UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Extending Automatic Animal Classification in Wildlife Environments for Native Species in The Amazon

María José Zurita Mena

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 15 de mayo de 2022

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Extending Automatic Animal Classification in Wildlife Environments for Native Species in The Amazon

María José Zurita Mena

Nombre del profesor, Título académico **Noel Pérez, Ph. D**

Quito, 15 de mayo de 2022

© DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en la Ley Orgánica de Educación Superior del Ecuador.

Nombres y apellidos: María José Zurita Mena

Código: 00200850

Cédula de identidad: 1726794751

Lugar y fecha: Quito, 15 de mayo de 2022

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

El análisis de las imágenes de las cámaras trampa, aunque fundamental para la conservación de los hábitats y las especies, suele ser una tarea manual, larga y costosa. Para el proyecto de cámaras trampa de la Estación de Biodiversidad Tiputini (TBS), la automatización de este proceso permitiría una investigación a gran escala de las especies de este foco de biodiversidad. El objetivo de este trabajo es crear dos modelos de detección y clasificación de objetos a nivel de especie mediante el uso de dos arquitecturas de última generación, YOLOv5 y Faster R-CNN, para dos especies: pecarí de labio blanco y pecarí de collar, utilizando imágenes TBS de 2004 a 2011. El conjunto de datos para los modelos contiene 7.733 imágenes obtenidas tras el aumento de datos. Los modelos se entrenan en el 70\% del conjunto de datos, se evalúan en el 20\% y se prueban en el 10\% de los datos disponibles. El modelo Faster R-CNN alcanzó un promedio de mAP de 0,26 en un umbral de 0,5 IoU y de 0,114 en un umbral de 0,5 a 0,95 IoU, lo que es comparable a los resultados de la prueba original del conjunto de datos Faster R-CNN MS COCO. El mAP medio de YOLOv5 en un umbral de 0,5 IoU es de 0,5525, mientras que su mAP medio en un umbral de 0,5 a 0,95 IoU es de 0,37997. Por lo tanto, el modelo YOLOv5 demostró ser más robusto, teniendo menos pérdidas y un valor mAP general más alto que el entrenamiento de Faster R-CNN y YOLO en el conjunto de datos MS COCO. Este es uno de los primeros pasos hacia la automatización del análisis de proyectos de cámaras trampa TBS y se sugiere que se mejore aún más en términos de optimización de hiper parámetros para un mayor rendimiento y un mayor uso de las especies nativas.

Palabras clave: YOLOv5, Faster R-CNN, deep learning, conservación, cámaras trampa

ABSTRACT

Camera trap images analysis, although critical for habitat and species conservation, is often a manual, time-consuming and expensive task. For the Ecuadorian Tiputini Biodiversity Station's (TBS) camera trap project, the automatization of this process would allow a large-scale research on biodiversity hotspot species. This paper aims to create two deep learning species-level object detection and classification models, using two state-of-the-art architectures, YOLOv5 and Faster R-CNN, for two species: \emph{white-lipped peccary} and \emph{collared peccary}, using TBS images from 2004 to 2011. The dataset for the models contains 7,733 images obtained after data augmentation. The models are trained on 70\% of the dataset, evaluated on 20\% and tested on 10\% of the available data. Faster R-CNN model achieved an average mAP of 0.26 at a 0.5 IoU threshold and 0.114 at a 0.5 to 0.95 IoU threshold, which is comparable to the original Faster R-CNN MS COCO Dataset test's results. YOLOv5's average mAP at a 0.5 IoU threshold is 0.5525, while its average mAP at a 0.5 to 0.95 IoU threshold is 0.37997. Therefore, YOLOv5 model proved to be more robust, having lower losses and a higher overall mAP value than Faster R-CNN's and YOLO's training on the MS COCO dataset. This is one of the first steps towards automating TBS camera trap project analysis and it is suggested that it is further improved in terms of hyper-parameter optimization for higher performance and a higher use of native species.

Key words: YOLOv5, Faster R-CNN, deep learning, conservation, camera traps

TABLE OF CONTENTS

INDEX OF TABLES

INDEX OF FIGURES

I. Introduction

Detailed, up-to-date and accurate information about the location and behavior of animals is necessary for their study and conservation. Considering human threats and their impact on natural habitats, there is an increasing urgency to monitor animal population trends. Trap cameras are an efficient method to do this job, as they allow for a permanent sampling in remote areas in a cheap and discrete way [1] [2]. They are also used for locating endangered species, identifying important habitats and monitoring areas of interest. Nevertheless, to extract information from their images is a manual, expensive and time-consuming task [2]. Moreover, although there is an increasing availability of deep learning models, their practical use for wildlife monitoring is limited, mainly because of the complexity of this technology and its high computing requirements [3]. It has also been a challenge to apply models trained in a specific region to images collected in a different geographical area, because of background changes and the presence of previously unseen species. Additionally, 70\% of images taken do not contain animals, because of a high rate of false triggers [4]. Consequently, to use these images for analysis, one must manually filter, count and classify them, using an expert opinion on the species present on each image.

Tiputini Biodiversity Station (TBS), a remote research center in the Yasuní Reserve in the Ecuadorian Amazon, is surrounded by a biodiversity hotspot. From 2006 to 2016, their camera trap project has produced about 100.000 photos and videos of approximately 70 wild species, some endangered or rare. It is difficult to label such a huge collection, and the task becomes increasingly harder as the database grows. TBS is now able to estimate a total species population by identifying the species in each image, rather than by identifying individuals in them. Although this is a much more efficient process than identifying

individuals within each species, it is still a time-consuming task. The automatization of this process would allow for a large-scale analysis easing manual workload of experts [5].

There have been several attempts to use deep learning to automate camera trap image analysis, through object detection tasks. For example, RetinaNet and Faster R-CNN were used in the UAS dataset, and it was found that the latter had a better performance [6]. Faster R-CNN has also been used to detect and classify individual pigs, to know how much food they would consume daily, in order to optimize their breeding process [7]. There was also a comparison between three state of the art object detection CNN-based algorithms: YOLO v3 (You Only Look Once), SSD (Single Shot Detector) and Faster R-CNN (Region-based Convolutional Neural Networks) using Microsoft's open-source COCO dataset. It concluded that YOLO v3 has an overall better performance and is better suited for real time video analysis, while Faster R-CNN works well with a small dataset that does not require speed in its analysis and SSD has a good balance between speed and accuracy [8]. Additionally, YOLO v5 was implemented for a species-level object detection and classification in a temperate polish forest, Bialowieza. This was the first time this architecture was used for an automated mammal recognition using camera trap images. It achieved an average accuracy of 85\% F1-score for the identification of the 12 most common mammal species in the forest, with a total of 2,659 images with animals [3]. YOLO v5 was used as well for identifying individual feral cats on an unbalanced small dataset [9]. In another study, YOLO v5 was applied for an automated pest detection in protected forests, with a 97\% accuracy rate, as a lightweight and efficient method for IoT devices [10].

The usability of a pre-trained Faster R-CNN + InceptionResNet v2 model has also been tested. It was applied to ten different wild mammal species on color and black \& white images, with a 93\% accuracy rate on classification. It concluded that this rate could improve

with a specific training on European mammal species [11]. Another study detected and classified more than 15 animal species using R-CNN architecture with two different backbones for training: ResNet-101 and InceptionResNet v2; and Inception v3 for classification. Faster R-CNN and YOLO were compared on their performance for identification, quantification and localization of the desert bighorn sheep, using camera trap images [12]. Finally, Cheema and Anand [13] successfully used Faster R-CNN as an object detection framework to detect animal individuals of patterned species, such as zebras, tigers and jaguars.

One threatened species of interest in the Yasuní Reserve is the jaguar (Panthera onca), the largest predator in Central and South America. Although information on tropical felids is key to their conservation, these species are difficult to study, as they have low densities, large home ranges, an elusive nature and a frequent nocturnal behavior. Nevertheless, camera traps have proven to be an efficient technique for this task. Specifically, there are various studies evaluating jaguar populations using this method in different habitats, and a few in lowland wet Amazon forests [14]. There was also a study that used camera traps to examine jaguar prey availability, which allowed a better understanding of jaguar foraging strategies [15]. Influences of increased landscape access in the Yasuní Reserve on the prey community were also studied through a survey of prey and jaguar abundance with camera traps, evaluating prey occurrence and estimating jaguar density. These types of analyses are essential for conservation and management of these wildlife populations, so significant effort should be invested in managing jaguar prey, as a jaguar conservation effort. White-lipped peccary (*Tayassu pecari*) and collared peccary (*Dicotyles tajacu*) are a primary target for hunters and one of the main prey for jaguars. In fact, white-lipped peccary populations are threatened by hunting and habitat loss [16].

A deep learning approach towards automating jaguar's prey detection in camera trap images could allow a better analysis of the jaguar population density and behavior, as well as possible threats to them and their prey. Neural convolutional networks have proven high accuracy scores when recognizing species, not only for high resolution images, but within the camera trap paradigm. Although there are limitations in this approach, such as an under-representation of endangered species and a predisposition towards false positives, various data augmentation techniques can be used to reduce this bias [5]. Therefore, this research will attempt to create a deep learning species-level object detection and classification model of white-lipped peccary and collared peccary using Faster R-CNN and YOLOv5, trained specifically with camera trap images taken from 2004 to 2011 in the Tiputini Biodiversity Station. Not only will this work help classify future camera trap images in the station, but it could also benefit other research centers in The Amazon. This would allow a large-scale analysis of population and habitat trends of white-lipped peccary and collared peccary in these hotspots, in benefit of their conservation and the jaguar's.

II. MATERIALS AND METHODS

A. Database

The pre-processing stage consists of an automated and manual classification of images. MegaDetector software is used for this purpose, a machine learning tool for camera trap images, which is able to find and locate animals in them, from a variety of ecosystems, but not identify them [4]. This reduces the database pre-processing time, filtering images that contain animals from those which contain people or vegetation only. Afterwards, a manual verification is needed to filter out any images wrongly classified as animals. Roboflow's Annotation tool is used for labeling, in order to have a dataset of representative images with bounding box annotations around the animals that ought to be detected. This research draws images taken from 2004 to 2011 from the Tiputini Biodiversity Station camera trap database. Within this time range, there are a total of 25,855 images, of which 22,314 contain animals. Out of these, 3,233 images containing white-lipped and collared peccaries were labeled. The total annotations from this dataset are 7,341.

B. Deep learning models

A deep model is a neural network with deep architecture. It is based on a typical Convolutional Neural Network (CNN) model and its input layer is a 3D matrix of pixel intensity. Object detection involves object localization, which determines the location of an object in the image, and object classification, which determines the object's category. Deep learning-based object detection models work differently from traditional approaches, as they have the ability to learn sophisticated features and a robust training algorithm. Generic object detectors localize an object using a rectangular bounding box to indicate the confidence of the object in the image and to classify it with a label. These detectors are divided into region proposal based detectors, which include Faster R-CNN, and regression/classification based, such as YOLO v5 [17].

The R-CNN model architecture has three modules: region proposal, deep CNN-based features extraction and classification/localization [17]. R-CNN extracts region proposals using selective search, resizes all the extracted crops, passes them through a network, which assigns a category from C+1 categories (including the 'background' label) for a given crop, and finally predicts delta Xs and Ys to shape a given crop. The Selective Search algorithm groups regions together based on pixel intensities. Regions with a minimum of 0.5 Intersection Over Union (IoU) (the intersection area between predicted and ground truth boxes, divided by their union area, as shown in equation 1) [18] are labeled and those with an IoU less than 0.3 are considered background. In the bounding-box regression, the CNN predicts the bounding box parameters (position and size) [19]. Faster R-CNN uses the same process as a R-CNN, but uses another CNN, the Region Proposal Network (RPN), to generate region proposals, and the Fast R-CNN as a detector network, consisting of a CNN backbone, a Region of Interest (ROI) pooling layer, fully connected layers and two sibling branches of classification and bounding box regression. In general, the Fast R-CNN framework consists of a pre-trained CNN and a ROI pooling layer. On the other hand, Faster R-CNN shares the full image convolutional features with the RPN, which is able to predict object bounding box and class confidence scores simultaneously [17]. Faster R-CNN network architecture is shown in Figure 1 [20].

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (1)$$

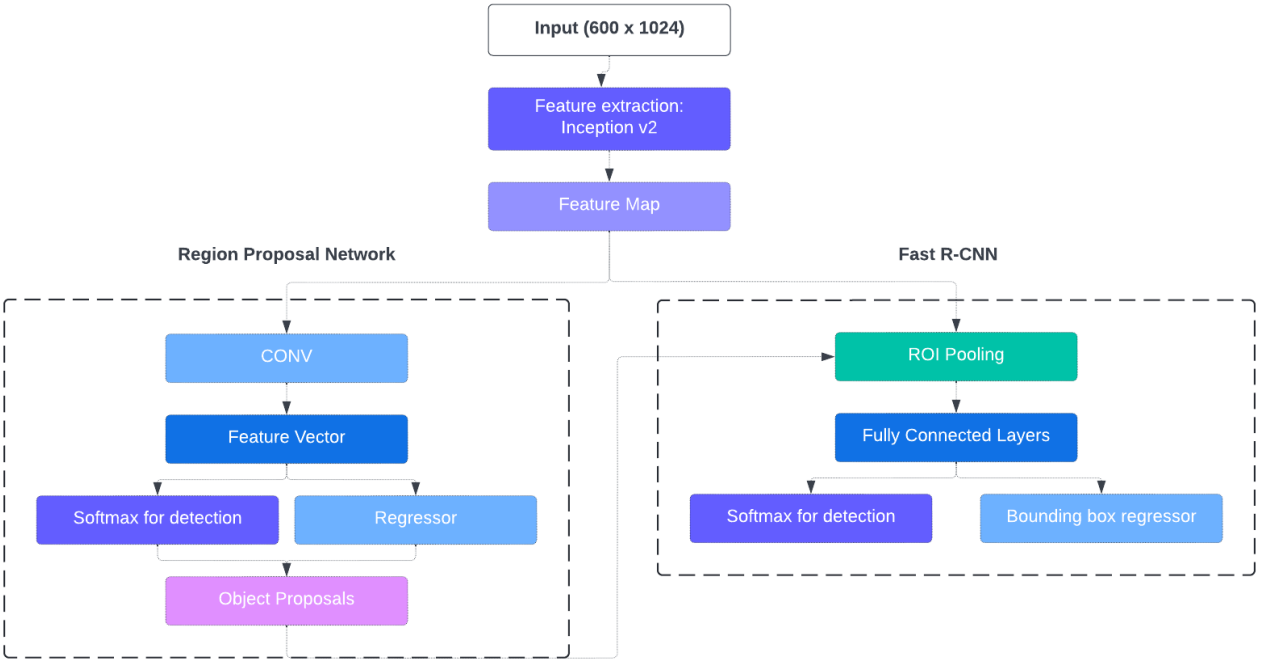

Figure 1: Faster R-CNN's architecture

While the region-proposal-based framework includes various correlated phases and, therefore, time spent handling different components, regression/classification object detectors reduce this time, as they are a single-stage framework based on class probabilities, mapping directly from image pixel to coordinates and global regression/classification. You Only Look Once (YOLO) predicts the bounding box that uses the topmost-feature map and evaluates class probabilities directly. The idea behind this algorithm is to divide the image into S x S grid cells, and each cell is responsible for predicting the center of the object in the grid cell [17]. Each cell will predict B bounding boxes and a confidence score for each one, which is the Intersection Over Union.

Although a cell can predict various bounding boxes and confidence scores, it can only predict one class. Each prediction will have a shape C + B*5, where C is the number of classes, B is the number of predicted bounding boxes and 5 is the number of elements in each box (x, y, width, height, and confidence). Therefore, for the S x S matrix, the shape will be S

x S x (C + B*5) [21]. YOLO's architecture has three components, head, neck and backbone, which work to first extract the image visual features and then classify and limit them. The backbone includes convolutional layers, which detect key features in an image and process them. The neck uses the features of the convolution layers with fully connected layers, in order to predict bounding box probabilities and coordinates. The head is the final output layer [21]. YOLO v4 uses CSPDarknet53 as a backbone and Path Aggregation Network (PAN, an instance segmentation method) for parameter aggregation. YOLO v3 head is used in YOLO v4. YOLO v5 is very similar to YOLO v4, with a few differences. While YOLO v5 is based on the PyTorch framework, YOLO v4 was released in the Darknet framework [22]. This framework allows a 16 bit floating point precision, which improves the model's inference time. YOLO v5 is based on the YOLO architecture, which consists of four main parts: input, backbone, neck and output, as seen in Figure 2. The input terminal involves data preprocessing (such as mosaic data augmentation and adaptative image filling). The backbone network uses a cross-stage partial network (CSP) and spatial pyramid pooling (SPP) to extract feature maps of different sizes from the input image by multiple convolution and pooling. Specifically, BottleneckCSP is used, which reduces calculations and increases the inference speed, while the SPP structure extracts features from different scales for the same feature map and generates three-scale feature maps, which improves the detection accuracy. In the neck network, on the other hand, feature pyramid structures of FPN and PAN are used. FPN conveys strong semantic features from the top feature maps into the lower feature maps, while PAN conveys strong localization features from lower feature maps to higher feature maps. The head is the final detection step, used to predict targets of different sizes on feature maps [23]. YOLO v5's network structure is shown in Figure 2. Glenn et. al in [24] list several recommendations for best performance for YOLOv5, i.e., at least 1,500 images per class and at least 10,000 labeled objects per class.

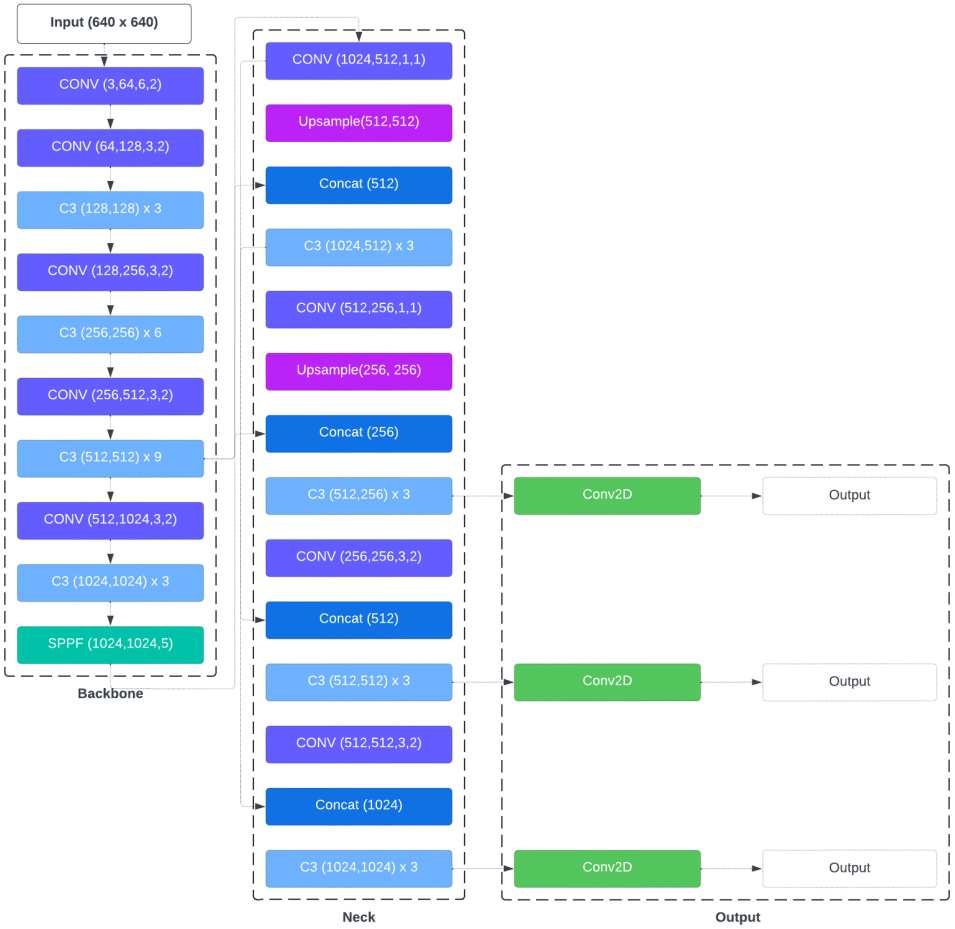

Figure 2: YOLOv5's architecture

C. Proposed method

One model for each object detector type is trained and evaluated for two classes: ``Taypec" (white-lipped peccary) and ``Taytaj" (collared peccary). These models are to be compared in terms of performance between them and the state-of-the-art results, in order to find improvement points and further tune the hyperparameters and increase the size of the dataset. This research focuses on setting up a baseline in order to pursue hyper parameter optimization in the future.

Faster R-CNN input is resized to a minimum of 600px for its shortest side and a maximum of 1024 for its larger side, keeping the aspect ratio. The feature extractor used is

Inception V2, using batch normalization and producing an output stride of 16px. For the Region of Interest (ROI) pooling layer, a bilinear interpolation based technique is used. For Max Pooling, in this layer, the kernel size is 2. The batch size used is 1 and the learning rate is constantly 0.0002. A Stochastic gradient descent (SGD) momentum optimizer is used as well. ReLU activation function is used for hidden layers and Softmax for the output layer. Gradient clipping method is used, using norm as a clipping threshold. Random horizontal flip is used in addition to those data augmentation techniques described in the preprocessing stage (please refer to section II-D1). The total steps \footnote{An epoch is an iteration over all training data, while a step is a gradient update, in which as much data as determined as batch-size is processed. An epoch has as many steps as the number of training images divided by the batch-size.} for the model's training are 180K.

YOLO v5's model is trained with an initial learning rate of 0.001 and a final learning rate of 0.1, using a once cycle policy. A SGD momentum optimizer is used too. The activation functions used are Leaky ReLU (for hidden layers) and Sigmoid (for the detection final layer). The batch size is the largest the hardware allows, 64, and the total epochs for the training stage is set to 300. YOLOv5s architecture is used, with 191 layers in total.

D. Experimental setup

1) Data processing

In this initial research, a total of 3,233 images containing two species (White-lipped peccary and collared peccary) have been labeled. Instances of all classes in all images are labeled and labels closely enclose each object, without spaces between objects and their bounding boxes. On average, there are 2.3 annotations per image across the two classes.

Also, to make sure there is image variety in the dataset, images from different times of day, weather, lighting and angles are used. Background images have also been added to the dataset to reduce False Positives. To prevent train/test bleed, duplicate images are removed

automatically by Roboflow. Roboflow's Dataset Health Check tool shows the average image size is 2.27 mp and the median image ratio is 1840 x 1232.

Roboflow's preprocessing tools are used to apply image transformations to all images. Auto-orient is used in order to strip the images of their EXIF data and standardize pixel ordering. Images are resized as well to a 416 x 416 dimension.

Data augmentation is also used, so the model generalizes better through multiple variations of each source image. Several bounding box level augmentation techniques have been used as well, which alter the content within each bounding box. Horizontal and vertical flips (at image and bounding-box levels) were added to help the model be less sensitive to subject orientation. Random crop technique (at image and bounding-box levels) is also used, in order to create a random subset of each image, increasing resilience to subject translations and camera position, which helps the model recognize animals that may not always be completely in frame or constantly at the same distance from the camera, which is the case for most of the camera trap images from the database. Gray-scale augmentation is also used, as it allows to increase training variance but not discard color information during testing. Hue augmentation is used as well, as it randomly alters color channels of each image, helping the model to consider alternative color schemes for objects and scenes, which is useful in this dataset as the animals and their backgrounds change colors depending on the time of the day or weather and even the camera's hardware state. Because of this, saturation augmentation is applied as well, adjusting how vibrant each image is. Brightness and exposure augmentation techniques are also applied (at image and bounding-box levels), helping the model be more resilient to lighting and camera setting changes. Several camera trap images do show a change in camera focus, so a random Gaussian blur augmentation is used as well (at image and bounding-box levels), helping the model be more resilient to camera focus. There are many images as well that contain animals located behind other objects, so cutout

augmentation helps the model detect these better by adding randomly generated black boxes on top of the images and, by doing so, encouraging the model to learn more distinguishing features about each class of object. Additionally, bounding-level noise variations help the model be more resilient to camera artifacts. With this step, the dataset contains 7,733 images.

2) Training and test sets

After labeling the images, Roboflow helps split the dataset into a training set (6,800 images), a testing set (340 images) and a validation set (643 images). As a rule of thumb, 70\% of the dataset is allocated to the training set, 20\% to the validation set and 10\% to the test set. This allocation ratio helps the training set have enough data to learn, the validation set get a proper tuning of the model and the testing set inform its final accuracy.

For Faster R-CNN, TFRecord files are needed. Roboflow's Annotation Tool is used as well to create two separate datasets and their respective TFRecord files.

On the other hand, for YOLOv5, files in YOLO V5 PyTorch format are exported. This zip is extracted, and its train and test folders contain the dataset used for the model’s training, validation and detection.

3) Model configuration

The Faster RCNN model's hyperparameters are determined as follows. The number of classes is 2. The number of steps for the model's training is 180K.

For the RPN, the input image is fed to the backbone CNN. The minimum size constraint is 600 and the maximum is 1024. The model resizes the input image satisfying these constrains and keeping the aspect ratio (the shortest side is at least 600px and the longer side doesn't exceed 1024px).

The feature extractor used is Inception V2, the second generation of the Inception CNN architecture, which uses batch normalization, removing local response normalization. Batch normalization allows the use of higher learning rates, reduces the need for a careful

parameter initialization, acts as regularization and, consequently, reduces the training steps needed [25].

The output stride is 16, which determines the output features size, as the consecutive pixels in the backbone output features correspond to points 16 pixels apart in each input image.

The anchors placed on the input image for each location on the output feature map, which show possible objects in different sizes and aspect ratios at each location, are set as well. The anchor generator's scales are set to [0.25, 0.5, 1.0, 2.0], the aspect ratios to [0.5, 1.0, 2.0], the height stride and the width stride to 16. The number of proposals for the first stage (region proposal network) is set to 300.

Non Max Supression (NMS), a greedy algorithm, is used to reduce the number of bounding boxes each object gives rise to, as, for each class, it checks for the IoU value between all the bounding boxes and, according to an IoU threshold, determines which refer to the same object and discards the lowest confidence score box. NMS loops over all the classes [26]. The score threshold for NMS for the RPN is set to 0, as recommended for Faster R-CNN. Boxes with a lower score than this number are suppressed. The IoU threshold for NMS on the boxes predicted by the RPN is set to 0.7.

For the Region of Interest (ROI) pooling layer, ROI Align, a bi-linear interpolation based technique, is used to crop a patch from a feature map based on a region proposal and resize it in order to extract a small feature map from each ROI [27] [28]. The initial bi-linear interpolation's output size is set to 14. Additionally, the kernel size (pool size or filter size) of the max pooling operation on the cropped feature map during ROI pooling is set to 2 with a stride of 2.

For the Second Stage Box Predictor, dropout's regularization technique is not used, so the value for the dropout keep probability in a hidden layer is set to 1.0. This means that the probability at which outputs of the layer are retained is 1.0.

As for training, the batch size is set to 1, as in one step the model feeds one image at a time.

The momentum optimizer is used with a value of 0.9. This is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions, gaining faster convergence and reduced oscillation [29].

The learning rate is set to 0.0002, which is the Faster RCNN Inception V2 model's original training learning rate.

Gradient clipping is used in order to clip the error derivative to a threshold during backward propagation, in order to avoid exploding gradients. The gradient clipping-by-norm variable is used, with a threshold of 10.0. This means the gradients are clipped multiplying the unit vector of the gradients with the threshold.

One augmentation option is used: random horizontal flip, which randomly flips each image with a 50\% chance.

The proposed YOLO v5 model has several parameters: img (input image size), batch (batch size), epochs (number of training epochs), data (the dataset location), weights (a path to weights to start transfer learning from) and the cache (cache images for faster training). YOLO v5's documentation explains that most of the time there is no need to change the model or training settings, provided that the dataset is sufficiently large and well labelled. However, several recommendations for best performance are listed [24].

YOLO v5's data-configurations file, which describes the dataset parameters, is edited, providing the number (2) and the names of the classes ('Taypec', 'Taytaj'). The

model-configurations file, which describes the model architecture, is edited as well providing the number of classes.

The initial learning rate is 0.01 and the final learning rate is 0.1. The once cycle policy for learning rates is applied. This means that in one learning cycle (epoch) there are 2 steps of equal length where in the first step one goes from a lower learning rate to the higher learning rate and then back to the lower learning rate in step 2. In the last iterations, the learning rate is set below the lower learning rate value [30]. This helps the speed of training and the discovery of the maximum practical learning rate.

The SGD momentum is 0.937. The warmup epochs are set to 3, which are the number of epochs in which the model will adjust to the dataset. The IoU threshold is set to 0.2. Several augmentations are also used in the model, such as image mosaic (1.0 probability), image horizontal flip (0.5), image scale (+/- 0.5 gain), image translation (0.1), image HSV-Hue augmentation (0.015), image HSV-Saturation augmentation (0.7) and image HSV-Value augmentation (0.4).

When running the training phase, the input image size is set to 640 x 640, the batch size is set to 64, the number of training epochs is set to 300 and the weights are initialized from pretrained weights, which is recommended for small to medium size datasets. Then, the model's accuracy is validated over the testing set, with a batch size of 64.

The model is finally used for detection, using the testing set, setting the confidence threshold to 0.6 and using the augmented inference parameter (test-time augmentations or TTA), in order to improve the accuracy.

4) Assessment metrics

Mean Squared Error (MSE) metric is used, as it calculates the average of the squared differences between inferred and actual values, as shown in equation 2. Cross-Entropy (CE) loss is used as well, summarizing the average difference between actual and predicted

probability distributions for predicting class 1. In binary classification, cross entropy, binary cross entropy (BCE), is calculated with equation 3, while in multi class classification it is calculated with equation 4.

$$\text{MSE} = \sum_{i=1}^{D} (x_i - y_i)^2 \quad (2)$$

$$\text{BCE} = -(y \log(p) + (1 - y) \log(1 - p)) \quad (3)$$

$$\text{CE} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \quad (4)$$

YOLOv5 loss function is composed of the bounding box regression loss (Mean Squared Error), the objectness loss or confidence of the object presence (Binary Cross Entropy) and the classification loss (Cross Entropy). The bounding box regression (localization loss) is the loss due to a box prediction nor covering an object, the objectness is the loss due to a wrong box-object IoU prediction and the classification is the loss due to deviations from predicting

“1” for the correct classes and “0” for all the other classes for the object in the box. These loss functions are computed for the training and the validation set. In this case, the results from the latter will be discussed. Faster RCNN loss function uses the same losses: classification, objectness and localization. However, it calculates localization loss for both the RPN and the Box Classifier.

Precision (P) and recall (R) are measured as well, the first quantifying how many positive class predictions actually belong to the positive class and the latter how many positive class predictions were made out of all positive instances. In YOLOv5, it describes how much of the bbox predictions are correct or how accurate the predictions are, as shown in equation 5. On the other hand, for this model, recall measures how much of the true bbox were correctly predicted, using equation 6.

$$\mathrm{P} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (5)$$

$$\mathrm{R} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (6)$$

The Mean Average Precision (mAP), as an accuracy function (see equation 8), is also used as a metric, as it incorporates the trade-off between precision (prediction accuracy) and recall (number of predictions) and considers false positives (FP) and false negatives (FN). Average Precision is the weighted mean of precisions at each threshold, having the increase in recall from the previous threshold as the weight. It is calculated with equation 7. Mean Average Precision is the average of AP of each class. It is calculated at IoU, calculated with equation 1, threshold of 0.5 and over different IoU thresholds (0.5 to 0.95). Calculating mAP over an IoU threshold range avoid the ambiguity of arbitrarily choosing an optimal IoU threshold. This metric compares the ground-truth bounding box to the predicted box. The higher the score, the more accurate the model is at prediction.

$$\mathrm{AP} = \sum_{n} (R_n - R_{n-1}) P_n \quad (7)$$

$$\mathrm{mAP} = \frac{1}{N} \sum_{n} AP_n \quad (8)$$

III. RESULTS AND DISCUSSION

A. Performance evaluation

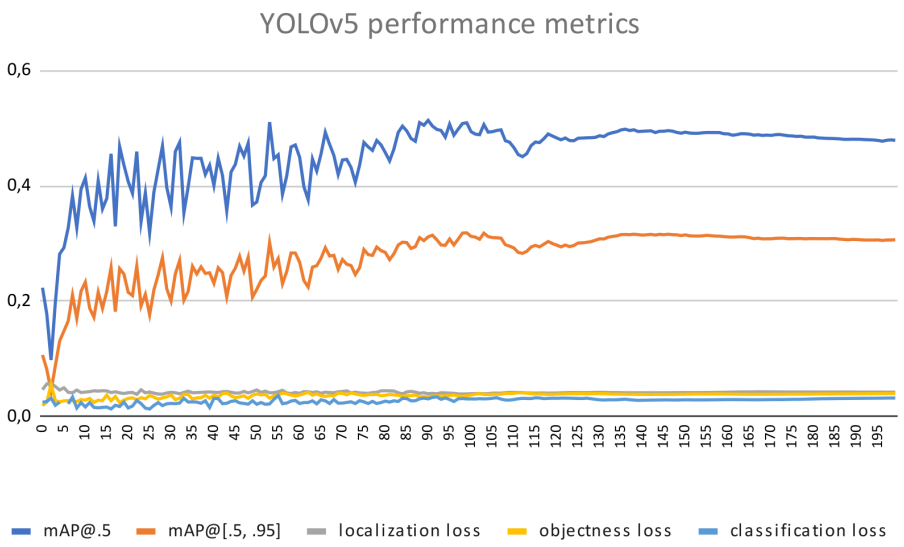

Figure 3: YOLOv5 metrics

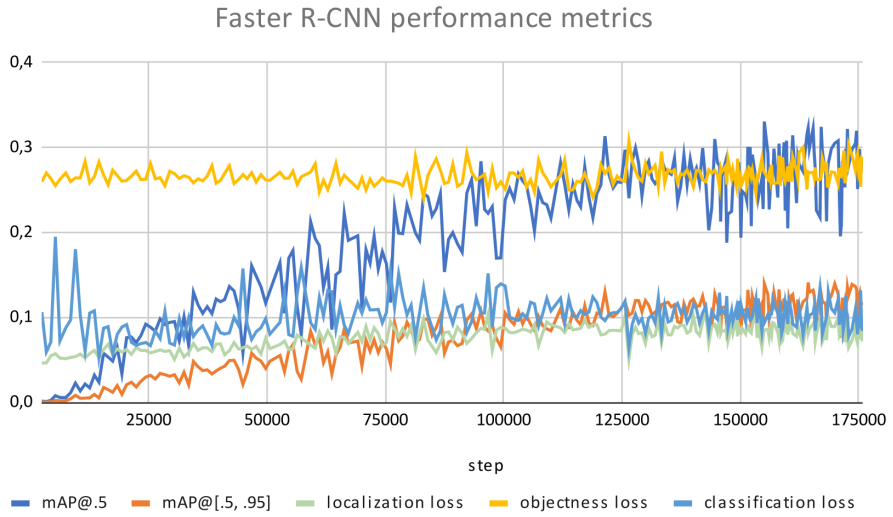

Figure 4: Faster R-CNN metrics

Overall, as seen in Figures 3 and 4, YOLOv5 shows a better mAP over classes averaged over an IoU threshold range of 0.5 to 0.95 than Faster-RCNN, which suggests it has a better accuracy and is a more robust model. The graph shows that an early stage (less than

20 epochs) the value increases significantly and then fluctuates. At 130 epochs approximately, the fluctuation decreases. There is an overall convergence effect observed. On the other hand, Faster R-CNN shows a slower increase in mAP over each step and fluctuations that do not affect the increasing trend.

Although mAP over classes averaged over an IoU of 0.5 for Faster R-CNN is still lower than YOLOv5, at this IoU threshold it does show better values than at the range from 0.5 to 0.95. This shows that at higher threshold values it is less accurate.

For Faster R-CNN, the mAP for small, medium and large objects shows there is a better accuracy when the model detects larger objects, as mAP for small objects does not show convergence and has large fluctuations, especially from 80K to 180K steps, while mAP for medium and large objects does show convergence and less significant fluctuations. This is true for Faster R-CNN as an architecture itself, as it still faces a challenge in detecting small objects. Therefore, improvements like introducing shallow features into the backbone network and ensuring sufficient spatial information for detecting small objects [31] could help improve this value.

It was also observed that Faster R-CNN's average recall with 100 detections per image has less fluctuations and is higher as the object size increases. This means that the larger the object, the more accurate the model is while predicting the object class. One can observe that between 10K to 30K steps the recall value significantly increases and then has small fluctuations after 70K steps.

Faster R-CNN classification's loss does not show convergence, as YOLOv5's does, but remains less than 0.15 for most of the training. The classification loss for YOLOv5 shows values under 0.04. The loss fluctuates until 120 epochs and then stabilizes approximately to 0.0325. This shows that deviations from predicting ``1" for the correct classes and "0" for all

the other classes for the object in the box are generally low for both models, but rarely happen in YOLOv5's model. This is probably due to the dataset having just two classes.

The Box Classifier localization loss increases over each step (with fluctuations after 80K steps), but, overall, the values are below 0.1. The RPN localization loss decreases until about 120K steps, when it increases and fluctuations become more significant. This value is constantly below 0.110. For YOLOv5, the value initially fluctuates (until about 120 epochs) and then stabilizes to approximately 0.0425. It is mostly below 0.025. Therefore, YOLOv5 proves to be better at having box predictions cover the objects in the images.

For Faster R-CNN, the objectness loss does not converge, seems to increase and fluctuates significantly after each step. This means that the model does not perform well at box-object IoU prediction. It is also observed that for YOLOv5 there is an initial spike in this value, it decreases, then slowly increases and fluctuates from 10 to 70 epochs and finally stabilizes at approximately 0.04. For YOLOv5, it is shown that the objectness loss value is mostly below 0.0425, while for Faster R-CNN it is between 0.25 to 0.29 approximately.

It was observed that Faster R-CNN's model had a better performance when detecting objects that were alone in the image, as it is shown in Figures 5, 6 and 8. However, for images with multiple objects, often overlapping between each other, it usually detected the animals that were less covered by others, as shown in Figure 7. This suggests that it is challenging for the model to discriminate instance boundaries, given that the features of the instances overlap between each other.

Figure 5: Faster R-CNN's detection result 1

Figure 6: Faster R-CNN's detection result 2

Figure 7: Faster R-CNN's detection result 3

Figure 8: Faster R-CNN's detection result 4

Figure 9: YOLOv5's detection result 1

Figure 10: YOLOv5's detection result 2

Figure 11: YOLOv5's detection result 3

Figure 12: YOLOv5's detection result 4

As seen in Figures 9, 10, 11 and 12, YOLOv5 model tends to label images that contain objects as background when the weather and lighting conditions make the contrast between the object and the background less visible. This could be improved by adding background images to the dataset, so the model could learn to differentiate it better. Moreover, color augmentations could be tuned in order to help the model be less sensitive to these changes. As for crowded images, it seems to perform better than Faster R-CNN, inferring several objects in images, even when there are overlapping.

These results suggest that, for improving the overall accuracy of Faster R-CNN, it may be useful to increase the number of steps, as its mAP value increased slowly over each step. Another method would be to use more augmentation options. Also, although Non Max Suppression is used, it may be improved by increasing the score threshold, in order to do a minimum filtering of proposals, which would help stabilize the training and have a better convergence. IoU threshold for NMS could also be lowered, given that this dataset does show a dense distribution of objects in each image, as the animals tend to appear in large numbers and close together. Repulsion Loss [32] and AggLoss [33] are two modified losses that may help reduce the sensitivity of results to the NMS IoU threshold, ensuring tighter bboxes. Additionally, there are several attempts to redesign NMS in order to handle occlusion, such as Adaptative NMS [34].

Learning rate scheduling could also help improve the overall model accuracy, as it has done for YOLOv5 using a Once Cycle Policy, and its training stability.

One could also improve accuracy using the alternating optimization algorithm, which allows RPN and Fast R-CNN to be trained to share convolutional features. This is a more accurate method than approximate joint training, because the latter ignores the weight's gradients with respect to the region proposals. With the alternation optimization, the RPN is first trained to generate region proposals, while the weights of the shared convolutional layers are initialized based on a pre-trained model on ImageNet and the other weights of the RPN are initialized randomly. After the region proposals boxes are produced, the weights of the RPN and shared convolutional layers are tuned. The proposals by the RPN are used to train the Fast R-CNN. The weights of the shared convolutional layers are initialized with the tuned weights by the RPN and the other Fast R-CNN weights are initialized randomly. The weights of Fast R-CNN and the shared layers are tuned while the fast R-CNN is trained. Then, the

tuned weights in the shared layers are used again to train the RPN and the process is repeated [35].

As a preprocessing step, analysing the geometry of the image objects would also be useful, as it would allow a better tuning of the anchor aspect rations and it would show the distribution for height to width and width to height ratios. Moreover, increasing the dataset adding more images would also be beneficial for both models' performance, as this type of dataset displays a somewhat high intra-class variety, having different animal sizes, positions and angles.

B. State of art based comparison

Faster R-CNN's performance was tested for MS COCO dataset with 300 proposals. For the validation test, it achieved a 0.415 mAP@.5 score and a 0.212 mAP@[.5, .95] [35]. On the other hand, YOLOv5 achieved a 0.54 mAP@.5 score and a 0.35 mAP@[.5, .95] [24]. As shown in table 1, for this experiment, Faster R-CNN's mAP@.5 score is 0.26 and mAP@[.5, .95] is 0.114, while YOLOv5's mAP@.5 score is 0.5525 and mAP@[.5, .95] is 0.37997.

Table 1: Metrics Comparison

Model	Average mAP@.5	Average mAP@[.5, .95]
YOLOv5	0.5525	0.37997
Faster R-CNN	0.26	0.114

In terms of speed (FPS), it is difficult to make comparisons, as they are usually measured at different mAP values. However, a comparison of accuracy and speed tradeoff may be easier to determine, depending on the application. In terms of Faster R-CNN, the feature extractor used does change the overall mAP achieved in a certain GPU time [24]. In this experiment, Inception V2 was used, which gives one of the lower accuracies in

comparison to other feature extractors [24]. On the other hand, it has been shown that Inception Resnet V2 gives the highest accuracy at 1 FPS [24]. Therefore, the overall mAP achieved in this experiment by Faster R-CNN could be further improved by using Inception Resnet V2, as it does not require more GPU time to do this.

In general, Faster R-CNN has shown a low performance on small objects in comparison to other methods [24]. This was seen in this experiment, as it achieved higher mAP values for medium and large objects.

IV. CONCLUSIONS AND FUTURE WORK

In a nutshell, this experiment managed to train and evaluate two state-of-the-art object detection and classification models for the white-lipped peccary and the collared peccary from 7,733 camera trap images from the Tiputini Biodiversity Station. In terms of performance, Faster R-CNN achieved an average mAP of 0.26 at a 0.5 IoU threshold and 0.114 at a 0.5 to 0.95 IoU threshold, which is comparable to the original Faster R-CNN MS COCO Dataset test's results. On the other hand, YOLOv5 achieved an average mAP of 0.5525 at a 0.5 IoU threshold and 0.37997 at a 0.5 to 0.95 IoU threshold. Therefore, YOLOv5 model proved to be more robust, having lower losses and a higher overall mAP value than Faster-RCNN's and YOLO's training on the MS COCO dataset. Although several optimization techniques were used in each model to improve performance, it is recommended that the dataset is augmented and both models hyperparameters are further tuned. Moreover, for a more extensive use of these models, it is suggested to train more classes (Tiputini species).

ACKNOWLEDGEMENT

Thanks to the Tiputini Biodiversity Station and the Applied Signal Processing and Machine Learning Research Group of USFQ for providing the database and the computing infrastructure (NVidia DGX workstation) to implement and execute the developed source code, respectively. Personal thanks to my family and professors.

Bibliographic references

[1] F. Trolliet, C. Vermeulen, M.-C. Huynen, and A. Hambuckers, "Use of camera traps for wildlife studies: a review," *Biotechnologie, Agronomie, Societ´ e et Environnement ´* , vol. 18, no. 3, pp. 446–454, 2014.

[2] N. M. Sadegh, N. Anh, K. Margaret, and S. Alexandra, "Palmer meredith s., packer craig, clune jeff. automatically identifying, counting, and describing wild animals in camera-trap images with deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 25, pp. E5716–E5725, 2018.

[3] M. Choinski, M. Rogowski, P. Tynecki, D. P. Kuijper, M. Churski, and J. W. Bubnicki, "A first step towards automated species recognition from camera trap images of mammals using ai in a european temperate forest," in International *Conference on Computer Information Systems and Industrial Management*. Springer, 2021, pp. 299–310.

[4] S. Beery, D. Morris, and S. Yang, "Efficient pipeline for camera trap image review," *arXiv preprint arXiv:1907.06772*, 2019.

[5] Y. He and Q. Weng, *High spatial resolution remote sensing: data, analysis, and applications*. CRC press, 2018.

[6] J. Peng, D. Wang, X. Liao, Q. Shao, Z. Sun, H. Yue, and H. Ye, "Wild animal survey using uas imagery and deep learning: modified faster r-cnn for kiang detection in tibetan plateau," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 169, pp. 364–376, 2020.

[7] Q. Yang, D. Xiao, and S. Lin, "Feeding behavior recognition for group-housed pigs with the faster r-cnn," *Computers and electronics in agriculture*, vol. 155, pp. 453–460, 2018.

[8] S. Srivastava, A. V. Divekar, C. Anilkumar, I. Naik, V. Kulkarni, and V. Pattabiraman, "Comparative analysis of deep learning image detection algorithms," *Journal of Big Data*, vol. 8, no. 1, pp. 1–27, 2021.

[9] Z. Yang, R. Sinnott, Q. Ke, and J. Bailey, "Individual feral cat identification through deep learning," in *2021 IEEE/ACM 8th International Conference on Big Data Computing, Applications and Technologies (BDCAT'21)*, 2021, pp. 101–110.

[10] K. Shim, A. Barczak, N. Reyes, and N. Ahmed, "Small mammals and bird detection using iot devices," in *2021 36th International Conference on Image and Vision Computing New Zealand (IVCNZ)*. IEEE, 2021, pp. 1–6.

[11] C. Carl, F. Schonfeld, I. Profft, A. Klamm, and D. Landgraf, "Automated" detection of european wild mammal species in camera trap images with an existing and pre-trained computer vision model," *European Journal of Wildlife Research*, vol. 66, no. 4, pp. 1–7, 2020.

[12] M. Vargas-Felipe, L. Pellegrin, A. A. Guevara-Carrizales, A. P. Lopez-Monroy, H. J. Escalante, and J. A. Gonzalez-Fraga, "Desert bighorn sheep (ovis canadensis) recognition from camera traps based on learned features," *Ecological Informatics*, vol. 64, p. 101328, 2021.

[13] G. S. Cheema and S. Anand, “Automatic detection and recognition of individuals in patterned species,” in Joint European *Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017, pp. 27–38.

[14] “Yasuni.”

[15] M. Weckel, W. Giuliano, and S. Silver, “Jaguar (panthera onca) feeding ecology: distribution of predator and prey through time and space,” *Journal of zoology*, vol. 270, no. 1, pp. 25–30, 2006.

[16] S. Espinosa, G. Celis, and L. C. Branch, “When roads appear jaguars decline: Increased access to an amazonian wilderness area reduces potential for jaguar conservation,” *PloS one*, vol. 13, no. 1, p. e0189740, 2018.

[17] L. Aziz, M. S. B. H. Salam, U. U. Sheikh, and S. Ayub, “Exploring deep learning-based architecture, strategies, applications and current trends in generic object detection: A comprehensive review,” *IEEE Access*, vol. 8, pp. 170 461–170 495, 2020.

[18] M. A. Rahman and Y. Wang, “Optimizing intersection-over-union in deep neural networks for image segmentation,” in *International symposium on visual computing*. Springer, 2016, pp. 234–244.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[20] Z. Deng, H. Sun, S. Zhou, J. Zhao, L. Lei, and H. Zou, “Multi-scale object detection in remote sensing imagery with convolutional neural networks,” I*SPRS journal of photogrammetry and remote sensing*, vol. 145, pp. 3–22, 2018.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[22] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” *arXiv preprint arXiv:2004.10934*, 2020.

[23] Z. Li, X. Tian, X. Liu, Y. Liu, and X. Shi, “A two-stage industrial defect detection framework based on improved-yolov5 and optimizedinception-resnetv2 models,” *Applied Sciences,* vol. 12, no. 2, p. 834, 2022.

[24] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, TaoXie, J. Fang, imyhxy, K. Michael, Lorna, A. V, D. Montes, J. Nadar, Laughing, tkianai, yxNONG, P. Skalski, Z. Wang, A. Hogan, C. Fati, L. Mammana, AlexWang1900, D. Patel, D. Yiwei, F. You, J. Hajek, L. Diaconu, and M. T. Minh, “ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference,” Feb. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6222936

[25] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[26] N. Bodla, B. Singh, R. Chellappa, and L. Davis, “Improving object detection with one line of code. arxiv 2017,” *arXiv preprint arXiv:1704.04503*.

[27] C. Cao, B. Wang, W. Zhang, X. Zeng, X. Yan, Z. Feng, Y. Liu, and Z. Wu, “An improved faster r-cnn for small object detection,” *IEEE Access*, vol. 7, pp. 106 838–106 846, 2019.

[28] K. He, G. Gkioxari, P. Dollar, and R. Girshick, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[29] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[30] L. N. Smith, “A disciplined approach to neural network hyperparameters: Part 1–learning rate, batch size, momentum, and weight decay,” *arXiv preprint arXiv:1803.09820*, 2018.

[31] L. Tang, F. Li, R. Lan, and X. Luo, “A small object detection algorithm based on improved faster rcnn,” in *International Symposium on Artificial Intelligence and Robotics 2021*, vol. 11884. SPIE, 2021, pp. 653–661.

[32] X. Wang, T. Xiao, Y. Jiang, S. Shao, J. Sun, and C. Shen, “Repulsion loss: Detecting pedestrians in a crowd,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7774–7783.

[33] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, “Occlusion-aware rcnn: detecting pedestrians in a crowd,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 637–653.

[34] S. Liu, D. Huang, and Y. Wang, “Adaptive nms: Refining pedestrian detection in a crowd,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 6459–6468.

[35] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *Advances in neural information processing systems*, vol. 28, 2015.