Towards an IoT-based Deep Learning Architecture for Camera Trap Image Classification

Imran A. Zualkernan
Comp. Science and Engineering
American University of Sharjah
Sharjah, UAE
izualkernan@aus.edu

Salam Dhou
Comp. Science and Engineering
American University of Sharjah
Sharjah, UAE
sdhou@aus.edu

Jacky Judas
Conservation Unit
Emirates Nature - WWF
Duai, UAE
jjudas@enwwf.ae

Ali Reza Sajun
Comp. Science and Engineering
American University of Sharjah
Sharjah, UAE
b00068908@aus.edu

Brylle Ryan Gomez
Comp. Science and Engineering
American University of Sharjah
Sharjah, UAE
b00067871@aus.edu

Lana Alhaj Hussain
Comp. Science and Engineering
American University of Sharjah
Sharjah, UAE
g00071496@aus.edu

Dara Sakhnini
Comp. Science and Engineering
American University of Sharjah
Sharjah, UAE
g00068368@aus.edu

2020 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT) | 978-1-7281-8420-3/20/$31.00 ©2020 IEEE | DOI: 10.1109/GCAIOT51063.2020.9345858

***Abstract*— Maintaining biodiversity is a key component of the United Nations (UN) "Life on Land" sustainability goal. Remote camera traps monitoring animals' movements support research in biodiversity. However, images from these camera traps are currently labeled manually resulting in high processing costs and long delays. This paper proposes an IoT-based system that leverages deep learning and edge computing to automatically label camera trap images and transmit this information to scientists in a timely manner. Inception-V3, MobileNet-V2, ResNet-18, and DenseNet-121 were trained on data consisting of 33,984 images taken during day and night with 6 animal classes. Inception-V3 yielded the highest macro average F1-score of 0.93 and an accuracy of 94%. An IoT-based system was developed that directly captures images from a commercial camera trap, does the inference on the edge using a Raspberry Pi (RPi), and sends the classification results back to a cloud database system. A mobile App is used to monitor the camera images classified on camera traps in real-time. The RPi could easily sustain a rate of processing 1 image every 2 seconds with an average latency of 1.8 second/image. After capture and pre-processing, each inference took an average of 0.2 Millisecond/image on a RPi Model 4B.**

Keywords—deep learning, transfer learning, convolutional neural networks, animal classification, camera trap, wildlife monitoring, edge computing, TensorFlow lite, raspberry pi, IoT

I. INTRODUCTION

According to a 2019 U.N. report authored by 450 experts, three quarters of the earth's land has been significantly altered by humans, and up to one million species are endangered and at the risk of going extinct [1]. Ecologists use wildlife population monitoring as an analytical tool to study and understand how to maintain biodiversity. Camera traps are specialized cameras used by ecologists to track the movement of animals by capturing images or videos along with their location and time. Camera traps are an efficient way to monitor animal populations and the data generated by these camera traps help ecologists study animals' population, behavior, and habitat [2]. Camera traps produce a large amount of image data that is currently processed manually in order to extract the required information. For example, images captured by a camera trap need to be labelled according to animals' species. Camera traps are also sensitive to events produced by environmental factors like the movement of vegetation due to wind, or sandstorms etc. Such 'ghost' images are plenty and require appropriate labeling and isolation from those images showing actual animals of interest. Camera traps are often deployed in remote locations, where they are left unattended for months, and hence accumulate a vast number of images to be processed.

In the current usage scenario, camera traps suffer from two key problems. First, it is time-consuming and expensive to manually label each image generated from a camera trap [3]. Secondly, it takes too long for the information in the image to reach the concerned authorities. For example, an image of an endangered species may not reach the ecologists for months because the camera trap is located in a remote location and the data is collected every six months.

The purpose of research reported here is to automatically classify images retrieved from camera traps and to send the species information instantly to the concerned authorities using deep learning and edge analytics. This research is aligned with UN sustainable goal 15, "Life on Land" that focuses on promoting and restoring sustainability on land. This approach will help accelerate research to address biodiversity loss because scientists will be able to acquire this data faster and cheaper.

The rest of the paper is organized as follows. Background information on various neural network architectures that have been used for species classification is presented next. This is followed by an overview of the IoT architecture used in the proposed system followed by a section on how the neural networks were trained. The paper ends with a discussion of deploying the neural networks on an edge device and the conclusion.

II. BACKGROUND

A. Neural Network Architectures for Species Classification

A variety of neural network architectures have been used for animal species classification. Use of each architecture in animal species classification is briefly described below.

Inception-V3 was used in many similar classification problems like flower [4] and fish species identification [5]. More specifically, Inception-V3 was used to classify animal images from the Caltech Camera Traps (CCT) dataset into 16 different categories with an accuracy of 79.17% [6]. Using Inception-V3, another study demonstrated the feasibility of using an IoT-based animal classification system consisting of camera traps and performing classifications on edge nodes as well as synthetic image generation for training the model [7]. Inception-V3 performs inverse graphic operations to extract features from images and thus it outperformed the capsule networks method with an accuracy of 95% as opposed to 79.6% by the capsule network [8]. Inception-V3 augmented with spatio-temporal information was found to improve CNN classification accuracy [9].

MobileNet architecture was also used by several species' classification studies. For example, MobileNet achieved 52.9% top-1 and 75.4% top-5 accuracy in classifying animal species [10]. Due to its lightweight nature, this architecture requires less computation power and memory storage and is thus popular among Internet of Things (IoT) applications. For example, MobileNet was used to classify images from the Climate-ecological Observatory for Arctic Tundra (COAT) dataset at 1.17 frames per second with 81.1% accuracy [11]. Another work used the same architecture to implement a wildlife monitoring system to prevent wild animals from crossing over forest-to-city borders [12].

ResNet-18, has also been used in animal species classification. For example, the ResNet-18 model was able to achieve an accuracy of 90.3%, classifying 48 classes from a subset of the Snapshot Serengeti dataset with 301,400 images [13]. This model was pretrained on the ImageNet dataset. Similarly, ResNet-18 achieved a mean per class accuracy of 72.7% in detecting wildlife at the Jasper Ridge Biological Preserve in Stanford, CA [14]. The model took the shortest amount of time on average to do an inference. Since ResNet-18 only has 18 convolutional layers, it is small and fast. The model used in [15] deployed ResNet-18 on the Snapshot Serengeti dataset and achieved an accuracy of 92.5% for species identification.

DenseNet has also been used for the species classification problem and has shown good performance. For example, DenseNet outperformed Inception-V3 and ResNet-50 architectures, and scored a top1-accuracy of 71.8% [16] on a plant classification problem. Similarly, DenseNet was used for dog breeds identification where DenseNet-169 performed the best with a test accuracy of 85.37% and DenseNet-121 with a test accuracy of 84.01% [17]. Similarly, DenseNet achieved a training accuracy of 85.14% for identifying 120 breeds of dogs [18]. According to [18], DenseNet has several advantages, which include reducing the influence on vanishing gradient problem, strengthening feature propagation, encouraging feature reuse, and reducing the number of parameters. These advantages were primarily due to the design of the dense block. However, as reported by [16], DenseNet201 used more GPU memory than Inception-V3.

Finally, a recent architecture called EfficientNet [19] achieved 84.4% top-1 and 97.5% top-5 accuracy on ImageNet while being 6.1x faster on inference than the best existing ConvNet and 8.4x smaller. The small size of the model and high accuracy achieved make this architecture a good candidate for running on edge devices.

B. Embedded Devices for the Edge Inference

Many neural networks have been deployed on edge devices for the purposes of animal classification. The most common edge device used was a Raspberry Pi (RPi) [11], [14], [20], [21],[22].

Mathur & Khatta [14] deployed various image classification models that could be connected to camera traps to do real-time detection of wildlife from video feeds. The model deployed on the edge device achieved a mean-per-class accuracy of 87.6%. The execution time and maximum power drawn were compared by varying whether the model used the original or the quantized version, and whether TensorFlow or TensorFlow Lite was being used. They noted that the power utilized showed a marginal increase over a baseline of about 3 Watts consumed by RPi 3 in the idle state. Thomassen [11] also deployed 4 different neural network models based on MobileNet on a Raspberry Pi and showed that the Raspberry Pi could achieve a classification speed of 1.17 FPS, an average precision of 61.0% and an average recall of 49.7% using a model size of 17MB whereas a GPU-based computer could classify at the rate of 12.5 FPS for the same model. The Raspberry Pi utilized 0.122Wh of energy for 5 minutes of work. Curtin & Matthews [20] used a Raspberry Pi based camera system that used deep learning to detect snow leopards with at least 74 percent accuracy at an acceptable speed for real-time processing. Each inference took 0.03 seconds/image disregarding the time taken to start up the deep learning model and loading it. They also noted that 0.10 additional Ampere of current was consumed when the Raspberry Pi was taking pictures and running the image recognition model. Monburinon *et al.* [21] proposed a hierarchical edge computing image recognition system in which the major processing was carried out on low-cost gateway devices like the Raspberry Pi. Their prototype lasted an average of 7 days before the battery ran out of power. The recognition engine was configured to do a recognition once every 5 minutes. Finally, Popat *et al.* [22] proposed an architecture for animal classification that had a final test accuracy of 87.9%. They suggested that the classification model could be chosen from many of the available options like various versions of Inception model or MobileNets where the latter are comparatively faster than the Inception-V3 model but at the cost of accuracy.

In summary, previous work strongly suggests that Raspberry Pi was a reasonable platform for deploying deep learning animal classification models.

III. System Architecture of the Proposed System

Fig. 1 shows the overall IoT architecture of the proposed system. As an animal passes by the camera trap, the onboard motion sensors activate the camera which captures an image of the animal and stores it on a Wi-Fi enabled SD Card as shown in Fig. 2. This then allows the RPi edge device to retrieve the image from the card using Wi-Fi and run the neural network model to classify the image. The image is then labelled and

stored locally while the label, timestamp and device ID are sent and stored in the Google's Cloud-hosted Firebase database (https://firebase.google.com/).

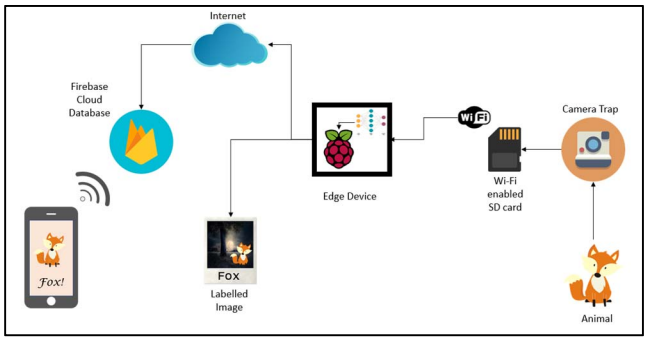


Fig. 1. System architecture of the proposed IoT system

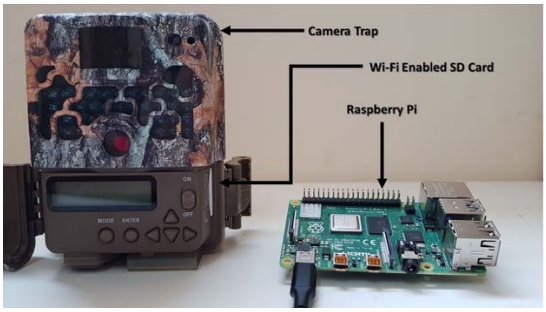
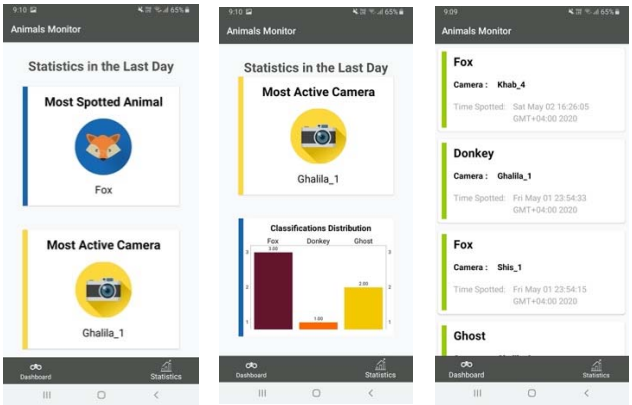


Fig. 2. Edge-node hardware using a commercial camera and RPi

Fig. 3 shows a mobile application that interacts with the cloud database and alerts an ecologist instantly about the status and history of observed animals at various camera traps. As Fig 3 shows, statistics about the most spotted animal, most active camera and number of total classifications can be easily retrieved from database. For example, in Fig 3(c), the App shows a list of all animal classifications showing the class, time and location of each inference.



(a) (b) (c)

Fig. 3. A Mobile Application for Camera Traps

The next section describes the methodology used to train the neural networks for classifying the animal images.

IV. Training the Neural Network

A. Data

The data used in this paper consisted of camera trap images taken within the Hajar Mountains of the United Arab Emirates (U.A.E.). Emirates Nature WWF conducts such wildlife surveys to identify biodiversity hotspots and to assist in the creation of a network of protected areas. The data had 33,984 camera trap images of various species. The largest number of images were of "ghosts" (non-animals and various anomalies) followed by goats, foxes (*Vulpes cana, Vulpes vulpes arabica*), donkeys and then sheep. Highly under-represented images were grouped into one class named "other" and included animals like cats, dogs, camels, birds and rats.

The data includes both night and day images. For example, Fig. 4(a) and 4(b) show night images of a fox and a donkey respectively. Fig. 4(c) shows a "ghost" image typically triggered by wind, grass, moths, or similar phenomenon. Fig. 4 (d) and 4 (e) show two different images of a goat.

Fig. 5 (a)-(d) show sample images of a camel, lizard, cat and a dog respectively. Due to class imbalance where not many instances of each such species were available, these under-represented images were grouped into the class category called "others."

B. The Training Regime

A 60-20-20 train/validate/test regime was used to randomly split the data into training, validation and testing sets. Table I shows the class-wise breakdown. As the Table shows, the data was unbalanced and the ghost images accounted for 40.15% percentage of the images. Similarly, the "other" class accounted for only 5.2% of the total number of images. In this first stage, SMOTE [23], oversampling or class weights were not used. These techniques will be pursued in a second phase of this research.

TABLE I. The Training Breakdown of Data

Class Name	Number of Images				
	Training	Validation	Testing	Total	%
Ghost	8,188	2,705	2,754	13,647	40.15
Goat	5,767	1,923	1,923	9,613	28.28
Fox	3,482	1,161	1,161	5,804	17.07
Donkey	1,178	393	393	1,964	5.77
Other	1,062	354	354	1,770	5.20
Sheep	723	232	231	1,186	3.48
Total	**20,400**	**6,768**	**6,816**	**33,984**	**100**

Inception-V3, MobileNet-V2, ResNet-18, and DenseNet-121, were all trained on this data. All architectures were pre-initialized to ImageNet weights. Initially, the models were trained by freezing the feature extraction layers and training the classification layers only. However, better results where the F1-score improved by about 20-30%, were obtained when both feature extraction and classifier layers of each architecture were configured to be trainable. Stochastic Gradient Descent (SGD) was used as the optimizer for all four models. Other hyper-parameters like batch size, learning rate, and number of epochs were obtained using the grid search. The models were implemented using Keras (https://keras.io/). The resulting hyperparameters for each model are shown in Table II.

Fig. 4. Samples of day and night images of the labeled classes

Fig. 5. Samples of images grouped into the "other" class due to class imbalance

TABLE II. HYPERPARAMETERS DETERMINED USING GRID SEARCH

Model Name	Parameters			
	Batch Size	Learning Rate	# of Epochs	Input Image Size
Inception-V3	48	0.05	75	299 × 299 × 3
MobileNet-V2	32	0.01	100	224 × 224 × 3
ResNet-18	32	0.005	150	224 × 224 × 3
DenseNet-121	16	0.001	100	224 × 224 × 3

C. Results

As Table III show all models performed reasonably well with macro-F1 averages above 91%. Because the models are to be deployed on IoT edge devices, the size of each model is an important consideration. As Table III shows the best performing model was Inception-V3 with a macro Average F1 score of 0.93, and the smallest size of 175 MB.

TABLE III. BEST RESULTS FOR EACH NN ARCHITECTURE

Model	Model Size (MB)	Accuracy	Macro Average F1
InceptionV3	175	94%	0.93
DenseNet-121	446	93%	0.93
ResNet-18	480	92%	0.91
MobileNetV2	507	93%	0.92

As Fig 6 shows, Inception-V3 converged only in 60 epochs and the trend of the validation loss strongly suggests that there was little overfitting.

As the ROC's in Fig. 7 shows, Inception-V3 performed well for all classes. The lowest Area Under the ROC Curve (AUC) was for the "other" class (AUC=0.98) which is to be expected given the high variability of images in this class.

Table IV shows the confusion matrix for the Inception-V3 model. As the Table shows, most large errors (shown in red) occurred due to the model not being able to classify the ghost images accurately. This is expected because the ghost images represented the largest number of data points in the training data, had the highest variability, and no data balancing techniques were used. The classes with labels (e.g., Fox, Donkey, Goat, and Sheep) were classified quite accurately.

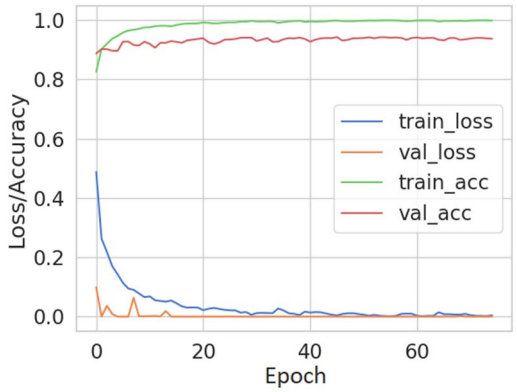


Fig. 6. Training loss and accuracy for Inception-V3 model

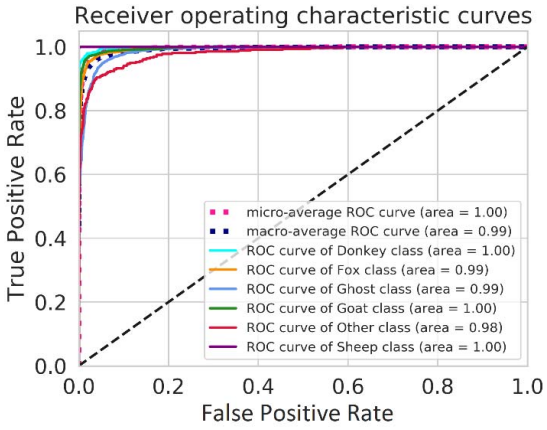


Fig. 7. ROC curves for the Inception-V3 model

It is interesting to note that even though sheep accounted for only 3.48% of the data, the neural network was able to classify this class almost perfectly (F1-score = 0.99). One hypothesis is that as Fig. 8 shows, the sheep often appeared in a herd, and therefore, this may have been one contributing factor because

other animals like foxes appeared individually. Out of the 231 sheep images used for Testing (see Table I), 134 (58%) images depicted a herd rather than an individual sheep.

TABLE IV. CONFUSION MATRIX FOR INCEPTION-V3 MODEL

		Predicted					
		Donkey	Fox	Goat	Sheep	Ghost	Other
Actual	Donkey	369	4	2	1	14	3
	Fox	3	1050	10	0	90	8
	Goat	0	8	1843	1	70	1
	Sheep	0	0	0	230	1	0
	Ghost	7	28	49	0	2643	27
	Other	2	10	10	0	81	251

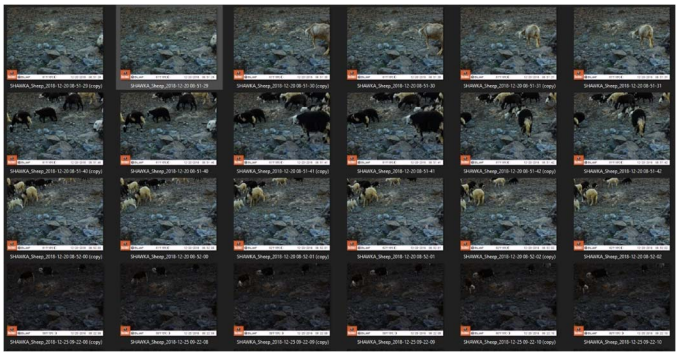


Fig. 8. Sample images for sheep showing they frequently appeared as a herd

V. DEPLOYMENT ON THE IOT EDGE NODE

A. The TFLite Model

The best trained Inception-V3 model was first converted to TensorFlow Lite (https://www.tensorflow.org/lite). The resulting model had a reduced size of 87 MB (from175MB), but the accuracy was reduced to 92% (from 93%), and macro average F1-score was reduced to 0.90 (from 0.93). F1-score for all classes except 'other' remained above 0.92. The F1-score of 'other' was reduced to 0.71 (from 0.79). Table V and Fig. 9 show the confusion matrix and ROC curves for the TFLite model respectively showing the model was still very capable.

TABLE V. CONFUSION MATRIX FOR TFLITE INCEPTION-V3 MODEL

		Predicted					
		Donkey	Fox	Goat	Sheep	Ghost	Other
Actual	Donkey	359	2	16	3	11	2
	Fox	2	1051	8	0	90	10
	Goat	2	8	1723	1	185	4
	Sheep	0	0	0	230	1	0
	Ghost	5	10	28	2	2685	24
	Other	3	5	7	5	116	218

B. Evaluation

In order to assess the response time and energy utilization of the of the proposed IoT edge device, 1000 images were consecutively shown and classified by a RPi Model 4B running the TensorFlow Lite model using Python. Power consumption of the RPi and latency were measured. The RPi took 30.5 minutes to process the 1000 images, or at the rate of about 1 image every two seconds. It should be noted that in real life

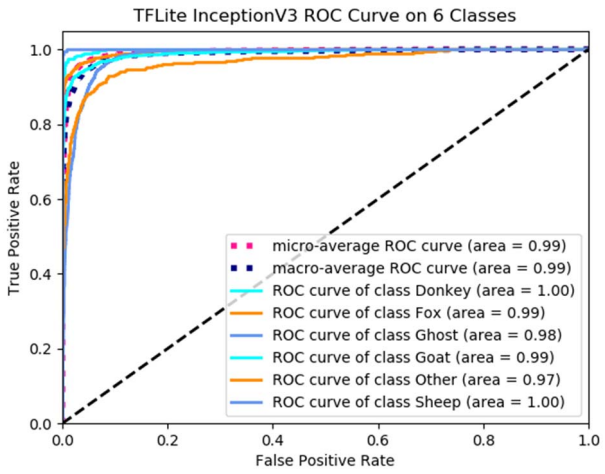


Fig. 9. ROC curves for the TFLite Inception-V3 model

the camera trap is only triggered when a movement is detected, and therefore, the average time between images will be significantly longer than once every two seconds

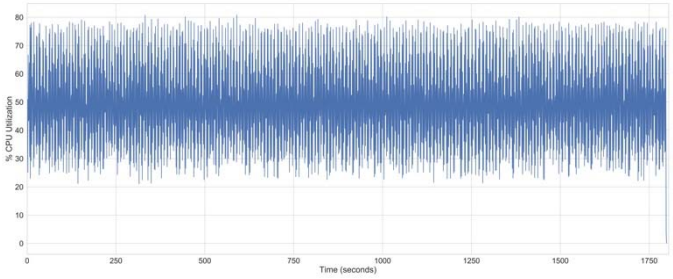


Fig. 10. % CPU utilization RPi 4B for 1000 inferences

Under these stress conditions, the mean current drawn by the RPi was 884.438 mA and the maximum current drawn was 1240 mA. In idle mode, a RPi 4B typically consumes around 540 mA. Based on this experiment one can conclude that under stress testing where an image is captured once every two seconds, the RPi will consume a total of 106.38 Wh of energy per 24 hours. Since a single 100-Watt PVC solar panel generates about 400 Wh/24-hours, a 20/25-Watt solar panel would be sufficient to power this system. Such a solar panel costs less than $10-15 in today's prices.

The mean %CPU utilization on RPi was 49.91% and the maximum %CPU utilization was 80.8%. As Fig. 10 shows, this means that the RPi CPU were never starved out under stress conditions, and it can be safely concluded that the CPU should be able to handle a typical load for a camera trap quite easily.

As Table VI shows, the mean latency between capturing the image and doing the classification was 1.8 second/image with a standard deviation of 0.03 second/image. On average it took 0.65 second/image to capture the image to the RPi buffer. Most of the latency (1.14 second/image) came from pre-processing the image once it was acquired. Pre-processing involved reshaping the image from 1920x1440 pixels to 299x299 pixels used as input to the Inception V3 model. As Table V shows, the actual inference only took an average of 0.0002 second/image.

TABLE VI. AVERAGE LATENCY IN SECONDS PER IMAGE (STANDARD DEVIATION)

Capture time (s)	Pre-processing time(s)	Inference time (s)	Total time (s)
0.65 (0.04)	1.14 (0.01)	0.0002 (< 0.0000)	1.80 (0.03)

VI. CONCLUSIONS AND LIMITATIONS

This initial investigation showed promising results. However, a number of issues need to be addressed further. First the data is highly unbalanced, and therefore, using SMOTE [23] or similar data balancing techniques may help improve the results further. Secondly, recent architectures like EfficientNet [19] that performed well in creating smaller networks can be explored further. In fact, our initial investigations show that EfficientNetB1 produced performance similar to Inception-V3 (F1-score macro average = 0.94) on this data set as well but with a much smaller model size (~ 54 MB).

This paper showed that potentially significant gains can be achieved in reducing both the time to reporting, and the effort required to label data from wildlife camera traps. Consequently, using deep learning technologies on the edge with an IoT infrastructure can potentially revolutionize the camera trap value-chain. Finally, since the inference is being done on the edge and does not require sending large amounts of data back to a server, low-power wide area networks [24], and specifically LoRAWAN [25] or NB-IoT [26] can be utilized for sending the classification results back to a central server.

ACKNOWLEDGMENTS

The authors would like to thank Emirates Nature-WWF, UAE for their technical help and for providing the data of camera trap images.

REFERENCES

[1] G. Zimmerman, "Nature Conservation Growning by Popular Demand." Wyss Campaign for Nature, Sep. 2019, [Online]. Available: https://static1.squarespace.com/static/5bbe12b6e5f7d12a614c4a2d/t/5d9b727cc78b862a70e0f91e/1570468477215/WyssCampaignforNature_PollingReport.pdf.

[2] *Living planet report*. Gland, Switzerland: WWF, 2018.

[3] B. G. Weinstein, "A computer vision for animal ecology," *J. Anim. Ecol.*, vol. 87, no. 3, pp. 533–545, 2018, doi: 10.1111/1365-2656.12780.

[4] I. Gogul and V. S. Kumar, "Flower species recognition system using convolution neural networks and transfer learning," in *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*, Mar. 2017, pp. 1–6, doi: 10.1109/ICSCN.2017.8085675.

[5] V. Allken, N. O. Handegard, S. Rosen, T. Schreyeck, T. Mahiout, and K. Malde, "Fish species identification using a convolutional neural network trained on synthetic data," *ICES J. Mar. Sci.*, vol. 76, no. 1, pp. 342–349, Jan. 2019, doi: 10.1093/icesjms/fsy147.

[6] S. Beery, G. Van Horn, and P. Perona, "Recognition in Terra Incognita," in *Computer Vision – ECCV 2018*, vol. 11220, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 472–489.

[7] A. R. Elias, N. Golubovic, C. Krintz, and R. Wolski, "Where's the Bear? - Automating Wildlife Image Processing Using IoT and Edge Cloud Systems," in *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, Apr. 2017, pp. 247–258.

[8] J. Kamdem Teto and Y. Xie, "Automatic Identification of Animals in the Wild: A Comparative Study Between C-Capule Networks and Deep Convolutional Neural Networks.," *Master Sci. Comput. Sci. Theses*, Nov. 2018, [Online]. Available: https://digitalcommons.kennesaw.edu/cs_etd/20.

[9] O. M. Aodha, E. Cole, and P. Perona, "Presence-Only Geographical Priors for Fine-Grained Image Classification," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South), Oct. 2019, pp. 9595–9605, doi: 10.1109/ICCV.2019.00969.

[10] G. V. Horn *et al.*, "The iNaturalist Species Classification and Detection Dataset," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 8769–8778, doi: 10.1109/CVPR.2018.00914.

[11] S. Thomassen, "Embedded analytics of animal images," Dec. 2017, Accessed: Aug. 23, 2020. [Online]. Available: https://munin.uit.no/handle/10037/12000.

[12] C. K. Sreedevi and E. Saritha, "Automated Wildlife Monitoring Using Deep Learning," Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 3447740, Sep. 2019. Accessed: Nov. 25, 2019. [Online]. Available: https://papers.ssrn.com/abstract=3447740.

[13] M. S. Norouzzadeh *et al.*, "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning," *Proc. Natl. Acad. Sci.*, vol. 115, no. 25, pp. E5716–E5725, Jun. 2018, doi: 10.1073/pnas.1719367115.

[14] A. Mathur and S. Khattar, "Real-time Wildlife Detection on Embedded Systems," Jun. 10, 2019. http://ilpubs.stanford.edu:8090/1165/ (accessed Nov. 23, 2019).

[15] M. Willi *et al.*, "Identifying animal species in camera trap images using deep learning and citizen science," *Methods Ecol. Evol.*, vol. 10, no. 1, pp. 80–91, 2019, doi: 10.1111/2041-210X.13099.

[16] J. Haupt, S. Kahl, D. Kowerko, and M. Eibl, "Large-Scale Plant Classification using Deep Convolutional Neural Networks," 2018.

[17] A. Ayanzadeh and S. Vahidnia, "Modified Deep Neural Networks for Dog Breeds Identification," ENGINEERING, preprint, Dec. 2018. doi: 10.20944/preprints201812.0232.v1.

[18] W. Shi, J. Chen, M. Liu, and F. Liu, "Dog Breed Identification." 2018, [Online]. Available: http://noiselab.ucsd.edu/ECE228_2018/Reports/Report18.pdf.

[19] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *ArXiv190511946 Cs Stat*, Nov. 2019, Accessed: Jul. 13, 2020. [Online]. Available: http://arxiv.org/abs/1905.11946.

[20] B. H. Curtin and S. J. Matthews, "Deep Learning for Inexpensive Image Classification of Wildlife on the Raspberry Pi," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, Oct. 2019, pp. 0082–0087, doi: 10.1109/UEMCON47517.2019.8993061.

[21] N. Monburinon, S. M. S. Zabir, N. Vechprasit, S. Utsumi, and N. Shiratori, "A Novel Hierarchical Edge Computing Solution Based on Deep Learning for Distributed Image Recognition in IoT Systems," in *2019 4th International Conference on Information Technology (InCIT)*, Oct. 2019, pp. 294–299, doi: 10.1109/INCIT.2019.8912138.

[22] P. Popat, P. Sheth, and S. Jain, "Animal/Object Identification Using Deep Learning on Raspberry Pi," in *Information and Communication Technology for Intelligent Systems*, Singapore, 2019, pp. 319–327, doi: 10.1007/978-981-13-1742-2_31.

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.

[24] W. Ayoub, A. E. Samhat, F. Nouvel, M. Mroue, and J.-C. Prévotet, "Internet of Mobile Things: Overview of LoRaWAN, DASH7, and NB-IoT in LPWANs Standards and Supported Mobility," *IEEE Commun. Surv. Tutor.*, vol. 21, no. 2, pp. 1561–1581, Secondquarter 2019, doi: 10.1109/COMST.2018.2877382.

[25] T. A. Balushi, A. Al Hosni, H. A. Theeb Ba Omar, and D. Al Abri, "A LoRaWAN-based Camel Crossing Alert and Tracking System," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, Jul. 2019, vol. 1, pp. 1035–1040, doi: 10.1109/INDIN41052.2019.8972063.

[26] H. Wang, Y. Wei, H. Zhu, Y. Liu, C. K. Wu, and K. Fung Tsang, "NB-IoT based Tree Health Monitoring System," in *2019 IEEE International Conference on Industrial Technology (ICIT)*, Feb. 2019, pp. 1796–1799, doi: 10.1109/ICIT.2019.8755153.