Endangered Animal Classification

Sammy Rodriguez, Yuto Mori, Marco Robles, Randolf Uy

Table of Contents

GitHub: https://github.com/randolfuy01/deep-learning

Introduction

The topic of our project will be a classification project utilizing Convolution Neural Network for our deep learning model. We came about this topic by first choosing to work with images and decided a classification problem would be the most interesting. The problem we are tackling is on endangered animal classification, with the goal of developing a model capable of identifying endangered animals in an image. We also plan to implement the model into a script that will classify what animal is shown. This classification model could potentially assist wildlife organizations in their conservation efforts by providing an automated tool for distinguishing species that are endangered and at risk of extinction. We want to create something that would make a positive impact to protect the real world problem of these endangered animals.

Set up

The first step in setting up our project was creating a repository, for collaboration on the project. We created a github repository and once it was set up, we chose our dataset. We acquired a publicly available dataset from Kaggle.com containing labeled images of endangered and non-endangered animal species. The dataset consisted of ~6500 images, with 11 the labels of animal species. We later found out our dataset was a bit small for our task. Once the dataset was chosen, we began the preprocessing stage by resizing all images to a uniform size of 224x224 pixels. We then converted the images to tensors and normalized them with a mean and standard deviation. After the preprocessing stage, our next step was to create the data loaders for training, validation and testing sets. For our environment, our model was implemented using PyTorch, leveraging its deep learning capabilities for designing and training CNNs. The training was performed on GTX 1070.

Architecture

In an effort to standardise everything to easily modify the architecture during training and testing we created a python script. This included different functions that would serve as the catalyst for our models and thus would require parameterization for editing things.

```
def get_dataloader(transform, root, batch_size, split_ratio=0.8, num_workers=2)
```

This data-loader function was created with the primary responsibility of ensuring standardized quality of the data and optimizing the time it would take the images to be loaded for training. This allows us to encapsulate most (if not all) the preprocessing within the function rather than worrying about it during the training loop or in later parts of the code.

For the baseline model, not the initial model, the architecture consisted of 3 convolutional layers and 3 linear layers. This allowed us to create a starting point from which we could either build upon or remove from the model. Each convolutional layer is designed to extract spatial features from input by applying filters. A layer would be represented by this structure:

```
nn.Conv2d(3, 32, kernel_size=3, padding=1)
```

In this case, 3 would be the input dimension, 64 is the output, the kernel size is 3, and the padding is 1. The kernel size is the receptive field and essentially determines the balance between local and global information. A larger kernel size (larger receptive field) means that we capture more global information while a smaller kernel size (smaller receptive field) means that we want to capture more local information. We decided to go with a small padding of 1 and increase it if necessary in the event that we would need to in preserving the spatial context of the

data. Each layer is followed by the ReLU activation and a maxpool layer where the ReLU is used to introduce non-linearity to learn complex patterns and the max pooling allows us to downsample the feature maps reducing the dimensions and computational load without sacrificing important information.

Mathematically, we can express each convolutional layer as an equation where:

$$y(i_1, i_2) = b + \sum_{j_1, j_2 = 0}^{r} K(j_1, j_2) u(i_1 - j_1, i_2 - j_2)$$

The r defines the size of the convolution kernel. Where we define a convolutional layer as a tuple of parameters. K implicitly defines the number of input channels c_{k-1} and the number of output channels c_k. This means that each kernel has a shape of (r, r, c_{k-1}). Where each kernel is applied across all input channels to produce one feature map. The c_k filters result in c_k output feature maps (Convolutional Neural Networks as 2-D Systems). This is essentially how feature extraction works where the larger the kernel, the more area it covers therefore it can account for more global features. Smaller kernels account for less area therefore account for more local features.

After which, is followed by our linear layers which are used to perform the high-level reasoning based on the features extracted by the convolutional layers. Which propagates until we get to the output layer which is 11 output units representing the 11 classes of endangered species.

```
nn.Linear(256 * 14 * 14, 512)
```

We are able to propagate until the output layer as we can express the linear layer as an expression of $y = Wx + b$ where:
$x \in \mathbb{R}^n$: represents the input vector of size n.
$W \in \mathbb{R}^{m*n}$: represents the weight matrix, where m is the number of output features (neurons) and n is the number of input features.

$b \in \mathbb{R}^m$: represents the bias vector
$y \in \mathbb{R}^m$: represents the output vector of size m (number of neurons in the layer)
Source: A Mathematical Introduction to Neural Networks

Through this we can derive and extract the key features from the convolution layer and have 10 output neurons which represent the 10 different classifications we have.

```
def training_loop(model, trainloader, validloader, num_epochs=20, learning_rate=0.001,
device="cpu")
```

The training loop function is our implementation for training and validating the model for a specified amount of epochs. We can plug and play different models into it, different training/validating data, the learning rate, and the device we want to use (if available). Within the training function, there are different components such as the loss function, optimizer, and learning rate scheduler. The loss function determines the error between our predicted output and the ground truth labels. In our initial studies we decided to go with cross-entropy as it is most common for classification tasks with image data. The optimizer is used to update our weights based on the gradients and learning rate. And the scheduler reduces the learning rate every n number of steps to ensure stability within our training. The training loops in this manner:

- Iterate n number of times (num epochs)
- Set the model to training mode
- For each batch in the trainloader
    - Inputs and labels are moved to the device we specified
    - Gradients are reset using the optimizer.zero_grad()
    - Predictions are obtained using outputs = model(inputs)
    - The loss is calculated by using our criterion

        - The gradients are computed in the backward step and the model parameters are updated by the scheduler
    - There is a running loss that is accumulated for all batches, and the training loss is aggregated and averaged along with the current learning rate for logging.
    - The scheduler then reduces the learning rate accordingly
    - The validation step is called in order to evaluate how well our model did on the validation data.

This modular approach makes the training loop adaptable for various models and datasets, in the event we decided to go with a completely different model to plug and play.

Building upon this modularity, we have the validate function which incorporates many of the same characteristics as the training loop when it comes to parameterization. In this case, the model, dataloader, criterion, and device are all ways we can utilize different models for validating.

```
def validate(model, dataloader, criterion, device)
```

This function serves as the way we can see how the model does after each epoch, a higher validation score after each epoch indicates that we are training effectively and that the accuracy of the model is going up. The validation proceeds as follows:

- Set the model to evaluation mode for inference.
- Iterate over all the batches in the validation dataset
    - Transfer each batch into the device (cuda /mps if available)
    - Predictions created through calling model and passing the images

- We can compute the validation loss for the batch by using the criterion
- Then the predicted class for each input image is determined and the correct predictions are tallied.
- $$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Samples}} * 100$$
- We then compare with the global max accuracy, from which if it is more accurate, we save the model as it is indicating the best performing model so far.

- After all batches are processed, we print the average validation loss and accuracy for the epoch and append to our validation losses and accuracies list.

Being able to monitor the validation metrics allows us to identify how our model is performing in a non-training environment and allows us to see performance plateaus or degradation.

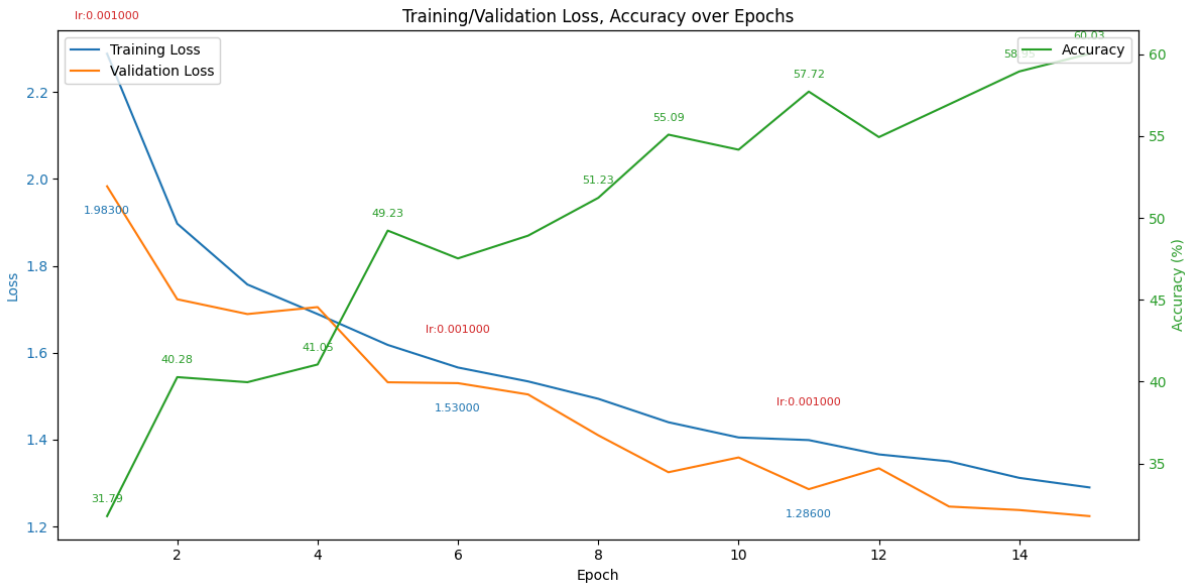
Methodology

Model 0: (Prototype)

Approach:

The number of workers was set to 4 to ensure efficient data loading and preprocessing on M3. We have set up default hyperparameters such as batch size 16, The dataset was split into 80% for training and 20% for validation, A constant learning rate of 0.001 and the epoch was set to 15 to see the initial behavior of the model. The model architecture consists of four stages of feature extraction and refinement. It begins with a convolutional layer (3 → 64 channels) to extract low-level features, followed by batch normalization for stability, ReLU activation for introducing non-linearity, and max-pooling to reduce spatial dimensions. Each subsequent stage progressively increases the number of channels, moving from 64 to 128 and then to 256, to enhance feature representation.

Result:



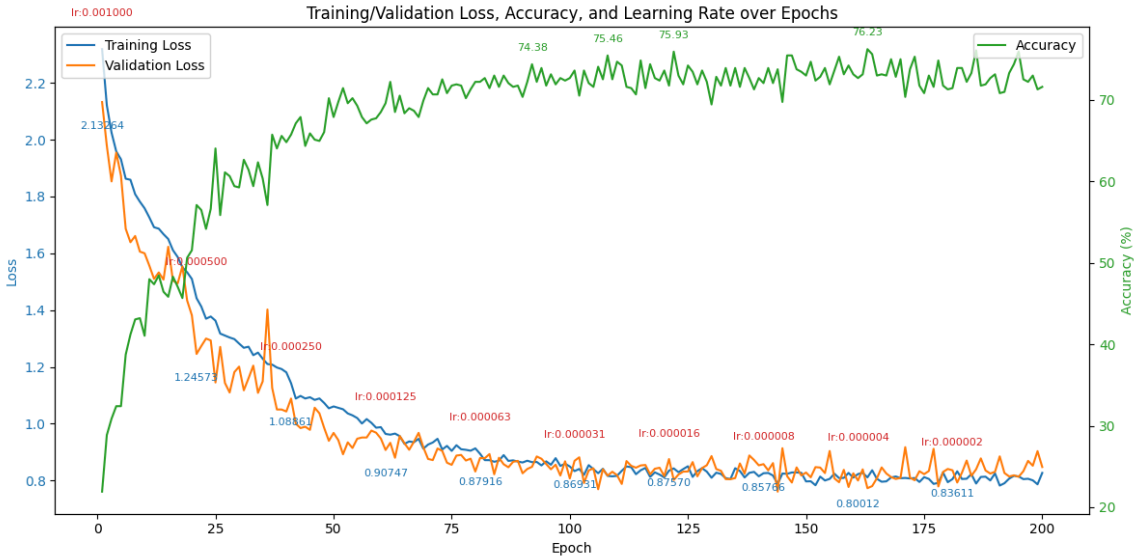
We observed that this setup has the potential to improve its accuracy with additional epochs and further optimization. However, the primary concern with this model and dataset is the risk of overfitting, where the model memorizes the dataset rather than learning the underlying patterns over time.

Model 1:

Approach:

To further analyze the behavior of Model 0, we decided to increase the number of epochs while addressing the risk of overfitting due to the small dataset. To optimize the learning curve and mitigate overfitting, we retained the original setup but introduced the Adam optimizer along with a scheduler and lowered batch size to 16. This approach gradually reduces the learning rate over time by halving it every 20 epochs, starting with an initial rate of 0.001. This adjustment aims to enhance model performance while maintaining generalization.

Result:



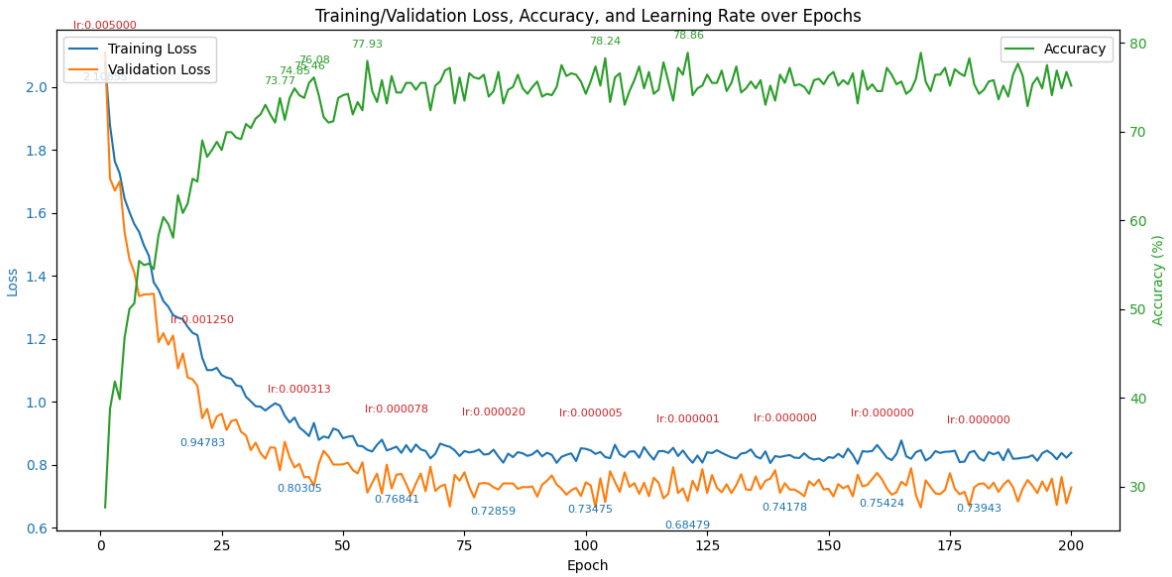
Accuracy increased to 76% from 60%, and loss decreased to 0.8% from 1.2%, but there are signs of overfitting and a shallow understanding of the differences between each animal.

Model 2:

Approach:

To address the overfitting issue, we adjusted the dataset split to 70% for training and 30% for validation, providing a more robust evaluation set. Batch normalization was introduced to stabilize and accelerate training, particularly for the dense layers. Additionally, the dropout rate was increased to reduce the risk of overfitting by improving regularization. The initial learning rate was also adjusted to 0.005, enabling a faster start to learning while maintaining control over convergence. These changes aim to enhance model performance and generalization.

Result:



The number of epochs in the previous setup was insufficient for the model to fully capture the dataset's complexity. Considering the relatively small dataset, we reduced the batch size and trained for more epochs while adjusting the hyperparameters based on the configurations of Model 1. To address the dataset's limitations, we increased the size of the validation and test sets, adopted a dynamic learning rate strategy, and introduced dropout for regularization. Additionally, the training data was augmented fivefold to improve diversity and prevent overfitting.

Despite these enhancements, the model struggled to learn finer details over time, achieving a validation loss of 0.717 and an accuracy of 79.01%. However, the model demonstrated robustness when tested on full-figure pictures, successfully classifying each animal with high accuracy. This suggests that while the model's performance on the training and validation sets has room for improvement, it generalizes well to unseen data in practical scenarios.

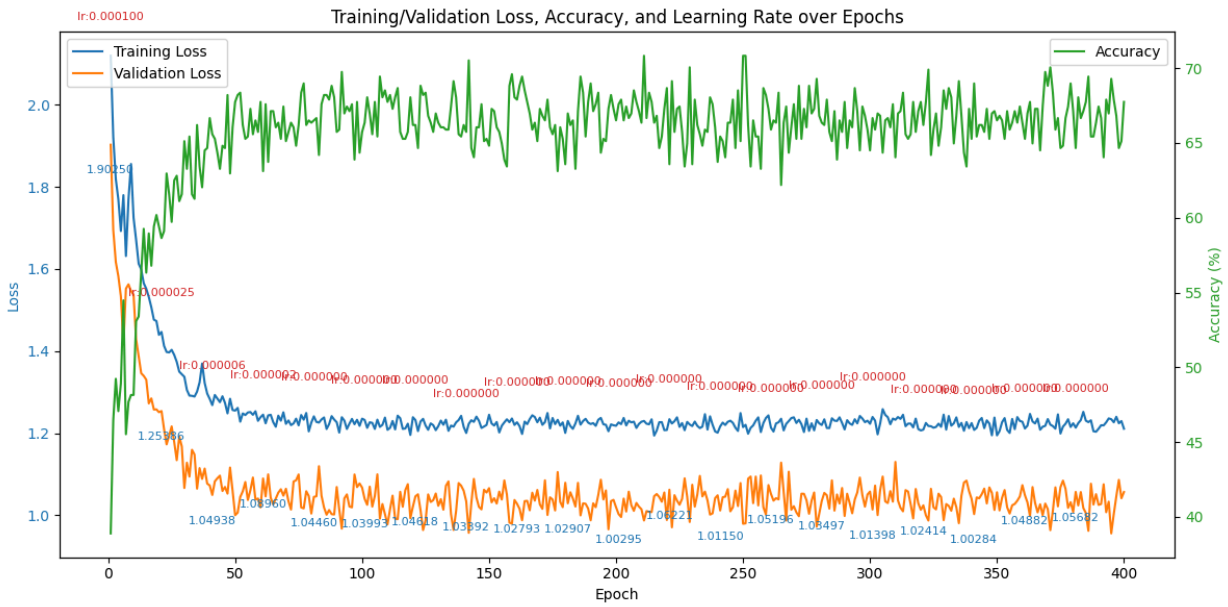
Model 3:

Approach:

In this iteration, we retained the previous hyperparameter settings except for the number of epochs and the learning rate, opting for smaller steps and more iterations to refine the training process. The dataset was split into 70% for training and 30% for validation to maintain a robust evaluation setup. The learning rate was adjusted to start at 0.0001, utilizing the Adam optimizer with a scheduler that halves the rate every 20 epochs to ensure gradual and stable learning. The model's capacity was increased in the initial layer to enable it to capture more diverse and complex features directly from the input images. Furthermore, the mid-layer configuration was

tuned to maintain a consistent 128 → 128 channel structure, facilitating the learning of more high level features within the same dimensional space before transitioning to higher dimensional feature spaces. These adjustments aim to enhance the model's ability to generalize and improve overall performance.

Result:



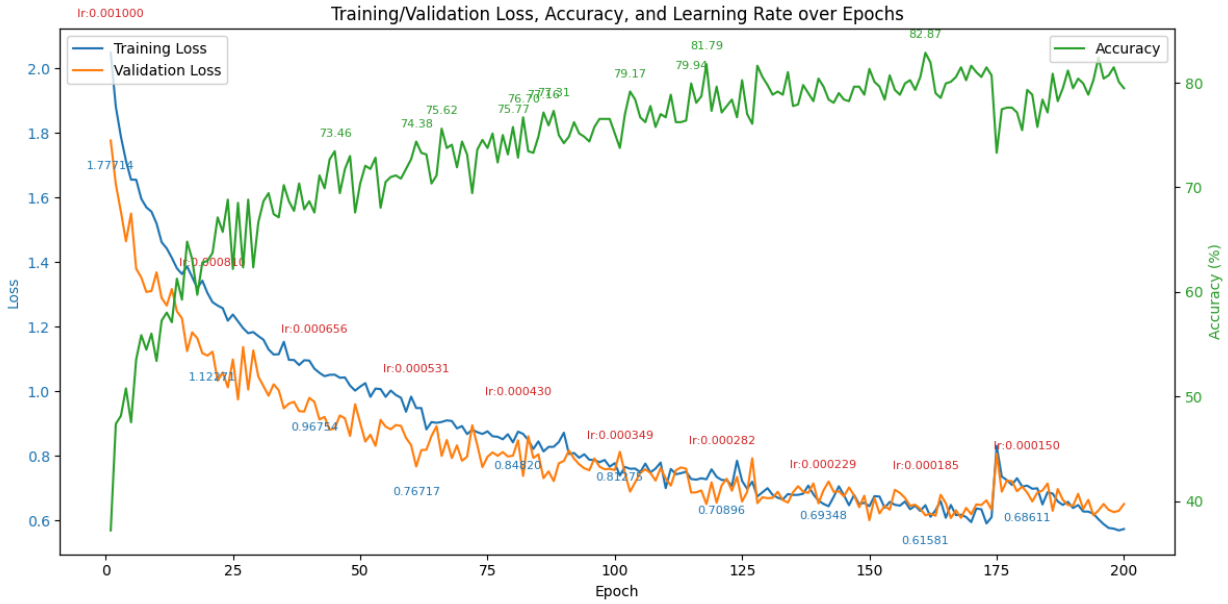
This time, the model was tested with 400 epochs and a very low learning rate. However, this configuration led to underfitting, as the model struggled to escape local minima and adequately learn from the data. The continuously decreasing learning rate exacerbated the issue, limiting the model's ability to make meaningful updates to its parameters. As a result, the model's performance remained suboptimal, even after completing 400 epochs of training. These outcomes highlight the need for a balance between the learning rate and the number of epochs to ensure effective learning and convergence.

Model 4:

Approach:

In Model 4, the initial learning rate was set to 0.001 to allow for more effective learning dynamics. Additionally, the number of epochs was reduced to 200, striking a balance between training duration and the risk of overfitting. This adjustment aimed to provide the model with sufficient time to learn meaningful patterns while avoiding the diminishing returns observed in longer training runs with lower learning rates.

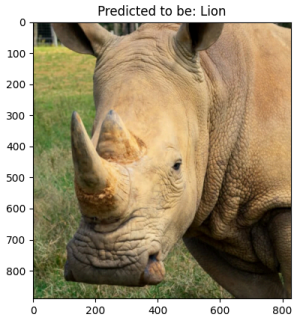
Result:

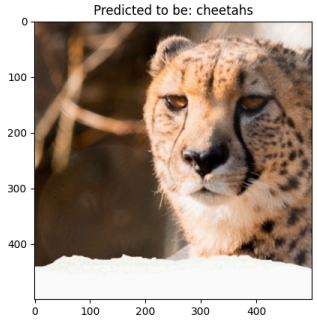


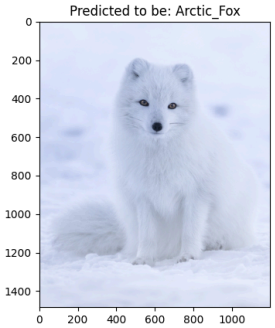
The model shows better accuracy and the best performance over epoch, training loss and validation loss keeps the distance of value over the epoch until the later epoch we observe the validation loss is lower than training loss indicating that the model started remembering the data rather than classifying. This seems like the limit of the model using this data. This is not something we can fix over tuning the hyperparameter but it is the matter of the size of the data.

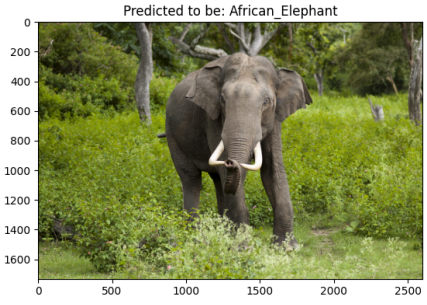
Evaluation

Throughout the course of our project, we trained and evaluated multiple CNN models to effectively classify endangered animals. While we identified one model that performed the best, the training process revealed several challenges and insights. Model 0 highlighted the need for significantly more epochs, as it struggled to fully capture the patterns within the dataset. Conversely, Model 2 demonstrated clear signs of overfitting, performing well on the training set but poorly on the validation set. This observation underscored the importance of incorporating better regularization and data augmentation techniques to improve the model's generalization and reduce overfitting. These findings guided our iterative adjustments to improve model performance across various configurations.

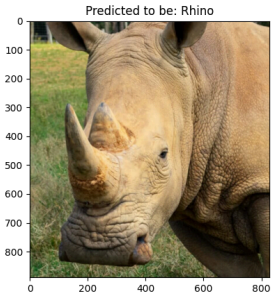


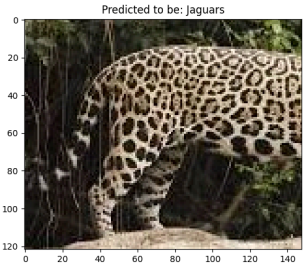


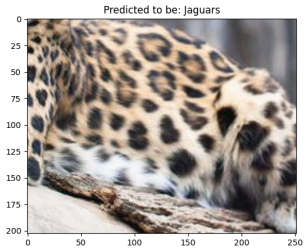




Using Model 1, we observed that one sample, featuring a rhino, was misidentified as a lion. This misclassification could be attributed to the orange hue in the background or an unusual angle in the image, which may have confused the model's feature extraction process. Despite this error, the remaining three samples were accurately identified, demonstrating the model's ability to classify the majority of the test images correctly. This specific misclassification highlights areas where the model's robustness to varying angles and backgrounds could be improved.







Model 2 not only successfully classified the rhino image correctly but also demonstrated robust performance by accurately identifying an additional set of seven full-figure animal images. Building upon this, Model 4 was evaluated using a more comprehensive test set of 22 images, including 11 full-figure animal images and 11 partial animal images where only parts of the animals were visible. Model 4 achieved an overall accuracy of 18/22 (82%), correctly classifying all 11 full-figure images and 7 of the 11 partial images. Interestingly, while partial images of some animals, such as jaguars and leopards, were challenging to classify due to their similarity and lack of distinguishing facial features, Model 4 excelled in classifying full-figure images. This suggests that the model relies heavily on holistic visual features for accurate classification, making it more effective when complete information about the animal's structure is available.

Overall, Model 4 is the best-performing model, with a loss of approximately 0.6 and an accuracy of 82%. Interestingly, the training loss is slightly higher than the validation loss. This indicates that the model has achieved a good balance between fitting the training data and generalizing to unseen data. The close alignment between the training and validation losses suggests that the model has successfully learned meaningful patterns from the data without overfitting or underfitting, making it well-suited to perform effectively on new, unseen inputs.

Conclusion

Since we have a relatively small dataset, we faced significant challenges in balancing overfitting and underfitting during model training. Overfitting models managed to achieve decent accuracy on the training set but plateaued after 100 epochs, showing no meaningful improvement and poor generalization to the validation set. Conversely, underfitting models failed to capture the underlying patterns in the data, resulting in subpar performance, and repeated adjustments to the optimizer did not help letting the model escape local minima.

We experimented with various regularization techniques such as dropout and normalization, alongside fine-tuning the training-validation split ratio, all of which played crucial roles in mitigating these issues. However, we found that the learning rate was the most critical factor. Through trial and error, we determined that starting with a learning rate of 0.001 and halving it every 10 epochs allowed the model to converge more effectively. This gradual reduction stabilized the loss changes to approximately 0.05 per epoch while ensuring that the validation loss remained only slightly higher than the training loss, a sign of good generalization and minimized overfitting.

Future Improvement and Real World Applications

Overall we are very satisfied with our final model achieving a 82.87% accuracy. Looking back on our model performance we would like to account for the possibility of low quality images and other objects like humans in the image to make our model more robust for real world applications. A possible solution to enhance our CNN to handle lower quality images better could lie in the findings from the research paper called "Fuzzy based Pooling in Convolutional Neural Network for Image Classification". In the paper they compare different pooling methods to see which performs the best on feature extraction. They test a pooling method called fuzzy logic to conventional pooling techniques such as max pooling and average pooling. The fuzzy logic pooling works by using two layers, the first identifies the dominant features by using the Gaussian membership function while the second uses the weighted average of the dominant features to reduce the size. The performance of their fuzzy logic pooling performed much better than maximum pooling and average pooling. Fuzzy pooling was able to extract the important features of the image better than previous pooling techniques. Changing our CNN from maximum pooling, to this fuzzy pooling could enhance the performance of our model in classifying low-quality images.

The push on improving our model in the scope of low quality images stems from our drive to implement this into a program that could help authorities and wildlife protection agencies to monitor these endangered animals using CCTV footage. Building on top of this idea we could implement our model into a web scraping tool that will monitor all kinds of media post for anything related to these endangered animals. The goal still in mind to catch any hunting related posts on youtube or social media to then alert wildlife agencies of the post for further

review or investigation. This would aim to help governments and wildlife protection agencies monitor posters on various types of media for people hunting these endangered animals.

Citations

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

T. Sharma, V. Singh, S. Sudhakaran and N. K. Verma, "Fuzzy based Pooling in Convolutional Neural Network for Image Classification," *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, New Orleans, LA, USA, 2019, pp. 1-6, doi: 10.1109/FUZZ-IEEE.2019.8859010.

Gramlich, Dennis, et al. 'Convolutional Neural Networks as 2-D Systems'. *arXiv [Math.OC]*, 6 Mar. 2023, http://arxiv.org/abs/2303.03042. arXiv.

Lichtner-Bajjaoui, Aisha. 'A Mathematical Introduction to Neural Networks'. *Advanced Mathematics Master Program of the Universitat de Barcelona*, 2020.