Classifying Small Animals in ADHriDFT Camera Traps

Course Project for DS5220, Fall 2024

Elizabeth Coquillette, Northeastern ID #002453121

Overview

The purpose of this project is to examine how to build a machine learning model that can accurately classify images of animals from a particular dataset. The dataset in question is the Ohio Small Animals dataset bosted on LILA BC. The purpose of this data collection is to track the populations of various animals in their native habitats, but it produces a very large quantity of images that take time to individually classify. Therefore, a machine learning model that could classify images automatically would be very valuable in managing and interpreting this data. I decided to try two different options: first, building a convolutional neural network model from scratch, and second, using the ResNet5050 pretrained model, both with and without fine tuning. I selected ResNet50 because it is a common and well-respected pre-trained model for image classification.

A graduate student at Ohio State has already attempted to address this problem (Balasubramaniam, "Optimized Classification in Camera Trap Images"). As his project was a master's thesis, rather than a course project, his was much more detailed and expansive than mine, and his used a larger dataset of 1.2 million images that was accessible to him via Ohio State, instead of the dataset of 118,555 that was available to the public on LILA BC. However, I think it is still useful for different people to look at this issue from different perspectives.

I obtained better results for the CNN model (test accuracy of 0.92) than for the ResNet50 model (test accuracy of 0.85), even after several rounds of fine-tuning on the ResNet50 model. This does make some sense in the context of this dataset, as the AHDriFT traps by definition produce a fairly consistent set of images that is well-suited to simple image classification models.

Experiment Setup

a. Dataset and Exploratory Data Analysis

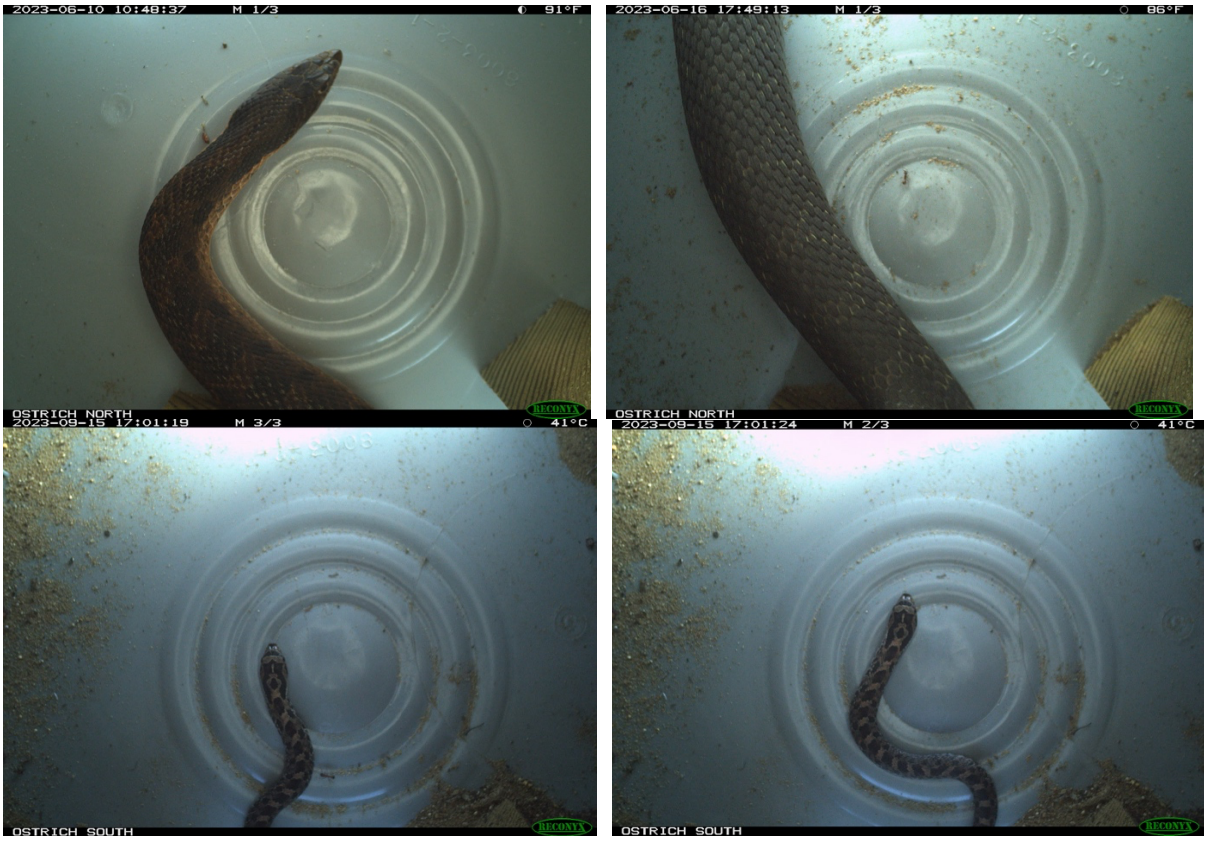
I downloaded this dataset from LILA BC repository, which stands for the Labeled Information Library of Alexandria: Biology and Conservation. They have a variety of datasets posted on their website related to animal classification and wildlife conservation efforts, intended for use by "those who want to harness machine learning for biology and conservation". The dataset accessible via LILA BC consists of a JSON metadata file and 118,555 images of animals that have been caught in specialized humane traps called adapted-Hunt drift fence technique (AHDriFT) traps by a group of researchers at The Ohio State University (Amber, Gregory, and Peterman, "Evaluation of the AHDriFT Camera Trap System to Survey for Small Mammals and Herpetofauna"). These traps are designed to naturally corral wildlife into a spot underneath a camera, take an image, and then release the animal. The traps were distributed throughout wildlife habitats in northern Ohio in an attempt to evaluate the population levels and changing population patterns of species in those areas, which ultimately will help with conservation and habitat preservation efforts for those species. The dataset that I used includes images from 168 different trap locations, with three cameras in each trap location for a total of 504 cameras. Each image is labeled with the animal species visible in that image. While this is the labeled dataset that is available to the public, researchers at Ohio State have much larger amounts of data from these cameras which has not yet been published.

An advantage of this dataset is that the nature of the AHDriFT traps means that the images are quite consistent in their backgrounds. Challenges in the dataset include that the images were taken with different light conditions during both day and night, and often only part of the animal was in the image. Therefore, the classification algorithm needs to identify species not just through their whole body, but also through their tail only, head only, etc. The below sets of images illustrate the variations in the dataset. The two different snake species illustrate the similarity between images in a certain animal class, which are harder to differentiate from one another than, for instance, differentiating between a snake and a chipmunk. The diagram below

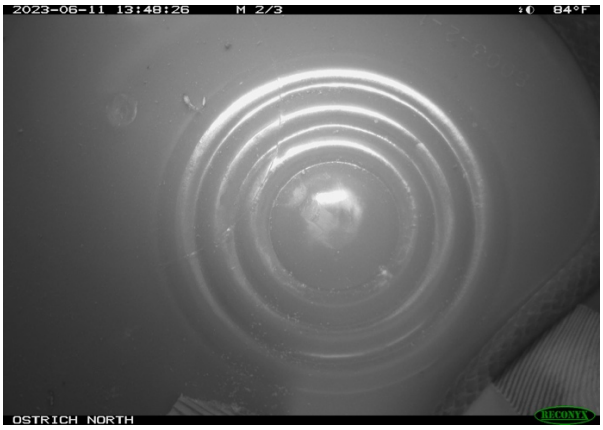
the images illustrates the number of species identified in each category. It indicates that certain categories, such as snakes, birds, and mammals, are more difficult than the others because they span a larger number of individual species, each of which have similar characteristics to one another.

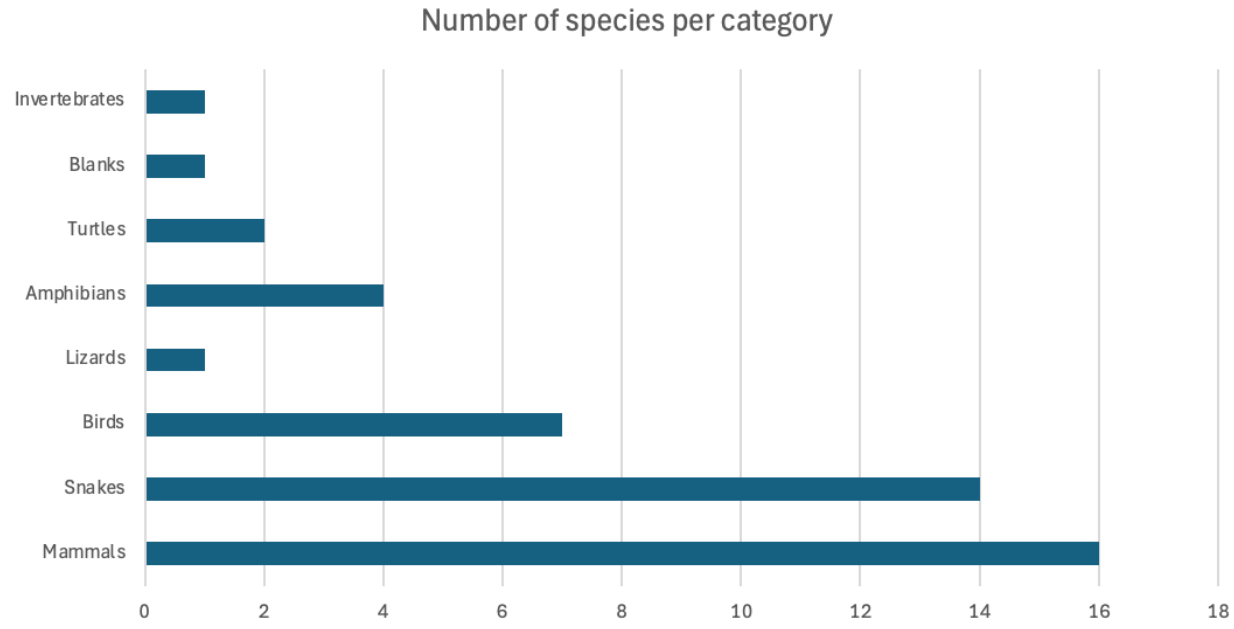
Example images of Eastern Chipmunk:

Example Images of Eastern Hog-Nosed Snake:

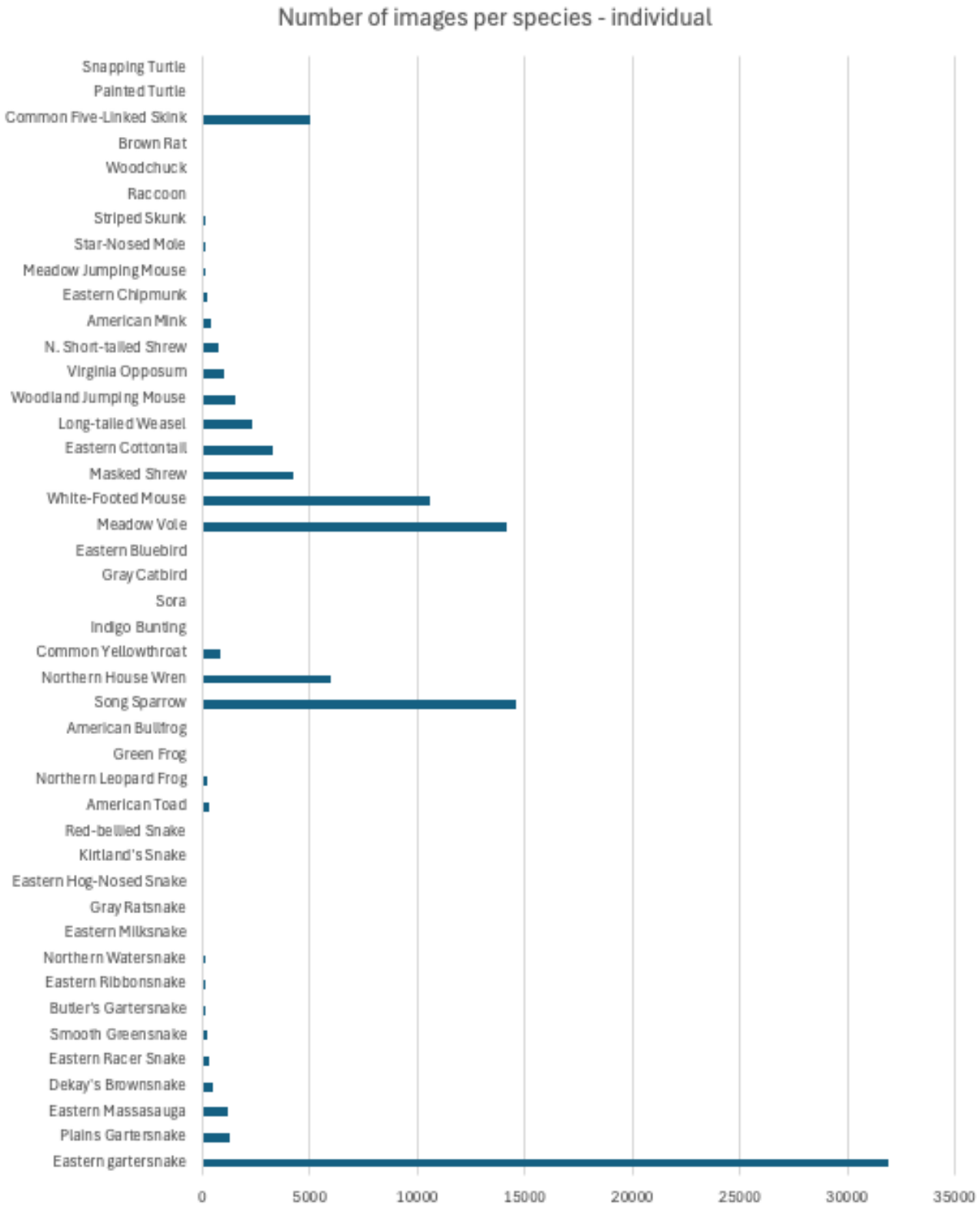


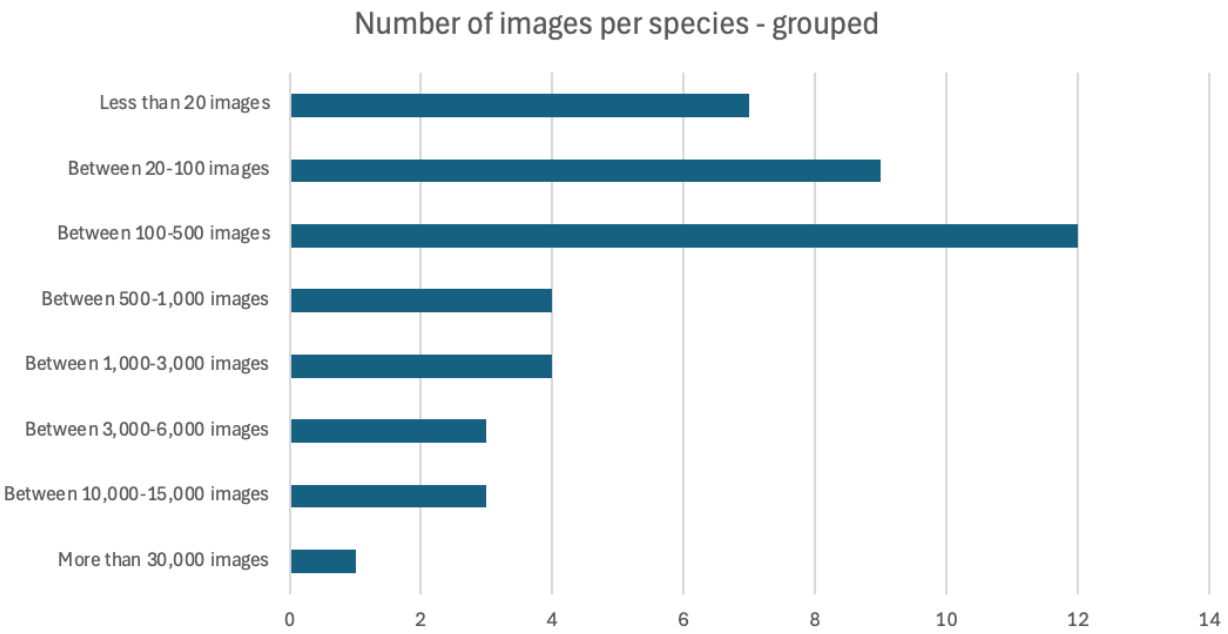
Example Images of Eastern Racer Snake:





Another disadvantage of this dataset is that while it was quite large, the number of images for each species varied substantially. This ranged from 31,900 images for the eastern gartersnake to 1 image for the eastern bluebird. The below diagrams illustrate the distribution of number of images per species, both broken out into individual species and grouped into size categories.


Number of images per species - individual
Snapping Turtle
Painted Turtle
Common Five-Linked Skink
Brown Rat
Woodchuck
Raccoon
Striped Skunk
Star-Nosed Mole
Meadow Jumping Mouse
Eastern Chipmunk
American Mink
N. Short-tailed Shrew
Virginia Opposum
Woodland Jumping Mouse
Long-tailed Weasel
Eastern Cottontail
Masked Shrew
White-Footed Mouse
Meadow Vole
Eastern Bluebird
Gray Catbird
Sora
Indigo Bunting
Common Yellowthroat
Northern House Wren
Song Sparrow
American Bullfrog
Green Frog
Northern Leopard Frog
American Toad
Red-bellied Snake
Kirtland's Snake
Eastern Hog-Nosed Snake
Gray Ratsnake
Eastern Milksnake
Northern Watersnake
Eastern Ribbonsnake
Butler's Gartersnake
Smooth Greensnake
Eastern Racer Snake
Dekay's Brownsnake
Eastern Massasauga
Plains Gartersnake
Eastern gartersnake
0
5000
10000
15000
20000
25000
30000
35000



b. Data Preprocessing and Cleaning

The dataset had several initial issues that I addressed through preprocessing and cleaning. First, the accompanying JSON metadata file had a lot of information, such as time at which the image was taken and at which trap the image was taken, it did not have the animal classifications in it. Instead, the animal classifications were identified by the images being sorted into folders, each of which was labeled with a species. Therefore, I created my own classification system by writing code that sorted through the files and attached the species classification to each of the images in the respective folder.

Second, the dataset had several corrupted files. After trying several methods, I could not get my code to automatically screen them out, so instead I wrote code that went through the images and opened and tried to process each one, and then it printed out the names and locations of the images that were corrupted. I then went through the dataset manually and removed the corrupted images.

Third, there were several species that had only a very small number of images attributed to them. I decided to remove five species that each had less than 15 images: red-bellied snake (9 images), American bullfrog (12 images), gray catbird (10 images), eastern bluebird (1 image),

and brown rat (8 images). Additionally, I combined the two turtle species—painted turtle (23 images) and snapping turtle (6 images)—into one category titled Turtle.

c. Implementation and Architecture

As discussed in the summary above, I ran both a CNN model built from scratch and the ResNet50 pre-trained model, without fine tuning and then with several different levels of fine tuning.

For both models, I set the batch size at 32, because that is a standard batch size and seemed reasonable for the size of my training dataset. My training dataset comprised of 92,943 images (80% of the dataset) and my test dataset comprised of 23,236 images (20% of the dataset). A batch size of 32 results in steps per epoch of 92,943 / 32 = 2904 steps, which seemed reasonable for this purpose based on numbers I have seen in other models. I then resized the images to 128x128 to fit the expected inputs for tensorflow and converted all of the labels into integers. Next, I created a TensorFlow dataset using the built-in Python TensorFlow library.

I wrote all of my code in Python and executed both models in a Jupyter Lab notebook.

For my convolutional neural network, I used 9 layers. Below is a summary of each layer, its type, and what it does. I chose the ReLU activation function for the convolutional layers and the softmax activation function for the last fully connected layer.

Layer	Function used	Layer type	Layer description
	InputLayer	Input layer	Defines the shape of the input images (128 x 128 x 3, where 3 represents the color channels RGB).
1	Conv2D	1st convolutional layer	Applies 32 filters of size (3,3) to the input. Uses the activation function ReLU, which introduces non-linearity.
2	MaxPooling2D	1st pooling layer	Performs down-sampling with a pooling window of (2,2) to reduce the spatial dimensions.
3	Conv2D	2nd convolutional layer	Applies 64 filters of size (3,3), also using ReLU activation, to capture more complex features.

4	MaxPooling2D	2nd pooling layer	Performs down-sampling with a pooling window of (2,2) to reduce the spatial dimensions.
5	Conv2D	3rd convolutional layer	Applies 128 filters of (3,3), also using ReLU activation, to capture even more complex features.
6	MaxPooling2D	3rd pooling layer	Performs down-sampling with a pooling window of (2,2) to reduce the spatial dimensions.
7	Flatten	Flattening layer	Converts the 2D feature map into a 1D vector to prepare it for the fully connected layers.
8	Dense	1st fully connected layer	Contains 128 units with ReLU activation and learns the non-linear combinations of the features extracted by previous layers.
9	Dense	2nd fully connected layer	Contains number of units equal to the number of classes. Uses the softmax activation function to produce a probability distribution over the classes.

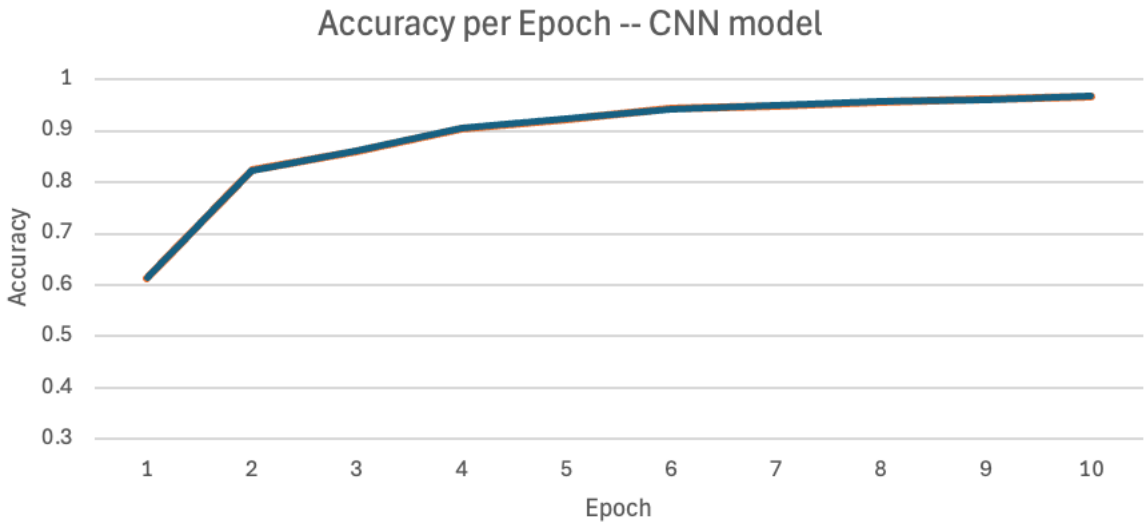
For the ResNet50 model, I used the pre-trained ResNet50 model as the base model. I removed the final fully connected layer of the model and added my own global average pooling layer, fully connected layer, and output layer for classification. For my first run of 10 epochs, I froze the layers in the ResNet50 model and did not use any fine tuning. For each of four additional sets of 5 epochs, I used fine-tuning by unfreezing the top layers of the pretrained ResNet50 model and retraining them. For the first set, I unfroze the top 25 layers, and for the second set, I unfroze the top 50 layers. For the third and fourth sets, I unfroze the top 100 and top 200 layers, respectively. If I had more time to run the models, I would try doing 20 epochs on each of those settings (number of unfrozen layers) to compare, rather than doing them sequentially.

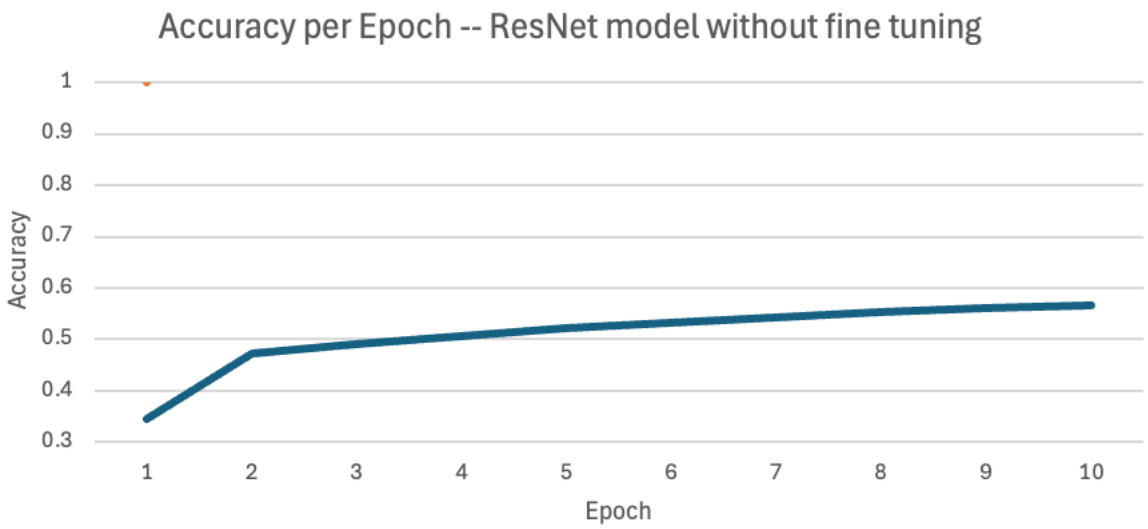
Experiment Results

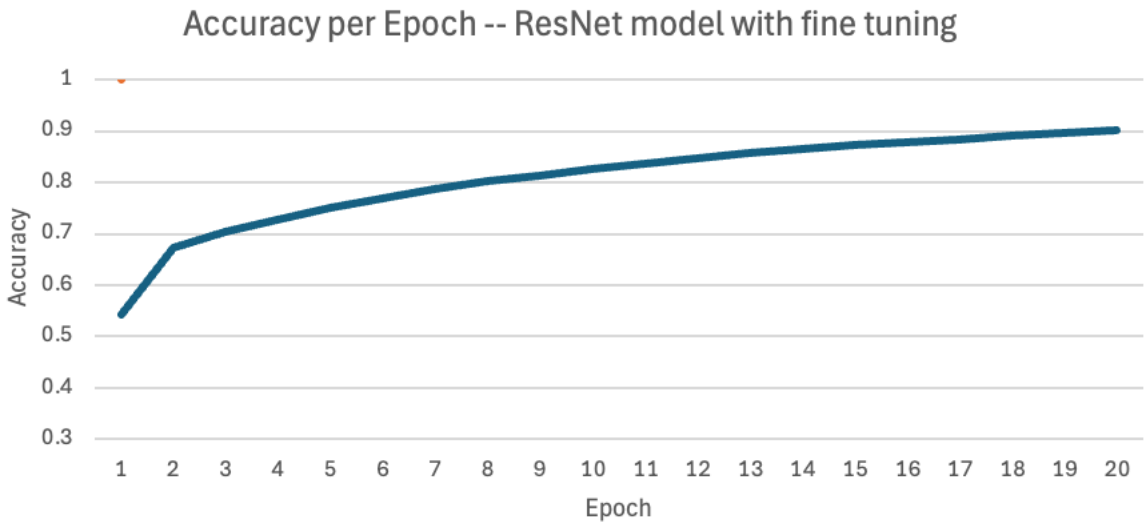
The test accuracies resulting from my model variations are shown in the following table. The 9-layer CNN model got to a test accuracy of 0.92 after 10 epochs. The ResNet50 model got a test accuracy of 0.58 with no fine-tuning and a test accuracy of 0.85 after 20 additional epochs of fine-tuning

Model	Test Accuracy
CNN model with 9 layers and 10 epochs	0.92
ResNet50 model with no fine tuning and 10 epochs	0.58
ResNet50 model with additional 5 epochs of fine tuning on the top 25 layers	0.72
ResNet50 model with additional 5 epochs of fine tuning on the top 50 layers	0.80
ResNet50 model with additional 5 epochs of fine tuning on the top 100 layers	0.84
ResNet50 model with additional 5 epochs of fine tuning on the top 200 layers	0.85

The results across individual epochs are shown in the below diagrams.







Discussion:

If I were to continue further with this project, there are definitely many more things that I could do with this model to improve it. For one, I would test more types of models and more variations on layers in CNN models and fine-tuning variations for pretrained models. Additionally, I would create new models that incorporate more data that just the images. The images also have corresponding metadata showing time of day captured, location captured, and camera angle captured, among other things. I think that including those elements as variables would improve the model, because the model could learn for contextual clues such as nighttime lighting or certain traps that are more likely to entrap certain types of animals.

Conclusion:

I have created and tested a CNN model on the Ohio Small Animals dataset and achieved a test accuracy of 0.92. I also tested a ResNet50 model before fine-tuning and after four different stages of fine tuning. The test accuracies after each training stage of the ResNet50 model were 0.58, 0.72, 0.80, 0.84, and 0.85, respectively.

References:

Ohio Small Animals dataset, Labeled Information Library of Alexandria: Biology and Conservation, https://lila.science/datasets/ohio-small-animals/.

Evan Amber, Gregory Lipps, and William Peterman, "Evaluation of the AHDriFT Camera Trap System to Survey for Small Mammals and Herpetofauna", *Journal of Fish and Wildlife Management* (2021).

Sowbaranika Balasubramaniam, "Optimized Classification in Camera Trap Images: An Approach with Smart Camera Traps, Machine Learning, and Human Inference." The Ohio State University master's thesis, 2024.