Categorizing Animals in the Wild

Deep Learning Project 2019

I. INTRODUCTION

Wild Cams are able to capture large amount of image data which can be exploited by biologists to monitor biodiversity and population density of animal species. To explore the huge data, the species in the image data need to be detected. The detection becomes challenging if the context of the image, such as geographical region, differs between the training and test data. In addition, species which are not present in the training data are almost impossible to detect in the test data. The task is to classify species automatically in the test data as accurately as possible. [1]

Naturally, classifying species from such a huge data is beyond human resources and is usually taken over by computers using methods such as machine learning and deep learning. The scope of this work is to study how to detect objects in images using latest state-of-the-art deep learning method called YOLO.

The work starts with introducing the dataset and describing the YOLO system. Then continues with methods to create, train and evaluate a deep learning model. Finally, the results are presented and conclusions are drawn.

The source code of this work can be found in https://letsirk.github.io/dl-project/.

II. DATASET

The dataset contains Wild Cam color images taken in two different areas: 196 299 labeled images captured in Southern California (Figure 1) and 153 730 non-labeled images in Idaco. The labels are numeric values from 0 to 22 corresponding to specific species (described in Table I). The labeled images are used to train the deep learning model and non-labeled images to test the model in Kaggle. [1]

TABLE I
THE LABELS OF THE DATASET

Id	Label	Id	Label	Id	Label
0	empty	8	rabbit	16	bobcat
1	deer	9	bighorn sheep	17	cat
2	moose	10	fox	18	dog
3	squirrel	11	coyote	19	opossum
4	rodent	12	black bear	20	bison
5	small mammal	13	raccoon	21	mountain goat
6	elk	14	skunk	22	mountain lion
7	pronghorn antelope	15	wolf		

The diversity of the labels in the training images are shown in the Figure 2. Notice that the dataset does not cover all the labels. In addition, the majority label (0 = empty) is dominating the dataset by having 131 457 images in total. Therefore, this label is excluded from the figure to capture the details of the other labels. The labels in the middle of the figure have similar spread unlike the labels in the beginning and end whereas their spread is imbalanced.

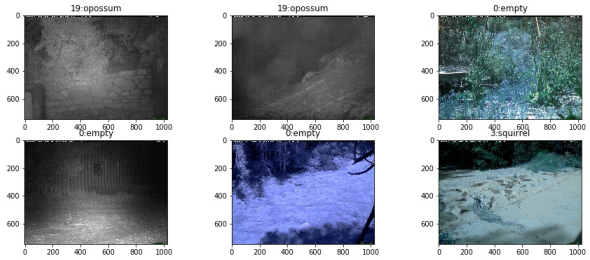

Fig. 1. Examples of labeled training images.

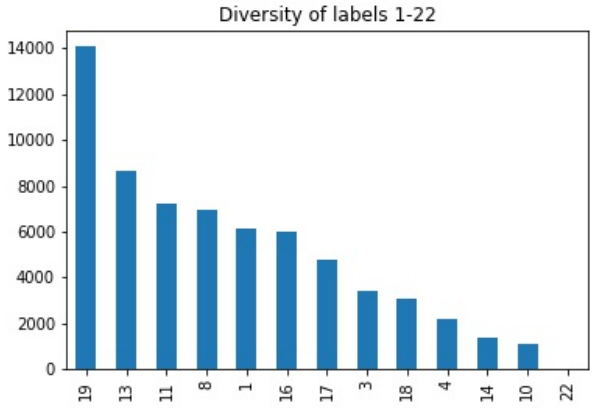

Fig. 2. The diversity of the labels in the labeled images.

This is a common problem in the machine learning tasks and can lead to ignoring the minority labels in favor of the majority labels. In the worst case the trained model predicts only majority labels and never minority labels. To tackle this problem, the minority labels can be up-sampled (randomly duplicating samples in order to strengthen label's occurrence in the dataset) and majority labels down-sampled (randomly removing samples in order to prevent domination of the dataset). [2]

The training images were processed as mentioned above yielding to 3000 samples/label, 42 000 training samples in total. Furthermore, the size of the training images (1024x748 pixels) and validation images (1024x645) was reduced to 32x32 pixels. The final dimension of each image is 3x32x32 where 3 stands for color channels: Red, Green and Blue.

III. YOLO DARKNET-53

Darknet-53 is a new neural network launched together with state-of-the-art real-time object detection system called YOLO (You Only Look Once). Unlike many other object detection systems, e.g. R-CNN and Fast R-CNN, which perform multiple image manipulations (e.g. scaling, dividing to regions) to a single image before feeding it into a neural network, YOLO takes the whole image without manipulation into a single neural network and returns predicted bounding boxes and its probabilities. Therefore, YOLO is exceptionally fast on predictions: 1000 times faster than R-CNN and 100 times faster than Fast R-CNN. [3]

The architecture of the Darknet-53 includes 53 convolutional layers (Figure 3) using mainly 3x3 and 1x1 kernels like the residual network in ResNet. The network ends up to layers average pooling, connected and softmax. The last layer describes the probability of the sample belonging to certain class. [4]

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig. 3. The architecture of the model.

This network yields to similar or even better results than ResNet-101 and ResNet-152. (The number after the network name indicates the number of the convolutional layers in the particular network.) In addition, Darknet-53 is faster than ResNet networks because its structure utilizes GPU better and it consists of fewer layers. [4]

This work utilizes only a part of the YOLO system; the architecture of the Darknet-53 is implemented in the next chapter.

IV. METHODS

This chapter describes the methods used in this work. The work starts with exploring a baseline code. Then continuing with the creation of the model, training and evaluating it. All of these methods are briefly over-viewed in the next subsections.

A. Baseline

Kaggle provides a CNN baseline code [5] which uses a four-layer convolutional model of Keras [6] and is implemented with TensorFlow library. It includes performance statistics such as Kaggle prediction result and achieved accuracy score in training. This work benefits from parts which are related to the data handling and compares performance results to its own results (more details in section V).

B. Model

The model is created according to the architecture of the Darknet-53 (Figure 3) using PyTorch library.

C. Training

The training images are split into two sets where 90 % of the images (37 800) cover the training of the model and 10 % (4 200) its validation. The training is executed on 90 epochs with batch size of 16. In the forward pass, after the last layer of the model called softmax, the output data is transferred through a loss function called negative log-likelihood (NLL). In practice, these two work well together. In the backward pass, stochastic gradient descent (SGD) is used as an optimizer. To speed-up the training, the display's GPU device is activated.

D. Evaluating

The evaluation is done by several metrics: accuracy score, confusion matrix and Kaggle results. The first one measures the ratio of correctly predicted labels to the total labels. The higher the value is, the better the model performs. The second metric, confusion matrix, is an error matrix in a table layout. The matrix visualizes true and predicted labels and captures which labels are classified under wrong label. The last metric is used to evaluate the performance of the model on the predictions of the test data. The predictions are submitted to Kaggle and the results are compared to other participants.

V. RESULTS

Once the model was trained according to IV-C, the accuracy scores of the training and validation data were compared to the CNN baseline (IV-A) scores. Although, the accuracy scores of the Darknet-53 model are satisfying, the results in Table II indicate that the level of the accuracy is better for CNN baseline model.

TABLE II
ACCURACY RESULTS

Description	Training score	Validation score
CNN Baseline	0.91	0.93
Darknet-53	0.85	0.86

In addition, when digging deeper to labels which are poorly predicted versus to the true labels using the confusion matrix (Figures 4 and 5), the model seems to predict every now and then some labels poorly. Especially, the labels of empty images (0=empty) are confused to other labels and vice versa. This can be explained that the target species are not always distinguishable from the image. One of the reasons could be because of bad lighting.

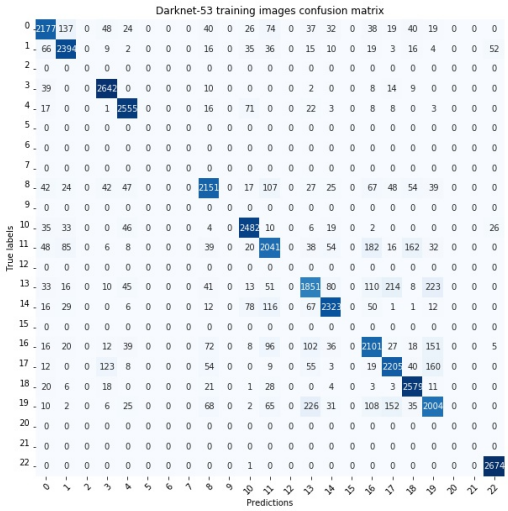

Fig. 4. Confusion matrix of the training images. The darker the color is, the better the model has predicted labels correctly.

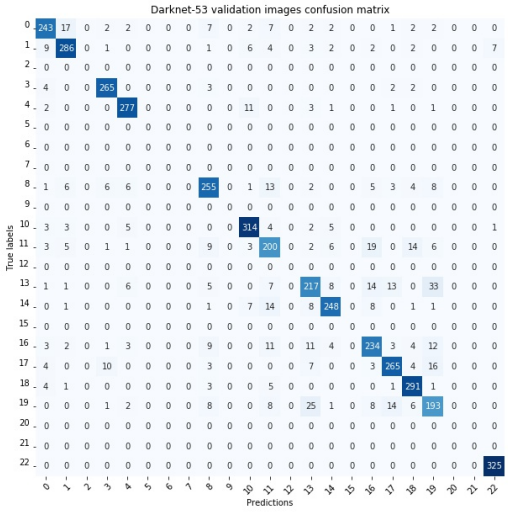

Fig. 5. Confusion matrix of the validation images. The darker the color is, the better the model has predicted labels correctly.

After the training, the model was used to predict the labels of the test images. The predictions were submitted to Kaggle and the results were compared to other participants (Table III). The model used in this work was placed somewhere in the middle of the leader board but below the CNN baseline model. Notice that even the best model achieves a quite low precision: only 25 % of the images are labeled correctly. To beat the best model, the model should at least benefit of pretrained weights and bounding boxes (e.g. coyote is located to x- and y-coordinate with specific width and height).

TABLE III
KAGGLE RESULTS (MAY 3^{rd}, 2019)

Description	Score
Worst	0.021
Darknet-53	0.089
CNN Baseline	0.125
Best	0.248

Furthermore, when exploring few examples of the test images (Figure 6), the model has clearly mislabeled the last two images. The labels of the other images are not obvious; the images might be empty or not.

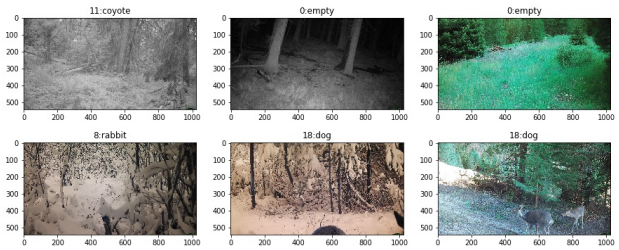

Fig. 6. Examples of the predicted labels of the test images.

VI. CONCLUSION

In the case of predicting species from the images of the Wild Cam, the results conclude that the architecture of the Darknet-53 used in YOLO alone is not the best model to be used. In the future studies, it would be interesting to try out the performance of the YOLO object detection model when using the exact bounding boxes of the training images and pretrained weights (on ImageNet). (Note that this work originally wanted to use the full implementation of the YOLO but there were some environment problems with Windows operating system.)

Furthermore, the training of the model could be improved by adding stopping rules (such as Dropout) and using k-fold cross-validation to compare and select the best performing model. In the cross-validation the training data is partitioned into k equal parts. Then the model is trained with $k-1$ parts and validated with the part which has not been used for training. This is carried on until every part of the split has been used for validation. The intention is to validate the model's ability to predict the unseen data and to avoid problems such as overfitting and high biases. [7]

REFERENCES

[1] Kaggle. iWildCam 2019 - FGVC6, Categorize animals in the wild. 2019. [Online] Available: https://www.kaggle.com/c/iwildcam-2019-fgvc6

[2] EliteDataScience. How to Handle Imbalanced Classes in Machine Learning. July, 2017. [Online] Available: https://elitedatascience.com/imbalanced-classes

[3] Farhadi, Ali. YOLO: Real-Time Object Detection. 2018. [Online] Available: https://pjreddie.com/darknet/yolo/

[4] Redmon, Joseph and Farhadi, Ali. YOLOv3: An Incremental Improvement. arXiv, 2018. [Online] Available: https://pjreddie.com/media/files/papers/YOLOv3.pdf

[5] xhlulu. CNN Baseline - iWildCam 2019. April 2019. [Online] Available: https://www.kaggle.com/xhlulu/cnn-baseline-iwildcam-2019

[6] Plotka, Szymon. CIFAR-10 CLASSIFICATION USING KERAS TUTORIAL. August, 2018. [Online] Available: https://ermlab.com/en/blog/nlp/cifar-10-classification-using-keras-tutorial/

[7] Wikipedia. Cross-validation (statistics). 2019. [Online] Available: https://en.wikipedia.org/wiki/Cross-validation_(statistics)