CS510 Assignment 5

Wild Animals Part 2

Gauthami, Sanket, Shreya

1. Recap: Wild Animals Part 1

Wild Animals part 1 had a total 1643 images of different classes namely: Bighorn (200), Bobcat (200), Coyote (200), Gray Fox (210), Javelina (200), Mule Deer (200), Raptor (402), White-tailed Deer (200).

There were a lot of discrepancies with the data, despite which we had conducted the following experimentations and successfully got results.

First, we had to normalize every image to the same dimension before we started the training process. Hence, we resized all of them to a size of (256, 256). This size was chosen as it is the least one an image has in the dataset. We also did data augmentation to increase the performance of our network. Images are horizontally flipped and rotated at an angle of 20 degrees. Further, we built a vanilla CNN with 3 convolutional layers followed by a linear layer for our classification task. We decided to use 5-fold cross validation for this task to make use of our dataset more effectively. With this in mind, we trained our model for two cases: 1] Resized images are converted to grayscale and then trained and 2] CNN is trained on the resized original images; and got the following results:

	Accuracy
Training set	100%
Test set	75%

Table 1: Images are not grayscale

	Accuracy
Training set	100%
Test set	50%

Table 2: Images are grayscale

2. Wild animals 2- introduction:

The wild animals' part 2 had a total of 29,394 with a size of 24 GB. Most of the images in the dataset were of size 1MB each. This makes the dataset very “heavy”. The data was segregated into its respective classes using the file name and the following class counts were observed:

Bighorn sheep	Gray Fox	Raptor	Bobcat	Javelina	White tailed Deer	Coyote	Mule deer
5363	1423	4000	806	422	1727	5955	9698

Table 3: Per class count of images

After segregating the classes, Exploratory Data analysis was performed to understand the combination of images and the range of image sizes present in the dataset. It was observed that the dataset had a combination of both day and night shot images.

All images in the dataset were colored with image dimensions of (1944, 2592, 3) (1920, 2560, 3) (2448, 3264, 3) (128, 128, 3)

3. Analyzing the data of Wild Animals Part 2:

The wild animals' part 2 dataset was quite interesting, and there were a lot of discrepancies as listed below:

i. Most of the image was taken by the background and not by the object of interest.
ii. Often the object of interest would be partially/completely occluded by any object.
iii. The presence of more than one object of interest in one image.
iv. Many images in the dataset often lacked the animal as whole. Images with just the head, back, horn, etc. were observed.
v. Often the resolution of the images was quite low, especially for images shot at night.
vi. Images often had minuscule sizes of objects of interest which were getting camouflaged with the background objects.

Figure 1: Discrepancies observed in the dataset

The images were taken at day and night. We guess those were taken by an automated camera. We wanted to segregate images into day and night categories as we planned to evaluate the performance of our models separately for the images taken in sunlight and those taken at night or dawn.

Classifying the images to days and nights wasn't easy due to the following reasons:

i. Wrong time stamps were provided for images of most of the classes.
e.g.: For Bighorn sheep the actual time shot was 5am while the time stamp in the filename of the image shows 4am. The actual time shot was 5am and 6am for bobcat while the time stamp in the filename shows 4am and 5am respectively.
ii. The day images couldn't be taken from the default 6am- 6pm as most images even at 5am and at 7pm had daylight. This had to be customized for almost all classes of animals.
iii. The night images had a mixture of artificial lighting images and images in darkness.

Figure 2: 3am images under streetlights of bighorn sheep

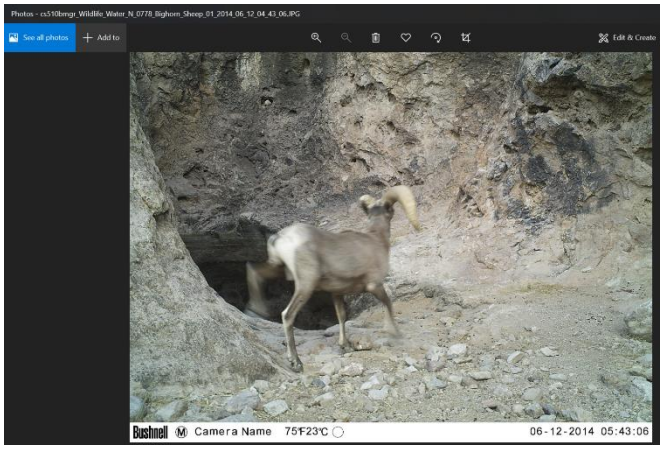

Figure 3: Wrong time stamps provided

3.1 Our solution to the above challenges:

We tackled these issues by implementing the day and night classification using the timestamp present in the file name of all the images. The last 3 figures of the file path indicated the time in Hr:Min:sec format which could be extracted using delimiting and split functions. Upon extraction of the time, each animal class was analyzed to find the duration of daylight and the upper, lower limits for the same were calculated. On figuring the day and night intervals, the respective images for each class were segregated into day and night using file handling

operations. This partitioned most of the images for each class into day and night, we left some images wrongly partitioned. We studied that it was mainly due to change of time as a reason of daylight savings.

4. Experimentation on Wild Animals Part 2:

The team had tested various conventional deep neural network architectures models such as ResNet 18, DenseNet121 etc.

ResNet 18

ResNet 18 is a 72-layer architecture with 18 deep layers. The architecture of the Resnet 18 network aimed at enabling enormous amounts of convolutional layers to function efficiently.

Figure 4 shows a sample of the Resnet 18 architecture. As you can see in the above picture, we added a linear layer to the existing ResNet 18 architecture. This architecture developed seemed to be robust and we can replace the upper ResNet 18 layer with any pre-trained or self-built CNN architecture and implement our purpose of image retrieval.

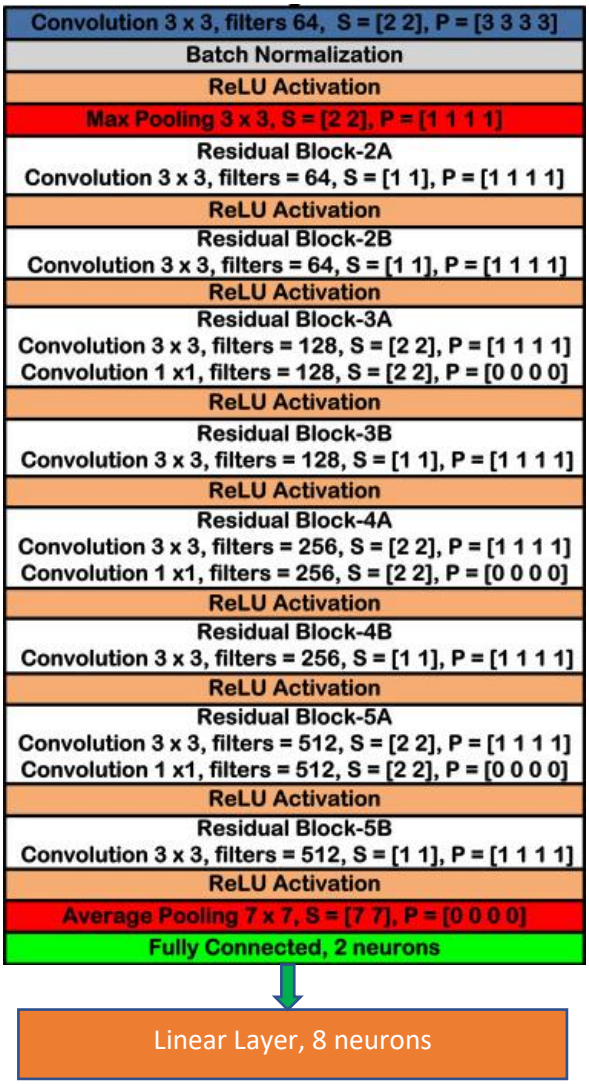

Figure 4: Sample architecture of Resnet 18

DenseNet 121

Dense Convolutional Network (DenseNet) connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer - our network has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

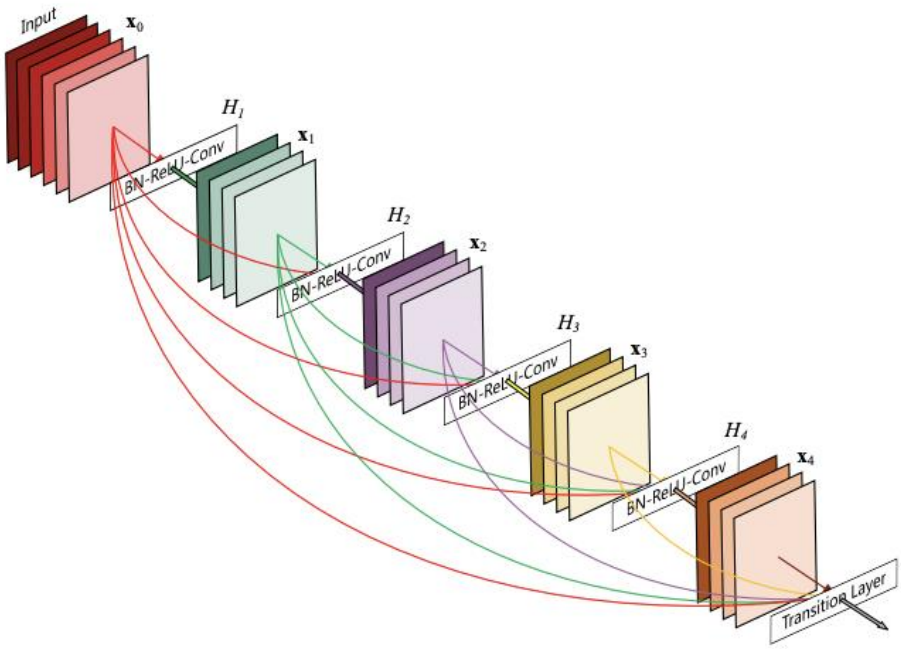

Figure 5: Sample architecture of DenseNet

Considering the dataset set to be huge with a size of 24GB and 29k samples, the team had decided to experiment with distributed training using PyTorch.

We have extensively used DenseNet 121 for our experiments. DenseNet-121 has the following layers:

- 1 7x7 Convolution
- 58 3x3 Convolution
- 61 1x1 Convolution
- 4 AvgPool
- 1 Fully Connected Layer

4.1 Distributed Training

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs. The team has utilized the distributed package of pytorch to implement distributed training. This has enabled us to easily parallelize the computations across processes and clusters of machines. All essential packages such as multiprocessing, Distributed Sampler, Data Loader, and Distributed Data Parallel as DDP are imported.

The distributed package uses the Gloo backend. The Gloo backend is quite handy as a development platform, as it is included in the pre-compiled PyTorch binaries and works on both Linux (since 0.2) and macOS (since 1.3). It supports all point-to-point and collective operations on CPU, and all collective operations on GPU.

On setting four environment variables on all machines, all processes will be able to properly connect to the master, obtain information about the other processes, and finally handshake with each other. The environment variables are as follows:

- MASTER_PORT: A free port on the machine that will host the process with rank 0.
- MASTER_ADDR: IP address of the machine that will host the process with rank 0.
- WORLD_SIZE: The total number of processes, so that the master node knows how many workers to wait for.
- RANK: Rank of each process, so they will know whether it is the master process of a worker.

The python APIs majorly used in this project are torch, torch.nn, torch.nn.functional, torch.Tensor. The torch package contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors. Additionally, it provides many utilities for efficient serializing of Tensors and arbitrary types, and other useful utilities. Torch.nn are the basic building blocks for graphs which includes Containers, Convolution Layers, pooling layers, Padding Layers, Non-linear Activations (weighted sum, nonlinearity), Non-linear Activations (other), Normalization Layers, Recurrent Layers, Transformer Layers, Linear Layers, Dropout Layers, Loss Functions etc.

So, in our case, we had in total 21000 images in our training set. With a batch size of 32, we get 657 batches. We distributed our data such that the master machine works on 328 batches while the slave on 329. This helped reduce the time required for training.

We used Lotus and Saturn machines from CS 120 lab as they have a CUDA memory of 12GB which was suitable for our tasks.

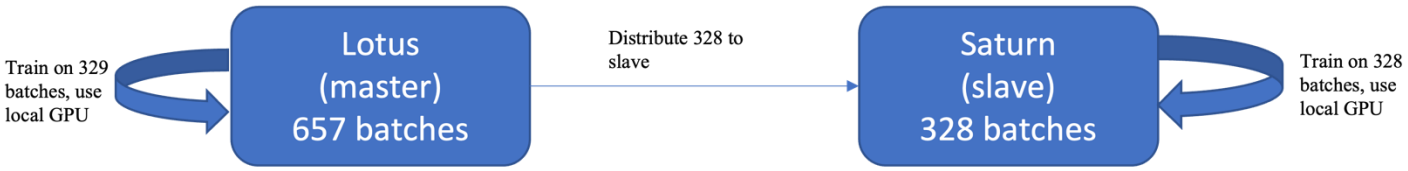

Figure 6: Distributed training

5. Results and Evaluation on Wild Animals Part 2:

First, our team experimented with the ResNet 18 model along with data augmentation. For data augmentation, we rotated and flipped each image in our dataset. Our dataset size increased to 50k samples. We experimented with our ResNet 18 model without data augmentation as well. The following are the results:

	ResNet 18 (with data augmentation)	**ResNet 18 (w/o data augmentation)**
Test set accuracy	54.57%	72.71%

Table 4: Test set accuracy comparison with and without data augmentation

As is visible, our model did well when we did not augment the data. **Our analysis is that the model over-trained when data was augmented and over-fitted the data. This is visible from the training and validation curve in Figure 7. So, we carried out further experiments with original data (without augmentation).**

Later, we tried to do the same experiment with other models without data augmentation and we had memory issues with GPU. So, we used a dataset containing 20k images on ResNet and DenseNet models. Table 5 shows a comparison of the test set accuracy of different ResNet18 and DenseNet121. Figures 8 & 9 show the respective training and validation curves.

	ResNet 18	**DenseNet 121**
Test set accuracy	72.71%	75.22%

Table 5: Comparison of ResNet 18 and Densenet 121 performance

As we can notice, DenseNet performs well than ResNet 18. This can be attributed to the fact that DenseNets contain purely convolutional layers connected to each other in a simple pattern. The layers are connected densely to the previous layers. This means in a normal CNN architecture (ResNet18), each convolution layer is connected to its previous layer only. But in DenseNet, each layer is connected to every other layer, i.e., each layer receives the feature maps from all its precedent input layers. This makes DenseNet extract more image features than ResNet at each layer. Furthermore, ResNet sums up the output feature maps of the layer with the incoming feature maps at each layer, while DenseNet just concatenates them. This helps to preserve the features from the previous layers as it is, without any computation. This explains why DenseNet 121 performed well in our case. But the difference is not significant as both ResNet and DenseNet do have dense connections.

We also analyzed the performance on daylight and night images separately, following are our results:

	DenseNet Daylight images	**DenseNet Night images**	**ResNet Daylight images**	**ResNet Night images**
Test set accuracy	76.79%	74.70%	70.04%	77.59%

Table 6: Testing accuracy on daylight and night images

As we can see, DenseNet 121 performs well in daylight while ResNet18 on night images. The reason we think for this is- DenseNet just concatenates the features from previous layers while ResNet does a summation and computation on the features. Night images are complex and require computation. Hence, ResNet18 works better as it can understand the complex patterns involved in the objects in the images taken at night.

For all our experiments, we have done an 80:20 split of our dataset into train and test.

Figure 7: ResNet18 train and validation loss with data augmentation

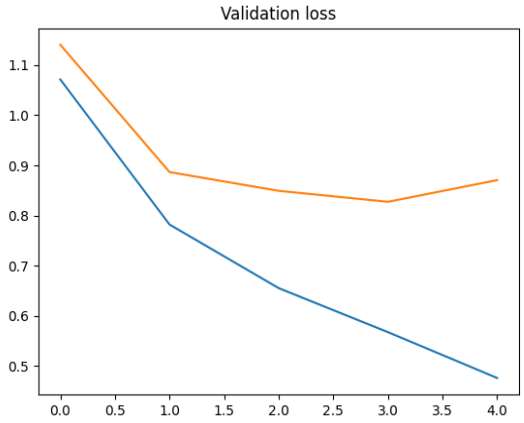

Figure 8: ResNet18 train and validation loss w/o data augmentation

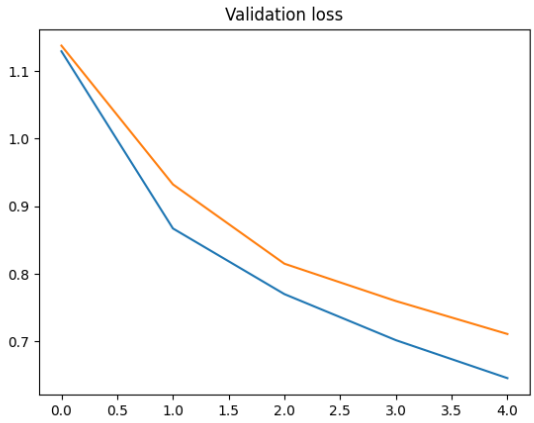

Figure 9: DenseNet121 train and validation loss w/o data augmentation

We have used PyTorch for all our experiments, including the distributed training.

6. Conclusion:

Our conclusion is that data augmentation did not help in our case. Instead, the models performed well on the original data. We then picked up two models for our study- ResNet18 and DenseNet121. We chose these models as they are widely being used for computer vision tasks. Our result is that, overall, DenseNet performs well than ResNet due to densely connected convolution layers and concatenation of features from all the layers. When we analyzed our models' performance separately on daylight and night images, we found out that DenseNet121 performs better on daylight while ResNet on the images taken at night. This can be attributed to the fact that ResNet does heavy computations within its internal layers while DenseNet just does concatenation.

Night images are mostly dark and hence it requires some computation to recognize an object within. Hence, ResNet is better at handling them.

Further, we were successful in training on a distributed environment on CS 120 Lab machines and this helped reduce our training time by 50% as we trained our model on two different machines with data distributed equally between them.

7. Future Work:

To obtain higher accuracy, we think more work should be done on the dataset. Like, we should remove the outliers from the dataset. Here, outliers can be those images which have more background visible than the animal itself, wherein the animal face is not visible, etc. Segmentation techniques can also be used to extract just the animal from the image and then feed it to CNN. We can also work on decreasing the size of each image by various image processing techniques. Also, we propose to use ensemble learning which uses multiple algorithms to get better performance than what we obtained with the use of single model alone.

8. References:

[1] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261-2269, doi: 10.1109/CVPR.2017.243.

[2] X. Ou et al., "Moving Object Detection Method via ResNet-18 With Encoder–Decoder Structure in Complex Scenes," in IEEE Access, vol. 7, pp. 108152-108160, 2019, doi: 10.1109/ACCESS.2019.2931922.

[3] https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a

[4] https://iq.opengenus.org/architecture-of-densenet121/

[5] https://www.sciencedirect.com/topics/computer-science/residual-network

[6] https://pytorch.org/hub/pytorch_vision_resnet/

[7] https://pytorch.org/hub/pytorch_vision_densenet/

[8] https://pytorch.org/tutorials/intermediate/ddp_tutorial.html