Building An Accurate And Efficient Animal Classification Application

A Deep Learning Project

Jingyi Qiu
Toyesha Kaushik

Index

Introduction

Our goal is to develop an efficient and accurate animal image identification algorithm. This algorithm will be integrated into an application that is accurate, compact, and fast to run on any device, including PCs, smartphones, and smart watches. Our focus will be on optimizing both accuracy and efficiency to identify the best-performing algorithm. We will evaluate and modify a baseline model as needed to meet our specifications and achieve optimal performance.

Accuracy is determined by using a function that computes the accuracy over the top-k predictions for the specified values of k. For example, in top-3 accuracy the output is considered the 'right answer' if the right answer appears in the top 3 guesses. Meanwhile, efficiency is determined using FLOPS (Floating-Point Operations per Second). This is a unit of measurement used to quantify the computing power (Sheldon, R. 2023). Efficiency is calculated by dividing accuracy by FLOPS count.

Model Enhancement

Data Preprocessing

Before the algorithm can be created, the quality of collected data, the number of samples per class (data distribution), and the image type should be thoroughly examined. Some images, despite being high quality, might have caused problems e.g clear reflections look like double-headed animals. Some images, such as drawings, were left in the dataset if accurate. Some subjects were far too camouflaged to be reliably used as training data. Most images, due to legal reasons, had watermarks. While most of them could be ignored, cases where the watermarks were too obstructive needed to be removed.
See *Appendix A* for some more examples.

Baseline Model Evaluation

Base models 1 and 2 use the original provided code, each containing 857, 239 training parameters. Note that model 1 utilizes the raw, unprocessed dataset whereas model 2 is trained on the preprocessed data. However, as shown in the Ablation Study below, the accuracy barely increases, with efficiency remaining below 60%. The Loss-Epochs graphs (see *Appendix B*) show a significant overfitting in the models. This could be due to the CNN model architecture, chosen learning rate, optimizer or number of epochs.

Model no.	Max Epochs	BN + Dropout	Optimizer	NN details	LR	Accuracy (%)	FLOPS (G)	Efficiency	Comments
1. (baseline)	10	N	Adam	CNN	0.001	37.86	0.69	54.87	*Overfitting*
2. (baseline)	10	N	Adam	CNN	0.001	40.25	0.69	58.33	*Overfitting*
3.	15	N	NAdam	CNN	0.001	37.17	0.69	53.87	*Overfitting, worse acc.*
4.	50	N	SGD	CNN	0.01	39.76	0.69	57.62	*Overfitting*
5.	15	Y - 0.5	Adam	CNN	0.001	58.31	0.69	84.51	*Less overfit Increased acc.*
6.	50	Y -0.5	Adam	CNN + DA (RandomRotation, RandomResizedCrop)	0.001	65.05	0.69	94.28	*Less overfit*
7.	15	Y - 0.5	Adam	CNN, Greyscale	0.001	37.62	0.62	60.68	*Less overfit*
8.	15	N	Adam	Transfer Learning, resnet18, RGB	0.001	75.32	0.96	78.46	*Trade-Off considered*
9.	15	N	Adam	Transfer Learning, diff. pre training data	0.001	-	-	-	*Insufficient processing power*

Ablation Study

Cross-entropy is consistently applied to all models. It is the most suitable lost function for multiclass classification problems (Andreieva, V. 2021), whereas other loss functions are more reliable for binary classification problems.

Evaluation and Visualization

Epochs: Started with 10 and increased to 50. This increase clearly exacerbated the overfitting tendencies of the SGD optimizer in Model 4. Therefore, the value was adjusted to 15 to achieve a balance between computational time and sufficient data.
Optimizers: The best optimizer returning highest accuracy, as gathered from models 2, 3, and 4, is Adam, and was therefore used to train the remaining models.
Learning rate (LR): Increasing from 0.001 to 0.01 led to significant gradient overshooting during training in Model 4. Hence, LR=0.001 was the most optimal value.
CNN architecture: Batch Normalization and Dropout layers were introduced in order to reduce the effects of overfitting. Results are immediate. Accuracy and efficiency increased significantly from Model 4 to Model 5 after implementing BN + dropout.
Data augmentation: Advanced techniques, such as RandomRotation and RandomResizedCrop, were introduced to expand the variety of training data and minimize overfitting. Accuracy & Efficiency increased further from Model 5 to Model 6.
Color channels: Retraining Model 6 on grayscale images decreased the accuracy significantly, possibly due to color being a key feature in differentiating animals of a similar species, e.g. birds.
Transfer learning: Most balanced performance in terms of accuracy/efficiency tradeoff.
See *Appendix B* for detailed visualization for all models.

Conclusion

Model Selection

Realistically, an animal classification program available to the general public would be more popular with a high accuracy. Therefore, the most optimal model is Model 8, which uses Transfer Learning to achieve this. The efficiency is also at an acceptable level, close to 80%. A tradeoff between accuracy and efficiency was favored, as efficiency may vary depending on the device being used.

Limitations

As seen in *Appendix D,* Google Colab GPU and RAM overloaded, and therefore local ones were used after Models 4 and 6 respectively. This may lead to minor discrepancies in expected accuracy values.
The main challenge faced when training all models, including our most efficient one, is creator bias in choosing the training data images. We found that:

1. Some images in the same class may be too similar, leading to overfitting
2. Choosing images to use is a highly objective process. What may be considered as a “good” image may not be considered one by another contributor
3. Limited knowledge could lead to data contamination. For example, *Panthera onca* and *Panthera pardus* look extremely similar to the naked eye, so an image of one could accidentally be classified as another.
4. Unequal numbers of training images for each class could lead to identification bias.

There is also difficulty in sourcing images. This could could be because:

1. There is a lack of representation of sex of an animal, leading to difficulty in identification. E.g. *Panthera leo* males are considered the ‘default’, leading to less images of females, which may mean more difficult identification
2. Lack of representation of life stages. E.g. Pupal/larval stages are not frequently present in datasets
3. Most images available on a public domain are watermarked. While some watermarks are less obvious and can be ignored, some cover the subject of the image, and therefore cannot be used.
4. Some animals are difficult to photograph. This could either be due to ease of access e.g. underwater animals such as *Monodon monoceros*, or due to extinction of certain classes.
5. Drawings or models of animals may not be scientifically accurate, leading to discrepancies.

Another possible issue is training the model to ignore certain aspects of a background or include the background. For example, the *Pavo cristasus* has feathers that may be ignored due to being considered a

background, whereas the *Lemur catta* images, due to the frequent presence of branches, may have the branches mistakenly identified as key features.

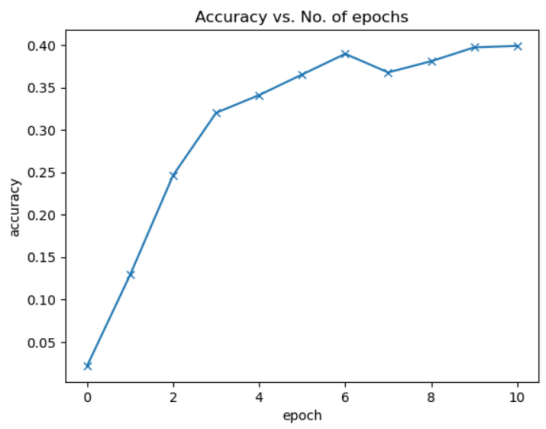
Appendix A - Dataset preprocessing

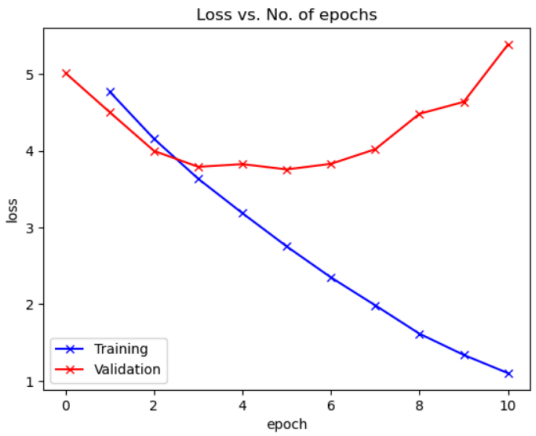
Here are some examples of the images that needed to be disregarded in the dataset:

Class	**Original Sample Size**	**Problems**	**Final Sample Size**	**Quality**
aethia-cristatella	50	Fake/Seemingly AI-generated Too compressed Subject too far away Odd angle Overlapping subjects Labels covering subject	46	High
agalychnis-callidryas	34	Other objects as the focal point Zoomed in too far to see distinguishable features Odd angle Uncommon posture	29	High
ailuropoda-melanoleuca	31	Other animals in the picture Albinism (rare case/misclassification)	28	Mid

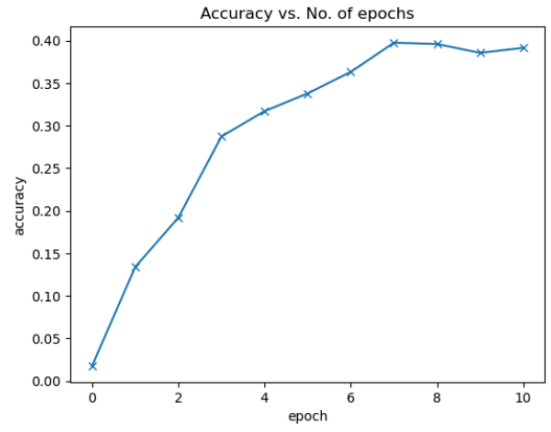
Appendix B - Visualization

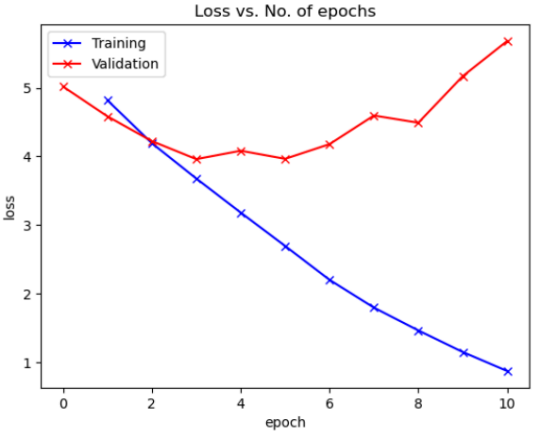
Model 1: Baseline



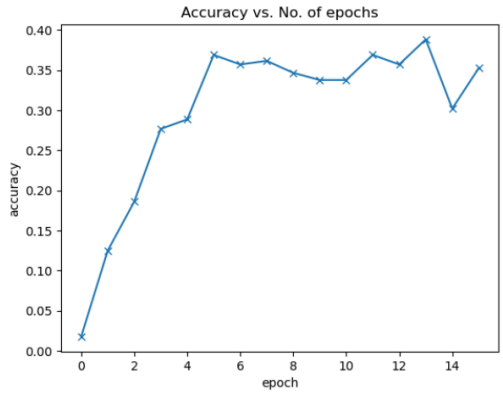


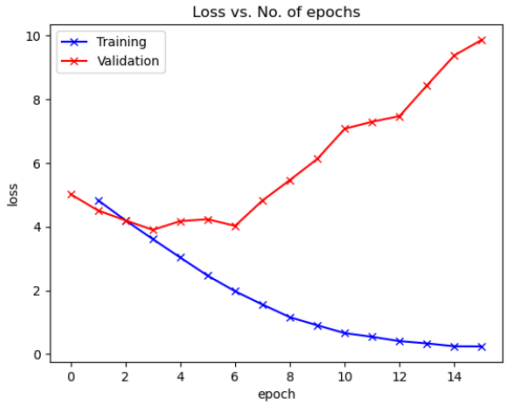
Model 2: Preprocessed images, Baseline Model



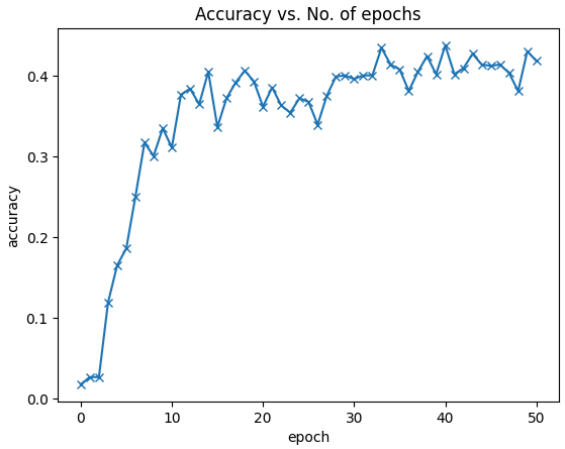


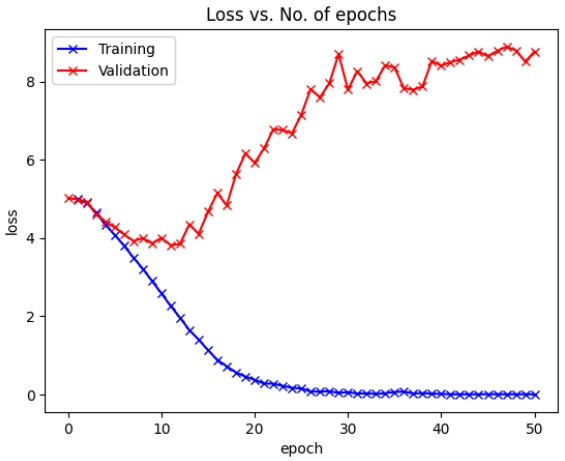
Model 3: NAdam



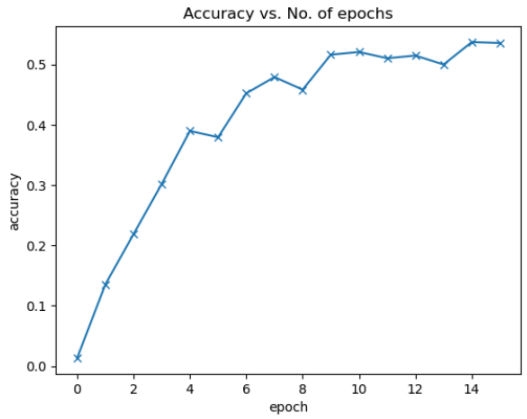


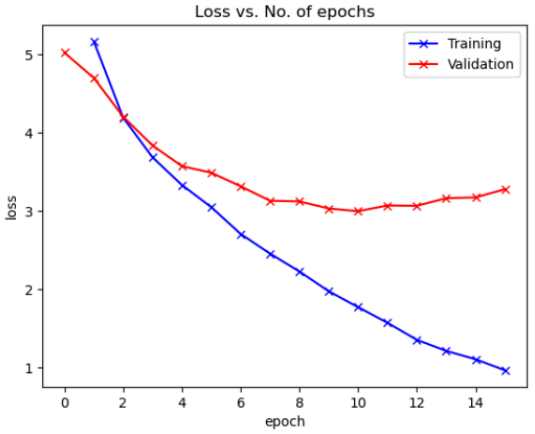
Model 4: SGD



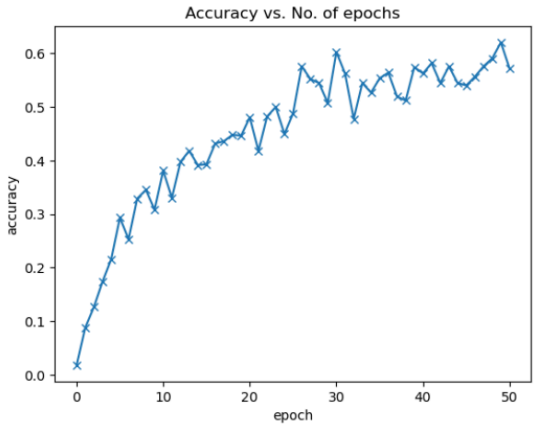


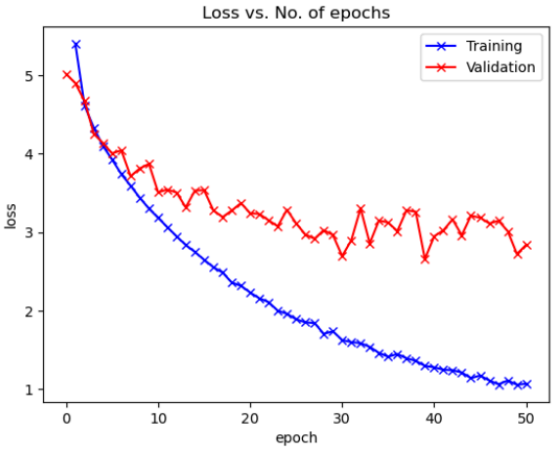
Model 5: BN, Dropout



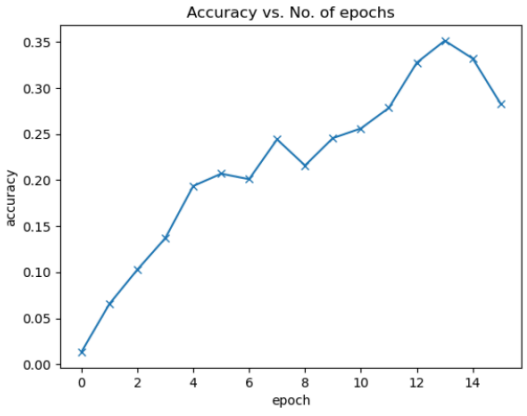


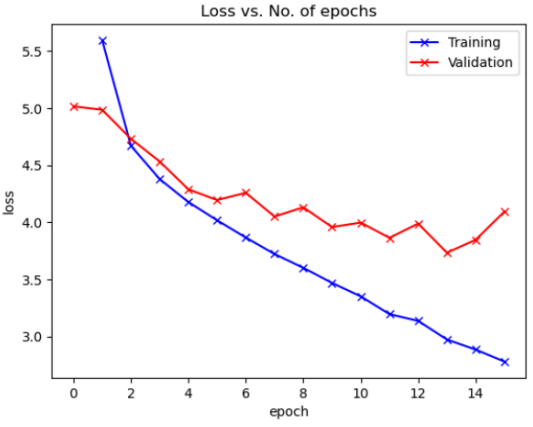
Model 6: Data Augmentation



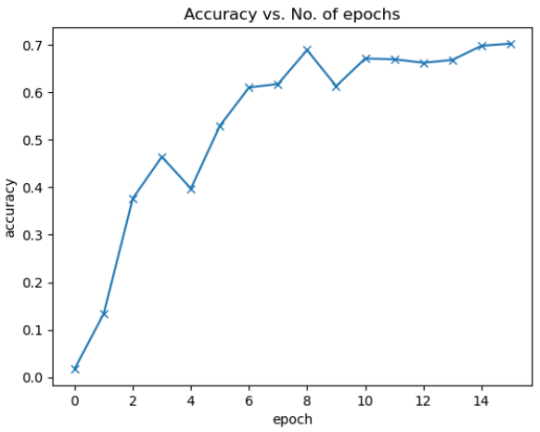
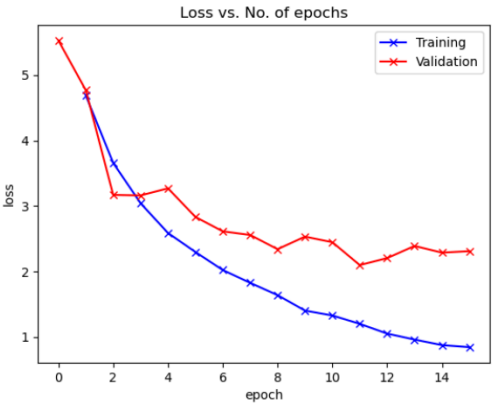


Model 7: Greyscale





Model 8: Transfer Learning **(FINAL/MOST OPTIMAL)**

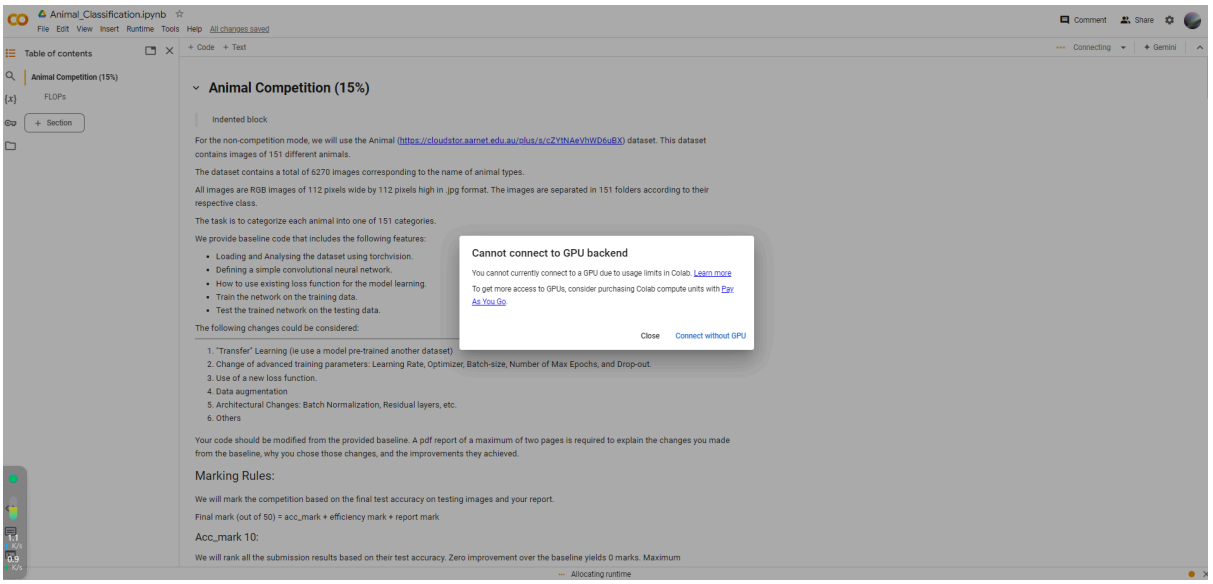
	
Model 9: Transfer learning with pre training data	
Invalid	

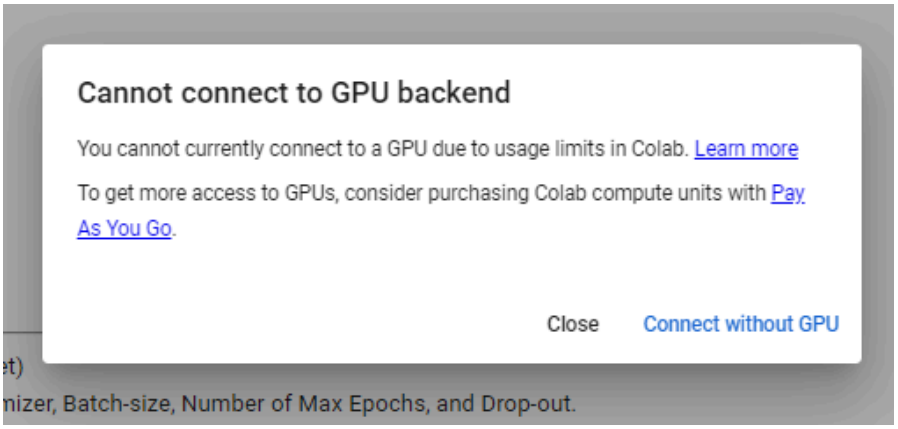
Appendix C - References

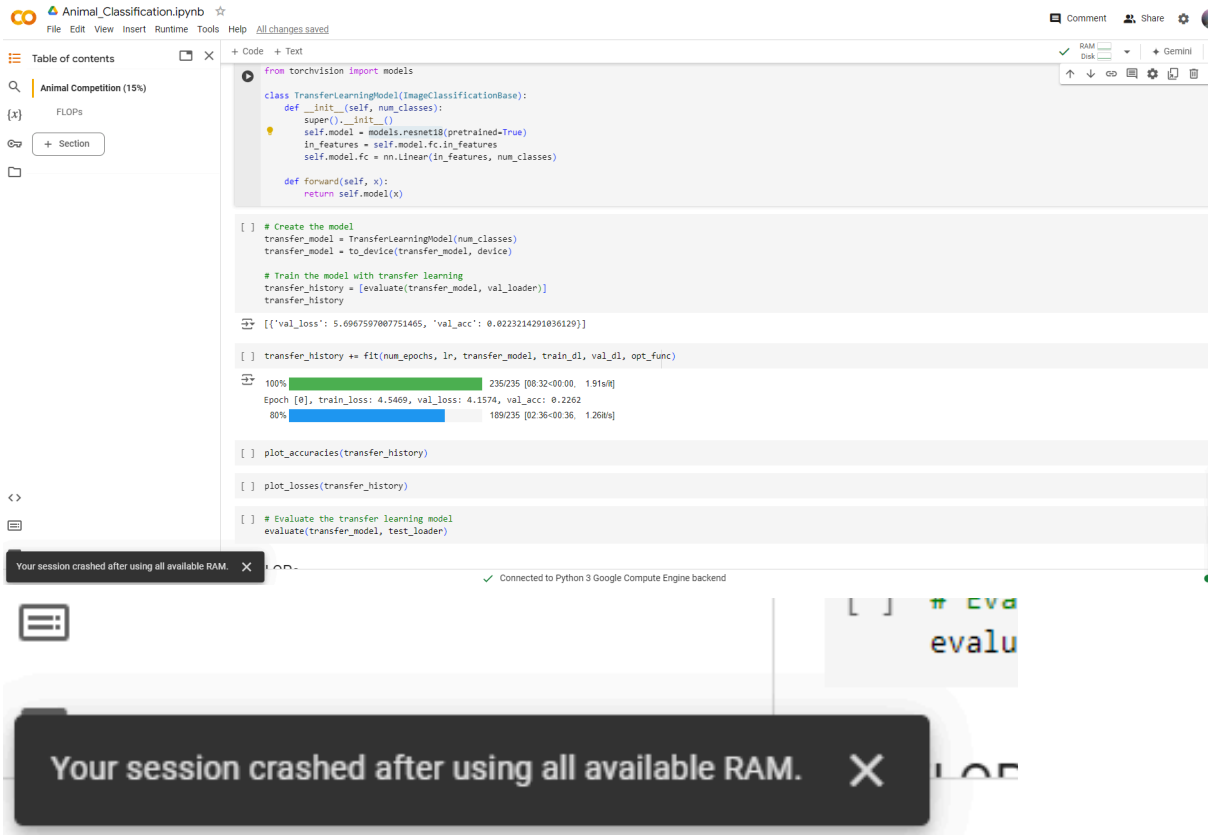
Andreieva, V. et al, *Generalization of Cross-Entropy Loss Function for Image Classification* (2021, Jan). Published via Mohyla Mathematical Journal, accessed via ResearchGate [19 Jun 2024] <https://www.researchgate.net/publication/349850933_Generalization_of_Cross-Entropy_Loss_Function_for_Image_Classification>

Sheldon, R. *floating-point operations per second (FLOPS)* (2023, Aug). Published and accessed via TechTarget [19 Jun 2024] <https://www.techtarget.com/whatis/definition/FLOPS-floating-point-operations-per-second>

Appendix D - GPU & RAM limits




Cannot connect to GPU backend
You cannot currently connect to a GPU due to usage limits in Colab. Learn more
To get more access to GPUs, consider purchasing Colab compute units with Pay As You Go.
Close
Connect without GPU
nizer, Batch-size, Number of Max Epochs, and Drop-out.


Animal_Classification.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Animal Competition (15%)
FLOPs
Section
+ Code + Text
Gemini
from torchvision import models
class TransferLearningModel(ImageClassificationBase):
def __init__(self, num_classes):
super().__init__()
self.model = models.resnet18(pretrained=True)
in_features = self.model.fc.in_features
self.model.fc = nn.Linear(in_features, num_classes)
def forward(self, x):
return self.model(x)
# Create the model
transfer_model = TransferLearningModel(num_classes)
transfer_model = to_device(transfer_model, device)
# Train the model with transfer learning
transfer_history = [evaluate(transfer_model, val_loader)]
transfer_history
[{'val_loss': 5.6967597007751465, 'val_acc': 0.0223214291036129}]
transfer_history += fit(num_epochs, lr, transfer_model, train_dl, val_dl, opt_func)
100%
235/235 [08:32<00:00, 1.91s/it]
Epoch [0], train_loss: 4.5469, val_loss: 4.1574, val_acc: 0.2262
80%
189/235 [02:36<00:36, 1.26it/s]
plot_accuracies(transfer_history)
plot_losses(transfer_history)
# Evaluate the transfer learning model
evaluate(transfer_model, test_loader)
Your session crashed after using all available RAM.
Connected to Python 3 Google Compute Engine backend
Your session crashed after using all available RAM.