To the University of Wyoming:

The members of the Committee approve the dissertation of Mohammad Sadegh Norouzzadeh presented on May 17, 2019.

Dr. Jeff Clune, Chairperson

Dr. Farhad Jafari, External Department Member

Dr. James Caldwell

Dr. John Hitchcock

Dr. Amy Banic

Dr. Chao Lan

APPROVED:

James Caldwell, Head, Department of Computer Science

Micheal Pishko, Dean, College of Engineering and Applied Science
Our ability to study and conserve ecosystems directly depends on how much information we have about them. Motion-activated cameras also known as camera traps are cheap and non-intrusive tools to gather millions of images from wildlife. However, extracting useful information such as species, count, and the behavior of animals from the collected images is often done manually, and it is so slow and expensive that a lot of invaluable information is not extracted and thus remain untapped. This manual labor is the main roadblock in the widespread usage of camera-trap arrays. I devoted my Ph.D. dissertation to reducing the manual burden of information extraction from camera-trap images using advanced machine learning methods. For the first step, I demonstrated that such information can be automatically extracted by deep learning, a cutting-edge type of artificial intelligence. I trained deep convolutional neural networks to identify, count, and describe the behaviors of 48 species in the 3.2-million-image Snapshot Serengeti dataset. Our deep neural networks automatically identify animals with over 94.9% accuracy, and we expect that number to improve rapidly in years to come. More importantly, if my system classifies only images it is confident about, it can automate animal identification for 99.3% of the data while still performing at the same 96.6% accuracy as that of crowdsourced teams of human volunteers. This automation saves more than 8.4 years (i.e., over 17,000 hours at 40 hours per week) of human labeling effort on this 3.2-million-image dataset.

Although I achieved outstanding results on the Snapshot Serengeti dataset, the accuracy of results highly depends on the amount, information-richness, quality, and diversity of the available data to train the models. Many camera-trap projects do not have a large, detailed set of available labeled images and hence cannot benefit from my suggested machine learning techniques. In the second part of my dissertation, I combined the power of advanced machine learning algorithms and human intelligence to build a scalable, fast, and accurate active learning system to maximally reduce the amount of manual work to identify and count animals in camera-trap images. I showed that my proposed procedure could achieve more than 90.9% accuracy on the SS dataset with as little as 14,000 labels, which matches state
of the art results while saving over 99.5% of human labor for labeling. Those efficiency gains highlight the importance of using deep neural networks to automate data extraction from camera-trap images, suggesting that deep learning could enable the inexpensive, unobtrusive, high-volume, and even real-time collection of a wealth of information about vast numbers of animals in the wild.
AUTOMATIC INFORMATION EXTRACTION
FROM CAMERA-TRAP IMAGES USING DEEP
LEARNING

by

Mohammad Sadegh Norouzzadeh, M.Sc.

A dissertation submitted to the
Department of Computer Science
and the
University of Wyoming
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY
in
COMPUTER SCIENCE

Laramie, Wyoming
August 2019
Copyright © 2019

by

Mohammad Sadegh Norouzzadeh
To my family, who always encouraged me to learn more...
Contents

List of Figures vii

List of Tables ix

Acknowledgments x

Chapter 1 Introduction 1

1.1 Research Unrelated to Camera Traps . . . . . . . . . . . . . . . . . . . . . . 3
1.2 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Dissertation Overview and Organization . . . . . . . . . . . . . . . . . . . . 6

Chapter 2 Automatically Identifying, Counting, and Describing Wild Ani-

mals in Camera-Trap Images with Deep Learning 7

2.1 Abstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.2 Significance Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.3 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.4 Background and Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . 12
   2.4.1 Machine Learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
   2.4.2 Deep Learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
   2.4.3 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   2.4.4 Snapshot Serengeti Project . . . . . . . . . . . . . . . . . . . . . . . . 15
2.5 Experiments and Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
   2.5.1 Datasets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.5.2 Task I: Detecting Images That Contain Animals ........................................... 19
2.5.3 Task II: Identifying Species ............................................................................. 20
2.5.4 Task III: Counting Animals ........................................................................... 21
2.5.5 Task IV: Additional Attributes ....................................................................... 22
2.5.6 Pre-processing and Training ........................................................................... 22
2.5.7 One-stage Identification ................................................................................... 27
2.5.8 Results on the Volunteer-Labeled Test Set ....................................................... 28
2.5.9 Comparing to Gomez et al. 2016 .................................................................. 28
2.5.10 Day vs. Night Accuracy .................................................................................. 30
2.5.11 Helping Small Camera-Trap Projects Via Transfer Learning ......................... 31
2.5.12 Prediction Averaging ...................................................................................... 34
2.5.13 Classifying Capture Events ............................................................................ 34
2.5.14 Saving Human Labor via Confidence Thresholding ....................................... 35
2.5.15 Improving Accuracy for Rare Classes ............................................................ 37
2.5.16 Validating Against the NACTI Dataset .......................................................... 42
2.5.17 Analysis of Visual Features Learned by Machines to Classify Species .......... 43
2.6 Discussion and Future Work .............................................................................. 43
2.7 Conclusions ....................................................................................................... 44

Chapter 3 An Active Deep Learning System for Species Identification and Counting in Camera-Trap Images 56
3.1 Abstract ............................................................................................................... 56
3.2 Introduction ......................................................................................................... 57
3.3 Background and Related Work .......................................................................... 58
3.3.1 Deep Learning ............................................................................................... 58
3.3.2 Image Classification ...................................................................................... 60
3.3.3 Object Detection ........................................................................................... 60
3.3.4 Transfer Learning .......................................................................................... 61
3.3.5 Active Learning ............................................................................................. 62
3.3.6 Embedding Learning ..................................................................................... 62
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.7</td>
<td>Datasets</td>
<td>63</td>
</tr>
<tr>
<td>3.4</td>
<td>Methods</td>
<td>64</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Proposed Pipeline</td>
<td>64</td>
</tr>
<tr>
<td>3.5</td>
<td>Experiments and Results</td>
<td>67</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Empty vs. Animal</td>
<td>67</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Counting</td>
<td>68</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Species Identification</td>
<td>68</td>
</tr>
<tr>
<td>3.6</td>
<td>Further Improvement</td>
<td>70</td>
</tr>
<tr>
<td>3.7</td>
<td>Conclusion</td>
<td>70</td>
</tr>
</tbody>
</table>

**Chapter 4** Neuromodulation Improves the Evolution of Forward Models 75

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Abstract</td>
<td>75</td>
</tr>
<tr>
<td>4.2</td>
<td>Introduction and background</td>
<td>76</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Neural Networks</td>
<td>78</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Plasticity</td>
<td>80</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Neuromodulation</td>
<td>81</td>
</tr>
<tr>
<td>4.3</td>
<td>Methods</td>
<td>81</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Test Problem</td>
<td>83</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Algorithm Details</td>
<td>85</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Fitness Evaluation</td>
<td>86</td>
</tr>
<tr>
<td>4.4</td>
<td>Experiments and Results</td>
<td>87</td>
</tr>
<tr>
<td>4.5</td>
<td>Conclusion</td>
<td>91</td>
</tr>
</tbody>
</table>

**Chapter 5** Discussion and Conclusion 93

References 96
List of Figures

1.1 Camera traps placed in wildlife ................................................. 2

2.1 An example camera-trap image and its deep-learning-extracted predictions. ............................. 10
2.2 Various factors that make identifying animals in camera-trap images difficult. ........................... 10
2.3 The hierarchy of abstractions learned by deep neural networks ................................................. 13
2.4 Example of a capture event and its noisy label .............................................................................. 15
2.5 Results against the expert-labeled test set ..................................................................................... 21
2.6 Examples of correct classifications .............................................................................................. 23
2.7 Examples of partially incorrect classifications ................................................................................ 24
2.8 An example of our pre-processing procedure ................................................................................ 25
2.9 Results against the volunteer-labeled test set .............................................................................. 29
2.10 Comparison to Gomez et al. 2016 ............................................................................................... 30
2.11 Results on classifying capture events .......................................................................................... 36
2.12 Accuracy and confidence of the best model for task I: detecting images that contain animals .......... 38
2.13 Accuracy and confidence of the ensemble of models for task II: identifying species .................... 39
2.14 Accuracy and confidence of the best model for task III: counting animals .................................. 40
2.15 The distribution of different species in the SS dataset ................................................................. 48
2.16 Results of applying different methods to combat the imbalance issue ........................................ 49
2.17 Confusion tables for task I and task III ....................................................................................... 50
2.18 Confusion table for task II ......................................................................................................... 51
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Different deep learning architectures employed in this dissertation.</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Accuracy of different models on task I: detecting images that contain animals.</td>
<td>20</td>
</tr>
<tr>
<td>2.3</td>
<td>Static training hyperparameters of different experiments.</td>
<td>26</td>
</tr>
<tr>
<td>2.4</td>
<td>Dynamic training hyperparameters of all experiments.</td>
<td>26</td>
</tr>
<tr>
<td>2.5</td>
<td>Results of the one-stage identification experiment.</td>
<td>27</td>
</tr>
<tr>
<td>2.6</td>
<td>Results of transfer learning vs. training from scratch.</td>
<td>46</td>
</tr>
<tr>
<td>2.7</td>
<td>An example of prediction averaging.</td>
<td>47</td>
</tr>
<tr>
<td>2.8</td>
<td>The accuracy on capture events for task I: detecting images that contain animals.</td>
<td>47</td>
</tr>
<tr>
<td>2.9</td>
<td>The accuracy, precision, and recall of the ensemble of models for task IV: additional attributes</td>
<td>52</td>
</tr>
<tr>
<td>3.1</td>
<td>Confusion table for the pre-trained object detection model applied to the SS dataset</td>
<td>67</td>
</tr>
<tr>
<td>4.1</td>
<td>The evolutionary algorithm parameters.</td>
<td>88</td>
</tr>
</tbody>
</table>
Acknowledgments

This dissertation describes the primary research that I have conducted as a Ph.D. student in the Computer Science Department at the University of Wyoming under the supervision of Dr. Jeff Clune. I acknowledge the tremendous support from my advisor, my labmates, the department, and the University of Wyoming for my research and accomplishments.

During my Ph.D. education, I had a chance to be a member of the Evolving Artificial Intelligence Laboratory and interact with and learn from its other members, especially: Dr. Henok Mengistu, Dr. Anh Nguyen, Dr. Joost Huizinga, Dr. Nick Cheney, Dr. Roby Velez, Christopher Stanton, and Tyler Jaszkowiak.

Moreover, I would like to thank other people who I worked with directly during my Ph.D., especially Allyson Anderson and ARCC members at the University of Wyoming; Dr. Micheal Tabak at USDA National Wildlife Research Center; Dr. Izzat Izzat and Dr. Yang Zheng at Aptiv; Dr. Dan Morris, Dr. Neel Joshi, and Dr. Nebojsa Jojic at Microsoft; Zhongqi Miao at UC Berkeley; and Sara Beery at Caltech.

My special thanks go to Professor Jeff Clune, who I learned a lot from: scientifically, intellectually, personally, and grammatically. None of my achievements would have been possible without his excellent vision and help. I also appreciate the advice from my Ph.D. committee members: Professors James Caldwell, John Hitchcock, Amy Banic, Chao Lan and Farhad Jafari (Math Department).

Last but not least, I should thank my lovely parents Ali Akbar and Zahra who always encouraged me to study more and work harder, my pretty wife Reihaneh who I always had her support, my supportive siblings Payam, Parya, and Amir who continuously pushed me forward, and my grandmother Touran for her uninterrupted kindness.
Chapter 1

Introduction

Camera traps are fixed cameras that can automatically take pictures of wildlife. Camera traps have revolutionized many aspects of wildlife ecology and conservation over the last two decades (Fig. 1.1) [1]. This essential tool has enabled biologists to study population sizes and distributions [2], evaluate habitat conditions [3], identify new species [4], etc. Although camera-trap networks are capable of collecting millions of images [5–7], humans are still needed to process (i.e. look at) the images and extract relevant information. This manual work is so costly and time-consuming that a lot of invaluable information in these repositories cannot be extracted within a reasonable time-frame and remains unprocessed.

To alleviate this backlog, the Snapshot Serengeti (SS) project employed crowdsourcing. They have coordinated a volunteer effort of nearly 28,000 citizen scientists, via their website (www.snapshotserengeti.org), to label millions of SS images. For each sequence of images, they gather labels from several non-expert citizen scientists. They then pool the votes with a simple plurality algorithm that is estimated to be 96.6% accurate for species identification and 90% accurate for counting [8]. The downside to the crowdsourcing is that it takes 2 to 3 months to label a 6-month batch of images, which is too slow for many ecological studies. Additionally, due to the monotony and tedium of the task, volunteers are leaving the project, which further slows things down. Moreover, there exist many other active camera-trap projects that are not lucky enough to recruit as many volunteers as the SS project. As technology advances and camera-trap devices get cheaper, more camera-trap projects are
expected, but the necessary manual labor is the main roadblock in the widespread usage of camera trap arrays.

An automatic, intelligent system that can extract information from images fed to it can improve camera-trap research by: 1) speeding up processing time and providing researchers with current, actionable data 2) making the technology more feasible and accessible for more institutions around the world and 3) freeing up resources, such as labor, for other projects. In this dissertation, I took steps toward automating the information extraction process using advanced computer vision techniques. My results show that this goal is reachable, and such a system could be made to operate with minimal human effort.

Figure 1.1: Researchers usually place camera traps on trees, or wooden poles, or steel poles. These devices can take one or more images whenever their motion-detection sensor triggers. Most of the camera-trap projects use an array of cameras to capture wildlife images within certain limits of an ecosystem. Images are from [8].

Deep neural networks (DNNs), also known as deep learning, have made a great stride in artificial intelligence in recent years. DNNs have been successfully applied to various kinds of problems ranging from image analytics and speech recognition to natural language processing. The outstanding performance of deep learning against other traditional machine learning techniques has made it an obvious choice to deal with complex problems. I dedi-
cated the main project of my Ph.D. dissertation to harnessing DNNs for fast and automatic information extraction from camera-trap images. DNNs exhibit near human-competitive performance at identifying objects in images [9][10]. They can also count the number of objects in images [11][12] and describe their context [13][14]. Therefore, DNNs are capable of extracting diverse kinds of information from camera-trap images.

1.1 Research Unrelated to Camera Traps

Before starting my main project, I researched forward models for robots. Forward models predict the outcomes of actions for robots, enabling them to rapidly simulate many actions without actually performing the actions [15][16]. However, such models must adjust over time to deal with changes in the environment or body, such as injury. Neural networks have shown the ability to model various complex functions with high accuracy; hence, they are good candidates to build forward models for robots. Neural networks are often static, which means that after training their parameters remain unchanged, and they cannot adapt themselves to changes in the world or their bodies. In contrast, plastic neural networks adjust their connections over time via local learning rules (e.g., Hebbian rule [17]) and can adapt to unforeseen changes. Neuromodulation is a biologically-inspired learning rule in which some neurons can control the rate of changes in connections between other neurons in different contexts [18]. As we demonstrated in my first paper, neuromodulation improves the evolution of forward models.

1.2 Contributions

My dissertation research has resulted in five major contributions, including three journal papers, one conference paper, and one pre-print paper targeted to a journal. Four of the contributions are about automatic information extraction from camera-trap images, and the fifth is about forward model for robots. In summary, my contributions are:

1. In chapter 2, we showed it is possible to automate identifying, counting, and describing
animals in camera trap images with state-of-the-art DNN computer vision algorithms. In particular, we trained DNN architectures on images from the SS project that were labeled by human volunteers. Our camera-trap algorithm is capable of identifying species with over 94.9% accuracy on the SS camera-trap dataset, one of the largest camera-trap dataset in history with 3.2 million labeled images. More importantly, we proved that if our system classifies only images, it is confident about, our system can automate animal identification for 99.3% of the data, while still performing at the same 96.6% accuracy as that of crowdsourced teams of human volunteers. This automation saves over 8.4 years (i.e., over 17,000 hours at 40 hours/week) of human labeling effort on this 3.2 million-image dataset. This research resulted in a paper on the cover of the Proceedings of the National Academy of Sciences (arguably the third-best journal in the world) and press coverage in BBC, MIT Technology Review, Science Daily, and Digital Trends, among others.

2. In collaboration with the US Department of Agriculture’s National Wildlife Research Center (USDA NWRC), we validated our camera-trap algorithm against the North American Camera Trap Images (NACTI) dataset (Ch. 2, Sec. 2.5.16). The NACTI dataset is a 3.6 million labeled images dataset from 70 different species gathered across five different locations within the United States. The model we trained on the NACTI dataset achieved 97.5% accuracy, which shows that our camera-trap model is effective against different camera-trap datasets. Moreover, we also trained a Wild Pig Detector Network which can distinguish between wild pig pictures and images from the other species with greater than 98.6% accuracy. This collaboration came with a $16,000 grant which covered my stipend for one semester and the cost of buying 10TB storage space on UW’s supercomputer. The resulting paper was appeared in Methods in the Ecology and Evolution Journal (Impact Factor= 6.36) in 2019.

3. Showing the effectiveness of DNNs to classify camera-trap images is incomplete without understanding how do DNNs process the images and what patterns they find in camera-trap images to classify them. In collaboration with a research group at UC Berkeley,
we studied the features DNNs learn to identify different species in camera-trap images (Ch. 2, Sec. 2.5.17). Our study showed that in most cases, DNNs find patterns that are similar to those humans employ to identify different species. Besides, our experiments exposed that DNNs not only learn animal patterns to classify the images, but they also associate some common background patterns to the species. This research will be published in Scientific Reports Journal (Impact Factor= 4.12) in 2019.

4. Although my research revealed that DNNs can extract different kinds of information such as species, count, and behavior of animals from camera-trap images accurately and quickly, to train an accurate DNN model we need a considerable amount of information-rich, noise-free, and diverse data. Many camera-trap projects do not have a large set of available labeled images and hence cannot benefit from the proposed deep learning technique in chapter 2. Furthermore, even in the case of having many labeled images, most of camera-trap projects only maintain coarse sequence-level or image-level labels with no further details such as the location of animals in the images or the animals boundaries. Without fine-grained labels, deep learning models tend to not only rely on species patterns to extract information but also patterns of the background scenes. Such a fact restraints the application of deep learning models to other similar datasets or even different seasons of the same dataset.

In chapter 3, we further tested whether these neural networks can automate knowledge extraction for projects with limited or no labeled images, catalyzing a shift towards creating BigData from the natural world and cheaply extracting information from them, which would transform many fields of biology. We combined the power of advanced machine learning with human intelligence to build an automatic, accurate, and inexpensive tool for extracting information from camera-trap pictures. Our suggested pipeline can save over 99.5% of manual labeling effort, while producing labels as accurate as of the state-of-the-art results. The automated extraction of such information could catalyze and enhance many studies in the fields of ecology, conservation biology, animal behavior, and zoology and save time and cost for scientists enabling them to put more effort on their more important missions.
5. In chapter 4, we tested the hypothesis that neuromodulation can improve the evolution of forward models by heightening learning after drastic changes such as injury. We compared forward models evolved with neuromodulation to those evolved with static neural networks and Hebbian plastic neural networks. The results show that neuromodulation produces forward models that can adapt to changes significantly better than the controls. Our findings suggest that neuromodulation is an effective tool for enabling robots (and artificial intelligence agents more generally) to have more adaptable, effective forward models. I orally presented the results of this research at the Genetic and Evolutionary Computation Conference (GECCO) 2016, and the resulting paper has been published in the proceeding of the conference.

1.3 Dissertation Overview and Organization

I have written the chapters of this dissertation so that they can be read independently. The rest of the dissertation is organized as follows. Chapter 2 contains my main paper: “Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning” and summarizes of the results of our collaborations with USDA NWRC and UC Berkeley. Chapter 3 contains my next paper: “An active deep learning system for species identification and counting in camera-trap images”. My evolutionary robotics paper: “Neuromodulation improves the evolution of forward models” is presented in chapter 4. Finally, chapter 5 consists of the conclusion of the dissertation and future work directions.
Chapter 2

Automatically Identifying, Counting, and Describing Wild Animals in Camera-Trap Images with Deep Learning

All the contents of this chapter, except section 2.5.16 and 2.5.17, are from the following publication:


Section 2.5.16 briefly summarizes how we applied technology developed in our PNAS paper for a different dataset and slightly different purpose, the result of which led to the following paper:

Section 2.5.17 is a high-level summary of our study on how do DNNs classify species in camera-trap images, which resulted in the following paper:


2.1 Abstract

Having accurate, detailed, and up-to-date information about the location and behavior of animals in the wild would improve our ability to study and conserve ecosystems. We investigate the ability to automatically, accurately, and inexpensively collect such data, which could help catalyze the transformation of many fields of ecology, wildlife biology, zoology, conservation biology, and animal behavior into “big data” sciences. Motion sensor “camera traps” enable collecting wildlife pictures inexpensively, unobtrusively, and frequently. However, extracting information from these pictures remains an expensive, time-consuming, manual task. We demonstrate that such information can be automatically extracted by deep learning, a cutting-edge type of artificial intelligence. We train deep convolutional neural networks to identify, count, and describe the behaviors of 48 species in the 3.2-million-image Snapshot Serengeti dataset. Our deep neural networks automatically identify animals with over 93.8% accuracy, and we expect that number to improve rapidly in years to come. More importantly, if our system classifies only images it is confident about, our system can automate animal identification for 99.3% of the data while still performing at the same 96.6% accuracy as that of crowdsourced teams of human volunteers, saving more than 8.4 years (i.e. over 17,000 hours at 40 hours per week) of human labeling effort on this 3.2-million-image dataset. Those efficiency gains highlight the importance of using deep neural networks to automate data extraction from camera-trap images, reducing a roadblock for this widely used technology. Our results suggest that deep learning could enable the inexpensive, unobtrusive, high-volume, and even real-time collection of a wealth of information about vast numbers of animals in the wild.
2.2 Significance Statement

Motion-sensor cameras placed in natural habitats offer the opportunity to inexpensively and unobtrusively gather vast amounts of data on animals in the wild. A key obstacle to harnessing their potential is the great cost of having humans analyze each image. Here we demonstrate that a cutting-edge type of artificial intelligence called deep neural networks can automatically extract such invaluable information. For example, we show deep learning can automate animal identification for 99.3% of the 3.2 million-image Snapshot Serengeti dataset while performing at the same 96.6% accuracy of crowdsourced teams of human volunteers. Automatically, accurately, and inexpensively collecting such data could help catalyze the transformation of many fields of ecology, wildlife biology, zoology, conservation biology, and animal behavior into “big data” sciences.

2.3 Introduction

To better understand the complexities of natural ecosystems and better manage and protect them, it would be helpful to have detailed, large-scale knowledge about the number, location, and behaviors of animal in natural ecosystems [19]. Placing motion sensor cameras called “camera traps” in natural habitats has transformed wildlife ecology and conservation over the last two decades [1]. These camera traps have become an essential tool for ecologists, enabling them to study population sizes and distributions [2], and evaluate habitat use [3]. While they can take millions of images [5-7], extracting knowledge from these camera-trap images is traditionally done by humans (i.e. experts or a community of volunteers) and is so time-consuming and costly that much of the valuable knowledge in these big data repositories remains untapped. For example, currently it takes 2-3 months for thousands of “citizen scientists” [8] to label each 6-month batch of images for the Snapshot Serengeti (hereafter, SS). By 2011, there were at least 125 camera-trap projects worldwide [5], and, as digital cameras become better and cheaper, more projects will put camera traps into action. Most of these projects, however, are not able to recruit and harness a huge volunteer force as SS has done to extract information of interest. Even if they are able to extract the
Figure 2.1: Deep neural networks can successfully identify, count, and describe animals in camera-trap images. Above the image: the ground-truth, human-provided answer (top line) and the prediction (second line) by a deep neural network we trained (ResNet-152). The three plots below the image, from left to right, show the neural network's prediction for the species, number, and behavior of the animals in the image. The horizontal color bars indicate how confident the neural network is about its predictions. All similar images in this paper are from the Snapshot Serengeti dataset [8].

Figure 2.2: Various factors make identifying animals in the wild hard even for humans (trained volunteers achieve 96.6% accuracy vs. experts).
information they originally intended to capture, there may be other important data that could be extracted for other studies that were not originally envisioned (e.g. information on non-focal animal species). Automating the information extraction procedure (Fig. 2.1) will thus make vast amounts of valuable information more easily available for ecologists to help them perform their scientific, management, and protection missions.

In this paper, we focus on harnessing computer vision to automatically extract the species, number, presence of young, and behavior (e.g. moving, resting, or eating) of animals, which are statistics that wildlife ecologists have previously decided are informative for ecological studies based on SS data [20–23]. These tasks can be challenging even for humans. Images taken from camera traps are rarely perfect, and many images contain animals that are far away, too close, or only partially visible (Fig. 2.2a-c). In addition, different lighting conditions, shadows, and weather can make the information extraction task even harder (Fig. 2.2d). Human-volunteer species and count labels are estimated to be 96.6% and 90.0% accurate, respectively, vs. labels provided by experts [8].

Automatic animal identification and counting could improve all biology missions that require identifying species and counting individuals, including animal monitoring and management, examining biodiversity, and population estimation [1]. In this paper, we harness deep learning, a state-of-the-art machine learning technology that has led to dramatic improvements in artificial intelligence in recent years, especially in computer vision [24]. Here, we do not harness the data we automatically extract to test a specific ecological hypothesis. Instead, we investigate the efficacy of deep learning to enable many future such studies by offering a far less expensive way to provide the data from large-scale camera-trap projects that has previously led to many informative ecological studies [20, 23].

Deep learning only works well with a lot of labeled data, significant computational resources, and modern neural network architectures. Here, we combine the millions of labeled data from the SS project, modern supercomputing, and state-of-the-art deep neural network (DNN) architectures to test how well deep learning can automate information extraction from camera-trap images. We find that the system is both able to perform as well as teams of human volunteers on a large fraction of the data, and identifies the few images that
require human evaluation. The net result is a system that dramatically improves our ability to automatically extract valuable knowledge from camera-trap images. Like every method, deep learning has biases (discussed below) that must be kept in mind, corrected, and or accounted for when using this technology. Swanson et al. 2016 [25] showed that the citizen-scientist approach also has its own set of systematic biases, but that they can be adequately corrected for.

2.4 Background and Related Work

2.4.1 Machine Learning

Machine learning enables computers to solve tasks without being explicitly programmed to solve them [26]. State-of-the-art methods teach machines via supervised learning i.e. by showing them correct pairs of inputs and outputs [27]. For example, when classifying images, the machine is trained with many pairs of images and their corresponding labels, where the image is the input and its correct label (e.g. “Buffalo”) is the output (Fig. 2.3).

2.4.2 Deep Learning

Deep learning [28] allows computers to automatically extract multiple levels of abstraction from raw data (Fig. 2.3). Inspired by the mammalian visual cortex [29], deep convolutional neural networks are a class of feedforward DNNs [28] in which each layer of neurons (to be “deep”, 3 or more layers) employs convolutional operations to extract information from overlapping small regions coming from the previous layers [24]. For classification, the final layer of a DNN is usually a softmax function, with an output between 0 and 1 per class, and with all of the class outputs summing to 1. These outputs are often interpreted as the DNN’s estimated probability of the image belonging in a certain class, and higher probabilities are often interpreted as the DNN being more confident that the image is of that class [30]. DNNs have dramatically improved the state of the art in many challenging problems [24], including speech recognition [31 33], machine translation [34 35], image recognition [36 37],
and playing Atari games.

Figure 2.3: Deep neural networks have several layers of abstraction that tend to gradually convert raw data into more abstract concepts. For example, raw pixels at the input layer are first processed to detect edges (first hidden layer), then corners and textures (second hidden layer), then object parts (third hidden layer), and so on if there are more layers, until a final prediction is made by the output layer. Note that which types of features are learned at each layer are not human-specified, but emerge automatically as the network learns how to solve a given task.

2.4.3 Related Work

There have been many attempts to automatically identify animals in camera-trap images; however, many relied on hand-designed features to detect animals, or were applied to small datasets (e.g. only a few thousand images). In contrast, in this work, we seek to (a) harness deep learning to automatically extract necessary features to detect, count, and describe animals; and (b) apply our method on the world’s largest dataset of wild animals i.e. the SS dataset. Reasons to learn features from raw data include that doing so often substantially improves performance, because such features can be transferred to other domains with small datasets, because it is time-consuming to manually design features, and because a general algorithm that learns features automatically can improve performance on very different types of data (e.g. sound, text), increasing the
impact of the approach. However, an additional benefit to deep learning is that if hand-designed features are thought to be useful, they can be included as well in case they improve performance [47–51].

Previous efforts to harness hand-designed features to classify animals include Swinnen et al. [7], who attempted to distinguish the camera-trap recordings that do not contain animals or the target species of interest by detecting the low-level pixel changes between frames. Yu et. al. [40] extracted the features with sparse coding spatial pyramid matching [52] and utilized a linear support vector machine [27] to classify the images. While achieving 82% accuracy, their technique requires manual cropping of the images, which requires substantial human effort.

Several recent works harnessed deep learning to classify camera-trap images. Chen et. al. [41] harnessed convolutional neural networks (CNNs) to fully automate animal identification. However, they demonstrated the techniques on a dataset of around 20,000 images and 20 classes, which is of much smaller scale than we explore here [41]. In addition, they obtained an accuracy of only 38%, which leaves much room for improvement. Interestingly, Chen et al. found that DNNs outperform a traditional Bag of Words technique [53, 54] if provided sufficient training data [41]. Similarly, Gomez et al. [55] also had success applying DNNs to distinguishing birds vs. mammals in a small dataset of 1,572 images and distinguish two mammal sets in a dataset of 2,597 images.

The closest work to ours is Gomez et al. [56], who also evaluate DNNs on the SS dataset: they perform only the species identification task, whereas we also attempt to count animals, describe their behavior, and identify the presence of young. On the species identification task, our models perform far superior to theirs: 92.0% for our best network vs. around 57% (estimating from their plot, as the exact accuracy was not reported) for their best network. There are multiple other differences between our work and theirs. (a) Gomez et al. only trained networks on a simplified version of the full 48-class SS dataset. Specifically, they removed the 22 classes that have the fewest images (Fig. 2.15, bottom 22 classes) from the full dataset and thus classify only 26 classes of animals. Here, we instead seek solutions that perform well on all 48 classes as the ultimate goal of our research is to automate as much of
Figure 2.4: While we train models on individual images, we only have labels for entire capture events, which we apply to all images in the event. When some images in an event have an animal and others are empty (as in this example), the empty images are labeled with an animal type, which introduces some noise in the training set labels and thus makes training harder.

The labeling effort as possible. (b) Gomez et al. base their classification solutions on networks pre-trained on the ImageNet dataset \[57\], a technique known as transfer learning \[43\]. We found that transfer learning made very little difference on this task when training on the full dataset (Sec. 2.5.11), and we thus chose not to use it for simplicity. We revisit the benefits of transfer learning on smaller datasets below. We conduct a more detailed comparison with Gomez et al. \[56\] in Sec. 2.5.9.

2.4.4 Snapshot Serengeti Project

The Snapshot Serengeti project is the world’s largest camera-trap project published to date, with 225 camera traps running continuously in Serengeti National Park, Tanzania, since 2011 \[8\]. Whenever a camera trap is triggered, such as by the movement of a nearby animal, the camera takes a set of pictures (usually 3). Each trigger is referred to as a *capture event*. The public dataset used in this paper contains 1.2 million capture events (3.2 million images) of 48 different species.

Nearly 28,000 registered and 40,000 unregistered volunteer citizen scientists have labeled 1.2 million SS capture events. For each image set, multiple users label the species, number of individuals, various behaviors (i.e. standing, resting, moving, eating, or interacting), and the presence of young. In total, 10.8 million classifications from volunteers have been recorded.
for the entire dataset. Swanson et al. \cite{8} developed a simple algorithm to aggregate these individual classifications into a final “consensus” set of labels, yielding a single classification for each image and a measure of agreement among individual answers. In this paper, we focus on capture events that contain only one species; we thus removed events containing more than one species from the dataset (1.2% of the events). Extending these techniques to images with multiple species is a fruitful area for future research. In addition to volunteer labels, for 3,800 capture events the SS dataset also contains expert-provided labels, but only of the number and type of species present.

75\% of the capture events were classified as empty of animals. Moreover, the dataset is very unbalanced, meaning that some species are much more frequent than others (Sec. 2.5.15). Such imbalance is problematic for machine learning techniques because they become heavily biased towards classes with more examples. If the model just predicts the frequent classes such as wildebeest or zebra most of the time, it can still get a very high accuracy without investing in learning rare classes, even though these can be of more scientific interest. The imbalance problem also exists for describing behavior and identifying the presence of young. Only 1.8\% of the capture events are labeled as containing babies; and only 0.5\% and 8.5\% of capture events are labeled as interacting and resting, respectively. We delve deeper into this problem in Sec. 2.5.15.

The volunteers labeled entire capture events (not individual images). While we do report results for labeling entire capture events (Sec. 2.5.13), in our main experiment, we focus on labeling individual images instead because if we ultimately can correctly label individual images it is easy to infer the labels for capture events. Importantly, we also found that utilizing individual images results in higher accuracy because it allows three times more labeled training examples (Sec. 2.5.13). In addition, training our system on images makes it more informative and useful for other projects, some of which are image-based and not capture-event-based.

However, the fact that we take the labels for each capture event and assign them to all the individual images in that event introduces noise into the training process. For example, a capture event may have one image with animals, but the remaining images empty (Fig.
Assigning a species label (e.g. hartebeest Fig. 2.4a) to all these images (Fig. 2.4b,c) adds some noise that machine learning models must overcome.

2.5 Experiments and Results

We investigate the feasibility of having computers automatically perform the laborious image labeling task currently performed by human volunteers or experts. We found that a two-stage pipeline outperforms a one-step pipeline (Sec. 2.5.7): in the first stage a network solves the empty vs. animal task (task I), i.e. detecting if an image contains an animal; in the second information extraction stage, a network then reports information about the images that contain animals. 75% of the images are labeled empty by humans, therefore automating the first stage alone saves 75% of human labor.

The information extraction stage contains three additional tasks: (II) identifying which species is present, (III) counting the number of animals, and (IV) describing additional animal attributes (their behavior and whether young are present). We chose to train one model to simultaneously perform all of these tasks, a technique —called multitask learning 58— because (a) these tasks are related, therefore they can share weights that encode features common to all tasks (e.g. recognizing animals); learning multiple, related tasks in parallel often improves the performance on each individual task 59, and (b) doing so requires fewer model parameters vs. a separate model for each task, meaning we can solve all tasks faster and more energy efficiently, and the model is easier to transmit and store. These advantages will become especially important if such neural network models run on remote camera traps to determine which pictures to store or transmit.

2.5.1 Datasets

In this paper, we only tackle identifying one instead of multiple species in an image (i.e. single-label classification 27). Therefore, we removed images that humans labeled as containing more than one species from our training and testing sets (approximately 5% of the dataset). The training and test sets for the information extraction stage are formed from
the 25% of images that are labeled as non-empty by humans.

If there are overly similar images in the training and test sets, models can just memorize the examples and then do not generalize well to dissimilar images. To avoid this problem, we put entire capture events (which contain similar images) into either the training or test set. From a total of 301,400 capture events that contained an animal, we created a training set containing 284,000 capture events, and two test sets. The expert-labeled test set contains 3,800 capture events with species and counts labels. The volunteer-labeled test set contains 17,400 capture events labeled by volunteers and it has labels for species, counts, behaviors, and the presence of young. The dataset contains images taken at day and at night, but we found this had little effect on performance (Sec. 2.5.10).

Table 2.1: In this paper, we employ different deep learning architectures to infer which one works the best and to be able to compare difference between accuracies come from different architectures.

<table>
<thead>
<tr>
<th>Architecture</th>
<th># of Layers</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>8</td>
<td>A landmark architecure for deep learning winning ILSVRC 2012 challenge [42].</td>
</tr>
<tr>
<td>NiN</td>
<td>16</td>
<td>Network in Network (NiN) is one of the first architectures harnessing innovative 1x1 convolutions [60] to provide more combinational power to the features of a convolutional layerters [60].</td>
</tr>
<tr>
<td>VGG</td>
<td>22</td>
<td>An architecture that is deeper (i.e. has more layers of neurons) and obtains better performance than AlexNet by employing effective 3x3 convolutional filters [37].</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>32</td>
<td>This architecture is designed to be computationally efficient (using 12 times fewer parameters than AlexNet) while offering high accuracy [10].</td>
</tr>
<tr>
<td>ResNet</td>
<td>18, 34, 50, 101, 152</td>
<td>The winning architecture of the 2016 ImageNet competition [36]. The number of layers for the ResNet architecture can be different. In this paper, we try 18, 34, 50, 101, and 152 layers.</td>
</tr>
</tbody>
</table>
Architectures

Different DNNs have different architectures, meaning the type of layers they contain (e.g. convolutional layers, fully connected layers, pooling layers, etc.), and the number, order, and size of those layers [24]. In this paper, we test 9 different modern architectures at or near the state of the art (Table 2.1) to find the highest-performing networks and to compare our results to those from Gomez et al. [56]. We only trained each model one time because doing so is computationally expensive and because both theoretical and empirical evidence suggests different DNNs trained with the same architecture, but initialized differently, often converge to similar performance levels [24,28,61].

A well-known method for further improving classification accuracy is to employ an ensemble of models at the same time and average their predictions. After training all the nine models for each stage, we form an ensemble of the trained models by averaging their predictions (Sec. 2.5.12). More details about the architectures, training methods, pre-processing steps and the hyperparameters are in Sec. 2.5.6. To enable other groups to replicate our findings and harness this technology for their own projects, we are publishing the software required to run our experiments as freely available, open-source code. We are also publishing the final, deep neural networks trained on SS so others can use them as is or for transfer learning. Both the code and the models can be accessed at https://github.com/Evolving-AI-Lab/deep_learning_for_camera_trap_images

2.5.2 Task I: Detecting Images That Contain Animals

For this task, our models take an image as input and output two probabilities describing whether the image has an animal or not (i.e. binary classification). We train 9 neural network models (Table 2.1). Because 75% of the SS dataset is labeled as empty, to avoid imbalance between the empty and non-empty classes, we take all 25% (757,000) non-empty images and randomly select 757,000 “empty” images. This dataset is then split it into training and test sets.

The training set contains 1.4 million images and the test set contains 105,000 images. Since the SS dataset contains labels for only capture events (not individual images), we
Table 2.2: Accuracy of different models on Task I: Detecting Images That Contain Animals

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Top-1 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>95.8%</td>
</tr>
<tr>
<td>NiN</td>
<td>96.0%</td>
</tr>
<tr>
<td><strong>VGG</strong></td>
<td><strong>96.8%</strong></td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>96.3%</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>96.3%</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>96.2%</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>96.3%</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>96.1%</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>96.1%</td>
</tr>
<tr>
<td>Ensemble of models</td>
<td>96.6%</td>
</tr>
</tbody>
</table>

assign the label of each capture event to all of the images in that event. All the architectures achieve a classification accuracy of over 95.8% on this task. The VGG model achieved the best accuracy of 96.8% (Table 2.2). To show the difficulty of the task and where the models currently fail, several examples for the best model (VGG) are shown in Sec. 2.5.8 and Fig. 2.17 shows the best model’s confusion matrix.

### 2.5.3 Task II: Identifying Species

For this task, the corresponding output layer produces the probabilities of the input image being one of the 48 possible species. As is traditional in the field of computer vision, we report top-1 accuracy (is the answer correct?) and top-5 accuracy (is the correct answer in the top-5 guesses by the network?). The latter is helpful in cases where multiple things appear in a picture, even if the ground-truth label in the dataset is only one of them. The top-5 score is also of particular interest in this work because artificial intelligence can be used to help humans label data faster (as opposed to fully automating the task). In that context, a human can be shown an image and the AI’s top-5 guesses. As we will report below, our best techniques identify the correct animal in the top-5 list 99.1% of the time. Providing such a list thus could save humans the effort of finding the correct species name in a list of 48 species over 99% of the time, although human user studies will be required to test that hypothesis.

Measured on the expert-labeled test set, the model ensemble has 94.9% top-1 and 99.1%
top-5 accuracy (Fig. 2.18 shows its confusion matrix), while the best single model (ResNet-152) obtains 93.8% top-1 and 98.8% top-5 accuracy (Fig. 2.5 top). The results on the volunteer-labeled test set along with several examples (like Fig. 2.1) are reported in Sec. 2.5.8.

Figure 2.5: Top: top-1 and top-5 accuracy of different models on the task of identifying the species of animal present in the image. Although the accuracy of all the models are similar, the ensemble of models is the best with 94.9% top-1 and 99.1% top-5 accuracy. Bottom: top-1 accuracy and the percent of predictions within +/-1 bin for counting animals in the images. Again, the ensemble of models is the best with 63.1% top-1 and 84.7% of the prediction within +/-1 bin.

2.5.4 Task III: Counting Animals

There are many different approaches for counting objects in images by deep learning [62–64] but nearly all of them require labels for bounding boxes around different objects in the image. Because this kind of information is not readily available in the SS dataset, we treat animal counting as a classification problem and leave more advanced methods for future work. In other words, instead of actually counting animals in the image, we assign the image to one of the 12 possible bins, each represents 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11-50, or +51 individuals respectively. For this task, in addition to reporting top-1 we also report the percent of images that are correctly classified within +/- 1 bin [8].

For this task, the ensemble of models on the expert-labeled test set gets 63.1% top-1
accuracy and 84.7% of predictions are within +/- 1 bin. Fig. 2.17 shows the ensemble’s confusion matrix. The same metrics for the best single model (ResNet-152) are 62.8% and 83.6% respectively (Fig. 2.5 bottom). The results on the volunteer-labeled test set along with several examples are reported in Sec. 2.5.8.

### 2.5.5 Task IV: Additional Attributes

The SS dataset contains labels for 6 additional attributes: standing, resting, moving, eating, interacting, and whether young are present (Fig. 2.1). Because these attributes are not mutually exclusive (especially for images containing multiple individuals), this task is a multi-label classification problem. A traditional approach for multi-label classification is to transform the task into a set of binary classification tasks. We do so by having, for each additional attribute, one two-neuron softmax output layer that predicts the probability of that behavior existing (or not) in the image.

The expert-labeled test set does not contain labels for these additional attributes, so we use the majority vote among the volunteer labels as the ground truth label for each attribute. We count an output correct if the prediction of the model for that attribute is higher than 50% and matches the ground-truth label.

We report traditional multi-label classification metrics, specifically, multi-label accuracy, precision, and recall. Pooled across all attributes, the ensemble of models produces 76.2% accuracy, 86.1% precision, and 81.1% recall. The same metrics for the best single model (ResNet-152) are 75.6%, 84.5%, and 80.9% respectively. More results for predicting additional attributes are reported in Sec. 2.5.8. For this and all previous tasks, we provide examples of correct predictions in Fig. 2.6 and incorrect network predictions in Fig. 2.7.

### 2.5.6 Pre-processing and Training

In this section, we document the technical details for the pre-processing step and for selecting the hyperparameters across all experiments in the paper.
Figure 2.6: Shown are nine images the ResNet-152 model labeled correctly. Above each image are a combination of expert-provided labels (for the species type and counts) and volunteer-provided labels (for additional attributes), as well as the model’s prediction for that image. Below each image are the top guesses of the model for different tasks, with the width of the color bars indicating the model’s output for each of the guesses, which can be interpreted as its confidence in that guess.
Figure 2.7: Shown are nine images the ResNet-152 model labeled incorrectly. Above each image are a combination of expert-provided labels (for the species type and counts) and volunteer-provided labels (for additional attributes), as well as the model’s prediction for that image. Below each image are the top guesses of the model for different tasks, with the width of the color bars indicating the model’s output for each of the guesses, which can be interpreted as its confidence in that guess. One can see why the images are difficult to get right. (g, i) contain examples of the noise caused by assigning the label for the capture camera, all images in these cases (also labeled) show the animal[s] being too far from the camera makes classification difficult.
Pre-processing

The original images in the dataset are 2,048×1,536 pixels, which is too large for current state-of-the-art deep neural networks owing to the increased computational costs of training and running DNNs on high-resolution images. We followed standard practices in scaling down the images to 256×256 pixels. Although this may distort the images slightly, since we do not preserve the aspect ratios of the images, it is a de facto standard in the deep learning community [24]. The images in the dataset are color images, where each pixel has three values: one for each of the red, green, and blue intensities. We refer to all the values for a specific color as a color channel. After scaling down the images, we computed the mean and standard deviation of pixel intensities for each color channel separately and then we normalized the images by subtracting the average and dividing by the standard deviation (Fig. 2.8). This step is known to make learning easier for neural networks [68,69].

Data Augmentation

We perform random cropping, horizontal flipping, brightness modification, and contrast modification to each image. Doing so, we provide an slightly different image each time, which can make the network resistant to small changes and improve the accuracy of the network [42].

Figure 2.8: An example of a camera-trap image in the SS dataset (left) and its down-sampled, normalized equivalent (upper right), which is what is actually input to the neural network.
Training

We train the networks via backpropagation using Stochastic Gradient Descent (SGD) optimization with momentum and weight decay [24]. We used the Torch [70] and Tensorflow [71] frameworks for our experiments. The SGD optimization algorithm requires several hyperparameters. The settings for those in our experiments are in Table 2.3. We train each model for 55 epochs with the learning-rate policy and the weight-decay policy that are shown in Table 2.4. We checkpoint the model after each epoch and at the end, we report the results of the most accurate model on the expert-labeled test set.

Table 2.3: The static neural network training hyperparameters for different experiments. The dagger symbol indicates a hyperparameter used when training the last layer only.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value (Train from scratch)</th>
<th>Value (Transfer learning)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Size</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Momentum</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Crop Size</td>
<td>224×224</td>
<td>224×224</td>
</tr>
<tr>
<td>Number of Epochs</td>
<td>55</td>
<td>40 (30†)</td>
</tr>
</tbody>
</table>

Table 2.4: The dynamic neural network training hyperparameters for all experiments.

<table>
<thead>
<tr>
<th>Method</th>
<th>Epoch Number</th>
<th>Learning Rate</th>
<th>Weight Decay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train from scratch</td>
<td>1-18</td>
<td>0.01</td>
<td>0.0005</td>
</tr>
<tr>
<td></td>
<td>19-29</td>
<td>0.005</td>
<td>0.0005</td>
</tr>
<tr>
<td></td>
<td>30-43</td>
<td>0.001</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>44-52</td>
<td>0.0005</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>53-55</td>
<td>0.0001</td>
<td>0</td>
</tr>
<tr>
<td>Transfer learning</td>
<td>1-5</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>5-10</td>
<td>0.001</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>10-20</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>20-27</td>
<td>0.001</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>27-35</td>
<td>0.0005</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>35-40</td>
<td>0.0001</td>
<td>0</td>
</tr>
<tr>
<td>Transfer learning (Last layer only)</td>
<td>1-5</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>5-10</td>
<td>0.001</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>10-25</td>
<td>0.0005</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>25-30</td>
<td>0.0001</td>
<td>0</td>
</tr>
</tbody>
</table>
2.5.7 One-stage Identification

In the main text, we employ a two-step pipeline for automatically processing the camera-trap images. The first step tries to filter out empty images and the second step provides information about the remaining images. One possibility is merging these two steps into just one step. We can consider the empty images as one of the identification classes and then train models to classify input images either as one of the species or the empty class. Although this approach results in a smaller total model size than having separate models for the first and second steps, there are three drawbacks to this approach. (a) Because ~75% of the images are empty images, this approach imposes a great deal of imbalance between the empty and other classes, which makes the problem harder for machine learning algorithms. (b) A one-step pipeline does not enable us to reuse an empty vs. animal module for other similar datasets. (c) We find out that one-step pipeline produces slightly worse results. In our experiment, to avoid the imbalance issue, we randomly select 220,000 empty images for the empty class, which is equal to the number of images for the most frequent class (wildebeest). Then we train four different architectures and measure their total accuracy, empty vs. animal accuracy, and species identification accuracy. The results are shown in Table 2.5.

Table 2.5: The results of the one-stage identification experiment. Although one-stage models do produce good results, their results are slightly worse than their corresponding two-stage comparator. For example, on task I: Detecting Images That Contain Animals, the one-step ResNet-50 model has 94.9% accuracy vs. 96.3% for the two-stage pipeline. For task II: Identifying Species the one-step ResNet-50 is 90.6% accurate with a one-step model vs. 93.6% for the two-stage pipeline.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Total Accuracy</th>
<th>Empty vs. Animal Accuracy</th>
<th>Animal Identification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>88.9%</td>
<td>93.7%</td>
<td>87.9%</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>90.5%</td>
<td>95.4%</td>
<td>89.5%</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>90.8%</td>
<td>94.7%</td>
<td>90.0%</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>91.3%</td>
<td>94.9%</td>
<td>90.6%</td>
</tr>
</tbody>
</table>
2.5.8 Results on the Volunteer-Labeled Test Set

As mentioned in the main text, the volunteer-labeled test set has 17,400 capture events labeled by human volunteers. It has labels for species, counts, descriptions of animal behaviors, and whether young are present. In the main paper we compared our model predictions to expert-provided labels; in this section we compare instead to the volunteer-provided labels. Fig. 2.9 shows the results. For task II: Identifying Species, all the models have top-1 accuracy over 89.2% and top-5 accuracy over 97.5%. For task III: Counting Animals, all models have top-1 accuracy over 62.7% and all of them can count within one bin for over 84.2% of the test examples.

For task IV: Additional Attributes, the models have over 71.3% accuracy, 82.1% precision, and 77.3% recall. The ensemble of models performs the best for the description task by a small margin. Overall, for all the tasks, the results of different architectures are similar. Moreover, our models predictions are closer to those of the experts on some tasks (e.g. animal identification), and closer to human-volunteers on others (e.g. counting), for reasons that are not clear.

2.5.9 Comparing to Gomez et al. 2016

In the closest work to ours, Gomez et al. [56] employed transfer learning [43, 72], which is a way to learn a new task by utilizing knowledge from an already learned, related task. In particular, they used models pre-trained on the ImageNet dataset, which contains 1.3 million images from 1,000 classes of man-made and natural images [57] to extract features and then, on top of these high-level features, trained a linear classifier to classify animal species. They tested six different architectures: AlexNet [42], VGG [37], GoogLeNet [10], ResNet-50 [36], ResNet-101 [36], and ResNet-152 [36]. To improve the results for two of these architectures, they also further trained the entire AlexNet and GoogLeNet models on the SS dataset (a technique called fine-tuning [24, 43, 72]).

To avoid dealing with an unbalanced dataset, Gomez et al. [56] removed all species classes that had a small number of images and classified only 26 out of the total 48 SS classes. Because we want to compare our results to theirs and since the exact dataset used
Figure 2.9: The results of task II: Identifying Species, task III: Counting Animals, and task IV: Additional Attributes on the volunteer-labeled test set. The top plot shows top-1 and top-5 accuracy of different models for the task of identifying animal species. The ensemble of models is the best with 92.5% top-1 accuracy and 98.4% top-5 accuracy. The middle plot shows top-1 accuracy and the percent of predictions within 1 bin for counting animals in the images. The ensemble of models has the best top-1 accuracy with 67.9% and ResNet-152 has the closest predictions with 88.4% of the prediction within 1 bin. The bottom plot shows accuracy for the task of describing additional attributes (behaviors and the presence of young). The ensemble of models is the best with 76.2% accuracy, 86.1% precision, and 81.1% recall.

in [56] is not publicly available, we did our best to reproduce it by including all images from those 26 classes. We call this dataset SS-26. We split 93% of the images in SS-26 into the training set and place the remaining 7% into the test set (the training vs. test split was not reported in Gomez et al. [56]).

Because we found transfer learning from ImageNet not to help on identifying animals in the SS dataset (Sec. 2.5.11), we train our networks from scratch on the SS-26 dataset. We train the same set of network architectures (with just one output layer for the identification task) as in Gomez et al. [56] on the SS-26 dataset. For all networks, we obtained substantially higher accuracy scores than those reported in [56] (Fig. 2.10): our best network obtains a top-1 accuracy of 92.0% compared to ~57% by Gomez et al. (estimating from their plot, as the exact accuracy was not reported). It is not clear why the performance of Gomez et
al. [56] is lower.

In another experiment, Gomez et al. [56] obtained a higher accuracy of 88.9%, but on another heavily simplified version of the SS dataset. This modified dataset contains only \( \sim 33,000 \) images and the images were manually cropped and specifically chosen to have animals in the foreground [56]. We instead seek deep learning solutions that perform well on the full SS dataset and without manual intervention.

![Figure 2.10](image)

Figure 2.10: For the experiment classifying the 26 most common species, shown is the top-1 and top-5 accuracy from Gomez et al. [56] and for the different architectures we tested. Our models yield significantly better results. On average, top-1 and top-5 accuracies are improved over 30%. The ResNet-50 model achieved the best top-1 result with 92% accuracy. Because Gomez et al. [56] did not report exact accuracy numbers, the numbers used to generate this plot are estimated from their plot.

### 2.5.10 Day vs. Night Accuracy

The SS dataset contains images taken during the day and night. We investigated whether the deep learning system performed better during the day. Based on the timestamp and location (latitude and longitude) where each image was taken, we computed whether the sun was six degrees or more below the horizon. If it was, we defined the image as a nighttime image, and otherwise as a daytime image.
From 11,502 images in the expert-labeled test set, 10,839 of them were taken during the day and 663 of them were taken at night. For the species identification task, our ensemble of classifiers obtained 94.9% top-1 and 99.1% top-5 accuracy for day images and 94.6% top-1 and 99.2% top-5 accuracy for night images. For the counting task, the ensemble of classifiers had 62.5% top-1 accuracy and 84.3% of the predictions were within 1 bin for daytime images. For nighttime images, the ensemble had 70.9% top-1 accuracy and 90.3% of the predictions were within 1 bin. Overall, the results reveal little performance difference between day and night, and even an increase in performance for counting at night.

2.5.11 Helping Small Camera-Trap Projects Via Transfer Learning

Deep learning works best with many (e.g. millions) labeled data points (here, images) [24]. Many small camera-trap projects exist that do not have the ability to label a large set of images. Deep learning can still benefit such projects through transfer learning [43, 44, 73], wherein a network can first be trained on images available in other large datasets (e.g. large, public datasets like Snapshot Serengeti) and then further trained on a different, smaller dataset (e.g. a small camera-trap project with just a few thousand labeled images). The knowledge learned on the first dataset is thus repurposed to classify the second, smaller dataset. We conducted experiments validating this approach for the identifying species task (Task I & Task II), which gives a sense of how well smaller projects can expect to do with various amounts of labeled data.

Transfer learning [43, 74] takes advantage of the knowledge gained from learning on one task and applies it to a different, related task. Our implementation of transfer learning follows methods from previous work in the image recognition field [73, 75, 76]. We first pre-train the AlexNet and ResNet-152 architectures on the ImageNet dataset [57]. These pre-trained models then become the starting point (i.e. initial weights) for further training the models on the SS dataset.

We first test whether transfer learning helps when the full SS dataset is available to train on. The static and dynamic hyperparameters for these runs are the same as in the
original experiment (Sec. 2.5.6). All transfer learning experiments use the test set with expert-provided labels. At the end of transfer learning, for task II: Identifying Species, the AlexNet model has 92.4% top-1 accuracy and 98.8% top-5 accuracy, while the ResNet-152 model has 93.0% top-1 accuracy and 98.7% top-5 accuracy. For task III: Counting Animals, AlexNet and ResNet-152 are 59.1% and 62.4% top-1 accurate and 80.7% and 82.6% of their predictions are only 1 bin off, respectively. Comparing the obtained results to those in Fig. 5 in the paper indicates that transfer learning from ImageNet does not help to increase accuracy when training with the full SS dataset (at least with these hyperparameters).

We also tested whether transfer learning would improve performance when fewer labeled images are available, simulating the reality for many smaller camera-trap projects. For these experiments, we tried two different transfer learning techniques: further training the entire network on the target (new, smaller) dataset, and only further training the last layer of the network (to harness features learned on the larger dataset without potentially corrupting them via overfitting, which is more likely to happen when the target dataset is very small). Even when training the entire network, we trained the last layer only for the first 10 epochs (passes through the dataset) to prevent unhelpful gradients produced by the initially random last layer from corrupting features learned at earlier layers. We found that the method of training the last layer only was helpful only on task I and when small amounts of labeled data are available for that task ($\leq$ 4000 labels), so we only used that approach in those cases.

Because we are not aware of any other publicly available labeled camera-trap datasets, to conduct this experiment we simulate camera-trap projects of various sizes by randomly creating labeled datasets of different sizes from SS data. To conduct transfer learning, we first train on the ImageNet dataset and then further train the network on a small simulated camera-trap dataset. ImageNet has 1.2M labeled images for 1000 categories (from synthetic objects such as bicycles and cars to wildlife categories such as dogs and lions). This dataset is commonly used in computer vision research, including research into transfer learning. Training on images from the real world can be helpful even if the classes of images are dissimilar because many lower-level image features (e.g. edge detectors of different orientations, textures, shapes, etc.) are common across very different types of

32
images \cite{43, 44, 73}. That said, transfer learning from the ImageNet dataset to SS likely underestimates what performance is possible with transfer learning between camera-trap specific datasets, because it has been shown that the more similar the classes of images are between the transfer-from and transfer-to datasets, the better transfer learning works \cite{43}. Transferring from the SS dataset to other wildlife camera-trap projects could thus provide even better performance.

The main takeaway is that a substantial fraction of the data can be automatically extracted at the same 96.6% accuracy level of citizen-scientists even if only a few thousand labeled images are available. Accuracy, and thus automation percentages, further improve as more labeled data are provided during training. With 1.5k images, over 41% of the entire pipeline can be automated. Assuming a conservative 10s per image, labeling these 1.5k images takes only 4.2h. With only 3k images (8.3h), that number jumps to over 50%. With 6k, 10k, and 15k images (16.7h, 27.8h, and 41.7h), 62.6%, 71.4%, and 83.0% of the data can be automatically labeled, respectively. With 50k images (138.9h), 91.4% of the entire pipeline can be automated. Thus, sizable cost savings are available to small camera-trap projects of various sizes, and, especially at the low end, investing in labeling a few more thousand images can provide substantial performance improvements.

Because we do not know what the ratio of empty images vs. full (those with animals) will be for any particular camera-trap project (it will depend on the cameras, their software, and the ecosystem), for each dataset size ($N$ total images) we create many simulated datasets with different ratios and report the mean performance across all of them. Specifically, we create datasets with $E$ empty images and $F$ full images (all randomly selected), for $E, F \in \{1k, 2k, \ldots, 9k, 10k, 20k, \ldots, 100k\}$. For each dataset size $N$ (the rows in Table 2.6), we then average performance over all combinations of $E$ and $F$ that sum to $N$, but where $F \geq E$ (e.g. the $N=50k$ total training image row shows performances averaged over the following two simulated datasets: $E=10k$, $F=40k$ and $E=20k$, $F=30k$). We do not create datasets with more empty images than full images to preserve data balance when training on task I (should a project have more empty images than images with animals, these extra empty images could be discarded for training, which would increase the number of labeled training
images required by that amount). For each created dataset, for task I we train on all $E$ empty images and an equal number of $E$ randomly selected full images to preserve data balance. For task II we train on all of the available $F$ full images. Tables 2.3 and 2.4 list the hyperparameters for these experiments. We did not include the images required to test the models in the budget because each camera trap project is free to choose the test set size to find the best tradeoff between the cost of labeling data and the variance of their accuracy estimate, and for some choices the test set size could be quite small. Due to the number of transfer experiments, for task II in them we use the highest-performing single model (ResNet-152) instead of the ensemble of models (which would have multiplied the number of models required to be trained by 9).

The results are provided in Table 2.6. As mentioned in the main text, even when few labeled examples are available, a sizable amount of the data can be automatically extracted at the same 96.6% accuracy level of human citizen scientists. For these smaller dataset sizes, transfer learning provided a substantial performance improvement over training from scratch.

### 2.5.12 Prediction Averaging

For each image, a model outputs a probability distribution over all classes. For each class, we average the probabilities from the $m$ models, and then either take the top class or top $n$ classes in terms of highest average(s) as the prediction(s). Table 2.7 shows an example.

### 2.5.13 Classifying Capture Events

The SS dataset contain labels for *capture events*, not individual images. However, our DNNs are trained to classify images. We can aggregate the predictions for individual images to predict the labels for entire capture events. One could also simply train a neural network to directly classify capture events. We try both of these approaches and report the results here.

To implement the former, we employ the same prediction averaging method as in Sec. Prediction Averaging except that in this case the classifications come from the same
The next experiment we tried was inputting all images from a capture event at the same time and asking the model to provide one label for the entire capture event. For computational reasons, we train only one of our high-performing models (ResNet-50). Because feedforward neural networks have a fixed number of inputs, we only consider capture events that contain exactly three images and we ignore the other 55,000 capture events. We put the three images from a capture event on top of each other and form a 9-channel input image for the model. On the expert-labeled dataset, the model achieved 90.8% top-1 accuracy and 97.4% top-5 accuracy for identification and 58.5% top-1 accuracy and 81.1% predictions within 1 bins for counting. Both scores are slightly below our results for any of the models trained on individual images. These results and those from the previous experiment suggest that training on individual images is quite effective and produces more accurate results.

There are other reasons to prefer classifying single images. Doing so avoids (a) the challenge of dealing with capture events with different numbers of images, (b) making the number of labeled training examples smaller (which happens when images are merged into capture events), (c) the larger neural network sizes required to process many images at once, and (d) choices regarding how best to input all images at the same time to a feedforward neural network. Overall, investigating the best way to harness the extra information in multi-image capture events, and to what extent doing so is helpful vs. classifying individual images, is a promising area of future research.

### 2.5.14 Saving Human Labor via Confidence Thresholding

One main benefit of automating information extraction is eliminating the need for humans to have to label images. Here we estimate the total amount of human labor that can be saved if our system is designed to match the accuracy of human volunteers.
Figure 2.11: The top-1 and top-5 accuracy of different architectures for entire capture events (as opposed to individual images) on the expert-labeled test set. Combining the classification for all the images within a capture event improves accuracy for all the models. The best accuracy belongs to the ensemble of models with 95.5% top-1 accuracy and 99.4% top-5 accuracy.

We create a two-stage pipeline by having the VGG model from the empty vs. animal experiment classify whether the image contains an animal and, if it does, having the ensemble of models from the second stage label it. We can ensure the entire pipeline is as accurate as human volunteers by having the network classify images only if it is sufficiently confident in its prediction.

The output probabilities per class (i.e. predictions) by deep neural networks can be interpreted as the confidence of the network in that prediction [30]. We can take advantage of these confidence measures to build a more accurate and more reliable system by automatically processing only those images that the networks are confident about and asking humans to label the rest. We threshold at different confidence levels, which results in the network classifying different amounts of data, and calculate the accuracy on that restricted dataset. We do so for task I: Detecting Images That Contain Animals (Fig. 2.12), task II: Identifying Species (Fig. 2.13), and task III: Counting Animals (Fig. 2.14). As mentioned above, we cannot perform this exercise for task IV: Additional Attributes because SS lacks expert-
Harnessing this confidence thresholding mechanism, we can design a system that matches the volunteer human classification accuracy of 96.6%. For Task I: Detecting Images That Contain Animals, we do not have expert-provided labels and thus do not know the accuracy of the human volunteers, so we assumed it to be the same 96.6% accuracy as on the animal identification task (Task II). Because the VGG model’s accuracy is higher than the volunteers we can automatically process 75% of the data (because 75% of the images are empty) at human-level accuracy. For Task II: Identifying Species, thresholding at 43% confidence enables us to automatically process 97.2% of the remaining 25% of the data at human-level accuracy. Therefore, our fully automated system operates at 96.6% accuracy on $75\% \times 100\% + 25\% \times 97.2\% = 99.3\%$ of the data. Applying the same procedure to Task III: Counting Animals, human volunteers are 90.0% accurate and to match them we can threshold at 79%. As a result, we can automatically count 44.55% of the non-empty images and therefore $75\% \times 100\% + 25\% \times 44.5\% = 86.1\%$ of the data. We cannot perform this exercise for Task IV: Additional Attributes because SS lacks expert-provided labels for this task, meaning human-volunteer accuracy on it is unknown.

Note that to manually label $\sim$5.5 million images, nearly 30,000 SS volunteers have donated $\sim$14.6 years of 40-hour-a-week effort [8]. Based on these statistics, our current automatic identification system saves an estimated 8.4 years of 40-hour-per-week human labeling effort (over 17,000 hours) for 99.3% of the 3.2 million images in our dataset. Such effort could be reallocated to harder images or harder problems or might enable camera-trap projects that are not able to recruit as many volunteers as the famous SS project with its charismatic megafauna.

### 2.5.15 Improving Accuracy for Rare Classes

As previously mentioned, the SS dataset is heavily imbalanced. In other words, the numbers of available capture events (and thus pictures) for each species are very different (Fig. 2.15). For example, there are more than 100,000 wildebeest capture events, but only 17 zorilla capture events. In particular, 63% of capture events contain wildebeests, zebras, and Thomson’s
Figure 2.12: To increase the reliability of our model we can filter out the images that the network is not confident about and let experts label them instead. Here we report the accuracy (top panel) of our VGG model on the images that are given confidence scores ≥ the thresholds (x-axis) for task I: Detecting Images That Contain Animals. **Top:** The top-1 accuracy of the VGG model when we filter out images at different confidence levels (x-axis). **Bottom:** The percent of the dataset that remains when we filter out images for which that same model has low confidence. If we only keep the images that the model is 99% or more confident about, then we can have a system with 99.8% accuracy for 76% of the data (rightmost column).

gazelle. Imbalance has been shown to negatively affect the performance of human citizen-scientists on rare species [25]. Imbalance can also produce pathological machine learning models because they can limit their predictions to the most frequent classes and still achieve a high level of accuracy. For example, if our model just learns to classify wildebeests, zebras, and Thomson’s gazelle, still it can achieve 63% accuracy while ignoring the remaining 94% of classes. Experimental results show that our models obtain extremely low accuracy on rare classes (i.e. the classes with only few training examples) (Fig. 2.16, bottom classes in the leftmost column have as low as ∼0% accuracy scores). To ameliorate the problem caused by imbalance, we try three methods which we describe in the following subsections. All the following experiments are performed on the volunteer-labeled test set for the ResNet-152
Figure 2.13: The figures are plotted in the same way as Fig. 2.12, but here for the ensemble of models for task II: Identifying Species. If we only keep the images that the model is 99% or more confident about, we have a system that performs at 99.8% top-1 accuracy on 66.1% of the data (the rightmost column). **Top:** The top-1 (red) and top-5 (blue) accuracy of the ensemble of models when we filter out images with different confidence levels (x-axis).

model (which had the best top-1 accuracy on classifying all 48 SS species).

**Weighted Loss**

For classification tasks, the measure of performance (i.e. accuracy) is defined as the proportion of examples that the models correctly classifies. In normal conditions, the cost associated with missing an example is equal for all classes. One method to deal with imbalance in the dataset is to put more cost on missing examples from rare classes and less cost for missing examples of the frequent classes, which we will refer to as the *weighted loss* approach [78]. For this approach, we have a weight for each class indicating the cost of missing examples from that class. To compute the weights, we divide the total number $N$ of examples in the set by the total number of examples $n_i$ from each class $i$ in the training set. Then, we calculate the associated weights for each class using Eq. 2.1 and 2.2. Because the dataset is
highly imbalanced, we would have some very large class weights and some very small class weights for our method. Our models are trained by the backpropagation algorithm which computes the gradients over the network. These extreme weights result in very small or very large gradients, which can be harmful to the learning process. A quick remedy for this problem is to clamping the gradients within a certain range. In our experiments, we clamped the gradients of the output layer in the $[-0.01, 0.01]$ range.

$$f_i = \frac{N}{n_i} \quad (2.1)$$

$$w_i = \frac{f_i}{\sum_{i=1}^{48} f_i} \quad (2.2)$$

The obtained results of this experiment (Fig. 2.16, middle-left column) show that applying this method can increase the accuracy for the rare classes while keeping the same
level of accuracy for most of the other classes. This method is especially beneficial for genet (40% improvement) and aardwolf (35% improvement). Applying the weighted loss method slightly hurts the top-1 accuracy, but it improved top-5 accuracy. The results suggest the weighted loss method is an effective way for dealing with imbalance in dataset.

**Oversampling**

Another method for dealing with dataset imbalance is *oversampling* [78], which means feeding examples from rare classes more often to the model during training. This means that, for example, we show each sample in the zebra class only once to the model whereas we show the samples from the zorilla class around 4,300 times in order to make sure that the network sees an equal number of samples per class. The results from this experiment (Fig. 2.16, middle-right column) show that the oversampling technique boosted the classification accuracy for rhinoceros (∼80%) and zorilla (40%) classes. We empirically found oversampling to slightly hurt the overall performance more than the other two methods (Fig. 2.16, the overall top-1 and top-5 accuracy are lower than those of the baseline, weighted loss and emphasis sampling methods). Further investigation is required to fully explain this phenomenon.

**Emphasis Sampling**

Another method for solving the imbalance issue, which can be considered as an enhanced version of oversampling is *emphasis sampling*. In emphasis sampling, we give another chance to the samples that the network fails on: the probability of samples being fed again to the network is increased whenever the network misclassifies them. Thus if the network frequently misclassifies the examples from rare classes it will be more likely to retrain on them repeatedly, allowing the model to make more changes to try to learn them.

For implementing the emphasis sampling method, we considered two queues, one for the examples that the top-1 guess of the network is not correct and one for the examples that all the top-5 guesses of the network are incorrect about. Whenever the model misclassifies an example, we put that example in the appropriate queue. During the training process, after feeding each batch of examples to the network, we feed another batch of examples taken
from the front of the queues to the model with probability of 0.20 for the first queue and 0.35 for the second queue. Doing so, we increase the chance of wrongly classified images to be presented to the network more often.

The results from this experiment (Fig. 2.16 right-most column) indicate that this method can increase the accuracy for some of the rare classes, such civet (∼40%) and rhinoceros (∼40%). Moreover, emphasis sampling improved top-5 accuracy for the dataset in overall.

Overall

We found that all three methods perform similarly and can improve accuracy for some rare classes. However, they do not improve the accuracy for all the rare classes. More future research is required to further improve these methods.

2.5.16 Validating Against the NACTI Dataset

In order to validate our method and confirm its usefulness for other datasets, we trained the ResNet-18 model on the North America Camera Trap Images (NACTI) Dataset. This was done in collaboration with US Department of Agriculture National Wildlife Research Center (USDA NWRC). In a departure from our previous work, to assess the model’s accuracy for each species present at each study site, we altered the ratio of testing to training images for each species based on the number of available images for that species. Specifically, if less than 10 labeled images were available for a species at a site, we used all the images for testing and none for training. If there were 10-30 images for a species, then we use 50% for training and 50% for testing. If a species possessed more than 30 images, then the ratio from our prior work of 90% training and 10% testing was implemented. The trained model performed well, achieving over 97.5% top-1 accuracy in identifying the correct species. Lastly, we also built a single-species detection model for wild pigs which had the best accuracy of any of our models (98.6% top-1 accuracy; Pig/no pig).

We also performed out-of-sample assessment by using the trained model on NACTI to label images of ungulates from Canada. For species identification task, we obtained 81.8%
top-1 accuracy with 90.9% top-5 accuracy. We also tested if our trained model on NACTI can accurately predict the presence or absence of an animal in the SS images. 85.1% of the empty images were correctly classified as empty, and 94.3% of images containing an animal were correctly classified as non-empty.

2.5.17 Analysis of Visual Features Learned by Machines to Classify Species

One crucial question which comes to mind is how deep neural networks classify the species. In other words, what features do they learn for each species to be able to distinguish them from other species. To shed light on the answer to this question, I collaborated with a research group at UC Berkeley to visualize the most salient pixels in a picture that determine the predictions of the network. We applied a gradient-weighted class-activation-mapping (Grad-CAM) procedure [79] to obtain the pixels with the highest level of activation in the last convolutional layer of the AlexNet architecture. Our results demonstrated that the extracted pixels often form visual features similar to those humans use to identify the same species (Fig. 2.20).

To show the regions of images that affect the predictions the most, we also discovered the neurons in the last convolution layer that frequently respond to images of a particular species. For each neuron, we then cropped image patches around the pixel with the highest stimuli for that neuron. We found that DNNs learn to associate some background pixels with particular species. This fact was one of our main motivations for designing the pipeline which is presented in chapter 3. Fig. 2.21 shows sample image patches that correspond to the top-5 most responsive neurons for several species.

2.6 Discussion and Future Work

There are many directions for future work, but here we mention the three most promising ones. The first is studying the actual time savings and effects on accuracy of a system hybridizing deep neural networks and teams of human volunteer labelers. Time savings
should come from three sources: automatically filtering empty images, accepting automatically extracted information from images for which the network is highly confident in, and by providing human labelers with a sorted list of suggestions from the model so they can quickly select the correct species, counts, and descriptions. However, the actual gains seen in practice need to be quantified. Additionally, the effect of such a hybrid system on human accuracy needs to be studied. Accuracy could be hurt if humans are more likely to accept incorrect suggestions from deep neural networks, but could also be improved if the model suggests information that humans may not have thought to consider. A second, but related, promising direction is studying active learning [81, 82], a virtuous cycle in which humans label only the images the network is not confident in, and then those images are added to the dataset, the network is retrained, and the process repeats. The third is automatically handling multi-species images, which we removed for simplicity. While our current trained pipeline can be applied to all images, for images with multiple species it provides only one species label. 97.5% of the time it correctly lists one of the species present, providing useful information, but the impact of missing the other species should be kept in mind and will depend on the use case. However, one could train networks to list multiple species via a variety of more sophisticated deep learning techniques [58, 83, 84], a profitable area for future research.

2.7 Conclusions

In this paper, we tested the ability of state-of-the-art computer vision methods called deep neural networks to automatically extract information from images in the SS dataset, the largest existing labeled dataset of wild animals. We first showed that deep neural networks can perform well on the SS dataset, although performance is worse for rare classes. Perhaps most importantly, our results show that employing deep learning technology can save a tremendous amount of time for biology researchers and the human volunteers that help them by labeling images. In particular, for animal identification, our system can save 99.3% of the manual labor (over 17,000 hours) while performing at the same 96.6% accuracy level of human volunteers. This substantial amount of human labor can be redirected to other
important scientific purposes and also makes knowledge extraction feasible for camera-trap projects that cannot recruit large armies of human volunteers. Automating data extraction can thus dramatically reduce the cost to gather valuable information from wild habitats, and will thus likely enable, catalyze, and improve many future studies of animal behavior, ecosystem dynamics, and wildlife conservation.
Table 2.6: Transfer learning can enable deep learning to perform well even when few labeled training examples are available, allowing a sizable percent of the data to be automatically processed at the same 96.6% accuracy level of human citizen scientists. For each dataset size (left column), we compare training a network from scratch on each dataset vs. transfer learning (see main text Sec. Helping Small Camera-Trap Projects Via Transfer Learning). Note that due to the large number of networks that needed to be trained for these experiments, for task II we used the single highest-performing model (ResNet-152) instead of the ensemble of 9 models, which explains why the performance on the entire dataset is lower than the best results reported elsewhere in this paper. Sec. 2.5.11 describes additional details for these experiments.

<table>
<thead>
<tr>
<th># Total training images</th>
<th>Accuracy and automation percent for task I: Detecting Images That Contain Animals</th>
<th>Accuracy and automation percent for task II: Identifying Species</th>
<th>Automation percent of the full pipeline (task I &amp; task II)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train from scratch</td>
<td>Transfer learning</td>
<td>Train from scratch</td>
</tr>
<tr>
<td>1,500</td>
<td>54.5, 0.0</td>
<td>82.5, 42.1</td>
<td>38.9, 0.0</td>
</tr>
<tr>
<td>2,000</td>
<td>58.9, 0.0</td>
<td>84.7, 51.5</td>
<td>40.2, 0.2</td>
</tr>
<tr>
<td>3,000</td>
<td>58.9, 0.0</td>
<td>84.7, 51.5</td>
<td>40.2, 0.2</td>
</tr>
<tr>
<td>4,000</td>
<td>54.3, 0.0</td>
<td>84.7, 51.5</td>
<td>41.0, 0.2</td>
</tr>
<tr>
<td>5,000</td>
<td>54.3, 0.0</td>
<td>84.7, 51.5</td>
<td>41.1, 0.2</td>
</tr>
<tr>
<td>6,000</td>
<td>53.1, 0.0</td>
<td>86.9, 62.6</td>
<td>42.7, 0.2</td>
</tr>
<tr>
<td>7,000</td>
<td>53.1, 0.0</td>
<td>86.9, 62.6</td>
<td>43.1, 0.1</td>
</tr>
<tr>
<td>8,000</td>
<td>52.4, 0.0</td>
<td>88.1, 68.7</td>
<td>44.1, 0.1</td>
</tr>
<tr>
<td>9,000</td>
<td>52.4, 0.0</td>
<td>88.1, 68.7</td>
<td>44.9, 0.1</td>
</tr>
<tr>
<td>10,000</td>
<td>53.6, 0.0</td>
<td>88.9, 72.1</td>
<td>46.7, 1.0</td>
</tr>
<tr>
<td>15,000</td>
<td>58.2, 0.0</td>
<td>92.1, 85.9</td>
<td>46.7, 1.0</td>
</tr>
<tr>
<td>20,000</td>
<td>63.9, 0.0</td>
<td>93.5, 91.5</td>
<td>62.6, 25.0</td>
</tr>
<tr>
<td>25,000</td>
<td>58.2, 0.0</td>
<td>92.1, 85.9</td>
<td>62.6, 25.0</td>
</tr>
<tr>
<td>30,000</td>
<td>63.9, 0.0</td>
<td>93.5, 91.5</td>
<td>68.1, 37.1</td>
</tr>
<tr>
<td>35,000</td>
<td>59.5, 0.0</td>
<td>92.8, 90.5</td>
<td>75.8, 53.8</td>
</tr>
<tr>
<td>40,000</td>
<td>63.6, 0.0</td>
<td>93.7, 93.5</td>
<td>75.8, 53.8</td>
</tr>
<tr>
<td>45,000</td>
<td>59.5, 0.0</td>
<td>92.8, 90.5</td>
<td>75.8, 53.8</td>
</tr>
<tr>
<td>50,000</td>
<td>63.6, 0.0</td>
<td>93.7, 93.5</td>
<td>77.5, 57.4</td>
</tr>
<tr>
<td>60,000</td>
<td>62.5, 0.0</td>
<td>93.7, 94.4</td>
<td>80.4, 63.8</td>
</tr>
<tr>
<td>70,000</td>
<td>62.5, 0.0</td>
<td>93.7, 94.4</td>
<td>81.3, 65.9</td>
</tr>
<tr>
<td>80,000</td>
<td>70.4, 24.0</td>
<td>94.0, 94.8</td>
<td>83.3, 70.4</td>
</tr>
<tr>
<td>90,000</td>
<td>70.4, 24.0</td>
<td>94.0, 94.8</td>
<td>84.0, 72.0</td>
</tr>
<tr>
<td>100,000</td>
<td>75.2, 38.6</td>
<td>94.3, 95.3</td>
<td>93.8, 96.1</td>
</tr>
<tr>
<td>All images</td>
<td>96.8, 100</td>
<td>96.6, 100</td>
<td>93.8, 96.1</td>
</tr>
</tbody>
</table>

46
Table 2.7: An example of classification averaging. The numbers are the probability the network estimates the input was of that class, which can also be interpreted as the network’s confidence in its prediction. For all classes (e.g. species in this example), we average these confidence scores across all the models. The final aggregate prediction is the class with the highest average probability (or the top $n$ if calculating top-$n$ accuracy). Due to space constraints, we show the top 7 species (in order) in terms of average probability.

<table>
<thead>
<tr>
<th>Species</th>
<th>Network 1</th>
<th>Network 2</th>
<th>Network 3</th>
<th>Average Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zebra</td>
<td>0.80</td>
<td>0.05</td>
<td>0.50</td>
<td>$(0.80+0.05+0.50)/3= 0.45$</td>
</tr>
<tr>
<td>Impala</td>
<td>0.00</td>
<td>0.90</td>
<td>0.08</td>
<td>$(0.00+0.90+0.08)/3= 0.33$</td>
</tr>
<tr>
<td>Topi</td>
<td>0.10</td>
<td>0.00</td>
<td>0.40</td>
<td>$(0.10+0.00+0.40)/3= 0.17$</td>
</tr>
<tr>
<td>Dikdik</td>
<td>0.07</td>
<td>0.04</td>
<td>0.00</td>
<td>$(0.07+0.04+0.00)/3= 0.04$</td>
</tr>
<tr>
<td>Reedbuck</td>
<td>0.03</td>
<td>0.00</td>
<td>0.02</td>
<td>$(0.03+0.00+0.02)/3= 0.02$</td>
</tr>
<tr>
<td>Gazelle Grants</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>$(0.00+0.01+0.00)/3= 0.00$</td>
</tr>
<tr>
<td>Eland</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>$(0.00+0.00+0.00)/3= 0.00$</td>
</tr>
</tbody>
</table>

Table 2.8: The accuracy on capture events (as opposed to individual images) of models for task I: Detecting Images That Contain Animals.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Top-1 accuracy for capture events</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>96.3%</td>
</tr>
<tr>
<td>NiN</td>
<td>96.6%</td>
</tr>
<tr>
<td>VGG</td>
<td>96.8%</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>96.9%</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>96.8%</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>96.8%</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>97.1%</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>96.8%</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>96.8%</td>
</tr>
<tr>
<td>Species</td>
<td>Count</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Wildebeest</td>
<td>30,080</td>
</tr>
<tr>
<td>Zebra</td>
<td>21,090</td>
</tr>
<tr>
<td>Gazelle Thomsons</td>
<td>4,090</td>
</tr>
<tr>
<td>Buffalo</td>
<td>12,380</td>
</tr>
<tr>
<td>Hartebeest</td>
<td>3,710</td>
</tr>
<tr>
<td>Elephant</td>
<td>3,040</td>
</tr>
<tr>
<td>Human</td>
<td>2,940</td>
</tr>
<tr>
<td>Giraffe</td>
<td>2,510</td>
</tr>
<tr>
<td>Impala</td>
<td>2,480</td>
</tr>
<tr>
<td>Guineafowl</td>
<td>2,330</td>
</tr>
<tr>
<td>Gazelle Grants</td>
<td>2,310</td>
</tr>
<tr>
<td>Warthog</td>
<td>2,240</td>
</tr>
<tr>
<td>Other Bird</td>
<td>1,660</td>
</tr>
<tr>
<td>Hyena Spotted</td>
<td>1,580</td>
</tr>
<tr>
<td>Lion Female</td>
<td>1,000</td>
</tr>
<tr>
<td>Reedbuck</td>
<td>860</td>
</tr>
<tr>
<td>Eland</td>
<td>800</td>
</tr>
<tr>
<td>Hippopotamus</td>
<td>780</td>
</tr>
<tr>
<td>Topi</td>
<td>690</td>
</tr>
<tr>
<td>Baboon</td>
<td>460</td>
</tr>
<tr>
<td>Dikdik</td>
<td>440</td>
</tr>
<tr>
<td>Cheetah</td>
<td>380</td>
</tr>
<tr>
<td>Lion Male</td>
<td>280</td>
</tr>
<tr>
<td>Koribustard</td>
<td>210</td>
</tr>
<tr>
<td>Ostrich</td>
<td>200</td>
</tr>
<tr>
<td>Jackal</td>
<td>170</td>
</tr>
<tr>
<td>Serval</td>
<td>140</td>
</tr>
<tr>
<td>Secretary Bird</td>
<td>130</td>
</tr>
<tr>
<td>Hare</td>
<td>120</td>
</tr>
<tr>
<td>Aardvark</td>
<td>120</td>
</tr>
<tr>
<td>Waterbuck</td>
<td>110</td>
</tr>
<tr>
<td>Vervet Monkey</td>
<td>90</td>
</tr>
<tr>
<td>Bateared Fox</td>
<td>90</td>
</tr>
<tr>
<td>Porcupine</td>
<td>90</td>
</tr>
<tr>
<td>Bushbuck</td>
<td>80</td>
</tr>
<tr>
<td>Mongoose</td>
<td>70</td>
</tr>
<tr>
<td>Leopard</td>
<td>70</td>
</tr>
<tr>
<td>Aardwolf</td>
<td>50</td>
</tr>
<tr>
<td>Reptiles</td>
<td>40</td>
</tr>
<tr>
<td>Hyena Striped</td>
<td>30</td>
</tr>
<tr>
<td>Caracal</td>
<td>20</td>
</tr>
<tr>
<td>Rodents</td>
<td>10</td>
</tr>
<tr>
<td>Wild Cat</td>
<td>10</td>
</tr>
<tr>
<td>Civet</td>
<td>10</td>
</tr>
<tr>
<td>Honeybadger</td>
<td>10</td>
</tr>
<tr>
<td>Rhinoceros</td>
<td>10</td>
</tr>
<tr>
<td>Genet</td>
<td>10</td>
</tr>
<tr>
<td>Zorilla</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 2.15: The number and percent of capture events that contain animals that contain each species. The dataset is heavily imbalanced. Wildebeests and zebras form ~50% of the dataset (top 2 bars), while more than 20 other species add up to only ~1% of the dataset (bottom 20 bars).
Figure 2.16: The effect of three different methods: weighted loss, oversampling, and emphasis sampling on the classification accuracy for each class. In all of them, the classification performance for some rare classes has been improved at the cost of losing some accuracy on the frequent classes. The color indicates the percent improvement each method provides. All three methods improved accuracy for several rare classes: for example, the accuracy for the rhinoceros class dramatically increases from near 0% (original) to ~40% (weighted loss), ~80% (oversampling) and ~60% (emphasis sampling). Although the difference in global accuracies is not substantial, the weighted loss method has the best top-1 accuracy and the emphasis sampling method has the best top-5 accuracy. Moreover, it is notable that the emphasis sampling method has top-5 accuracy score of 98.2% which is slightly higher than the 98.1% accuracy of the baseline. In this plot, all classes are arranged based on their class sizes in descending order from the top to bottom.
Figure 2.17: Confusion matrices showing classifications and misclassifications of our highest-performing model for task I: Detecting Images That Contain Animals (VGG) and task III: Counting Animals (the ensemble). The numbers report how many images were collectively classified (green) or misclassified (red) on the test set with volunteer-provided labels for task I and expert-provided labels for task III (expert labels are not available for task I). The shades of red and green are a function of the percent of that row in that square.
Figure 2.18: A confusion matrix showing classifications and misclassifications for task II: Identifying Species for the ensemble of models, which performed the best on this task. The numbers report how many images were collectively classified (green) or misclassified (red) on the test set with expert-provided labels. The shades of red and green are a function of the percent of that row in that square.
Table 2.9: The accuracy, precision, and recall of the ensemble of models for task IV: Additional Attributes. The last column shows the percent of non-empty images (those with animals) that contain images of each class (each species) in the test set (the same numbers for the entire dataset are shown in Fig. 2.15). Note that, due to imbalanced classes (e.g. there are far more wildebeests images in the test set than zorillas), the total accuracy, precision, and recall are not a simple averaging of those statistics for each row, but are instead an average weighted by the percent of data in that row.

<table>
<thead>
<tr>
<th>Species</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>% of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aardvark</td>
<td>58.5%</td>
<td>75.6%</td>
<td>65.9%</td>
<td>00.089%</td>
</tr>
<tr>
<td>Aardwolf</td>
<td>80.0%</td>
<td>80.0%</td>
<td>80.0%</td>
<td>00.033%</td>
</tr>
<tr>
<td>Baboon</td>
<td>65.4%</td>
<td>79.3%</td>
<td>69.0%</td>
<td>00.499%</td>
</tr>
<tr>
<td>Bateared Fox</td>
<td>80.2%</td>
<td>86.0%</td>
<td>80.2%</td>
<td>00.094%</td>
</tr>
<tr>
<td>Buffalo</td>
<td>70.9%</td>
<td>83.2%</td>
<td>76.9%</td>
<td>04.103%</td>
</tr>
<tr>
<td>Bushbuck</td>
<td>52.8%</td>
<td>58.3%</td>
<td>55.6%</td>
<td>00.039%</td>
</tr>
<tr>
<td>Caracal</td>
<td>57.1%</td>
<td>57.1%</td>
<td>57.1%</td>
<td>00.015%</td>
</tr>
<tr>
<td>Cheetah</td>
<td>64.3%</td>
<td>74.5%</td>
<td>66.5%</td>
<td>00.542%</td>
</tr>
<tr>
<td>Civet</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>00.015%</td>
</tr>
<tr>
<td>Dikdik</td>
<td>63.7%</td>
<td>77.0%</td>
<td>65.1%</td>
<td>00.375%</td>
</tr>
<tr>
<td>Eland</td>
<td>69.6%</td>
<td>80.0%</td>
<td>72.3%</td>
<td>00.836%</td>
</tr>
<tr>
<td>Elephant</td>
<td>73.4%</td>
<td>82.3%</td>
<td>77.7%</td>
<td>03.284%</td>
</tr>
<tr>
<td>Gazelle Grants</td>
<td>73.6%</td>
<td>82.6%</td>
<td>77.8%</td>
<td>02.387%</td>
</tr>
<tr>
<td>Gazelle Thomsons</td>
<td>74.7%</td>
<td>85.7%</td>
<td>80.0%</td>
<td>14.178%</td>
</tr>
<tr>
<td>Genet</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>00.015%</td>
</tr>
<tr>
<td>Giraffe</td>
<td>79.2%</td>
<td>85.7%</td>
<td>81.1%</td>
<td>02.657%</td>
</tr>
<tr>
<td>Guineafowl</td>
<td>62.0%</td>
<td>76.5%</td>
<td>69.2%</td>
<td>02.548%</td>
</tr>
<tr>
<td>Hare</td>
<td>36.2%</td>
<td>43.8%</td>
<td>45.0%</td>
<td>00.087%</td>
</tr>
<tr>
<td>Hartebeest</td>
<td>84.5%</td>
<td>89.8%</td>
<td>87.2%</td>
<td>04.515%</td>
</tr>
<tr>
<td>Hippopotamus</td>
<td>61.4%</td>
<td>79.2%</td>
<td>62.8%</td>
<td>00.392%</td>
</tr>
<tr>
<td>Honeybadger</td>
<td>46.2%</td>
<td>61.5%</td>
<td>61.5%</td>
<td>00.028%</td>
</tr>
<tr>
<td>Human</td>
<td>66.5%</td>
<td>81.0%</td>
<td>73.0%</td>
<td>03.326%</td>
</tr>
<tr>
<td>Hyena Spotted</td>
<td>79.1%</td>
<td>86.9%</td>
<td>81.2%</td>
<td>01.248%</td>
</tr>
<tr>
<td>Hyena Striped</td>
<td>88.9%</td>
<td>100.0%</td>
<td>88.9%</td>
<td>00.020%</td>
</tr>
<tr>
<td>Impala</td>
<td>74.6%</td>
<td>84.8%</td>
<td>79.9%</td>
<td>02.692%</td>
</tr>
<tr>
<td>Jackal</td>
<td>55.3%</td>
<td>60.6%</td>
<td>60.6%</td>
<td>00.102%</td>
</tr>
<tr>
<td>Koribustard</td>
<td>73.7%</td>
<td>79.9%</td>
<td>77.3%</td>
<td>00.303%</td>
</tr>
<tr>
<td>Leopard</td>
<td>35.7%</td>
<td>71.4%</td>
<td>35.7%</td>
<td>00.030%</td>
</tr>
<tr>
<td>Lion Female</td>
<td>74.8%</td>
<td>86.6%</td>
<td>76.4%</td>
<td>01.109%</td>
</tr>
<tr>
<td>Lion Male</td>
<td>85.7%</td>
<td>92.2%</td>
<td>87.4%</td>
<td>00.320%</td>
</tr>
<tr>
<td>Mongoose</td>
<td>75.0%</td>
<td>87.0%</td>
<td>78.0%</td>
<td>00.109%</td>
</tr>
<tr>
<td>Ostrich</td>
<td>72.4%</td>
<td>76.5%</td>
<td>72.4%</td>
<td>00.185%</td>
</tr>
<tr>
<td>Other Bird</td>
<td>63.4%</td>
<td>76.0%</td>
<td>68.9%</td>
<td>01.455%</td>
</tr>
<tr>
<td>Porcupine</td>
<td>61.7%</td>
<td>73.3%</td>
<td>68.3%</td>
<td>00.065%</td>
</tr>
<tr>
<td>Reedbuck</td>
<td>76.5%</td>
<td>88.2%</td>
<td>77.6%</td>
<td>00.488%</td>
</tr>
<tr>
<td>Reptiles</td>
<td>55.6%</td>
<td>83.3%</td>
<td>55.6%</td>
<td>00.039%</td>
</tr>
<tr>
<td>Rhinoceros</td>
<td>57.1%</td>
<td>85.7%</td>
<td>57.1%</td>
<td>00.015%</td>
</tr>
<tr>
<td>Rodents</td>
<td>50.0%</td>
<td>100.0%</td>
<td>50.0%</td>
<td>00.026%</td>
</tr>
<tr>
<td>Secretary Bird</td>
<td>75.6%</td>
<td>90.6%</td>
<td>76.1%</td>
<td>00.196%</td>
</tr>
<tr>
<td>Serval</td>
<td>69.7%</td>
<td>73.0%</td>
<td>77.0%</td>
<td>00.133%</td>
</tr>
<tr>
<td>Topi</td>
<td>80.5%</td>
<td>89.8%</td>
<td>83.7%</td>
<td>00.775%</td>
</tr>
<tr>
<td>Vervet Monkey</td>
<td>79.4%</td>
<td>79.4%</td>
<td>81.7%</td>
<td>00.131%</td>
</tr>
</tbody>
</table>
Figure 2.19: From the empty vs. animal task, shown are nine images, the human-volunteer answer, and the VGG network’s answer along with its confidence. The first row of the images shows three correct answers by the model. The middle row shows three examples in which the model is correct, but volunteers are wrong, showing that volunteer labels are imperfect. The bottom row of images shows three examples in which volunteers are correct, but the model is wrong.
Figure 2.20: GG-CAM generated localized visual features for identifying Baboons and Impalas. The DNN relies on faces and tails to recognize Baboons. For Impalas, the black streaks on the rear, the white underbelly and dark back, and the black spots between the rear legs and underbelly are the most distinctive features learned by the DNN. The image is from [80].
Figure 2.21: Top nine image patches for the top-5 most responsive neurons. Some neurons primarily respond to backgrounds (e.g., Top 1, Top 3, and Top 4 for Oribi), which means DNNs associate some common backgrounds with the species. This fact was one of our main motivations for designing the pipeline which is presented in chapter 3. The image is from [80].
Chapter 3

An Active Deep Learning System for Species Identification and Counting in Camera-Trap Images

3.1 Abstract

Biodiversity conservation depends on accurate, up-to-date information about wildlife population distributions. Motion-activated cameras, also known as camera traps, are a critical tool for population surveys, as they are cheap and non-intrusive. However, extracting useful information from camera-trap images is a cumbersome process: a typical camera-trap survey may produce millions of images that require slow, expensive manual review. Consequently, critical information is often lost due to resource limitations, and critical conservation questions may be answered too slowly to support decision-making. Computer vision is poised to dramatically increase efficiency in image-based biodiversity surveys, and recent studies have successfully harnessed deep learning techniques for automatic information extraction from camera-trap images. However, the accuracy of results depends on the amount, quality, and diversity of the data available to train models, and the literature has focused on projects with millions of relevant, labeled training images. Many camera-trap projects do not have a large set of labeled images (or any labeled images), and hence cannot benefit from existing
machine learning techniques. Furthermore, even projects that do have labeled data from similar ecosystems have struggled to adopt deep learning methods, because image classification models highly depend on specific image backgrounds (i.e., camera locations). In this paper, we focus not on automating the labeling of camera-trap images, but on accelerating this process. We combine the power of machine intelligence and human intelligence to build a scalable, fast, and accurate active learning system to minimize the manual work required to identify and count animals in camera-trap images. Our proposed scheme can decrease manual labeling effort by over 99.5%, while producing labels comparable to state-of-the-art results. Our system achieves over 92.5% accuracy on a 3.2 million image dataset with as few as 25,000 manual labels. That is in contrast to our previous system, which achieved 90.9% top-1 accuracy, but required 3.2 million images to do so.

3.2 Introduction

Wildlife population distribution studies depend on tracking observations, i.e. occurrences of species at recorded times and locations. This information facilitates the modeling of population sizes, distributions, and environmental interactions [85–87]. Motion-activated cameras, or camera traps, provide a non-intrusive and comparatively cheap method to collect observational data, and have transformed wildlife ecology and conservation in recent decades [1, 88]. Although camera-trap networks can collect large volumes of images, turning raw images into actionable information is done manually, i.e. human annotators view and label each image [8]. The burden of manual review is the main disadvantage of camera-trap surveys, and limits the use of camera traps for large-scale studies.

Fortunately, recent advances in artificial intelligence have significantly accelerated information extraction. Inspired by the human nervous system, deep neural networks [24, 28] have advanced the state of the art in tasks such as machine translation [34, 35], speech recognition [31, 33], and image classification [36, 37]. Deep convolutional neural networks are a class of deep neural networks designed specifically to process images [24, 42].

Recent work has demonstrated that deep convolutional neural networks can achieve a
high level of accuracy in extracting information from camera-trap images, including species labels, count, and behavior, while being able to process hundreds of images in a matter of seconds [89,90]. The wide availability of deep learning for fast, automatic, accurate, and inexpensive extraction of such information could save substantial amounts of time and money for conservation biologists.

The accuracy of deep neural networks depends on the abundance of their training data [24]; state-of-the-art networks typically required hundreds of thousands of labeled training images. This volume of labeled data is not typically available for camera-trap projects; therefore, most projects cannot yet effectively harness deep learning. Even in cases where an extensive training set is available, training labels are almost always in the form of image-level or sequence-level species labels, i.e. they do not contain information about where animals occur within each image. This results in a strong dependency of deep networks on image backgrounds, which limits the ability of deep learning models to produce accurate results even when applied to regions that are similar to their training data.

This paper aims to address these issues, and to enable camera-trap projects with few or no labeled images to take advantage of deep neural networks for fast, transferable, automatic information extraction. Using object detection models, transfer learning, and active learning, our results show that our suggested method can achieve the same level of accuracy as [89], with a 99.5% reduction in manually-annotated training data.

### 3.3 Background and Related Work

#### 3.3.1 Deep Learning

The most common type of machine learning used for image classification is supervised learning, where input examples are provided along with corresponding output examples (for example, camera-trap images with species labels), and algorithms are trained to translate input to output [27].

Deep learning is a specific type of supervised learning, built around the artificial neural network [24,91], a class of machine learning algorithms inspired by the structure of biological
nervous systems. Each artificial neuron in a network takes in several inputs, performs a set of transformations to combine those inputs, and passes the result along as input to other neurons. Neurons are usually arranged in several layers; neurons of each layer receive input from the previous layer, process them, and pass their output to the next layer. A deep neural network is simply a neural network with three or more layers.

In a fully-connected layer, each neuron receives input from all the neurons in the previous layer. On the other hand, in convolutional layers, each neuron is only connected to a small group of neurons in the previous layer, and this small group of neurons learns to scan the previous layer’s output for patterns. A neural network with one or more convolutional layers is called a convolutional neural network, or CNN. CNNs are specifically designed to deal with spatial patterns in the input data, and they have shown excellent performance on image-related problems.

Every neuron in a neural network has several parameters that it uses to translate its input to its output; training a neural network means adjusting these parameters for every neuron so that the whole network produces the desired output for each input example. To tune these parameters, a measure of the discrepancy between the current output of the network and the desired output is computed; this measure of discrepancy is called the loss function. There are numerous loss functions used in the literature, appropriate for different problem classes. After calculating the loss function, an algorithm called back-propagation calculates the contribution of each parameter to the loss value, then adjusts the parameters so that the loss value is minimized. The back-propagation algorithm is an iterative algorithm, i.e. it is applied many times during training. At every iteration of the back-propagation algorithm, the parameters take one step toward a minimum.

The accuracy of deep learning compared to other machine learning methods makes it applicable to a variety of complex problems. In this paper, we focus on enhancing deep neural networks to extract information from camera-trap images more efficiently.
3.3.2 Image Classification

In the computer vision literature, *image classification* refers to assigning images into several pre-determined classes. More specifically, image classification algorithms typically assign a probability that an image belongs to each class. For example, species identification in camera-trap images is an image classification problem in which the input is the camera-trap image and the output is the probability of the presence of each species in the image [89][90]. Image classification models can be easily trained with image-level labels, but they suffer from several limitations:

1. Typically the most probable species is considered to be the label for the image; consequently, classification models cannot deal with images containing more than one species.

2. They are not designed to perform counting which is a non-classification problem. Hence, adapting them for counting can lead to a poor performance.

3. What the image classification models see during training are the images and their associated labels; they have not been told what *parts* of the images they should focus on. Therefore, they not only learn about patterns representing animals, but will also learn some information about *backgrounds*. This limits their transferability to new locations. Therefore, when applied to new datasets, accuracy is typically lower than what was achieved on the training data. For example, Tabak et al. [90] showed that their model trained on North American images was less accurate at separating empty from non-empty images in an African dataset.

3.3.3 Object Detection

*Object detection* algorithms attempt to not only classify images, but to locate instances of pre-defined object classes within images. Object detection models output coordinates of bounding boxes containing objects plus a probability that each box belongs to each class. Object detection models thus naturally handle images with objects from multiple classes.
A hypothesis of this paper is that object detection models may also be less sensitive to image backgrounds (because the model is told explicitly which regions of each image to focus on), and may thus generalize more effectively to new locations.

Figure 3.1: Object detection models are capable of detecting multiple occurrences of several object classes.

The ability of object detection models to handle images with multiple classes, makes them appealing for camera-trap problems, where multiple species may occur in the same images. However, training object detection models requires bounding box and class labels for each animal in the training images. This information is rarely relevant for ecology, and obtaining bounding box labels is costly; consequently, few camera-trap projects have such labels. This makes training object detection models impractical for many camera-trap projects, although recent work has demonstrated the effectiveness of object detection when bounding box labels are available [93].

3.3.4 Transfer Learning

Transfer learning is the application of knowledge gained from learning a task to a similar but different task [24,43]. Transfer learning is highly beneficial when we have a limited
number of labeled samples to learn a task (for example, species classification in camera-trap images), but we have a large amount of labeled data for learning a different, relevant task (for example, general-purpose image classification). In this case, a network can first be trained on the large dataset and then fine-tuned on the target dataset.[24][43].

### 3.3.5 Active Learning

In contrast to machine learning problems in which we first gather a set of labeled samples and then attempt to learn how to do a task, *active learning* is a technique for training a machine learning model in which we have a human annotator in the loop to ask for some labels, but we try to minimize the number of such requests. In an active learning scenario, we have a large pool of unlabeled data and an oracle (e.g., a human) who can label the samples upon request. The active learning algorithm must select the samples from the pool for the oracle to label so that the underlying machine learning model can quickly learn the requested task.

Active learning algorithms maintain an underlying machine learning model, such as a neural network, and try to improve that model by selecting training samples. Active learning algorithms typically start training the underlying model on a randomly-selected labeled sample set. After training the initial model, various criteria can be employed to select the most informative unlabeled samples to be passed to the oracle for labeling[94]. Model uncertainty[95], query-by-committee (QBC)[96], expected model change[97], expected error reduction[98], and density-based methods[97,99] are among the most popular query selection strategies for active learning. After obtaining the new labels from the oracle, the same active learning procedure can be repeated until a pre-determined number of queries have been labeled, or until an acceptable accuracy is reached. Algorithm 1 summarizes an active learning workflow in pseudocode.

### 3.3.6 Embedding Learning

An *embedding function* maps data from a high-dimensional space to a lower-dimensional space, for example from the millions of pixel values in an image (high-dimensional) to a
Algorithm 1 Active learning procedure

1: Start from a randomly-selected labeled subset of data
2: while stopping criteria not met do
3:     Train the underlying model with the available labeled samples
4:     Compute a selection criterion for all the samples in the unlabeled pool
5:     Select n samples that maximize the criterion
6:     Pass the selected samples to the oracle for labeling
7:     Gather the labeled samples and add them to the labeled set
8: end while

vector of dozens or hundreds of values. Many dimensionality reduction algorithms such as PCA \[100\] and LDA \[100\], or visualization algorithms like t-SNE \[101\], can be regarded as embedding functions.

Deep neural networks are frequently used for dimensionality reduction: the input to a deep network often has many values, but layers typically get smaller throughout the network, and the output of a layer can be viewed as a reduced representation of the network’s input. In some cases we may learn such a representation in the course of training another task (e.g. image classification), but we can also specifically learn an embedding that meet some desired criteria. For example, deep neural networks can be trained so that they map samples from the same class to nearby regions in the learned embedding space \[102\,103\].

### 3.3.7 Datasets

Three data sets will be used for training and evaluating models in our experiments: Snapshot Serengeti, eMammal Machine Learning, NACTI, and Caltech Camera Traps.

**Snapshot Serengeti**

The Snapshot Serengeti dataset contains 1.2 million multi-image sequences of camera-trap images, totaling 3.2 million images. Sequence-level species, count, and description labels are provided for 48 animal categories by citizen scientists \[8\]. Approximately 75% of the images are labeled as empty. Wildebeest, zebra, and Thomsons gazelle are the most common species.
eMammal Machine Learning

eMammal is a data management platform for both researchers and citizen scientists working with camera-trap images. We worked with a dataset provided by the eMammal team specifically to support machine learning research, containing over 450,000 images and over 270 species from a diverse set of locations across the world [104].

NACTI

The North America Camera Trap Images (NACTI) dataset contains 3.7M camera-trap images from five locations across the United States, with labels for 28 animal categories, primarily at the species level (for example, the most common labels are cattle, boar, and red deer) [90]. Approximately 12% of images are labeled as empty.

Caltech Camera Traps

The Caltech Camera Traps (CCT) dataset contains 245K images from 140 camera traps in the Southwestern United States. The dataset contains 22 animal categories. The most common species are opossum, raccoon, and coyote. Approximately 70% of the images are labeled as empty.

3.4 Methods

In this paper, we propose a pipeline to tackle several of the major roadblocks preventing the application of deep learning techniques to camera-trap images. Our proposed pipeline takes advantages of transfer learning and active learning to concurrently help with the transferability issue, dealing with multiple species, inaccurate counting, and limited-data problems. In this section, we explain the details of our procedure and the motivations for each step.

3.4.1 Proposed Pipeline

Our suggested pipeline starts with running a pre-trained object detection model, based on the Faster-RCNN object detection algorithm [83], over the images. This model has only
one class (*animal*) and was trained on several camera-trap datasets that have bounding box annotations available. The pre-trained model is available to download and use [105]. We threshold the predictions of the model at 90% confidence and do not consider any detection with less than 90% confidence. The pre-trained object detection model accomplishes three related tasks:

1. It can tell us if an image is empty or contains animals; any image with no detections above 90% confidence is marked as empty.

2. It can count how many animals are in an image; we count animals by summing the number of detections above 90% confidence.

3. It can crop the images to contain animals without background; we crop detections above 90% confidence and use these cropped images to recognize species in the next steps of the pipeline.

Thus, after running the object detection model over a set of images, we have already marked empty images, counted animals in each image, and gathered the crops to be further processed. Image classification models require fixed-sized inputs; since crops are variable in size, we resize all the crops to 256*256 pixels via the image resize function in SciPy v.0.19.1. This set of cropped, resized images, which now contain animals with very little background, is the data we process with active learning.

Our active learning procedure leverages two neural networks: an *embedding model* trained on a very large dataset that is related to our target data (e.g. any large camera-trap dataset), and a *classification model* that is trained only on labeled data from our target dataset.

One of the major challenges of active learning is that we expect to have relatively few labeled images for our target dataset, typically far too few to train a deep neural network. Consequently, when training a model for a new dataset, we would like to leverage knowledge derived from related datasets (i.e., other camera-trap images); this is a form of *transfer learning*. To this end, before processing the crops from a target dataset, we learn an *embedding model* (a deep neural network) on a large data set, and use this model to embed the crops
from our target data set into a 256-dimensional feature space. That is, the embedding model turns each image into a 256-dimensional feature vector. Several methods exist for training embeddings; we examine these options in Sec. 3.5.3. We also fine-tune this embedding model throughout the active learning process.

After obtaining the features for each crop in the lower-dimensional space, we have all the necessary elements to start the active learning loop over our data. We employ a simple neural network with one hidden layer consisting of 100 neurons as our classification model. We start the active learning process by asking the oracle to label 1,000 randomly-selected images. We then train our classification model using these 1,000 labeled images. At each subsequent step, we select 100 unlabeled images that maximize our image selection criteria (we will discuss different image selection strategies in Sec. 3.5.3), and ask the oracle to label those 100 images; the classifier model is re-trained after each step. We also fine-tune the embedding model for the initial 1,000 samples and also after every 20 steps, starting from 2,000 samples.

Our pipeline is presented in pseudocode form as Algorithm 2.

**Algorithm 2 Proposed pipeline**

1: Run a pre-trained object detection model on the images  
2: Run a pre-trained embedding model on the crops produced by the objection detection model  
3: Select 1,000 random images and request labels from the oracle  
4: Fine-tune the embedding model on the labeled set  
5: Run the embedding model on the labeled set to produce feature vectors  
6: Train the classification model on the labeled feature vectors  
7: while Termination condition not reached do  
8: Select 100 images using the active learning selection strategy, pass these to the human oracle for labeling  
9: Fine-tune the classification model on the entire labeled set of the target dataset  
10: if number of examples % 2,000==0 then  
11: Fine-tune the embedding model on all the labels obtained for the target dataset  
12: end if  
13: end while
3.5 Experiments and Results

As explained above, our suggested pipeline consists of three steps: (1) running a pre-trained detector model on images, (2) embedding the obtained crops into a lower-dimensional space, and (3) running an active learning procedure. In this section, we report the results of our pipeline and analyze the contribution of these steps to the overall results. For these results, the eMammal Machine Learning dataset is used to train the embedding model, and the target dataset is Snapshot Serengeti. We chose eMammal Machine Learning for training our embedding because it is the most diverse of the available datasets and thus likely provides the most general model for applying to new targets. We chose Snapshot Serengeti as our target dataset to facilitate comparison with the results presented in [89].

3.5.1 Empty vs. Animal

As per above, we first run a pre-trained object detection model on the target dataset, and we consider detections above 90% confidence. As the results in Table 3.1 show, the detector model has 91.71% accuracy, 84.47% precision, and 84.23% recall. Compare these results with those of [89] for detecting empty images which has 96.83% accuracy, 97.50% precision, and 96.18% recall. We stress that this accuracy came “for free,” without labeling any image manually for the target dataset, while Norouzzadeh et al. [89] used 1.6 million labeled images from the target dataset to obtain their results. Moreover, we expect this accuracy to improve as the detection model gets trained on more diverse data.

Table 3.1: The confusion matrix for the pre-trained object detection model applied to the Snapshot Serengeti dataset

<table>
<thead>
<tr>
<th>Model Predictions</th>
<th>Empty</th>
<th>Animal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground Truth Labels</td>
<td>Empty</td>
<td>2,219,404</td>
</tr>
<tr>
<td></td>
<td>Animal</td>
<td>133,769</td>
</tr>
</tbody>
</table>
3.5.2 Counting

Using a pre-trained object detection model allows us to not only distinguish empty images from images containing animals, but also to count the number of animals in each image. This simply means counting the number of bounding boxes with more than 90% confidence for each image. This simple counting scheme can give us the exact number of animals for 72.4% of images, and the predicted count is either exact or within one bin for 86.8% of images. Comparing the accuracy of counting with Norouzzadeh et al. [89], both the top-1 accuracy and the percent within +/- 1 bin are slightly improved, and this improvement comes “for free” (i.e., without additional labeling).

3.5.3 Species Identification

After eliminating empty images and counting the number of animals in each image, the next task is to identify the species in each image. As per above, for species identification, we first embed the cropped boxes into a lower-dimensional space, then we run an active learning algorithm to label the crops. In the next three subsections, we discuss the details of each step and compare several options for implementing them.

Embedding Spaces

There are various embedding techniques to map high-dimensional images to a smaller set of features. We experimented with two common embedding techniques: (1) using features of the last layer of an image classification network trained on a similar dataset to identify species and (2) using features obtained from training a deep neural network using triplet loss [103, 106] on a similar dataset. Each triplet consists of a baseline sample (the anchor), another sample with the same class as anchor (positive), and a sample belonging to a different class (negative). To minimize the triplet loss, the distance between the anchor and the positive sample should be minimized, and the distance between the anchor and the negative sample should be maximized. We employed the ResNet-50 architecture [36] for both; only the loss function differs between these methods. The cross-entropy loss function is used for
training the image classification network (this is the most common loss function used for classification problems).

After extracting the features with both techniques, we run the same active learning strategy (k-Center [99]) on both sets of features. For 25,000 labels, we achieved 85.23% accuracy for the features extracted from the last layer of a classification model and 91.37% accuracy with the triplet loss features. Fig. 3.2 depicts the t-SNE visualization of the learned embedding space. These results indicate that using triplet loss to build the embedding space provides better accuracy than features derived from an image classification model. We expect this gap to get smaller as we fine-tune the embedding model with more labeled images (Fig. 3.3).

**Active Learning Strategies**

Different strategies can be employed to select samples to be labeled by the oracle. The most naive strategy is selecting queries at random. Here we try five different query selection strategies and compare them against random strategy. In particular, we try (1) model uncertainty criteria (confidence, margin, entropy) [95], information diversity [107], margin clustering [108], and k-Center [99]. For all of these experiments, we use triplet-loss features. All the active learning strategies work better than random. The highest accuracy is achieved with the k-Center strategy, which reaches 92.2% accuracy with 25,000 labels. The k-Center method selects a subset of unlabeled samples such that the loss value of the selected subset is close to the “expected” loss value of the remaining data points. Fig. 3.4 summarizes the results. At 14,000 labels, we match the accuracy of Norouzzadeh et al. for the same architecture (ResNet-50); this is over a 99.5% reduction in human labor for labeling to achieve the same result.

**Crops vs. Full-Image Classification**

As per above, we identify species in images that have been cropped by the object detection model. To assess the contribution of this choice to our overall accuracy, we also tried to classify species using full images. Fig 3.5 shows that using crops can produce significantly better...
results than using full images. This is likely because cropped images eliminate background pixels, allowing the classification model to focus on the animals.

### 3.6 Further Improvement

This paper demonstrates the potential to significantly reduce human annotation time for camera-trap images via active learning. While we have explored some permutations of our active learning pipeline, we have not extensively explored the space of parameters within this pipeline. We believe there are at least three mechanisms by which our results could be improved.

1. Every deep learning algorithm has numerous *hyperparameters*, options selected by the data scientist before machine learning begins. For this paper, we used well-known values of hyperparameters to train our models. Tuning hyperparameters is likely to improve results. In particular, we only used the ResNet-50 architecture for embedding and a simple one-hidden-layer architecture for classification; further probing of the architecture space may improve results.

2. We use a pre-trained detector, and we do not modify this model in our experiments. However, after obtaining labels from the oracle, we can fine-tune the detector model in addition to the embedding and classification models.

3. After collecting enough labeled samples for a dataset, it is possible to combine the classification and detection stages into a single multi-class detector model. This may improve accuracy, but almost certainly will improve computational efficiency when applying the model to new data sets.

### 3.7 Conclusion

In this work, we demonstrated that *active learning* - the use of machine learning methods to optimize human annotation resources - can dramatically reduce the human effort needed
to extract information from camera-trap datasets. Our pipeline may facilitate the deployment of large camera-trap arrays for wildlife population studies, by reducing the annotation bottleneck.
Figure 3.2: t-SNE visualization of the cross-entropy (a) and (b) triplet embedding spaces for 100,000 randomly selected crops from the Snapshot Serengeti dataset. Both the embedding models are trained on 25,000 labeled samples from the target dataset. The embedding based on triplet features shows a more intuitive distribution of species in the embedding space.
Figure 3.3: The accuracy of an active learning process using triplet loss features vs using cross-entropy loss features. The triplet loss features perform better.

Figure 3.4: Performance of different active learning query strategies using triplet loss features over the Snapshot Serengeti dataset. k-Center achieves the best accuracy at 30,000 queries.
Figure 3.5: The accuracy of k-Center active learning strategy using triplet loss features over crops vs k-Center active learning strategy using triplet loss features over full images on the Snapshot Serengeti dataset. Crops provide a substantial increase in accuracy.
Chapter 4

Neuromodulation Improves the Evolution of Forward Models

This chapter is a direct copy of the following publication:


4.1 Abstract

Many animals predict the outcomes of their actions by internal models. Such “forward models” enable animals to rapidly simulate many actions without performing them to choose an appropriate action. Robots would similarly benefit from forward models. However, such models must change over time to account for changes in the environment or body, such as injury. Thus, forward models must not only be accurate, but also adaptable. Neural networks can learn complex functions with high accuracy, hence they are suitable candidates to build forward models for robots. Generally, neural networks are static, which means once they pass the training phase, their weights remain unchanged and they thus cannot adapt themselves if something about the world or their body changes. Plastic neural networks change their connections over time via local learning rules (e.g. Hebbian rule) and can thus
deal with unforeseen changes. A more complex, yet still biologically-inspired, technique is neuromodulation, which can change per-connection learning rates in different contexts. In this paper, we test the hypothesis that neuromodulation may improve the evolution of forward models because it can heighten learning after drastic changes such as injury. We compare forward models evolved with neuromodulation to those evolved with static neural networks and Hebbian plastic neural networks. The results show that neuromodulation produces forward models that can adapt to changes significantly better than the controls. Our findings suggest that neuromodulation is an effective tool for enabling robots (and artificial intelligence agents, more generally) to have more adaptable, effective forward models.

![Figure 4.1: The overall diagram of a typical forward model.](image)

4.2 Introduction and background

Living animals such as insects, birds, and humans employ internal models that allow them to predict the sensory outcomes of their movements [109,110]. These models take an internal copy of an action-generating signal and predict the sensory outcomes that will result from that action. A famous experiment that showed the existence of forward models in animals is the Helmholtz experiment [109]. Helmholtz was a 19th-century German physician and physicist who suggested that the brain predicts the position of the eye based on the movement signal rather than actually sensing the eye’s position. He found that when the eye is moved by an external source the brain wrongly feels that the world around it is moving [109]. His experiment shows that there is an internal model in the brain that predicts the position of
Figure 4.2: Neuromodulation enables fine-grained control over local learning rules. (a) In plain Hebbian learning, a connection strength is updated as a function of pre- and post-synaptic activity only. (b) With neuromodulation, the rate of learning can be up-regulated or down-regulated in different contexts depending on the data. That regulation occurs via a modulatory signal output by modulatory neurons within the network (red circle). Thus, parts of a neural network recognize contexts in which the learning rates of individual connections in the network should be increased and decreased. For example, learning could be turned off if a model is error-free, but turned on if the model begins producing errors (e.g. after fatigue or damage occurs).

the eye according to the movement signal. Another fact supporting the existence of forward models is that we cannot tickle ourselves because our brains predict what happens when we stimulate ourselves, but other people can tickle us because our brain does not have copies of their movement signals, hence it cannot predict the outcome [111].

Robots can similarly benefit from internal models because such models could help robots predict the sensory outcomes of their movements. Being able to predict the consequences of actions could grant robots the ability to better adapt to changes to their environments or bodies; specifically, they could mentally simulate possible movements to select an appropriate one without having to incur the costs and hazards of trying each action the real world [15][16]. Such models can also enable robots to model other agents, whether humans or other robots, improving their ability to plan and respond accordingly [112].

Formally, a forward model is a function that at each time step $t$ takes a motor command $C^t$ as input and predicts the sensory outcomes $S^t$ as output (Fig. 4.1). After the predictions have been made, we can calculate the difference between the predictions and actual sensory
outcomes (i.e. error) and those error signals can be used to enhance the predictions of the forward model. Many modern robots take advantage of forward models, but there are two major issues with current forward models.

The first is that they are often manually designed by humans for a specific environment. As robots and the tasks they perform grow more complex, this approach becomes increasingly untenable. Accordingly, researchers have developed several methods for automatically building forward models. Dearden and Demiris [113] proposed an adaptable autonomous forward model based on Bayesian networks and applied it to predict outcomes of motor commands. In their experiments, they had an arm with two motors along with a simple vision system to capture the outcomes. Moriguchi and Lipson [15] proposed a framework for learning forward models via symbolic regression; they applied their model for motion planning and control of a motored pendulum.

The second drawback to most current forward models is that they are static (i.e. unchanging). If the environment does change, either the forward models fail completely or they must be retrained from scratch. An example of the latter is Bongard, et al. [16]. Instead of creating a single, adaptable forward model, they efficiently synthesize a set of (non-adaptable) models and actively improve that set to find a new model that is accurate given the drastic change that has occurred.

This paper makes two major contributions. The first is to introduce a method that evolves adaptable (i.e. plastic) forward models. The second is to test whether neuromodulation improves the evolution of such forward models. The results confirm that evolution can automatically make effective forward models and the performance of such models is improved when they are evolved with neuromodulatory learning.

4.2.1 Neural Networks

In animals, the nervous system is responsible for building internal models. Artificial Neural Networks or simply Neural Networks (NN) are a class of computational models inspired by animal brains that consist of several interconnected computational elements (neurons). Each neuron takes signals from its inputs, processes them, and passes the result through weighted
connections to interconnected neurons. Training a neural network involves adjusting its weights to compute a useful output as a function of its inputs. After we train a neural network it can predict outputs for novel inputs. Neural networks are able to model complex input-output relationships, making them suitable for building internal models. Here, we automatically design neural networks that serve as forward models for a simple robot.

There are two general approaches to train NNs. The first approach is with supervised learning via the backpropagation algorithm \[114\]. Given a fixed network structure and a set of training input-output examples, the backpropagation algorithm feeds each example to the network, computes the outputs of the network with the current weights, calculates the difference between the outputs and the target outputs (error) and then adjusts the network’s weights such that the error for each example is reduced \[114\].

Because we are interested in learning not only how to produce effective forward models for robotic control, but also to learn about the evolution of forward models in animals, the forward models in this paper are produced by evolving neural networks, a field called neuroevolution \[115–119\]. Neuroevolution (NE) trains neural networks with evolutionary algorithms (EAs), some of which can evolve both the structure (i.e. topology) and weights of neural networks given a measure of performance \[115–117\]. Specifically, NE builds a population of candidate networks and then refine them by applying small changes (mutations) \[116\]. These small changes may include adding or removing neurons, adding or removing a random connection, changing the weight of a connection, changing the type of a neuron (standard or modulatory), and so on.

NE methods can work in reinforcement learning situations in which the correct output is not known for every input shown to the network during training. Instead, NE only requires a measure of performance to be periodically provided (e.g. at the end of a trial that includes exposing a network to many unlabeled inputs). For example, in robot navigation, it is easy to evaluate the final position of the robot to see if it is near a goal, but it is difficult to provide a set of correct actions for each situation the robot encounters. NE has been applied successfully for a variety of problems including Atari game playing \[120\], control \[121, 122\], and automatic feature selection \[123\].
4.2.2 Plasticity

Neural networks usually have two phases. The first is a training phase and the second is a testing phase in which the trained network predicts outputs for unseen inputs. Usually, NNs are static, which means we only change their weights during their training phase, after which the weights remain unchanged during testing while the NN makes decisions.

An ideal forward model would be able to respond to unforeseen changes that occur within the lifetime of the organism. Generally, if a change occurs in the environment, static NNs cannot adapt appropriately because their weights are unchangeable. On the other hand, plastic neural networks are capable of adjusting their structure (structural plasticity) or weights (synaptic plasticity) during their lifetime [124], making them capable of behavioral change and providing more behavioral robustness [125].

Synaptic plasticity, which we limit our focus to in this paper, can come via local learning rules that change weights [126–128]. In general, each connection within a network could have its own learning rule. The general form of a local learning rule is given in Eq. 4.1 and 4.2 where \( w_{new} \) is the new weight of the connection, \( w_{old} \) is the old weight of the connection, \( \Delta w \) is the change of weight, \( \Phi \) is the learning rule, \( o_{pre} \) is pre-synaptic activity, and \( o_{post} \) is post-synaptic activity.

\[
\begin{align*}
    w_{new} &= w_{old} + \Delta w \\
    \Delta w &= \Phi(o_{pre}, o_{post}, w_{old})
\end{align*}
\]

A well-known local learning rule formula is the plain Hebbian rule (Eq. 4.3), which is applied to all of the connections of a network [17]. The plain Hebbian rule can be generalized to specific types of pre- or post-synaptic correlations [129], but here we focus on the simpler, more commonly used version in Eq. 4.3. During training and testing phases, an input is presented to the network, its output for that input is calculated, pre- and post-synaptic activities are captured, a learning rate (\( \eta \)) is applied, and each weight is updated. This cycle then repeats for the next input.

\[
\Delta w = \eta \cdot o_{pre} \cdot o_{post}
\]
4.2.3 Neuromodulation

In the brain, some neurons control the learning behavior of the connections between other neurons by releasing one or more chemical transmitters. This phenomenon is usually referred to as neuromodulation [18]. Inspired by this natural phenomenon, researchers have added an abstraction of neuromodulation to NNs to improve short-term memory [130], evolvability [131], learning and adaptability [132], and fine-grained plasticity control [18].

Following nature, in the NN abstraction of neuromodulation, some neurons called modulatory neurons control the plasticity of connections between other neurons (Fig. 4.2). That requires two types of neurons: standard neurons and modulatory neurons [18]. Each neuron $i$ has two ways it can be activated: standard activations $a_i$ come from standard neurons (Eq. 4.4), and modularity activations $m_i$ come from modularity neurons (Eq. 4.5). In Eq. 4.4 and Eq. 4.5 $i$ and $j$ are neuron indices, $w_{ji}$ is the connection weight between neuron $j$ and neuron $i$, and $o_j$ is the post-synaptic activity of neuron $j$.

\[
a_i = \sum_{j \in \text{Std}} w_{ji} \cdot o_j \quad (4.4)
\]

\[
m_i = \sum_{j \in \text{Mod}} w_{ji} \cdot o_j \quad (4.5)
\]

With neuromodulation, weight changes are computed by Eq. 4.6. In comparison with other plasticity models, neuromodulation enables fine-grained control over plasticity because it grants the capability of turning local rules on or off for some neurons dynamically depending on the data.

\[
\Delta w_{ji} = \tanh(m_i/2) \cdot \eta \cdot [o_i \cdot o_j] \quad (4.6)
\]

4.3 Methods

In this paper, in order to automatically build forward models, we evolve neural networks. To build adaptable forward models (those that can change during an organism’s lifetime), we evolve plastic neural networks. Specifically, we compare two different types of learning:
Figure 4.3: The simple robot arm from our experiments. (a) In normal conditions, a motor command of $\theta$ will cause the arm to move to that angle. (b) In case of fatigue, a motor command of $\theta$ will cause the arm to move to slightly less than $\theta$. (c) In case of damage, a motor command of $\theta$ does not have any effect because the arm is stuck at a previous angle.

Figure 4.4: Linear and step-wise fatigue variants. With linear fatigue, fatigue increases linearly over time. With step-wise fatigue, fatigue increases in several discrete steps.

Figure 4.5: The three steps of the command implementation process. (a) The first neural network pass: a motor command and zeros are input into the model and the neural network forward model is run to compute predictions. (b) Error signals are computed from the first pass, which determine fitness. Note that this step does not require an update of the neural network. (c) The second neural network pass: the motor command and error signals from the first pass are fed into the model so that its learning algorithm can adjust the weights of the model. The error signals from this pass are ignored and do not affect fitness.
simple Hebbian learning and neuromodulation. Our test problem is to build a forward model for a simple robot arm (Fig. 4.3) that experiences changes over time (specifically, fatigue and damage). This paper tests the hypothesis that neuromodulation helps to produce more adaptable forward models. For this purpose, we compare a static NN model, a Hebbian plastic NN model, and a neuromodulation-controlled Hebbian plastic NN model.

4.3.1 Test Problem

The test problem involves a robot arm moved by a single degree of freedom at its base that can rotate 360 degrees. The robot is exposed to several sequences of commands within its lifetime. Initially, the arm is set to 0 degrees (due East). Applying each motor command sets a desired angle for the arm. For example, with a motor command of 90 degrees, the arm moves to 90 degrees (due North). After applying each command, we reset the position of the arm to the original position (0 degrees). The goal of the forward model is to predict the coordinates \( x \) and \( y \) of the arm’s end effector (i.e. the endpoint of the arm) given the motor command that is applied to the joint. The mathematical formula for the position of the end effector in the absence of any changes is shown in Eq. 4.7 and 4.8 where \( \theta \) indicates the motor command (desired angle) and \( l \) indicates the length of the arm, which is 1 in our experiments.

\[
x = l \cdot \cos(\theta)
\]  

\[
y = l \cdot \sin(\theta)
\]  

We test two different types of changes: fatigue and damage. For each sequence of motor commands, these two phenomena may happen with an independent probability of 0.5. Thus, there are four possible conditions: neither fatigue nor damage (i.e. normal conditions), fatigue only, damage only, and both fatigue and damage. The details of how fatigue and damage are implemented are explained next.

83
Fatigue

In idealized situations (e.g. simulation), when we send a motor command to the joint of a robot, it performs that command without error. Muscle fatigue is a well-known phenomenon for humans; when our muscles are very tired, we lose some of our control over them [133]. Robot motors may similarly suffer from fatigue because of extreme motor temperatures or batteries dropping in voltage [134]. Here we simulate fatigue for robot motors by setting motors to a fraction of the motion that would have occurred due to the command in normal conditions. For example, in the presence of fatigue, if we give a command of 30 degrees to the joint, the arm does not move all the way from its initial position of 0 degrees to the desired angle, meaning the final angle will be less than 30 degrees. Fatigue increases over time, meaning that the fraction of the requested angle that is achieved decreases over time (Eq. 4.9).

\[ \hat{\theta} = (1 - \text{fatigue}) \cdot \theta \]  

In Eq. 4.9, \( \theta \) is a motor command and \( \hat{\theta} \) is the actual angle that the arm moves to. Here we test with two different types of fatigue: linear fatigue and step-wise fatigue (Fig. 4.4). With linear fatigue, we increase the amount of fatigue linearly to 50% by the end of the motor command sequence (Eq. 4.10). In step-wise fatigue we increase the fatigue in a fixed number of discrete steps (here 10) to 45% (Eq. 4.11). In Eq. 4.10 and 4.11, \( N \) indicates the length of the command sequences and \( i \) is the index of a motor command in the sequence.

\[ \hat{\theta}_i = \theta_i \ast (1 - \frac{i}{2N}) \]  

\[ \hat{\theta}_i = \theta_i \ast (\lfloor \frac{10i}{N} \rfloor \ast 0.05) \]  

Damage

The second type of drastic change a robot may have to cope with is extreme damage to a motor (e.g. the motor stops working entirely). We simulate that by making damage
randomly occur at some time in the second half of the motor sequence, after which the robot’s joint is not able to follow any command and remains fixed at the angle it was set to when the damage occurred. Because our simple robot has only one motor, when damage occurs the position of the end effector that the forward model is trying to predict remains fixed. These dynamics are described in the following equation:

\[
\hat{\theta}_i = \begin{cases} 
\theta_i & i < t \\
\theta_{t-1} & i \geq t 
\end{cases} \tag{4.12}
\]

where \(i\) is the index of a motor command in the sequence and \(t\) is the instant that the damage occurs. Damage overwrites fatigue, therefore, if we have both damage and fatigue, we ignore the fatigue and just keep the last angle.

### 4.3.2 Algorithm Details

Forward models must be able to detect if their predictions are incorrect in order to improve. If we only input the motor command to the forward model, there will be no way for the forward model to notice changes. In order to enable the forward model to improve itself if it makes errors, we also input the differences between the predicted coordinates of the end effector and the actual coordinates (i.e. error signals) into the network (Eq. 4.13 and 4.14).

\[
\text{error}_x = |x_{\text{actual}} - x| \tag{4.13}
\]

\[
\text{error}_y = |y_{\text{actual}} - y| \tag{4.14}
\]

\[
\text{error} = \text{error}_x + \text{error}_y \tag{4.15}
\]

Therefore, our forward models have 3 inputs (a motor command, the error signal of \(x\), and the error signal of \(y\)) and 2 outputs (predicted \(x\) coordinate and \(y\) coordinate of the end effector). In order to provide the error signals for the network, we have three steps for inputting each motor command into the network (Fig. 4.5). In the first step, we input the
motor command into the network (for this step, the error inputs are set to zero) and compute the predictions. In the second step, we calculate the prediction error signals. In the third step, we input both the motor command and the error signals again. The fitness function only considers the predictions of the first step. The purpose of the third step is to provide the network the magnitude of its prediction errors so its learning algorithm can modify the weights of the model to improve future predictions. Any prediction errors that result in this third step do not affect fitness.

We evolve networks with the NeuroEvolution of Augmenting Topologies (NEAT) algorithm [115] because it is widely used and because it can evolve both the structure and the weights of networks. Because NEAT has been described in detail in numerous previous papers, we only describe it briefly here, for details about it, see [115]. NEAT starts with a population of simple two-layer NNs that become more complex over time via mutations that add nodes and connections. Solutions are also refined by mutations that change connection weights. Like all evolutionary algorithms, NEAT needs a criterion for comparing the performance of various candidate solutions (i.e. a fitness function), which is described in the next section.

### 4.3.3 Fitness Evaluation

The fitness for a network is a score summarizing its performance over its entire lifetime. During its lifetime, each network is exposed to 10 sequences of motor commands. Each sequence is composed of 500 motor commands. Each motor command is an angle the motor is commanded to move to. Each individual motor command involves a three-step command implementation process (outlined in Fig. 4.5) that involves running the NN twice (the first pass provides the command and no error signals, and the second provides the command again along with the error signals from the first pass). The fitness function only incorporates errors produced after the first pass (Fig. 4.5).

Each sequence is randomly allocated to one of four conditions: (1) no fatigue-no damage, (2) fatigue-only, (3) damage-only, (4) fatigue and damage. Thus, every network has a chance of encountering any of these conditions and must handle it appropriately. The existence of
different conditions, including a condition without damage or fatigue, prevents the network from adopting simple open-loop hacks to solve the problem, such as approximately solving fatigue via a constant decay schedule.

Thus, in total, each network is run twice per command, for 10 sequences of 500 motor commands each, for a total of \(2 \times 500 \times 10 = 10,000\) network updates per lifetime. Because only the error of the first pass affects fitness, the error vector is of length \(1 \times 500 \times 10 = 5000\) for both \(x\) and \(y\) outputs, producing a total error vector of length 10,000 per fitness evaluation. Those 10,000 numbers are summed to produce the total error, and fitness is its negative (Eq. 4.16).

\[
fitness = - \sum_{i=1}^{\text{No.ofSeq.}} \sum_{j=1}^{N} \text{error}
\]

(4.16)

Adding behavioral diversity is known to be beneficial for finding better solutions to deceptive problems \([135, 136]\). We thus use a multi-objective evolutionary algorithm with two objectives: increasing fitness and increasing behavioral diversity of the population. Specifically, experiments are performed with the multi-objective Non-dominated Sorting Genetic Algorithm II (NSGA-II) \([137]\) as implemented in the Sferes framework \([138]\). The parameters for running the algorithm are presented in Table 4.1.

We measure behavioral diversity as the average of behavioral distance between each pair of individuals in the population. Specifically, for each individual, we store all of the outputs of the first pass of the network for both \(x\) and \(y\) for all motor commands from all sequences (thus, 10,000 outputs in total). We then compute the Euclidean distance between these arrays to obtain pairwise distances between individuals. Each individual’s diversity is then its mean distance to every other member of the population.

### 4.4 Experiments and Results

In order to evaluate the effectiveness of the neuromodulation technique for building adaptable forward models, we compare three models: the static NN forward model, the plain Hebbian plastic NN forward model, and the neuro-modulation-based forward model. We design two
Table 4.1: The evolutionary algorithm parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Connection Rate</td>
<td>0.2</td>
</tr>
<tr>
<td>Add Neuron Rate</td>
<td>0.2</td>
</tr>
<tr>
<td>Remove Neuron Rate</td>
<td>0.2</td>
</tr>
<tr>
<td>Remove Connection Rate</td>
<td>0.2</td>
</tr>
<tr>
<td>Change Type of a Neuron Rate</td>
<td>0.2</td>
</tr>
<tr>
<td>Change Connection Weight Rate</td>
<td>0.2</td>
</tr>
<tr>
<td>Change Activation Function Rate (Sigmoid, Tanh, Linear, Sin., Cos.)</td>
<td>0.2</td>
</tr>
<tr>
<td>Probability of a Neuron Being Modulatory</td>
<td>0.3</td>
</tr>
<tr>
<td>Hebbian Learning Rate</td>
<td>0.05</td>
</tr>
<tr>
<td>Minimum Value of a Connection weight</td>
<td>-5</td>
</tr>
<tr>
<td>Maximum Value of a Connection weight</td>
<td>+5</td>
</tr>
<tr>
<td>Minimum Number of Neurons</td>
<td>5</td>
</tr>
<tr>
<td>Maximum Number of Neurons</td>
<td>20</td>
</tr>
<tr>
<td>Minimum Number of Connections</td>
<td>3</td>
</tr>
<tr>
<td>Maximum Number of Connections</td>
<td>100</td>
</tr>
</tbody>
</table>
Figure 4.6: Error of the best model from each treatment on example sequences from each damage condition. The static model performs well under normal conditions, but it fails to adapt to changes. The plain Hebbian is not able to effectively adapt to reduce error. In stark contrast, forward models evolved with neuromodulation exhibit close to the theoretical minimum error: they have almost no error throughout, except right after major changes. They quickly adapt the model to account for such changes, reducing error back to near zero.

set of experiments: one in which fatigue is linear, and another in which fatigue progresses in a step-wise manner (Fig. 4.4). Each experiment is run 35 times.

Fig. 4.6 shows NN error values for the best model from each treatment on an example motor command sequence from each of the four different possible conditions: (1) normal (no fatigue or damage), (2) fatigue-only, (3) damage-only, and (4) fatigue-and-damage conditions. The static model is very accurate at normal sequences and damage-only sequences before the damage occurs, but it cannot adapt to the damage-induced change because it is unable to change its weights during its lifetime. The strategy evolution seems to adopt, if there is no error, is to predict as if there were no fatigue or damage for all sequences. When there is an error, it adjusts its output based on that error, but cannot utilize that information to reduce error anywhere near zero. The plain Hebbian forward model can change its weights during its lifetime, but it does not evolve an intelligent way to make changes at appropriate times (e.g. only after damage) to make extremely accurate predictions. In stark contrast, the neuromodulation-controlled forward model has low errors in throughout all four conditions, except for a few updates after extreme damage (spikes in the error plots) as it rapidly adapts its model to account for the sudden change (Fig. 4.6). While less visually apparent,
Figure 4.7: Forward models evolved with neuromodulation are significantly better at predicting the outcomes of their actions and adapting to changes such as damage and fatigue. In experiments with linear fatigue (a) and stepwise fatigue (b), the error produced by neuromodulation models is significantly lower across evolutionary time than that produced with static models or Hebbian learning. Below the plot, an asterisk is shown if a significant difference exists between each pair of treatments at a threshold of \( p < 0.05 \). Often, \( p \) values are far lower than this threshold. For example, comparing mean performance in the final generation, the \( p \) value is \( p < 10^{-13} \) comparing neuromodulation to the other treatments. The performance differences between static models and Hebbian models is slight. In the linear fatigue case (a) it is not significant after initial generations. In the stepwise fatigue case (b), the static model actually significantly outperforms the Hebbian model. The overall conclusion is that, even when plasticity is required, Hebbian plasticity is not powerful enough. To truly realize the benefits of plasticity, a more powerful form of plasticity like neuromodulation is required. Plotted are the median and range of fitness values (minimum and maximum) across 35 independent runs for the static, plain Hebbian, and neuromodulation forward models. All \( p \) values here and elsewhere in the paper are computed by a Mann-Whitney U test.

The quantitative results averaged over all runs support these qualitative observations. In both sets of experiments (with linear and step-wise fatigue), forward models evolved with neuromodulation significantly outperform the models evolved in both the Hebbian and static treatments across evolutionary time (Fig. 4.7, \( p < 0.05 \)). That trend holds when comparing final populations (\( p < 10^{-13} \)).

Overall, models evolved with plain Hebbian learning do not perform substantially bet-
ter than static models. In the set of experiments with linear fatigue, there is no significant
difference between the two treatments after initial generations (Fig. 4.7, $p > 0.05$). Surpris-
ingly, in the set of experiments with step-wise fatigue, the static model actually significantly
outperforms the Hebbian model, although the difference is small (Fig. 4.7, $p < 0.05$). These
results underscore the importance of neuromodulation when evolving plastic neural network
controllers. In situations when plasticity is required, Hebbian plasticity alone may not be a
powerful enough form of plasticity. Instead, to truly see the benefits of plasticity, evolution
must be given the power to regulate learning in different situations based on the data.

4.5 Conclusion

In many applications, robots need to deal with various unforeseen changes in their body and
their environment. Predicting all possible changes in order to handle all possible circum-
stances is infeasible. Therefore, modern robots need accurate, adaptable forward models to
adapt themselves to various changes, such as changes to the environment or their bodies like
damage. Such models enable robots to simulate dangerous actions without performing them
to select an appropriate action.

This paper introduced a method for evolving adaptable forward models and tested the
hypothesis that neuromodulation improves the evolution of forward models. The data show
that neuromodulation produces significantly better forward models than Hebbian learning
and also significantly outperforms static models. The paper also demonstrates that Hebbian
learning is not able to provide the required plasticity required for adaptable forward models,
at least in our experiments.

Overall, the performance of the neuromodulation forward models is very impressive.
Error is near zero in a wide range of conditions, from when there is neither fatigue nor
damage to when there are both. As expected and required, error spikes just after a major
change, but the model quickly adapts to account for it. That said, our experiments are in the
very simple setup of a robot arm controlled by one motor. It will be an interesting question
for future research to what extent neuromodulation can enable the evolution of accurate,
adaptable forward models for more complex robots.

Additionally, it has previously been shown that neuromodulation can enable organisms to adapt to changing environments [18,128,129,139]. That work and our new results suggest that neuromodulation could enable the evolution of forward models that model environmental changes as well as changes to an agent’s body. Such models should ultimately improve the ability of robotics to plan ahead and select appropriate actions without costly, real-world trial and error.
Chapter 5

Discussion and Conclusion

In the last two decades, camera traps have become an important tool in ecology and biology studies, facilitating many scientific findings [1, 140, 141]. Camera-trap networks currently collect vast amounts of invaluable data and could collect much more. The main roadblock of exploiting their tremendous potential is the time and cost of extracting information from these obtained images, which is currently done manually by human experts or volunteers. Manually extracting information is time-consuming and expensive. Citizen scientists donate their time for free, but creating websites to organize their efforts, gathering enough volunteers, and managing citizen science projects requires a considerable amount of time and funding. In this dissertation, I proposed automating the extraction of knowledge from camera-trap images using advanced machine learning techniques. I experimented with different deep learning techniques for extracting information from these image datasets, including the testing of various DNN architectures. My results on the Snapshot Serengeti dataset (the largest existing labeled dataset of wild animals) and the North American Camera Trap Images dataset revealed that DNNs can perform well on camera-trap dataset, although accuracy is worse for rare species.

Perhaps most importantly, my results show that deep learning technology can save a tremendous amount of time for biology researchers and the human volunteers that help them by labeling images. In particular, for animal identification, our system can save 99.3% of the manual labor (over 17,000 hours per 3.2 million images) while performing at the same
96.6% accuracy level of human volunteers. I also showed that transfer learning (without active learning) could greatly help to projects with a limited number of labeled examples, although achieving a high accuracy still needs several hundred thousands of labeled images.

We also confirmed the effectiveness of our suggested algorithm by applying DNNs to another dataset from a different ecosystem (North America) and getting over 97.5% accuracy for species identification. In addition, we trained a single species detection model for wild pigs with 98.6% accuracy to show higher accuracy is reachable for studies that deal with only one kind of species.

In the second part of my work, I designed an active learning pipeline to help camera-trap projects with limited or no labeled images. I showed that the proposed pipeline can save more than 99.5% of the human effort needed to collect information from camera-trap projects, and thus significantly speed up consequent studies. In addition, saving a substantial amount of human labor can be purposed to other important scientific missions. Because manual information extraction is one of the major roadblocks of taking advantage of camera-trap networks, our pipeline could be a huge benefit for more easily deploying large camera-trap arrays to gather actionable information from wildlife in a relatively cheap and unobtrusive manner. Automating data extraction can thus dramatically reduce the cost to gather valuable information from wild habitats and will thus likely enable, catalyze, and improve many future studies of animal behavior, ecosystem dynamics, and wildlife conservation.

I also published my opensource software to train the DNNs for camera-trap projects to help many camera-trap projects in the world accelerate their efforts to automatically extract knowledge from the large datasets they are generating.

Moreover, the ability to automatically extract such information will encourage the creation and expansion of more camera-trap projects, which will enable science at scales previously unimaginable. Ultimately, reliably extracting data from camera-trap images could be the catalyst that transforms many fields of biology and conservation to BigData sciences, where vast amounts of data are collected and analyzed to significantly increase our understanding of the natural world and our ability to protect it.

In my dissertation, I not only suggested a remedy for automatically identifying, counting,
and describing animals in camera-trap images, but also showed the potential of deep learning techniques to automate several other tasks in the camera-trap domain, such as inferring the age of animals or their health conditions. For future work, there are two immediate directions of interest: 1) Researching more accurate, more transferable, more informative, and more data-efficient models 2) Harnessing deep learning to extract different kinds of information. For example, in many ecological studies researchers are not only interested in recognizing different species, but also need to distinguish between different individual animals of the same species. Currently, researchers use colorful tags or collars to identify individual, but this task could also be automated using advanced deep learning techniques. Although in my work I showed the potential of machine learning to help ecological studies, the full usage of this capacity deserves more research.
References


[122] F. J. Gomez and R. Miikkulainen, Robust non-linear control through neuroevolution. Computer Science Department, University of Texas at Austin, 2003.


