AICaTS - An Intelligent Camera Trapping System

Jinchen Ge
University of Cambridge
jg983@cam.ac.uk

Yucheng Ji
University of Cambridge
yj323@cam.ac.uk

Abstract

Camera trapping, in which cameras with motion sensors are set up to automatically capture images or videos of wild animals, has become one of the most important and efficient ways for scientists to understand the distribution or behaviour of wildlife. However, the constraints in power and storage, as well as the requirement of manual data processing still significantly limit the flexibility and applicability of camera trapping. To cope with these issues, we propose an intelligent camera trapping system we call AICaTS. Our system employs model compression techniques to integrate Convolutional Neural Networks (CNNs) with low power camera trap devices. Thereafter, image classification is performed on the edge to evaluate the contents of the captured images and then make valuable decisions like whether to start video recording, what kind of recording quality should be used, etc. Even with the strict resource restrictions, our compressed models still achieve competitive accuracies, which proves the feasibility of machine learning on common camera-trapping hardware. Finally, we also built a simple camera trap to demonstrate our ideas.

1 Introduction

Camera traps, also known as trail cameras, are widely used by researchers to observe and record the most natural states of the ecosystems. These cameras are designed to be automatically triggered by animals, so minimum maintenance is required and thus minimum interference to the wildlife. Apart from the camera sensor, a typical camera trap usually has one or more motion sensors, a flashlight (which is usually infrared) and an SD card [1]. The motion sensor used is often a passive infrared (PIR) sensor which is triggered by rapid change in heat.

Ideally, camera traps should only be triggered by animals, but false triggers are common. For instance, in the largest camera trap dataset Snapshot Serengeti [2] which contains 7.1 million images, 76% of images are labeled as empty. The major reasons for false triggers are the movement of the camera trap itself, the movement of the surroundings or the animals are moving too fast to be captured [1]. The false trigger could fill up the SD card and drain the battery but capturing only meaningless data. The extra data could also mean extra processing when the data is retrieved. Setting cameras up carefully and reducing the trigger sensitivity could alleviate the problem [1], but they are not perfect solutions. Our work - which we call AICaTS - introduces one or more compressed CNN models into the camera trapping systems to address this issue. These models could firstly confirm whether an animal is in the scene from capture images. If not, then no power-consuming or storage-consuming operation would be started. If an animal is present, the system could then identify its species. The species information could be useful because it enables us to treat different animals differently. For example, for the species that are not of the users' interest, the system could ignore them. For the most interesting species, the system could increase photo quality and frequency, or even turn on high-quality video recording which could capture more information, but consumes significantly more storage and power [3], especially in low-light conditions when the flashlight is turned on. Meanwhile, although users should not count on the classification accuracy on the edge, the processing on our system would still be able to save a significant amount of human labour, which could be very expensive, during manual

data processing. Moreover, when combined with extra modules such as a wireless communication module, the system may be able to enable instant notification to the user, which might be useful for tracking specific species or even anti-poaching.

2 Related work

2.1 Machine learning in camera trapping

There have been a lot of attempts to apply machine learning to animal images. To our best knowledge, the first attempt specifically on camera trap images is from Yu et al. [4]. They applied a multi-class linear Super Vector Machines (SVM) on hand-crafted Scale-Invariant Feature Transform (SIFT) [5] features combined with cell structured Local Binary Patterns (cLBP) [6]. An accuracy 82% was achieved on a relatively small dataset.

Chen et al. [7] were presumably the first to use a CNN for camera trap image classification. Instead of dealing with the whole image, they used the Ensemble Video Object Cut (EVOC) algorithm [8] to segment out animals and then apply a neural network with 6 layers on the crops. An overall accuracy of 38.315% was achieved.

Thereafter, Gomez et al. [9] were among the first to perform classification on the huge-scale Snapshot Serengeti dataset. They systematically tested various deep CNN architectures and confirmed the significance of a balanced dataset and the benefits of using segmented animal images. Later, Gomez et al. [10] also applied a similar method to a camera trap dataset from South America and discovered the accuracy improvement of classifying grouped animal sets instead of specific species.

Norouzzadeh et al. [11] also worked with Snapshot Serengeti dataset but their work was on a much larger scale than Gomez et al.'s [9] work. Their experiments were almost based on the full dataset which contains more than a million images, and their best species identification result (92%) was significantly better than the previous works. Moreover, they also performed other experiments including empty-or-animal classification, animal counting and animal status classification.

Willi et al. [12] took a more realistic approach by focusing on datasets with different properties and some with more practical sizes. They also attempted to use transfer learning to transfer information from CNNs trained on large datasets and observed a significant increase in accuracy. They were also the first to study how human effort could be reduced in a real project with the help of machine learning.

Classification-based techniques usually only output one prediction per image, so Schneider et al. [13] instead took a detection perspective. They labelled bounding box for a small portion of the Snapshot Serengeti dataset and then applied both the Faster R-CNN [14] and the YOLO [15] detection algorithm. Finally, they achieved an accuracy and IOU as high as 76.7% and 0.72 on the Snapshot Serengeti dataset.

All previous researches assume the machine learning models run on relatively capable workstations or even on the cloud, instead of on camera trap hardware which has extreme low computational power and memory. To our best knowledge, we are the first to systematically study techniques to deploy machine learning models on camera trap devices. Outside the field of research, there are some hardware or commercial projects that attempt to combine machine learning with edge camera trap devices [16][17][18], but they tend to focus on remote monitoring and for the projects that have technical details available, we find that they require extra machine learning hardware which could have significantly more resources than what a typical camera trap should have. In comparison, though we are not working with a real camera trap, we take into consideration the hardware specifications of a common camera trap and aim to cost minimum extra resource. Ideally, our pipeline might be deployed to available camera traps without significant hardware changes. On the other hand, our work could be used for remote monitoring, but unlike previous works, the major focus of this project is to optimize the workflow logic of camera traps with machine learning on the edge.

2.2 Model compression

Although popular convolutional neural network generates good predicting accuracy for image classification on the camera trap dataset [11], the expensive and intensive memory and computation requirements make the models impractical to deploy on resource-constrained devices. For instance,

a vgg16 model [19] pre-trained on ImageNet has 138 million parameters with the model size 500 MB. According to [20], about 90% of parameters and MAC operations are from convolutional layers. Therefore, decreasing the number of parameters of the dense layer operations in the convolutional layer can significantly reduce the storage and overheads for computation, and further fit the model onto the edge devices usually containing limited memory, storage and computing resources such as Camera Trap or other edge devices.

Among the standard model compression techniques, Pruning, Quantization and Knowledge distillation are the most powerful ones mainly applied to the convolutional neural network. Pruning [20] [21] targets removing redundant parameters that hardly contribute to decreasing the error and increasing the generality of the model. After that, the model size and the required computations are squeezed. When fitting on-board, a pruned model would also have less memory access from DRAM and power consumption during inference [21].

Another essential aspect in model compression is quantization, first introduced by Fiesler et al. [22]. Weights are typically stored as 32-bit floating-point numbers in the deep learning modules. Quantize such full-precision weights representations to lower the number of bits (e.g. 8-bit, 4-bit, or even with 1-bit) would contribute to a considerable reduction in model size and the total number of operations at the cost of acceptable performance lost. Moreover, quantization could further apply the idea to gradient and activation for a quantized form. Quantization can be used along with the training for an efficient process and less model size or post-training. Zhou et al.[23] train the floating-point CNN model and quantized it for efficient inference float Quantization. Similarly, our CNN model for animal classification can be trained with a full precision model offboard and focus more on the post-training quantization to save inference time and energy, which are critical criteria for a camera trapping system.

Besides, cumbersome networks still suffer from the trade-off between compression and performance loss though they could extract features effectively. We aim to deploy the model onto a microcontroller with a camera that might have a strict constraint on its peak memory size. Hinton et al. showed in the systematic study [24] that the knowledge acquired by the large model (as a teacher) could be transferred to a smaller model (as a student) via training the student model to match the soft targets generated by the teacher model. Both the student training and the soft target generation are performed under the same temperature to produce softer probability distribution over classes. Based on suggestions from Hinton et al. [24], the model could be generalized better if combine training on hard and soft targets at the same time. On the ground of the theory, our work could be seen as a particular case to concentrate on obtaining high-accuracy students from the cumbersome CNN model that cannot fit to the camera. Meanwhile, the smaller model and lower memory usage could improve the response time when implementing the system in reality.

3 Methods and implementation

3.1 Dataset

To train and evaluate our animal classification model, we use a subset from the largest benchmark dataset, Snapshot Serengeti [2], which contains approximately 2.65 million sequences and total 7.1 million images for 60 species of wild animals in Serengeti National Park. On the one hand, we choose the photos with no animals to cover each of the camera sites that would be used in the first binary classification task to determine whether there exist animal present in the image or not. In this work, we balance the binary classification categories with 1:1 and around 20,000 images for each. On the other hand, we selected the animal images based on the available 78,000 (around 11% of the total images) images with bounding box information provided together with the public dataset [1] from season 1-6 to ensure the animal is presented in the frame and filtered out the boxes with unrecognizable size. Combining the bounding box information could improve the labelling quality because the dataset's annotations are only reliable at the sequence level, which means there exist empty images with non-empty labels [2]. We build and test the models with the top twenty wild animal categories for convenience and efficiency. The largest class size is controlled within 3,000 to balance the classes to some extent. All of the images for both of the classification tasks are resized to a 96×96 RGB format that is feasible for inference on the camera.

[1] Serengeti dataset public from LILA BC: http://lila.science/datasets/snapshot-serengeti

Figure 1: An example pair of the original image and augmented crop.

Regarding the training and testing set split, we follow the suggestions from the public dataset [2] to separate them with different camera location (179 sites for training and 46 for testing). In such a splitting method, our model would be trained to increase its generalized ability to identify animals from an unseen location compared to the splitting approach based on the capture events by most previous works, which assumes all of the environment information has been seen already. After the splitting, we have 30,860 and 8,880 images for training and testing for the binary classification model; 20,546 and 6,650 images for training and testing the multi-categories classification.

To be noticed, the animals are highly possible to appear at different location of the same site and the pixels only take a limit proportional to the whole image even we already filter by the bounding box size. In such a scenario, the model would be confused by learning too much on the environment instead of the animals we care during especially the multi-categories animal classification task. Therefore, motivated from the work of Gomez et al. [9], we augment the training data by adding animal images cropped using the bounding box and randomly padding the segmented image within the box with average environment pixel across the sites in its surroundings to adjust them to the desired size. Figure 1 demonstrates examples of the original image together with the augmented one. Thereafter, the model would be expected to learn more about the target animals and increase performance in classification accuracy, which would also be discussed further in the results section. Eventually, we augment the training set with 22,514 cropped animal images and the test set remains the same. The extra training data could also alleviate the over-fitting problem to some extent due to limited training data.

3.2 Models and compression

Task 1: Detecting images with animal

For the first task of binary classification, our model will output the probability that animals exist when given an image as input. We use the balanced animal and empty images dataset to train several CNN models pre-trained on image net using TensorFlow. The models includes MobileNet [25], MobileNetV2 [26], SqueezeNet [27] and VGG [19]. The final prediction results could be further optimized by adjusting the classify threshold for different circumstances.

To deploy the model to the micro-controller, we convert the TensorFlow model to TensorFlow lite format, which is more suitable for an MCU deployment. Meanwhile, the models are compressed using affine quantization [28] from 32-bit floating-point number to 8-bit integer parameters for inference on the MCU after training off-board on Google Colab. The model size is expected to reduce about four times through this compression, and the model would be three times faster when making the inference. In addition to the model size, the model's peak memory usage is another essential criterion to successfully run the model on the camera. It is worth to notice that our system will run multiple models and there would be other application (e.g. taking photos or recording high-quality video) which would also take up memory space when running simultaneously. Therefore, the model should be compressed, considering these factors, and we choose to deploy a binary classification model with minimum memory usage. We could further reduce the model memory usage by controlling the width multiplier for instance in the MobileNet to reduce the model parameters and redundant operations and get our model with minimum memory usage at the cost of few accuracies.

Task 2: Identify animal species

[2] Serengeti dataset public from LILA BC: http://lila.science/datasets/snapshot-serengeti

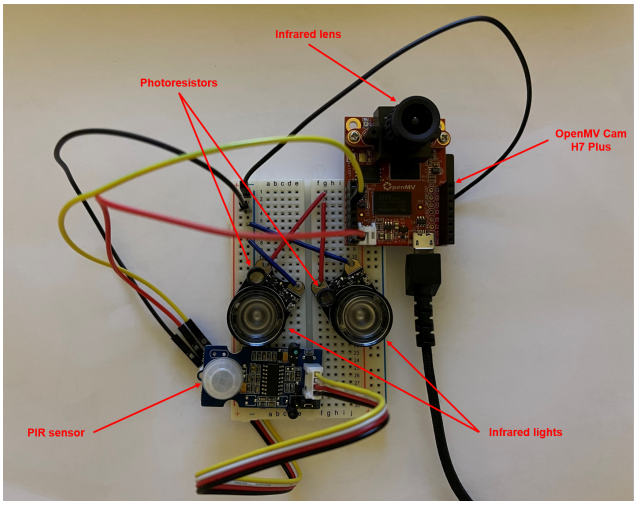

Figure 2: Hardware implementation of AICaTS. (Power bank not shown)

For the second task, the model would predict the probability of the input animal image belonging to each of the twenty categories in our dataset selected from Snapshot Serengeti. Although our model is explicitly training on images with only one animal species, we would also report top-1 and top-3 accuracy as metrics for model selections. The top-3 accuracy metric will be beneficial if the target is to avoid missing the desired category from the top-1 classification result (helpful in decreasing false negatives). Following the first task, we continue taking advantage of the pre-trained CNN models including VGG, Xception, mobile net and mobile net v2. The final prediction performance could be enhanced by augmenting the training data with the segmented animal images, motivated by Gomez et al. [9] mentioned in section 3.1.

Regarding the compression, we also apply the conversion to TensorFlow lite and post-training quantization to 8-bit integer parameters to reduce the model size and memory usage. Additionally, knowledge from cumbersome models like VGG and Xception, which contain up to 20 million parameters could be exploited to increase the performance of a smaller model like MobileNet. As consequences, the prediction accuracy of MobileNet and MobileNet with controlled width could be further improved when training with extra soft labels distilled from the cumbersome teacher models.

3.3 Camera trap implementation

To demonstrate the feasibility and possible benefits of AICaTS, we built a dummy camera trap with machine learning models embedded in. As illustrated in Figure 2, the core of our implementation is an OpenMV Cam H7 Plus, which has relatively similar hardware specifications compared to a commercial camera trap. Apart from a camera senor with up to 2592x1944 (5MP) resolution, this OpenMV Cam has its own ARM Cortex M7 processor running at 480 MHz with 32MB of SDRAM and 1MB of SRAM. In comparison, the only two commercial camera traps that we could find some specifications are a relatively old (2017) and low-end model[29] with 16MB of RAM[30], and a modern mid-range model[31][32] with 1GB of RAM and a processor running at 450 MHz. However, do note that since there are other functions running with the CNN models, and also we need to balance the speed and accuracy of these models, only a small portion of the total memory would be used for machine learning prediction.

Our implementation also includes a PIR sensor for motion sensing, an SD card for storage and two infrared illuminators controlled by photoresistors. The OpenMV Cam actually has two built-in infrared LEDs, but due to OpenMV's hardware design problem, they would illuminate the camera sensor and overexpose the photos. Also, the original standard lens of the OpenMV Cam is replaced by an infrared lens so that the camera could work with infrared flashlights at night. The whole system could be powered by a 3.7V lithium polymer (LiPo) battery, but for convenience, we simply power it with a USB power bank.

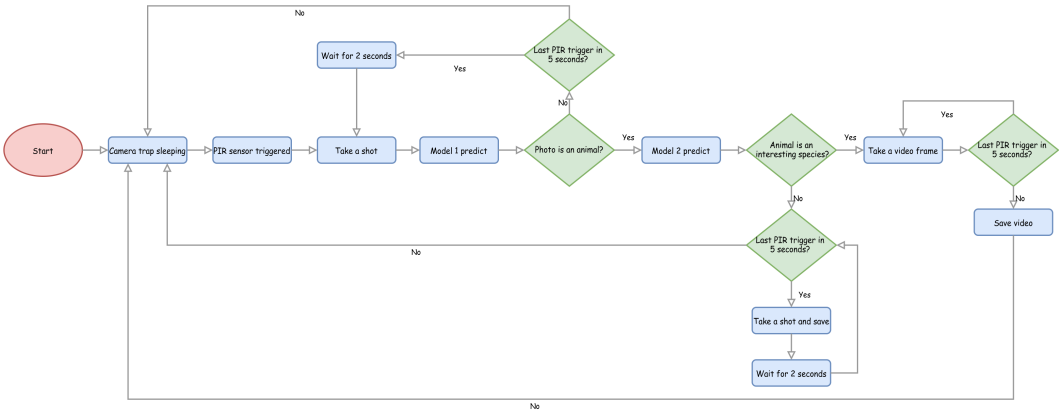

Figure 3: An example AICaTS pipeline.

On the hardware implementation, we then deploy the two-stage models we acquired in a pipeline as shown in Figure 3. After being turned on, the device would automatically go into a sleep or low-power mode. Once the PIR motion sensor is triggered, a shot would be taken followed by an empty-or-animal prediction. If the photo is predicted to be empty, this pipeline chose to take a shot and predict every 2 seconds, then go back to sleep once the PIR sensor is no longer triggered. It is worth noting that there are a lot of alternative strategies. For instance, if one is not very confident about model 1 or one do not want to take the risk of missing any positives, the interval between shots in this loop could be reduced or all shots could be saved for future examination if necessary. On the other hand, if the priority is to conserve power and storage, then the interval could be longer or a maximum number of iterations could be set to the loop. If the presence of an animal is confirmed by model 1, the model 2 would then predict the species of this animal. Here we choose to take videos on interesting species and two 2-second-apart shots on other species, but again there are many other choices in reality. For example, one could try different photo or video resolutions on different groups of species, or one could totally ignore certain species by doing nothing. As mentioned above, if wireless communication hardware is available, a very interesting species may even trigger a notification. Finally, do note that all these choices could be easily appended with some extra "if" statements.

4 Experiments and results

4.1 Model accuracies

During training the first model to detect animal existence in the image, we applied different data preprocessing techniques and finally it shows that randomly rotate the image with 1 to 10 degrees and the random flip are the most effective ones for augmenting the training dataset. Table 1 demonstrates the model performance for the binary classification model with classification accuracy (before and after Model Compression), Peak memory and inference time for one iteration of the compressed TensorFlow lite model. Among the different models we evaluated, VGG 16 has the best performance of 95.0% (after compression). Nevertheless, its peak memory (around 1.1 MB) and inference time (7.8 seconds) are exceptionally high. As a result, the camera cannot perform other tasks like video recording or the other models as most of the memory has already been taken. Since the binary classification model would operate at the first stage of the AICaTS pipeline, it would run for most of the times once the PIR sensor is activated. Thus the inference speed is another essential criterion to select the feasible model for our system. Considering all of the performances, we would choose the MobileNet($\alpha = 0.25$) which have the minimum peak memory usage (73.5 KB) and inference time (0.024 seconds/iteration), though the model accuracy (92.3%) is not the optimum.

Regarding the second task to identify the animal species, we have summarized the model performance of top-1 and top-3 accuracy (before and after model compression) in Table 2. The peak memory usage and model inference time is approximately the same compared to the first binary classification model. Specifically, all models are trained with additional segment animal images except for one MobileNet

Table 1: Evaluation results for Task 1, detect images with animals. The Accuracy includes both original model and compressed model.

Model	**Accuracy %**	**Peak Memory** (B)	**Inference Time**(s)
VGG16	**95.4**, **95.0**	1,179,648	7.8
SqueezeNet	92.4, 92.1	175,232	0.128
MobileNet	94.5, 93.8	301,120	0.294
MobileNet ($\alpha = 0.25$)	93.0, 92.3	**75,280**	**0.024**
MobileNetV2	95.3, 93.5	451,680	0.156
MobileNetV2 ($\alpha = 0.35$)	93.7, 92.5	225,840	0.039

Table 2: Evaluation results for Task 2, identify animal species. The Accuracy includes both original model and compressed model. KD stands for the model trained with Knowledge Distillation, and one of the MobileNet is trained with no bounding box augmentation images (No Bbox)

Model	**Top-1 Accuracy %**	**Top-3 Accuracy %**
VGG16	73.0, —	**88.8**, —
Xception	71.2, —	88.4, —
MobileNet ($\alpha = 0.25$, No Bbox)	51.2, 51.4	75.0, 75.3
MobileNet ($\alpha = 0.25$)	56.0, 54.4	77.4, 76.8
MobileNet ($\alpha = 0.25$, KD)	58.6, 56.2	78.8, 77.8
MobileNetV2	68.6, 63.2	87.2, 84.2
MobileNetV2 (KD)	**73.4**, **69.7**	87.9, **85.9**

for comparison purpose. The increment of prediction accuracies for the MobileNet($\alpha = 0.25$) has proved the effectiveness of augmenting the training data with additional segmented animal images. The two cumbersome models VGG16 and Xception, each with approximately 20 million parameters, are evaluated and trained to choose the best as the knowledge distillation teacher model. We dismiss their performance after compression due to slow inference and high memory usage. By applying knowledge distillation to MobileNet($\alpha = 0.25$) and MobileNetV2 with VGG16 as a teacher, we could further improve their final performance for 1.8 and 6.5 respectively. The overfitting problem is relieved somehow by training with extra soft-labels could be the reason for better results of MobileNetV2 than the teacher model (VGG16). MobileNetV2(KD) could be a good choice with feasible memory size, short inference time and achieve competitive inference performance.

4.2 Power consumption

Since one major purpose of our implementation is to conserve energy, some power consumption statics could help us understand whether energy could really be saved and how much could be saved. As shown in Table 3, the power consumption between shots is much higher than the power consumption during sleep, but also much lower than the power consumption during video recording. Take the pipeline in Figure 3 as an example, if the system is triggered by some background movement that lasts for a while, we only need to take a shot then inference every two seconds. Since a shot and inference would be very fast, most of the time the power consumption is 580mW. On the other hand, without the AICaTS system, the video recording may be triggered by the background movement, and that could drain the battery very quickly. Do note that the OpenMV Cam H7 Plus has an upper limit on its output current, and a typical commercial camera trap usually has a much larger infrared lights array that runs at 12V. Another interesting fact is that in some common environments such as a jungle, infrared lights might be needed the whole day, but maybe not always running at full power. As a result, in reality, the difference in power consumption between taking low-frequency shots and video recording could be much larger than our measurement, which might improve the necessity of AICaTS.

Table 3: Power consumption statistics of our AICaTS hardware. (Excluding PIR sensor power)

Model	Power consumption (mW)
Sleep	195
Between shots (Idle)	580
During shots and inference	1000-2000
Video recording (IR lights off)	1383
Video recording (IR lights on)	2128

5 Discussion

5.1 Limitations

This work's main limitation is that we cannot evaluate our demo version of AICaTS in the wild to detect real animals due to movement constraint under the COVID situation. It would also be instructive to compare the results with an existing commercial camera trap model for further proving the effectiveness of our system. Furthermore, our demo version is not complete to be used directly in the wild since it still requires a more robust battery and disguised outside cover compared to a real Camera Trap equipment. Similarly, the system does not support GUI and network function compared to some existing works, such as, the TrailGuard [33] which could send alert through GSM or satellite when a specific category is recognized. However, our system's ideas are portable to transfer to a different system with further advanced hardware support. Regarding the embedded machine learning models, the performance of the second task, to identify the animal species, might still have potential to increase compared to the high prediction accuracy (94%) in the previous work [11].

5.2 Future work

Therefore, our system, AICaTS, could be extended in several directions. Regarding the hardware implementation, our system's logic could be transferred with other advanced accessories or high-end cameras to support visual display, recording higher resolution videos or wireless communication module for sending notification and previewing captured information. Like the Trailguard [33], the system can also be further extended to identify human beings and prevent poaching in the wild if training with images including human. On the other hand, the embedded model performance improvement is another meaningful direction, especially for the multi-class classification model, if trained with a grander size of training data and more advanced pre-processing and augmentation techniques. Adjusting the models to perform a detection task instead of classification might also be worthwhile to explore motivated by Schneider et al. [13]. It would be beneficial to distinguish animals from the noisy environment features if the model could simultaneously detect the animal bounding box's position. This idea has already been initially proved through involving the segmented animal images in our experiments.

6 Conclusion

We introduced AICaTS, a novel, accessible, and more importantly, an effective pipeline for deploying compressed neural network models on camera traps to save labour, power, storage and enable many other possibilities. Our experiments prove the feasibility of performing image classification with competitive accuracies on camera-trap-level hardware without requiring extra resources. Though cannot be fully tested in the wild, our camera trap implementation which hosts the AICaTS pipeline still demonstrates the potential benefits of AICaTS. Further improvements could focus on building a piece of more robust demo hardware and evaluating the pipeline in the wild.

Acknowledgments

We thank Nicholas Lane, Javier Fernandez-Marques for their guidance and hardware support. We also thank Pedro Porto Buarque De Gusmao for his feedback and technical suggestions.

References

[1] Peter Apps and J. Mcnutt. "How camera traps work and how to work them". In: *African Journal of Ecology* 56 (Dec. 2018), pp. 702–709. DOI: 10.1111/aje.12563.

[2] AB Swanson et al. *Data from: Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna*. 2015. DOI: doi:10.5061/dryad.5pt92. URL: https://doi.org/10.5061/dryad.5pt92.

[3] Tim van Berkel. *Expedition Field Techniques: Camera Trapping*. June 2014. ISBN: 0-907649-93-9.

[4] Xiaoyuan Yu et al. "Automated identification of animal species in camera trap images". English (US). In: *Eurasip Journal on Image and Video Processing* 2013 (Oct. 2013). ISSN: 1687-5176. DOI: 10.1186/1687-5281-2013-52.

[5] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vision* 60.2 (Nov. 2004), 91–110. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[6] T. Ahonen, A. Hadid, and M. Pietikainen. "Face Description with Local Binary Patterns: Application to Face Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.12 (2006), pp. 2037–2041. DOI: 10.1109/TPAMI.2006.244.

[7] G. Chen et al. "Deep convolutional neural network based species recognition for wild animal monitoring". In: *2014 IEEE International Conference on Image Processing (ICIP)*. 2014, pp. 858–862. DOI: 10.1109/ICIP.2014.7025172.

[8] X. Ren, T. X. Han, and Z. He. "Ensemble Video Object Cut in Highly Dynamic Scenes". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 1947–1954. DOI: 10.1109/CVPR.2013.254.

[9] Alexander Gomez Villa, Augusto Salazar, and Francisco Vargas. "Towards automatic wild animal monitoring: Identification of animal species in camera-trap images using very deep convolutional neural networks". In: *Ecological Informatics* 41 (2017), pp. 24 –32. ISSN: 1574-9541. DOI: https://doi.org/10.1016/j.ecoinf.2017.07.004. URL: http://www.sciencedirect.com/science/article/pii/S1574954116302047.

[10] Alexander Gomez et al. "Animal Identification in Low Quality Camera-Trap Images Using Very Deep Convolutional Neural Networks and Confidence Thresholds". In: *Advances in Visual Computing*. Ed. by George Bebis et al. Cham: Springer International Publishing, 2016, pp. 747–756. ISBN: 978-3-319-50835-1.

[11] Mohammad Sadegh Norouzzadeh et al. "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning". In: *Proceedings of the National Academy of Sciences* 115.25 (2018), E5716–E5725. ISSN: 0027-8424. DOI: 10.1073/pnas.1719367115. eprint: https://www.pnas.org/content/115/25/E5716.full.pdf. URL: https://www.pnas.org/content/115/25/E5716.

[12] Marco Willi et al. "Identifying animal species in camera trap images using deep learning and citizen science". In: *Methods in Ecology and Evolution* 10.1 (2019), pp. 80–91. DOI: https://doi.org/10.1111/2041-210X.13099. eprint: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.13099. URL: https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13099.

[13] Stefan Schneider, Graham W. Taylor, and Stefan C. Kremer. "Deep Learning Object Detection Methods for Ecological Camera Trap Data". In: *CoRR* abs/1803.10842 (2018). arXiv: 1803.10842. URL: http://arxiv.org/abs/1803.10842.

[14] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[15] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[16] Deepak Mallya. *AiCatcher: Intelligent camera trap for wildlife conservation*. 2020. URL: https://deepakmallya.com/aicatcher (visited on 01/02/2021).

[17] *Smart Camera Trap*. URL: https://www.hackthepoacher.com/smart-camera-trap (visited on 01/02/2021).

[18] *WAMCam - Wildlife Advanced Monitoring Camera*. URL: https://business.esa.int/projects/wamcam (visited on 01/02/2021).

[19] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[20] Hao Li et al. "Pruning filters for efficient convnets". In: *arXiv preprint arXiv:1608.08710* (2016).

[21] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems* 28 (2015), pp. 1135–1143.

[22] Emile Fiesler, Amar Choudry, and H John Caulfield. "Weight discretization paradigm for optical neural networks". In: *Optical interconnections and networks*. Vol. 1281. International Society for Optics and Photonics. 1990, pp. 164–173.

[23] Aojun Zhou et al. "Incremental network quantization: Towards lossless cnns with low-precision weights". In: *arXiv preprint arXiv:1702.03044* (2017).

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[25] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[26] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[27] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[28] Benoit Jacob et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2704–2713.

[29] *Teardown Tuesday: Motion-Activated Trail Camera*. URL: https://www.allaboutcircuits.com/news/teardown-tuesday-motion-activated-outdoor-trail-camera/ (visited on 01/04/2021).

[30] *M12L128168A-6T Datasheet (PDF) - Elite Semiconductor Memory Technology Inc.* URL: https://pdf1.alldatasheet.com/datasheet-pdf/view/146581/ESMT/M12L128168A-6T.html (visited on 01/04/2021).

[31] *Browning Recon Force Edge*. URL: https://shop.naturespy.org/product/browning-recon-force-edge-wildlife-camera-btc-7e/ (visited on 01/04/2021).

[32] *V35MA Hybrid Camera Controller*. URL: https://www.icatchtek.com/Upload/20190613090857741209 1.pdf (visited on 01/04/2021).

[33] Thomas K Grose. "WILDLIFE PRESERVER". In: *ASEE Prism* 28.6 (2019), pp. 10–10.

Appendix A Access information

We have made our code, implementations and datasets' samples accessible from Google Drive at `https://drive.google.com/drive/folders/1FWRz7Q9NONzzpeQ53nKozujkie_1xTHq?usp=sharing`.

Inside the drive folder, it includes the following contents:

- **Datasets**: the datasets pickle files used to train and test, also includes image examples
- **ModelTraining**: the online notebooks with code to train the models
- **OnboarCode**: the Micro Python code for running the pipeline on the OpenMV camera
- **ProcessData**: the code for processing the images such as resizing, cropping, segmenting or packing
- **OtherFiles**: the meta information of the datasets and relevant files