Automated Detection of Jackals and Foxes in the Arava Valley

by

Lillian Carleu

Liam Hall

Jacob Reiss

Jason Rockmael

Automated Detection of Jackals and Foxes in the Arava Valley

An Interactive Qualifying Project

submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

degree of Bachelor of Science

By

Lillian Carleu

Liam Hall

Jacob Reiss

Jason Rockmael

Date:

24 February 2023

Report Submitted to:

Dr. Nitzan Segev

Arava-Dead Sea Science Center

Professors Isa Bar-On and Erin Solovey; Graduate Student Co-advisor Tess Meier

Worcester Polytechnic Institute

ABSTRACT:

Monitoring animal populations has been a long-standing challenge in wildlife ecology. One focus of the Arava-Dead Sea Science Center (ADSSC) is monitoring jackal and fox populations as agricultural zones expand. Researchers at the ADSSC use camera traps for ecological monitoring, and manually sort the images into categories. This process takes at least 12 hours of work per week. To reduce the workload of researchers at the ADSSC, their image sorting process was automated using machine learning and integrated into the current workflow. Over 90% accuracy in classifying “jackals”, “foxes”, “other”, and “empty” was achieved using our machine learning algorithm. A comprehensive user manual was created to allow researchers at the ADSSC and future users to utilize the software with ease.

ACKNOWLEDGEMENTS:

We would like to thank Dr. Nitzan Segev for her cooperation and contributions to the project. She is very passionate about her work and helped motivate the team to produce high quality results. Without the data that she provided, this project would not have been achievable. Special thanks goes to Liri Koplevich, Dr. Segev's research assistant, for his help with gathering data and his witty humor. We would also like to thank our advisors, Dr. Isa Bar-On, Dr. Erin Solovey, and Tess Meier. Their guidance throughout the project helped us stay on track. Extra thanks goes to Dr. Solovey for her consultation regarding the code for the project. Lastly, we would like to thank Kibbutz Ketura, the Arava Institute, and the Arava-Dead Sea Science Center for their hospitality and enthusiasm for our project.

AUTHORSHIP PAGE:

Paper Section	Primary author(s)	Primary editor(s)
Abstract	Lillian Carleu	All
Chapter 1: Introduction		
1.1 Arava-Dead Sea Science Center	Lillian Carleu	Lillian Carleu
1.2 The ADSSC's Current Workflow for Monitoring Large Carnivore Populations	Lillian Carleu, Jason Rockmael	Lillian Carleu
1.3 Goal Statement and Objectives	All	All
Chapter 2: Background		
2.1 The Arava Valley	Lillian Carleu	Lillian Carleu
2.2 Standard Image Processing Workflow of Ecologists	Lillian Carleu, Jacob Reiss	Lillian Carleu
2.3 Timelapse Software	Liam Hall	Lillian Carleu
2.4 MegaDetector and EcoAssist	Jason Rockmael	Lillian Carleu
2.5 Machine Learning and Artificial Intelligence	Jacob Reiss, Jason Rockmael	Lillian Carleu
2.6 Transfer Learning with ImageNet	Jacob Reiss	Lillian Carleu
2.7 AI Hyperparameters	Jason Rockmael	Lillian Carleu
2.8 Implications of Machine Learning	Lillian Carleu	Lillian Carleu
2.9 Use Cases of the Arava-Dead Sea Science Center	Lillian Carleu, Jason Rockmael	Lillian Carleu
Chapter 3: Automating Image Classification Overview	Lillian Carleu, Jacob Reiss	Lillian Carleu, Jacob Reiss
Chapter 4: Determining Image Sorting	Lillian Carleu, Jason	Lillian Carleu,

Categories	Rockmael, Jacob Reiss	Jacob Reiss
Chapter 5: Data Collection and Curation		
5.1 Data Collection	Jason Rockmael	Lillian Carleu, Jacob Reiss
5.2 General Dataset Description	Jacob Reiss	Lillian Carleu
5.3 Dataset Preprocessing	Jacob Reiss	Lillian Carleu
5.4 Dataset Labeling	Jacob Reiss	Lillian Carleu
5.5 Dataset Overview	Jacob Reiss	Lillian Carleu
Chapter 6: AI Model for Detection of Jackals and Foxes		
6.1 MegaDetector Results	Jason Rockmael	Lillian Carleu
6.2 Implementing and Training the Model	Lillian Carleu, Jacob Reiss	Jacob Reiss, Jason Rockmael
6.3 Best Performance	Jason Rockmael	Jacob Reiss
6.4 Optimization Experiment Overview and Methodology	Lillian Carleu, Jacob Reiss, Jason Rockmael	Lillian Carleu, Jacob Reiss, Jason Rockmael
6.5 Epoch Experimentation	Jacob Reiss	Lillian Carleu, Jason Rockmael
6.6 Variation of Steps per Epoch Experimentation	Jacob Reiss	Lillian Carleu, Jason Rockmael
6.7 Variation of Validation Steps Experimentation	Jacob Reiss	Lillian Carleu, Jason Rockmael
6.8 Variation of Optimizer	Jason Rockmael	Jacob, Lillian Carleu
6.9 Variation of Activation Function	Jason Rockmael	Jacob Reiss
6.10 Variation of Unit Size	Jason Rockmael	Jacob Reiss
Chapter 7: Integration with Research Workflow		

7.1 Integration with Timelapse	Liam Hall, Jacob Reiss	Lillian Carleu, Jason Rockmael
7.2 Finalizing an Application and User Manual	Lillian Carleu, Jason Rockmael	Lillian Carleu, Jacob Reiss
7.3 Use by the Arava-Dead Sea Science Center	Jason Rockmael	Lillian Carleu
7.4 Hardware Requirements and Recommendations	Jason Rockmael	Liam Hall, Jacob Reiss
Chapter 8: Conclusions and Recommendations	Lillian Carleu, Liam Hall	Lillian Carleu, Jason Rockmael

OTHER CONTRIBUTIONS

Person	Contribution(s)
Lillian Carleu	Team manager, researched extensively, head of writing, labeled some of the dataset, worked with design elements.
Liam Hall	Wrote the user manual, integrated MegaDetector with the application, labeled some of the dataset.
Jacob Reiss	Designed and programmed the application, integrated Timelapse with application, integrated MegaDetector with the application, ran optimization experiments and tests, wrote the user manual, helped sort labeled data, did citation review, helped sponsor in integrating our software.
Jason Rockmael	In charge of communications, labeled the majority of the dataset, integrated MegaDetector with the application, general optimization experiments and tests, confusion matrices, team GitHub organizer, helped write user manual, and helped sponsor in integrating our software.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Arava-Dead Sea Science Center

The Arava-Dead Sea Science Center (ADSSC) is a research institution that operates along the Dead Sea and the entire Arava Valley, a region that covers approximately 20% of the area of the State of Israel. The valley contains an array of extreme geographical, physical, and climatic conditions, displaying unique flora and fauna that do not exist anywhere else in the world (ADSSC, n.d.). The ADSSC conducts research based on cross-border scientific cooperation and sustainable development, focusing on climate change, infrastructure, water, biodiversity, and sustainable agriculture in a hyper-arid climate (ADSSC, n.d.). Much of the research involves monitoring the impact of agricultural expansion on both native and invasive flora and fauna.

As agricultural zones expand in the Arava Valley, these zones serve as oases for animals, providing easy access to both food from crops and water from irrigation pipes (Barocas et al., 2018). In a study conducted by Barocas, et al. on the behavioral adaptations of large carnivores to human activity in an extremely arid landscape, it was shown that "driven by availability of food subsidies, large carnivore populations are increasingly inhabiting the vicinity of humans" (Barocas, et al., 2018). Researchers at the ADSSC have interest in these patterns and have noted an increase in the populations of large carnivores where food subsidies are more abundant.

Jackals and foxes are the largest and most harmful of the large carnivores in this area, because of the damage they do to crops by trampling them while playing, and damage to irrigation pipes by chewing through them, which wastes water and puts crops at risk (Figure 1). In addition to this, these animals prey on livestock and they pose a health risk to communities because they carry diseases that are dangerous and transferable to humans and domesticated animals, such as rabies.[1] Currently, jackals are classified as a pest in Israel (Moran, 2003).

[1] N. Segev, personal communication, January 23, 2023

Mori Chen (2018) Retrieved from N. Segev

Mori Chen (2018) Retrieved from N. Segev

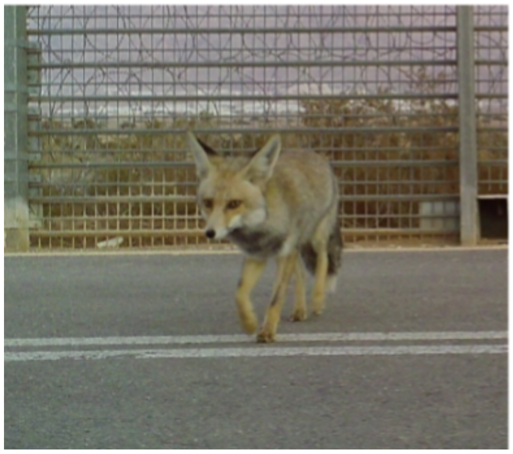
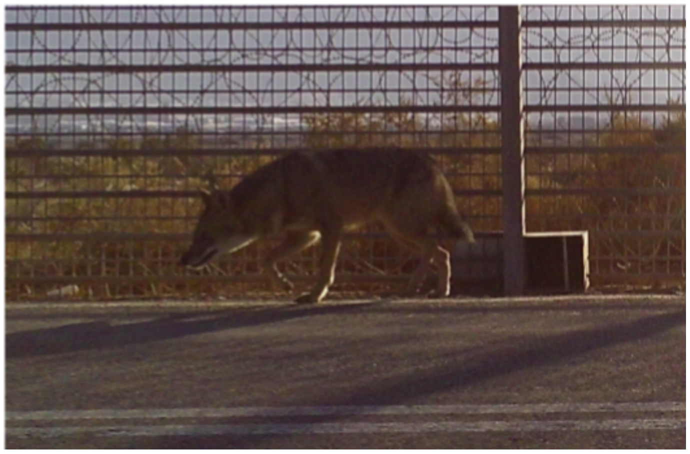



Figure 1: Top: Damage to a pumpkin patch and irrigation system caused by jackals Bottom: Jackal and fox crossing the Israeli-Jordanian border

1.2 The ADSSC's Current Workflow for Monitoring Large Carnivore Populations

Currently, the researchers at the ADSSC set up trail cameras twice per season to photograph and monitor wildlife. Poles are hammered into the ground across from points of interest and motion-sensing trail cameras are attached. The cameras are left for two weeks, after which the images are downloaded to a laptop and sorted. Researchers have interest in sorting these images to gain quantitative data about specific species' populations and behavioral patterns.

The manual sorting process for these images involves going through the images individually by hand, and determining which animal, if any, is in each image. Researchers label the images with the season and time of day visually, as well as the number of animals present in the photo using a software called Timelapse. It provides quick dropdown menus to classify each of the images and has several display modes. Additionally, this software can quickly sort the data in a table and export it as a comma-separated value (CSV) file which allows for easy transfer to external applications.

At the ADSSC, approximately 800 photos are taken per day, and the researchers label images at a rate of approximately 420-480 images per hour. This is a time-consuming process, as it takes over 11 hours of work per week to process a week's worth of photos, taking away time from other research tasks.[2] This process does not factor in misclassification due to human error and the additional time consumed from rechecking photos. Furthermore, teaching others this manual labeling process is time-consuming.

1.3 Goal Statement and Objectives

The goal of our project is to develop an automated system to detect and record jackal and fox activity in the Arava Valley to reduce the workload for researchers at the Arava-Dead Sea Science Center (ADSSC). To achieve this goal, our team developed the following objectives: 1) identify clear sorting categories for photographs taken by the ADSSC to categorize invasive jackals and foxes, 2) automate the detection and classification of foxes and jackals using machine learning, 3) optimize our software to have high accuracy in classifying jackals and foxes, and 4) create a practical process, as well as user manual, to integrate our software into the ADSSC's current workflow. Moving forward, the ADSSC is considering the construction of an automated deterrence system for these mammals. They could use our software as a foundation for such a project, in order to determine whether an object in front of a camera should be deterred or not.

[2] N. Segev, personal communication, January 23, 2023

CHAPTER 2: BACKGROUND

2.1 The Arava Valley

The Arava Valley is a hyper-arid zone in the Negev desert (Portnov & Safriel, 2003). Desert ecosystems have low biological activity and biodiversity and it is typically difficult for non-native species to survive (Faragalla, A., 1988).

Agricultural zones are now expanding at unforeseen rates due to technological advances. Highly productive agroecosystems in the desert are an intrusion to the natural ecosystem. Irrigation systems in these zones create favorable environments for pest species: the continuity of oases and crops creates refuges for pests to multiply (Faragalla, A., 1988). Agricultural zones have human communities in close proximity, where large quantities of food scraps are dumped. Predators that would not be able to survive on natural prey in the Negev alter their dietary preferences to survive, and rely heavily on the availability of garbage. The availability of human-provided resources to predators results in behavioral changes and trophic cascades (Newsome, et al., 2015). A study by Greenville et al. showed that the strongest effect on prey populations in the desert ecosystem is suppression from introduced predators (Greenville, Wardle, & Dickman, 2017). A study showed that foxes are much more commonly occuring and active at locations close to agricultural zones (Shanas, Shapira, & Sultan, 2008).

2.2 Standard Image Processing Workflow of Ecologists

Monitoring animal populations has been a long-standing challenge in wildlife ecology (Bayne, et al., 2015). However, with the improvement of remote-sensing and other technical capabilities, camera traps have become widely adopted by researchers to survey wildlife distribution, abundance, and behavior (Berger-Tal & Lahoz-Monfort, 2018). A trail camera refers to a motion-sensing camera utilized for field research. Camera trapping does have sources of sampling error such as imperfect detection (Bayne, et al., 2015). These cameras take thousands of photos, many of which do not contain animals (Andrews, et al., 2017). Even so, each image must be analyzed to determine what species is in the photo. This can take researchers days to process, depending on how many cameras they use and how much movement occurs in these areas. Thus, researchers have sought different ways of making the process more efficient.

In the case of the Wildlife Spotter Project, citizen science was used to classify images, with specific categories being provided for the volunteers (Andrews, et al., 2017). While this achieved 96% accuracy, it still took days to process all of the data, and required willing volunteers to manually classify sets of images.

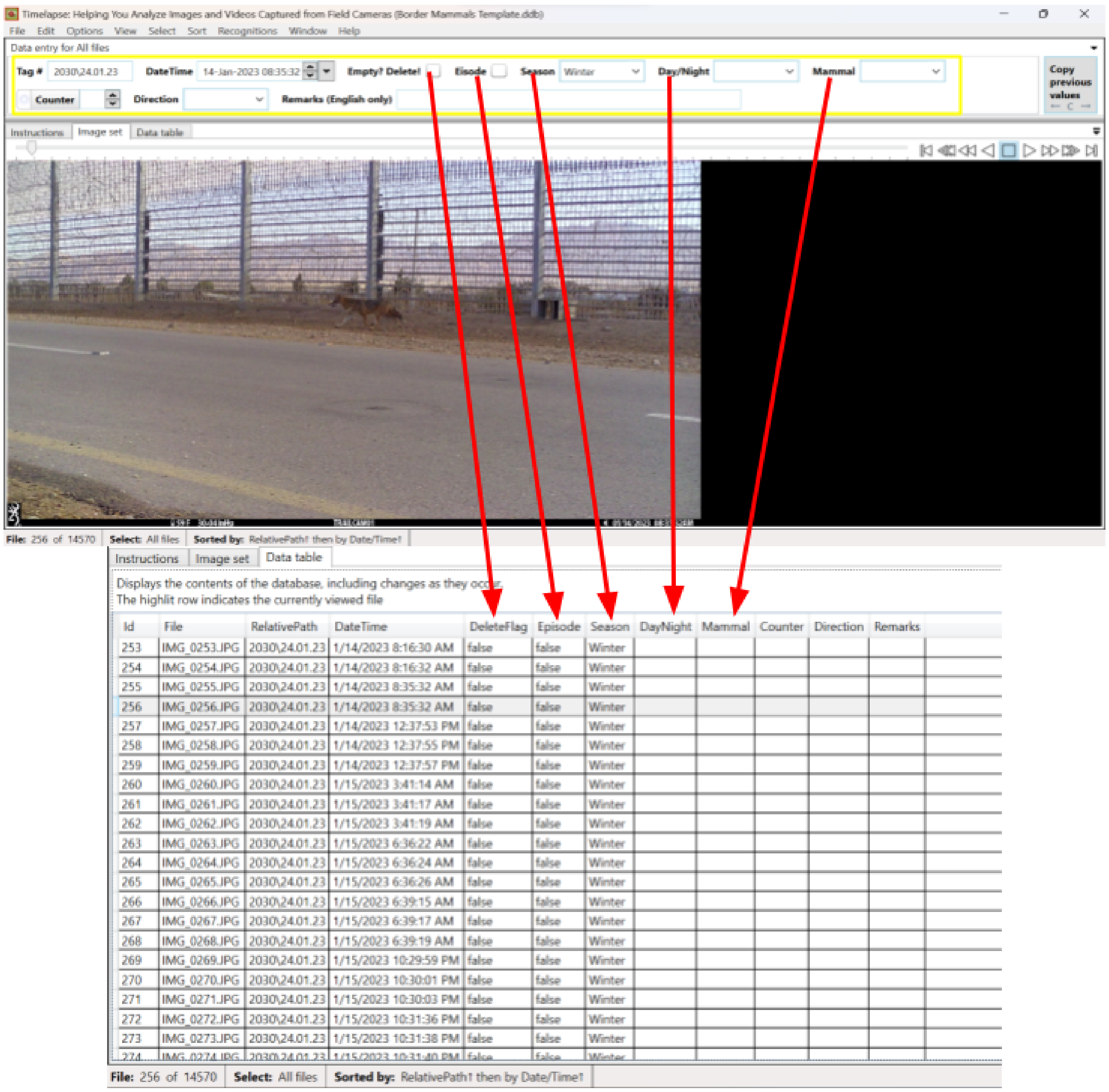
To speed up this process, ecologists have also looked into machine learning, a field of artificial intelligence that involves training a program to perform a specific task. Many studies have shown that neural networks, an implementation of machine learning, have achieved 90-98% accuracy at classifying images, greatly reducing the workload on ecologists (Andrews, et al., 2017). A study investigating the performance of various machine learning models on the Wildlife Spotter dataset was found to be up to 96% accurate (Andrews, et al., 2017). Another study that used multiple neural network implementations on the Northeast Tiger and Leopard National Park dataset found that neural networks could be up to 88% accurate at making predictions on videos (Chao, et al., 2022). Another benefit of neural networks is that they take significantly less time to process images.

2.3 Timelapse Software

In the ADSSC's current workflow, a software called Timelapse is used to analyze and label the image data of the local ecology.[3] Timelapse provides a graphical user interface that allows the user to label the wildlife in the image using a dropdown list of animals (Timelapse, 2021). Other information can be added to the images, including the season, whether it is day or night, and a counter for how many of the specified animals are in the image as shown in Figure 2. This information is automatically formatted into a data table as shown in Figure 2, which can be exported and used elsewhere. Sorting within the data table is also supported, which allows the user to view various columns of the data in either ascending or descending order. The number of images that the user views can be specified, from one to twenty-four images at a time. Images can also be 'auto-played', or cycled through automatically, and the speed of this can be varied. Comma-separated values (CSV) files can be imported into and exported out of Timelapse. This

[3] N. Segev, personal communication, January, 2023

feature helps the user to integrate Timelapse into their workflow, as CSVs can be read by spreadsheet applications, such as Excel and Google Sheets.


Timelapse: Helping You Analyze Images and Videos Captured from Field Cameras (Border Mammals Template.ddb)
File Edit Options View Select Sort Recognitions Window Help
Data entry for All files
Copy previous values
Instructions Image set Data table

Tag # 2030\24.01.23 DateTime 14-Jan-2023 08:35:32 Empty? Delete! Eisode Season Winter Day/Night Mammal
Counter Direction Remarks (English only)

File: 256 of 14570 Select: All files Sorted by: RelativePath↑ then by Date/Time↑

Displays the contents of the database, including changes as they occur.
The highlit row indicates the currently viewed file

Id	File	RelativePath	DateTime	DeleteFlag	Episode	Season	DayNight	Mammal	Counter	Direction	Remarks
253	IMG_0253.JPG	2030\24.01.23	1/14/2023 8:16:30 AM	false	false	Winter					
254	IMG_0254.JPG	2030\24.01.23	1/14/2023 8:16:32 AM	false	false	Winter					
255	IMG_0255.JPG	2030\24.01.23	1/14/2023 8:35:32 AM	false	false	Winter					
256	IMG_0256.JPG	2030\24.01.23	1/14/2023 8:35:32 AM	false	false	Winter					
257	IMG_0257.JPG	2030\24.01.23	1/14/2023 12:37:53 PM	false	false	Winter					
258	IMG_0258.JPG	2030\24.01.23	1/14/2023 12:37:55 PM	false	false	Winter					
259	IMG_0259.JPG	2030\24.01.23	1/14/2023 12:37:57 PM	false	false	Winter					
260	IMG_0260.JPG	2030\24.01.23	1/15/2023 3:41:14 AM	false	false	Winter					
261	IMG_0261.JPG	2030\24.01.23	1/15/2023 3:41:17 AM	false	false	Winter					
262	IMG_0262.JPG	2030\24.01.23	1/15/2023 3:41:19 AM	false	false	Winter					
263	IMG_0263.JPG	2030\24.01.23	1/15/2023 6:36:22 AM	false	false	Winter					
264	IMG_0264.JPG	2030\24.01.23	1/15/2023 6:36:24 AM	false	false	Winter					
265	IMG_0265.JPG	2030\24.01.23	1/15/2023 6:36:26 AM	false	false	Winter					
266	IMG_0266.JPG	2030\24.01.23	1/15/2023 6:39:15 AM	false	false	Winter					
267	IMG_0267.JPG	2030\24.01.23	1/15/2023 6:39:17 AM	false	false	Winter					
268	IMG_0268.JPG	2030\24.01.23	1/15/2023 6:39:19 AM	false	false	Winter					
269	IMG_0269.JPG	2030\24.01.23	1/15/2023 10:29:59 PM	false	false	Winter					
270	IMG_0270.JPG	2030\24.01.23	1/15/2023 10:30:01 PM	false	false	Winter					
271	IMG_0271.JPG	2030\24.01.23	1/15/2023 10:30:03 PM	false	false	Winter					
272	IMG_0272.JPG	2030\24.01.23	1/15/2023 10:31:36 PM	false	false	Winter					
273	IMG_0273.JPG	2030\24.01.23	1/15/2023 10:31:38 PM	false	false	Winter					
274	IMG_0274.JPG	2030\24.01.23	1/15/2023 10:31:40 PM	false	false	Winter					

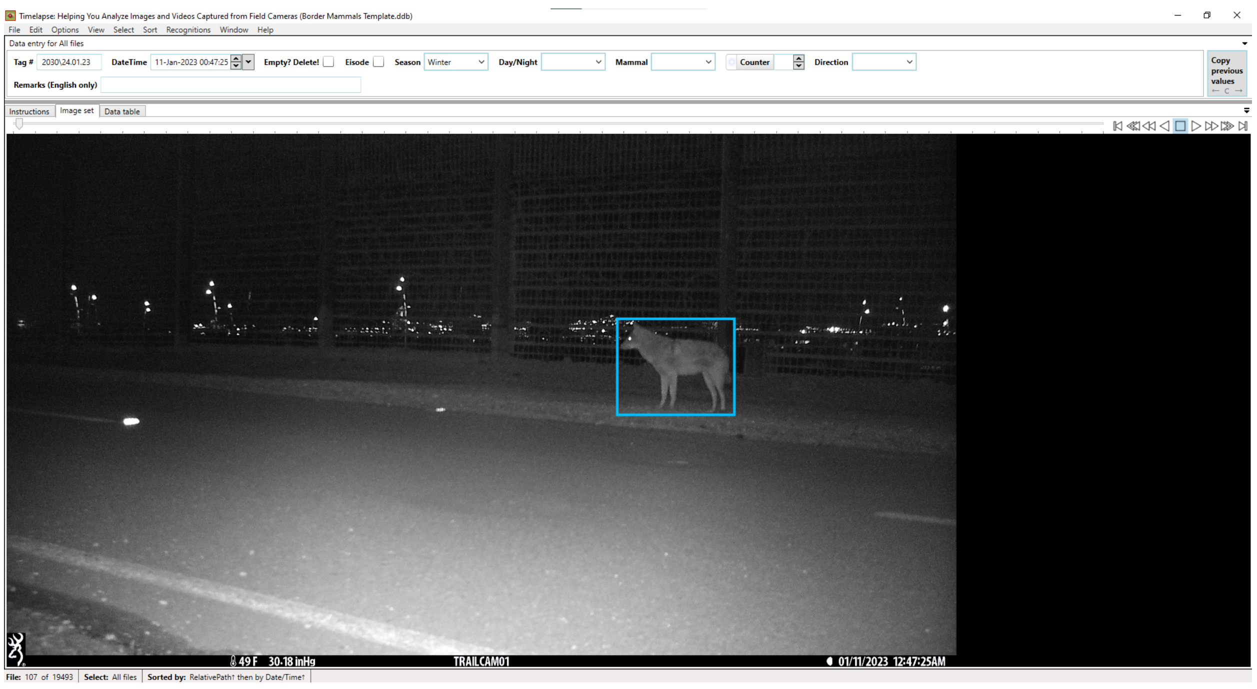
File: 256 of 14570 Select: All files Sorted by: RelativePath↑ then by Date/Time↑

Figure 2: Timelapse being used to process camera trap data of a jackal crossing the Israeli-Jordanian border

2.4 MegaDetector and EcoAssist

The most time consuming portion of the current categorization process is manually sorting through thousands of images and differentiating empty images from those that contain animals or other objects. Microsoft's MegaDetector software (Appendix E) classifies animals, people, vehicles, and empty images automatically. MegaDetector has been used successfully in conjunction with Timelapse by ecologists for past research, and there is a full guide (Appendix D) on how to import MegaDetector results into Timelapse. MegaDetector is also widely used because of its fast runtime on most NVIDIA graphics cards. One concern with this software is that it must be run in Windows Command Prompt, so users that are not familiar with shell scripts will have difficulty running the software. To address this issue, researchers use EcoAssist (Appendix E) as well. EcoAssist is a graphical user interface developed by an independent wildlife researcher to make the MegaDetector software more user friendly for other researchers that are less familiar with Windows Command Prompt.

With EcoAssist installed, the user chooses the desired folder for classification within the graphical user interface, runs the MegaDetector software on the chosen dataset, and has the option to post-process the data to sort the data automatically. After MegaDetector has completed its run, it automatically outputs a JavaScript Object Notation (JSON) file that contains data for blue, red, and white bounding boxes that correspond to animal, human, and vehicle, respectively. Additionally, JSON files produced by MegaDetector contain confidence values for its detections on each image, which correspond to the certainty of the algorithm. These JSON files can then be imported into Timelapse and the images in the dataset can be sorted by both the classification and confidence values. If objects are detected, the image will have a blue, red, or white bounding box around the objects of interest (Figure 3).


Timelapse: Helping You Analyze Images and Videos Captured from Field Cameras (Border Mammals Template.ddb)

File Edit Options View Select Sort Recognitions Window Help

Data entry for All files

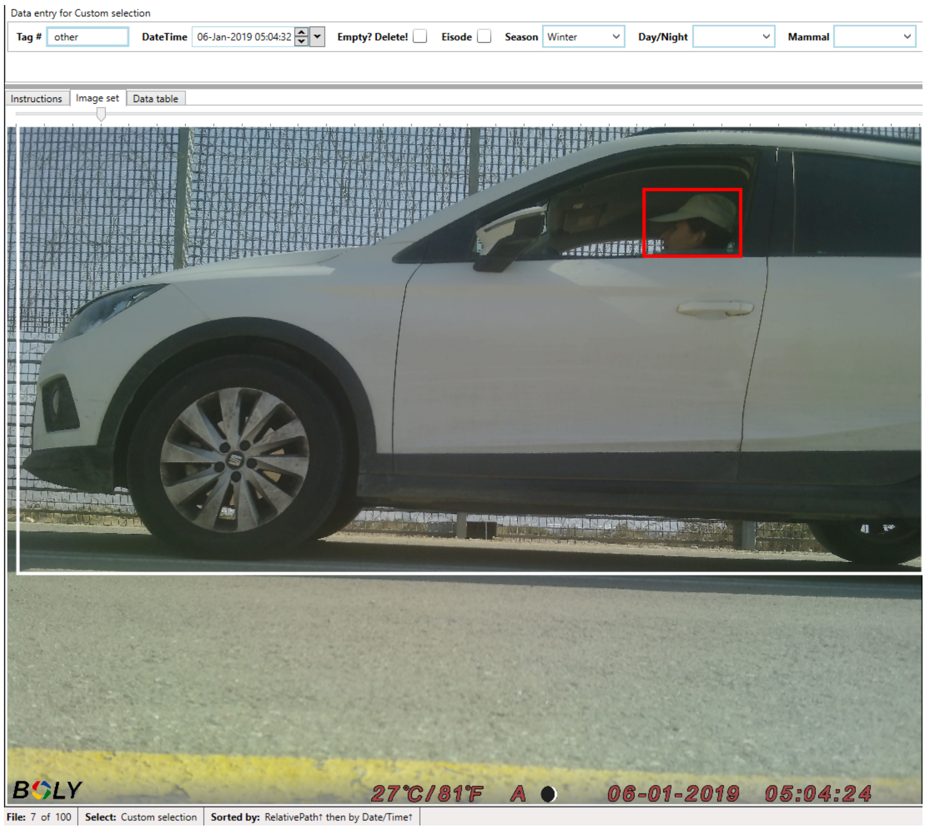
Tag # 2030\24.01.23 DateTime 11-Jan-2023 00:47:25 Empty? Delete! Eisode Season Winter Day/Night Mammal Counter Direction

Remarks (English only)

Instructions | Image set | Data table

49 F 30.18 inHg TRAILCAM01 01/11/2023 12:47:25AM

File: 107 of 19493 | Select: All files | Sorted by: RelativePath↑ then by Date/Time↑


Data entry for Custom selection

Tag # other DateTime 06-Jan-2019 05:04:32 Empty? Delete! Eisode Season Winter Day/Night Mammal

Instructions | Image set | Data table

BOLY 27°C/81°F A 06-01-2019 05:04:24

File: 7 of 100 | Select: Custom selection | Sorted by: RelativePath↑ then by Date/Time↑

Figure 3: Image showing bounding boxes around an animal, person, and vehicle in Timelapse

2.5 Machine Learning and Artificial Intelligence

Machine learning is a field of artificial intelligence (AI) that uses computer systems which learn and adapt by using algorithms and statistical models to analyze and draw inferences from patterns in data (Baker, Herbert Chan, & Nichols, 2019). A neural network is analogous to a brain, where each component in the neural network represents a neuron.

Neural networks come in many forms and are highly configurable. In the machine learning field, convolutional neural networks (CNNs) are widely used for image classification (Andrews, et al., 2017). CNNs can be trained on large quantities of image data that have been pre-processed in order to recognize patterns that appear in the data. Image processing CNNs take in an image as an input, process it through the layers of the network, and then output a classification with a confidence value. A more detailed breakdown of CNNs can be found in Appendix A.

Optimizing CNN models is also highly important. For the software to be useful to users, a model has to be fast and accurate in its classification.

2.6 AI Hyperparameters and Architecture

There are several variables in the code of an AI model that affect the AI's classification. These variables are called "hyperparameters" and are manually set by the developer to improve the accuracy and loss values of the model. Accuracy in image recognition refers to the AI's ability to correctly categorize photographs. Accuracy is a percent value represented as a decimal of how many images the AI correctly identifies. Loss measures the magnitude of error the algorithm makes when analyzing a photo in the dataset. Loss is not a percentage and can have any positive value. Loss values under one are typically considered "good" values, with values closer to zero being the best.

One hyperparameter in CNNs is the number of epochs. An epoch refers to one iteration through all the training data. While the number of epochs is directly proportional to the accuracy, too many epochs will cause the model to overfit since it has seen the same data too many times. Overfitting means that the artificial intelligence would have a harder time making predictions on images outside of the training set.

Another hyperparameter to consider is the number of steps per epoch in CNNs. A step refers to the number of times a batch is fed into a network per epoch. A batch refers to a set of images that are processed together. Steps per epoch is also directly proportional to the accuracy at the expense of an increased runtime, until overfitting occurs.

In order to decrease the loss values of a model, an optimizer is used. There are several different optimizers such as Adam, Adamax, and Adagrad that employ different algorithms in order to minimize the loss of the model. Each implementation requires a different algorithm, thus multiple optimizers should be tested to determine the best one for the specified task.

While the previous hyperparameters affect the AI model as a whole, there are also hyperparameters that only affect one specific layer of the model. One of these hyperparameters is the activation function, which determines how many and which individual neurons should be activated within a particular layer. Neurons are a fundamental unit of a neural network, which store information as it passes through the network. Another hyperparameter for each layer is the unit size, which refers to the number of neurons within a layer. Both the activation function and the unit size can greatly affect accuracy and loss values.

Each of these hyperparameters are typically experimented with individually in order to determine how they affect the model. However, there are certain hyperparameters, such as activation functions, where the combinations between layers matter. An improvement to hyperparameters drives an increase in accuracy and a decrease in loss. An optimal set of hyperparameters is determined once the highest overall accuracy and lowest overall loss is achieved.

The model architecture also affects the accuracy and runtime of the model. As the number of layers increases, the accuracy of the model also increases, however, runtime greatly increases as well. Overfitting can also occur if the model has too many layers. An example of a model is the VGG-16 architecture. VGG-16 has 16 overall layers with the last three layers being dense. For more information on dense layers, refer to Appendix A. For more information on the specifications of the VGG-16 model, refer to Appendix G.

2.7 Transfer Learning with ImageNet

Transfer learning is a concept from machine learning in which information learned by one implementation of a neural network model is transferred and applied to another model trying to accomplish a similar task. It is often used for models when there is not enough training data to achieve a reliable accuracy (Khoshgoftaar, Wang, & Weiss, 2016). Transfer learning improves the accuracy of a model by utilizing pretrained weights, as well as prebuilt architecture, from another model which was trained on a dataset of a suitable size (Keras, 2020). This also helps to reduce overfitting.

There are open-source databases that can be used for transfer learning. Google's ImageNet is one of these databases, and it contains over 14 million images that are freely accessible to use for image recognition in AI models. The use of ImageNet for transfer learning is widespread in the machine learning discipline. Models pretrained on ImageNet have been used in many applications, including AIs that detect medical issues in medical images (Richmond & Xie, 2018), animal identification (Cowley, et al., 2021), and species classification (Andrews, et al., 2017).

2.8 Implications of Machine Learning

Technology can provide key tools to collect more data and to improve the monitoring of wildlife. As innovation progresses, technology becomes less resource intensive, further enabling researchers to use new technological tools. Therefore, there is a strong drive for ecologists to collaborate with technologists (such as engineers, computer scientists, and data analysts) to expand the scope of current technologies and tailor tools optimized to fulfill specific research goals (Berger-Tal & Lahoz-Monfort, 2018).

One concern is the trustworthiness and explainability of machine learning, particularly by users from outside the field. The abstract nature of 'black box' systems, such as those found in deep learning, can make it difficult for researchers to fully understand and trust the software that will be implemented (Ahamed, et al., 2022). As neural network-based technologies are still quite new, they face the problem of technology abandonment, as they will not be used if they are not clearly understood.

With the average user not understanding the inner workings of machine learning, overtrust can become an issue, as well (Kaluarachchi, Nanayakkara, & Reis, 2021). Overtrust refers to the phenomenon where the user trusts the system beyond its capabilities (Butz, Diefenbach, & Ullrich, 2021). When utilizing AI models, it is of paramount importance that the model is manually checked for inaccuracies at regular time intervals. While the dataset the model trains on is as large and diverse as possible, biases may still develop and inaccuracies tend to compound. As shown in Figure 4, it can be hard to differentiate between animals even by human reviewers, especially during the night. If the model is overtrusted, incorrect data may be used to make hypotheses, and may lead to false information being published or used for grant applications.

Alternatively, there is a sense in which the intelligent systems that we are using approach determinism by means of probability and statistics. The most relevant example of this is confidence values, which indicate how confident the system is in its classification. Low confidence values indicate that the user should manually review photos.

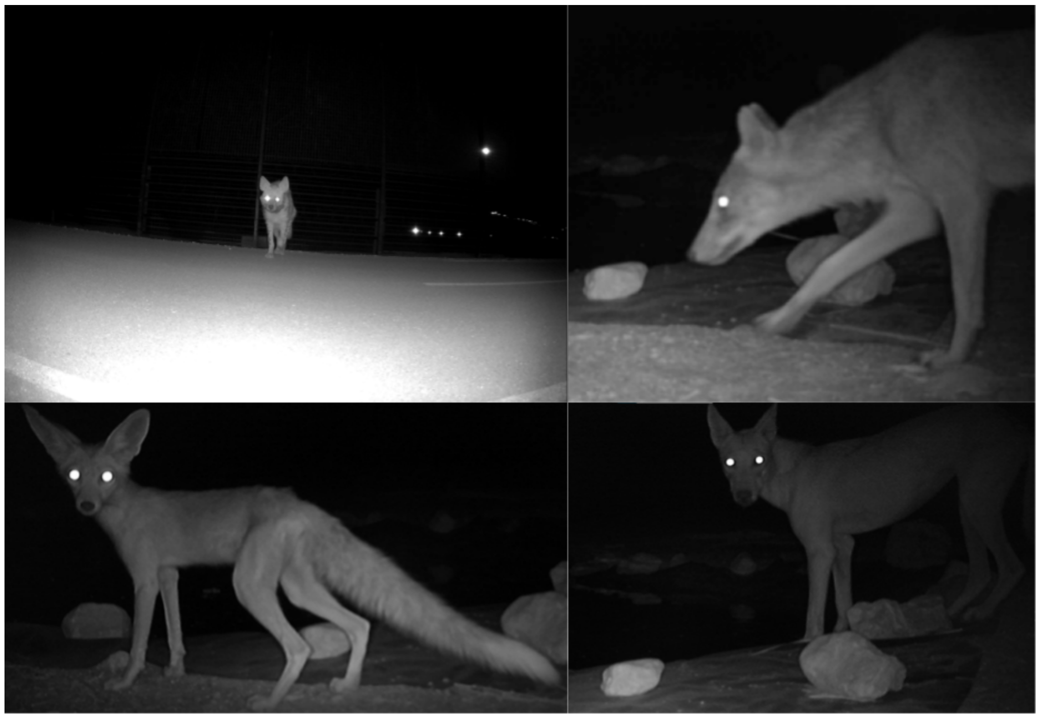


Figure 4: Photo of a hyena (top left), jackal (top right), fox (bottom left), and wolf (bottom right) at night, taken by one of our sponsor's cameras

2.9 Use Cases of the Arava-Dead Sea Science Center

At the ADSSC, our software may be utilized by researchers and interns. While they are tech-savvy, the importance of integrating our system into their current workflow as well as making a clear manual to explain how to use our software in detail was of the utmost importance. Troubleshooting guides and installation instructions were highly important to ensure that technological abandonment would not occur, and that the software could be implemented with ease.

CHAPTER 3: AUTOMATING IMAGE CLASSIFICATION OVERVIEW

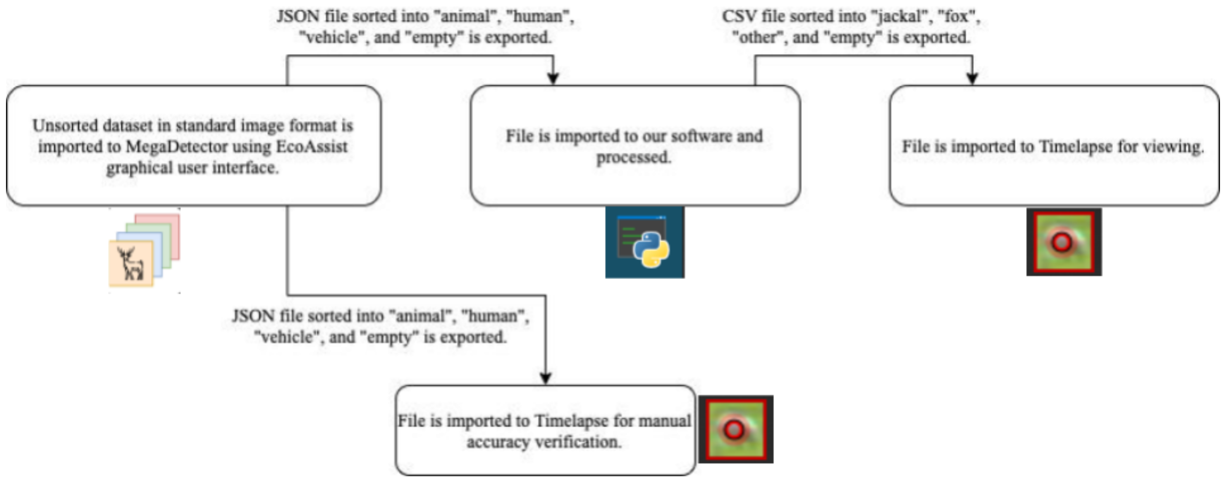

JSON file sorted into "animal", "human", "vehicle", and "empty" is exported.
CSV file sorted into "jackal", "fox", "other", and "empty" is exported.
Unsorted dataset in standard image format is imported to MegaDetector using EcoAssist graphical user interface.
File is imported to our software and processed.
File is imported to Timelapse for viewing.
JSON file sorted into "animal", "human", "vehicle", and "empty" is exported.
File is imported to Timelapse for manual accuracy verification.

Figure 5: Overview of our automated sorting process

Our first objective was to identify clear sorting categories for photographs. We conducted various interviews with our sponsor to gauge what our application should focus on. Additionally, trail cameras were placed at strategic intervals across from designated animal border crossings and left out for two weeks before photographic data was downloaded to a laptop and analyzed.

To address our second objective of automating the detection and classification of foxes and jackals using machine learning, and our third objective of optimizing our software to have high accuracy in classifying jackals and foxes, a deep learning model was constructed. The photographic data collected was used for training, and was preprocessed using various methods to make it more suited for the model to train on. Multiple optimization experiments were needed to ensure a suitable accuracy in making predictions.

To address our fourth objective of creating a practical process, as well as user manual, to integrate our software into the ADSSC's current workflow, demonstrations and interviews were conducted with our sponsor throughout the process to help ensure our application integrated smoothly with her current workflow, which utilizes Timelapse (see Figure 5). Additionally, various applications required to run our software were consolidated to reduce the amount of steps required to successfully use our software, thus making our application more practical. Interviews, demos, and consultations were conducted regularly to help ensure that our final product will be usable in the future.

CHAPTER 4: DETERMINING IMAGE SORTING CATEGORIES

Sorting categories were determined through several interviews on the use cases of the software with our sponsor, as well as our analysis of the 14,570 images we collected. These photos, obtained from the 20 cameras, were sorted and analyzed to gain an understanding of the dataset we would be working with. Our sponsor's main subjects of focus were jackals and foxes, as these were the most common animals found in the images and they cause the most damage out of the large carnivores in the Arava Valley. She also wanted to be able to easily analyze other images that contain animals in them while being able to skip over all the images that contained nothing of interest. To address this, four main categories were determined: “empty”, “fox”, “jackal”, and “other”. Images that contained nothing of interest would be sorted into the “empty” category, while foxes, jackals, and other animals would be sorted into their respective categories.

We decided to further break down the “fox” and “jackal” categories to reduce the AI's confusion when it sees animals from different angles. “Fox” and “jackal” both had subcategories of “back”, “front”, and “side”. Each of these subcategories, as well as the “other” and “empty” categories, were stored as subdirectories under one folder labeled “Training Data.” The sorting process was done by navigating to the directory that stored the data of each camera and then manually sifting through the images using Microsoft Photos, moving them into their respective subdirectories.

The “Training Data” folder was then used for training the deep learning model. When the model made a prediction, it classified an image as one of the eight categories, and then condensed the “fox” and “jackal” subcategories into either “fox” or “jackal”.

CHAPTER 5: DATA COLLECTION AND CURATION

5.1 Data Collection

Twenty trail cameras were placed at strategic points across from designated animal border crossings in order to track behavioral patterns (Figure 6). These cameras were left out for two weeks at a time, with the locations, spanning the Eilat-Eilot region, being rotated twice per season (Figure 7). There were three types of cameras used in this data collection process: Bushnell Aggressors, Browning Dark Ops (Figure 8), and ATC 70s. The Bushnells and Brownings use SD cards (Figure 8) to store all the photographic data from the camera which was then downloaded to a laptop as each camera was collected. The ATCs use a USB-A to micro USB adapter to connect directly to the laptop. The data was downloaded from the cameras’ storage.

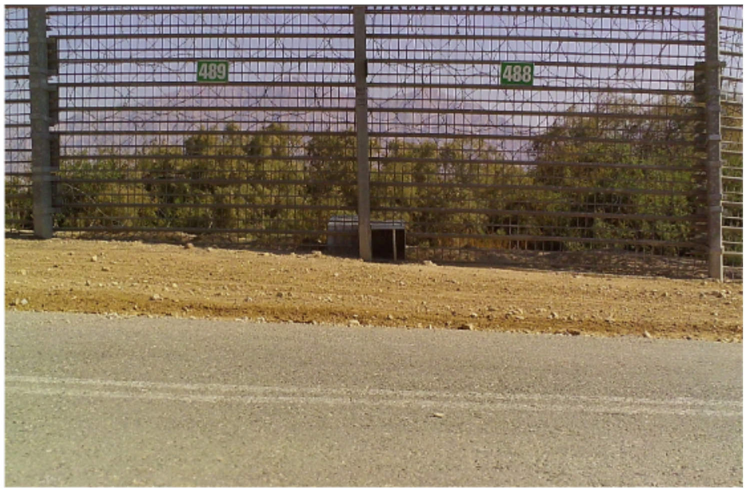


Figure 6: Example of an Israeli-Jordanian border crossing

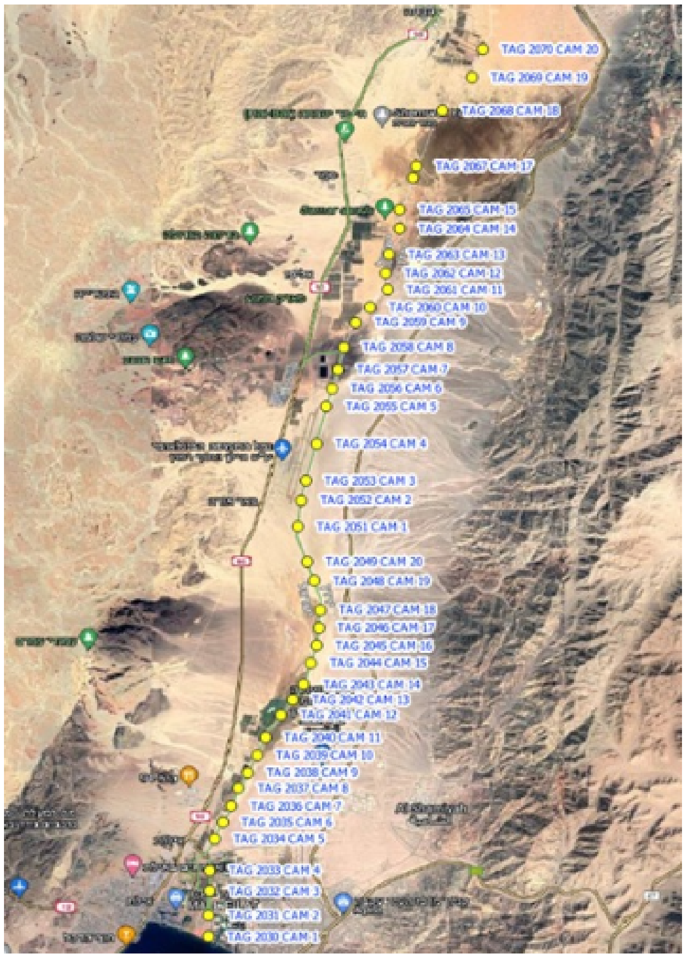


Figure 7: Map of trail cameras from winter 2023

Figure 8: Browning Dark Ops trail camera with SD card

5.2 General Dataset Description

A total of 5,751 images were used for training and validating the model. Of this, 809 images were of foxes (170 fox-back, 139 fox-front, 502 fox-side), 1,380 were of jackals (311 jackal-back, 236 jackal-front, 833 jackal-side), 2,946 were empty (no animal present), and 614 were of other (cars, other animals, people, etc.).

5.3 Dataset Preprocessing

To properly train the neural network, photographic data from the trail cameras had to be pre-processed. We relied on manual editing in addition to using the Keras.utils and Tensorflow.image modules to process the images. Many of the images in the dataset were unfocused or too dark, which could confuse the AI. To address this, we manually edited these images using Microsoft Photos. Unfocused images were cropped to have the animal of interest in the center, and images that were too dark were brightened for clarity. This was done solely for training and validation. The images the model will make predictions on will be unmodified.

Keras.utils was used to construct a set of training and validation data from our image directory. These images were a mix of both grayscale and color images that were all converted to grayscale for the purpose of standardization. The images were compressed to adhere to the size constraints of the model, which takes inputs of 224 by 224 pixels. The dataset was shuffled, batched into sets of eight, and split for training and validation. The split is 70-30 between training and validation, which is needed in order to avoid biased results. In this case, the validation set refers to a small subset of images that are classified at the end of each epoch to determine the model's accuracy on new data. The dataset must be shuffled to improve the accuracy of the network: without randomization, the network may overfit, thus hindering its learning capabilities as it analyzes different categories. The data was batched to reduce stress on computer memory. Since we worked on a dataset of thousands of images, having a small subset loaded onto memory is less taxing on the computer than the entire dataset. The shuffle/validation split seed was chosen arbitrarily. The images were then converted into the RGB color model through the use of the Tensorflow.image module. This was done to implement the VGG-16 model trained on Google's ImageNet, which inputs RGB images, for transfer learning.

5.4 Dataset Labeling

The dataset was originally mixed together and unsorted. First, we sifted through collected data both from the ADSSC and from images we collected, sorting each image into “empty”, “jackal”, “fox”, and “other”. The folders “jackal” and “fox” were further sorted into “fox-back”, “fox-front”, “fox-side”, “jackal-back”, “jackal-front”, and “jackal-side”. During this process, some images that were unclear or unable to be identified without the context of the photos taken in the same episode were removed from the dataset.

5.5 Dataset Overview

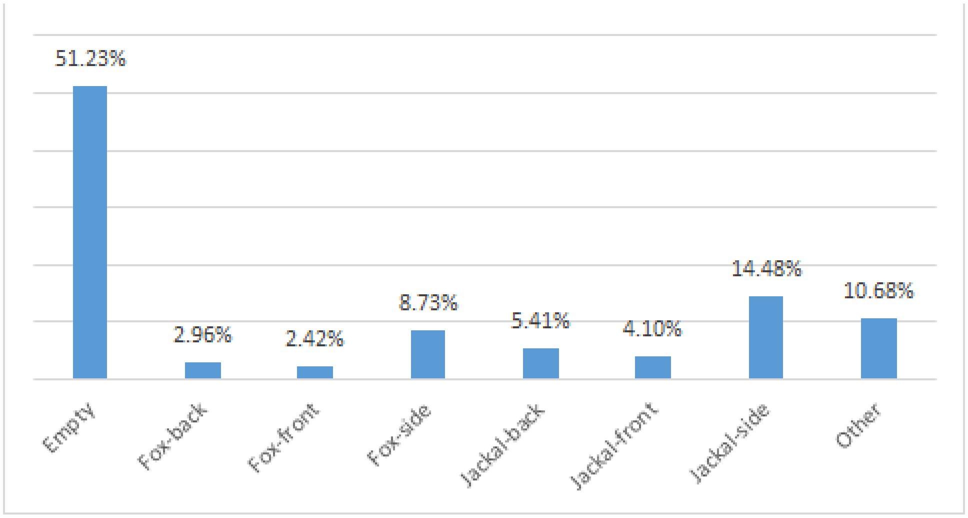


Figure 9: Percentage breakdown of the training and validation data

70% of the dataset was used for testing (4,026 images), while 30% was reserved for validation (1,725 images). Figure 9 shows the breakdown of the dataset. Note that the dataset is unbalanced, but representative of the data from the trail cameras.

CHAPTER 6: AI MODEL FOR DETECTION OF JACKALS AND FOXES

6.1 MegaDetector Results

Using an NVIDIA GeForce RTX 3050 graphics processing unit, we were able to classify 34,493 images in 2 hours and 43 minutes with above 90% accuracy using MegaDetector. Not only is this a 405% increase in speed compared to the ADSSC’s current process, but other tasks can be completed while the software runs in the background, as well. By automating this process, this will save researchers at the ADSSC several hours per week.

6.2 Implementing and Training the Model

Given the ubiquity and success of the convolutional neural network (CNN) within the machine learning industry for image classification, we decided to implement a CNN model. After comparing different CNN models such as ResNet50 and VGG-19, we settled on the VGG-16 model. Despite requiring more storage space than other models, the ADSSC was not concerned with storage space and the VGG-16 model performed the best. The VGG-16 model was used throughout the entire project.

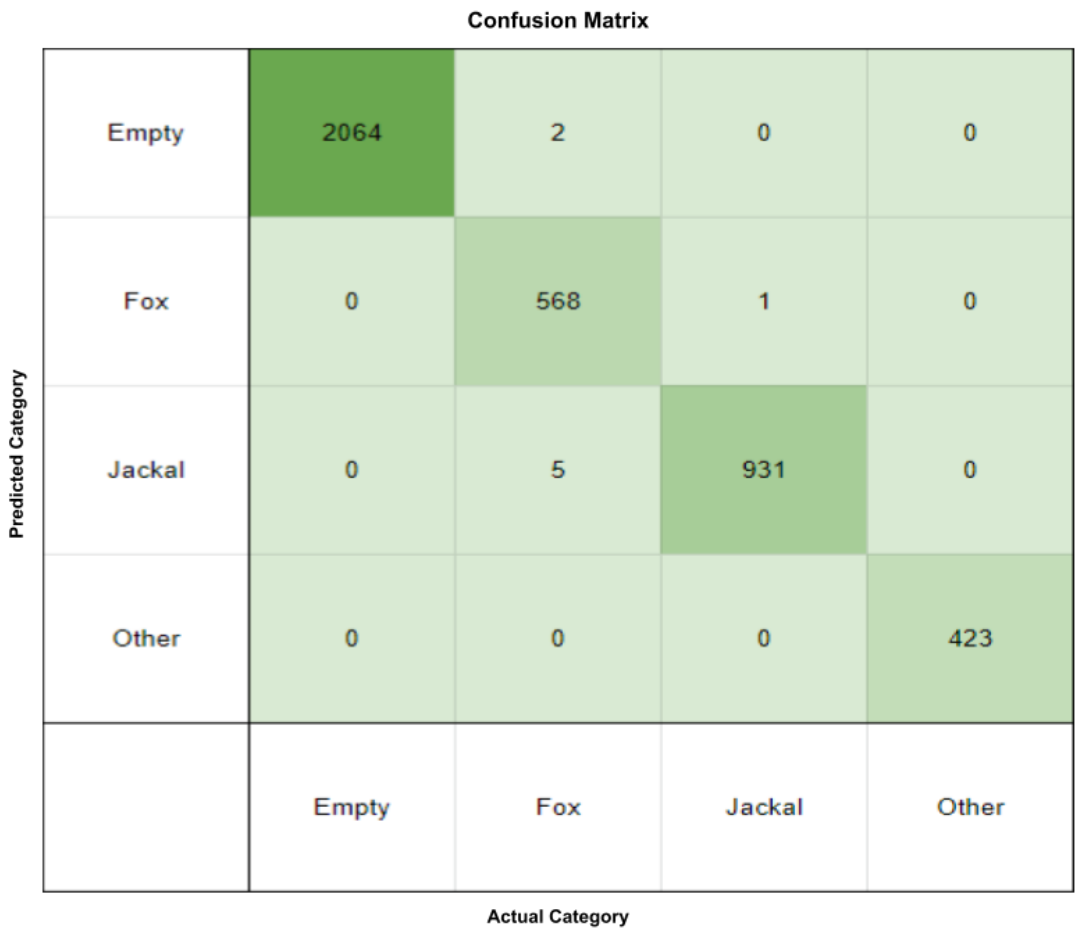
6.3 Classification Results

Below, we report our main result of the classification model, demonstrating our software's ability to classify images as being empty, or containing a jackal, fox, or other. Four metrics were used to evaluate the performance of the model: loss, validation loss, accuracy, and validation accuracy. Loss measures the magnitude of error the algorithm made when analyzing a photo in the training dataset. Validation loss measures the magnitude of error the algorithm made when analyzing a photo in the validation dataset. Accuracy is derived from making predictions on the training dataset, while validation accuracy is derived from making predictions on the validation dataset.

After determining the optimal parameters from our experiments, reported in sections 6.5-6.10, a final run was conducted using these parameters with steps per epoch being set to None. This resulted in the steps per epoch defaulting to the number of batches in the dataset (in this case, 504). This method results in significantly higher accuracy with little to no overfitting, especially compared to the previous experiments, in which 30 steps per epoch were used.

Furthermore, validation steps were also revisited, being set to None. We found that the difference in performance between validation steps being set to None and to 10 after all other hyperparameters had been chosen was negligible, with the value of 10 resulting in a significantly faster run time. Thus, we decided to use 10 validation steps over None, despite our initial experimentation.

Additionally, transfer learning was implemented to improve the accuracy of our model in all experiments. We decided to use the weights of a VGG-16 model that was trained on Google's ImageNet database. The improved accuracy of the model trained by ImageNet was verified by comparison to the same VGG-16 model without transfer learning.



Confusion Matrix

Predicted Category	Empty	Fox	Jackal	Other
Empty	2064	2	0	0
Fox	0	568	1	0
Jackal	0	5	931	0
Other	0	0	0	423

Actual Category

Figure 10: Confusion Matrix showing actual vs predicted categories

Figure 11 shows a confusion matrix which demonstrates where the AI misclassified images on a dataset of 4,025 images. The x-axis represents the actual labeled category of the image and the y-axis represents the category predicted by our AI. If the AI were to have 100% accuracy, the diagonal of the matrix will add up to the total number of images in the dataset. In this matrix, the diagonal adds up to 4,017. This means that only eight images out of 4,025 were misclassified. The only mistakes in which the animal was not correctly identified was labeling two images of foxes as empty, five images of foxes as jackals, and one image of a jackal as a fox.

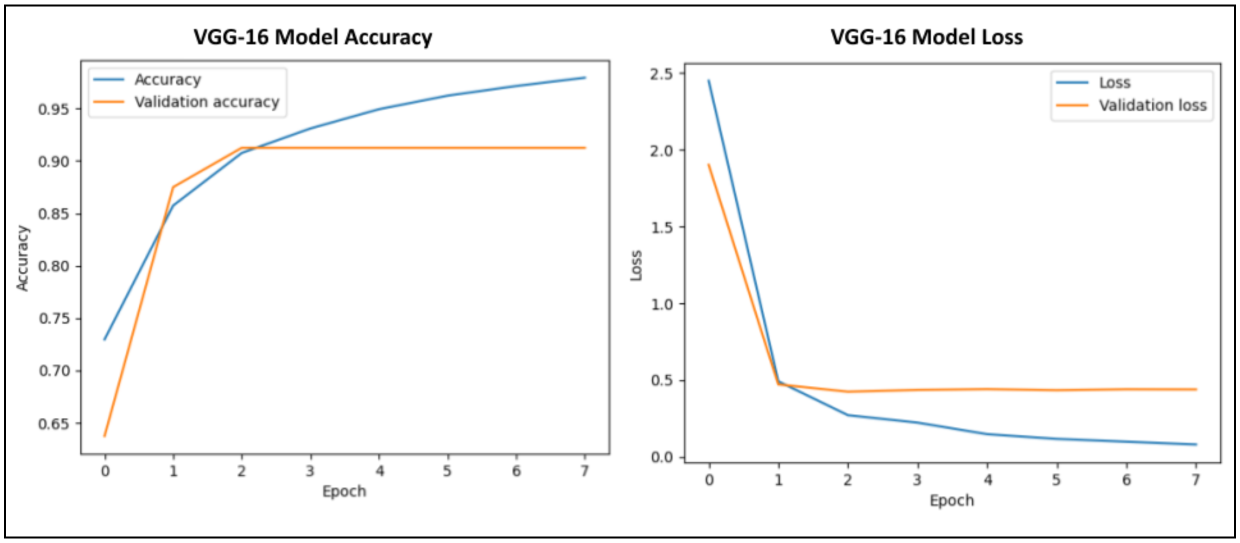


Figure 11: Final run using optimal parameters from previous experiments

The other parameters and hyperparameters used for this run were: 15 epochs, SELU/ReLU activation functions, AdaGrad optimizer, 1,024 unit size for the 14th and 15th layers, and 10 validation steps. As shown in Figure 10, there was no overfitting in this run, which resulted in very high accuracy and validation accuracy and very low loss and validation loss values. After early stopping at epoch nine, the final values had an accuracy of 0.9794, a loss of 0.0804, a validation loss of 0.4392, and a validation accuracy of 0.9125. This means that the overall accuracy of categorizing photos into “fox”, “jackal”, “empty”, and “other” was 91.25%. This means that the ADSSC researchers will only have to manually sort through approximately 1,000 images every two weeks rather than the current 14,000-15,000. This will also reduce the workload of the researchers from 11 or more hours per week to approximately one hour per week.

6.4 Improving Accuracy: Optimization Experiment Overview and Methodology

Neural networks have different hyperparameters that affect the speed, accuracy, and overall performance of a model, thus, experimentation was needed to determine the best performing set of hyperparameters that would result in the highest accuracy, as reported in section 6.3. Thus, experiments were conducted with each hyperparameter in an effort to maximize accuracy as described in the sections below.

The experiments conducted intended to determine the best performing hyperparameters for the model. In the first three experiments, the batch size was set to eight, the activation functions on the first and second dense layers were set to ReLU, and the optimizer used was Adam. Early stopping was also incorporated with a patience value of three. The last three experiments conducted tested three other important hyperparameters: optimizer algorithm, activation function, and unit size. After the experiments were conducted, it was concluded that training with 15 epochs, steps per epoch set to None, and validation steps set to 10 had the best performance.

In reference to the network's structure, it was determined that the first and second dense layers would consist of 1,024 neurons each. The Scaled Exponential Linear Unit (SELU) activation function was utilized for the first dense layer since it learns faster and more efficiently than other activation functions. The Rectified Linear Unit (ReLU) activation function was utilized for the second dense layer because it avoids the learning slowdown that results from using other activation functions. The pairing of SELU and ReLU was also chosen because they performed best out of the combinations tested. The SoftMax activation function was used for the final dense layer to specify the probability of output categories from multiple neurons. Eight neurons were implemented in the final dense layer to represent the eight output categories.

The AdaGrad optimizer was used for the model. Multiple industry-standard optimizers were tested and AdaGrad performed the best. The model was trained for 15 epochs with steps per epoch set to None and an additional 10 validation steps. This was found to be the most optimal when compared to other variations.

6.5 Epoch Experimentation

The first experiment was carried out to determine what number of epochs resulted in the best performing validation accuracy and validation loss. All other hyperparameters were kept constant. Table 1 shows the results of the trials. Figure 23 shows the best performing model of the experiment. Three variations of epoch count were performed. The values experimented with were 5, 10, and 15. Each value was tested three times.

Table 1: Variation of epoch count

Variation	Trial	Epoch Count	Steps per Epoch	Validation Steps	Loss	Accuracy	Validation Loss	Validation Accuracy	Avg validation loss	Avg validation accuracy
1	1	5	10	10	10.399	0.675	15.149	0.625		
	2	5	10	10	14.466	0.663	6.249	0.638		
	3	5	10	10	14.479	0.638	11.966	0.688	11.121	0.650
2	1	10	10	10	4.866	0.713	6.852	0.663		
	2	10	10	10	4.521	0.800	8.780	0.600		
	3	10	10	10	5.277	0.713	2.924	0.725	6.185	0.663
3	1	15	10	10	3.819	0.763	5.340	0.625		
	2	15	10	10	4.869	0.600	1.785	0.675		
	3	15	10	10	2.628	0.788	5.653	0.638	4.259	0.646

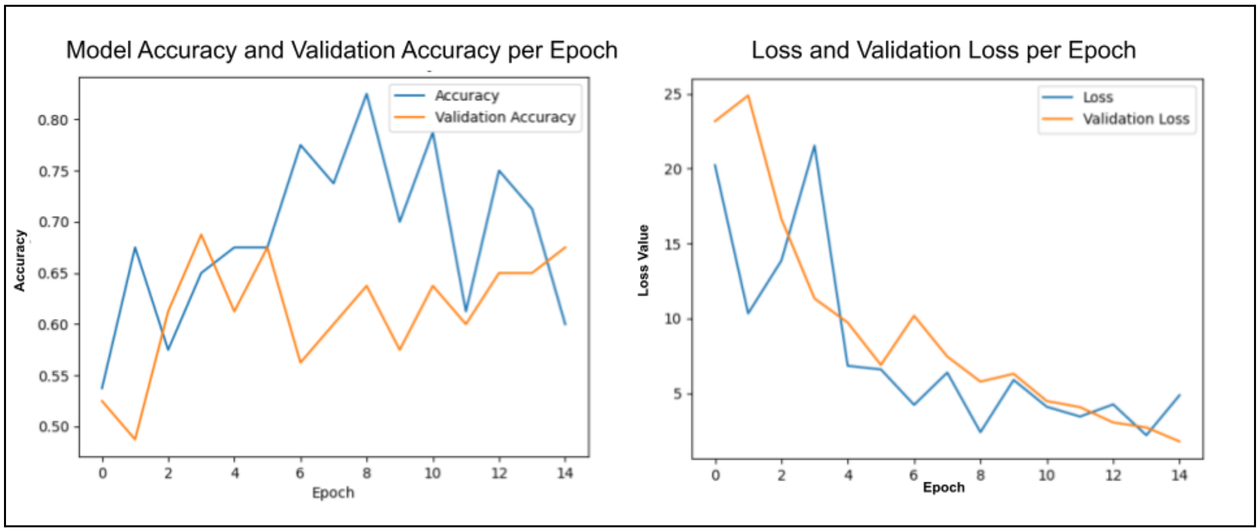


Figure 12: Performance metrics of variation 3, trial 2

When 5 epochs were used, the average validation loss was 11.121, the average validation accuracy was 0.650, the standard deviation of validation loss was 4.510, and the standard deviation of validation accuracy was 0.033.

When 10 epochs were used, the average validation loss was 6.185, the average validation accuracy was 0.663, the standard deviation of validation loss was 2.984, and the standard deviation of validation accuracy was 0.063.

When 15 epochs were used, the average validation loss was 4.259, the average validation accuracy was 0.646, the standard deviation of validation loss was 2.148, and the standard deviation of validation accuracy was 0.026.

In all nine trials, overfitting occurred, resulting in a significant difference between validation accuracy and training accuracy. In Variation 3 Trial 3, early stopping occurred at epoch 9. Early stopping occurs when the validation loss does not improve after 3 epochs, and terminates the training program earlier to reduce overfitting. From these trials, it was determined that an epoch count of 15 would perform best out of the other tried values. It had the best average validation loss, a slightly less average validation accuracy than 10 epochs, and a more consistent performance than other trials.

The experiments concluded that the model will use 15 epochs as a hyperparameter. While an epoch count of 10 did perform slightly better than 15 in terms of validation accuracy, it was more inconsistent in performance than with 15 and the validation loss was significantly higher. Furthermore, the incorporation of early stopping will help prevent overtraining of the model, should 15 epochs ever prove to be too much. Moving forward, all experiments had the epoch hyperparameter set to 15.

Figure 12 shows the best performance with the epoch count set to 15. The validation accuracy of the model gradually increased with each epoch, however the training accuracy dropped from over 80% in epoch 8 to 62% in epoch 15. This was most likely caused by the learning rate of the model being too high, resulting in the weights in the last three dense layers changing too frequently.

6.6 Variation of Steps per Epoch Experimentation

The second experiment intended to determine how many steps per epoch performed best. All other hyperparameters were kept constant. Table 2 shows the results of each trial. Figure 13 shows the results of the best performing model. Three variations in steps per epoch were investigated: None[4], 20 steps, and 30 steps. 10 steps was skipped as its values can be found in Table 1. Each variation was tested three times.

Table 2: Variation of Steps per epoch

Variation	*Trial*	*Epoch Count*	*Steps per Epoch*	*Validation Steps*	*Loss*	*Accuracy*	*Validation Loss*	*Validation Accuracy*	*Avg validation loss*	*Avg validation accuracy*
1	1	15	None	10	0.744	0.747	0.808	0.688		
	2	15	None	10	0.763	0.733	1.319	0.650		
	3	15	None	10	0.708	0.748	0.807	0.688	0.978	0.675
	1	15	20	10	0.910	0.706	1.108	0.650		
	2	15	20	10	2.479	0.688	5.310	0.650		
2	3	15	20	10	4.305	0.681	5.554	0.688	3.991	0.663
	1	15	30	10	2.124	0.700	3.505	0.663		
3	2	15	30	10	0.903	0.738	1.434	0.650		
	3	15	30	10	1.081	0.725	2.383	0.650	2.441	0.654

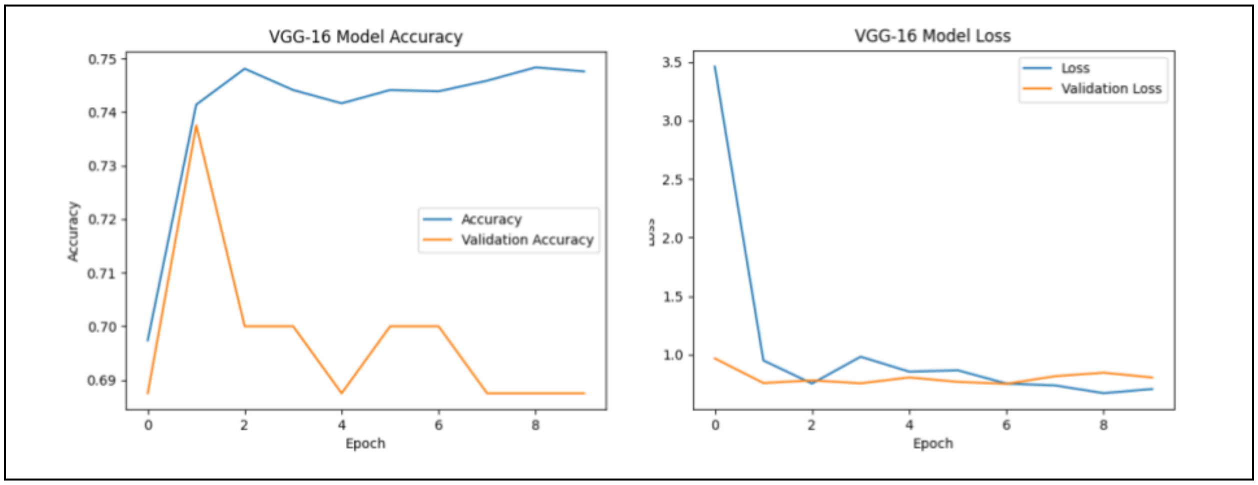


Figure 13: Performance Metrics of variation 1, trial 3

When the steps value was set to None, the average validation loss was 0.978, the average

[4] When specifying the number of steps per epoch in Keras' implementation of a neural network, entering "None" defaults the number of steps to be the number of batches for training data. "None" is a default python value.

validation accuracy was 0.675, the standard deviation of validation loss was 0.295, and the standard deviation of validation accuracy was 0.022.

When 20 steps were used, the average validation loss was 3.991, the average validation accuracy was 0.663, the standard deviation of validation loss was 2.499, and the standard deviation of validation accuracy was 0.022.

When 30 steps were used, the average validation loss was 2.441, the average validation accuracy was 0.654, the standard deviation of validation loss was 1.037, and the standard deviation of validation accuracy was 0.007.

In all nine trials, overfitting occurred, which can be seen in Figure 13, where validation accuracy continues to decrease as training accuracy increases. This resulted in a significant difference between validation accuracy and training accuracy. Early stopping occurred in all trials except Variation 2, Trial 1. From these trials, we concluded that the best value for the steps per epoch hyperparameter was None. It had the lowest average validation loss, and the highest validation accuracy out of all the trials.

As seen in Figure 13, the validation accuracy differed greatly from the accuracy metric with an unspecified amount of steps, decreasing as training accuracy increased. This was most likely caused by the model overfitting. However, this model still performed the best out of the three variations, and as such this value would be revisited once all the hyperparameters had been experimented with.

All three trials in Variation 1 took three to five hours to train the model, which led to all future experiments not related to the epoch hyperparameters being conducted with 30 steps per epoch, the second best performing variation, which took 45 minutes to an hour to train.

6.7 Variation of Validation Steps Experimentation

The third experiment intended to determine how many validation steps would result in the highest validation accuracy and lowest validation loss. All other hyperparameters were kept constant. Table 3 shows the results of each trial. Figure 14 shows the results of the best performing model. Three variations in validation steps were investigated: None, 20 steps, and 30 steps. 10 steps was skipped as its values can be found in Table 2. Each variation was tested three times.

Table 3: Variation of validation steps

Variation	Trial	Epoch Count	Steps per Epoch	Validation Steps	Loss	Accuracy	Validation Loss	Validation Accuracy	Avg validation loss	Avg validation accuracy
1	1	15	None	None	0.825	0.745	0.799	0.750		
	2	15	None	None	0.922	0.744	0.951	0.744		
	3	15	None	None	0.766	0.748	0.816	0.748	0.855	0.747
2	1	15	None	20	0.671	0.754	0.783	0.725		
	2	15	None	20	0.685	0.759	0.864	0.725		
	3	15	None	20	0.713	0.747	0.795	0.713	0.814	0.721
3	1	15	None	30	0.783	0.751	0.712	0.733		
	2	15	None	30	0.806	0.746	0.929	0.663		
	3	15	None	30	0.795	0.740	1.008	0.725	0.883	0.707

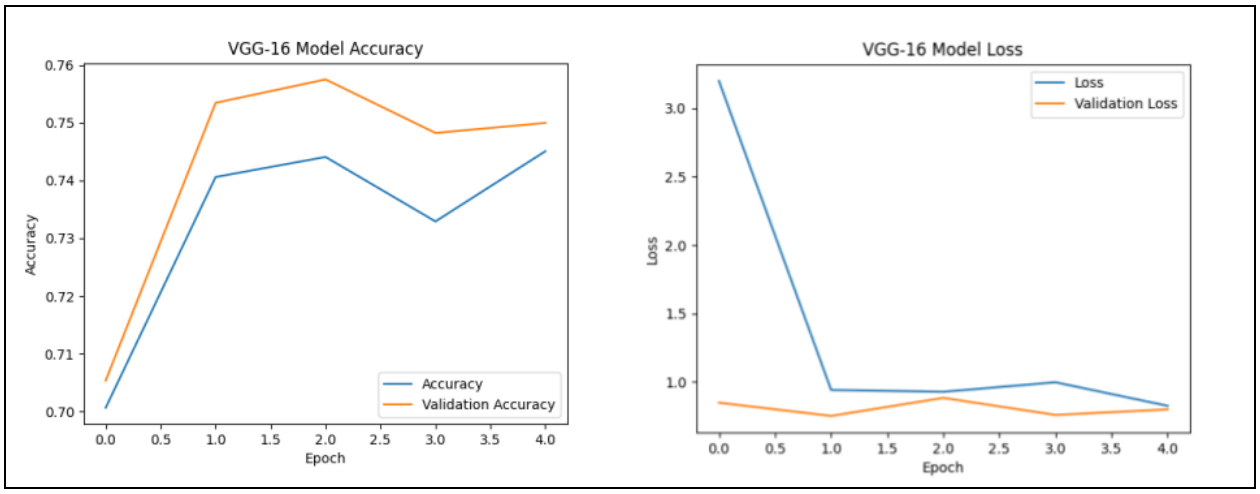


Figure 14: Performance metrics of variation 1, trial 1

When the steps value was set to None, the average validation loss was 0.855, the average validation accuracy was 0.747, the standard deviation of validation loss was 0.083, and the standard deviation of validation accuracy was 0.003.

When 20 steps were used, the average validation loss was 0.814, the average validation accuracy was 0.721, the standard deviation of validation loss was 0.043, and the standard deviation of validation accuracy was 0.007.

When 30 steps were used, the average validation loss was 0.883, the average validation accuracy was 0.707, the standard deviation of validation loss was 0.153, and the standard deviation of validation accuracy was 0.039.

In all nine trials, early stopping occurred, from as early as epoch 5 to as late as epoch 11. The validation accuracy curve was closest to logarithmic when validation steps were set to None, as seen in Figure 14. While validation accuracy was best when validation steps were set to None, all future experiments were conducted with 10 validation steps as it significantly decreased the validation time. Further, when this value was revisited in future experimentation, 10 validation steps also performed better than None in certain models.

6.8 Variation of Optimizer

A major part of our AI model design process was choosing the best performing optimizer. The Tensorflow optimizer module contains eight different optimizers: Adam, AdaMax, AdaGrad, AdaDelta, Nadam, FTRL, RMSprop, and SGD. Each of these optimizers was tested at least once, but AdaMax, AdaGrad, and FTRL showed significantly better performance than the other five and were selected for further testing. The purpose of the optimizer experiment was to improve the accuracy and validation accuracy values of the model.

Table 4: Variation of optimizer

Optimizer	*Trial*	*Loss*	*Accuracy*	*Validation loss*	*Validation accuracy*	*Avg validation loss*	*Avg validation accuracy*
AdaMax	1	1.344	0.754	1.653	0.688		
	2	1.029	0.821	1.201	0.713		
	3	1.202	0.767	1.255	0.725	1.369	0.708
AdaGrad	1	0.639	0.813	0.801	0.788		
	2	0.785	0.779	1.185	0.675		
	3	0.606	0.808	0.844	0.763	0.943	0.742
FTRL	1	0.608	0.788	0.724	0.738		
	2	0.614	0.779	0.720	0.738		
	3	0.605	0.783	0.689	0.738	0.711	0.738

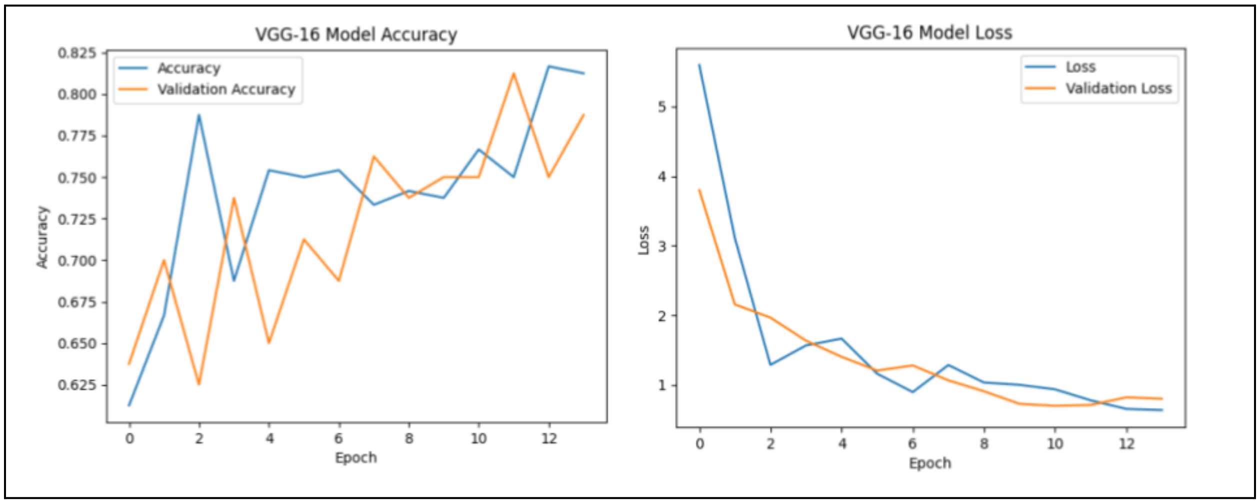


Figure 15: Performance Metrics of AdaGrad trial 1

Each test was performed using 15 epochs, 30 steps per epoch, and 10 validation steps. After testing these three optimizers at least three times each, AdaGrad was chosen as our optimizer going forward. As shown in Table 4, AdaGrad had good values for all four categories. While FTRL had better loss values, AdaGrad had significantly better accuracy and validation accuracy. Also shown in Table 4, FTRL had a standard deviation of zero for the validation accuracy, meaning that it was not capable of increasing above a value of 0.7375 with the given parameters. Therefore, while AdaGrad's validation accuracy dipped below this value on occasion, the average validation accuracy was higher than that of FTRL, with values as high as 0.825 in later tests. In addition, Table 1 shows tests using the Adam optimizer, and compared to the results in Table 1, AdaGrad showed nearly a 0.1 increase in average validation accuracy for 15 epochs. Figure 15 shows the best performance of the three AdaGrad trials.

6.9 Variation of Activation Function

Activation function combinations can determine how quickly the model learns as well as how accurately the model learns. As aforementioned, the Scaled Exponential Linear Unit (SELU) and Rectified Linear Unit (ReLU) functions were our main focus for two dense layers at the end of our model. Previous experiments were conducted using purely ReLU/ReLU, so we have tested SELU/ReLU, ReLU/SELU, and SELU/SELU for comparison.

Table 5: Variation of activation function

Optimizer	*Trial*	*Loss*	*Accuracy*	*Validation Loss*	*Validation Accuracy*	*Avg validation loss*	*Avg validation accuracy*
SELU/ReLU	1	0.968	0.725	0.707	0.775		
	2	0.851	0.758	0.636	0.788		
	3	0.829	0.754	0.738	0.788	0.694	0.783
ReLU/SELU	1	1.075	0.746	1.191	0.750		
	2	0.859	0.738	0.709	0.763		
	3	0.819	0.750	0.727	0.788	0.876	0.767
SELU/SELU	1	1.069	0.754	0.763	0.788		
	2	1.129	0.746	0.779	0.775		
	3	0.875	0.763	0.715	0.750	0.753	0.771

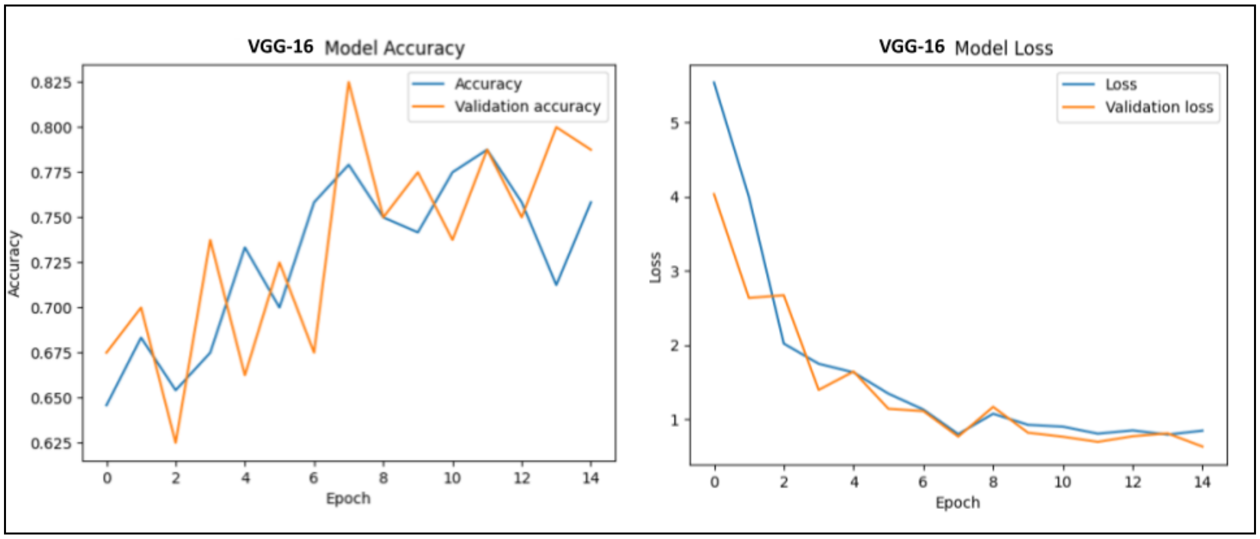


Figure 16: Performance Metrics of SELU/ReLU trial 2

15 epochs with 30 steps per epoch, 10 validation steps, AdaGrad optimizer, and 256 unit size were used for this experiment. As shown in Table 5, the average validation accuracies for each were very similar, however, SELU/ReLU resulted in a significantly lower average validation loss. The standard deviations for validation loss and validation accuracy were both very low with values of 0.052 and 0.007, respectively. This is important because while Figure 16 shows that there appears to be heavy overfitting, the low standard deviation in validation accuracy means that it is insignificant and our results are accurate. SELU/SELU had similarly low standard deviations for validation loss and validation accuracy and it did not early stop, however, SELU/ReLU was chosen as our combination due to the better average values. This means that the 14th layer in our neural network uses the SELU activation function and the 15th layer uses the ReLU activation function.

6.10 Variation of Unit Size

Unit size is another factor to consider in our AI model as it can greatly reduce the loss values at the expense of runtime. Unit size refers to how many neurons are in a particular layer of the neural network. Previous experiments were run using a unit size of 256, therefore, sizes of 128, 512, and 1,024 were chosen for comparison.

Table 6: Variation of Unit size

Unit Size	*Trial*	*Loss*	*Accuracy*	*Validation loss*	*Validation accuracy*	*Avg validation loss*	*Avg validation accuracy*
512	1	0.692	0.783	0.669	0.788		
	2	0.923	0.746	0.664	0.763		
	3	0.920	0.783	0.827	0.700	0.720	0.750
128	1	0.789	0.750	0.703	0.775		
	2	0.769	0.742	0.491	0.813		
	3	0.819	0.758	0.668	0.763	0.620	0.783
1024	1	0.919	0.758	0.673	0.825		
	2	0.794	0.775	0.582	0.763		
	3	1.060	0.750	0.702	0.788	0.652	0.792

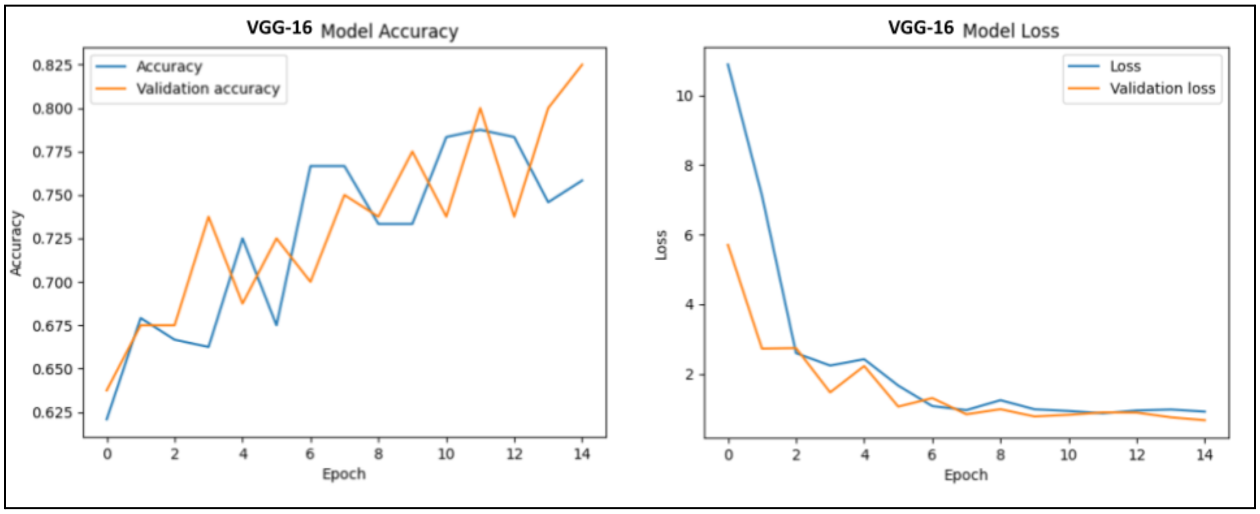


Figure 17: Performance metrics of 1024 trial 1

ReLU/ReLU activation functions and the AdaGrad optimizer were used in this experiment with 30 steps per epoch, 15 epochs, and 10 validation steps. As shown in Table 6, 128 validation steps resulted in the same runtime as 256, with the best average validation loss value and close to the best average validation accuracy value. However, the standard deviation of validation loss was 0.114, which was significantly higher than the standard deviation values for 512 and 1,024, which were 0.092 and 0.062, respectively. Consequently, 1,024 was chosen as the best unit size value as the validation loss values were more stable and it had the highest average validation accuracy (Figure 17). Additionally, the runtime increase was minimal, with only a four second increase per epoch compared to a unit size of 256. This means that overall, the total runtime increase was only one minute with 15 epochs.

CHAPTER 7: INTEGRATION WITH RESEARCH WORKFLOW

7.1 Integration with Timelapse

For the final software to be easily usable by researchers at the ADSSC, it needed to be integrated with Timelapse, a software the ADSSC uses to process images from trail cameras. Interviews and demonstrations were conducted to gain an understanding of how Timelapse is used and to explore potential methods of integration. Timelapse can import and export data through CSV files, so our final product reads and writes CSV files as well.

Timelapse is CSV compatible, so we programmed our software to be able to read and write CSV files. Our software can take in a directory containing a set of images that have or have not been labeled yet and then make a prediction of what is in each image. A decoder reads the prediction and confidence of said prediction, and then writes to a CSV file. It reads the file's metadata to extract its name and relative path in the directory. It adds this to a CSV file, and then adds the prediction of what is in the image, and saves the CSV in the same directory as the images. Once Timelapse is opened, a datatable can be filled out by importing the CSV, thus connecting the two programs.

7.2 Finalizing the Application and User Manual

Our application was developed to have a "no-code" interface.[5] Additionally, user manual drafts were created, reviewed and tested, and rewritten to address any problems users might encounter, from installation to troubleshooting and maintenance (Appendix C).

Multiple use case interviews were conducted with our sponsor to gauge what was required of the final application. The information gained from these interviews was used as a basis to frame the total process of the final application, as well as a framework for the user manual. It was determined that integrating our software to be used alongside Timelapse was ideal for our

[5] No-code interface refers to a user interface that requires no programming knowledge whatsoever to interact with.

sponsor. Multiple different applications and programming languages were consolidated within our software. This reduced the number of steps required to run our application, thus decreasing risk of random error and increasing ease of use.

A comprehensive user manual was created for ease of use (Appendix C). The manual details all the steps of utilizing our software, from installation to running a dataset through for classification. The manual also included troubleshooting options, as well as frequently asked questions. Initial tests of the manual were conducted by a team member and they were successful in running a dataset through the software. Then, the manual was brought to our sponsor and her research assistants, who were also successful in running a dataset.

Our user manual is sufficient for troubleshooting or for a clean install of the program in the event that the application crashes, gets uninstalled, or a change is made at the ADSSC that requires maintenance on the hardware or software of the lab computers. Our sponsor indicated that she has some programming experience and is comfortable and familiar with installing applications similar to ours. Additionally, she indicated that this program will be utilized by interns from Ben Gurion University, Northeastern University, and Worcester Polytechnic Institute, all of whom are tech savvy and can easily follow this process. However, our sponsor mentioned that she does not have experience using Python and is not familiar with Windows Command Prompt. Therefore, our user manual contains highly detailed guides, as well as external contacts, for these portions of the installation and maintenance process. In the event that the hardware of the ADSSC's computers is upgraded, our application will run significantly faster, especially if the graphics processing unit is upgraded.

7.3 Use by the Arava-Dead Sea Science Center

In an interview with our sponsor, she expressed that she was pleased with the accuracy and runtime of the software. She also believes that it will be incredibly useful for the ADSSC. During an initial technical demonstration of our software with researchers at the ADSSC, confusions were addressed and our user manual was updated accordingly. More detailed step-by-step guides of each software within the new workflow were written. Our sponsor was able to successfully install all necessary software on her own computer as well as complete a full trial run on a small set of images. She was able to run EcoAssist, our AI model, and import the generated CSV into Timelapse using only our user manual. In order to address overtrusting or undertrusting our AI, we had our sponsor run multiple different sets of images on multiple computers in order to confirm that the software works reliably in multiple settings. We also clearly defined which files are necessary to manually review in order to ensure that the data is not blindly trusted as well. Overall, our sponsor was pleased with the results of the project and has already begun implementing it into the ADSSC's workflow.

7.4 Hardware Requirements and Recommendations

The following is a list of the minimum and recommended computer requirements to run our software as efficiently as possible.

Table 7: List of recommended hardware components

Component	Minimum	Recommended	Best	Purpose
Central Processing Unit (CPU)	Intel i5 or AMD Ryzen 5	Intel i7* or AMD Ryzen 7	Intel i9 or AMD Ryzen 9	This will determine the processing speed of the AI classification portion of the software
Graphics Processing Unit (GPU)	NVIDIA GeForce GTX 750 Ti	NVIDIA GeForce GTX 1070	NVIDIA GeForce RTX 3090	This will determine the processing speed of MegaDetector
Random Access Memory (RAM)	8GB DDR4 or DDR5	16GB DDR4* or DDR5	32GB or more DDR4 or DDR5	This will determine the quantity of images that can be processed at once by the AI

*- This component is currently in use by the ADSSC

"Minimum" is a recommendation from the experimenters of a component that is capable of running the software incredibly slowly, but at minimal or low cost. "Recommended" is the experimenters' recommendation for a component that will have good runtime and a medium cost. “Best” is the experimenters’ recommendation for a component that will run the software incredibly quickly, but at a high cost.

CHAPTER 8: CONCLUSIONS AND RECOMMENDATIONS

Strategies for automating detection and previously manual classification of images of jackals and foxes were investigated to save time for researchers at the Arava-Dead Sea Science Center. First, sorting categories and subcategories were determined to train the AI. Microsoft’s MegaDetector was used for the detection of animals, humans, and vehicles in the images. These images were then classified by our software as “jackal”, “fox”, “other”, or “empty”. The resulting images and classification tags were exported as a CSV file. The CSV files were then able to be imported to Timelapse for viewing, thus integrating our software into the ADSSC’s established workflow. Our software achieved above 90% accuracy in both the detection and classification softwares. A comprehensive user manual was created after thorough interviews and user testing. Overall, this will greatly reduce the workload for researchers at the ADSSC from approximately 11-12 hours per week to 1-1.5 hours per week, allowing researchers to monitor jackal and fox populations more efficiently.

The team recommends using MegaDetector to do a granular detection of animals in the images, and to consider decreasing the lower confidence threshold if the researcher desires to reduce the count of false negatives. These images should then be put through our software in order to get a granular classification of the animal(s) present in the images. It is recommended that the researcher classify the animal images of low classification confidence manually, according to the confidence threshold lower bound that they see fit. Following this, the images may be analyzed in Timelapse as would be otherwise standard procedure.

REFERENCES

ADSSC. (n.d.). *About.* Retrieved January 2023, from https://www.adssc.org/en/about/

ADSSC. (n.d.). *Segev Nitzan Ph.D.* Retrieved January 2023, from https://www.adssc.org/en/segev-nitzan-phd-desert-ecology/

Ahamed, F., Farid, F., Gordon, S., et al. (2022). An Interpretable Artificial Intelligence Based Smart Agriculture System. *Tech Science Press.* Retrieved from https://researchdirect.westernsydney.edu.au/islandora/object/uws:63228/datastream/PDF/view.

Andrews, K., Flemons, P., Maclagan, S.J., et al. (2017). Animal Recognition and Identification with Deep Convolutional Neural Networks for Automated Wildlife Monitoring. IEEE International Conference on Data Science and Advanced Analytics (DSAA). Retrieved from https://ieeexplore.ieee.org/abstract/document/8259762/authors#authors.

Baker, M.A.B., Herbert Chan, H.W., & Nichols, J.A. (2019). Machine learning: applications of artificial intelligence to imaging and diagnosis. *National Library of Medicine.* Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6381354/.

Barocas, A., Hefner, R., Ucko, M., et al. (2018). Behavioral adaptations of a large carnivore to human activity in an extremely arid landscape. Animal Conservation, 21(5), 433–443. doi: 10.1111/acv.12414

Bayne, E., Boutin, S., Burton, C., et al. (2015). REVIEW: Wildlife camera trapping: a review and recommendations for linking surveys to ecological processes. *Journal of Applied Ecology.* Retrieved from https://doi.org/10.1111/1365-2664.12432.

Berger-Tal, O. & Lahoz-Monfort, J. J. (2018). Conservation technology: The next generation. *Society for Conservation Biology.* Retrieved from https://doi.org/10.1111/conl.12458.

Butz, A., Diefenbach, S., & Ullrich, D. (2021). The Development of Overtrust: An Empirical Simulation and Psychological Analysis in the Context of Human–Robot Interaction. *Frontiers in Robotics and AI.* Retrieved from https://www.frontiersin.org/articles/10.3389/frobt.2021.554578/full.

Chao, W., Chen, J. K., Feng, L., et al. (2022). Animal Detection and Classification from Camera Trap Images Using Different Mainstream Object Detection Architectures. *Open Access Journals.* Retrieved from https://doi.org/10.3390/ani12151976.

Cowley, F.C., Falzon, G., Hadavi, N., et al. (2021). Automated Muzzle Detection and Biometric Identification via Few-Shot Deep Transfer Learning of Mixed Breed Cattle. Data-Driven Agricultural Innovations. Retrieved from https://www.mdpi.com/2073-4395/11/11/2365.

Faragalla, A. A. (1988). Impact of agrodesert on a desert ecosystem. *Journal of Arid Environments*, 15(1), 99–102. Retrieved from https://doi.org/10.1016/S0140-1963(18)31010-3.

Greenville, A. C., Wardle, G. M., & Dickman, C. R. (2017). Desert mammal populations are limited by introduced predators rather than future climate change. *Royal Society Open Science*, 4(11). doi: 10.1098/rsos.170384

Hobbs MT, Brehme CS (2017) An improved camera trap for amphibians, reptiles, small mammals, and large invertebrates. PLoS ONE 12(10): e0185026. https://doi.org/10.1371/journal.pone.0185026

Kaluarachchi, T., Nanayakkara, S., & Reis, A. (2021). A Review of Recent Deep Learning Approaches in Human-Centered Machine Learning. *Sensors.* Retrieved from https://doi.org/10.3390/s21072514.

Keras. (2020). Transfer learning & fine-tuning. *Keras.i*o. Retrieved from https://keras.io/guides/transfer_learning/.

Khoshgoftaar, T. M., Wang, D., & Weiss, K. (2016). A survey of transfer learning. Journal of Big Data. Retrieved from https://doi.org/10.1186/s40537-016-0043-6.

Moran, S. (2003). Checklist of vertebrate damage to agriculture in Israel, updated for 1993-2001. 511 Phytoparasitica, 31(2), 109–117. doi: 10.1007/BF02980779

Newsome, T. M., Dellinger, J. A., Pavey, C. R., et al. (2015). The ecological effects of providing resource subsidies to predators. *Global Ecology and Biogeography*, 24(1), 1–11. doi: 10.1111/geb.12236

Portnov, B A, and U N Safriel. “Combating Desertification in the Negev: Dryland Agriculture vs. Dryland Urbanization.” Journal of Arid Environments 56, 659-680. Academic Press, 23 Sept. 2003. www.sciencedirect.com/science/article/pii/S0140196303000879.

Research Gate. Research Gate- Nitzan Segev. Nitzan Segev. Retrieved January 2023, from https://www.researchgate.net/profile/Nitzan-Segev.

Richmond, D. & Xie, Y. (2018). Pre-training on Grayscale ImageNet Improves Medical Image Classification. European Conference on Computer Vision. Retrieved from https://openaccess.thecvf.com/content_ECCVW_2018/papers/11134/Xie_Pre-training_on_Grayscale_ImageNet_Improves_Medical_Image_Classification_ECCVW_2018_paper.pdf.

Sarker, I. H. (2021). Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *Advances in Computational Approaches for Artificial Intelligence, Image Processing, IoT and Cloud Applications.* Retrieved from https://link.springer.com/article/10.1007/s42979-021-00815-1.

Shapira, I., Sultan, H., & Shanas, U. (2008). Agricultural farming alters predator-prey interactions in nearby natural habitats. Animal Conservation, 11(1), 1–8. doi: 10.1111/j.1469- 1795.2007.00145.x

Timelapse. (2021). Timelapse: An Image Analyser for Camera Traps. *Greenberg Consulting Inc.* Retrieved from https://saul.cpsc.ucalgary.ca/timelapse/.

APPENDIX A

In-depth Explanation of Neural Networks

The premise of a neural network is that it is a multi-layered set of neurons that stores values known as activations. The activation for each neuron is calculated in the following way:

$$a' = f(wa + b)$$

where a' represents the calculated activation, f is the activation function, w represents the weight-matrix, a represents a vector containing all of the previous layer's neuron activations, and b represents the bias of the neuron for which the activation is being calculated (a real number). This equation describes the feedforward nature of a network. There are four equations that represent the backpropagation of a network, and they use an error value starting at the output layer of the network, and backpropagate this error back through the network, at which point the weights and biases of the network can be updated (Nielsen).

In a traditional neural network, each neuron in a given layer is connected to every neuron in the next layer. These are known as dense layers. The final layer in a neural network contains as many neurons as the number of possible outputs. For example, if a network were trying to classify an image into two categories, the final layer would have two neurons, and so on. This can be seen in figure A1.

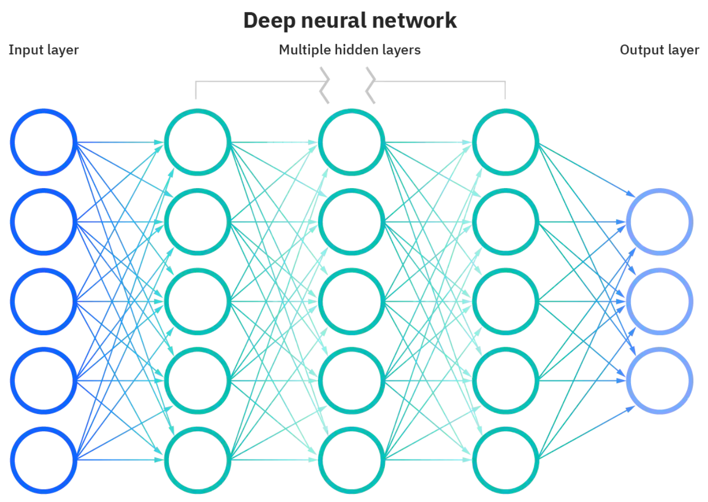


Figure A1. Structure of a deep neural network. Retrieved from https://www.ibm.com/topics/neural-networks.

The type of neural network used in the project is a convolutional neural network (CNN). CNNs are one of the most efficient models to perform image classification (O'Shea and Nash). This is because CNNs utilize three dimensions (height, width, depth) of images (Nguyen et al.). Height and width refer to the pixel dimensions of the image, while depth refers to the color channels of the image. For example, the depth of a grayscale image is one, while the depth of an RGB image is three. Furthermore, not all layers in a CNN are dense layers. CNNs utilize local connectivity, in which each neuron is connected to a local cluster of neurons in the next layer, drastically reducing the time and memory required to analyze inputs (Nguyen et al.).

APPENDIX B

GitHub Repository

The codebase can be found here: https://github.com/CitrusFroot/IQPMammalsC23.git

APPENDIX C

User manual

The user manual for the application can be found here: Current User Manual

APPENDIX D

Timelapse Manuals

Manuals on how to use the Timelapse software can be found here:
http://saul.cpsc.ucalgary.ca/timelapse/pmwiki.php?n=Main.UserGuide

The Timelapse manual on image recognition can be found here:
http://saul.cpsc.ucalgary.ca/timelapse/uploads/Guides/TimelapseImageRecognitionGuide.pdf

The Timelapse manual on utilizing CSVs can be found here:
http://saul.cpsc.ucalgary.ca/timelapse/uploads/Guides/TimelapseReferenceGuide.pdf

APPENDIX E

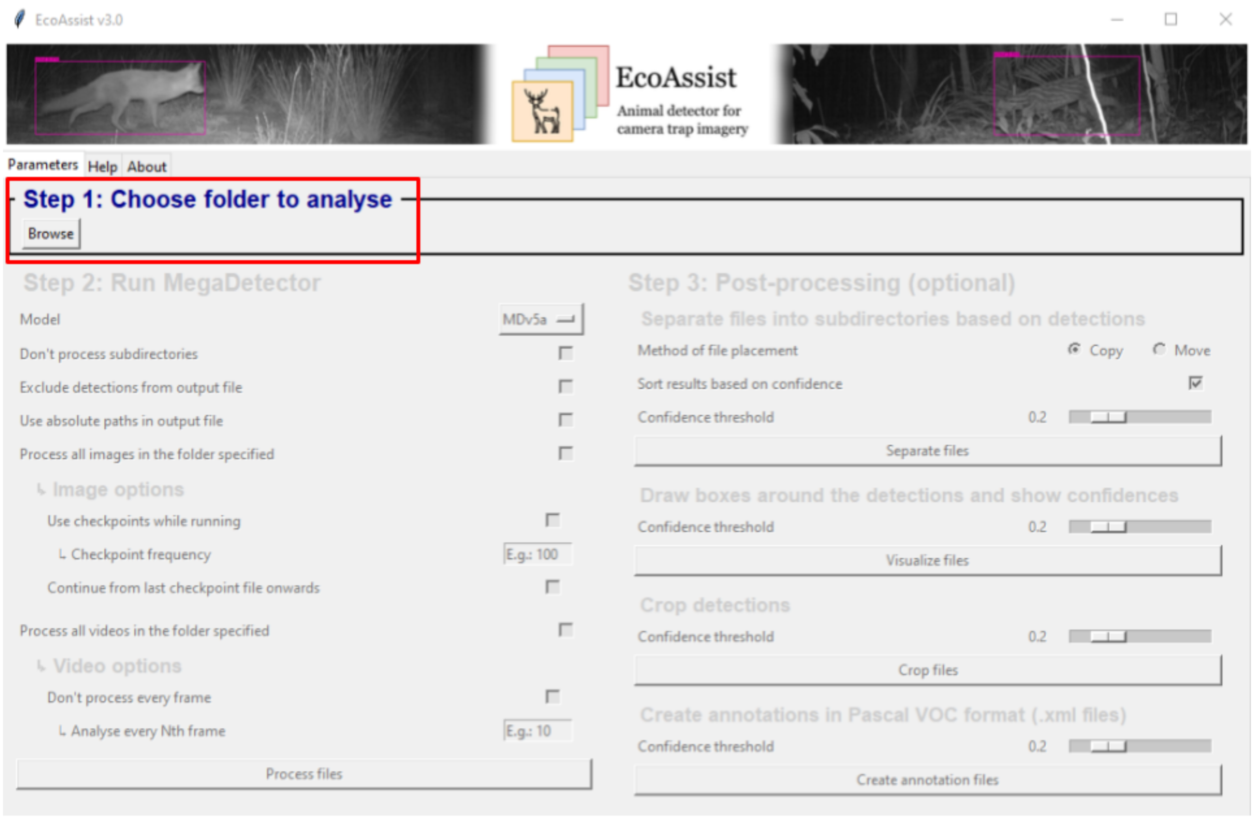
MegaDetector Links

MegaDetector About page:
https://github.com/microsoft/CameraTraps/blob/main/megadetector.md

EcoAssist: https://github.com/PetervanLunteren/EcoAssist

APPENDIX F

EcoAssist Instructions



EcoAssist v3.0

EcoAssist
Animal detector for camera trap imagery

Parameters | Help | About

Step 1: Choose folder to analyse
Browse

Step 2: Run MegaDetector

Model	MDv5a
Don't process subdirectories	☐
Exclude detections from output file	☐
Use absolute paths in output file	☐
Process all images in the folder specified	☐
↳ Image options	
Use checkpoints while running	☐
↳ Checkpoint frequency	E.g.: 100
Continue from last checkpoint file onwards	☐
Process all videos in the folder specified	☐
↳ Video options	
Don't process every frame	☐
↳ Analyse every Nth frame	E.g.: 10

Process files

Step 3: Post-processing (optional)

Separate files into subdirectories based on detections

Method of file placement	[✓ Copy] Move
Sort results based on confidence	✓
Confidence threshold	0.2

Separate files

Draw boxes around the detections and show confidences

Confidence threshold	0.2

Visualize files

Crop detections

Confidence threshold	0.2

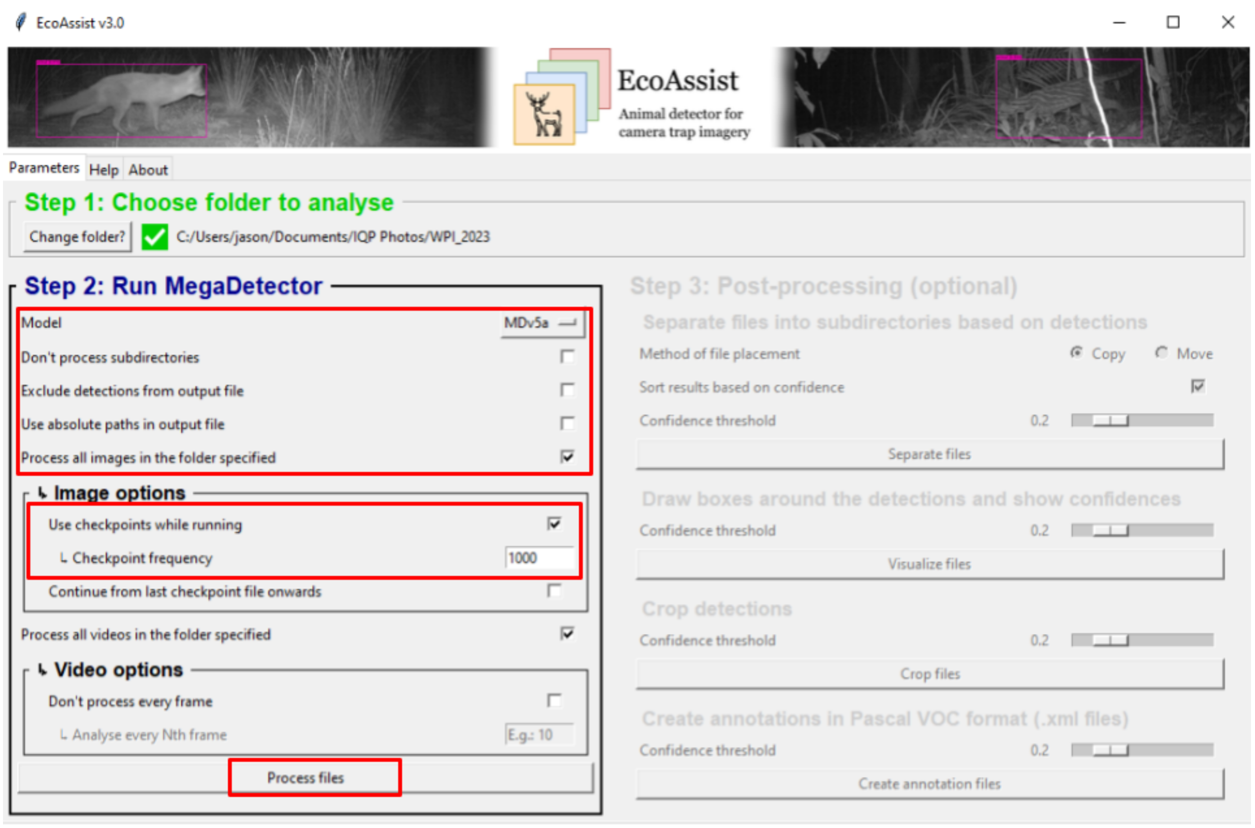
Crop files

Create annotations in Pascal VOC format (.xml files)

Confidence threshold	0.2

Create annotation files

Figure F1: EcoAssist on startup



EcoAssist v3.0

Step 1: Choose folder to analyse

Change folder? ✓ C:/Users/jason/Documents/IQP Photos/WPI_2023

Step 2: Run MegaDetector

Model MDv5a

Don't process subdirectories ☐

Exclude detections from output file ☐

Use absolute paths in output file ☐

Process all images in the folder specified ☑

Image options

Use checkpoints while running ☑

Checkpoint frequency 1000

Continue from last checkpoint file onwards ☐

Process all videos in the folder specified ☑

Video options

Don't process every frame ☐

Analyse every Nth frame E.g.: 10

Process files

Step 3: Post-processing (optional)

Separate files into subdirectories based on detections

Method of file placement [✓ Copy] Move

Sort results based on confidence ☑

Confidence threshold 0.2

Separate files

Draw boxes around the detections and show confidences

Confidence threshold 0.2

Visualize files

Crop detections

Confidence threshold 0.2

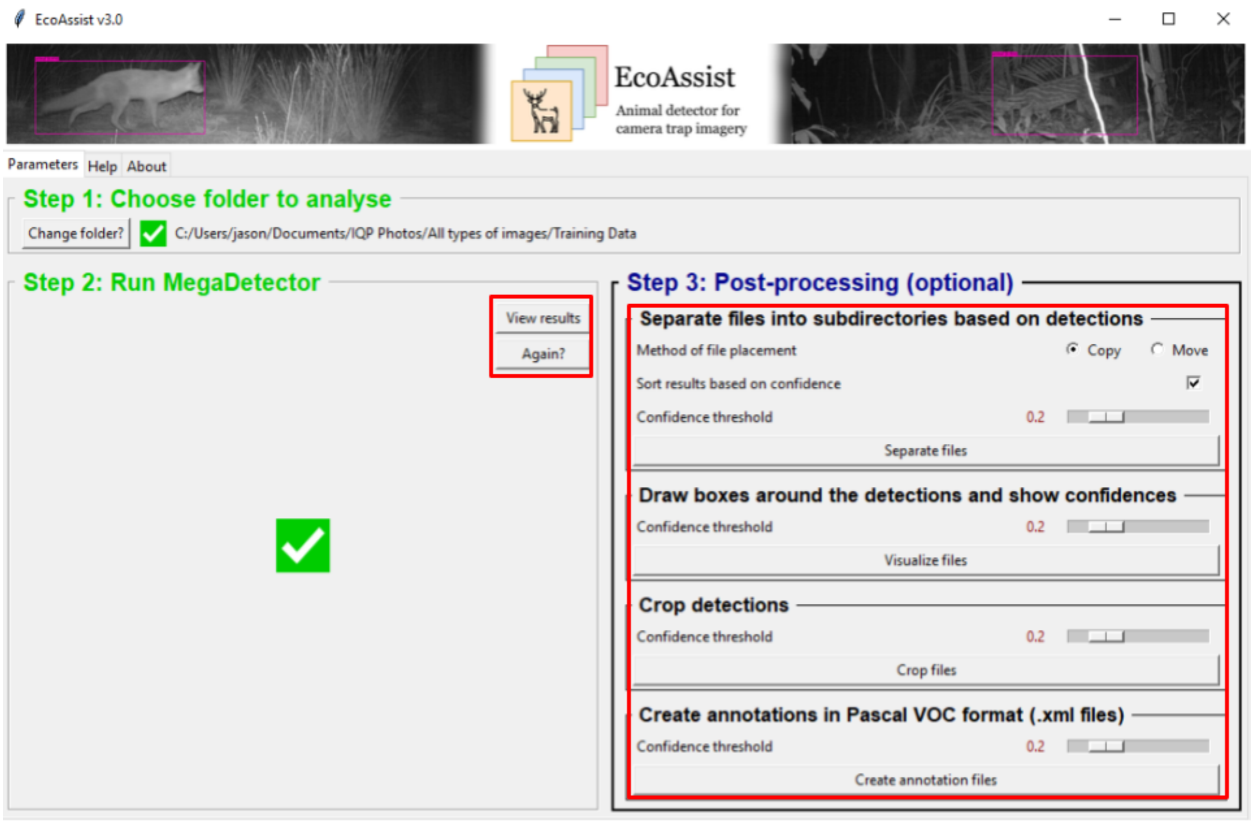
Crop files

Create annotations in Pascal VOC format (.xml files)

Confidence threshold 0.2

Create annotation files

Figure F2: EcoAssist with options for running MegaDetector



EcoAssist v3.0

EcoAssist
Animal detector for camera trap imagery

Parameters Help About

Step 1: Choose folder to analyse

Change folder? ✓ C:/Users/jason/Documents/IQP Photos/All types of images/Training Data

Step 2: Run MegaDetector

View results
Again?

Step 3: Post-processing (optional)

Separate files into subdirectories based on detections

Method of file placement [✓ Copy] Move
Sort results based on confidence ✓
Confidence threshold 0.2

Separate files

Draw boxes around the detections and show confidences

Confidence threshold 0.2

Visualize files

Crop detections

Confidence threshold 0.2

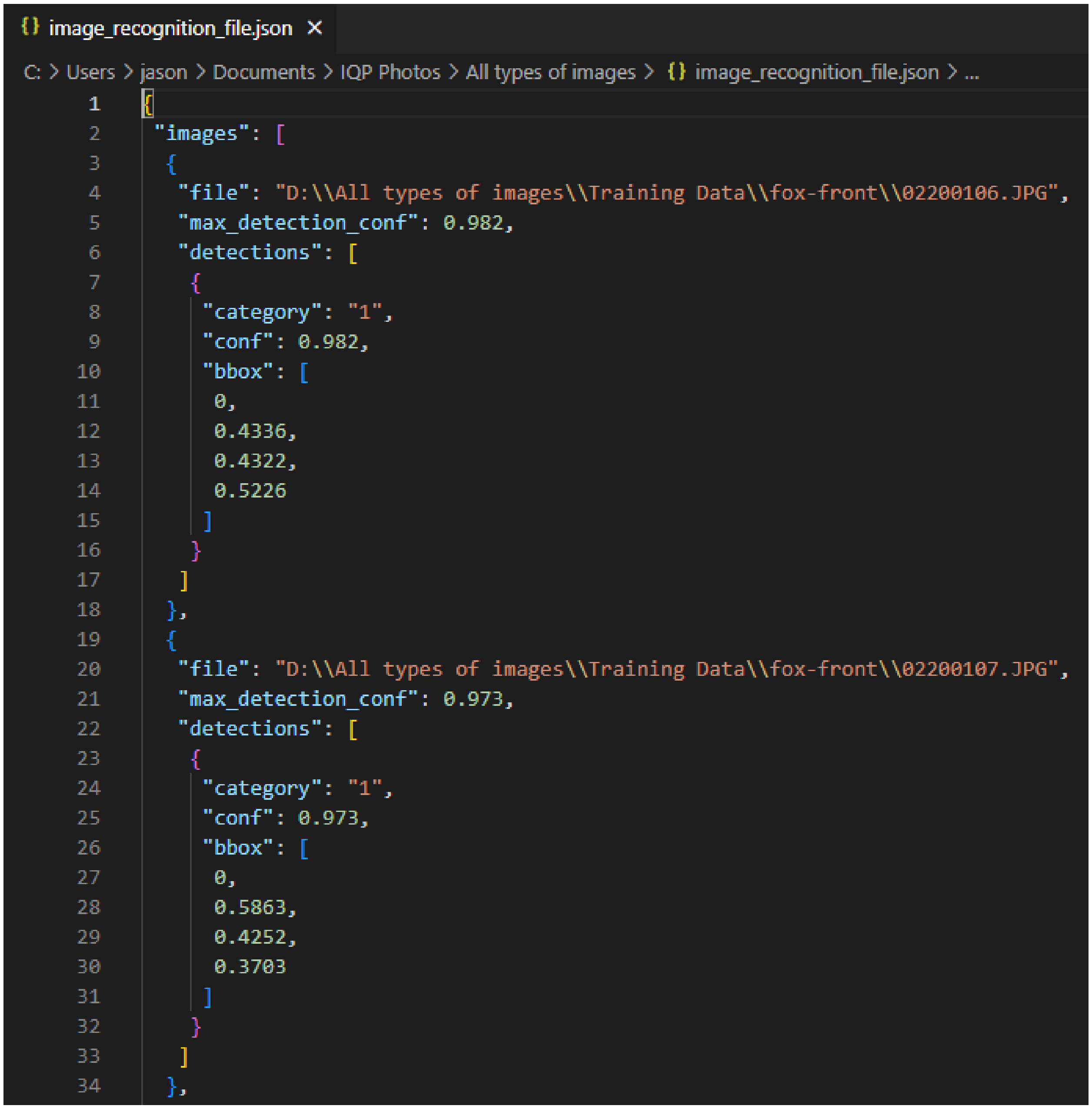
Crop files

Create annotations in Pascal VOC format (.xml files)

Confidence threshold 0.2

Create annotation files

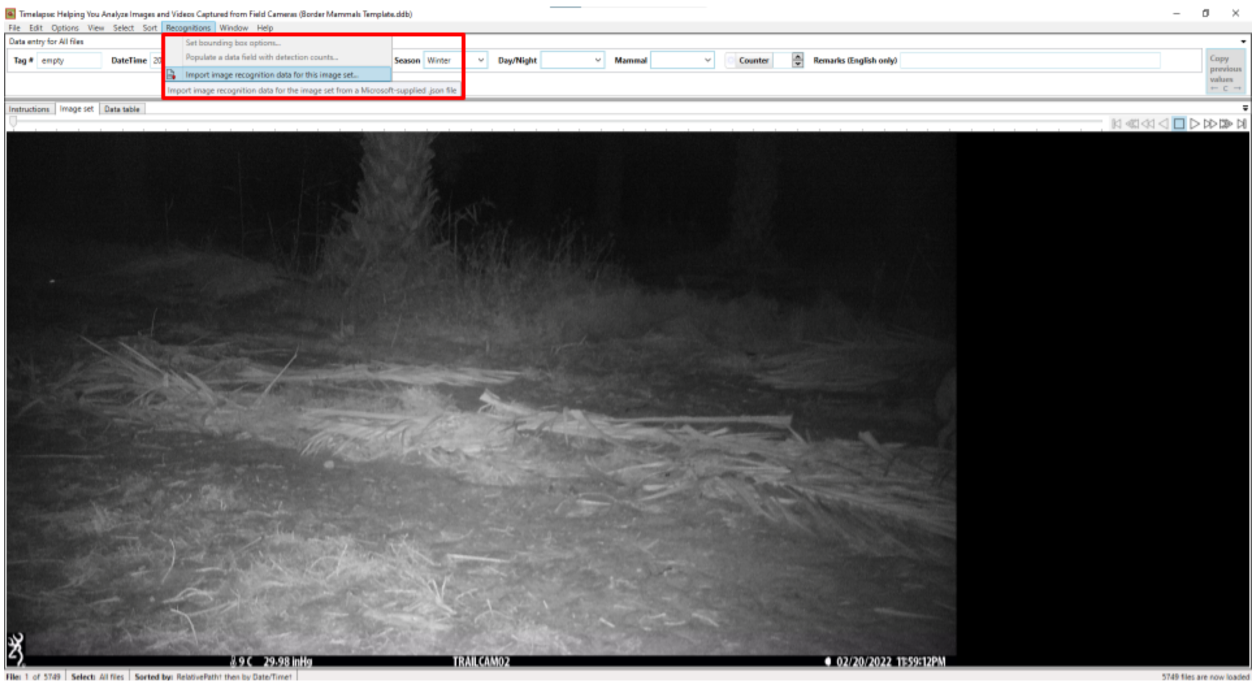
Figure F3: EcoAssist after completing run


image_recognition_file.json ×

C: > Users > jason > Documents > IQP Photos > All types of images > image_recognition_file.json > ...

```
{
 "images": [
  {
   "file": "D:\\All types of images\\Training Data\\fox-front\\02200106.JPG",
   "max_detection_conf": 0.982,
   "detections": [
    {
     "category": "1",
     "conf": 0.982,
     "bbox": [
      0,
      0.4336,
      0.4322,
      0.5226
     ]
    }
   ]
  },
  {
   "file": "D:\\All types of images\\Training Data\\fox-front\\02200107.JPG",
   "max_detection_conf": 0.973,
   "detections": [
    {
     "category": "1",
     "conf": 0.973,
     "bbox": [
      0,
      0.5863,
      0.4252,
      0.3703
     ]
    }
   ]
  },
```

Figure F4: Example of image data in a JSON File


Timelapse: Helping You Analyze Images and Videos Captured from Field Cameras (Border Mammals Template.ddb)

File Edit Options View Select Sort Recognitions Window Help

Set bounding box options...
Populate a data field with detection counts...
Import image recognition data for this image set...
Import image recognition data for the image set from a Microsoft-supplied .json file

Data entry for All files

Tag # empty | DateTime 20 | Season Winter | Day/Night | Mammal | Counter | Remarks (English only)

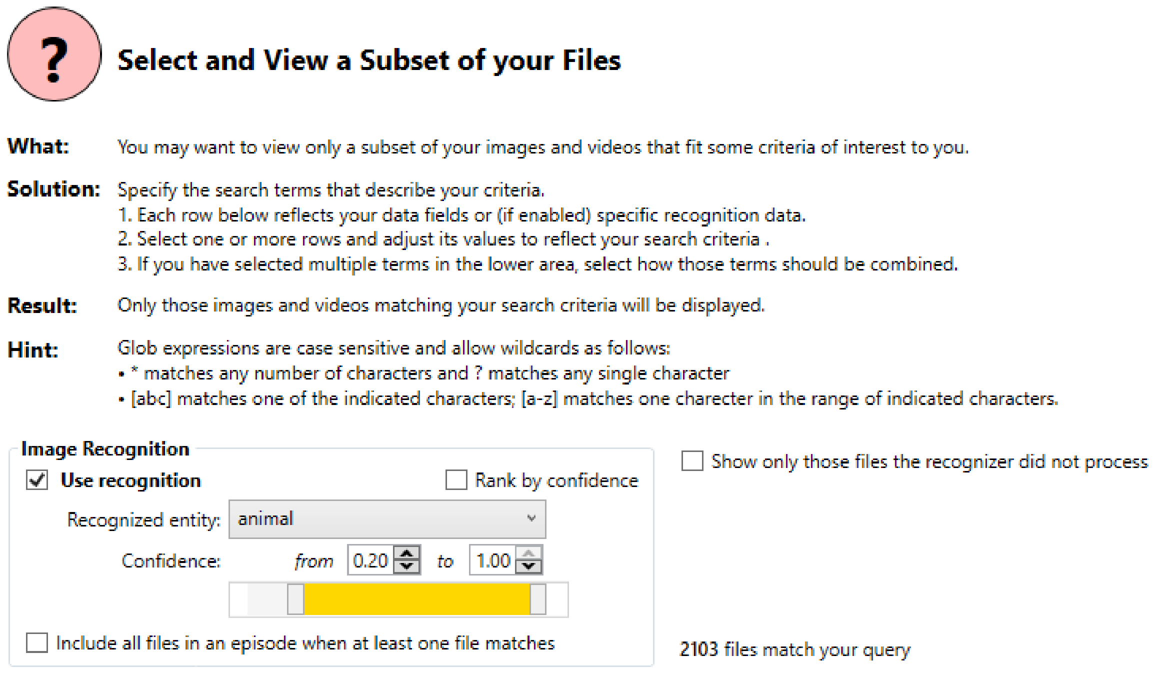
Copy previous values

Instructions | Image set | Data table

9 C 29.98 inHg TRAILCAM02 02/20/2022 11:59:12PM

File: 1 of 5749 | Select: All files | Sorted by: RelativePath! then by Date/Time! 5749 files are now loaded

Figure F5: Timelapse with recognition options shown


Select and View a Subset of your Files

What: You may want to view only a subset of your images and videos that fit some criteria of interest to you.

Solution: Specify the search terms that describe your criteria.
1. Each row below reflects your data fields or (if enabled) specific recognition data.
2. Select one or more rows and adjust its values to reflect your search criteria .
3. If you have selected multiple terms in the lower area, select how those terms should be combined.

Result: Only those images and videos matching your search criteria will be displayed.

Hint: Glob expressions are case sensitive and allow wildcards as follows:
• * matches any number of characters and ? matches any single character
• [abc] matches one of the indicated characters; [a-z] matches one charecter in the range of indicated characters.

Image Recognition

☑ **Use recognition** ☐ Rank by confidence

Recognized entity: animal

Confidence: *from* 0.20 *to* 1.00

☐ Include all files in an episode when at least one file matches

☐ Show only those files the recognizer did not process

2103 files match your query

Figure F6: Settings for sorting images in Timelapse using recognition data

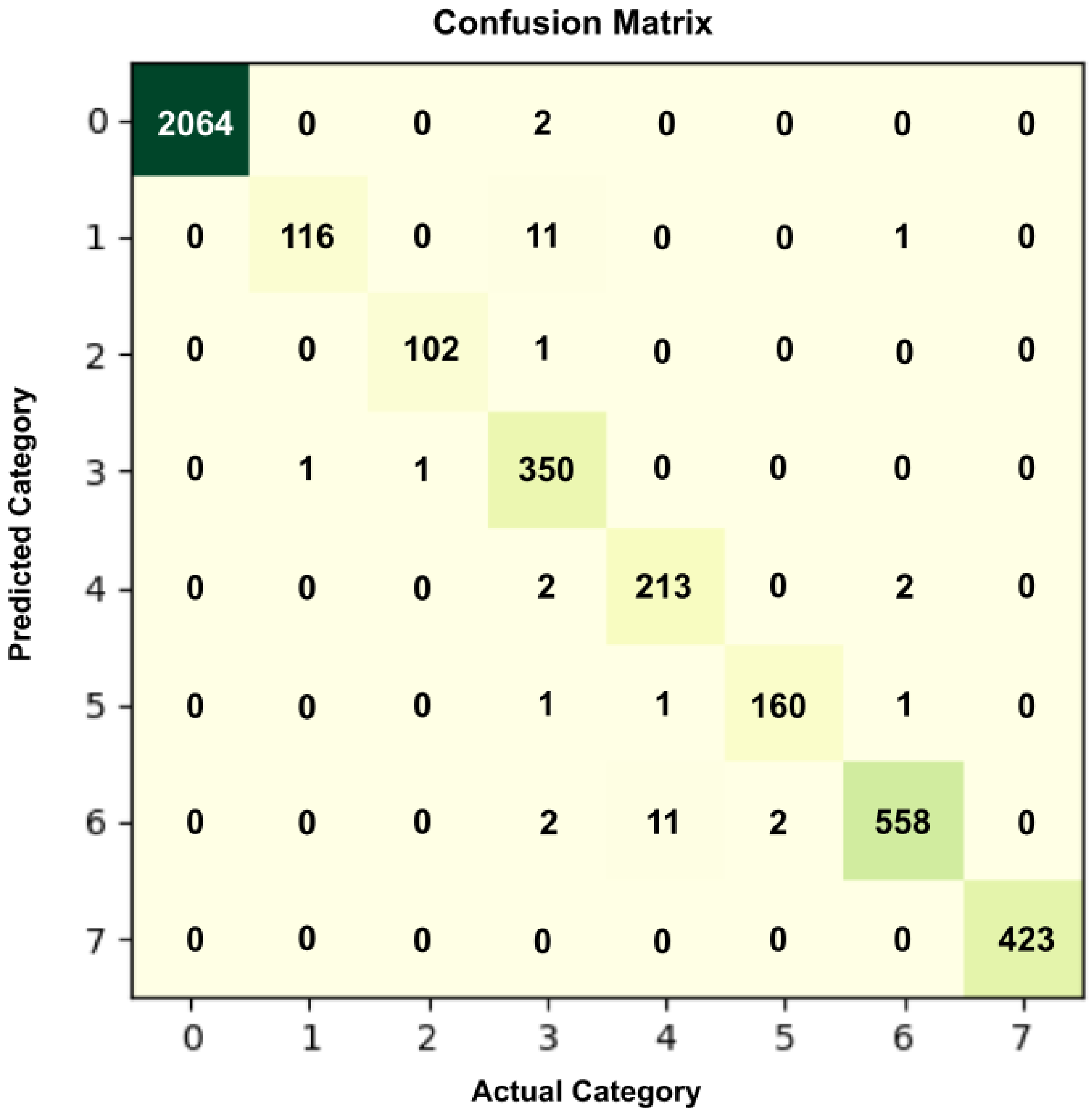
APPENDIX G

VGG-16 Description

https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/

APPENDIX H

Expanded Confusion Matrix



	0	1	2	3	4	5	6	7
0	2064	0	0	2	0	0	0	0
1	0	116	0	11	0	0	1	0
2	0	0	102	1	0	0	0	0
3	0	1	1	350	0	0	0	0
4	0	0	0	2	213	0	2	0
5	0	0	0	1	1	160	1	0
6	0	0	0	2	11	2	558	0
7	0	0	0	0	0	0	0	423

Figure H1: Full Confusion Matrix

Figure H1 shows a full confusion matrix generated by the AI predictions. The labels 0-7 correspond to empty, fox-back, fox-front, fox-side, jackal-back, jackal-front, jackal-side, and other, respectively. As shown by Figure H1, on this run the AI misclassified 39 images out of a set of 4025. However, its main mistakes were confusion between categories one and three as well as four and six. This means that the back and side of each animal were frequently confused, but the animal itself was correct.