Where's The Bear?- Automating Wildlife Image Processing Using IoT and Edge Cloud Systems

Andy Rosales Elias, Nevena Golubovic, Chandra Krintz, and Rich Wolski
Computer Science Dept
Univ. of California
Santa Barbara, California

ABSTRACT

We investigate the design and implementation of *W*here's *T*he *B*ear (WTB), an end-to-end, distributed, IoT system for wildlife monitoring. WTB implements a multi-tier (cloud, edge, sensing) system that integrates recent advances in machine learning based image processing to automatically classify animals in images from remote, motion-triggered camera traps. We use non-local, resource-rich, public/private cloud systems to train the machine learning models, and "in-the-field," resource-constrained edge systems to perform classification near the IoT sensing devices (cameras).

We deploy WTB at the UCSB Sedgwick Reserve, a 6000 acre site for environmental research and use it to aggregate, manage, and analyze over 1.12M images. WTB integrates Google TensorFlow and OpenCV applications to perform automatic image classification and tagging. To avoid transferring large numbers of training images for TensorFlow over the low-bandwidth network linking Sedgwick to public clouds, we devise a technique that uses stock Google Images to construct a synthetic training set using only a small number of empty, background images from Sedgwick. Our system is able to accurately identify bears, deer, coyotes, and emtpy images and significantly reduces the time and bandwidth requirements for image transfer, as well as end-user analysis time, since WTB automatically filters the images on-site.

CCS CONCEPTS

•Computer systems organization →Embedded and cyber-physical systems;

KEYWORDS

Cloud Computing; Internet-of-Things; Image Processing; Edge Computing; Animal Surveillance

ACM Reference format:
Andy Rosales Elias, Nevena Golubovic, Chandra Krintz, and Rich Wolski. 2017. Where's The Bear?- Automating Wildlife Image Processing Using IoT and Edge Cloud Systems. In *Proceedings of The 2nd ACM/IEEE International Conference on Internet-of-Things Design and Implementation, Pittsburgh, PA USA, April 2017 (IoTDI 2017),* 12 pages.
DOI: 10.1145/3054977.3054986

IoTDI 2017, Pittsburgh, PA USA

DOI: 10.1145/3054977.3054986

1 INTRODUCTION

Wildlife monitoring plays a key role in a wide range of scientific activities and societal interests. Understanding animal behavior and activity patterns [3] is useful for evaluating biodiversity and changes in habitats and land use, avoiding dangerous human-wildlife encounters [32] and destructive habitat overlap, monitoring species health and population dynamics, and providing people with high impact educational experiences. Digital photography provides an effective, non-intrusive way to monitor wildlife for many of these applications. It is safe, cost effective, and accessible to people with a wide range of expertise and backgrounds. To scale the process of wildlife monitoring in remote locations researchers are increasingly turning to automatically activated, battery or solar powered cameras equipped with motion detection sensors and network connectivity (e.g. radios or WIFI), (i.e. camera traps or trail cameras) [7, 38].

Despite the success of these technological approaches there are still several challenges that scientists and citizen scientists face in classifying images and identifying animals in images. First, automatically activated cameras can generate an enormous number of images (especially when motion-triggered), making it time consuming if not infeasible to perform classification and identification manually. Second, the remote locations of the cameras (in the wild) can make it costly (in terms of time and/or monetary cost) to upload images to the Internet where they are increasingly stored, processed, and shared. And finally, motion sensors commonly trigger events due to non-animal activity (e.g. the wind or rain), for animals of no interest to a particular study, or redundant pictures of the same animal in slightly different poses, introducing significant overhead (to copy, store, communicate, and analyze unimportant images) in the classification process.

In this paper, we address these challenges with the design and implementation of a new wildlife monitoring system that leverages recent advances in the Internet-of-Things (IoT) and in open source image processing and deep learning [37] for image recognition, to automate image classification and analysis. Our system, called **W**here's **T**he **B**ear (WTB), is an end-to-end, distributed data acquisition and analytics system that implements an IoT architecture consisting of of sensors (cameras), "edge clouds," and a back-end public or private cloud. We describe a specialization of this architecture for wildlife monitoring and a novel integration of image processing software for scalable image classification (animal identification). WTB is extensible in that different algorithms for animal classification and image processing can be easily integrated (i.e. "plugged" into the system) via application programming interfaces (APIs) and wrapper scripts. The key innovation is that WTB performs classification *near* where the images are produced to avoid

unnecessary image transfer (of unimportant, redundant, or empty images) over expensive, long-haul, and/or low bandwidth network links.

We implement WTB using open source software and "off-the-shelf" inexpensive equipment to ensure its accessibility to a broad audience of users. For the experiments we describe, we integrate Google TensorFlow [1] for image recognition and OpenCV [6] for image analysis within WTB. We also present a novel training technique for our TensorFlow animal image classifier that leverages the WTB architecture by using freely available, labeled images from cropped Google Images [15] instead of a training set from the camera traps. By doing so, this technique avoids transferring a large training set to a private or public cloud where there is sufficient computational capacity to train a TensorFlow model. Instead our approach combines a small number of empty images from our camera traps at different times of the day with images of animals of interest taken from Google Images. That is, we construct training images by creating a montage of an empty background from the camera traps with Google Image examples of different animals. We generate thousands of these "fake" images automatically overlaying the animal images the camera trap background (empty image) in different orientations and illumination levels. This process produces a very large training dataset with which we train a TensorFlow model – without ever requiring an image containing an animal from the camera traps be part of the training set.

We implement our system at the UCSB Sedgwick Reserve, an ecology and wildlife educational and research reserve [35]. The reserve is 6,000 acres that comprises critical wildlife habitats, two watersheds at the foot of Figueroa Mountain, and a 300 acre farm easement. Sedgwick has 11 camera trap locations and high bandwidth, wireless networking throughout much of the property. Its meeting house is connected to the UCSB campus via a lower-bandwidth microwave radio link. WTB at Sedgwick currently integrates nine of cameras which have collected over 1 million images (approximately 550GB of data).

We evaluate the performance of WTB and its accuracy in classifying bears, deer, and coyotes for one of the Sedgwick camera traps with over 600K images. Our results indicate that by training a model in a public or private cloud and then using it to classify images "at the edge" near where the images are gathered drastically reduces the time and expense associated with the classification process. Further, the automated image classification method enables high accuracy with few false positives and false negatives, making it possible to replace what had become a tedious, error prone, and ultimately infeasible manual process. Finally, we investigate the extensibility of WTB by using it to integrate image classification with optical character recognition (OCR) to label each image with additional metadata. For example, we find that we are able to extract temperature values recorded by each camera from the images with 100% accuracy.

In summary, with this paper, we contribute

- An integration of multiple disparate technologies (sensor data aggregation and management with automatic image processing using advanced machine learning techniques as "black boxes") into an end-to-end system for wildlife monitoring,
- A distributed IoT architecture that is customized for this application domain, which leverages edge cloud systems to implement image and sensor data processing and storage near where the data is generated, and
- An empirical evaluation of the benefits that this architecture enables.

In the sections that follow, we overview the challenges of wildlife monitoring faced by Sedgwick scientists and present the design and implementation of WTB. We then evaluate WTB for a subset of the images collected by Sedgwick camera traps. We measure accuracy of the machine learning technologies that we integrate into WTB and our use of "fake" images for training. We also report the amount of time and bandwidth that WTB saves if we are able to filter the images at the source. We then present related work and conclude.

2 WTB: END-TO-END IOT-CLOUD SYSTEM FOR WILDLIFE TRACKING

The goal of our work is to simplify and expedite the process of animal identification using images from motion-triggered cameras deployed in the wild. Our experimental setting is the UCSB Sedgwick Reserve located 40 miles from the UCSB campus in Santa Ynez, CA. Sedgwick is a research and educational reserve for study of environmental stewardship and protection, restoration of natural biological systems, and technology-driven agriculture [24]. Scientists from fields including ecology, biology, computer science, geography, and others use parts of the over nine square miles of the Sedgwick property to perform measurement, experimentation, demonstration, and hands-on and multi-disciplinary pedagogy. As a research reserve, there are a small number of structures with electricity and Internet connectivity, but they are clustered together in one location on the property – the remainder of the reserve is wild.

Many Sedgwick scientists and management personnel are interested in monitoring the wildlife at Sedgwick for various purposes including to estimate population size and health for different locally occurring species, to identify changes in animal behavior patterns due to external forces (drought, human activity, invasive species), to identify/prosecute illegal hunting activities, and to track and recover stray grazing livestock. To facilitate these activities, Sedgwick manages 11 camera traps at watering holes and popular animal pathways throughout the property. Some cameras have support for wireless communications, while others require manual download of the images using storage cards. The Sedgwick staff plans to convert all to WIFI for download over time.

Figure 1 shows the map of the property with the camera traps and headquarters building identified (in green). The headquarters building is connected to the UCSB Campus network via a long-distance microwave radio link and wireless connectivity is available directionally between the headquarters and each camera trap. The average bandwidth between the cameras and headquarters is approximately 114 Megabits per second (Mbps). The connectivity between headquarters and the UCSB campus (which must traverse several microwave links) rarely exceeds 5 Mbps even for optimized file transfer.

We use one camera trap (called *Main*) in this study to evaluate our system. We mark this camera in red on the map. The distance

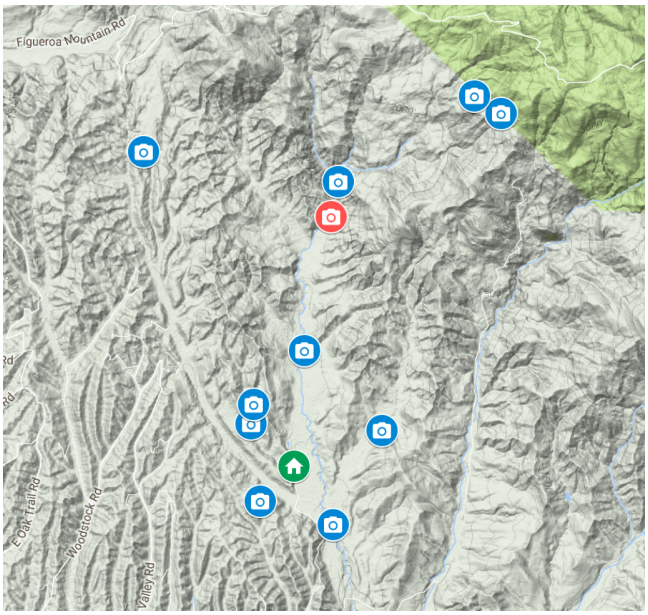

Figure 1: UCSB Sedgwick Reserve with Camera Traps and Headquarter Building Icons. The reserve is approximately 9 square miles in size and the distance between the camera (red camera icon) and headquarters (dark green house icon) is 3.05km. There are 11 trail cameras on the property in total. The northeast corner boarders Los Padres National Forest (light green).

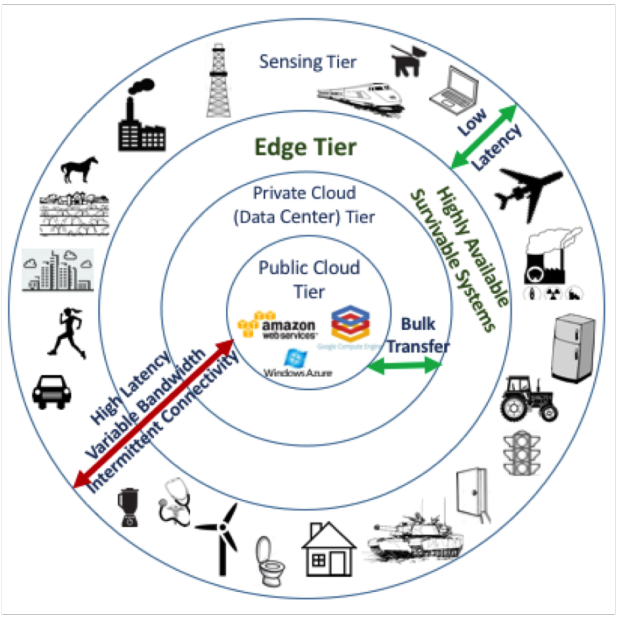

Figure 2: IoT-Edge Architecture for WTB. Adding an Edge Tier capable of low-latency, high bandwidth, high availability, low cost communications and fast response to/from the Sensing Tier. We define a new computing entity for the Edge Tier called the edge cloud that mirrors the functionality of popular public cloud systems at a smaller scale, to provide advanced data fusion and complex analytics via a Cloud Service Distribution Network to the Sensing Tier. The Private/-Data Center Tier provides similar functionality as Public Clouds for data and compute (software) but with privacy, security, and cost control. The high latency, variable, intermittent, and costly connectivity to the Public Cloud Tier is leveraged for long term data backup and latency-tolerant batch processing only if/when needed (on-demand).

between the camera and headquarters is 3.05km. We store camera trap images on a computer system in the headquarters building for on-site analysis; researchers copy the images to campus or via Internet to other organizations for sharing with others and for analysis and processing. The cameras collect images continuously and to date they have been in use intermittently over the past 3 years to collect over 1.12M images (716.3 GB).

This technology configuration and workflow presents multiple challenges for scientists and researchers attempting to analyze these images. First, they must copy the images from Sedgwick to UCSB or another Internet site where the necessary software and sufficient computational resources are available. Given Sedgwick's remote location, connectivity between it and the main campus or the Internet is intermittently available and imposes high overhead on such copies.

For example, copying the 638,062 images from the Main camera alone, between Sedgwick headquarters and campus, took researchers 13 days. Given that public cloud services, including Amazon Web Services (AWS) Simple Storage Service (S3) and Box.com (which we have tested as part of this project), limit the transfer speed and access frequency to and from their sites for stability and fair sharing purposes, transfer of the Main camera images to these services from Sedgwick headquarters took 14.92 days and 13.09 days, respectively. Such delays are costly in terms of time, result in many copies of the images (wasted storage), and preclude the use of real time processing and actuation (e.g. thwarting poachers, detecting escaped livestock) based on the images at Sedgwick.

Second, transmitting all images wastes bandwidth (and power), since many contain no animals (motion detectors are triggered by wind and weather events), or they contain repeated images of the same animal (time lapsed) in slightly different poses, or they contain animals of no interest to a particular study. Moreover, classifying images by hand is time consuming (expensive), tedious, and ultimately infeasible. For example, we found that 10 dedicated students can process (label) 2500 of our images in approximately 2 hours. Some techniques and online services are available for automatically classifying images but are only available in "the cloud" and as such, require upload over slow, long-haul networks. The final issue is that although many new automated techniques for image processing and classification have emerged and are freely available, they require extensive technical expertise and significant experience with machine learning to be used effectively.

To address these challenges, we pursue an IoT approach to camera trap image classification that, using edge clouds, "brings the cloud to the data" to avoid unnecessary data transfer from the IoT sensors (cameras) to the point of analysis, and that simplifies the

classification process (no prior knowledge of cloud computing or machine learning is needed). To enable this, we build upon and extend existing approaches to IoT that attempt to lower latency via in-network processing of data (aggregation, filtering, caching) near where the IoT devices generate data and where IoT applications consume it – at the edge of the network as depicted in Figure 2. The use of an edge tier as shown in the figure is referred to variously as edge networking, fog computing, edge computing, or cloudlets in the literature [5, 8, 10, 14, 36]. literature. In addition, we encapsulate image processing techniques into this system so that they can be easily employed as "black boxes", i.e. via automatic configuration and deployment, enabling their use for customized image classification by non-experts.

Our approach, called WTB, is a distributed system that implements the multi-tier IoT architecture depicted in the figure. We employ our campus cloud computing infrastructure (UCSB Aristotle [4]) for the Private Cloud Tier. We use AWS and Box.com for the Public Cloud Tier. The sensing tier includes the Sedgwick camera traps and other sensors (weather stations) located on the property. Unique to our approach is the ability migrate cloud-based applications to an Edge Tier where they are hosted by one or more "edge clouds.". Instead of performing only caching and simple filtering near the Sensing Tier, we propose, develop, and deploy edge clouds that are capable of implementing advanced computing and analytics as an *appliance* – with little or no expert system administration or programming skills required for operation.

Our edge cloud appliance is a scaled-down, open source, highly available, version of the AWS public cloud infrastructure implemented using Eucalyptus [26]. That is, we configure a small cluster of computers (currently six 4-core computational "bricks" [20]) with this open source distributed system that mirrors Amazon's public Elastic Compute Cloud (EC2) and Simple Storage Service (S3). The use of Eucalyptus enables our *edge cloud* to run any software or services that run on EC2 and/or use S3, without modification, so that we are able to leverage the vast repositories of open source data analysis, machine learning toolkits, and web services available today with no porting effort.

Moreover, the high availability configuration of Eucalyptus enables us to construct a self-managing, resource constrained system without an IT staff to manage it (an appliance). If/when components fail, the edge cloud automatically reconfigures itself to operate using only the remaining nodes and disks until insufficient resources cause total system failure. Users of edge cloud services access the software and data (camera trap images in this case) via their browsers as they would any other Internet service or website. The edge cloud is located in the Sedgwick headquarters building and IoT devices at Sedgwick connect directly to it using the Sedgwick private wireless network.

In this paper, we customize this multi-tier architecture for remote camera trap image processing applications (the WTB system). WTB implements a software stack that automates image processing via advanced machine learning technologies that classify images of animals. Thus WTB is able to extract images with animals (or other characteristics) of interest and transmit them to end users and cloud services for further processing or sharing, significantly reducing bandwidth use, transfer delay, and end user storage requirements. Doing so also reduces the latency associated with classification

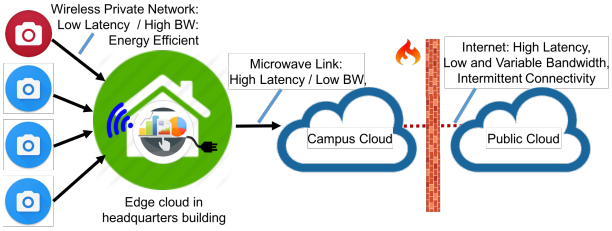

Figure 3: The WTB System.

(since it is performed on-site), which enables WTB to use results from its image processing for realtime control and actuation of other IoT sensors and devices at Sedgwick (to control sensor measurement frequency, remote release of water to the watering holes, sending of alerts, etc.).

2.1 WTB Implementation

Our WTB deployment for Sedwick Reserve is depicted in Figure 3. Cameras connect to the Sedgwick private network via radio links where available. The cameras consist of different makes and models. The one we target is this work is a Reconyx HC500 Hyperfire. We currently have 638,062 images (205.5GB) from this target camera trap and of these, we consider only day images (due to time constraints), which we consider to be between 9AM and 4PM year round. The images were taken for 897 days between July 13, 2013 and Aug 10, 2016 [a]. Of the 638,062 images, there are 260,159 day images with an average of 290 day-images per day for this camera. We show an example image from this camera trap in Figure 4.

Our WTB edge cloud implements Eucalyptus v4.1 and is configured with high availability (HA). The edge cloud comprises 6 6th-generation Intel "Next Unit of Computation" (NUCs [20]). Each NUC has a 4-core i7 processor and 45W thermal design power (TDP). The edge cloud also includes an Uninterrupted Power Supply (UPS), a gigabit Ethernet switch, a total of 1 TB of storage (with 2× redundancy configured), and a wireless access point (for communication with IoT devices on the property). The switch is connected to the main Sedgwick network linking the reserve to the main UCSB campus networking infrastructure via a series of microwave radio links that connects Sedgwick to UCSB via Santa Cruz Island.

2.2 WTB Image Processing

WTB processes and filters the images automatically so that only those of interest to the end user (scientist) require transfer and remote storage. To enable this, we define a workflow engine that runs over Eucalyptus (as a Linux virtual machine instance), that is "pluggable". That is, we store the images in a disk volume that we make available for network attachment to virtual machine instances (that process the images). Within an instance, any machine learning or image processing toolkit that can process JPEG encoded images can be used to process and filter images.

[a]During this period, the camera was down for 11 months (scattered across this time period) for repairs and use at different locations and for other purposes.

Figure 4: Example image from the Sedgwick watering hole understudy which is surveiled via a Reconyx HC500 Hyperfire.

To enable this, we provide a toolset written in the Python programming language and deployed in the edge cloud that automatically deploys virtual machine instances and disk volumes, and that invokes the image processing application. The disk volume stores the images using a POSIX-based hierarchical directory structure based on the date of the image, e.g. /imageSetID/YYYY/MM/DD, at the root of the mounted file system in the instance, for easy, uniform access by applications. The toolset interface requires that image processing applications read images from the directories and return a list of image filenames that they deem "of interest". The toolset can make the resulting images of interest available via a web service for browser-based viewing. Or it can perform a remote copy of them to cloud storage on the UCSB campus or to an Internet service. We currently support Internet services AWS S3, Box, and Dropbox.

As part of this experiment, we have investigated the integration of two applications into WTB: Google TensorFlow [1] and OpenCV Optical Character Recognition (OCR) [27] with JPEG processing [13, 28]. We "plug" these applications into the WTB system via the WTB wrapper and use them to process and filter Sedgwick images on-site using the WTB edge cloud.

Google TensorFlow is an open source library for machine learning and deep neural networks [1]. It defines an Application Programming Interface (API) that facilitates its use for a wide variety of machine intelligence applications. Many tools for working with TensorFlow on specific machine learning problems have been developed and released as open source by Google researchers and the community. In this work, we leverage the tools for defining, training, and evaluating models, for competitive networks in the field of image classification [2, 12, 16, 21, 30, 31]. In particular, we use Inception-V3 [31] a computer vision model for recognizing objects in images. The model is trained using images from the 2012 ImageNet [11] Large Visual Recognition Challenge. The model is one of the most successfully developed to date, recognizing objects ImageNet images with an error rate of around 4%.

Researchers such as ourselves, interested in using TensorFlow and the Inception-v3 model for image classification, do so by retraining the model for a particular class of images. For example, one technically adept Japanese farmer used TensorFlow successfully for classifying cucumber quality using images of cucumbers (to automate the manual process for doing so that the farm had used to date). Retraining the model required over 7000 images of cucumbers and 2—3 days using very powerful, GPU-based computing systems [29].

Given the processing requirements for training the model and the limited processing capability and storage capacity of our edge cloud, we separate the training process from classification. Classification requires significantly less processing power versus training and the representation of the resulting trained model is small. Thus our approach is to train the model on the campus or public cloud and then transmit the model to the edge cloud where the images are being produced (at the edge) to perform image classification locally. Note that it is necessary to be able to run the same version of TensorFlow during the training phase and the classification phase if the process is to be automated. By ensuring that the edge cloud is capable of running the same software as the private or public AWS cloud, we ensure that the model produced by TensorFlow during the training phase will work correctly at the edge for classification, and that we can automate the process end-to-end.

To keep training time and cost to a minimum in the public/-campus cloud, WTB uses transfer learning [34]. Transfer learning is a technique that shortcuts the training process by leveraging a fully-trained model for a set of categories such as those in ImageNet (Inception-v3), and retrains using existing weights for new classes. The problem with such an optimized approach is that retraining of the model requires access to images similar to the those we wish to classify (Sedgwick camera trap images). A naive approach is to transmit a large subset of images to the public/campus cloud to perform the retraining, however doing so defeats the purpose of processing the images and only communicating those of interest over the long haul network between Sedgwick and UCSB and the public cloud.

To address this challenge, we have developed a new approach to training the model for Sedgwick cameras and animals, that precludes the need for transmitting a large number of Sedgwick images to the cloud. To enable this, we generate a large number of "fake" Sedgwick images. We depict our workflow in Figure 5. We manually identify and transmit a small number of "empty" images from the camera to the cloud. Empty images are those that are triggered by motion but that do not contain animals in the image. We need multiple empty images because of the feature changes (light, size of water hole) throughout the 24 hour time period and at different times of the year. We expedite the process of finding empty images by using weather station data for Sedgwick (another IoT sensor in WTB) and extracting windy days. We then use the date and time of high wind events to target images that are likely to be empty. We identified and transmitted 250 empty day images totaling 0.1GB for this research.

On the cloud end (campus or public), WTB implements a software system that accesses Google Images over the Internet to obtain labeled images of each of the animals of interest (or classes). In this study, we use bears, deer, and coyote, the three most popular animals under study at Sedgwick. We query for images of these animals with white or transparent backgrounds (e.g. "bear on white background" or "bear on transparent background". These queries result in images of bears of different species and, in particular,

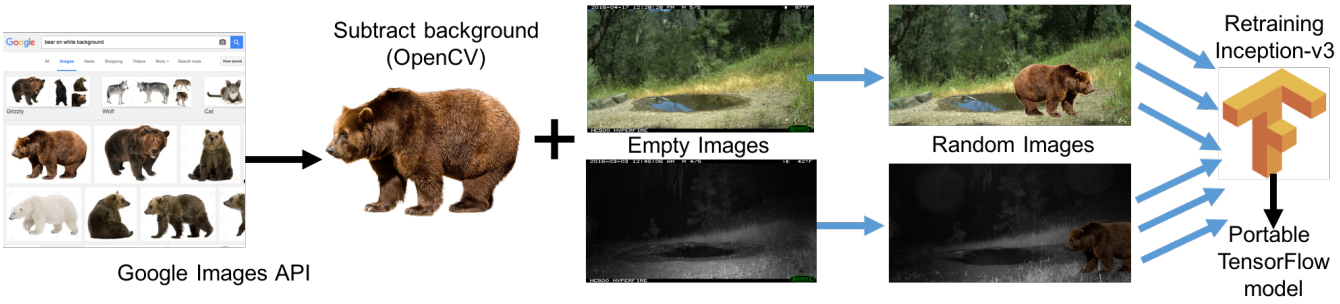

Figure 5: WTB TensorFlow Training Workflow for Generating Model Used for Sedgwick Image Classification.

different from those commonly occurring at Sedgwick. Thus it is possible that the trained model (if effective) could recognize the first occurrence of a species that had not been previously photographed at Sedgwick as well as to recognize those more familiar in the area. We then automatically subtract the background of the image from the animal by replacing all white pixels with transparent pixels. We perform this step by using OpenCV, an open source library of computer vision and image processing tools and algorithms.

We next overlay the animal objects on the empty background images from Sedgwick. We place the objects in the lower half of the image randomly across three positions vertically and ten positions horizontally. We also randomly flip the images as part of this placement. We next use the time of day and brightness on the empty images to adjust the color contrast of the objects to match the background. In total, we have produced 5000 fake images for each animal. We use the 15000 fake images and 250 empty images to retrain the TensorFlow Inception-v3 model in the cloud. We then evaluate how well using this type of training performs in identifying animals at Sedgwick (cf Section 3).

The second analytics technology we plug into the WTB edge cloud, is OpenCV Optical Character Recognition (OCR) and JPEG processing. We do so to investigate the generality of our application-integration approach. We use OCR to extract the air temperature that is embedded within the image itself. Our application is a Python program that processes each image in the directory structure (volume storage mounted as a Linux file system). The application crops each image (using fixed coordinates that we identified manually) and passes it to the OCR library.

We train the OCR library for each camera manually. To enable this training, we use temperature values (cropped images) between 0 and 9 (and the minus sign), which we extract from images after performing edge detection (a built in function in the OpenCV toolkit). We generate a training set of labeled digits by manually entering the value of each of the digits. We store the trained model on the edge cloud and use it with the OCR library which uses it to recognize any unlabeled character value in a previously unseen, cropped image, by selecting the object in the labeled set that is most similar (using the K-nearest neighbor algorithm [23]) to the digit being analyzed. The OCR tool combines the digits into a temperature value and reports the value back to the caller. We have implemented the OCR tool as a library call and as a cloud service in the WTB edge cloud so that we can compare the performance of the two implementation alternatives.

Table 1: Sedgwick Camera Trap Statistics. We use camera Main for the empirical evaluation of WTB in this paper.

Camera ID	Count	Tot. Size (GB)	Avg. Img Size (KB)	Day Images		
				Days	Count	Size (GB)
Blue	72900	43.39	624	324	54486	33.59
BoneH	153215	334.88	2291	350	153215	136.01
BoneT	5886	5.49	977	32	2277	2.12
Fig	24449	10.05	431	239	9919	5.61
Lisque	88183	24.41	290	219	14031	4.93
Main	**638062**	**204.26**	**335**	**897**	**238489**	**93.81**
NE	86645	39.77	481	324	43897	22.85
URes	5311	8.31	1641	24	4765	7.50
Vulture	17028	14.76	908	123	12468	12.01
WMill1	27332	27.87	1069	385	8396	9.82
WMillG	2310	0.93	421	32	1107	0.53
Total	1121321	714.12	9468	2949	543050	328.78

The Sedgwick camera that we employ for this study stores the temperature that it embeds in each image as part of the JPEG metadata of the image. Not all cameras do so and, indeed, none of the other Sedgwick cameras have this capability. For the camera we study, we perform and validate the OCR analysis. To validate OCR analysis on the temperature string in the image, we compare the output of the OCR analysis to the value recorded in the JPEG image metadata. We compute accuracy as the percentage of time the OCR analysis and the JPEG metadata agree.

3 EVALUATION

We use the WTB deployment at the UCSB Sedgwick Reserve to evaluate empirically different components that comprise its design. In this section, we first overview the details of the deployment. We then evaluate the accuracy and performance of the machine learning technologies that WTB integrates (TensorFlow and OpenCV OCR). We use this evaluation to investigate the costs associated with image transfer using different cloud technologies and to measure the savings of performing image classification and filtering in the edge tier.

We present the details of the Sedgwick camera traps in Table 1. The table shows for each camera name (column 1), the total number and size (in GB) of the images each produced between July 13, 2013 and Aug 10, 2016. Column 4 is the average size of each image in kilobytes (KB). The final three columns show statistics for when

Figure 6: Examples of Images Considered *Other*. Other images contain small birds and rodents. These images contain are not empty but animals not of interest.

we consider only images taken during daylight hours (between 9AM and 4PM inclusively). Column 5 is the total number of days for which we have images, columns 6 and 7 show the number and size (in GB) of day images. Each camera uses motion detection to trigger taking a photograph. Most triggers are caused by animals (birds, bears, deer, coyotes, squirrels, mountain lions, bobcats, etc.). However, the cameras are also triggered by movement of vegetation (caused by the wind). We refer to these images as empty.

We use the WTB on-reserve edge cloud to store and automatically analyze the images from the Main camera and consider bears, deer, and coyotes animals of interest for our analysis and machine learning study. The edge cloud also enables access to the images by researchers over the Internet and campus-to-reserve network. As of August 10, we have 1.12M total images totaling over 714GB. We have 630062 images totaling 204GB for the Main camera. The bandwidth between the Main camera and the edge cloud is 114Mbps on average. The bandwidth between the edge cloud and campus is 1.93Mbps for single image transfer and 4.63Mbps for optimized file transfer on average.

We trained a TensorFlow model using four categories of synthetically generated images: Bear, Deer, Coyote, and Empty. The fifth category – Other – contained a sufficient diversity of wildlife to make training for it difficult. We show two examples of images from Other in Figure 6. Thus, our approach attempts to classify bear, deer, coyote, and empty images from the image set but does not attempt to completely identify all of the images.

Table 2: Manually Classified Images from random sample of size 4890

Image Type	Count
Bear	227
Deer	1795
Coyote	22
Empty	1028
Other	1818

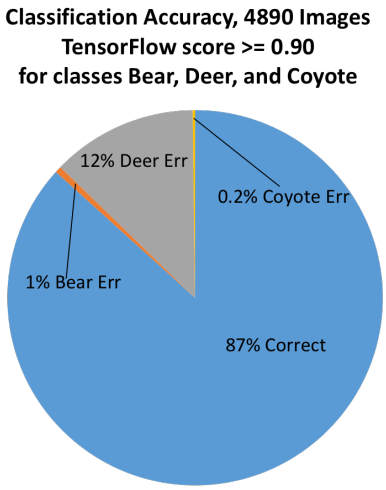

Figure 7: Common Animal Classification Accuracy Using a TensorFlow Score ≥ 0.90 for Bear, Deer, and Coyote, for 4890 Randomly Selected Images.

3.1 Classification Performance

In our first experiment, we investigate the effectiveness of our training and classification approach using 4890 randomly selected images from the full corpus of Main day images. We classified these images manually into the five categories: *Bear*, *Deer*, *Coyote*, *Empty*, and *Other*. Table 2 shows the number of each type of image in this sample.

We use the WTB model to classify these images automatically. We use the class with the highest score (highest probability) output by TensorFlow, for which the score is at least 0.90 for Bear, Deer, and Coyote. Figure 7 shows the accuracy of the resulting model with respect to identifying bears, deer, and coyote, when applied to the random sample of 4890 images.

The error percentages in the figure comprise both false positive errors and false negative errors with respect to the manually labeled images. False positive errors are those that TensorFlow says contains one type of animal, but the image does not contain that animal. False negative errors are those that are identified during manual labeling as containing a particular type of animal, but TensorFlow fails to categorize the image as containing that animal with a score of 0.90 or higher.

Overall, from the perspective of identifying bears, deer, and coyote, the results are promising. The overall "miss rate" is 13%,

Figure 8: Examples of of False Positive Classifications. TensorFlow classified these images as coyote (top) and bear (bottom) when both contain deer.

Figure 9: Examples of of False Negative Classifications. TensorFlow classified these images with a low score (top) and as Empty (bottom) when the top contains a bear (or part of one) and the bottom contains a deer running off in the distance.

Table 3: Correct, False Positive, and False Negative Image Counts Using a TensorFlow Score of ≥ 0.90 for Bear, Deer, and Coyote

Image Type	Correct	False Pos.	False Neg.
Bear	200	16	27
Deer	1187	22	608
Coyote	12	36	10

almost all of which is accounted for by misclassification of deer images. These error rates (which include false positives and false negatives) are quite low for bear and coyote images, and while the error rate for the Deer category (12%) may seem large, it is consistent with the 90% probability threshold which the TensorFlow score ≥ 0.90 represents. Thus, we believe that the methodology is working correctly. We show examples of false positives in Figure 8 and examples of false negatives in Figure 9.

To determine the effect of these error rates on the bandwidth usage, we detail the correct count, false positive counts, and false negative counts for each category in Table 3. This data indicates that the bandwidth savings resulting from the use of an edge cloud are substantial. If the system were attempting only to transfer images of bear, deer and coyote, only the false positive counts for the each would "waste" bandwidth. Put another way, WTB would have transferred a total of 1473 images (the sum of columns 2 and 3 in the table). Of these images, 1399 are correctly classified in terms of their animal type and 74 are misclassifications of each animal type. If the 74 images were strictly images of Empty or Other, they would be completely unnecessary transfers. However many of them are misclassifications within the categories of Bear, Dear, and Coyote. Transferring these images is necessary but they would then need to be manually reclassified. The worst-case overhead is 74 images out of 1473 total transferred or approximately 5% and, if manual inspection of the false positives were implemented, the true overhead is substantially less.

From Table 3, it is evident that the bulk of the Deer category errors are false negatives. That is, the system appears to be successful at identifying bear, coyote, and deer images, but it also fails to classify correctly approximately 12% of the images as deer. However it does not incur a similar miss rate for bear or coyote images. Thus the system is successful at answering a query of the form "What type of animal is in this image?" but is less successful fulfilling a query of the form "How many deer images were taken at the Main camera." Improving this latter capability is also the subject of our on-going work.

While the purpose of the experiment is to identify images of commonly occurring species, the image corpus contains a number of images that either contain no animals (the Empty category) or small birds, rodents, or other fauna. Figure 10 details the the accuracy of the model with respect to differentiating bears, deer, and coyote, from empty images and images containing other species.

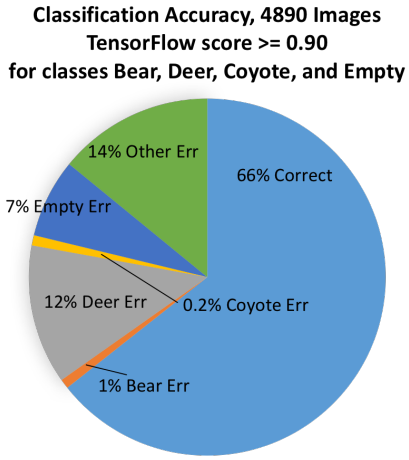

Figure 10: Common Animal Classification Accuracy Across all Test images Using a TensorFlow Score ≥ 0.90 for Bear, Deer, Coyote, and Empty for 4890 Randomly Selected Images.

Note that the error rates for bear, deer, and coyote images are the same as in Figure 7. However in Figure 10 it is clear that the model incurs more error when attempting to differentiate an empty image from one that may contain other features. This figure does indicate, however, that our technique is better at identifying animal species for which it has been explicitly trained.

3.2 Comparing to a Traditional Training Approach

We next compare WTB to more traditional model training. Using the traditional approach, we train the model using real images from the Sedgwick Main camera trap that contain bear and deer, and that are empty. We refer to results that use this model as REAL. Note that this scenario requires that we transmit all training images to the cloud to perform the training.

We randomly select 10000 day images and manually label them. In this set, there are 17 corrupted images, 458 bear images, 3725 deer images, 53 coyote images, 2828 empty images, and 2919 images containing other animals (birds and rodents in our case). Since TensorFlow requires at least 100 examples for training, we are unable to classify coyotes using the REAL method (a second disadvantage of a non-WTB approach). Thus, we do not train or test WTB or REAL using coyote images.

We divide the images into two sets, one containing 80% of the images for training, and one containing 20% of the images (i.e. the test set), which we use for evaluating the accuracy of each approach. In the 80% set, there are 366 bear images, 2980 deer images, and 2262 empty images, for a total of 5608 training images. In the 20% set, there are 92 bear images, 745 deer images, and 566 empty images, for a total of 1403 testing images. We train the REAL model using the 80% training set for bear, deer, and empty. We train the WTB using synthesized images for bear and deer and the 250 empty images that provide the base for the synthesized images. WTB thus does not transmit any additional images to the cloud for training.

Table 4: Comparison to Traditional Training (using real images): Classification Performance for WTB versus the traditional approach (REAL) for 1403 random test images. Classifications constitute images classified by TensorFlow with a score of 0.90 or higher for Bear, Deer, and Empty. The data shows the number of correctly classified images (Cor.), false positives, and false negatives (due to low scores).

	WTB			REAL		
Image Type	Cor.	False Pos.	False Neg.	Cor.	False Pos.	False Neg.
Bear	87	5	5	89	0	3
Deer	537	0	208	698	0	47
Empty	409	13	157	550	8	16

We use the same 20% sets (bear, deer, and empty images) to test the accuracy of both models (WTB and REAL). We present the results when we use a TensorFlow score of 0.90 or higher for Bear, Deer, and Empty in Table 4. This table shows the total number of correct classifications (Cor.), false positives, and false negatives.

Both models perform similarly and have a very low error rate for bears. Using WTB, 5 deer images are misclassified as Bear and 13 deer images were classified as Empty. For REAL, 2 bear images are classified as Empty and 6 deer images are classified as Empty. The false negative values represent images that are not classified correctly given the 0.90 TensorFlow scoring threshold. WTB has significantly higher false negatives for Deer and Empty relative to traditional training. Both approaches, however, have low overall error percentages per class when we consider false negatives and false positives together. Using the traditional approach (REAL training), the total error rate is 0.07% for Bear, 1.07% for Deer, and 0.55% for Empty. For WTB, the total error rate is 0.23% for Bear, 4.75% for Deer, and 3.89% for Empty.

Finally, we analyze images in the test set that have image types not trained for. In this experiment, we process the images that we manually classified as Other. We did not train either WTB or REAL using these images or classes and thus test using all 2919 images labeled Other. We find that using the REAL model, TensorFlow classifies 94% of the images with a score of 0.90 or higher (for Bear, Deer, or Empty). Of these images, REAL classifies 99% of the Other images as Empty. WTB classifies only 46% of the images with a score of 0.90 or higher; of these WTB classifies 94% of the Other images as Empty.

3.3 Network Transfer Time Savings

From the data from Table 1 for the Main camera daytime images, it is possible to compute the time savings that WTB would have enabled for our first random sample of 4890 images. Table 5 summarizes this comparison.

In our experiments, we use an unoptimized file transfer protocol to move data from the Sedgwick edge cloud to the Aristotle campus private cloud [4] located on the UCSB main campus. To provide a more realistic estimate of production usage, however, we compute the time savings assuming that a parallel file transfer capability (i.e. once that uses multiple network streams) is available. Using an optimistic estimate of 5 megabits per second (Mb/s) from Sedgwick

to Aristotle (for parallel file transfer), moving all 4890 images would have required approximately 319 seconds. Alternatively, including false positives, transferring only the classified 1473 classified images would have require 96 seconds – approximately one third of the time.

If the accuracy data shown in Figure 7 and Table 3 are representative for the entire set of 238,489 daytime images from the Main camera, WTB would cut down the transfer time from 19,212 seconds (5.3 hours) to 5,763 seconds (approximately 1.6 hours) as shown in the third row of Table 5. Note also that the trained model, which 490MB in size must be moved from UCSB to the Sedgwick edge cloud. Assuming that the bandwidth is symmetric in both directions, using 5 MB/s as the transfer rate, the time to move the model is a constant 98 seconds which is insignificant in the context of the full image corpus. Clearly, with respect to the edge cloud to private cloud network connectivity, it is more advantageous to move the code to the data than it is to move the data to the code.

However, the advantage could be overshadowed by the time necessary to move the images from the cameras themselves to the edge cloud. That is, if the throughput from the cameras to the edge is less than or equal to the throughput from the edge to the private cloud or the public cloud, the advantage is lost. In our examples, the cameras communicate with the edge cloud in the Sedgwick headquarters building using high-performance directional 802.11 wireless networking that delivers approximately 114 Mb/s. That is, the bandwidth to the edge cloud from the cameras is more than a factor of 20 larger than from the edge cloud to the Aristotle campus private cloud at UCSB.

Moreover, the 5 Mb/s rate from the edge cloud to the campus cloud is significantly higher than the rate from the edge cloud at Sedgwick to either the Box or AWS public clouds. Because these services provide free data ingress, the transfer rates they offer to our project (without a premium fee) are 1.36 Mb/s and 1.61 Mb/s respectively. The fourth and fifth rows of Table 5 show the transfer times for the full corpus of daytime images from the Main camera. Thus the WTB edge cloud architecture offers even greater network transfer time savings (by almost a factor of 3) if the data is moved to the UCSB private cloud rather than the public cloud. Most strikingly, however, the savings resulting from the use of an edge cloud at Sedgwick and the Aristotle private cloud at UCSB versus directly sending the images to Box or AWS S3 is more than a factor of 10 (5,763 seconds versus 70,633 seconds and 59,665 seconds respectively).

Table 5: Data Transfer Time Comparison for Different Cloud Destinations.

Destination	Full Image Set	Edge Classified Images Only
UCSB Aristotle (rand. sample)	319 sec.	96 sec.
UCSB Aristotle (all Main)	19,212 sec.	5,763 sec.
Box (all Main)	70,633 sec.	17,889 sec.
AWS S3 (all Main)	59,665 sec.	21,190 sec.

3.4 OCR Analysis

To evaluate the generality of WTB, we replace the TensorFlow analysis plug-in with an OpenCV image analysis tool for optical character recognition (OCR) and a utility for processing JPEG images (pyexinfo and exiftool). We use OCR to extract the air temperature that each camera prints on each picture. The Main camera (unlike all others in our set) also records this information in the image metadata. We use the JPEG processing tool to extract the temperature from the metadata. This enables us to evaluate the accuracy of the OCR tool since we can compare its output with that of the JPEG processing tool.

We train the OCR tool as described previously. We identify images with temperature values that cover all digits (0-9). We expedite this search using the WTB weather sensor data which enables us to pinpoint dates and times with different temperature values. In this study, we consider all images (day and night) from the Main camera (638062 images).

On average, JPEG processing requires 0.01 seconds per image. OCR when implemented as a library (wrapped via the WTB wrapper) requires 0.33 seconds on average per image with a standard deviation of 0.05 seconds. We also investigated implementing OCR as a simple web service (instead of a library). The service processes each image in 0.41 seconds on average with a standard deviation of 0.02 seconds. The difference between the performance of the service and the library is that the service implements the call and return as a request/response pair via HTTP, and transmits arguments and return values via JSON, which adds communication overhead.

Without training the OCR algorithm (i.e. using the default OCR model in the tool), the tool is able to achieve 91% accuracy (55190 errors) across these images. With training the OCR algorithm, WTB achieves 100% accuracy in temperature extraction. Thus it is possible to quickly extract the temperature encoded in our images (including those without temperature values in the JPEG metadata) and to transmit or export via HTTP, a subset of images within queried temperature ranges by scientists and citizen scientists using WTB.

4 RELATED WORK

WTB integrates technologies from a number of different research areas. These areas include animal monitoring, image classification and object detection, and IoT-based remote sensing via integration with edge tier systems.

In the area of animal monitoring and wild life tracking, some early work includes usage of wearable tracking devices with GPS, memory, and computing capabilities and can be deployed in the wild and operate as a peer-to-peer network. [22]. A more recent work by Huang et al. [18] uses collars as well to track animal interactions. Authors demonstrate how the cost of the devices needed to deploy animal tracking systems decreased over time. An ongoing project, CraneTracker [3], uses a leg band tracker with multiple sensors, including GPS and multi-modal radio, to track migration of Cranes over 4000km from north-central Canada to southern Texas. Mainwaring et al. [25] provide an in depth overview of the system design requirements for tracking wild animals with the focus on sensor networks, efficient sampling, and communication. Their setup includes sensors deployment in the natural habitat, data sampling, and data display through an online web interface. OzTrack

[19] is another, more recent, example of an end-to-end system that can store, analyze, and visualize data coming from wild life trackers.

In the area of image classification for animal detection and tracking, Burghardt et al. [7] use human face detection to detect and track lion faces to achieve a temporally coherent detection and tracing. They further use information generated by tracker to boost the priors. Zeppelzauer [39] presents a method that can identify elephants in wildlife videos by first dynamically learning their color model. The authors of this work use this model to detect elephants while also considering spatially and temporally consistent detections that appear in sequences. With this approach most elephants can be detected with greater than 90 percent accuracy. Tillett et al. [33] describe a use of a point model distribution to detect and track pigs in videos. The authors fit a model through sequences of data that provide information on the animals' position and posture. They use the same model to capture the interaction and activities of multiple pigs in the same frame and follow animals through the sequence of frames. Finally, Xiaoyuan et al. [38] propose a framework that automatically identifies wildlife species from the pictures taken by remote camera traps. In the process of species identification they manually crop out the animal images from the background and then perform multiple algorithms for image classification including Sparse Coding Spatial Pyramid Matching (ScSPM) and Support Vector Machines. They report 82% accuracy on average from 7000 images and 18 species.

Our work differs from this prior work in that we use convolutional neural networks to classify our images. In addition, we do not require learning using pictures of the animals we are attempting to classify. To avoid transfer of thousands of images for the learning phase from the sensing tier to the private/public cloud tiers where computationally and data intensive can be performed, we instead transmit a very small number of empty images for each camera trap at different times of the day. We then automatically generate "fake" images of for training the neural network (TensorFlow with Inception-v3 in our case) by randomly placing images of the animals of interest (bears, deer, coyotes in our case) from Google Images on the empty background.

Extant investigations into IoT have identified the need for latency reduction for sensor driven, cloud-backed applications, but promote and pursue only content/data caching [5, 17], aggregation, compression, and filtering in the edge tier [8, 14], or data processing using very resource restricted and mobile devices (e.g. smart phones) [9]. WTB is unique in that it places a self-managing distributed system in the edge tier at fixed, well known locations, and uses them to mirror the functionality available from public cloud vendors but at a smaller scale. As such, we are able to leverage edge clouds to perform complex analytics, machine learning, and provide robust decision support for IoT applications at or near the data source (with very low latency), unlike any system or service available today.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we design, implement, and empirically evaluate an end-to-end, IoT edge-cloud system for image processing and analytics that enables automatic wildlife monitoring in remote locations. We customize and deploy our system at the UCSB Sedgwick Research Reserve and use it to categorize camera-trap pictures that contain bears, deer, and coyotes. Our multi-tier IoT system, called WTB, connects motion-triggered cameras via WIFI to an on-reserve, Internet-connected, distributed system (which we call an edge cloud).

We use WTB to develop a new approach to neural network training for animal recognition in images. Since this process is computationally intensive and requires both GPU support and a large number of CPUs (for parallelism), model training must be performed at the campus or public cloud. To avoid transmission of a large number (1000s) of example (animal) images from Sedgwick to the campus or public cloud to train with, we instead a small number of "empty" (background-only) images. At the campus or public cloud, we then automatically synthesize a large training data set from them by overlaying randomly placed animals available and easily accessible from Google Images. Once training is complete, we ship the trained model to the images – i.e. to the edge cloud at Sedgwick, where we perform classification. By performing this classification on-site, the WTB system can then transmit only those images containing animals of interest to each of the scientists that request them, saving significant time (both for transfer and manual analysis) and network bandwidth.

We implement and deploy WTB between UCSB and Sedgwick Reserve and empirically evaluate its accuracy and bandwidth savings using a large number of images from one of the Sedgwick camera traps. We find that WTB achieves 0.2% error for coyote, 1% error for bear, and 12% error for deer. Compared to the traditional training approach in which we use (and transfer) actual (non-synthesized) images, we find that WTB has an overall error rate (including both false positives and false negatives) of 9% versus 3% for the traditional approach, for bear, deer, and empty images. Moreover, WTB is able to classify images (e.g. for coyote) for which there are too few actual images to do so using the traditional approach. Finally, we find that WTB can reduce network transfer over the slow, long-haul network by 70% for the use cases we consider.

As part of future work, we are investigating how to reduce our error rates further, and how to apply this approach to other animals of interest. To do so, we are investigating the incorporation of specific ecological details at particular sensor sites (fauna, habitat, time of year, water availability etc.) and employing additional animal positioning (e.g. facing toward and away from the camera) in our image construction and training process. We also plan to develop a notification system that uses real-time feedback from image analysis to alert authorities of poachers and to alert visitors (hikers, students, and researchers) to potential animal presence. Finally, we are also investigating how to implement different types of queries via image classification and combining image analysis with other types of IoT sensor and Internet data. In particular, we are interested in using this approach to provide data-driven decision support for sustainable agriculture processes and ranching at Sedgwick [24].

ACKNOWLEDGMENTS

This work is supported in part by NSF (CCF-1539586, CNS-0905237, and ACI-1541215), NIH (1R01EB014877-01), the Naval Engineering Education Consortium (NEEC-n00174-16-C-0020), the California

Energy Commission (PON-14-304), and Sedgwick Reserve. We especially thank Grant Canova-Parker, Kate McCurdy, and Anthony Nelson of Sedgwick and UCSB for their help with this effort.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Zhifeng Chen, A. Davis, J. Dean, M. Devin, Sanjay Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Symposium on Operating Systems Design and Implementation (OSDI)*.

[2] A. Alemi. 2016. Improving Inception and Image Classification in TensorFlow. https://research.googleblog.com/2016/08/improving-inception-and-image.html. (2016). [Online; accessed 22-Aug-2016].

[3] David Anthony, William P Bennett, Mehmet C Vuran, Matthew B Dwyer, Sebastian Elbaum, Anne Lacy, Mike Engels, and Walter Wehtje. 2012. Sensing through the continent: towards monitoring migratory birds using cellular sensor networks. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*. ACM, 329–340.

[4] Aristotle Cloud Federation 2017. Aristotle Cloud Federation. (2017). https://federatedcloud.org [Online; accessed 22-Aug-2016].

[5] AWS Edge Locations 2016. AWS Edge Locations. (2016). https://aws.amazon.com/cloudfront/ [Online; accessed 22-Aug-2016].

[6] G. Bradski. 2000. OpenCV. *Dr. Dobb's Journal of Software Tools* (2000).

[7] Tilo Burghardt, Janko Calic, and Barry T Thomas. 2004. Tracking Animals in Wildlife Videos Using Face Detection.. In *EWIMT*.

[8] Cisco. 2016. Fog Data Services - Cisco. (2016). http://www.cisco.com/c/en/us/products/cloud-systems-management/fog-data-services/index.html [Online; accessed 22-Aug-2016].

[9] Cloudlet-based Mobile Computing 2016. Cloudlet-based Mobile Computing. (2016). http://elijah.cs.cmu.edu/ [Online; accessed 22-Aug-2016].

[10] Content Distribution Network 2016. Content Distribution Network. https://en.wikipedia.org/wiki/Content_delivery_network. (2016). [Online; accessed 22-Aug-2016].

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[12] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition.. In *ICML*. 647–655.

[13] ExifTool 2016. ExifTool. http://www.sno.phy.queensu.ca/~phil/exiftool/. (2016). [Online; accessed 22-Aug-2016].

[14] D. Floyer. 2016. The Vital Role of Edge Computing in the Internet of Things. (2016). http://wikibon.com/the-vital-role-of-edge-computing-in-the-internet-of-things/ [Online; accessed 22-Aug-2016].

[15] Google. 2016. Google Images API. http://images.google.com. (2016). [Online; accessed 22-Aug-2016].

[16] Google 2016. TensorFlow Image Classification for Image Classification. https://www.tensorflow.org/versions/r0.10/tutorials/image_recognition/index.html. (2016). [Online; accessed 22-Aug-2016].

[17] Google Cloud CDN 2016. Google Cloud CDN. https://cloud.google.com/cdn/docs/. (2016). [Online; accessed 22-Aug-2016].

[18] Jyh-How Huang, Ying-Yu Chen, Yu-Te Huang, Po-Yen Lin, Yi-Chao Chen, Yi-Fu Lin, Shih-Ching Yen, Polly Huang, and Ling-Jyh Chen. 2010. Rapid prototyping for wildlife and ecological monitoring. *IEEE Systems Journal* 4, 2 (2010), 198–209.

[19] Jane Hunter, Charles Brooking, Wilfred Brimblecombe, Ross G Dwyer, Hamish A Campbell, Matthew E Watts, and Craig E Franklin. 2013. OzTrack–E-Infrastructure to Support the Management, Analysis and Sharing of Animal Tracking Data. In *eScience (eScience), 2013 IEEE 9th International Conference on*. IEEE, 140–147.

[20] Intel NUC 2016. Intel NUC. (2016). http://www.intel.com/content/www/us/en/nuc/overview.html [Online; accessed 14-Feb-2016].

[21] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).

[22] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. 2002. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet. *ACM Sigplan Notices* 37, 10 (2002), 96–107.

[23] K-Nearest Neighbor 2017. K-Nearest Neighbor Algorithm. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. (2017). [Online; accessed 22-Feb-2017].

[24] C. Krintz, R. Wolski, N. Golubovic, B. Lampel, V. Kulkarni, B. Sethuramasamyraja, B. Roberts, and B. Liu. 2016. SmartFarm: Improving Agriculture Sustainability Using Modern Information Technology. In *KDD Workshop on Data Science for Food, Energy, and Water*.

[25] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. 2002. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. ACM, 88–97.

[26] Daniel Nurmi, Richard Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. 2009. The Eucalyptus Open-Source Cloud Computing System. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*. IEEE, 124–131.

[27] OpenCV. 2016. OpenCV Optical Character Recognition Using K-Nearest Neighbors. http://docs.opencv.org/3.1.0/dd/de1/classcv_1_1ml_1_1KNearest.html. (2016). [Online; accessed 22-Aug-2016].

[28] PyExifInfo 2016. PyExifInfo. https://github.com/guinslym/pyexifinfo. (2016). [Online; accessed 22-Aug-2016].

[29] K. Sato. 2016. How a Japanese cucumber farmer is using deep learning and TensorFlow. https://cloud.google.com/blog/big-data/2016/08/how-a-japanese-cucumber-farmer-is-using-deep-learning-and-tensorflow. (2016). [Online; accessed 22-Aug-2016].

[30] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[31] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567* (2015).

[32] Eranda Tennakoon, Charith Madusanka, Kasun De Zoysa, Chamath Keppitiyagama, Venkat Iyer, Kasun Hewage, and Thiemo Voigt. 2015. Sensor-based breakage detection for electric fences. In *Sensors Applications Symposium (SAS), 2015 IEEE*. IEEE, 1–4.

[33] R. Tillett, C. Onyango, and J. Marchant. 1997. Using model-based image processing to track animal movements. *Computers and electronics in agriculture* 17, 2 (1997), 249–261.

[34] L. Torrey and J. Shavlik. 2009. *Transfer Learning*. IGI Global, E. Soria, J. Martin, R. Magdalena, M. Martinez and A. Serrano, eds. http://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf.

[35] UCSB Sedgwick Reserve 2013. (2013). http://sedgwick.nrs.ucsb.edu [Online; accessed 14-Feb-2016].

[36] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. 2012. Cloudlets: bringing the cloud to the mobile user. In *ACM workshop on Mobile cloud computing and services*. ACM.

[37] Wikipedia. 2016. Deep Learning. https://en.wikipedia.org/wiki/Deep_learning. (2016). [Online; accessed 22-Aug-2016].

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

[39] Matthias Zeppelzauer. 2013. Automated detection of elephants in wildlife video. *EURASIP journal on image and video processing* 2013, 1 (2013), 1.