Animal Detection and Recognization through images to study wildlife

Nam C. Nguyen
Advanced Program in Computer Science
Ho Chi Minh University of Science, VNU-HCM
Ho Chi Minh, Viet Nam
ncnam@apcs.vn

Quynh-Chau V. Long
Advanced Program in Computer Science
Ho Chi Minh University of Science, VNU-HCM
Ho Chi Minh, Viet Nam
lvqchau@apcs.vn

Hien N. Tran
Advanced Program in Computer Science
Ho Chi Minh University of Science, VNU-HCM
Ho Chi Minh, Viet Nam
tnhien@apcs.vn

***Abstract*—**Educating students' awareness on the animal world is a necessity to prevent the extinction of rare species. This motivates the authors to propose a way that is capable of classifying animals, and showing their habitats, their traits instantly. Our main objective is to detect, recognize and quantify animal species through images and videos. To be efficient in a realistic environment, the authors choose a collection of over 1 million camera trap images in an African savanna. The authors' current approach is using pre-trained models such as Faster R-CNN Inception Resnet_v2, and SSD Mobilenet_v1 on a different dataset to evaluate them. Over a period of 3 training times and testing on each model, Faster R-CNN Inception Resnet_v2 outperforms SSD Mobilenet_v1, obtaining the average accuracies of 00%. Our proposed work is potential to developed further to be applied in applications for learning and traveling in wildlife.

***Keywords*—**Animal Detection, CNN, Computer Vision, Tensorflow, Serengeti Snapshot

1 Introduction

Our planet Earth, with over 5 billion species of animal, is dependent on plants and animals for sustaining the ecosystem. However, due to overpopulation, habitat destruction, poaching and other causes, there has been a great decline in the numbers of animal. Understanding the nature of these species is a critical step to develop appropriate strategies to conserve them.

This motivates our proposal of developing a detection application to easily recognize animal through images and videos to study wildlife.

Gold Standard Serengeti Snapshot is the dataset chosen for our approach. This dataset is the results of annotated cameras placed in an African savanna. It contains over 3.2 million image sequences of 46 mammal species. In this paper, we use the labelled GSSS dataset [6] with bounding boxes to train the existing methods.

There are two main approaches for animal detection, one is segmentation and another one is general identification. With identification, we have [4] and [8], and with segmentation, there are [1], [2] and [3].

Our method is reapplying the existing method and adjusting number of object classes and some parameterss. We train the GSSS Dataset on Faster Region-Convolutional Neural Network and Single Shot Multi-Box Detector-Mobilenet_v1 to get the models prepared. Faster R-CNN is considered fast and efficient in detecting objects, whereas SSD Mobilenet_v1 might be less efficient but is faster to train. We then choose Faster R-CNN to change some parameterss for training.

The experiment begins with changing the parameters for Faster R-CNN Inception Resnet_v2 before training the labelled GSSS Dataset. Afterwards, we get the best models for testing. The chosen models have the models total loss minimize as much as possible. Subsequently, we evaluate on test images to get the accuracy of the model.

The experiment results show that the Animal De-

tection is able to recognize animals almost accurately from faraway or nearby with only minor mistakes. However, the case of the animal blending in with the background is still hard to detect an animal in it without separating the background first. There is also the problem with only a small part of the animal going with another one appearing on the camera. In some cases, the detection sees the animals as one individual. Nevertheless, this approach has the capability of being applied in a large outdoor environment, similar to the dataset environment. The application can also be improved furthermore for real-time purposes.

The existing methods, our main approaches, the experiment and results, and the impact are presented accordingly in Section 2, Section 3, Section 4 and Section 4.

2 Background and Related Work

There are two works relating to animal detection, one is segmentation and another one is general identification. Kumar et al. first extract the color texture moments. Then they apply the features into a Probabilistic Neural Network along with K-Nearest Neighbor. Yu et al. use Invariant Feature Transform (SIFT) with cell structured Local Binary Pattern in local feature extraction along with linear SVM to classify those features. and with segmentation, there are [1], [2] and [3]. Chen et al. [1] use Ensemble Video Object Cut to segment the object then use a novel DCNN based species recognition algorithm. Gomez et al. [3] seperate the classification into two parts. It is classifying species identification problem in two scenarios using very deep Convulational Networks such as VGGNet, AlexNet, Resnet and GoogLeNet then classify between the two specific species using deep CNN. The images chosen contain some of the animal's parts only to add high complexity.Giraldo et al. [2] use background subtraction and deep CNN to solve the problem of automatic genera recognition. This method includes feature-extraction of CNN, LASSO selection, Multi-Layer RPCA segmentation and SVM classification.

With segmentation, Chen et al. [1] use Ensemble Video Object Cut to segment the object. Then to recognize the species of the image sequence, they use a novel DCNN based species recognition algorithm. Gomez et al. [3] seperate the classification into two parts. It is classifying species identification problem in two scenarios. First, they classified the species in the Snapshot Serengeti dataset, using very deep Convulational Net such as VGGNet, AlexNet, Resnet and GoogLeNet. The second step is to classify between two specific species using deep CNN. The images chosen contain some of the animal's parts only to add high complexity. Giraldo et al. [2] use background subtraction and deep CNN to solve the problem of automatic genera recognition. This method includes feature-extraction of CNN, LASSO selection, Multi-Layer RPCA segmentation and SVM classification.

3 Proposed Method

3.1 Overview

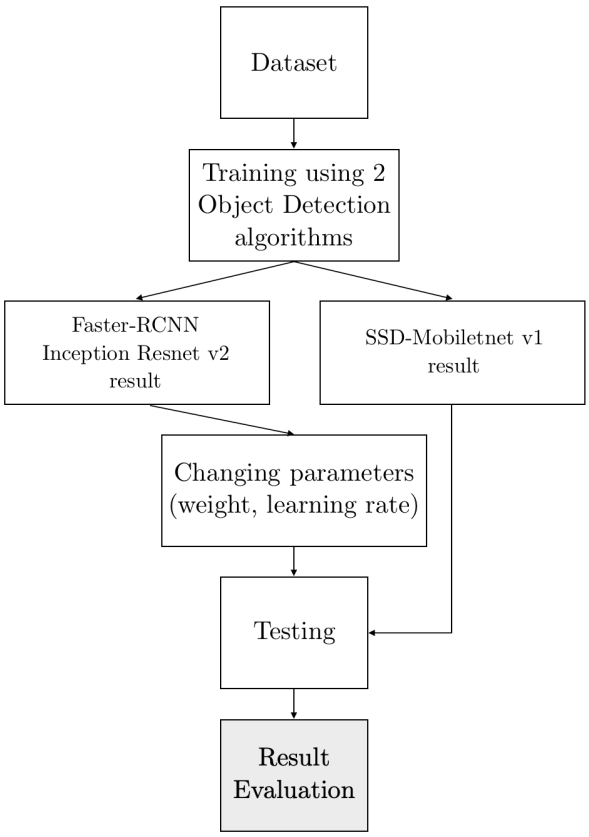


Figure 1. Overview of the proposed method

Our method in Figure 1 includes four main steps. Starting off, we use the already labelled GSSS Dataset with annotated bounding-box as our training and testing dataset. After preparing the dataset with the right format, we go on to train the models based on this dataset, using 2 different object detection algorithms, SSD and Faster R-CNN. For Faster R-CNN, we alter its classification weight. With the models trained in the previous step of both SSD and Faster R-CNN, we test out their detection ability.

Lastly, we evaluate the results based on the detection accuracy, training time and testing time.

3.2 Dataset and Preparation

3.2.1 Golden Standard Serengeti Snapshot Dataset

The GSSS Dataset [6] comes from a Zooniverse project. The Zooniverse project enables volunteers to assist professionals in their fields, such as Arts, History, Medicine, and in our case, Nature. For this specific dataset, Zooniverse offers a wide range of images taken from highly annotated cameras placed in an African savanna, Serengeti. The images on Zooniverse are then classified by volunteers from all around the world. These animals are indentified correctly using their characteristics, such as tails, stripes, e.t.c. Until now, there has been over 3.2 million image sequences of 46 mammals species and the dataset is still growing along with the help of volunteers.

3.2.2 Configuration

For training, we use Tensorflow API object detection. In this project, we use the CPU version of Tensorflow. We run our training on a 2.3GHz dual-core Intel Core i5, Turbo Boost up to 3.6GHz, with 64MB of eDRAM Mac.

3.2.3 Data preparation

Rather than using LabelImg to do the labeling ourselves, we decide to use an already labelled dataset of GSSS Dataset [6] with 2017 images and over 19.000 bounding box coordinates. There exists a tabular file along with the downloaded GSSS Dataset. This file contains the coordinates of the bounding box, image file name and its dimensions. We then transform the original tabular file format of both training and testing to TFrecord file format. This format is convenient since there are many functions in Tensorflow that will come in useful for TFRecord format. We also prepare a labelled map, which contains 46 animal names.

To do the training, we download a training pipeline configuration file of Faster-RCNN Inception Resnet_v2 and SSD Mobilenet_v1 pre-trained models on COCO Dataset. Training a model from the beginning can take longer than normal with our current hardware settings. By doing this, we save time in getting good resulting model for the GSSS Dataset. In the configuration file, we change the number of classes along with the number of training steps, and later on change classfication loss weights.

3.3 Training with default parameters

Without changing the default parameters of classification weight, we then train the models using the following command in terminal.

```
\\PATH: path to folder containing pipeline
config file and labelled map pbtxt file
python train.py --logtostderr
--train_dir={PATH}
--pipeline_config_path={PATH/pipeline.config}
```

3.3.1 SSD Mobilenet_v1

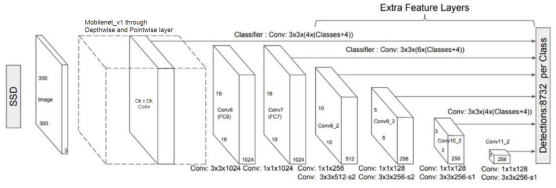
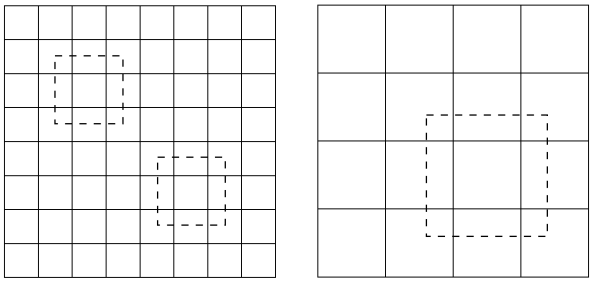


Figure 2. SSD Mobilenet_v1 architecture

First, we resize the image dimensions to 300x300. Mobilenet_v1 architecture will then create feature maps. The images afterwards will go through a depthwise convolution (DkxDk convolution). After that, a pointwise convolution (1x1 convolution) will be applied to change the dimension of the first feature maps.



8 x 8 feature map 4 x 4 feature map

Figure 3. Multi-scale layers

After obtaining the feature maps, 6 multi-scale convolution filters are applied for classification and bounding-box prediction simultaneously, including Conv6, Conv7, Conv8_2, Conv9_2, Conv10_2 and Conv11_2. These filters make the resolution of feature maps lower respectively in order to detect large-scale objects much easier.

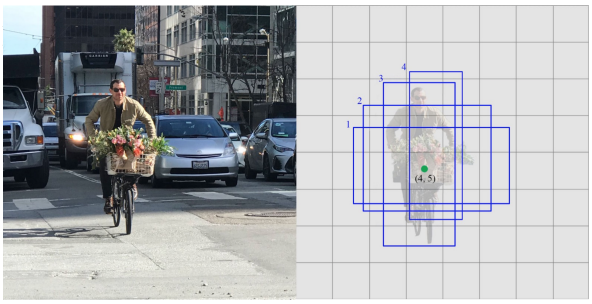


Figure 4. 4 predicted boxes in each cell

In each of the multi-scale layer, the algorithm finds the localization and classification probabilities of each cell in the feature map. For each cell, there are 5 bounding box predictions and each bounding box has different ratios between width and height. The 5 aspect ratios applied to the bounding box are 1.0, 2.0, 0.5, 3.0, 0.333. Each bounding box prediction consists of 50 output channels (46 probabilities for classification and 4 figures relating to position - x and y, width and height). The localization weight in the pre-trained model is 1.0 and classification weight is 1.0.

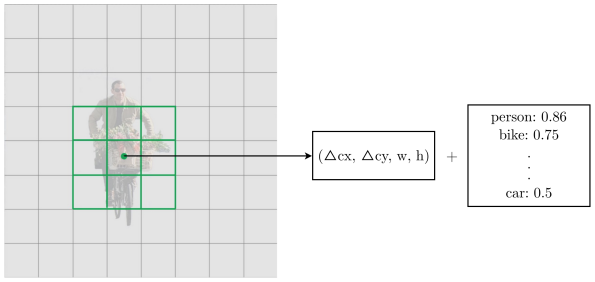


Figure 5. Components in each prediction

Based on 5 bounding box predictions, it can calculate the default box for each cell. Last is computing the localization loss between the predicted box and ground-truth box by the smooth L1 loss with IoU larger than 0.69999, the multi-classification loss by softmas loss, and the total loss based on the two previous computed loss as shown below respectively.

$$L_{loc}(x,l,g) = \sum_{i\in Pos}^{N} \sum_{m\in\{cx,cy,w,h\}} x_{ij}^{k} \text{smooth}_{L1}(l_i^m - \hat{g}_j^m) \quad (1)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^{w} = \log(\frac{g_j^w}{d_i^w}) \qquad \hat{g}_j^{h} = \log(\frac{g_j^h}{d_i^h})$$

$$L_{conf}(x,c) = -\sum_{i\in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i\in Neg} log(\hat{c}_i^0) \quad (2)$$

$$\text{where} \begin{cases} \hat{c}_i^p = \dfrac{\exp(c_i^p)}{\sum_p \exp(c_i^p)} \\ \boldsymbol{N} \text{ is the number of matched default boxes.} \end{cases}$$

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)) \quad (3)$$

3.3.2 Faster-RCNN Inception Resnet_v2

First, the images are resized to a fitted dimension - 600x1024. Secondly, using Inception_Resnet_v2 architecture, the feature maps are created. The images will then go through 9 layers, which includes Stem, 5xInception_Resnet_A, 10xInception_Resnet_B, Reduction_B, 5xInception_Resnet_C, Average Pooling, Dropout and Softmax.

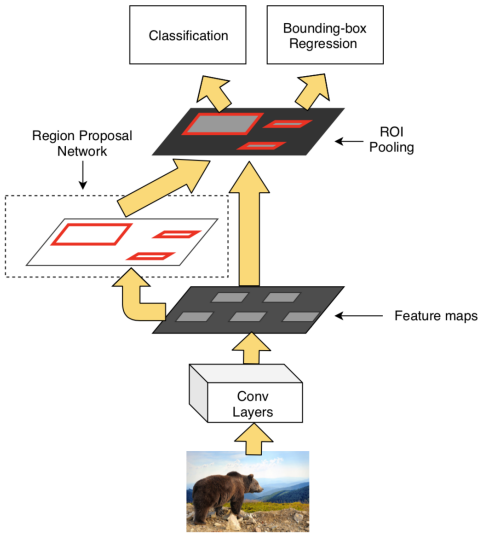


Figure 6. Main steps in Faster-RCNN algorithm

Thirdly, input feature maps will pass through 2 main networks: Region Proposal Network (RPN) for generating region proposals and Region of Interest (RoI) pooling network for creating fully connected layers used to detect animals.

In RPN, each cell in feature maps extracted in the previous step are slided through by 12 different sliding windows and producing 12 anchors. In the pre-trained model, there are 4 different scales for each

window (0.25, 0.5, 1.0, 2.0) and 3 aspect ratios (0.5, 1.0, 2.0). Every anchor has a localization probability P describing how much accuracy it overlaps with the ground-truth bounding box. In this model, we choose the Intersection over Union between predicted and ground truth box (IoU) to be 0.6999, which is the default IoU. If IoU of anchor is more than 0.6999, the localization will be 1, and 0 otherwise.

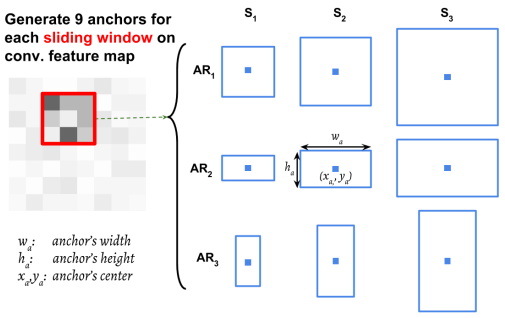


Figure 7. Anchor boxes in RPN

In each anchor having P equals 1, there are 2 more tasks: objectness classification and bounding box regression. The result of regressor indicates figures of the predicted box, which includes position x,y, width and height. The localization weight used for regression is 2.0. Result of classification decides whether there is any object contained in the predicted box (P=1 or P=0).

In the next step, RoI pooling network applies the feature maps in the second step and object proposals in the third step as input. Each object proposals takes a section of the input feature map corresponding to it, then resizes that section. To resize it into a new fixed-size feauter map, first is dividing the region proposals into fixed 17 sections. Next is finding the largest value in each section by applying max pooling.

In the last step, RoI pooling output is used for passing through fully connected layers for softmax classification and localization. The default classification and localization weights used in pre-trained model are 1.0 and 2.0 respectively.

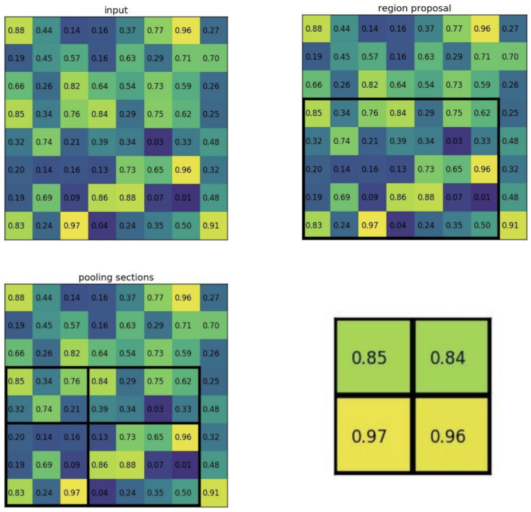


Figure 8. Four steps in ROI pooling

3.3.3 Conclusion

After training, the results show even though SSD Mobilenet_v1 trains at a faster rate, the loss of it is far higher at the beginning compared to that of Faster-RCNN Inception Resnet_v2, which is 181.5232 and 4.0359 respectively. Therefore, we decide to use Faster-RCNN as our main training algorithm.

3.4 Training with changed parameters

To see if there are any big changes, we change the classification weights in the configuration file two times and learning rate one time. The weight is changed from 1.0 to 1.3 and 0.8 for the first two training times. Then we proceed on to change learning rate from to the initial 0.003 to 0.004.

Afterwards, we use Tensorboard to inspect all the graphs. We then choose the models with the smallest total loss for each of the 3 training times to evaluate later on.

3.5 Testing

In this step, we test the results of the models we picked. The results and evaluation will be further discussed in Section 4. To evaluate the results, we run the following command in terminal.

```
python eval.py --logtostderr
--pipeline_config_path={PATH/pipeline.config}
--checkpoint_dir={PATH} --eval_dir={PATH}
```

3.6 Algorithm Comparison

SSD generally applies Single Shot Detector for predicting bounding-box while Faster-RCNN uses the Region Proposal Network for detecting boundaries followed by a classifier. Region-based detectors in Faster-RCNN are more accurate compared to Single Shot Detector. However, the running time for training and testing are slower. Moreover, in Region Proposal Network, sliding windows are created and run through feature maps to detect objects. For different object types, it uses different window shapes. The fatal drawback of the sliding windows is that we use it as the final boundary-box. For that, we need too many shapes to cover most objects.

SSD model uses a more effective solution for treating the window as an initial guess. It has a detector to predict the class and the boundary box from the current sliding window simultaneously. This concept is very similar to the anchors in Faster-RCNN. However, Single Shot detectors predict both the boundary box and the class at the same time. Despite lower resulting accuracy in detection, the running time for training and testing acquires higher speed. Therefore, Single Shot Detector often trades real-time processing speed. Because of using multi-scale layers, Single Shot Detectors tend to have issues in detecting objects that are too close or too small while large-size objects are detected better.

4 Experiments and Results

4.1 Data

In the experiment, we use 3 types of training data based on this order:

1. Data with 2 species: Wildebeest and Impala
2. Data with all 46 species
3. Data with 11 species

Overview of the data above can be found at Table IV, Table II, and Table III. The original observations of the full data with 46 animals is from Schneider et al. [7].

We will be associating the 3 datasets with the 2 methods throughout the whole process to see the conclusion better. We start training using the modified data with only 2 species in it. We decide to check the speed and the total loss of the two methods first to get used to the training process before going through the whole dataset. The training process only goes for about 300 steps since we want to focus more on the data chosen for evaluation.

Afterwards, we train on the original data, with 46 species in it. Due to the distribution of this dataset, the results are not quite satisfactory. This time, during our training, we test out some of the images. For both SSD and Faster R-CNN, the result only yields towards detecting Wildebeest and Zebra. These two species have its distribution at 40.0% and 18.9%, which cover more than half of the dataset. However, further on for about 100 steps, we are able to detect another species. And because of the highly distributed dataset, we omit some of the dectections out of the testing file to maintain the balance between the species. Besides that imbalance, there are also some that only have 1 to 11 detection in the training dataset.

Our chosen data after omitting out some of the few species contains only 11 animals as shown in Table III. This time, we train all of the 4 methods to 1000 steps.

4.2 Training with the first dataset

We decide to train using SSD Mobilenet_v1 first due to its real-time processing speed. While training, we also inspect the total loss graph in Tensorboard. The total loss starts at a really high rate, over 100 and gradually falls to about 7 and 8 at step 300. The training time, however, is quite fast. This is due to the fact that SSD only takes one single shot to detect numerous animals within the image. It passes an input image into a convolutional network only once, then computes a feature map and predicts the classes and the bounding boxes directly.

Afterwards, we start training on Faster R-CNN with Inception Resnet_v2 as the backbone. We decide to use this to change parameters rather than SSD-Mobilener_v1 because of its high accuracy. This process only takes 300 steps so the accuracy of the two algorithms may not be high. For this training, the total loss at the beginning is about 4, and is 0.3 at the 300th step. This time it takes a lot longer for the whole process to finish. Faster R-CNN has to run a region proposal network first and then use a second network to classify and output the boundary boxes. Therefore, it takes more time to finish but get higher accuracy in detecting animals.

4.3 Training with the second dataset

This time we start training on 4 different models: SSD, Faster R-CNN with default parameters, with weight equals to 1.3, and with weight equals to 0.8. While training on the original dataset, we also evaluate the results to see how it will detect the species

since this dataset is highly imbalance. Because there are a few species that have its distribution too high and some are too low, the trained models can only detect the two top species, Wildebeest and Zebra. After the 500th step, it can only detect another one or two species. So we decide to change the dataset we use and this will be our final dataset.

4.4 Training with the last dataset

Since this will be the last dataset, we also change the parameter for the learning rate of Faster R-CNN. All of the training of the 4 previous models retain the same, with the training time quite similar to each other. For the change with learning rate, because it is higher than the default one, it is harder to reach the minimal point in the loss function. The learning rate is too high causing it to jump through steps too far, not being able to make the total loss reach 0.

We export the best models of the 5 different methods for evaluation. The overview of the training process is shown in Table VI.

4.5 Testing

The evaluation for the chosen dataset on 5 different models includes 322 testing images. Tabel I shown below is the overall accuracy mAP@0.5IOU of the 5 models.

TABLE I
COMPARISON OF SSD AND DIFFERENT FASTER R-CNN MODELS BASED ON ACCURACY

Method	Classification Weight	Learning Rate	mAP
SSD*	-	-	14.8
Faster R-CNN†	-	-	27.9
Faster R-CNN†	0.8	-	21.1
Faster R-CNN†	1.3	-	22.5
Faster R-CNN†	-	0.04	15.9

*with backbone MobileNetv1

†with backbone Inception ResNetv2

We can clearly see from the mAP of each model that the highest mAP comes from the default Faster R-CNN Inception Resnet v2. While changing the classification weight decreases the mAP of the model. For learning rate, the overall accuracy is quite low. That is because we choose the learning rate to be a little bit higher than the default one. The learning rate should be fitted so that the loss function can reach its minimum.

If the rate is too high, it will jump through many steps at once. This will make it harder to reach the minimum since it keeps passing the minimum loss location. And it just keeps on making large steps which will make the mAP lower than normal.

If the rate is too low, it will jump step by step. It will gradually reach the minimum spot if keeps going. However, this is not efficient either since going step by step takes too much time.

A solution for this is to find the right value for learning rate for the model to have its loss coming towards 0. After finding the correct value, we also might want to decay the learning rate.

Table V shows the accuracy AP@0.5IOU of each 11 species. For some of the species, the accuracy is near 0. Table VI is the overview of training time and testing time. For the training, we decide to train on 2 different configurations, but still mainly on the original configurations stated in sub section 3.2.2

5 Conclusion

Our Animal Detection method is based on using the available methods - SSD Mobilenet v1 and Faster R-CNN Inception Resnet v2. We also utilize the available source code from Tensorflow library. However, our training shows unsatisfactory results. Using GSSS Dataset for training and testing, we achieve the accuracy at around 20%. The main reason for this is the imbalance in the dataset and the training steps being too low (1000 steps). In the future, the authors will create or re-organized the dataset to be more balance. This dataset will have the same amount of data for each species and we will also alter the images using rotation and cropping. At the same time, we will raise the training steps much higher to create a fitting model with a higher accuracy. And we will apply all of what we have done and fixed into a real-time mobile application.

References

[1] Guobin Chen, Tony X. Han, Zhihai He, Roland Kays, and Tavis Forrester. Deep convolutional neural network based species recognition for wild animal monitoring. *2014 IEEE International Conference on Image Processing, ICIP 2014*, pages 858–862, 01 2015.

[2] Jhony Giraldo Zuluaga, Augusto Salazar, Alexander Gomez Villa, and Anglica Diaz-Pulido. Automatic recognition of mammal genera on camera-trap images using multi-layer robust principal component analysis and mixture neural networks. 05 2017.

[3] Alexander Gomez Villa, German Diez, Augusto Salazar, and Anglica Diaz-Pulido. Animal identification in low quality camera-trap images using very deep convolutional neural networks and confidence thresholds. 10072:747–756, 12 2016.

[4] Y H Kumar, N Manohar, and H Chethan. Animal classification system: A block based approach. *Procedia Computer Science*, 45, 12 2015.

[5] Mohammad Sadegh Norouzzadeh, Anh Nguyen, Margaret Kosmala, Ali Swanson, Craig Packer, and Jeff Clune. Automatically identifying wild animals in camera trap images with deep learning. *CoRR*, abs/1703.05830, 2017.

[6] Stefan Schneider, Stefan Kremer, and Graham Taylor. Gold Standard Snapshot Serengeti Bounding Box Coordinates, 2018. https://doi.org/10.5683/sp/tpb5id.

[7] Stefan Schneider, Graham W. Taylor, and Stefan C. Kremer. Deep learning object detection methods for ecological camera trap data. *2018 15th Conference on Computer and Robot Vision (CRV)*, pages 321–328, 2018.

[8] Xiaoyuan Yu, Wang Jiangping, Roland Kays, Patrick Jansen, Tianjiang Wang, and Thomas Huang. Automated identification of animal species in camera trap images. *EURASIP Journal on Image and Video Processing*, 1, 09 2013.

TABLE II
GOLD STANDARD SERENGETI SNAPSHOT (GSSS) DATASET OVERVIEW
OF 46 SPECIES CREATED BY SCHNEIDER ET AL., 2018 [7]

Species	Total Quantity	Total Images	Image Class Distribution (%)
Wildebeest	11321	1610	40.0
Zebra	3677	767	18.9
Buffalo	987	227	6.00
Gazelle Thomson	938	198	4.88
Impala	541	149	3.67
Hartebeest	351	242	5.96
Guineafowl	195	54	1.33
Gazelle Grants	176	61	1.50
Warthog	162	105	2.59
Elephant	125	85	2.10
Giraffe	121	87	22.14
Other Bird	77	48	1.18
Human	67	59	1.45
Stork	63	12	0.296
Spotted Hyena	62	54	1.33
Eland	48	24	0.592
Reedbuck	44	29	0.715
Oxpecker	43	14	0.345
Baboon	35	22	0.542
Lion	34	17	0.419
Hippopotamus	32	28	0.690
Buff Crested Bustard	27	15	0.370
Topi	24	16	0.394
Cattle Egret	86	15	1.50
Mongoose	11	5	0.123
Porcupine	10	8	0.197
Kori Bustard	10	8	0.197
Cheetah	7	6	0.148
Dik-dik	7	7	0.173
Superb Starling	6	3	0.0739
Serval	6	6	0.148
Aardvark	4	4	0.0986
Secretary Bird	4	4	0.0986
Leopard	4	3	0.0739
Buckbuck	4	4	0.0986
Jackal	3	3	0.0739
Other Rodent	3	1	0.0246
Wattled Starling	3	1	0.0246
Aardwolf	2	2	0.0493
Ostrich	2	2	0.0493
Hare	1	1	0.0246
Grey Backed	1	1	0.0246
Fiscal	1	1	0.0246
Rhinoceros	1	1	0.0246
Vervet Monkey	1	1	0.0246
Waterbuck	1	1	0.0246
White-headed Buffalo Weaver	1	1	0.0246

TABLE III
GOLD STANDARD SERENGETI SNAPSHOT (GSSS) TRAINING DATASET WITH A REDUCED IN THE TOTAL QUANTITY OF 11 SPECIES

Species	Total Quantity	Total Images	Image Class Distribution (%)
Wildebeest	1500	253	16.28
Zebra	1000	273	17.57
Buffalo	834	227	14.61
Gazelle Thomson	821	182	11.71
Impala	500	139	8.94
Hartebeest	299	212	13.64
Guineafowl	100	24	1.54
Gazelle Grants	100	36	2.32
Warthog	100	71	4.57
Elephant	100	67	4.31
Giraffe	100	70	4.51

TABLE IV
GOLD STANDARD SERENGETI SNAPSHOT (GSSS) TRAINING DATASET WITH AN EVEN DISTRIBUTION BETWEEN TWO SPECIES

Species	Total Quantity	Total Images	Image Class Distribution (%)
Wildebeest	300	51	54.3
Impala	240	43	45.7

TABLE V
COMPARISON OF SSD AND DIFFERENT FASTER R-CNN MODELS BASED ON DETAILED ACCURACY OF 11 SPECIES (%)

Method	Buffalo	Elephant	Gazelle Grants	Gazelle Thomson	Giraffe	Guinea-fowl	Harte-beest	Impala	Warthog	Wilde-beest	Zebra
SSD*	7.63	8.19	0	24.2	39.6	0	8.68	1.78	26.0	5.30	2.67
Faster R-CNN†	23.8	16.6	0	25.2	18.1	0	32.6	16.0	0	17.6	39.2
Faster R-CNN†$^{\alpha}$	11.3	0	0	27.4	0	0	23.6	2.03	0	26.1	39.9
Faster R-CNN†$^{\beta}$	13.7	0	0	27.6	0	0	33.9	0	0	19/1	40.8
Faster R-CNN†$^{\gamma}$	9.12	0	0	13.1	0	0	16.5	0	0	16.7	23.1

*with backbone MobileNetv1
†with backbone Inception Resnetv2
$^{\alpha}$with classification weight = 0.8
$^{\beta}$with classification weight = 1.3
$^{\gamma}$with learning rate = 0.04

TABLE VI
COMPARISON OF SSD AND DIFFERENT FASTER R-CNN MODELS
BASED ON TRAINING AND TESTING PROCESS

Method	Step	Training Time(hour)	Total Loss	Testing Time(hour)	Testing Time(sec/img)
SSD*	1000	15.0	5.51	0.50	5.50
Faster R-CNN†	1000	12.6	0.81	3.00	33.5
Faster R-CNN†$^{\alpha}$	1000	12.4	0.64	2.33	26.1
Faster R-CNN†$^{\beta}$	1000	24.0	1.62	5.33	59.5
Faster R-CNN†$^{\gamma}$	300	6.50	1.63	5.50	61.5

*with backbone MobileNetv1
†with backbone Inception Resnetv2
$^{\alpha}$with classification weight = 0.8
$^{\beta}$with classification weight = 1.3
$^{\gamma}$with learning rate = 0.04