Snaking Through a Challenge: Snake Species Identification

Andrey Pak
Georgia Institute of Technology
andrey.pak@gatech.edu

Hyunsu Park
Georgia Institute of Technology
hpark379@gatech.edu

Joshua Reno
Georgia Institute of Technology
jreno@gatech.edu

Abstract

Snakebite is an extremely deadly injury that causes nearly hundreds of thousands of deaths and disabilities every year. Snake species are extremely diverse and dozens of new species are discovered every year. For the final project, we decided to participate in the AIcrowd Snake Species Identification Challenge [1] *with an objective to train a snake species image classification model given the provided dataset. First, we approached the problem by doing basic data pre-processing including the removal of the corrupted images and the images that are unlikely to be classified as snakes. In our approach, we used a pretrained model to exclude images without top-5 labels containing references to snakes or lizards as a result of extraneous elements appearing in images. Second, due to the heavily imbalanced nature of the dataset, we attempted to up-sample rare classes and down-sample well populated classes using a weighted random sampler and augmented images from rare classes to boost the population of rare species images. Due to time considerations, we decided to fine-tune pre-trained models provided in PyTorch* [2] *framework rather than training it from scratched. We have achieved reasonable results with significant room for improvement.*

1. Introduction

Snakebite is the deadliest neglected tropical disease (NTD) causing approximately 100,000 human deaths and 400,000 disability victims globally every year [3]. They largely affect poor and rural communities in developing countries, which overlap with range and habitats of venomous snake diversity and are likewise unable to cope with limited access to medicine. Antivenoms are crucial but must be administered based on correct species classifications. This is challenging due to diversity, misleading or incomplete information from victims, and medical professionals lacking herpetological expertise. There are also over 24 families, 528 genera, and 3709 species of snakes with an average of 30 new species discovered annually since 2000. Comparing the evolutionary history of snakes with that of mammals, as shown in Figure 1, snakes are extremely diverse with vastly different colors, textures, and environments.

The identification challenge gets even more complicated due to snake species having different appearances depending on age and environment intraspecies. Even snakes that appear to originate from identical species can, in fact, come from different species and differ in the harm caused by their venom, as shown in Figure 2.

While neural network based image classification is a hot topic, there are very few published works on this particular problem of image-based classification of species within a single taxonomic suborder. In [4], authors used carefully selected dataset for classifying fish, plant, and butterfly species from Europe and South America. Another work on taxonomic classification [5] used DNA sequences rather than images to perform this task. We were able to find an app [6] that is focused on snake identification, but it is unclear how exactly it operates.

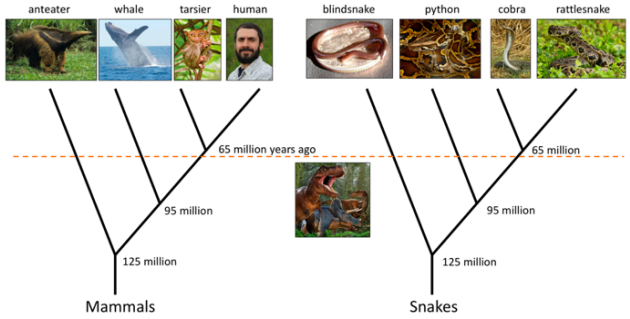

Figure 1. Evolutionary history of mammals and snakes

2. Approach

Our goal is to make a classification model that is robust to a dataset consisting of a significant number of bad quality images and a heavy class imbalance. To solve these problems, we propose three methods: 1) filtering the whole dataset to exclude bad quality images before training, 2) up-sampling infrequent classes and down-sampling frequent classes in each batch, and 3) augmenting images belonging to infrequent classes.

2.1. Dataset overview

The dataset for the third and final round of the challenge contains 129,773 images spread across 85 classes. The detailed class distribution is shown in Figure 4. As can be seen from the figure, the dataset is heavily imbalanced with largest class heavily dominating others. Images vary heavily in quality, spatial resolution, gamma and exposure Additionally, approximately 180 images were found to be corrupt and were neglected. As a rule of thumb, an 80/10/10 train/validation/test split was used in the training process.

Figure 2. Identical species across age groups (top left). Species in different environments (top right). Venomous species and similar harmless species (bottom).

Figure 3. Different classes of snakes held up by human hands.

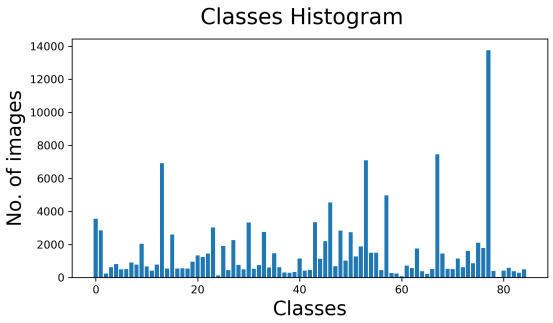

Figure 4. Dataset statistics. Total of 85 classes, 129,773 images, maximum of 13753 and minimum of 30 images per class, $\mu \approx 1526$, $\mu_{1/2} = 762$, $\sigma \approx 2007$.

A significant number of images contain textures or environments that are likely irrelevant to snake identification. Additionally, some of them also included "snake-unrelated" features that could also interfere with the classification such as a human hand, as shown in Figure 3, or copyright watermarks. The number of images with people holding snakes might create bias towards the presence of human hands in the images. Moreover, images with low spatial resolution combined with motion blur make it difficult to show the presence of a snake on the image even for a human observer. Finally, snakes' camouflage makes it difficult to detect a snake in its natural habitat.

2.2. Dataset pre-processing

Dataset is provided as a single folder with all images as well as .csv file with all the labels. First, all images are sorted in the class subfolders in order to follow the required input for the `PyTorch ImageFolder` dataset class. When loading data all images are resized to 224x224x3 pixels and normalized using pre-calculated mean and standard deviation of the dataset.

To filter out some of the bad quality images discussed in the previous subsection, we preprocess the dataset before training. First, we use the pretrained network [7] to predict top-5 ImageNet labels of images in the dataset. If the top 5 ImageNet labels of an image do not contain snake or lizard-like labels, we exclude the image from the dataset. We chose to include images with lizard-like labels because snakes and lizards are biologically and evolutionally closely related.

With filtering, number of images in the training dataset decreases from 103818 to 96454. This is not a significant decrease in the number of images in the dataset. So, we can expect to have a better-quality data without losing too much samples.

2.3. Up-Sampling Infrequent Classes and Down-Sampling Frequent Classes

The largest class in the dataset contains 13753 samples while the smallest only contains 30. The big difference of number of samples in classes lead to the class imbalance, which can significantly degrade our model's performance on infrequent classes. Since the performance of our model on infrequent classes is equally important to the performance of our model on frequent classes, we decide to mitigate the class imbalance.

To solve the issue with the class imbalance, we use the weighted random sampler from `PyTorch`. By pre-computing weights of training images based on their classes and passing the weights to the sampler, we can expect each batch of our data loader to have uniform distribution of classes. It follows that, with enough epochs, our model will perform equally well on every classes.

2.4. Augmentation based on class frequency

With weighted random sampler discussed in the previous section, we encounter another problem: since the infrequent classes will appear the same number of times with the frequent classes in a batch, our model will be trained on duplicate images of infrequent classes, sampling with replacement is used by default as well as is necessary due to class imbalance. Repeatedly trained on the same images for infrequent classes, our model will highly overfit to the training data. This is later observed in the results section.

Augmenting images for infrequent classes can solve this problem. Note that we do not need to augment frequent classes because we have enough different images for the classes. To simulate equal number of samples among classes, we set

$$P(transform\ of\ an\ image\ in\ C) = 1 - \frac{|C|}{\max_i |C_i|} \quad (1)$$

For example, assume we have 30 samples in *C* and 300 samples in the most frequent class. According to the equation above, probability of augmentation of an image in class *C* should be 0.9. With the probability, we can generate 270 augmented images and simulate 300 samples for class *C*.

While the methods discussed in 2.3 and 2.4 can solve the class imbalance problem, it requires more epochs for the model to see all the samples in the dataset.

2.5. Loss function

In order to further accommodate class imbalance, class weights are further used in the cross-entropy loss function:

$$L(x, c) = w_c \left[-x_c + \log\left(\sum_j e^{x_j} \right) \right], \quad (2)$$

where c is the class index and w_c is the weight corresponding to the class frequency shown in Figure 4 The losses are averaged across observations for each minibatch.

2.6. Fine-tuning

Due to the project time constraints, we decided to fine-tune the ImageNet pre-trained models provided in `torchvision.models` subpackage of `PyTorch`. This would have probably resulted in less-than-desired accuracy due to the fact, that pretrained models will likely have learned features that are irrelevant to the snake classification problem, but we couldn't afford training from scratch for hundreds of epochs like it is usually done to achieve state-of-the-art results.

In fine-tuning, the final fully connected layer is replaced by another layer that matches the output of the previous fully connected layer of the model to the desired number of classes, so for this problem, the final layer provided 85 outputs for each snake class. While the rest of the model is fine-tuned, this last layer is learned from scratch.

3. Experiments and Results

We have started with the scaffolding code from Homework 1, because it was exactly the same type of problem, but with a different type of dataset. We decided to use Adam as our optimizer and weighted cross entropy as our loss function.

We used one machine equipped with NVIDIA GeForce RTX2080 and another one with GTX1070 to fine-tune the models. Second GTX1070 was obtained closer to the end of the project, but we encountered several hardware and software challenges when trying to parallelize the execution across two GPUs: in Ubuntu 18.04 wrapping the pretrained `torchvision` model in `PyTorch DataParallel` resulted in system hanging with low GPU resources consumption and any attempt to stop the script resulted in an unrecoverable crash only fixed by hard reset, while tutorial examples that wrapped a dummy model in `DataParallel` worked without any problem. This appeared to be a known issue, with no clearly identifiable solution. On Windows machine, parallelizing was

successful, but `PyTorch` was compiled without NVIDIA Collective Communications Library (NCCL) support that probably possibly resulted in sub-optimal performance. We were able to train in batches twice larger, but it took longer time. We did not have time to confirm whether it was positive and negative scaling performance-wise.

3.1. Initial experiments

Initially, we have chosen to fine-tune some pretrained models. First model of choice was Squeezenet [8] due to its size, and, consequently, the larger batch size (128) to train. The baseline results for Squeezenet with no accounting for imbalance in the dataset were obtained with learning rate = 1e-3, learning rate step = 10, learning rate gamma = 0.1, epochs = 20, weight decay of 1e-3. The results are shown in Figure 5. One can notice the effect of the L2 regularization – while it prevented the overfitting, it resulted in significantly less accuracy on both test and validation datasets (33% vs. 44%).

Resnet18 was the next model chosen for fine-tuning. The baseline results were obtained using the following parameters: batch size = 10, learning rate = 1e-5, learning rate step = 10, learning rate gamma = 0.1, and epochs = 20. The result of testing dataset accuracy of 53% was achieved. The results are shown in Figure 6.

The third model was batch normalized VGG19 [9]. On a single GPU we were limited to the batch size of 10-14, which resulted in a very noisy learning curve. VGG19bn showed faster training reaching ~60% accuracy on the test dataset after 10 epochs. Increased number of epochs did not show an improve in accuracy. The baseline results for VGG19bn were obtained with batch size = 10, learning rate = 1e-5, learning rate step = 10, learning rate gamma = 0.1, and epochs =10. The results are shown in Figure 7.

The main hyperparameters that affected the training process were the learning rate and the number of epochs. For example, Squeezenet was trained with learning rate of 1e-3 while for VGG19bn we had to bring it down to 1e-4 or 1e-5. We also maxed out the batch size for all models according to available GPU memory.

Based on these results we decided to proceed further with VGG19bn.

3.2. Filtering out low quality images

To see the results of the filter option, we trained a Squeezenet with learning rate = 1e-3, learning rate step = 10, learning rate gamma = 0.1, and epochs =20 on the filtered dataset containing 93% of the original dataset.

With the filter option excluding bad quality images based on ImageNet label predictions, we were able to train our model with a comparable performance and shorter training time as shown in Table 1.

Table 1. Image Filtering Performance

	Test accuracy	**Training time**
filter = True	~40%	~ 11 hours
filter = False	~40%	~ 12 hours

The result suggests that we may be able to achieve better performance within a shorter time if we use a more careful filtering criterion.

3.3. Weighted sampling and Loss Functions

The introduction of sampling based on the frequency of classes did not affect much (+/- 1%) the final accuracy on the testing dataset, however, different behavior of the learning curves can be observed on Figure 8. Comparing to Figure 7, the curves are less noisy because this training instance was done on two GPUs with the larger batch size of 30. We can also see significantly less overfitting likely due to weighted approach.

3.4. Discussion and Future Work

Weight regularization couldn't address the significant overfit on the baseline models without loss of accuracy. For some training instances of weighted VGG19bn we also tried introducing weight regularization, but it did not qualitatively affect the overfit. One of the possible reasons that might have contributed to the imbalanced dataset, besides the proposed weighting approach, is the fact that the `PyTorch random_split` function that we used to split the dataset into train, test and validation subsets does not account for class frequencies. We believe implementing stratified split can possibly improve the results this problem.

Our implementation of augmentation based on class frequency was not successful and the model failed to train. We tried to augment images in the training loop but the model was not able to run with the augmentation. This is probably because the augmentation in the training loop requires a different implementation from the augmentation outside the training loop.

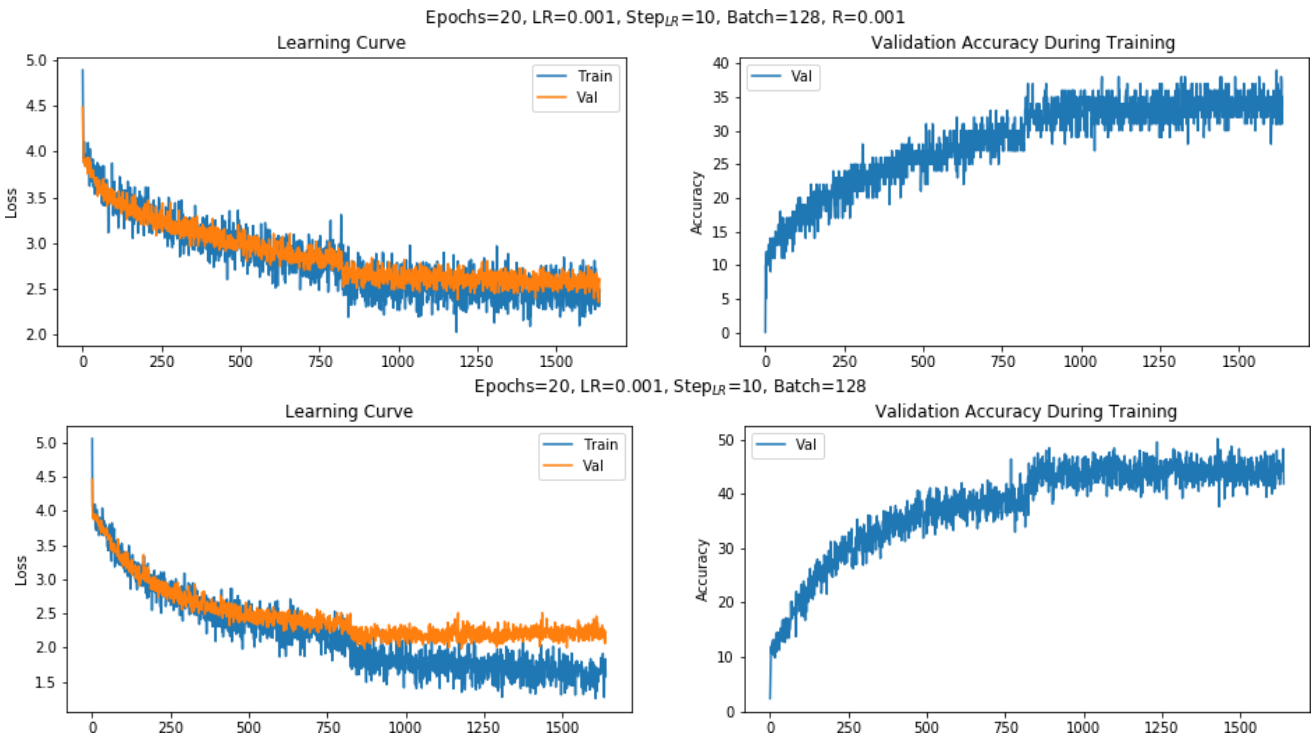

Figure 5. Baseline Squeezenet training results for 20 epochs. Bottom graphs show the results for regularized version. Accuracy on test dataset were 44%/33% for non-regularized version and regularized version accordingly.

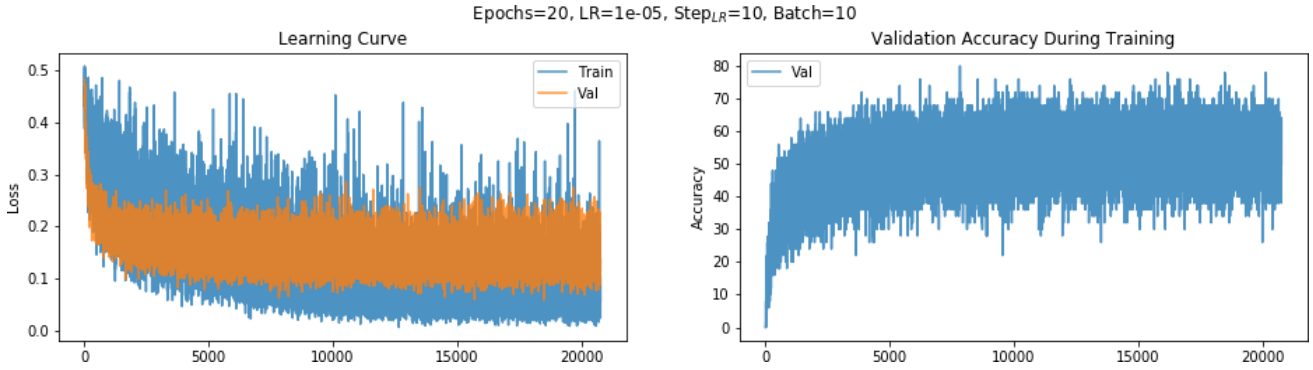

Figure 6. Baseline Resnet18 training results for 10 epochs. Final accuracy of 53% on test set was achieved. Note the step after 10 epochs when the learning rate is adjusted.

To summarize, we have achieved reasonable results with significant room for improvement. The best accuracy of 61% using VGG19bn with weighted sampling and loss function was achieved in the task of classifying 85 snake species given heavily imbalanced dataset. The challenges included handling the imbalanced data, filtering relevant images, and dealing with hardware and software issues. Some final results were not included because we experienced CUDA-related software crashes for our final training runs.

The possible future work and/or improvements are as follows:

- Manual augmentation outside `PyTorch` augmentation paradigm. In this case we will need less epoch to train, however the training dataset will be larger. We can also cut the size of the training dataset but subsampling dominant classes and possibly staying within the approximate size of the original dataset.
- Training the network from scratch can possibly yield better results as the network will learn only problem-related features.

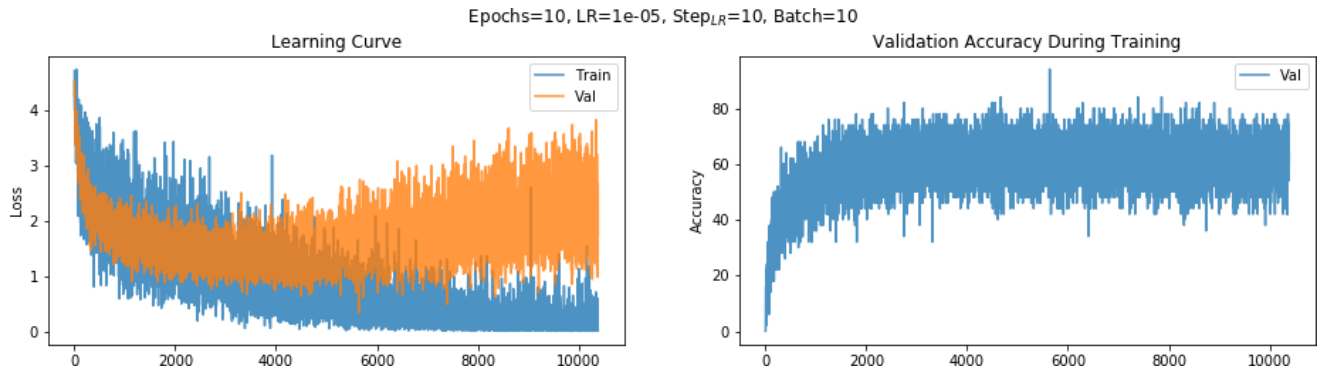

Figure 6. VGG19bn baseline results for 10 epochs, batch 10. Noisy curves due to the small batch size. Testing accuracy of 61% after 10 epcohs.

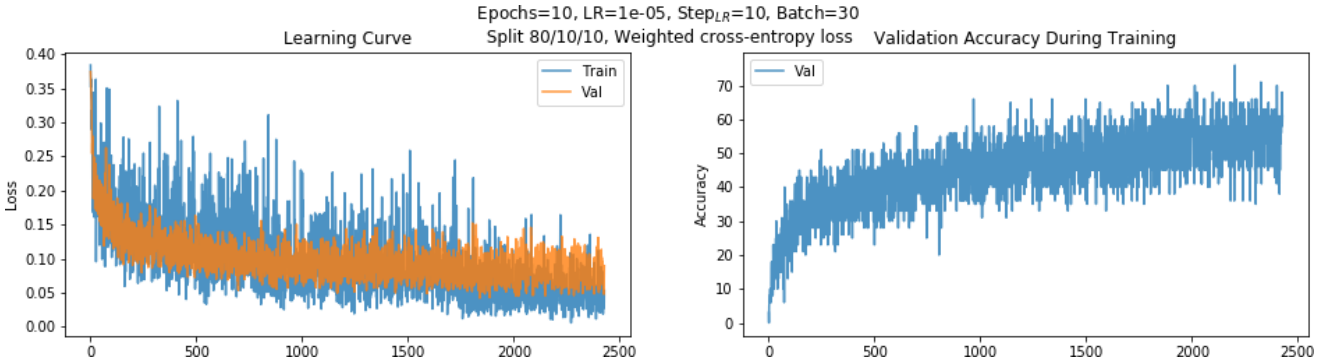

Figure 6. VGG19bn results for 10 epochs, batch 30, weighted sampling and loss. Note the reduced noisiness of the curve with the increased batch size. Final accuracy of ~60% on test set.

- Exploration of other pre-trained models provided by PyTorch.
- Implementation of stratified sampling for the train, validation and test splits in order to ensure that all classes are represented properly in the subsets.
- Submission for the challenge on AIcrowd.

4. References

[1] Institute of Global Health, "Snake Species Identification Challenge," 31 11 2019. [Online]. Available: https://www.aicrowd.com/challenges/snake-species-identification-challenge.

[2] "PyTorch," [Online]. Available: https://pytorch.org/.

[3] World Health Organization, "Animal bites," [Online]. Available: https://www.who.int/en/news-room/fact-sheets/detail/animal-bites.

[4] A. a. L. F. J.-S. Hernández-Serna, "Automatic identification of species with neural networks," *PeerJ 2,* p. 563, 2014.

[5] S. e. a. Khawaldeh, "Taxonomic Classification for Living Organisms Using Convolutional Neural Networks," *Genes 8.11 ,* 2017.

[6] "SnakeSnap," [Online]. Available: https://www.snakesnap.co/.

[7] K. e. a. He, "Deep residual learning for image recognition.," *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 770-778, 2016.

[8] F. N. e. a. Iandola, "AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size," *arXiv preprint arXiv:1602.07360.,* 2016.

[9] Z. A. Simonyan K., "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556.,* 2014.