these two parameters creates a learning rate schedule. Initially both values are set to 0.1.
- **Momentum**: For the SGD (Stochastic Gradient Descent) optimiser, this parameter controls the momentum term, which helps to speed up the gradient descent in the relevant direction and dampens oscillations.
- **Weight Decay**: Also known as L2 regularisation, this parameter adds a penalty term to the loss function to prevent overfitting by discouraging large weight values.
- **Warmup Epochs**: Specifies the number of warmup epochs during which the learning rate gradually increases from a small value to the initial learning rate (lr0).
- **Warmup momentum**: Sets the initial impulse value during the warm-up period.
- **Warmup bias learning rate**: Sets the initial learning rate for the bias parameters during the warm-up.
- **box**, **cls**, **cls_pw**, **dfl**, **obj_pw**: These parameters relate to the various components of the loss function used to train the model. They assign weights or adjust the behaviour of different loss terms, such as localisation loss (box), classification loss (cls), focal loss (cls_pw), objectness loss (obj_pw), and others.
- **IoU_t**: Specifies the Intersection over Union (IoU) training threshold, which determines whether a predicted bounding box is considered a true positive during training.
- **Anchor_t**: Sets the threshold for the anchor multiple, which influences the selection of anchor boxes.
- **fl_gamma**: Sets the gamma parameter for the focal loss function, which helps to address class imbalance by focusing on hard examples.
- **hsv_h**, **hsv_s**, **hsv_v**: These parameters control the hue, saturation and value components of the image's HSV (Hue, Saturation, Value) colour space expansion.
- **Degree**, **translate**, **scale**, **shear**, **perspective**: These parameters control various types of image transformations such as rotation, translation, scaling, shearing and perspective distortion.
- **Flipud**, **fliplr**, **mosaic**, **mixup**, **copy_paste**: These parameters control additional data augmentation techniques such as flipping, mosaic augmentation, mixup augmentation, and segment copy-paste augmentation, which help improve model robustness and generalisation by introducing variation in the training data.

Each of these hyperparameters plays a critical role in effectively training a deep learning model and tuning them appropriately can have a significant impact on the model's performance and convergence behaviour. Apart from the default set 2 additional sets of parameters were created and tested for a short training period of 25 epochs (see Appendix 2).

In Set 1, several adjustments were made to fine-tune the training process and potentially improve the performance of YOLOv9. The learning rate (lr0) and final learning rate (lrf) were decreased to allow for smoother convergence, with a longer warmup period (warmup_epochs) to allow the model to adapt slowly. Momentum was slightly reduced for smoother optimisation, while weight decay was reduced to prevent overregularisation. The class loss weight (cls) and focus loss (dfl) parameters were adjusted to balance classification accuracy and focus. In addition, augmentation strategies such as flipud and mosaic were introduced with small probabilities to increase the diversity of the dataset without unduly distorting the training samples.

In Set 2, further adjustments were made to refine the training process and potentially improve YOLOv9's performance. Momentum was slightly increased for faster convergence, and weight decay was moderately reduced for improved generalisation. The warm-up period was slightly shortened to speed up adaptation, with adjustments to the warm-up momentum and bias learning rate to allow for faster ramp-up. Box parameters were increased to improve coverage, while class loss weight (cls) was decreased to prioritise localisation. Focal loss (dfl) and objectness loss (obj_pw) parameters were adjusted to sharpen the focus and increase object detection confidence. In addition, augmentation probabilities for horizontal flipping (fliplr) and mosaic were increased to further improve dataset diversity and training robustness.

The aim of these parameter adjustments is to optimise the training process and potentially improve the performance of YOLOv9 by achieving a more optimal trade-off between convergence speed, regularisation, classification accuracy and dataset diversity. In addition, it is recommended that a parameter evolution procedure be used to fine-tune the model; however, this functionality was not available for the latest YOLO architecture at the time of implementation. Although parameter evolution could potentially lead to improved results, it was not considered necessary due to the satisfactory results obtained.

4 Results

4.1 Model Evaluation and Validation

4.1.1 Selection of parameters

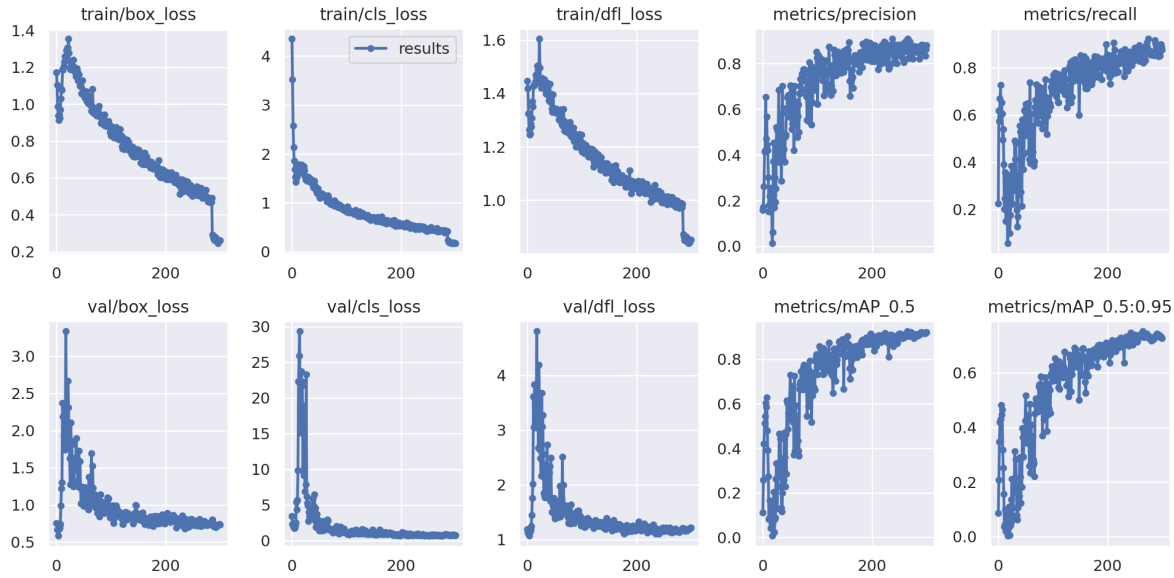
In relation to the so-called parameter evolution, two additional sets of parameters were created that deviated from the initial set of parameters. With these parameter sets, training was performed for a batch size of 16 with 25 epochs. The table shows the abbreviated results. The full results can be found in the Appendix 2. The best results were obtained with set number 2, so the model was trained with this set of parameters, as recommended, with a batch size of 64 for 300 epochs.

Set	Precision	Recall	mAP50	mAP50-95

Init	0.812	0.772	0.848	0.672
1	0.827	0.781	0.888	0.752
2	0.858	0.790	0.889	0.769

4.1.2 Model training 300 Epochs

The training results below demonstrate the successful progression of model training. After an initial stabilisation phase of about 25 epochs, key training metrics such as box loss, classification loss and distribution focal loss (DFL) show a gradual but consistent decrease. In particular, within the last 15 epochs of the 300-epoch training period, there is an accelerated reduction in training losses, likely due to the deactivation of mosaic augmentation, a practice commonly used to enhance model stabilisation towards the end of training.



The plots illustrate that training losses decrease at a faster rate than validation losses. Specifically, box and distribution focus losses show slower rates of decline compared to classification losses, indicating that the model requires less time for image classification relative to object localisation within an image.

Analogous to the patterns observed for training and validation losses, the performance metrics reflect an initial stabilisation phase, followed by a steady increase in both precision and recall throughout the training period. In summary, the model shows robust generalisation capabilities without any significant signs of overfitting.

4.1.3 Per-Class Validation

Since the number of images nor instances per image are balanced due to fewer data for the bridled nail-tail wallaby, a stratified train-test split was used. The stratified approach makes sure, that proportions in the training data are preserved in the validation and testing set. The overall split was 70, 20, 10 for training, validation and testing respectively. The data was split randomly.

After model training the model was validated to evaluate the robustness and generalisation of the model with the results below.

Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
all	153	146	0.878	0.874	**0.926**	**0.784**
Bridled-nail-tail wallaby	153	12	1.000	0.759	0.917	0.853
Brush-tailed rock-wallaby	153	21	0.867	1.000	0.990	0.800
Eastern grey kangaroo	153	23	0.714	0.867	0.904	0.785
Red kangaroo	153	24	1.000	0.846	0.938	0.793
Red-necked wallaby	153	20	0.863	0.85	0.899	0.714
Swamp wallaby	153	21	0.984	0.952	0.989	0.826
Western grey kangaroo	153	25	0.718	0.84	0.846	0.720

The total number of images shows the number of images, while the instances indicate the number of *Macropodidae* objects in each image. The total number of instances is lower than the number of images because some images are just background. The following are the results for overall performance (All):

- Precision (P): 0.878,
- Recall (R): 0.874,
- mAP50: 0.926,
- and mAP50-95: 0.784.

The overall performance of the model, as indicated by precision (P), recall (R), mean average precision at IoU threshold 0.50 (mAP50) and mean average precision between IoU thresholds 0.50 to 0.95 (mAP50-95), shows strong capabilities. With a precision of 0.878, the model effectively identifies and correctly classifies the relevant instances among the predicted positives. Furthermore, a recall of 0.874 reflects the model's ability to capture a high proportion of true positives from the total number of actual positives. The mean Average Precision at IoU threshold 0.50, measured at 0.926, highlights the model's accuracy in delineating object boundaries and distinguishing between different classes with high confidence. However, the mean average precision between IoU thresholds 0.50 to 0.95, at 0.784, indicates a slight drop in performance when considering a wider range of IoU thresholds, suggesting some challenges in accurately localising objects under varying conditions.

These results were achieved for individual classes:

- For the *Bridled-nail-tail wallaby* class, the model achieves a perfect precision of 1.000, indicating that all predicted instances are true positives, with a recall of 0.759, demonstrating strong performance in identifying actual instances.
- The *Brush-tailed rock-wallaby* class exhibits impressive precision and recall scores of 0.867 and 1.000, respectively, alongside high mAP50 and mAP50-95 values of 0.990 and 0.800, indicating excellent performance in accurately detecting instances of this class.

- Despite a relatively lower precision of 0.714, the model achieves a respectable recall of 0.867 for the *Eastern grey kangaroo* class, coupled with commendable mAP50 and mAP50-95 values of 0.904 and 0.785, suggesting solid performance in capturing instances of this class.
- The *Red kangaroo* class showcases exceptional precision (1.000) and a good recall (0.846), with robust mAP50 and mAP50-95 scores of 0.938 and 0.793, demonstrating the model's high accuracy in detecting instances of this class.
- For the *Red-necked wallaby* class, the model achieves a precision of 0.863 and a recall of 0.850, along with respectable mAP50 and mAP50-95 values of 0.899 and 0.714, indicating reliable performance in identifying instances of this class.
- The *Swamp wallaby* class demonstrates outstanding performance with a precision of 0.984 and a high recall of 0.952, supported by impressive mAP50 and mAP50-95 values of 0.989 and 0.826, indicating the model's excellent ability to detect instances of this class accurately.
- While the *Western grey kangaroo* class exhibits a precision of 0.718 and a recall of 0.840, the model achieves decent mAP50 and mAP50-95 values of 0.846 and 0.720, suggesting satisfactory performance in identifying instances of this class.

The evaluation indicates that most classes have strong detection performance, with some showing exceptional precision and recall, while others have moderate to good performance.

4.1.4 Model performance

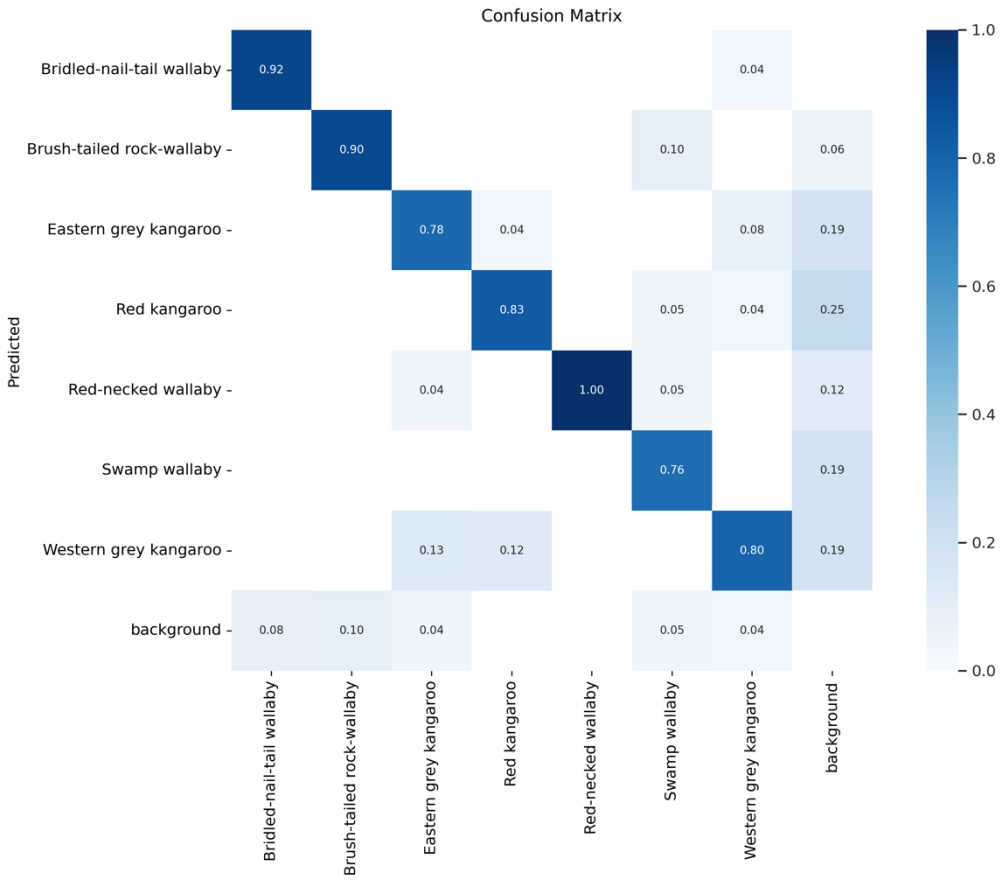
The model shows moderate to strong performance across the different classes of Victorian kangaroos and wallabies.

- It exhibits high precision in distinguishing between various species, accurately identifying specific Victorian kangaroo and wallaby species.
- However, the variability in recall rates across different classes suggests potential challenges in consistently detecting certain species, such as the Eastern grey kangaroo.
- The mAP50-95 metric offers a comprehensive evaluation of the model's performance across a wider range of IoU thresholds, providing valuable insights into its robustness under varying criteria.

Although the model performs well in certain classes, there is still room for improvement to ensure consistent and high-quality detections across all species, especially those with lower recall rates. Further analysis and adjustments could enhance detection accuracy and address challenges effectively. The validation results for one batch are displayed in the image below. It is to add here, that the batch size for model training differs from the batch size of the model validation.

4.1.5 Confusion Matrix

The confusion matrix presents a detailed breakdown of the model's performance by showing the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions across all classes. This evaluation aims to analyse the model's ability to correctly classify instances within each class and identify any patterns of misclassification.



In the confusion matrix, the background columns represent samples that were excluded from the validation and should be neglected when interpreting the species detection.

The confusion matrix indicates that the model is relatively robust. In general, species are detected with probabilities above 0.75. The model has the most difficulty distinguishing between eastern, western grey and red kangaroos. The brush-tailed rock and swamp wallaby are also difficult for the model to distinguish with values above 0.1. The remaining confusions are in the range of 0.04 to 0.08.

4.2 Justification

4.2.1 Resources

Two models were tested during the project, and only the most favourable results are reported here. The efficiency of training a reliable model depends, to some extent, on the available resources. In some cases, CUDA may be required. Although there is growing support for the *Apple M series*, the training protocols need to be adapted, which negatively impacts the training time, making timely model training almost unfeasible. PyTorch is slower on M series chips compared to CPU-based training, especially when using multithreading. To improve performance, it is recommended to set the number of workers to zero. Online resources like Google Colab can be used for computer vision model training. However, setting up larger projects with multiple functions and scripts can be challenging when compared to using an IDE like PyCharm. A YOLOv9 model was trained on Colab using a NVIDIA A100 40GB

GPU. The setup process for YOLO is simple requiring only one script for training, validation, or detection respectively. Before training, hyperparameters are configured in a YAML file.

4.2.2 Data

Optimal model performance requires the use of substantial data, with more than 1,500 images per class and more than 10,000 labelled instances per class recommended. Limited data diversity may hinder performance, as certain types of images, such as those captured by night observation cameras, were excluded from the dataset. Introducing greater diversity in various parameters, including time of day, season, weather conditions and lighting, could potentially improve model robustness.

The reliability of species identification within the dataset, derived from an open online database, remains uncertain. Batch prediction of unused images suggests discrepancies between model predictions and manual observations, indicating potential inaccuracies in species labelling.

4.2.3 Model

YOLOv9 was chosen for its ability to produce satisfactory training results with limited data, due to its remarkable computational efficiency and reduced number of parameters. Even with only 25 training epochs, YOLOv9 outperformed Faster RCNN using the default setup (initial parameter set).

	mAP50	**mAP50:95**
YOLOv9	0.848	0.672
Faster R-CNN	0.706	0.383

In addition, YOLOv9 has significantly faster prediction times. While Faster RCNN took 3-5 seconds for image prediction, YOLOv9 achieves the same task in generally less than half a second.

Controlling hyper-parameters and setting up pre-processing and image augmentation procedures contribute to the ease of training YOLOv9. Setting up PyTorch's preprocessing, and augmentation techniques is not inherently complex, but it requires meticulous tracking to ensure consistent application across training and testing data. In contrast, YOLO simplifies these processes by using a YAML file saved in the training folder. Additionally, PyTorch does not offer the mosaic augmentation technique, which is considered more promising than techniques like CutMix and MixUp that are offered on PyTorch.

Comparing different model architectures requires consistent preprocessing and augmentation techniques. However, some YOLO augmentations are not compatible with PyTorch, limiting direct comparisons. Due to resource constraints, only the results of an initial 25 training epochs were used to select the optimal model architecture. It is plausible that unexplored models could produce better results.

Resource constraints influence parameter selection, with larger models generally achieving higher accuracy at the cost of longer training and inference times. Given the resource constraints, the model was trained using the smallest parameter set available. The introduction of Programmable Gradient Information (PGI) and Generalised Efficient Layer Aggregation

Network (GELAN) in the latest YOLO architecture aims to improve object detection efficiency, although this claim requires further validation over time.

4.2.4 Biology

Cross-species mating introduces complexity that may affect the model's ability to predict species, particularly in hybrid offspring. The human construct of species classification may not fully account for fluidity between species, further complicating model predictions.

4.2.5 Improvements

Significant improvements in model performance were achieved through data refinement, including the removal of images with unnatural objects and those taken at night. Increasing the stringency of the data quality yielded notable improvements.

Tuning hyperparameters could further improve model performance, a process called hyperparameter evolution in YOLO terminology. However, resource and time constraints precluded exhaustive hyperparameter optimisation within the scope of this project.

5 Conclusion

5.1 Reflection

An app has been developed to help identify various species of *Macropodidae* as part of the exploration of Australian wildlife in Victoria. The app is hosted on Google Cloud and can be accessed publicly via https://roovision-6bnwkndfqa-km.a.run.app/. To account for unreliable reception and internet connectivity in remote areas of Australia, offline usage of the app is possible. To achieve this, clone the GitHub repository as it contains all necessary requirements.

One of the main challenges faced in this project was managing resources efficiently while assessing the viability of ongoing training runs. Traditional local training methods often involve long waits, which hinder workflow efficiency and impede progress. Delays in training can be frustrating, especially when desired results are not forthcoming.

A trained model that cannot demonstrate its efficacy serves little purpose. Therefore, integrating the model into an application capable of detecting and classifying kangaroo species is crucial for effectively communicating the project's outcomes to a non-specialized audience. To make it easier for users to access the application, it is essential to deploy it to a hosting provider and provide a URL to showcase its functionality.

Although users can clone a repository and install project dependencies manually, this may be challenging due to variations in personal environments, such as operating systems, hardware configurations, library dependencies or secret accessibility. To mitigate these issues, a decision was made to deploy a dockerized application on Google Cloud Run. This approach requires additional steps beyond the core data science project. These steps include ensuring app security, managing secrets securely, and configuring the docker environment to operate effectively on Google Cloud.

The app currently has limited functionality and lacks some advanced features, such as real-time kangaroo detection using the user's camera. While most of the app's functionality can

be achieved through the Python Flask backend, certain elements must be coded in the front-end using JavaScript or HTML. Local approaches, such as accessing the user's camera using OpenCV, do not work in a dockerized application and must be coded differently. For example, WebRTC can be used in the frontend with JavaScript.

In summary, significant behind-the-scenes work is necessary to ensure a seamless user experience, despite users being unaware of the intricate details involved.

5.2 Improvement

Discussion is made as to how at least one aspect of the implementation could be improved. Potential solutions resulting from these improvements are considered and compared/contrasted to the current solution.

Improving the application means acquiring additional high-quality data from reliable sources capable of confirming the species depicted in the images. Ideally, YOLO should be trained on approximately 1,500 images per class or 10,000 annotations to optimise performance. However, due to limited data availability and computational constraints, a smaller dataset was used in this project.

Manual inspection of misclassified images can provide valuable insights into the model's weaknesses, facilitating targeted adjustments to preprocessing techniques or augmentation strategies. Furthermore, optimisation of hyperparameters, such as the selection of an appropriate optimiser, can increase the efficiency of model training and potentially improve overall performance metrics such as average precision.

Improving the user experience of the application involves incorporating additional functionality and conducting A/B testing to assess the impact of new features. This iterative process ensures that user feedback is effectively incorporated, leading to continuous improvement in the usability and performance of the application.

6 References

Jocher, G. (2023). Hyperparameter evolution. URL: https://docs.ultralytics.com/yolov5/tutorials/hyperparameter_evolution/

Jocher, G., Laughing-q, Qaddoumi, B.(2024). YOLOv9: A Leap Forward in Object Detection Technology. URL: https://docs.ultralytics.com/models/yolov9/

Padilla, R., Passos, W. L., Dias, T. L. B., Netto, S. L., & da Silva, E. A. B. (2021). A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. URL: https://www.mdpi.com/2079-9292/10/3/279

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv preprint arXiv:1506.01497. URL: https://arxiv.org/abs/1506.01497

Wang, C.-Y., Yeh, I.-H., & Liao, H.-Y. M. (2023). YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. URL: https://arxiv.org/abs/2402.13616

7 Appendix

Parameter	Initial	Set 1	Set 2
lr0	0.01	0.001	0.001
lrf	0.01	0.001	0.001
momentum	0.937	0.9	0.95
weight_decay	0.0005	0.0001	0.0003
warmup_epochs	3.0	5.0	4.0
warmup_momentum	0.8	0.85	0.9
warmup_bias_lr	0.1	0.05	0.08
box	7.5	7.0	8.0
cls	0.5	0.6	0.4
cls_pw	1.0	1.2	1.5
dfl	0.7	1.0	1.2
obj_pw	1.0	1.2	1.5
iou_t	0.2	0.25	0.3
anchor_t	5.0	4.0	3.0
fl_gamma	0.0	0.0	0.0
hsv_h	0.015	0.015	0.015
hsv_s	0.7	0.7	0.7
hsv_v	0.4	0.4	0.4
degrees	0.0	0.0	0.0
translate	0.1	0.1	0.1
scale	0.9	0.9	0.9
shear	0.0	0.0	0.0
perspective	0.0	0.0	0.0
flipud	0.0	0.1	0.0
fliplr	0.5	0.5	0.5
mosaic	1.0	0.8	0.9
mixup	0.0	0.0	0.0
copy_paste	0.015	0.015	0.015

Appendix 1: Parameter sets for 25 epoch test training

Set	Class	Images	Instances	P	R	mAP50	mAP50-95
Initial	all	153	146	0.812	0.772	**0.848**	**0.672**
Initial	Bridled-nail-tail wallaby	153	12	1	0.703	0.9	0.756
Initial	Brush-tailed rock-wallaby	153	21	0.79	0.897	0.864	0.674
Initial	Eastern grey kangaroo	153	23	0.818	0.696	0.733	0.567
Initial	Red kangaroo	153	24	0.898	0.792	0.907	0.703
Initial	Red-necked wallaby	153	20	0.583	0.9	0.869	0.689
Initial	Swamp wallaby	153	21	0.882	0.71	0.855	0.684
Initial	Western grey kangaroo	153	25	0.716	0.707	0.809	0.629
1	all	153	146	0.827	0.781	**0.888**	**0.752**
1	Bridled-nail-tail wallaby	153	12	0.88	0.833	0.899	0.839
1	Brush-tailed rock-wallaby	153	21	0.883	0.905	0.963	0.854
1	Eastern grey kangaroo	153	23	0.8	0.697	0.835	0.704
1	Red kangaroo	153	24	0.906	0.804	0.921	0.781
1	Red-necked wallaby	153	20	0.699	0.75	0.841	0.673
1	Swamp wallaby	153	21	0.893	0.795	0.943	0.744
1	Western grey kangaroo	153	25	0.726	0.68	0.811	0.666
2	all	153	146	0.858	0.79	**0.889**	**0.769**
2	Bridled-nail-tail wallaby	153	12	1	0.681	0.928	0.915
2	Brush-tailed rock-wallaby	153	21	0.765	0.952	0.946	0.827
2	Eastern grey kangaroo	153	23	0.719	0.696	0.725	0.656
2	Red kangaroo	153	24	0.925	0.792	0.92	0.808
2	Red-necked wallaby	153	20	0.893	0.835	0.912	0.708
2	Swamp wallaby	153	21	0.987	0.857	0.989	0.802
2	Western grey kangaroo	153	25	0.716	0.72	0.804	0.667

Appendix 2: Results for 25 epoch test training