computers

Article

An IoT System Using Deep Learning to Classify Camera Trap Images on the Edge

Imran Zualkernan [1,*], Salam Dhou [1], Jacky Judas [2], Ali Reza Sajun [1], Brylle Ryan Gomez [1] and Lana Alhaj Hussain [1]

1 Computer Science and Engineering Department, American University of Sharjah, Sharjah 26666, United Arab Emirates; sdhou@aus.edu (S.D.); b00068908@aus.edu (A.R.S.); b00067871@aus.edu (B.R.G.); g00071496@aus.edu (L.A.H.)
2 Conservation Unit, Emirates Nature—WWF, Dubai 454891, United Arab Emirates; jackyjudas@gmail.com
* Correspondence: izualkernan@aus.edu

Abstract: Camera traps deployed in remote locations provide an effective method for ecologists to monitor and study wildlife in a non-invasive way. However, current camera traps suffer from two problems. First, the images are manually classified and counted, which is expensive. Second, due to manual coding, the results are often stale by the time they get to the ecologists. Using the Internet of Things (IoT) combined with deep learning represents a good solution for both these problems, as the images can be classified automatically, and the results immediately made available to ecologists. This paper proposes an IoT architecture that uses deep learning on edge devices to convey animal classification results to a mobile app using the LoRaWAN low-power, wide-area network. The primary goal of the proposed approach is to reduce the cost of the wildlife monitoring process for ecologists, and to provide real-time animal sightings data from the camera traps in the field. Camera trap image data consisting of 66,400 images were used to train the InceptionV3, MobileNetV2, ResNet18, EfficientNetB1, DenseNet121, and Xception neural network models. While performance of the trained models was statistically different (Kruskal–Wallis: Accuracy H(5) = 22.34, $p < 0.05$; F1-score H(5) = 13.82, $p = 0.0168$), there was only a 3% difference in the F1-score between the worst (MobileNet V2) and the best model (Xception). Moreover, the models made similar errors (Adjusted Rand Index (ARI) > 0.88 and Adjusted Mutual Information (AMU) > 0.82). Subsequently, the best model, Xception (Accuracy = 96.1%; F1-score = 0.87; F1-Score = 0.97 with oversampling), was optimized and deployed on the Raspberry Pi, Google Coral, and Nvidia Jetson edge devices using both TenorFlow Lite and TensorRT frameworks. Optimizing the models to run on edge devices reduced the average macro F1-Score to 0.7, and adversely affected the minority classes, reducing their F1-score to as low as 0.18. Upon stress testing, by processing 1000 images consecutively, Jetson Nano, running a TensorRT model, outperformed others with a latency of 0.276 s/image (s.d. = 0.002) while consuming an average current of 1665.21 mA. Raspberry Pi consumed the least average current (838.99 mA) with a ten times worse latency of 2.83 s/image (s.d. = 0.036). Nano was the only reasonable option as an edge device because it could capture most animals whose maximum speeds were below 80 km/h, including goats, lions, ostriches, etc. While the proposed architecture is viable, unbalanced data remain a challenge and the results can potentially be improved by using object detection to reduce imbalances and by exploring semi-supervised learning.

Keywords: deep learning; animal classification; image classification; Internet of Things; image processing; edge computing; animal surveillance; wildlife monitoring; edge computing

check for updates

Citation: Zualkernan, I.; Dhou, S.; Judas, J.; Sajun, A.R.; Gomez, B.R.; Hussain, L.A. An IoT System Using Deep Learning to Classify Camera Trap Images on the Edge. *Computers* **2022**, *11*, 13. https://doi.org/10.3390/computers11010013

Academic Editors: Rytis Maskeliunas and Robertas Damaševičius

Received: 9 December 2021
Accepted: 6 January 2022
Published: 13 January 2022

1. Introduction

Animals' behavior, movements, and locations can be captured using various monitoring techniques. Ecologists often use camera traps to capture images of animals in a non-invasive manner. Camera traps are especially useful because they allow for standardized data collection while causing minimal disturbance to wildlife. Camera traps generate a large amount of data that can be used regionally or globally [1]. Camera traps are typically left in the wild for months, accumulating a large number of images. However, data captured using camera traps often does not reach its full potential due to the costly and time-consuming manual procedures needed to label and classify the captured images. Camera traps are also easily triggered by environmental events such as wind, temperature changes, or vegetation movement. This creates 'ghost' images that do not contain animals and need to be filtered out. A survey conducted by Glover-Kapfer et al. [2] identified image classification as one of the top five camera-trapping limitations. Another top constraint was poor camera-sensor performance that can increase false positives by capturing empty images. Other challenges included remote access to camera trap data. Subsequently, a vision for camera-trapping was formulated, which involved the wireless transmission of data, networks of connected sensors, automation of various processes—such as image filtering and movement tracking—and collaborative tools for managing and analyzing the camera trap data.

This paper addresses two key shortcomings in the current use of camera traps for wildlife monitoring, which are the real-time transmission of data and labelling of images. These challenges are addressed by proposing and evaluating an IoT-enabled deep learning architecture for camera trap image classification. This proposed system is designed to instantaneously detect the type of animal from images captured by the camera traps. The system does so by using deep learning techniques deployed right at the camera trap. Information about the recognized animal is then sent to the ecologists using a low-power network that can be easily deployed in remote locations where camera traps are typically deployed. Each ecologist is provided with a mobile app showing the latest animal sighting, including the location, time, and the type of animal sighted. Using the proposed system, an ecologist who currently has to wait several months for camera trap images to be retrieved from a camera trap, and then has to manually detect the animals in the images, will immediately receive the same information a few seconds after the image is captured by the camera trap without any human intervention whatsoever. The proposed system will hence not only reduce the overall cost of image capture and classification, but also significantly decrease the response time for any interventions or actions the ecologists may decide to take upon a particular animal sighting.

The contributions of this paper include:

- A survey and comparison of the deep learning models used for the task of animal classification
- Proposing an end-to-end IoT camera trap system that uses a low-power, wide-area network to classify the animal images captured in the wild in real-time
- Building and evaluating a deep learning animal classifier based on images from the camera traps
- An experimental evaluation of the commonly embedded edge hardware devices that perform the animal classification on the edge

The paper is organized as follows: Section 2 presents the literature review of the related and relevant works, including the camera trap image classification and edge deployment of deep learning models; Section 3 presents a proposed IoT architecture for the camera traps; Section 4 presents the materials and methods for the study; and Section 5 presents and discusses the results. The paper ends with a conclusion section.

2. Literature Review

2.1. Animal Image Classification Using Convolutional Neural Networks

Ever since the pioneering work of LeCun et al. [3] on handwritten zip code detection, Convolutional Neural Networks (CNNs) have been used for many image recognition problems. CNNs have also shown good results when applied to the animal classification problem, reporting accuracies of 90% and above. For example, Okafor et al. [4] trained a fine-tuned InceptionV1 model to achieve a top-1 test classification accuracy of 99.93% on a Flicker-obtained dataset of 5000 images consisting of bears, elephants, leopards, lions, and wolves. Similarly, Zmudzinski [5] trained GoogLeNet on a 1000-image dataset to distinguish between three species of guinea pig and obtained a classification accuracy of 99.31%. Yousif et al. [6] trained AlexNet specifically for camera trap image classification and obtained an accuracy of 95.6%. Huang and Basanta [7] proposed a novel CNN comprising of five convolution layers that classified a 27-species bird dataset of 3563 images with 95.37% accuracy. This proposed CNN contained skip connections, much like the ResNet architecture [8], which obtained the highest training accuracy when compared to non-residual and Support Vector Machine (SVM) models. Allken et al. [9] used a fish species classifier based on InceptionV3 using a synthetic 10,000-image dataset to obtain an accuracy of 94%. Hu and You [10] used ResNet18 on the 19,465-image Animal-10 dataset to obtain an accuracy of 92%. Lai et al. [11] trained Xception on the Stanford Dogs Dataset of 20,580 images and 120 breeds to achieve a species classification accuracy of 91.29%. Tabak et al. [12] trained a ResNet-18 CNN on a 3.37 million-image dataset comprising of images from five U.S. states and achieved a multi-class classification accuracy of 97.6%. Whytock et al. [13] trained ResNet-50 on a 300,000-image dataset comprising of 26 different species and attained 77.63% accuracy. Schneider et al. [14] also trained DenseNet201 and Xception on the Parks Canada dataset that contained 47,279 images of 55 classes and achieved 95.6% and 95.4% classification accuracy and 0.794 and 0.786 F1-scores, respectively. However, Schneider et al. [14] observed a 26.9% drop in accuracy in both models and a 0.096 and 0.14 drop in F1-scores, respectively, when applying the trained models to animals in unseen locations, raising questions about overfitting of the models. This issue was also highlighted by Tabak et al. [15], who proposed to solve this issue by training ResNet-18 on a mixed dataset comprising of animal images from ten different states in the U.S., achieving a classification accuracy of 96.8% and packaging these trained models into an R package named Machine Learning for Wildlife Image Classification (MLWIC) [16].

Mixed approaches have also been used in animal classification. For example, rather than using individual images for image classification, Shashidhara et al. [17] took a different approach of capturing animal images in "burst", wherein each instance of animal movement was captured not by a single image but by a series of three to five images. These images were then concatenated and represent a single data point whose temporal information could be leveraged to aid in classification. Using this technique, they trained ResNet-152 to achieve an ROC AUC of 0.857. Finally, Norouzzadeh et al. [18] proposed a hybrid approach of combining manual image labelling with CNN image classification. This active learning approach involved training a model to not only classify images, but to determine which images, from a dataset of mostly unlabeled data, were the best candidates for manual labelling in order to aid the model in labelling the rest by itself. Using this approach, they trained ResNet-50 to achieve 91.37% accuracy.

In order to deal with the "non-animal" or ghost images problem prevalent in camera traps, prior classification or object-detection techniques have been used. For example, Vargas-Felipe et al. [19] used SVM and achieved a 99.95% accuracy in classifying between empty images and those with Ovis canadensis. Similarly, Willi et al. [20] trained ResNet18 on the Camera CATalogue dataset to distinguish between animals ("non-empty"), vehicles (under "empty"), and other "empty" objects using randomly initialized weights and achieving an accuracy of 98.0%. They also trained a fine-tuned ResNet18, initialized on weights based on other unspecified camera trap projects, on the Elephant Expedition dataset and achieved up to 96.6% accuracy. Similarly, Chen et al. [21] proposed an AlexNet-based CNN with five convolution layers initialized on AlexNet weights, which achieved a binary (badger vs. no-badger) classification accuracy of 98.05% and a multi-class (six classes: badger, bird, cat, fox, rat, and rabbit) accuracy of 90.32% when trained on an 8368-image dataset. Using drone imagery, Sahu [22] trained U-Net to detect and classify lizards from other background objects using a slightly different two-step approach. The approach included training the model on a dataset with only lizards (positive class) and then training it on the full dataset of background images that contained multiple, single, or no lizards. Using this approach, the model was able to achieve 98.63% accuracy. Similarly, Nguyen et al. [23] used a binary classifier based on VGG-16 and a multi-species classifier ResNet-50 to attain 96.6% and 84.39% accuracies, respectively. Gomez et al. [24] used ResNet101 to achieve a binary (birds vs. no birds) accuracy of 97.5% and multi-class (bird species) accuracy of 90.23%. Others, like Beery et al. [25] and Shepley et al. [26], used object detection techniques to eliminate non-animal images before classification. An object detection approach proposed by Wei et al. [27] outperformed MLWIC [16]. Finally, Cunha et al. [28] compared the performance of the classification and object detection approaches to the empty-image identification problem and showed that object-detection approaches, like EfficientNet-D0 and SSDLite + MobileNet-V2, outperformed classifier-only approaches, like MobileNet-V2, EfficientNet-B0, and EfficientNet-B3.

Besides using state-of-the-art CNNs such as GoogLeNet, AlexNet, ResNet, U-Net, and VGG, many have proposed novel CNN architectures for the animal classification problem (e.g., [7,21]). For example, Teto and Xie [29] proposed a capsule network that achieved a 79% accuracy compared to VGG-16 (96.4%) and InceptionV3 (95%) after being trained on a 500,000-image subset of the Snapshot Serengeti dataset. Rathi et al. [30] proposed a novel three-layer CNN trained on the 27,142-image Fish4Knowledge dataset with a 96.29% classification accuracy. Zuluaga et al. [31] proposed a novel CNN called MixtureNet that is comprised of the concatenated features and last pooling layers from GoogLeNet, ResNet50, ResNet101, and ResNet152, all of which were trained on the 9208-image Alexander von Humboldt database. MixtureNet obtained an accuracy of 92.65% when classifying between ten classes. Tariq et al. [32] proposed a binary classifier comprised of five convolution layers to classify between snow leopards and other animals. In their work, the CNN was trained on an unspecified local dataset of 1500 images and achieved an accuracy of 91%. Jiang et al. [33] proposed a Bilateral Convolution Network (BCNet) comprised of two parallel networks and designed to achieve a trade-off between the classification accuracy and model size in order to be feasible on mobile and edge computing devices. BCNet4 contained two convolution layers and two fully connected layers that performed with 83.9% classification accuracy on a 50-species, 37,322-image dataset with a model size of 16.80 MB and a computational complexity of 0.62 GFlops. This is comparable to another lightweight CNN architecture, EfficientNetB0 (of the EfficientNet series), that has a size of 29.00 MB [34], a complexity of 0.4 GFlops [35], and a macro-F1-score of 0.225 according to the top seven (out of 336) entry on the iWildCam 2019 animal image recognition competition [36]. Finally, an aquatic species classifier based on a CNN with two convolution layers obtained an accuracy of 85.59% [37].

Table 1 shows a summary of applying CNNs for animal image classification.

Table 1. Applications of CNNs on animal image classification.

Author	Model Name	Model Size (MB) [a,*]	Dataset Size	Accuracy (%)
Okafor et al. [4]	InceptionV1	28	5000	99.93
L. Zmudzinski [5]	InceptionV1	28	1098	99.31
R. Sahu [22]	U-Net	62.3	2800	98.63
Tabak et al. [12]	ResNet18	45	3,741,656	97.60
Willi et al. [20]	ResNet18	45	8,740,000	96.60
J. K. Teto & Y. Xie [29]	VGG16	528	500,000	96.40
Rathi et al. [30]	Novel 5L CNN	NR	23,000	96.29
Yousif et al. [6]	AlexNet	233	60,000	95.60
Schneider et al. [14]	DenseNet201	80	47,279	95.60
Y. Huang & H. Basanta [7]	Novel 7L CNN	NR	3563	95.37
Allken et al. [9]	InceptionV3	92	2400	94.00
Zuluaga et al. [31]	MixtureNet	NR	9208	92.65
Hu M. & You F. [10]	ResNet18	45	19,465	92.00

Table 1. *Cont.*

Author	Model Name	Model Size (MB) [a,*]	Dataset Size	Accuracy (%)
Lai et al. [11]	Xception	88	20,580	91.29
Tariq et al. [32]	Novel 5L CNN	NR	1500	91.00
Nguyen et al. [23]	ResNet50	98	107,022	90.40
Chen et al. [21]	Novel 7L CNN	200	8368	90.32
Gomez et al. [24]	ResNet101	171	2597	90.23
Villa et al. [38]	ResNet101	171	783,726	88.90
Miao et al. [39]	ResNet50	98	111,476	88.10
Yin Z. & You F. [40]	VGG16	528	1200	88.00
Jiang et al. [33]	BCNet4	16.8	4200	83.90
Khalifa et al. [37]	Novel 4L CNN	NR	1521	85.59

[a] Besides the papers themselves, model sizes were retrieved from [34,41–43]. * NR = not reported.

Table 1 shows the model used, its size in MB, size of the dataset used, and the accuracy achieved. As the table shows, it was possible to use pre-trained models like Inception and ResNet to obtain high accuracies. In addition, the model sizes varied between 28 MB to 528 MB.

2.2. Using Edge Devices for Camera Trap Animal Classification

Using machine learning and deep learning approaches to process large amounts of data requires considerable computational resources, and therefore, many such resource-intensive programs run on the cloud. However, deploying a deep learning model on the edge could be a better solution and more cost-effective for applications where the network bandwidth is scarce, low latency is required, and/or security is an issue. For example, camera traps are usually deployed in remote areas where the network bandwidth and speed are limited, and therefore, sending all images captured by the camera trap to the cloud is potentially slow and expensive. This is especially relevant because many captured images are empty or contain ghosts and do not provide any interesting or important information. Being able to classify the animal images captured by camera traps on the edge will help eliminate empty images and identify animals on the spot without the need to send anything to the cloud for processing or to collect data manually from the camera traps. IoT cameras, including dash mounted cameras, drone cameras, Google glass, and phone cameras, are already prevalent in our lives and have several applications in law enforcement and traffic monitoring. Such edge cameras allow for the crowdsourcing of videos that are used to find a variety of targets [44]. Camera traps can similarly be incorporated within an IoT ecosystem to help identify animals.

Edge devices like Raspberry Pi [45–52] and Nvidia Jetson [45,53–57] have been previously used for edge-based detection of animals. Raspberry Pi 4 Model B is a single-board computer (SBC) with a Broadcom BCM2711 System-on-a-Chip (SoC) comprising of a Quad-core Cortex-A72 (ARM v8) 64-bit 1.5 GHz CPU and a VideoCore VI 3D 500 MHz GPU. Pi can support up to 8 GB of LPDDR4-3200 SDRAM and uses a microSD for storage. Pi has a two-lane MIPI CSI camera port, HDMI and Display Port for the display output, and 4 USB ports. The Pi also supports Gigabit Ethernet, Bluetooth 5.0, BLE, and 2.4/5 GHz Wi-Fi for connectivity [58]. Similarly, NVIDIA Jetson Nano is another small, low-power computer specifically designed for AI and neural network applications. Nano uses a Quad-core Cortex A-57 CPU clocked at 1.43 GHz and an NVIDIA 128-core GPU. Nano allows the GPU to run neural networks in the TensorRT format. Nano contains a 64-bit, 4 GB LPDDR4 memory with a RAM speed of 25.6 GB/s and a microSD for storage. Nano also has a 2-lane MIPI CSI-2 DPHY camera port, HDMI and Display Port for display output, and 5 USB ports [59]. Google Coral is another potential candidate edge device specially designed for machine learning applications. Coral contains an on-board Edge TPU, which performs up to 4 trillion operations per second. The Coral has a Quad-core Cortex A-53 CPU clocked at 1.5 GHz and an integrated GC7000 Lite Graphics card. Coral has 1 GB of LPDDR4 RAM and uses a microSD for storage. Coral has a 4-lane MIPI-CSI-2 camera port, HDMI and MIPI-DSI for display output, and 4 USB ports. Coral supports Gigabit Ethernet, Bluetooth

4.2, and 2.4/5.0 GHz Wi-Fi for connectivity [60]. Any of the above-mentioned devices can be used to process images on the edge because they support camera, native neural network processing, and microSD image storage capabilities.

Neural network models are typically optimized to run on an edge device. Tensorflow Lite and Tensor RT are two common target edge-optimized frameworks for mobile and embedded devices [61]. However, optimizing a neural network for the edge may have an impact on the speed of inference, power consumption, and performance. For example, Mathur and Khattar [46] compared TensorFlow and TensorFlow Lite versions of their model and noticed that the ResNet-50 took about $\frac{1}{4}$ of the time to classify a single image while drawing twice the amount of power. Similarly, Ramos-Arredondo et al. [53] compared MobileNetV2 against a quantized version and a TensorFlow Lite version. They noticed that the quantized and TensorFlow Lite versions had more frequent false detections. In addition to using the standardized optimization frameworks, custom optimizations can also be utilized. For example, a pruned CNN architecture that resulted in a smaller, less complex network that had the same performance as the original network was proposed by Rohilla et al. [56]. In doing so, they were able to reduce the number of parameters of a five-layer CNN model by 94% (from 40,000 to 2400 parameters).

An earlier work using machine learning methods using edge devices to classify and detect animals automatically was proposed by Matuska et al. [62]. They proposed an Automatic System for Animal Recognition (ASFAR). The system detected and classified animals through the use of distributed watching devices that consisted of a video camera, computation unit, control unit, communication unit, and a supply unit, which could detect an animal presence and send back information to a main computing unit (MCU) for evaluation. The MCU classified unknown objects using a bag of visual points method with an SVM classifier. The model was trained on 300 images from 6 different animals and was tested on 50 images and achieved a classification score of 94%. Similarly, Ramos-Arredondo et al. [53] developed autonomous and portable photo identification by optimizing the image classification algorithms that were based on Scale Invariant Feature Transform (SIFT). The system classified dorsal fin images of the blue whale through offline information processing on a mobile platform. The system was deployed in a Jetson Tegra TK1 GPU platform with 192 cores. The model was trained on 300 images and tested on 1172 images from the CICIMAR-IPN blue whale database. Tests were performed on segmented images and unsegmented images of the blue whale dorsal fin. Results showed that the accuracy of classifying six classes using unsegmented ROIs ranged between 66.27% and 95.36%, while the segmented ROIs showed better accuracy ranging between 79% and 93%.

More recently, Elias et al. [57] proposed an end-to-end distributed data acquisition and analytics system that implemented an IoT architecture consisting of sensors (cameras), edge cloud, and a back-end public or private cloud. This system was implemented for wildlife monitoring and integrated with remote motion-triggered camera traps. The proposed methodology included a novel deep learning training technique where the model was trained using synthetic images constructed using freely available labelled images from Google Images. Those images were combined with empty images captured by camera traps at different times of the day. The trained model, based on InceptionV3, was then transferred to the edge cloud where the classification was performed. Only the classified animals of interest were transmitted to scientists as requested. This approach avoided transferring large training sets to a private or public cloud to train a model, hence reducing the time and bandwidth requirements for image transfer, as well as saving end-user analysis time. Transferring only the classified images required approximately one-third of the time of transferring all the images.

Raspberry Pi is a popular choice for running animal classification on edge. For example, Curtin and Mathews [50] used a Raspberry Pi-based camera system augmented with a deep learning image recognition model for detecting wildlife. The system had a total estimated cost of $50, as compared to the trail cameras which cost upwards of $100. The system used a simple CNN with two convolutional layers to classify images into one of three classes: snow leopard, human, and other. The model was trained on 3600 images, of which 1262 pictures were of human pedestrians from the VIPeR database, 1568 pictures were of snow leopards, and 768 pictures were of natural background environments (for the "other" category) obtained from the ImageNet database. The model was designed to run on the edge so that only pictures of the desired animals were transferred to the user. The system was tested in different settings, one in which the images were pre-downloaded on the Raspberry Pi, and another in which the images were captured and classified in real-time. The Raspberry Pi model was able to classify 347 pre-downloaded images in 29.2 s. However, in the live capture experiment, classification and image capture took 0.12 s per image. The accuracy achieved through the live testing was only 74% for snow leopards, 77% for humans, and 72% for the "other" category which was much lower than the pre-downloaded image results of 96%, 85%, and 99%, respectively. This difference could be because of the lower resolution of the onboard camera. The system consumed 0.10 amperes in the live testing mode with an up time of 10 h using four Alkaline AA batteries. Similarly, Thomassen [47] deployed MobileNet on a Raspberry Pi. The best model achieved an accuracy of 81.1% on the COAT dataset with a model size of 17 MB. The Raspberry Pi had a classification speed of 1.17 frames per second (fps) and could analyze 2.1 million images using a car battery. Liu et al. [63] deployed several image classification models on a Raspberry Pi that could be connected to camera traps to perform image classification. They got the best results from a mean-per-class accuracy of 86.7% using the ResNet-50 224 $\times$ 224 model. Monburinon et al. [49] proposed a hierarchical edge computing solution to detect the intrusion of wild animals in agricultural fields using a CNN based on MobileNet. Their system had more than two times faster evaluation time, consumed less than 6% of the network bandwidth, and was more energy efficient. Finally, Ramos-Arredondo [53] also used a Raspberry Pi as an edge device and selected the MobileNetV2 model, despite not having the highest accuracy nor the fastest inference time because of its low false positive rate, higher precision, and its small size.

Nvidia Jetson represents another candidate edge computing device for animal classification. For example, Liu et al. [63] used MobileNetV2 deployed on a Jetson TX2. MobileNetV2 outperformed MobileNetV1 and InceptionV3 with a validation set accuracy of 92.89%. MobileNetV2 also provided a suitable model size of 40 MB. Rohilla et al. [56] used the Nvidia Jetson TX1 to process images captured by camera traps. They trained a five-layer CNN on 27,370 fish images captured from a live video feed and pruned the trained model by removing all of the zero activation neurons. Arshad et al. [54] used an Nvidia Jetson TX2 as the edge device with a CNN-based YOLOv3 object detector coupled with a discriminative correlation filter-based tracker. The system was used to identify and count deer in the wild, where the deer count was transmitted to a remote server through 4G connectivity. In a 14-day trial, this system was able to process 170 videos that needed 18.5 h of manual processing in 0.3 h, resulting in a 98% reduction in effort. The system was able to identify 17 deer out of the 20 deer present in the trial. Finally, Islam and Valles [55] used NVIDIA Jetson Nano along with a camera module to run a CNN network that could detect snakes, lizards, and toads/frogs from the camera trap images. A commercial camera trap was used in an experiment where the data was captured and saved in a microSD card and was transferred using a portable adapter.

Table 2 summarizes the performance obtained by prior CNN models. As the table shows, mostly the Tensor Flow Lite versions of known models, like Inception V3 or MobileNet-V2, have been deployed. The accuracies range between 70 to 90%. Table 2 also shows that the size of the models varies between 15 to 87 MB. The size is generally proportional to the number of weights, and hence, the number of operations required for inference. Therefore, this will have an impact on the inference time as well.

Table 2. Model Performance of Animal Classifiers on the Edge.

Reference	Model Name	Model Size (MB) *	Dataset Size *	Performance
Zualkernan et al. [45]	TFLite Inception-V3	87	33,948	Accuracy: 92%, F1-Score: 0.90
Thomassen [47]	MobileNet-V2	17	11,211	Avg. precision: 0.61, avg. recall: 0.49, accuracy: 81.1%
Mathur & Khattar [46]	TFLite ResNet-18 (64 × 64), (224 × 224), TFLite ResNet-50 (64 × 64), (224 × 224)	NR	64,000	Mean per-class accuracy: 72.7%, 84.2%, 72.7%, 87.6%
Monburinon et al. [49]	MobileNet	NR	NR	Accuracy: 90%
Curtin et al. [50]	Two-layer CNN	19	3600	Mean accuracy: 74.33%
Tydén & Olsson [52]	Inception-V2 MobileNet-V2 MobileNet-V2 quantized MobileNet-V2 TFLite	55 25 NR NR	4008	Mean Avg. precision: 0.84 Mean Avg. precision: 0.82 Mean Avg. precision: 0.81 Mean average precision: 0.8
Liu et al. [63]	MobileNet-V1 MobileNet-V2 Inception-V3	NR 40 NR	27,406	Accuracy: 92.6% Accuracy: 95% Accuracy: 93.6%

* NR = not reported.

Table 3 shows the hardware metrics when reported. As the table shows, the inference speeds vary widely between 0.09 frames per second to 17.5 frames per second when an edge GPU like the Jetson TX2 was used. Capturing an image on a camera trap depends on the trigger speed, which is how long it takes a motion detector to detect and capture and image followed by the inference speed that determines how long it takes to classify the image as one of the known animals. For example, most commercial camera traps have a trigger delay of between 0.3 and 0.7 s. Therefore, a Pi that can do an inference at 0.09 frames/s or 11 s per frame (Mathur & Khattar [46]) will effectively take about 11.3 s to process a single image. While this delay may be acceptable for slow moving or far away animals, it may not be feasible to capture running animals such as a goat, which can achieve speeds of up to 80 km/h. In general, the ability to capture an image will depend on many other factors, like the camera's sensor size, focal length, and, eventually, the angle and field of view. These are discussed in more detail in the results section of this paper.

Table 3. Hardware Performance of Animal Classifiers on the Edge.

Reference	Edge Device	Power Consumption (W) *	Inference Speed (FPS)	CPU Utilization (%) *
Zualkernan et al. [45]	RPI 4B	4.43	0.56	mean 49.94, max 80
Thomassen [47]	RPI 3B	1.46	1.17	min 48.8, max 57
Mathur & Khattar [46]	RPI 3	max 1 max 2 max 2 max 2	2.56 0.25 1.09 0.09	mean 15
Monburinon et al. [49]	RPI 3B	3.21	min 0.3 max 1.04	NR
Curtin & Matthews [50]	RPI 3B+	max 0.5	8.33	NR
Tydén & Olsson [52]	RPI	NR	1.24 3.18 3.16 3.18	NR
Liu et al. [63]	Jetson TX2	NR	17.57	NR

* NR = not reported.

3. Proposed IoT Architecture

A prototype based on the proposed IoT architecture of a camera trap system shown in Figure 1 was built. As Figure 1 shows, a commercial camera trap (Bushnell Trophy Trail

Camera Trap) equipped with motion sensors can capture images. This camera trap has a trigger speed of 0.7 s and can store 32 GB of images on an SD card. The camera can detect animals up to 24 m using a PIR motion detection sensor. The camera has a night vision flash to capture images over 24 h, day or night. The images captured were stored on a Wi-Fi enabled SD card, which enabled an AIoT edge device (e.g., Jetson Nano) to retrieve each image that was captured. The AIoT device then ran the deployed CNN neural network model to classify the image captured and labelled the image. The label, a timestamp, and the device ID for each image were then sent through an ESP32 single channel LoRa gateway to a RAK2245 LoRaWAN multi-channel gateway, which then forwarded the data to Google's cloud-hosted Firebase database for storage. The ESP32 LoRa gateway is a three-network capable device with an ESP32 WROOM module and an RFM95W LoRa modem. The RFM95W handles the 915 MHz LoRa band, while the ESP32 supports Bluetooth and Wi-Fi. The RAK2245 LoRaWAN can be installed on a tower to handle multiple edge cameras in a region of few kilometers. Edge devices, including the camera, can easily be powered through solar panels in remote locations. This architecture allows a distributed monitoring approach where camera traps can be placed in optimal locations chosen by ecologists and operate independently with only classification notifications being sent back to the centralized server. This is a common approach in IoT systems, and is seen in a variety of applications, like multi-agent anomaly detection [64]. As Figure 1 shows, a mobile application interacted with the real-time Firebase cloud database to alert the ecologists about any new animal spotting.

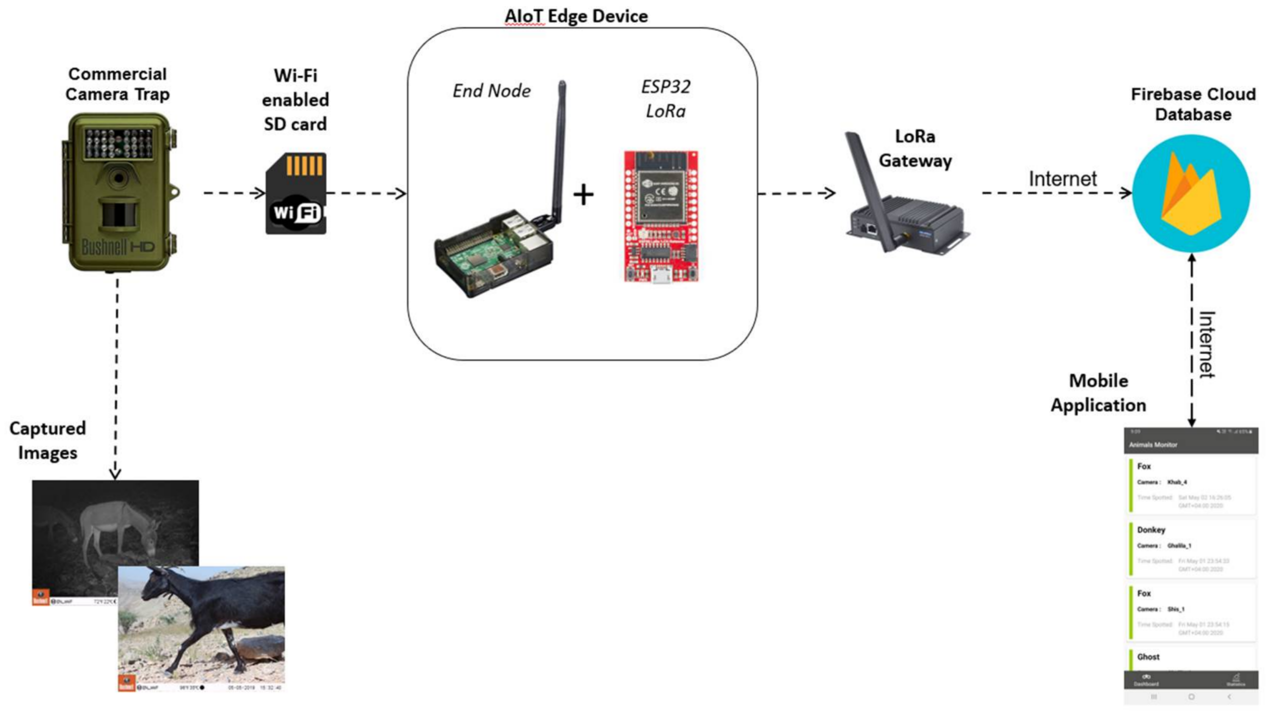
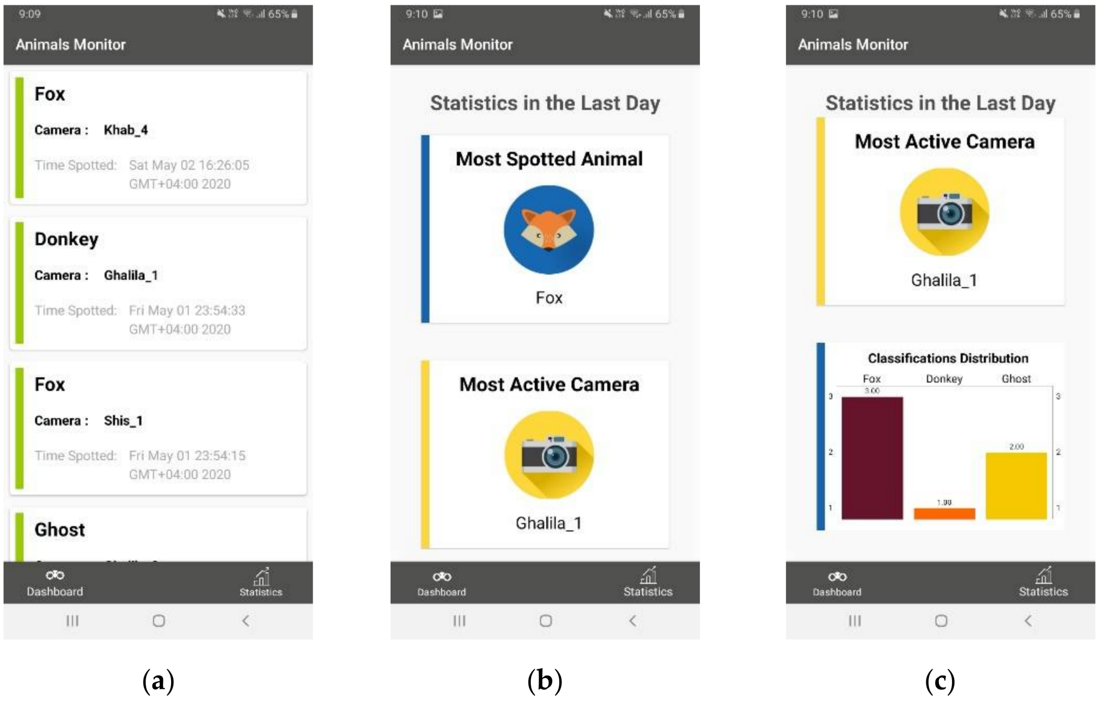


Figure 1. Proposed IoT system architecture diagram.

Figure 2 shows a few screenshots of the mobile application. The dashboard showing the status and history of the captured animals is shown in Figure 2a. As can be seen in the figure, each classification done by each edge device is displayed as a card with the classification label, camera trap location, and the time the animal was spotted. Figure 2b,c shows the statistics pages; the mobile application shows information about the most spotted animal, most active camera, and the disribution of total classifications done in a day.



9:09
Animals Monitor

Fox
Camera : Khab_4
Time Spotted: Sat May 02 16:26:05 GMT+04:00 2020

Donkey
Camera : Ghalila_1
Time Spotted: Fri May 01 23:54:33 GMT+04:00 2020

Fox
Camera : Shis_1
Time Spotted: Fri May 01 23:54:15 GMT+04:00 2020

Ghost

Dashboard Statistics

(**a**)

9:10
Animals Monitor
Statistics in the Last Day
Most Spotted Animal
Fox
Most Active Camera
Ghalila_1
Dashboard Statistics

(**b**)

9:10
Animals Monitor
Statistics in the Last Day
Most Active Camera
Ghalila_1
Classifications Distribution
Fox Donkey Ghost
3.00 1.00 2.00
Dashboard Statistics

(**c**)

Figure 2. A Mobile Application for Camera Traps. (**a**) Recent Classifications, (**b**) Most Reported, (**c**) Distribution.

The proposed architecture in Figure 1 used LoRaWAN, which is an open access low-power, wide-area network (LPWAN) technology [65]. The network has a star topology, meaning the edge devices communicate only with the gateways (e.g., RAK2245) for transferring data. Multiple gateways are connected to a central network server. This protocol is well suited for implementation in an IoT network, as most IoT nodes are battery powered and therefore need a low power solution for the data transfer. However, certain limitations have been reported in terms of the ability of LoRaWAN gateways to handle a large number of packets at the same time [66], and the sensitivity of LoRa in the presence of obstacles and reflectors [67]. The bandwidth limitation of LoRaWAN is not relevant here because there is a low likelihood of a large number of camera traps simultaneously being triggered continuously, and, since the edge devices are responsible for inference, only the labels of the animals observed will be transferred as required. The network design will, however, need to incorporate reflectors and obstacles, especially in the deep bush, hilly environments, or when large bodies of water are present.

4. Materials and Methods

A six-step methodology comprising of data pre-processing, hyperparameter tuning, model training, preparing for edge deployment, model testing, and, finally, hardware testing was followed. A flowchart of the proposed methodology is shown in Figure 3. Each component of the methodology is described next.

4.1. Data Description and Preprocessing

The training dataset used in this paper was a private dataset provided by Emirates Nature-WWF. The sample images from the dataset can be seen in Figure 4. The original data consisted of 66,851 images of 1920 × 1440 pixels, divided into 40 different classes, 39 of which were animal species classes—while one was a non-animal “ghost” class.

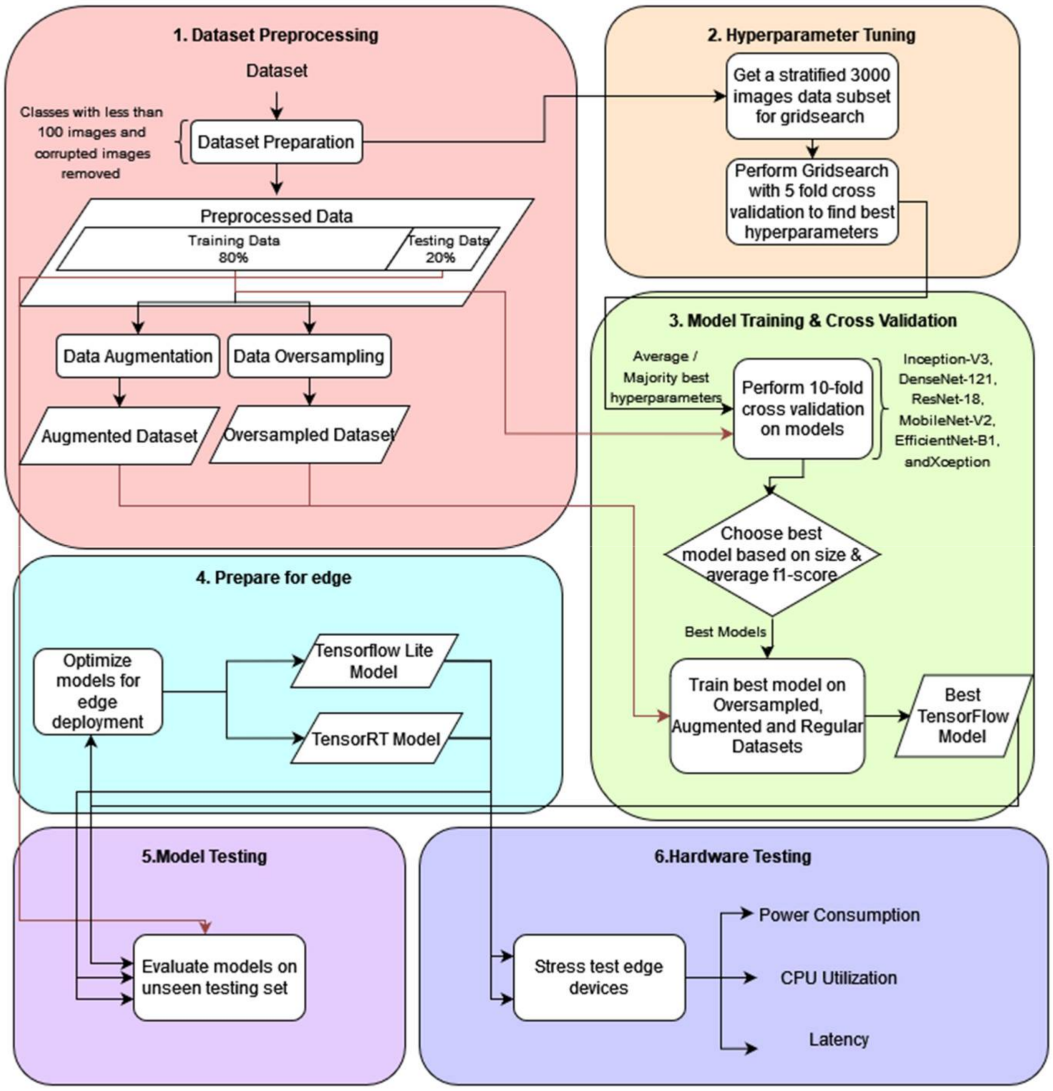

1. Dataset Preprocessing
Dataset
Classes with less than 100 images and corrupted images removed
Dataset Preparation
Preprocessed Data
Training Data 80%
Testing Data 20%
Data Augmentation
Data Oversampling
Augmented Dataset
Oversampled Dataset
2. Hyperparameter Tuning
Get a stratified 3000 images data subset for gridsearch
Perform Gridsearch with 5 fold cross validation to find best hyperparameters
3. Model Training & Cross Validation
Average / Majority best hyperparameters
Perform 10-fold cross validation on models
Inception-V3, DenseNet-121, ResNet-18, MobileNet-V2, EfficientNet-B1, andXception
Choose best model based on size & average f1-score
Best Models
Train best model on Oversampled, Augmented and Regular Datasets
Best TensorFlow Model
4. Prepare for edge
Optimize models for edge deployment
Tensorflow Lite Model
TensorRT Model
5.Model Testing
Evaluate models on unseen testing set
6.Hardware Testing
Stress test edge devices
Power Consumption
CPU Utilization
Latency

Figure 3. Flowchart of Proposed Methodology.

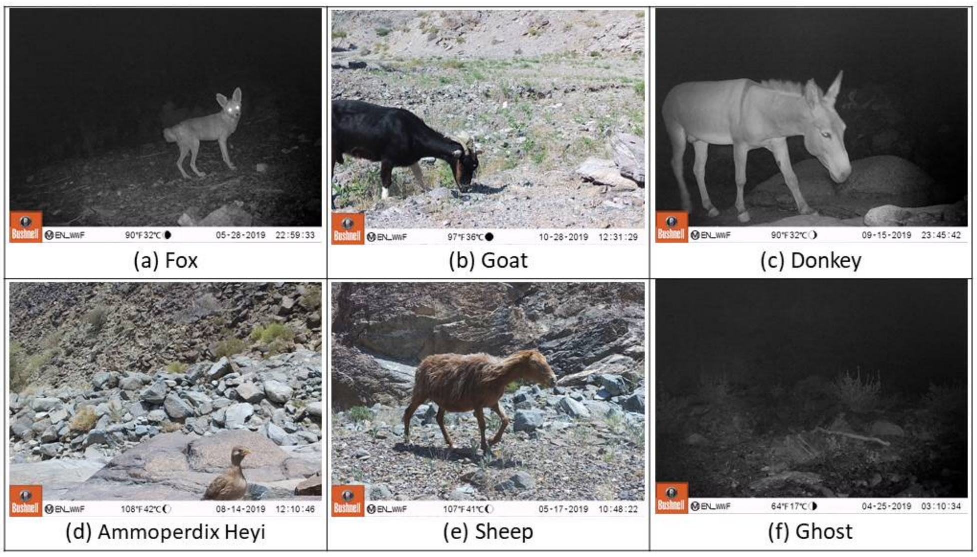


Figure 4. Sample images from the Emirates Nature-WWF dataset.

Due to a significant class imbalance, classes that contained less than 100 images were removed. This resulted in a dataset containing 66,513 images divided into 15 classes, 14 of which were animal species and one of which was non-animal or "ghost". Finally, corrupted images were removed, resulting in a final dataset size of 66,400 images. Table 4 shows the image distribution for the various classes in the dataset. It should be noted that, primarily due to the goat and ghost class, the data was still highly imbalanced.

Table 4. Dataset Class Distribution.

Class Name	No. of Images
Acomys cahirinus (AC)	196
Ammomanes deserti (AD)	150
Ammoperdix heyi (AH)	821
Camel	100
Cat	216
Dog	475
Donkey	3284
Emberiza striolata (ES)	186
Fox	9275
Ghost	34,549
Goat	15,617
Oenanthe albonigra (OA)	287
Rat	138
Sheep	779
Spilopelia senegalensis (SS)	327
Total	66,400

4.2. Model Selection and Hyperparameter Tuning

Based on the literature review, Inception-V3, DenseNet-121, ResNet-18, MobileNet-V2, EfficientNet-B1, and Xception were selected for training and evaluation. All models were pre-trained on ImageNet. The pre-trained Inception-V3, DenseNet-121, ResNet-18, and MobileNet-V2 were used from Zualkernan et al. [45]. The original data consisting of 66,400 images were split into an 80:20 training/testing split using stratified sampling to preserve the class imbalance. The 20% test data was unseen and reserved for testing only.

A grid search was conducted to find appropriate hyperparameters for each model. The datasets for the grid search were derived from the dataset in Table 4 by taking the stratified 3000-image subsets of the complete dataset. The hyperparameter tuning was conducted on all six models using GridSearchCV [68]. The grid search training and testing models used every single combination of the values of the various hyperparameters provided [69]. This type of search is exhaustive in nature, making it much slower but more systematic than alternative methods, like random search. Even though this method is slower, it can easily be parallelized because the different hyperparameters used in training are usually independent of each other.

In this research, batch sizes of 8, 16, and 32, and learning rates of 0.001, 0.01, and 0.05 were used to conduct a grid search for the best combination of hyperparameters. A five-fold cross-validation was used in the grid search to arrive at the best batch size and learning rate for each model. This means that the grid search was conducted by systematically varying the training and testing data using five folds; each time four out of five equal segments of the 3000-image subset were used to train, and the remaining segment was used to test the resulting model. The average performance across the five folds was then used to determine the best hyperparameters for each model. Table 5 shows the parameters for each of the models used in the training process.

4.3. Model Training and Cross Validation

Each of the chosen models were then trained for 50 epochs using the appropriate hyperparameters and using the Stochastic Gradient Descent (SGD) optimizer. All layers of the pre-trained models were unfrozen to ensure appropriate learning of the unique features of the dataset. Furthermore, in order to avoid overfitting, a ten-fold cross-validation strategy is applied, and the results averaged out for each of the models to allow for proper comparison.

Table 5. Parameters for the various CNN Models evaluated.

Model Name	Batch Size	Learning Rate	No. of Weights (million)	Size (MB)	Input Size (pixels)	No. of Fully Connected Layer Neurons
InceptionV3	16	0.05	23.9	92	299 × 299	128
MobileNetV2	16	0.01	3.5	14	224 × 224	512
ResNet18	16	0.01	11.7	45	224 × 224	128
EfficientNetB1	8	0.05	7.9	31	224 × 224	512
DenseNet121	16	0.01	8.0	33	224 × 224	128
Xception	8	0.05	22.9	88	299 × 299	128

4.4. Preparing for Edge Deployment

Based on the previous cross-validation step, the best model was selected for edge deployment. This model was converted to the two common optimized model formats, namely TensorFlow Lite [70] and TensorRT [71]. This allowed for deployment onto the selected edge devices.

The converted edge-optimized models were then deployed onto three different edge devices, namely the Raspberry Pi 4, the Nvidia Jetson Nano, and the Google Coral Dev Board. The specifications of each of these devices are discussed in Section 2.2. While the TensorFlow Lite model was deployed on the Raspberry Pi and the Coral Dev Board, both the models were deployed on the Jetson Nano in order to enable a comparison.

4.5. Testing and Evaluation

In order to better evaluate the edge-optimized models, performance testing (i.e., calculating the F1-scores) was carried out by classifying the 20% unseen data set using the original Xception model as well as the edge-optimized models.

For the hardware evaluation, each edge device (e.g., Jetson Nano) was stress-tested by doing 1000 consecutive image classifications on each hardware device. Each edge-optimized model was tested on each hardware device. Each hardware device used its respective camera modules to create an extreme situation where intensive wildlife activity triggers the system continuously. Each image classification consisted of the device camera capturing the image, pre-processing the image to get the image ready as an input to the neural network model, and then running the neural network to classify the captured image. Resources like the current consumption and CPU utilization of each device were measured over the 1000 image classifications for each model for each edge device. In addition, time taken for each camera module to capture an image, time taken to pre-process the image (i.e., resizing and scaling), and the time taken for the neural network model to classify or infer an animal label were also measured. A comparison was then performed between each of the devices in order to determine the best device for deployment in the field.

5. Results

5.1. Performance of Various Classifiers

Figure 5 shows the sample loss curves of the trained models. As the figure shows, while some models like ResNet-18 showed some volatility, no systematic overfitting was observed.

Table 6 shows the results of a ten-fold cross-validation across all six models, including the average accuracy and F1-score, standard deviation of each performance metric, and size of each model. As the table shows, Xception was the best model with an average F1-score of 0.878 and an average accuracy of 96.1%. MobileNet V2 performed the worst with an accuracy of 95.4% but with a significantly smaller size of 18.7 MB, as opposed to 167 MB for the Xception model.

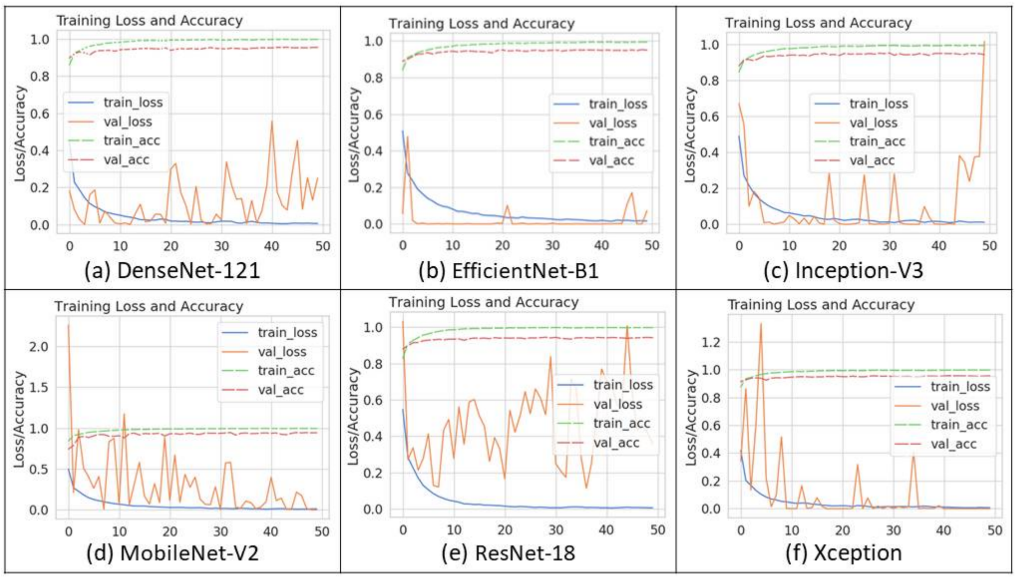


Figure 5. Loss curves of trained models.

Table 6. Ten-Fold Summary Results.

Model Name	Average Accuracy	Std. Accuracy	Average F1-Score	Std. F1-Score	Size (MB)
MobileNetV2	0.954	5.16×10^{-3}	0.858	1.23×10^{-2}	18.7
InceptionV3	0.960	2.34×10^{-16}	0.864	1.71×10^{-2}	174.0
ResNet18	0.955	5.27×10^{-3}	0.858	1.40×10^{-2}	87.5
EfficientNetB1	0.960	2.34×10^{-16}	0.863	1.34×10^{-2}	52.1
DenseNet121	0.960	4.71×10^{-3}	0.871	1.60×10^{-2}	55.2
Xception	0.961	3.16×10^{-3}	0.878	1.03×10^{-2}	167.0

Figure 6 shows a graphical representation of the average F1-score versus the size comparison between the MobileNetV2, InceptionV3, ResNet18, EfficientNetB1, DenseNet121, and Xception. The figure shows that, as expected, the larger models resulted in better performances. However, as the figure shows, there was only a 3% increase in the F1-score between Xception and MobileNetV2, while MobileNetV2 was 1/10th the size of Xception.

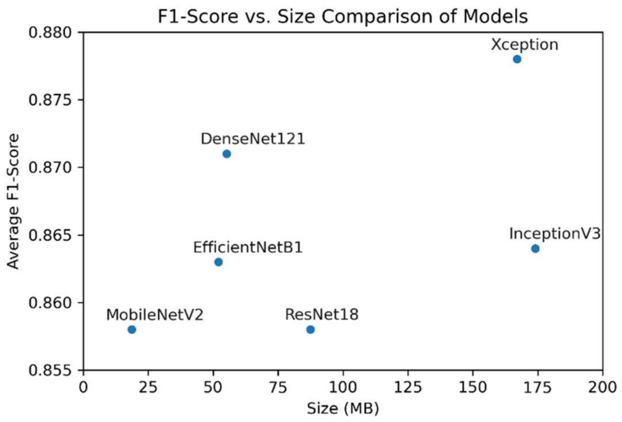


Figure 6. K-Fold Average Model F1-scores vs. Size.

Accuracy (Kruskal–Wallis; H(5) = 22.34, $p < 0.05$) and F1-score (Kruskal–Wallis; H(5) = 13.82, $p = 0.0168$) were statistically different across the six models. Table 7 displays the Kruskal–Wallis H-test ranks. The table clearly shows that Xception was statistically the best model, followed by DenseNet.

Table 7. Kruskal-Wallis H-test Ranks.

Model	F1-Score Median Rank	Accuracy Median Rank
Xception	45.3	37.9
DenseNet	36.05	35
InceptionV3	29.3	35.5
EfficientNetB1	28.7	35.5
ResNet-18	22.3	21
MobileNetV2	21.35	18.1

Since the performance of the various models was so similar, it is instructive to determine if the models made different types of errors. Adjusted Rand Index (*ARI*) and Adjusted Mutual Information (*AMU*) scores across the models were computed to determine how closely their answers—and hence, errors—matched. The formulas for these metrics are shown in Equations (1) and (2), respectively.

$$ARI = \frac{\Sigma_{ij}\binom{n_{ij}}{2} - \frac{\Sigma_i\binom{a_i}{2}\Sigma_j\binom{b_j}{2}}{\binom{n}{2}}}{\frac{\Sigma_i\binom{a_i}{2}+\Sigma_j\binom{b_j}{2}}{2} - \frac{\Sigma_i\binom{a_i}{2}\Sigma_j\binom{b_j}{2}}{\binom{n}{2}}} \tag{1}$$

where:

n_{ij} is the value of an element in row i and column j in the contingency table;
a_i is the sum of the elements in row i;
b_j is the sum of the elements in column j;
n is the total count in the table.

$$AMU(U,V) = \frac{MI(U,V) - E(MI(U,V))}{avg(H(U),H(V)) - E(MI(U,V))} \tag{2}$$

where:

$MI(U,V)$ is the mutual information between $U\ and\ V$;
$E(MI(U,V))$ is the expected mutual information between $U\ and\ V$;
$H(U)$ is the entropy associated with U.

The resulting computed scores are shown in Tables 8 and 9. As the tables show, both the *ARI* and *AMU* were high, indicating that there was a close correspondence among the models. This means that the models generally made the same types of errors. As well, this means that the models were qualitatively making the same type of inference based on the data, and none of the models could be preferred based on the types of errors they made.

Table 8. *ARI* Scores Comparing Various Models.

	DenseNet-121	EfficientNet-B1	Inception-V3	MobileNet-V2	ResNet-18	Xception
DenseNet-121	1	0.9249	0.9057	0.915	0.9141	0.9262
EfficientNet-B1	0.9249	1	0.8997	0.9126	0.9003	0.9244
Inception-V3	0.9057	0.8997	1	0.8873	0.8849	0.9181
MobileNet-V2	0.915	0.9126	0.8873	1	0.9005	0.9104
ResNet-18	0.9141	0.9003	0.8849	0.9005	1	0.9058
Xception	0.9262	0.9244	0.9181	0.9104	0.9058	1

Table 9. *AMU* Scores Comparing Various Models.

	DenseNet-121	EfficientNet-B1	Inception-V3	MobileNet-V2	ResNet-18	Xception
DenseNet-121	1	0.8692	0.8543	0.8542	0.8568	0.8772
EfficientNet-B1	0.8692	1	0.8466	0.8508	0.8376	0.8709
Inception-V3	0.8543	0.8466	1	0.8307	0.8231	0.8696
MobileNet-V2	0.8542	0.8508	0.8307	1	0.842	0.8517
ResNet-18	0.8568	0.8376	0.8231	0.842	1	0.8467
Xception	0.8772	0.8709	0.8696	0.8517	0.8467	1

5.2. Analysis of Best Classifier

Figure 7 shows the confusion matrix of the best classifier (Xception) when tested on the 20% unseen data set. The matrix shows that the primary source of errors was the Goat class that got confused with Acomys Cahirinus (a type of Mouse) and Spilopelia Senegalensis (a type of Pigeon). One reason for goats getting confused could be that, in many of the images, the goats appeared as a herd rather than a single goat. It is also important to point out that Goats was a majority class, so it is natural to expect this class to be confused with many others.

In order to determine if the class imbalance could be addressed using data augmentation or oversampling, the Xception model was retrained using data augmentation and oversampling. As Table 10 shows, the random oversampling yielded the best results, while the visual augmentation yielded the worst results. Oversampling increased the average F1-score from 0.87 to 0.97, clearly indicating that class imbalance was the problem. The Synthetic Minority Oversampling Technique (SMOTE) was also tried and performed worse than random oversampling.

Table 10. Xception Ten-Fold Performance with Different Sampling Techniques.

Dataset	Average Accuracy	Average F1-Score
Regular	0.961	0.878
Oversampled	0.967	0.977
Augmented	0.947	0.829

5.3. Results of Deploying on the Edge Devices

5.3.1. Size

The best model, Xception, was converted to the quantized TensorFlow Lite and TensorRT formats. Table 11 shows the resulting sizes, showing that the original TensorFlow model was reduced to 20 MB (87% reduction in size) and 82 MB (50% reduction in size) when using TensorFlow Lite or TensorRT, respectively. This reduction in size allowed the models to be deployed onto various edge devices in order to benchmark performance.

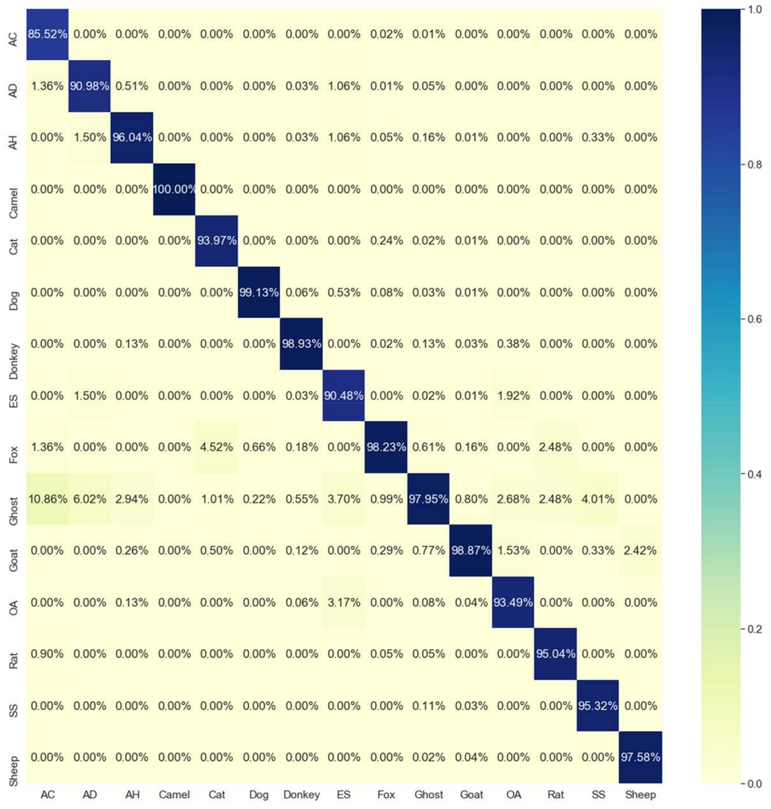


Figure 7. Confusion Matrix of Xception Model.

Table 11. Xception Models Sizes by Optimization.

Model	Size (MB)
TensorFlow (Original)	165
TensorFlow Lite	20
TensorRT	82

5.3.2. Model Performance

Evaluation was then conducted on the performance of each of the edge-optimized models, as well as on the original unoptimized Xception model using the 20% unseen test set. The F1-scores for the TensorFlow Lite and TensorRT models compared with those of the original model are shown in Table 12. As the table shows, while there was no overfitting, as evidenced by the learning curves shown in Figure 5, there was a significant drop in performance for both TensorFlow Lite and TensorRT models, with the F1-score dropping from 0.977 to 0.7. This performance drop is directly attributed to the optimization performed on the original Xception model during the conversion to TensorFlow Lite and TensorRT. While most classes dropped in performance, the most pronounced reduction was seen for the minority classes. For example, Camel and Ammomanes deserti were the smallest classes and had the worst performance. However, the classifier remained robust in the cases of the majority classes such as Ghost, Donkey, Fox, and Goat.

Table 12. Xception Model Performance on Reduced Models.

Class	Original F1-Score	TF Lite F1-Score	Tensor RT F1-Score
Ghost	0.99	0.95	0.95
Fox	0.98	0.92	0.92
Goat	0.98	0.92	0.92
Donkey	0.99	0.91	0.92
Sheep	0.98	0.89	0.89
AC	0.91	0.87	0.86
ES	0.91	0.79	0.79
AH	0.94	0.72	0.74
Cat	0.90	0.71	0.73
Rat	0.89	0.68	0.65
Dog	0.97	0.64	0.63
SS	0.91	0.61	0.61
OA	0.89	0.45	0.43
AD	0.86	0.31	0.32
Camel	1.00	0.18	0.18
Accuracy	0.98	0.92	0.93
Macro F1-Score	0.94	0.7	0.7
Weighted Average	0.98	0.92	0.92

5.3.3. Resource Utilization

The edge-optimized models were deployed on each edge device and evaluated in terms of current, CPU utilization, and inference time by performing a stress test of 1000 consecutive inferences. Table 13 shows the current consumed and the CPU utilization over the 1000 inferences for each of the devices. As the table shows, Raspberry Pi consumed the least current when running the TFLite model in addition to also consuming the least amount of CPU resources, and thus leaving sufficient CPU resources for other tasks, like networking and communications. Jetson Nano was the worst, consuming twice as much current with a similar CPU utilization.

Table 13. Power Consumption and CPU Utilization by Edge Device (1000 inferences).

Device (Model)	Average Current (mA)	Max Current (mA)	Average CPU Util. (%)	Max CPU Util. (%)
Raspberry Pi (TFLite)	838.99	977	23.62	57.3
Google Coral (TFLite)	1074.88	1140	50.41	56.8
Jetson Nano (TFLite)	874.37	1057	34.85	60.6
Jetson Nano (TensorRT)	1665.21	2005	27.14	53.7

Table 14 shows the time taken for each of the stages outlined previously. As the table shows, Jetson Nano was about 12 times faster than Raspberry Pi. One difference seems to be due to the capture time, which took 0.518 s on the RPi camera as opposed to 0.001 s for the Jetson Nano. As expected, Jetson Nano performed badly on the TFLite model because it is optimized to run TensorRT models. However, surprisingly, Google Coral did not perform that much better than a Raspberry Pi despite using the native TensorFlow Lite model. Even if the capture time is ignored due to camera variability, it is clear that Jetson Nano outperformed the other platforms by a wide margin. Jetson Nano was able to capture and classify an animal image roughly once every 0.3 s.

Table 14. Average Latency (Standard Deviation) per Image by Edge Device (1000 inferences).

Device (Model)	Capture Time (s)	Pre-Process. Time (s)	Inference Time (s)	Total Time (s)
RPI-TFLite	0.518 (0.014)	0.011 (0.002)	2.835 (0.036)	3.365 (0.039)
Coral-TFLite	0.013 (0.001)	0.014 (<0.000)	2.739 (0.002)	2.765 (0.002)
Nano-TFLite	0.001 (<0.000)	0.006 (<0.000)	4.316 (0.026)	4.324 (0.026)
Nano-TensorRT	0.002 (<0.000)	0.006 (<0.000)	0.276 (0.002)	0.283 (0.002)

Given the capture times recorded for the respective camera modules, the field effectiveness of each edge device can be calculated by the types of animals it can capture in full sprint. For example, is the camera able to capture a running cheetah from one meter away? This calculation depends on the field of view of the camera at a particular distance. Field of view indicates the geometric bounds of what a camera can record. Field of view for a camera can be calculated using Equations (3) and (4).

$$Angle_{of_{view}} = 2 \times tan^{-1}\left(sensor_{width/(2\times focal_{length})}\right) \times (180/\pi) \tag{3}$$

$$Field\_of\_view = 2 \times tan(angle\_of\_view/2) \times distance\_to\_subject \tag{4}$$

For example, the Camera Module V2 on a Raspberry Pi uses the Sony IMX219 sensor, which has a sensor size of 3.68 × 2.76 mm (4.6 mm diagonal), and hence, a horizontal angle of view of 62.2 degrees and a vertical angle of view of 48.8 degrees. A review of 55 current commercial camera trap models from various companies (e.g., Browning, Bushnell, Covert, Moultrie, Spypoint, etc.) showed that the field of view of commercial cameras ranged between 22 to 125.9 degrees and the mean angle of view of commercial cameras was 42.15 degrees (s.d. = 13.14). Figure 8 shows the horizontal field of view for the Camera Module V2 and for an average commercial camera trap at various distances from the camera. For example, if the animal was 4 m away from the camera, the RPI camera can horizontally record only 4.83 m, while an average commercial camera can record about 3 m width for the field of view. Figure 8 also shows that the RPI camera will outperform an average commercial camera trap because it can cover a wider field of view at the same distance, owing to its larger angle of view.

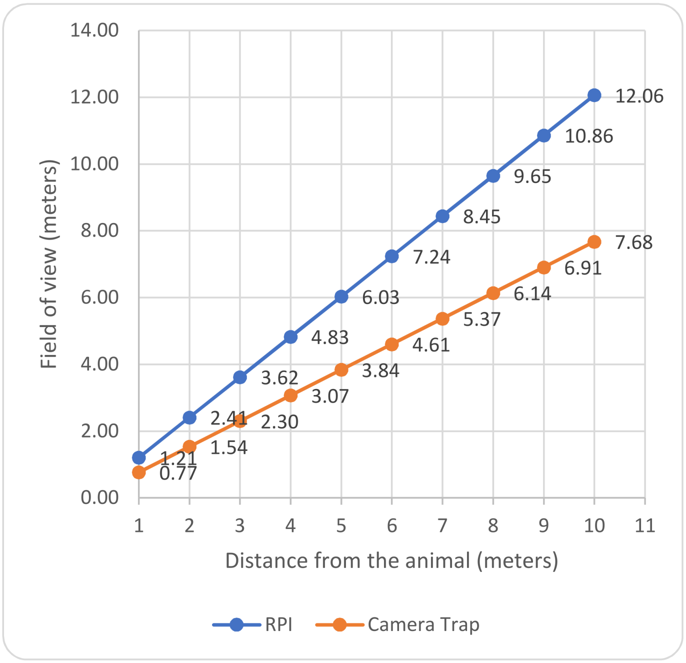


Figure 8. Field of view for the Raspberry Pi Camera Module V2 and an average camera trap.

Table 15 shows the minimum distance required to capture an animal in full flight using the RPI camera. For example, a mouse's maximum speed is 13 km/h—or it takes 0.28 s to cross a meter. From Table 15, it is evident that a Pi can only capture the image of the mouse if the mouse is 10.07 m away from the camera. A Nano, on the other hand, can capture a fleeing mouse at as close as 0.85 m from the camera. Table 15 clearly shows that Nano was the only viable option because neither Coral nor Pi could guarantee that they could capture a fleeing mouse at less than 8 m away. Realistically, a Nano could comfortably capture animals as fast as a goat. Many other animals, like elk, ostriches, lions, and kangaroos, are about the same or less in terms of their maximum speed. It should be indicated that, if used with the currently available commercial camera traps like the Bushnell Trophy Trail Camera, there is a trigger limit of 0.3 to 0.7 s, which means that the camera will take at least 0.3 to 0.7 s to capture the image and, hence, even the performance of the Nano will be downgraded to about 1 image per second. This limitation can easily be obviated by using the native camera of Nano and adding a PIR motion detection sensor to it. PIR is the same sensor used in many commercial camera traps. The PIR can detect movement of up to 30 m, so the Nano will be able to detect movement from up to 30 m away.

Table 15. Minimum Distance Required to Capture a Fleeting Animal.

Animal	Maximum Speed			Minimum Distance to Capture (meters)		
	km/h	m/s	s/m	Coral	Pi	Nano
Cheetah	120.00	33.33	0.03	76.39	92.97	7.82
Goat	80.00	22.22	0.05	50.93	61.98	5.21
Wild Dog	72.50	20.14	0.05	46.15	56.17	4.72
Cat	48.00	13.33	0.08	30.56	37.19	3.13
Elephant	40.00	11.11	0.09	25.46	30.99	2.61
Brown Bear	35.00	9.72	0.10	22.28	27.12	2.28
Black Mamba	32.00	8.89	0.11	20.37	24.79	2.09
Squirrel	20.00	5.56	0.18	12.73	15.50	1.30
Mouse	13.00	3.61	0.28	8.28	10.07	0.85

5.4. Discussion of Results

A number of observations can be made. First, most neural network models performed reasonably well, and the selection of a model will depend primarily on how mission critical it is to not make errors. However, the practical difference between the smallest (i.e., MobileNet) and the largest model (i.e., Xception) is only 3% in terms of the F1-score. Secondly, at least in this instance, the different models made similar errors and, hence, the types of errors are not a good criterion for selecting a model. In addition, the model performance on the minority classes was significantly worse after optimizing the models for edge devices. This was observed for both the TensorFlow Lite and TensorRT frameworks. This means that there is a need to better handle the unbalanced data problem. Ghost images are one source of this imbalance and perhaps object-detection approaches can be explored to filter this data out in the first stage of a multi-stage approach, as is indicated by Cunha et al. [28]. Semi-supervised learning techniques represent another venue to partially mitigate the minority class issues faced in this research. Semi-supervised learning involves leveraging a small set of labeled data with a much larger set of unlabeled data to train a model [72]. Therefore, it is plausible that using semi-supervised learning with the small data sets of minority classes could result in better performance. It was also observed that when choosing an edge architecture (e.g., TensorFlow Lite versus TensorRT), a smaller model size did not guarantee a faster response time, as the deploying hardware architecture was a mediating variable. For example, both TensorFlow Lite and TensorRT resulted in a similar performance reduction, and while the TensorFlow Lite model was one fourth the size of the TensorRT model, the TensorRT outperformed the TensorFlow Lite model with ten times less latency. Finally, the hardware of the edge device does make a difference because, in this instance, the only viable hardware option was the Jetson Nano due to its GPU cores.

6. Conclusions

While this paper has proposed a general IoT-based architecture for camera traps for the end-to-end monitoring of animal images, there are certain limitations to the results. Firstly, only a single dataset was used for model training and evaluation. A more robust solution for wildlife monitoring could evaluate performance across different camera trap datasets, perhaps with different camera trap models and terrain locations. Secondly, a rather significant drop in performance was seen in the minority classes when the models were converted for edge deployment. This clearly pointed to there not being enough data for the model to have fully learned the boundaries for these classes. Therefore, alternative learning strategies borrowed from semi-supervised deep learning could be used. This is especially appropriate for this situation since labelling images to produce further training data was a very time-consuming process and therefore, using semi-supervised algorithms, such as FixMatch [73], can allow the unlabeled images themselves to be leveraged alongside the existing labelled dataset.

In summary, this paper showed that the proposed IoT camera trap system can support ecologists as they monitor animals in the wild by providing them instant image classification, and saving the time and effort required to manually collect camera trap images and label them. Hence, using deep learning techniques on IoT edge devices can become a great asset for wildlife monitoring. Six different CNN models were trained, following which Xception was chosen as the network to be deployed having had an average accuracy of 0.98 and F1-score of 0.94. Using an oversampled dataset yielded the best results compared to the regular and augmented datasets with an average accuracy of 0.967 and an F1-score of 0.977. After the deployment of the Xception network on three edge devices—Raspberry Pi, Nvidia Jetson, and Google Coral Board—a trade-off between the performance and power consumption was noticed. The Raspberry Pi edge device was the least power consuming with a low CPU utilization, while the Nvidia Jetson used almost double the power but took 12 times less time to classify captured images. Despite utilizing the most power, Jetson Nano was found to be the only viable edge device candidate in this situation.

Author Contributions: Conceptualization, I.Z. and S.D.; Methodology, A.R.S., B.R.G. and L.A.H.; Software, A.R.S., B.R.G. and L.A.H.; Validation, A.R.S., B.R.G. and L.A.H.; Formal Analysis, A.R.S., B.R.G. and I.Z.; Investigation, A.R.S., B.R.G. and L.A.H.; Resources, I.Z., S.D., A.R.S. and B.R.G.; Data Curation, J.J.; Writing—Original Draft Preparation, A.R.S., B.R.G. and L.A.H.; Writing—Review & Editing, I.Z. and S.D.; Visualization, B.R.G.; Supervision, I.Z., S.D. and J.J. All authors have read and agreed to the published version of the manuscript.

Funding: The work in this paper was supported, in part, by the Open Access Program from the American University of Sharjah [grant number: OAPCEN-1410-E00036].

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The code used for training the models in this work can be found at https://github.com/AliRSajun/KerasCNN_Automated (accessed on 23 November 2021).

Acknowledgments: The authors would like to thank Emirates Nature-WWF, UAE for their technical help and for providing the data of camera trap images. This paper represents the opinions of the authors and does not mean to represent the position or opinions of the American University of Sharjah.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wearn, O.R.; Glover-Kapfer, P. Snap Happy: Camera Traps Are an Effective Sampling Tool When Compared with Alternative Methods. *R. Soc. Open Sci.* **2019**, *6*, 181748. [CrossRef]

2. Glover-Kapfer, P.; Soto-Navarro, C.A.; Wearn, O.R. Camera-Trapping Version 3.0: Current Constraints and Future Priorities for Development. *Remote Sens. Ecol. Conserv.* **2019**, *5*, 209–223. [CrossRef]
3. LeCun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.* **1989**, *1*, 541–551. [CrossRef]
4. Okafor, E.; Pawara, P.; Karaaba, F.; Surinta, O.; Codreanu, V.; Schomaker, L.; Wiering, M. Comparative Study between Deep Learning and Bag of Visual Words for Wild-Animal Recognition. In Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece, 6–9 December 2016; pp. 1–8.
5. Zmudzinski, L. Deep Learning Guinea Pig Image Classification Using Nvidia DIGITS and GoogLeNet. In *CS & P, Proceedings of the 27th International Workshop on Concurrency, Specification and Programming, Berlin, Germany, 24–26 September 2018*; Humboldt-Universität zu Berlin: Berlin, Germany, 2018; Volume 2240.
6. Yousif, H.; Yuan, J.; Kays, R.; He, Z. Animal Scanner: Software for Classifying Humans, Animals, and Empty Frames in Camera Trap Images. *Ecol. Evol.* **2019**, *9*, 1578–1589. [CrossRef] [PubMed]
7. Huang, Y.P.; Basanta, H. Bird Image Retrieval and Recognition Using a Deep Learning Platform. *IEEE Access* **2019**, *7*, 66980–66989. [CrossRef]
8. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
9. Allken, V.; Handegard, N.O.; Rosen, S.; Schreyeck, T.; Mahiout, T.; Malde, K. Fish Species Identification Using a Convolutional Neural Network Trained on Synthetic Data. *ICES J. Mar. Sci.* **2019**, *76*, 342–349. [CrossRef]
10. Hu, M.; You, F. Research on Animal Image Classification Based on Transfer Learning. In Proceedings of the 4th International Conference on Electronic Information Technology and Computer Engineering, Xiamen, China, 6–8 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 756–761.
11. Lai, K.; Tu, X.; Yanushkevich, S. Dog Identification Using Soft Biometrics and Neural Networks. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–8.
12. Tabak, M.A.; Norouzzadeh, M.S.; Wolfson, D.W.; Sweeney, S.J.; Vercauteren, K.C.; Snow, N.P.; Halseth, J.M.; Salvo, P.A.D.; Lewis, J.S.; White, M.D.; et al. Machine Learning to Classify Animal Species in Camera Trap Images: Applications in Ecology. *Methods Ecol. Evol.* **2019**, *10*, 585–590. [CrossRef]
13. Whytock, R.; Świeżewski, J.; Zwerts, J.A.; Bara-Słupski, T.; Pambo, A.F.K.; Rogala, M.; Bahaa-el-din, L.; Boekee, K.; Brittain, S.; Cardoso, A.W.; et al. High Performance Machine Learning Models Can Fully Automate Labeling of Camera Trap Images for Ecological Analyses. *bioRxiv* **2020**. [CrossRef]
14. Schneider, S.; Greenberg, S.; Taylor, G.W.; Kremer, S.C. Three Critical Factors Affecting Automated Image Species Recognition Performance for Camera Traps. *Ecol. Evol.* **2020**, *10*, 3503–3517. [CrossRef]
15. Tabak, M.A.; Norouzzadeh, M.S.; Wolfson, D.W.; Newton, E.J.; Boughton, R.K.; Ivan, J.S.; Odell, E.A.; Newkirk, E.S.; Conrey, R.Y.; Stenglein, J.; et al. Improving the Accessibility and Transferability of Machine Learning Algorithms for Identification of Animals in Camera Trap Images: MLWIC2. *Ecol. Evol.* **2020**, *10*, 10374–10383. [CrossRef] [PubMed]
16. Tabak, M.; Norouzzadeh, M.S.; Wolfson, D.; Sweeney, S.; Vercauteren, K.; Snow, N.; Halseth, J.; Salvo, P.; Lewis, J.; White, M.; et al. *MLWIC: Machine Learning for Wildlife Image Classification in R v0.1*; CERN: Meyrin, Switzerland, 2018.
17. Shashidhara, B.M.; Mehta, D.; Kale, Y.; Morris, D.; Hazen, M. Sequence Information Channel Concatenation for Improving Camera Trap Image Burst Classification. *arXiv* **2020**, arXiv:2005.00116.
18. Norouzzadeh, M.S.; Morris, D.; Beery, S.; Joshi, N.; Jojic, N.; Clune, J. A Deep Active Learning System for Species Identification and Counting in Camera Trap Images. *Methods Ecol. Evol.* **2021**, *12*, 150–161. [CrossRef]
19. Vargas-Felipe, M.; Pellegrin, L.; Guevara-Carrizales, A.A.; López-Monroy, A.P.; Escalante, H.J.; Gonzalez-Fraga, J.A. Desert Bighorn Sheep (Ovis Canadensis) Recognition from Camera Traps Based on Learned Features. *Ecol. Inform.* **2021**, *64*, 101328. [CrossRef]
20. Willi, M.; Pitman, R.T.; Cardoso, A.W.; Locke, C.; Swanson, A.; Boyer, A.; Veldthuis, M.; Fortson, L. Identifying Animal Species in Camera Trap Images Using Deep Learning and Citizen Science. *Methods Ecol. Evol.* **2019**, *10*, 80–91. [CrossRef]
21. Chen, R.; Little, R.; Mihaylova, L.; Delahay, R.; Cox, R. Wildlife Surveillance Using Deep Learning Methods. *Ecol. Evol.* **2019**, *9*, 9453–9466. [CrossRef]
22. Sahu, R. Detecting and Counting Small Animal Species Using Drone Imagery by Applying Deep Learning. In *Visual Object Tracking with Deep Neural Networks*; Luigi Mazzeo, P., Ramakrishnan, S., Spagnolo, P., Eds.; IntechOpen: London, UK, 2019; ISBN 978-1-78985-157-1.
23. Nguyen, H.; Maclagan, S.J.; Nguyen, T.D.; Nguyen, T.; Flemons, P.; Andrews, K.; Ritchie, E.G.; Phung, D. Animal Recognition and Identification with Deep Convolutional Neural Networks for Automated Wildlife Monitoring. In Proceedings of the IEEE International Conference on Data Science and Advanced Analytics (DSAA), Tokyo, Japan, 19–21 October 2017; pp. 40–49.
24. Gomez, A.; Diez, G.; Salazar, A.; Diaz, A. Animal Identification in Low Quality Camera-Trap Images Using Very Deep Convolutional Neural Networks and Confidence Thresholds. In *Advances in Visual Computing, Proceedings of the International Symposium on Visual Computing, Las Vegas, NV, USA, 12–14 December 2016*; Bebis, G., Boyle, R., Parvin, B., Koracin, D., Porikli, F., Skaff, S., Entezari, A., Min, J., Iwai, D., Sadagic, A., et al., Eds.; Springer: Cham, Switzerland, 2016; pp. 747–756.
25. Beery, S.; van Horn, G.; Perona, P. Recognition in Terra Incognita. In *Computer Vision—ECCV 2018*; Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y., Eds.; Springer: Cham, Switzerland, 2018; Volume 11220, pp. 472–489. ISBN 978-3-030-01269-4.

26. Shepley, A.; Falzon, G.; Meek, P.; Kwan, P. Automated Location Invariant Animal Detection in Camera Trap Images Using Publicly Available Data Sources. *Ecol. Evol.* **2021**, *11*, 4494–4506. [CrossRef] [PubMed]
27. Wei, W.; Luo, G.; Ran, J.; Li, J. Zilong: A Tool to Identify Empty Images in Camera-Trap Data. *Ecol. Inform.* **2020**, *55*, 101021. [CrossRef]
28. Cunha, F.; dos Santos, E.M.; Barreto, R.; Colonna, J.G. Filtering Empty Camera Trap Images in Embedded Systems. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Nashville, TN, USA, 19–25 June 2021; pp. 2438–2446.
29. Kamdem Teto, J.; Xie, Y. Automatic Identification of Animals in the Wild: A Comparative Study between C-Capule Networks and Deep Convolutional Neural Networks. Master's Thesis, Kennesaw State University, Kennesaw, GA, USA, 27 November 2018.
30. Rathi, D.; Jain, S.; Indu, S. Underwater Fish Species Classification Using Convolutional Neural Network and Deep Learning. In Proceedings of the Ninth International Conference on Advances in Pattern Recognition (ICAPR), Bangalore, India, 27–30 December 2017; pp. 1–6.
31. Giraldo-Zuluaga, J.-H.; Salazar, A.; Gomez, A.; Diaz-Pulido, A. Automatic Recognition of Mammal Genera on Camera-Trap Images Using Multi-Layer Robust Principal Component Analysis and Mixture Neural Networks. *arXiv* **2017**, arXiv:1705.02727.
32. Tariq, N.; Saleem, K.; Mushtaq, M.; Nawaz, M.A. Snow Leopard Recognition Using Deep Convolution Neural Network. In Proceedings of the 2nd International Conference on Information System and Data Mining, Lakeland, FL, USA, 9–11 April 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 29–33.
33. Jiang, B.; Huang, W.; Tu, W.; Yang, C. An Animal Classification Based on Light Convolutional Network Neural Network. In Proceedings of the International Conference on Intelligent Computing and Its Emerging Applications (ICEA), Tainan, Taiwan, 30 August–1 September 2019; pp. 45–50.
34. Team, K. Keras Documentation: Keras Applications. Available online: https://keras.io/api/applications/ (accessed on 9 March 2021).
35. Tan, M.; Le, Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; Chaudhuri, K., Salakhutdinov, R., Eds.; PMLR: Cambridge, MA, USA, 2019; Volume 97, pp. 6105–6114.
36. Abuduweili, A.; Wu, X.; Tao, X. Efficient Method for Categorize Animals in the Wild. *arXiv* **2019**, arXiv:1907.13037.
37. Khalifa, N.E.M.; Taha, M.H.N.; Hassanien, A.E. Aquarium Family Fish Species Identification System Using Deep Neural Networks. In Proceedings of the International Conference on Advanced Intelligent Systems and Informatics, Cairo, Egypt, 1–3 September 2018; Hassanien, A.E., Tolba, M.F., Shaalan, K., Azar, A.T., Eds.; Springer: Cham, Switzerland, 2019; Volume 845, pp. 347–356, ISBN 978-3-319-99009-5.
38. Gomez Villa, A.; Salazar, A.; Vargas, F. Towards Automatic Wild Animal Monitoring: Identification of Animal Species in Camera-Trap Images Using Very Deep Convolutional Neural Networks. *Ecol. Inform.* **2017**, *41*, 24–32. [CrossRef]
39. Miao, Z.; Gaynor, K.M.; Wang, J.; Liu, Z.; Muellerklein, O.; Norouzzadeh, M.S.; McInturff, A.; Bowie, R.C.K.; Nathan, R.; Yu, S.X.; et al. Insights and Approaches Using Deep Learning to Classify Wildlife. *Sci. Rep.* **2019**, *9*, 8137. [CrossRef] [PubMed]
40. Yin, Z.; You, F. Animal Image Recognition Based on Convolutional Neural Network. In Proceedings of the 4th International Conference on Electronic Information Technology and Computer Engineering, Xiamen, China, 6–8 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 545–549.
41. Inception v1 1.2. Available online: https://gallery.azure.ai/Model/Inception-v1-1-2-3 (accessed on 9 March 2021).
42. Papers with Code—The Latest in Machine Learning. Available online: https://paperswithcode.com/paper/atrous-convolutional-neural-network-acnn-for/review/ (accessed on 9 March 2021).
43. Samuel Albanie/Convnet-Burden. Available online: https://github.com/albanie/convnet-burden (accessed on 9 March 2021).
44. Khazbak, Y.; Qiu, J.; Tan, T.; Cao, G. TargetFinder: A Privacy Preserving System for Locating Targets through IoT Cameras. *ACM Trans. Internet Things* **2020**, *1*, 14. [CrossRef]
45. Zualkernan, I.A.; Dhou, S.; Judas, J.; Sajun, A.R.; Gomez, B.R.; Hussain, L.A.; Sakhnini, D. Towards an IoT-Based Deep Learning Architecture for Camera Trap Image Classification. In Proceedings of the IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT), Dubai, United Arab Emirates, 12–15 December 2020; pp. 1–6.
46. Mathur, A.; Khattar, S. Real-Time Wildlife Detection on Embedded Systems. Available online: http://ilpubs.stanford.edu:8090/1165/ (accessed on 23 November 2019).
47. Thomassen, S. Embedded Analytics of Animal Images. Master's Thesis, UiT The Arctic University of Norway, Tromsø, Norway, 14 December 2017.
48. Popat, P.; Sheth, P.; Jain, S. Animal/Object Identification Using Deep Learning on Raspberry Pi. In *Information and Communication Technology for Intelligent Systems, Proceedings of the 2nd International Conference on Technology, Innovation, Society and Science-to-Business (ICTIS 2018), Padang, Indonesia, 25–26 July 2018*; Satapathy, S.C., Joshi, A., Eds.; Springer: Singapore, 2019; pp. 319–327.
49. Monburinon, N.; Zabir, S.M.S.; Vechprasit, N.; Utsumi, S.; Shiratori, N. A Novel Hierarchical Edge Computing Solution Based on Deep Learning for Distributed Image Recognition in IoT Systems. In Proceedings of the 4th International Conference on Information Technology (InCIT), Bangkok, Thailand, 24–25 October 2019; pp. 294–299.
50. Curtin, B.H.; Matthews, S.J. Deep Learning for Inexpensive Image Classification of Wildlife on the Raspberry Pi. In Proceedings of the IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON), New York, NY, USA, 10–12 October 2019; pp. 82–87.

51. Patil, H.; Ansari, N. *Smart Surveillance and Animal Care System Using IOT and Deep Learning*; Social Science Research Network: Rochester, NY, USA, 2020.
52. Tydén, A.; Olsson, S. Edge Machine Learning for Animal Detection, Classification, and Tracking. Master's Thesis, Linköping University, Linköping, Sweden, 11 June 2020.
53. Ramos-Arredondo, R.I.; Carvajal-Gámez, B.E.; Gendron, D.; Gallegos-Funes, F.J.; Mújica-Vargas, D.; Rosas-Fernández, J.B. PhotoId-Whale: Blue Whale Dorsal Fin Classification for Mobile Devices. *PLoS ONE* **2020**, *15*, e0237570. [CrossRef]
54. Arshad, B.; Barthelemy, J.; Pilton, E.; Perez, P. Where Is My Deer?–Wildlife Tracking And Counting via Edge Computing And Deep Learning. In Proceedings of the IEEE SENSORS, Online, 25–28 October 2020; pp. 1–4.
55. Islam, S.B.; Valles, D. Identification of Wild Species in Texas from Camera-Trap Images Using Deep Neural Network for Conservation Monitoring. In Proceedings of the 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 6–8 January 2020; pp. 0537–0542.
56. Rohilla, R.; Banga, P.S.; Garg, P.; Mittal, P. GPU Based Re-Trainable Pruned CNN Design for Camera Trapping at the Edge. In Proceedings of the International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2–4 July 2020; pp. 558–563.
57. Elias, A.R.; Golubovic, N.; Krintz, C.; Wolski, R. Where's the Bear?—Automating Wildlife Image Processing Using IoT and Edge Cloud Systems. In Proceedings of the IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI), Pittsburgh, PA, USA, 18–21 April 2017; pp. 247–258.
58. Foundation, T.R.P. Raspberry Pi 4 Model B Specifications. Available online: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ (accessed on 7 June 2021).
59. Getting Started with Jetson Nano Developer Kit. Available online: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit (accessed on 30 April 2020).
60. Dev Board. Available online: https://coral.ai/products/dev-board/ (accessed on 8 June 2021).
61. Prado, M.D.; Su, J.; Saeed, R.; Keller, L.; Vallez, N.; Anderson, A.; Gregg, D.; Benini, L.; Llewellynn, T.; Ouerhani, N.; et al. Bonseyes AI Pipeline—Bringing AI to You: End-to-End Integration of Data, Algorithms, and Deployment Tools. *ACM Trans. Internet Things* **2020**, *1*, 26. [CrossRef]
62. Matuska, S.; Hudec, R.; Benco, M.; Kamencay, P.; Zachariasova, M. A Novel System for Automatic Detection and Classification of Animal. In Proceedings of the ELEKTRO, Rajecké Teplice, Slovakia, 19–20 May 2014; pp. 76–80.
63. Liu, X.; Jia, Z.; Hou, X.; Fu, M.; Ma, L.; Sun, Q. Real-Time Marine Animal Images Classification by Embedded System Based on Mobilenet and Transfer Learning. In Proceedings of the OCEANS 2019—Marseille, Marseille, France, 17–20 June 2019; IEEE: Marseille, France, 2019; pp. 1–5.
64. Forestiero, A. Metaheuristic Algorithm for Anomaly Detection in Internet of Things Leveraging on a Neural-Driven Multiagent System. *Knowl.-Based Syst.* **2021**, *228*, 107241. [CrossRef]
65. Haxhibeqiri, J.; de Poorter, E.; Moerman, I.; Hoebeke, J. A Survey of LoRaWAN for IoT: From Technology to Application. *Sensors* **2018**, *18*, 3995. [CrossRef]
66. Adelantado, F.; Vilajosana, X.; Tuset-Peiro, P.; Martinez, B.; Melia-Segui, J.; Watteyne, T. Understanding the Limits of LoRaWAN. *IEEE Commun. Mag.* **2017**, *55*, 34–40. [CrossRef]
67. Ojo, M.O.; Adami, D.; Giordano, S. Experimental Evaluation of a LoRa Wildlife Monitoring Network in a Forest Vegetation Area. *Future Internet* **2021**, *13*, 115. [CrossRef]
68. Sklearn.Model_selection.GridSearchCV. Available online: https://scikit-learn/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (accessed on 29 November 2021).
69. Liashchynskyi, P.; Liashchynskyi, P. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. *arXiv* **2019**, arXiv:1912.06059.
70. TensorFlow Lite. ML for Mobile and Edge Devices. Available online: https://www.tensorflow.org/lite (accessed on 23 November 2021).
71. NVIDIA TensorRT. Available online: https://developer.nvidia.com/tensorrt (accessed on 30 April 2020).
72. Van Engelen, J.E.; Hoos, H.H. A Survey on Semi-Supervised Learning. *Mach. Learn.* **2020**, *109*, 373–440. [CrossRef]
73. Sohn, K.; Berthelot, D.; Li, C.-L.; Zhang, Z.; Carlini, N.; Cubuk, E.D.; Kurakin, A.; Zhang, H.; Raffel, C. FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence. *arXiv* **2020**, arXiv:2001.07685.