STAT 4984 Final Project: Snake Species Image Classification

Megan Schaeb
Department of Statistics
Virginia Tech
meganvs@vt.edu

Abstract

Snakes are found in almost every climate on Earth and have adapted various colorings and patterns to camouflage with their surroundings. With over 3,000 species, identifying snakes can prove troublesome for the untrained eye. This paper explores the use of ResNet-18, a convolutional neural network (CNN) architecture with 18 layers, for snake species image classification using two different optimizers: the Adam optimizer and stochastic gradient descent (SGD) with momentum. Running both models on training and testing images reveals that the model using SGD with momentum is better suited for this image classification problem, as it produces lower training loss and higher testing accuracy than the model with the Adam optimizer.
Code: https://github.com/MeganSchaeb/STAT4984_FinalProject

1 Introduction

When exploring out in nature, one thing people should be wary of is snakes. With over 3,000 species on Earth, more than 150 of which are found in North America, snakes live in almost every climate [11], [12]. Although most species pose no significant threat to humans, about 200 can inflict bites that prove lethal or cause severe injury [12]. According to John Hopkins Medicine, if bitten, a person should try to recall any details of the snake's appearance [10]. Knowledge of the species of snake helps medics administer proper treatment by understanding whether or not the snake was venomous, and if so, to what extent.

However, there are times when a medical professional or nature expert is not readily accessible. In that case, since not everyone possesses the ability to identify snakes, computer vision could be of assistance. An image classification model could help someone identify a species of snake and understand the gravity of a situation if they were to be bitten. Such technology could also be used for recreational purposes for people curious about the wildlife they spot.

2 Related Work

2.1 HackerEarth Challenge

The motivation for undertaking this project came after I recently discovered a "Deep Learning Challenge" to identify snake breed issued by HackerEarth, and as such, there are many existing implementations of classification models that were built in response to this challenge [4].

One solution to the HackerEarth challenge, written by Tanay Shah, used the ResNet101V2 model included in the TensorFlow library [9]. This solution also utilized the RMSprop optimizer, the cross entropy loss function with a learning rate of 0.0008 and decay rate of $1\text{x}10^{-7}$, and a 30% dropout

on the last three layers [9]. This model was trained with a batch size of 54 over 20 epochs and then tested with a batch size of 128 [9]. While the notebook containing this code does not provide an assessment of model performance either by training loss or by accuracy, the use of the ResNet algorithm served as inspiration for my image classification model. The model structure I chose, however, was ResNet-18 instead of ResNet-101, due to time and computer memory constraints. I also opted to use an optimizer, however, I chose to compare Adam and SGD with momentum since both were used frequently in class (STAT 4984). Additionally, I used the cross entropy loss function in my models but increased the learning rate to speed up the code runtime.

2.2 Other Related Work

Outside of the HackerEarth challenge, several other papers have explored the employment of machine learning for snake classification.

In "A comparative study on image-based snake identification using machine learning", authors Mahdi Rajabizadeh and Mansoor Rezghi utilized two neural network algorithms-VGG-16 and MobileNetV2- to classify images of snakes from six species found in Iran [7]. The authors used 500 epochs to train the VGG-16 model and 150 epochs for the MobileNetV2 model, and both were trained using the SGD optimizer with momentum set to 0.9 and a learning rate of 0.0001 [7]. After 150 epochs, the training accuracy on MobileNetV2 surpassed that of VGG-16, and MobileNetV2 also had lower loss [7]. Although I did not explore the MobileNetV2 model for my snake classification problem, the model I chose, ResNet-18, also uses shortcut connections. Additionally, I also included SGD with momentum of 0.9 in one of the models I trained, but I used fewer epochs for training and a larger learning rate to speed up the runtime, as previously mentioned. Some of the challenges with snake identification that the authors mentioned were "wide variations in the pose and deformation of the body" given the flexible nature of snakes' bodies and the fact that snakes camouflage with their surroundings [7], both of which I anticipate my model will struggle with as well.

3 Dataset and Features

The public domain dataset I used for this problem was published on Kaggle by Debadri Dutta, who used a web scraper to gather the images from Google [2]. This dataset contains 13,185 images organized into 35 directories by species name and is stored as `snake_data_original` on Github.

To divide the images into training and testing sets, I used the Python `splitfolders` package with an 80%-20% split for training and testing, respectively [3]. Splitting the data resulted in 10,538 images allocated for training and 2,647 for testing. The `snake_train_test` directory on Github contains the split data in two sub-directories, one for training and one for testing, each of which contains 35 directories of images classified by species name.

Each image contains one species of snake, as well as sometimes background noise such as leaves or grass. After loading the images into Jupyter Notebook, I converted the images to tensors and resized to them have dimensions 224 x 224. Figure 1 below displays a couple of the training images after having been loaded into Jupyter Notebook and transformed.

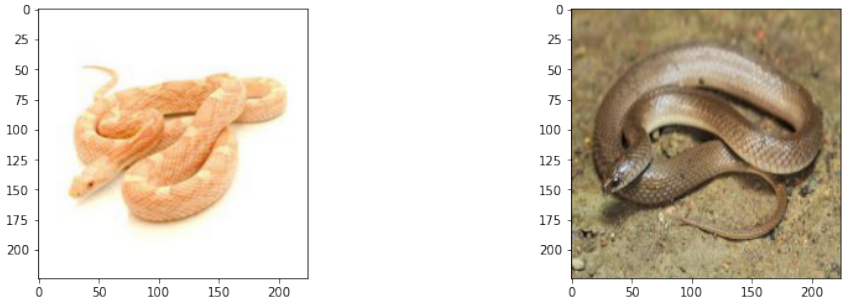

Figure 1: Sample of Training Data

4 Methods

4.1 Selected Model

To tackle this image classification problem, I utilized ResNet-18, an 18-layer architecture using CNNs and shortcut connections.

4.1.1 Model Architecture

The ResNet-18 model features 8 building blocks, each of which passes an input through two convolutional layers with an equivalent number of channels. Figure 2, from the original ResNet research paper, illustrates the structure of a building block [5]. The input to each block, **x**, is passed through two convolutional layers to produce the residual mapping **F(x)**, which is then added to the unmodified input **x** through a shortcut connection. Batch normalization is applied after each convolutional layer, and the ReLU activation function is applied between the two convolutional layers and after **x** is added to **F(x)**.

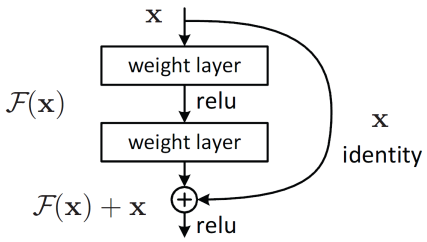

Figure 2: Building Block [5]

Starting with images of size 224 x 224 x 3, Figure 3 illustrates the structure of ResNet-18, showing how an input image passes through 17 convolutional layers and one fully connected layer. In the diagram, each convolutional layer gives the filter dimensions in the form of height x width x channels, as well as the stride (s) and zero-padding (p). The dashed lines represent shortcut connections where there is a difference in dimensions between **x** to **F(x)**. From CONV1 to MAX POOL and CONV2 through CONV5, batch normalization and the ReLU activation function are applied between each layer.

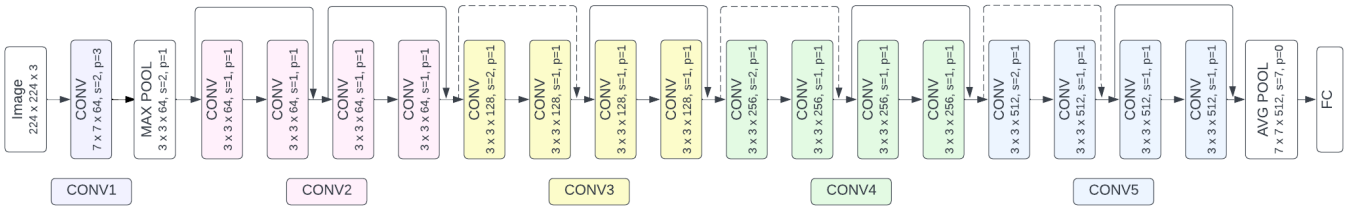

Figure 3: ResNet-18 Architecture [8]

Table 1 gives the output dimensions for each layer of the neural network, where the layer names correspond to Figure 3. The output dimensions of the convolutional layers given in Table 1 come from the original ResNet research paper [5], and can be calculated using the formulas below:

$$H_o = \frac{H_i - k + 2 * p}{s} + 1 \tag{1}$$

$$W_o = \frac{W_i - k + 2 * p}{s} + 1 \tag{2}$$

In equations 1 and 2, *H* refers to height, *W* refers to width, subscript *i* denotes an input dimension, subscript *o* denotes an output dimension, *k* is the kernel size, *p* is the amount of zero-padding, and *s* is the stride.

Layer	Output Dimensions
CONV1	112 x 112 x 64
MAX POOL	56 x 56 x 64
CONV2	56 x 56 x 64
CONV3	28 x 28 x 128
CONV4	14 x 14 x 256
CONV5	7 x 7 x 512
AVG POOL	1 x 1 x 512
FC	35 x 1

Table 1: ResNet-18 Layer Output Dimensions

4.1.2 Convolutions

Obtaining the output of a convolution operation on an input using a filter requires knowing the input matrix, kernel, stride, and zero-padding. For a kernel of size k_h x k_w, a 1 x 1 section of the output can be obtained by performing element-wise multiplication on the values in a k_h x k_w section of the input matrix and the kernel, then summing the products. The kernel is then moved over the image from left to right, top to bottom such that the number of pixels the kernel is moved is equal to the stride. If there is zero-padding present such that $p > 0$, p rows and columns are added as a border to the input image matrix.

For a 3 x 3 input image with a 3 x 3 kernel, a stride of one, and one row and column of zero-padding on each side, Figure 4 below shows how the first row of the output of a convolution is calculated.

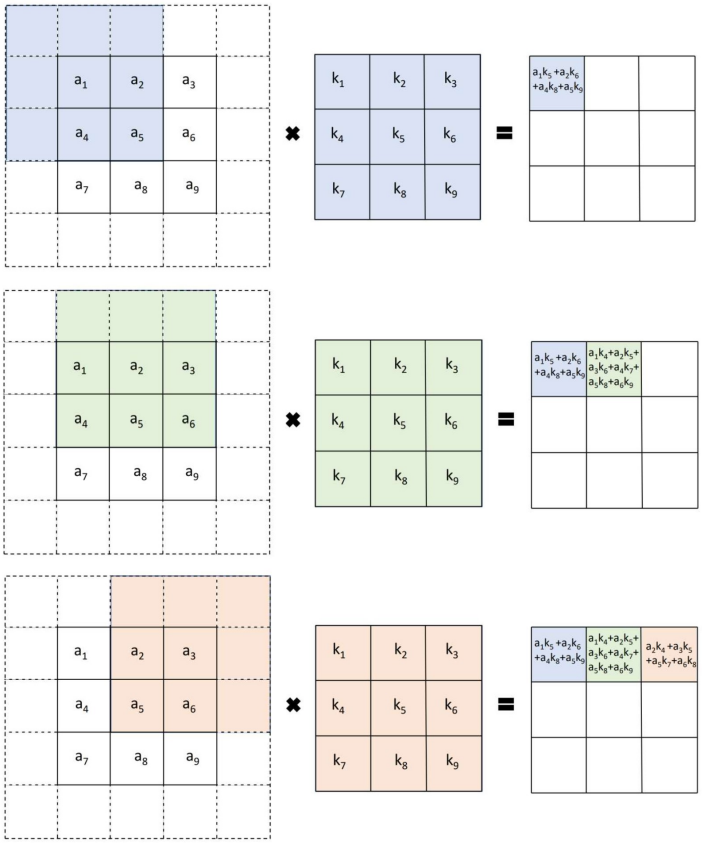

Figure 4: Convolution Operation

4.1.3 Model Advantages

The use of shortcut connections in the ResNet structure helps alleviate the vanishing gradient problem present in other CNN architectures. In a neural network, the weights used to generate the predicted output are updated during training following the gradient descent formula:

$$W_{t+1} = W_t - \eta * \nabla L(W_t), \tag{3}$$

where W_{t+1} are the updated weights, W_t are the current weights, η is the learning rate, and $\nabla L(W_t)$ is the gradient of the loss function. Since the gradient of the loss function is used to calculate the new weights in the model, without shortcut connections, the weights stop updating when the partial derivative of a layer's output with respect to the layer's input ($\frac{\partial F(x)}{\partial x}$) gets too small.

However, when there are shortcut connections, this problem is avoided, as the partial derivative of a layer's output taken with respect to the layer's input becomes $\frac{\partial H(x)}{\partial x} = \frac{\partial (F(x)+x)}{\partial x}$, where $H(x) = F(x) + x$. This can be simplified to $\frac{\partial (F(x)+x)}{\partial x} = \frac{\partial F(x)}{\partial x} + \frac{\partial x}{\partial x} = \frac{\partial F(x)}{\partial x} + 1$. The $\frac{\partial x}{\partial x} = 1$ term increases the partial derivative and prevents the gradient from disappearing.

4.2 Loss Function

When training the ResNet models, I utilized the cross entropy loss function, given in equation 4, to compare predicted categories to actual categories.

$$\mathcal{L}_i = -\sum_{j=1}^{n} y_{ij} * \log(p_{ij}) \tag{4}$$

In this equation, y_i is a binary vector where each row *j* represents an image category. The vector contains one entry of "1" for the true label of image *i* and *n*-1 entries of "0" for the other categories. The log(p_{ij}) component represents the logarithm of the predicted output scores for the image *i*, where the category (row) with the highest score is the predicted category. The cross-entropy loss function takes the sum of the products of the actual category vector and the log of the predicted category vector for all *n* categories, multiplied by negative one. Loss over all N images is averaged as in equation 5 below, and the goal of the model is to minimize the average loss by using backpropagation to update the weights of the model.

$$\mathcal{L} = \frac{1}{N} \sum_i \mathcal{L}_i(f(x_i, W), y_i) \tag{5}$$

4.3 Optimizers

When constructing my ResNet model, I decided to train and test two versions of the model, one using SGD with momentum and one using the Adam optimizer. I chose these two optimizers because both have been discussed during class and used on homework assignments.

Gradient descent is a model optimization approach that updates a model's weights by subtracting the product of a learning rate and the gradient of the loss function from the current weights. Stochastic gradient descent (SGD) refers to performing this calculation, given in equation 3, on mini-batches of training data rather than the entire training dataset, which increases the efficiency of the weight-update process.

As shown by the subscript *t* in equation 3, SGD updates the model weights using the gradient from one particular point in time [6]. Ignoring past gradients can cause the model to undergo more iterations than necessary before reaching convergence [6]. This issue can be solved with the inclusion of momentum, denoted by γ, in the weight update calculation [6]. Momentum can be set to any value between 0 and 1, and, following the authors mentioned in Section 2.2 [7], I chose to use γ = 0.9 for my model. The value of momentum controls how much the gradient from the previous step, also known as velocity (ΔW), impacts the updated weights. Equation 6 shows how momentum

fits into the updated model weight calculation and results in fewer iterations being needed to reach convergence by weighting past gradients.

$$W_{t+1} = W_t - (\eta \nabla L(W) + \gamma \Delta W), \tag{6}$$

For my second model, I chose to use the Adam optimizer, which is similar to SGD with momentum in that it uses an average of the gradient of the loss function rather than the gradient at just one time period. Adam uses first and second moments estimates, the latter of which relies on the square of the gradient, to scale the learning rate when updating the model weights. Generally, Adam is known to perform well in image classification problems, but, as noted in the *Cornell University Computational Optimization Open Textbook*, there are some instances where Adam does not find the optimal solution [1]. Section 5 below explores whether that was the case for this snake classification problem.

4.4 Training and Testing Process

For both of my models, I split the images in the training set into batches of size 50 and then trained the models sequentially, first the Adam model and then the SGD with momentum model, over 10 epochs. While each model was training, I recorded the average loss per epoch to use for comparison. After the models were done training, I used images in the testing set, also with a batch size of 50, to test the accuracy of the model. The batch size of 50 was selected using Tanay Shah's batch size of 54, as mentioned in Section 2.1, as a guide [9], and 10 epochs were run during training due to time constraints limiting the number of epochs used.

4.5 Code

To write the code for my ResNet-18 model, I referenced several sources. First, to load the data into Jupyter Notebook, I adapted the Pytorch DataLoader lines of code used in almost every STAT 4984 weekly code example following the steps outlined in "Loading Image Data into PyTorch" [14]. I then referenced a Stack Exchange post to learn how to visualize images loaded through the DataLoader class [13]. For the model code itself, I made changes to the code in ResNet_stud.ipnyb, written by Youhui Ye and available on Canvas, [16] in order to fit the image size and model structure outlined in the sections above. Lastly, to make the confusion matrices shown in Section 5, I adapted the code from STAT4984_week07_code.ipynb, also available on Canvas [15].

5 Results

As mentioned in Section 4.4, I recorded the average loss per epoch during training for both models, and Figure 5 below illustrates those values.

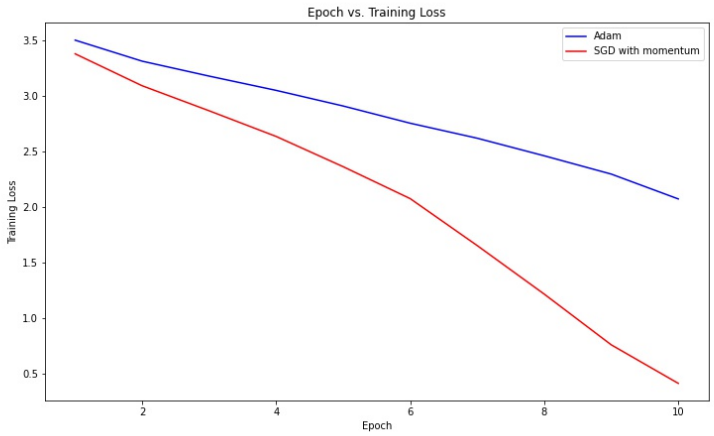

Figure 5: Average Training Loss Per Epoch

As can be seen in Figure 5, the model using SGD with momentum produced lower loss over all 10 epochs compared to the model with the Adam optimizer, and the difference in the two models' losses increased each epoch. From the loss values alone, the SGD with momentum optimizer appears to be preferable to the Adam optimizer for this problem.

After training, both models were fed images in the testing dataset and each model's accuracy of prediction was calculated as the number of correct predictions divided by the number of total predictions. After working through all of the testing images, the model using SGD with momentum produced an accuracy of about 14.9%, while the model using the Adam optimizer produced an accuracy of about 6.8%. The testing results are displayed in the confusion matrices below, where the rows represent the true labels and the columns are the predicted labels. As can be seen in Figure 6, the diagonal, representing correct predictions, is clearly defined by green to yellow coloring for the SGD with momentum model, indicating a relatively high proportion of correct predictions made by the SGD with momentum model. This model reached about 17.5% accuracy in predicting the Coluber constrictor, Crotalus horridus, and Pantherophis alleghaniensis species. The Adam model, on the other hand, lacks a similar strength in prediction, as seen in Figure 7. The proportions of correct predictions made by the Adam model are lower than the SGD with momentum model, with the largest proportion of correct predictions being between 12.5% and 15% for the Coluber constrictor and Pantherophis guttatus species.

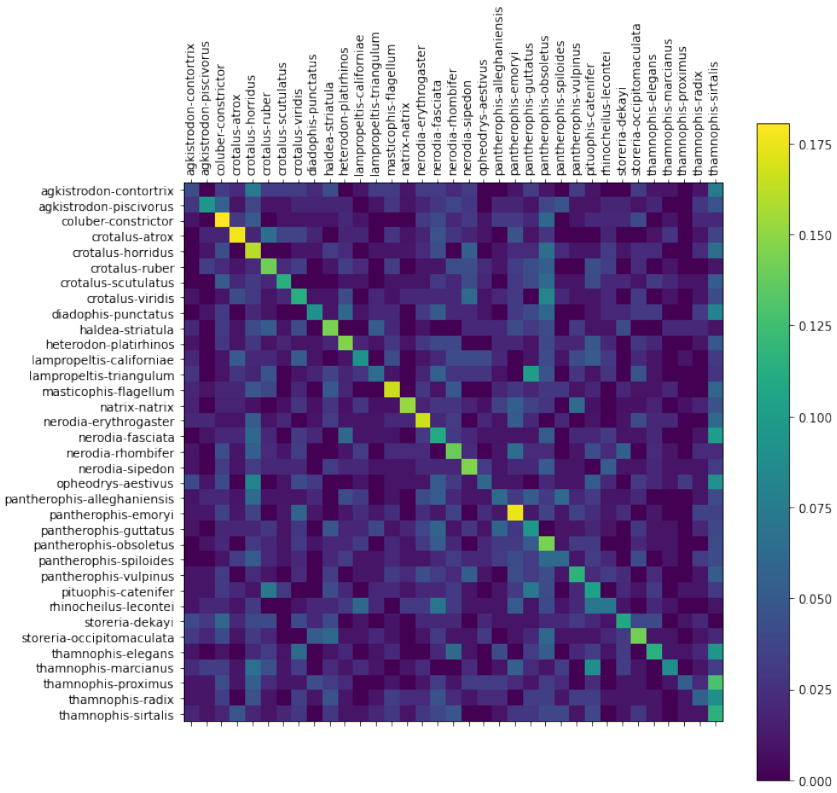

Figure 6: SGD with Momentum Model: Testing Data Confusion Matrix

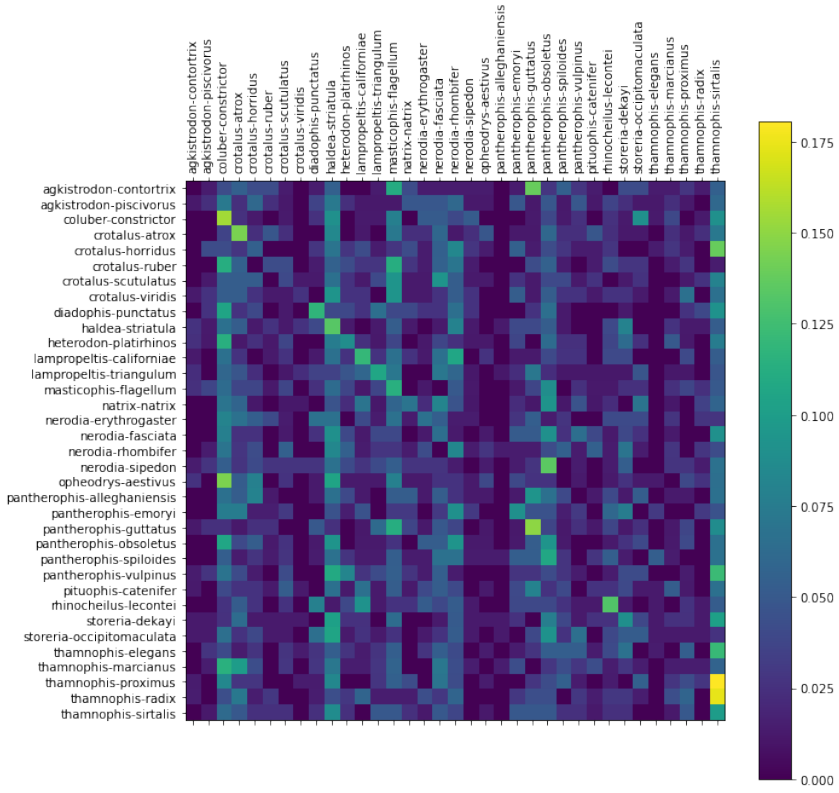

Figure 7: Adam Model: Testing Data Confusion Matrix

Another noticeable part of both confusion matrices is the last column, representing predictions in the Thamnophis sirtalis species. The last five rows in each matrix represent the actual labels of Thamnophis elegans, Thamnophis marcianus, Thamnophis proximus, Thamnophis radix, and Thamnophis sirtalis. That there appears to be a high number of mispredictions for the first four of these classes into the Thamnophis sirtalis class likely stems from the fact that all of these snakes are from the same genus (Thamnophis, commonly referred to as garter snakes), and thus share similarities in colorings. However, there are multiple other snake species of the same genera in the dataset, and none produce such a noticeable level of misprediction in both models as the Thamnophis genus does. For future work, perhaps a deeper model or more training epochs could increase the accuracy of predictions for the five snakes of the Thamnophis genus.

6 Conclusion and Discussion

From both the lower training loss and higher testing accuracy results, the SGD with momentum optimizer is better suited for this snake classification problem than the Adam optimizer, with the performance difference between both models growing over epochs. Figure 5 shows that the Adam optimizer loss failed to converge towards zero in the way that the SGD with momentum optimizer did, confirming the possibility of the Adam optimizer not converging that was mentioned by the authors of *Cornell University Computational Optimization Open Textbook* in Section 4.3 [1].

Some of the limitations I faced with this problem were limited time and computational resources, but with a more powerful computer, these models could be trained over a larger number of epochs to observe changes in performance. Given current results, I would expect the performance of the SGD with momentum model to increase with more training epochs, although I would not expect the Adam model to fare much better. A deeper model could also be considered for this problem, where hopefully the increased number of parameters could help differentiate the appearance of snake species. Overall, although the accuracy was not exceptionally high for either model, running both

models revealed that SGD with momentum is a better choice for this data than the Adam optimizer, which provides a solid foundation for future work.

References

[1] A. Ajagekar. "Adam," Cornell University. (Dec. 2021), [Online]. Available: https://optimization.cbe.cornell.edu/index.php?title=Adam.

[2] D. Dutta. "Identifying different breeds of snakes," Kaggle. (n.d.), [Online]. Available: https://www.kaggle.com/datasets/duttadebadri/identifying-different-breeds-of-snakes?resource=download. (accessed: 04.17.2023).

[3] J. Filter. "Split-folders 0.5.1," PyPI. (2022), [Online]. Available: https://pypi.org/project/split-folders/. (accessed: 04.17.2023).

[4] "Hackerearth deep learning challenge: Snakes in the hood," HackerEarth. (2020), [Online]. Available: https://www.hackerearth.com/challenges/new/competitive/hackerearth-deep-learning-challenge-snake-breed-detection/.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Dec. 2015. [Online]. Available: https://arxiv.org/pdf/1512.03385.pdf, (accessed: 04.27.2023).

[6] T. Lee, G. Gasswint, and E. Henning. "Momentum," Cornell University. (Dec. 2021), [Online]. Available: https://optimization.cbe.cornell.edu/index.php?title=Momentum.

[7] M. Rajabizadeh and M. Rezghi, "A comparative study on image-based snake identification using machine learning," vol. 11, no. 19142 (2021), Sep. 2021. [Online]. Available: https://www.nature.com/articles/s41598-021-96031-1.

[8] M. Schaeb. "Resnet-18 architecture," Lucidchart. (Apr. 2023), [Online]. Available: https://lucid.app/lucidchart/fdc61699-2900-4a26-95fa-9b35e21b745d/edit?viewport_loc=-16%5C%2C-268%5C%2C2283%5C%2C1200%5C%2C0_0&invitationId=inv_ef141311-5910-4a31-ac7f-d51415bc9b7e.

[9] T. Shah. "Resnet101 efficientnet," Kaggle. (2021), [Online]. Available: https://www.kaggle.com/code/tanay27/resnet101-efficientnet/notebook. (accessed: 04.28.2023).

[10] "Snake bites," Johns Hopkins Medicine. (n.d.), [Online]. Available: https://www.hopkinsmedicine.org/health/conditions-and-diseases/snake-bites. (accessed: 04.28.2023).

[11] "Snakes," Internet Center for Wildlife Damage Management. (n.d.), [Online]. Available: https://icwdm.org/species/reptiles/snakes. (accessed: 04.28.2023).

[12] "Snakes," National Geographic. (n.d.), [Online]. Available: https://www.nationalgeographic.com/animals/reptiles/facts/snakes-1#:~:text=There%5C%20are%5C%20more%5C%20than%5C%203%5C%2C000,or%5C%20significantly%5C%20wound%5C%20a%5C%20human.. (accessed: 04.28.2023).

[13] "Using dataloader to display an image," Data Science. (), [Online]. Available: https://datascience.stackexchange.com/questions/112918/using-dataloader-to-display-an-image.

[14] R. Wingate. "Loading image data into pytorch." (May 2020), [Online]. Available: https://ryanwingate.com/intro-to-machine-learning/deep-learning-with-pytorch/loading-image-data-into-pytorch/.

[15] X. Xing. "Stat4984_week07_code.ipynb," Canvas. (2023).

[16] Y. Ye. "Resnet_stud.ipynb," Canvas. (2023).