CS 484 Project Report

Sait Aktürk
21501734
sait.akturk@ug.bilkent.edu.tr

ChulHyeong Lee
21500003
chulhyeong.lee@ug.bilkent.edu.tr

Ali Altaf Salemwala
21500413
altaf.salemwala@ug.bilkent.edu.tr

I. INTRODUCTION

This project entailed using a ResNet50 implementation to extract features which would be used to classify images and detect locations for certain objects within the images.

II. APPROACH AND IMPLEMENTATION STRATEGIES

We used an approach similar to what was in the assignment, but at several parts, we chose a different path which we felt was more suitable for us.

To obtain the images, we first saved the paths for both the training and the testing sets, and then sequentially accessed these paths to load each image.

After loading the images, we resized, padded, and normalized them so that they were all squares of 224x224 pixels, and were suitable for use by the ResNet50 model of the Keras package. We first take the ratio of the larger of the two dimensions of the original image, and then divide both sides of the image with it, thus conserving the original aspect ratio. After resizing, we pad the image by centering it in a 224x224x3 array, and then adding zeros to each end not occupied by the image values. Normalization was done by subtracting 103.939, 116.779, 123.68 from the Blue, Green, and Red channels respectively. These values were provided in the Keras documentation.

We chose the Keras library because we were familiar with it and knew it would provide us more freedom for choosing pooling options as compared to other libraries.

We then loaded the model and used it to predict the features for each image in each class in the training set. To use the images with the model, we increase their dimensionality to four. The model we obtain is from the Keras library, with **inlude_top** kept as 'false' and **pooling** as 'average'. It provides a 1x2048 feature vector for each image. The **include_top** flag prevents the model from providing classification results, and the **pooling** settings provide the the correct dimensions for the feature vector and use a **global averaging layer**, the same as was used in the resnet50 architecture. Selecting 'max' gave poor SVM accuracy results while 'none' gave a very high dimensionality.

We used the scikit-learn implementation of SVM, and experimented with the 'rbf', 'linear', and 'poly' kernel options. Since the 'rbf' kernel performed the best, we selected it. We suspect this is due to the high dimensionality of the feature space, which is used in the kernel trick by 'rbf'. Other hyperparameters were automatically set by the library itself. We made 10 different SVMs with 10 sets of labels and features; one for each class.

We used Python for the entire previous code implementation, and so continued to use it to detect edge boxes. We used the implementation from the opencv-python library. Although we tried initially with 50 edge boxes, we later saw that 30 edge boxes provide better results, and so stuck with that. The edge boxes for each object type are calculated for each image and then provided to the SVM, where the one with the highest score is selected.

After calculating the confusion matrix, we calculate the total numbers of true positives, false positives, true negatives, and false negatives. To find the precision, we divide the number of true positives by the number of true and false positives, and to find the recall, we divide the number of true positives by the number of true positives and false negatives. To calculate localization results, we calculate the intercepting and total areas between our obtained edge boxes, and the edge boxes provided; the ratio of the intercepting area to the total area is the localization accuracy. Our code is directed to maximize classification accuracy, and as a result localization accuracy suffers.

III. RESULTS

A. EDGE BOX RESULTS

The following ten figures show the 30 candidate windows for one image from each class. The windows are overlaid on the images with varying colors. The classes are shown in order: n02317335, n02504458, n02391049, n02099601, n02123159, n02422699, n02481823, n01615121, n02410509, n02129604.

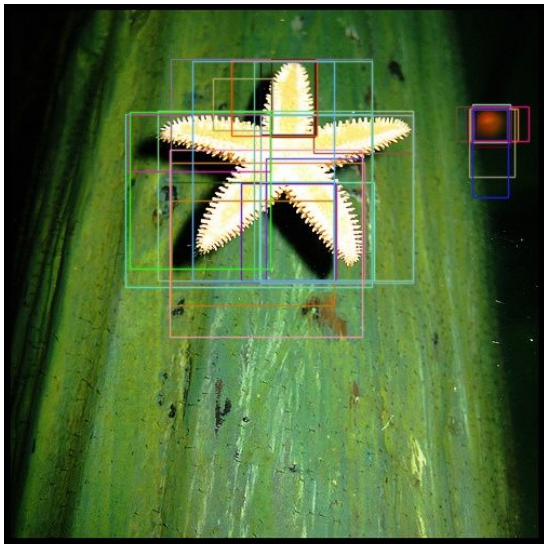


Fig. 1. Edge boxes for the n02317335 class

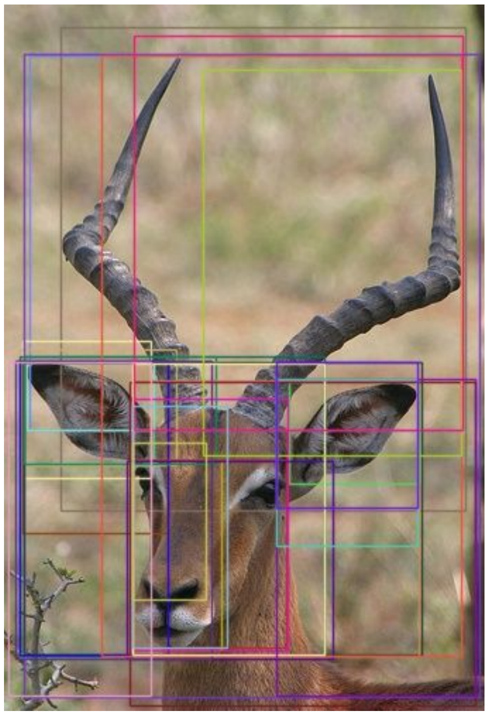


Fig. 2. Edge boxes for the n02422699 class

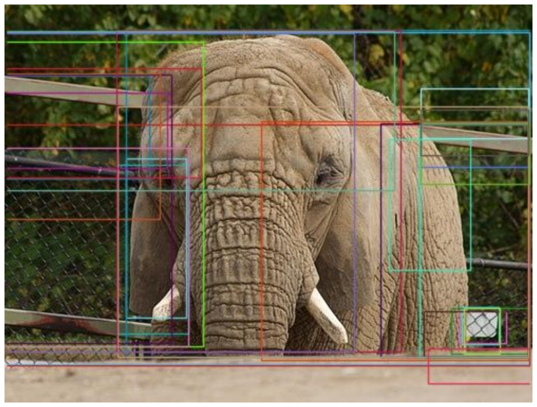


Fig. 3. Edge boxes for the n02504458 class

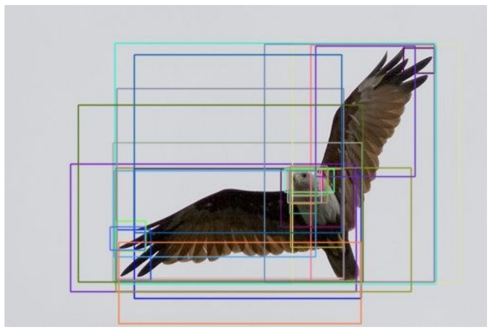


Fig. 4. Edge boxes for the n01615121 class

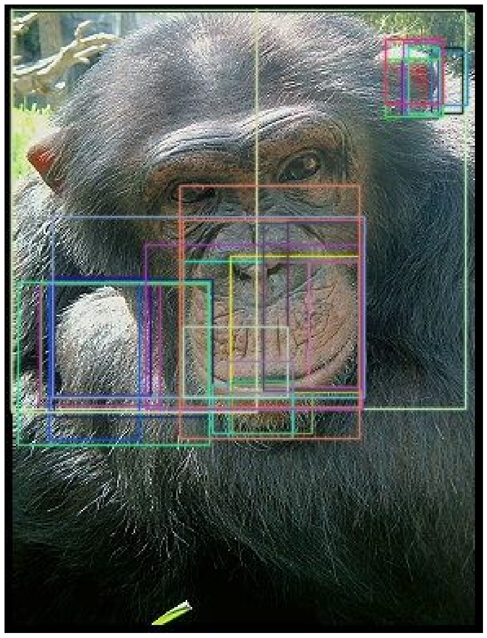


Fig. 5. Edge boxes for the n02481823 class

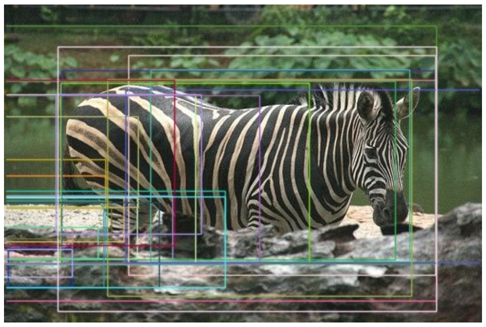


Fig. 6. Edge boxes for the n02391049 class

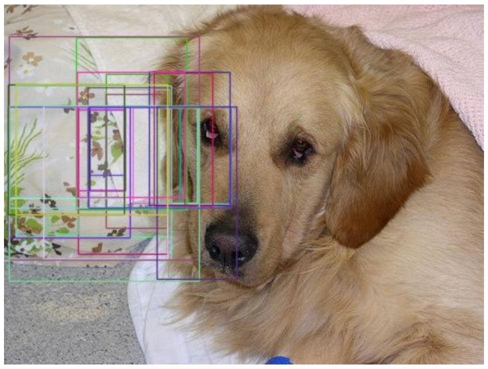


Fig. 7. Edge boxes for the n02099601 class

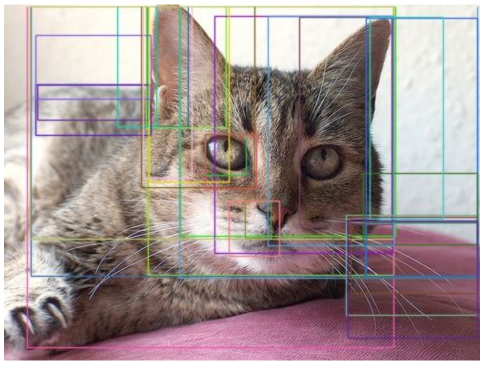


Fig. 8. Edge boxes for the n02123159 class

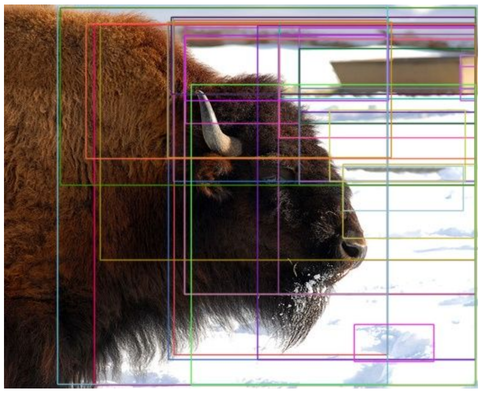


Fig. 9. Edge boxes for the n02410509 class

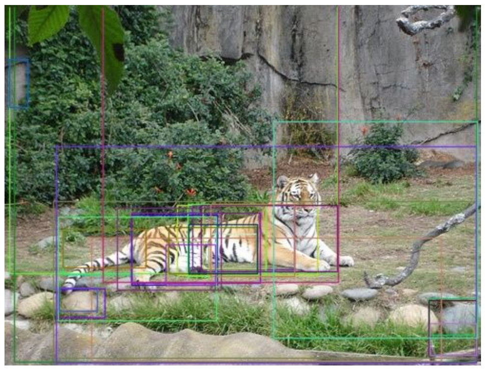


Fig. 10. Edge boxes for the n02129604 class

B. LOCALIZATION RESULTS

The following 20 figures show the localization results, with two results for each class. Each image has two boxes; the blue one shows the supplied boundary box, while the red one shows the obtained boundary box. One image is a correct result, one is an incorrect result, following the criteria that a correct result has an overlap ratio exceeding 50%.(Blue boxes are provided and red boxes are our findings.*)*

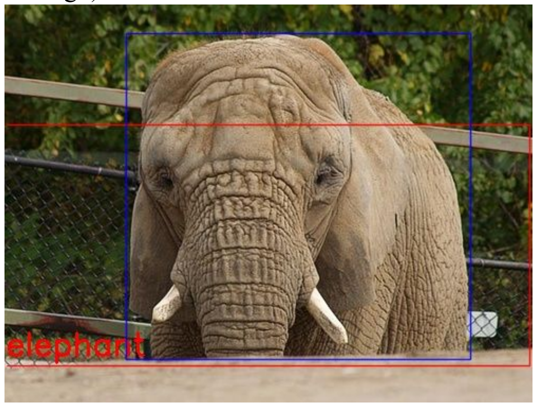


Fig. 11. Incorrect boundary box for the n02504458 class

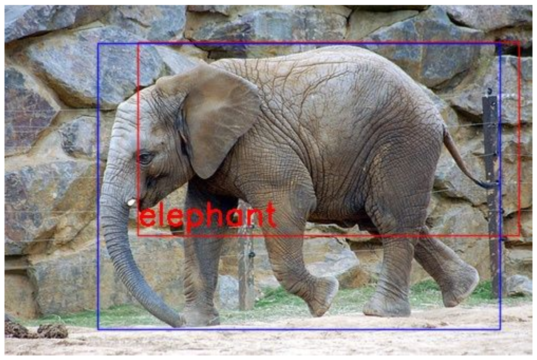


Fig. 12. Correct boundary box for the n02504458 class

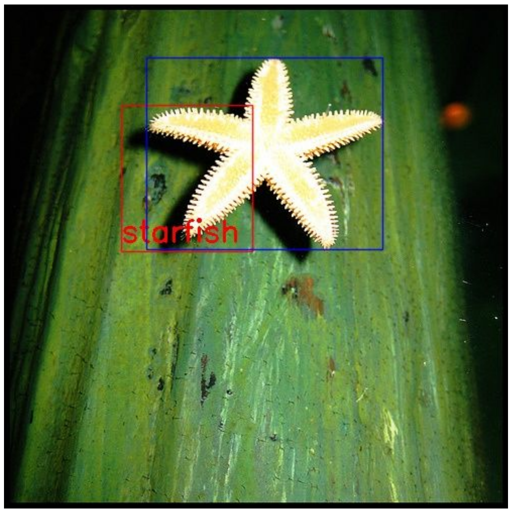


Fig. 13. Incorrect boundary box for the n02317335 class

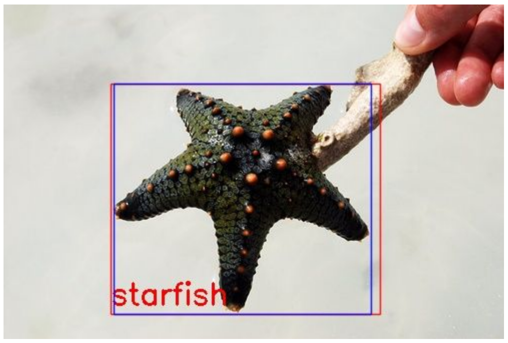


Fig. 14. Correct boundary box for the n02317335 class

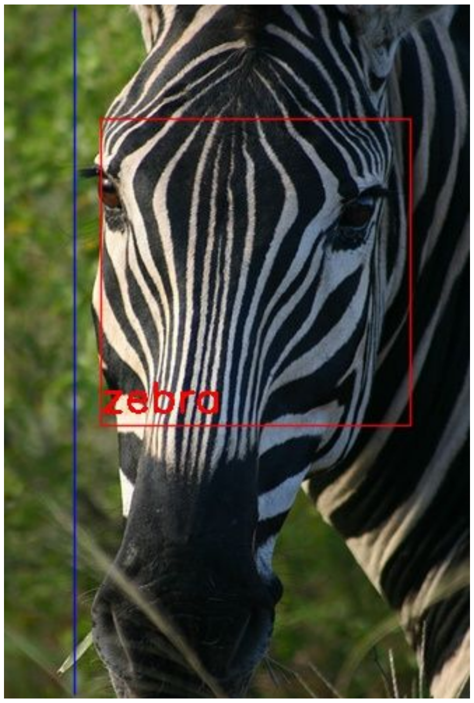


Fig. 15. Incorrect boundary box for the n02391049 class

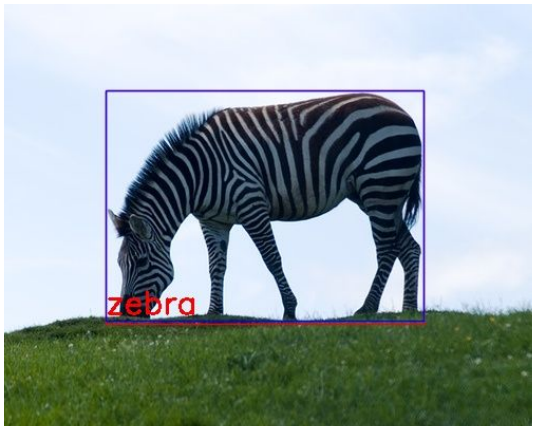


Fig. 16. Correct boundary box for the n02391049 class

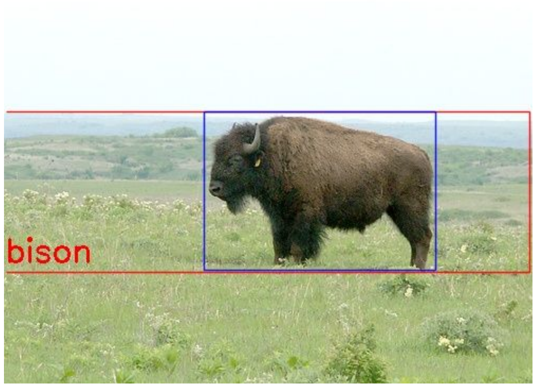


Fig. 17. Incorrect boundary box for the n02410509 class

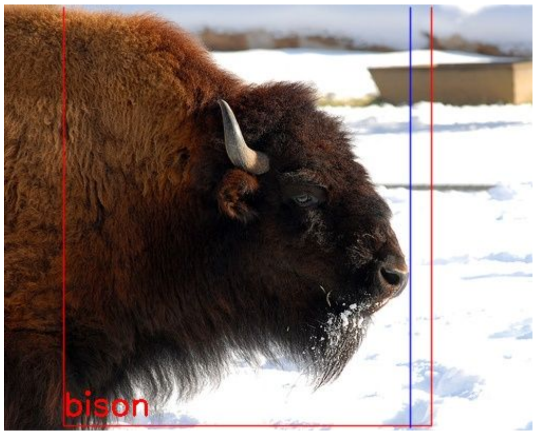


Fig. 18. Correct boundary box for the n02410509 class

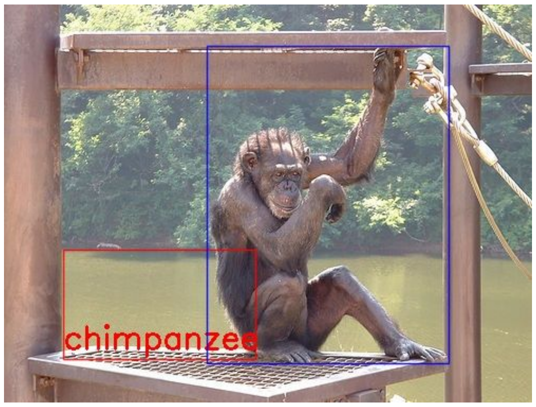


Fig. 19. Incorrect boundary box for the n02481823 class

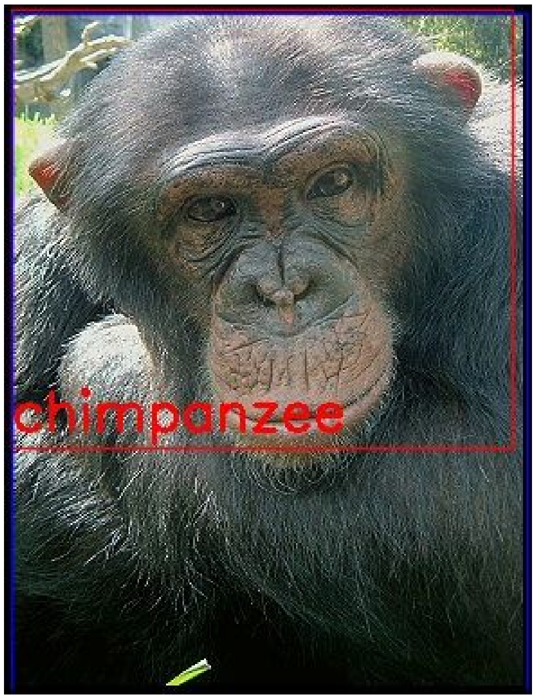


Fig. 20. Correct boundary box for the n02481823 class

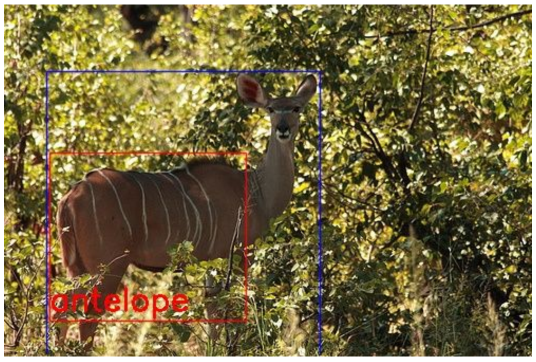


Fig. 21. Incorrect boundary box for the n02422699 class

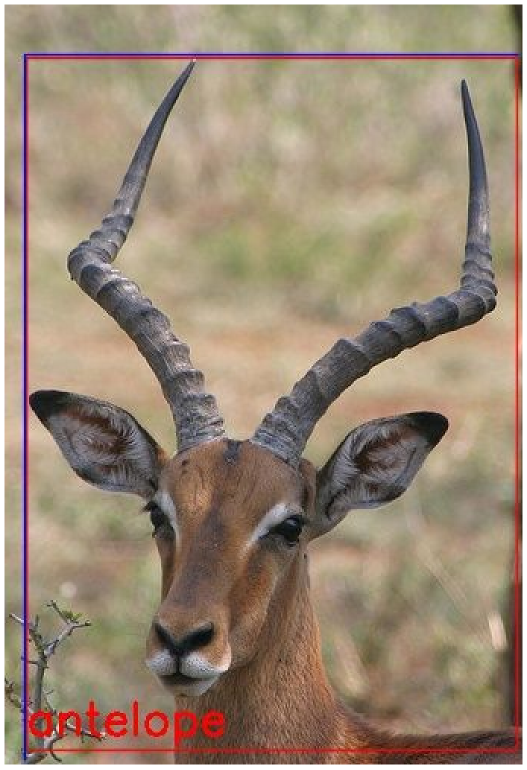


Fig. 22. Correct boundary box for the n02422699 class

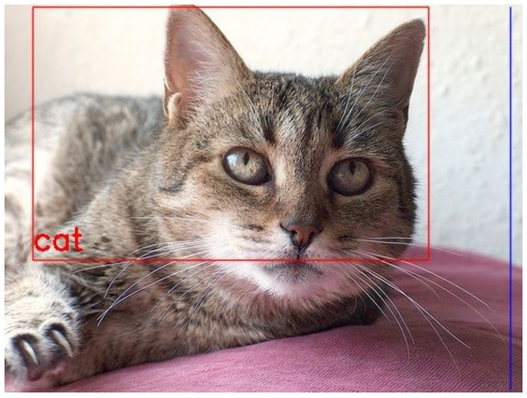


Fig. 23. Incorrect boundary box for the n02123159 class

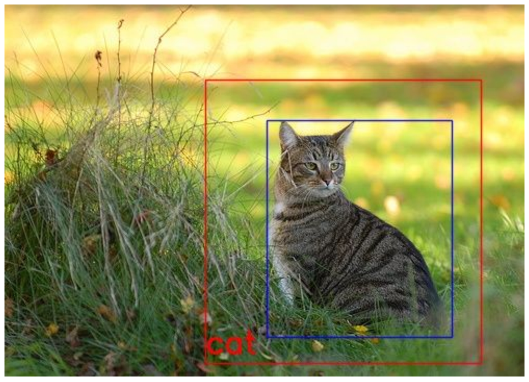


Fig. 24. Correct boundary box for the n02123159 class

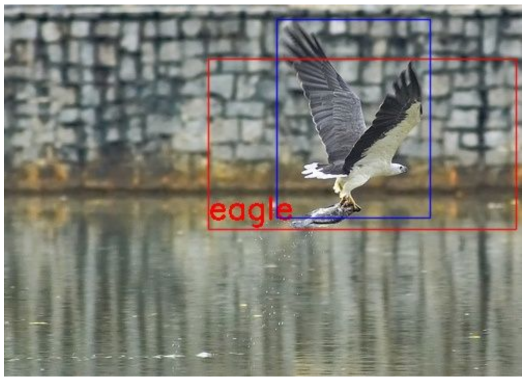


Fig. 25. Incorrect boundary box for the n01615121 class

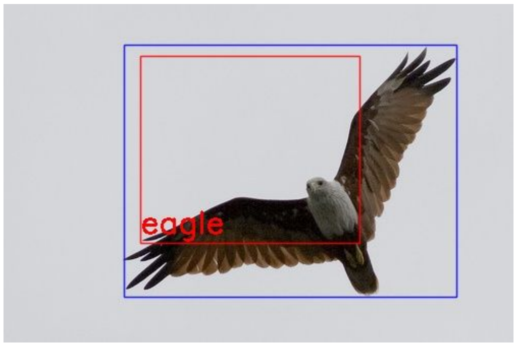


Fig. 26. Correct boundary box for the n01615121 class

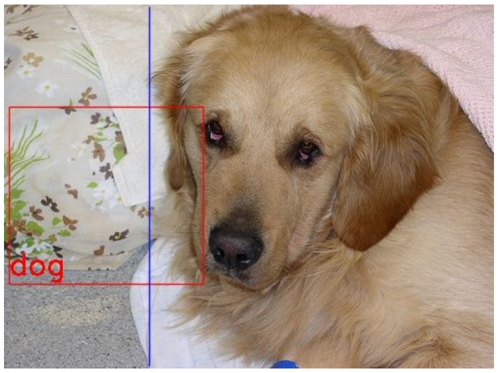


Fig. 27. Incorrect boundary box for the n02099601 class

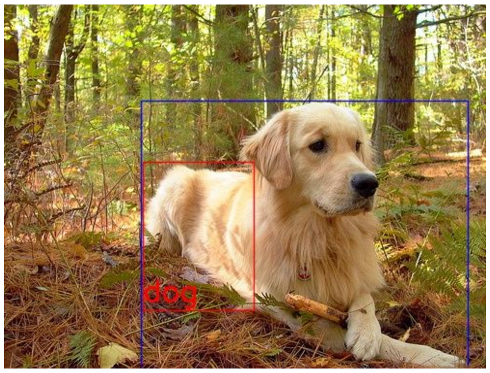


Fig. 28. Incorrect boundary box for the n02099601 class

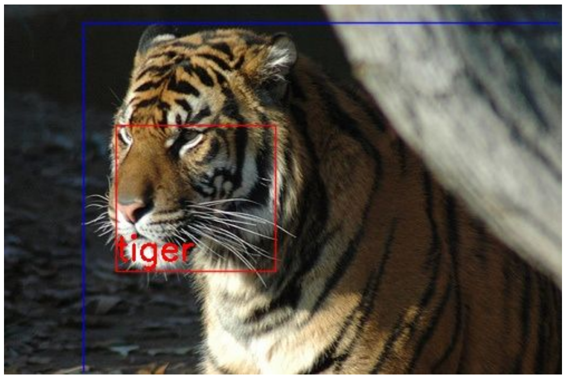


Fig. 29. Incorrect boundary box for the n02129604 class

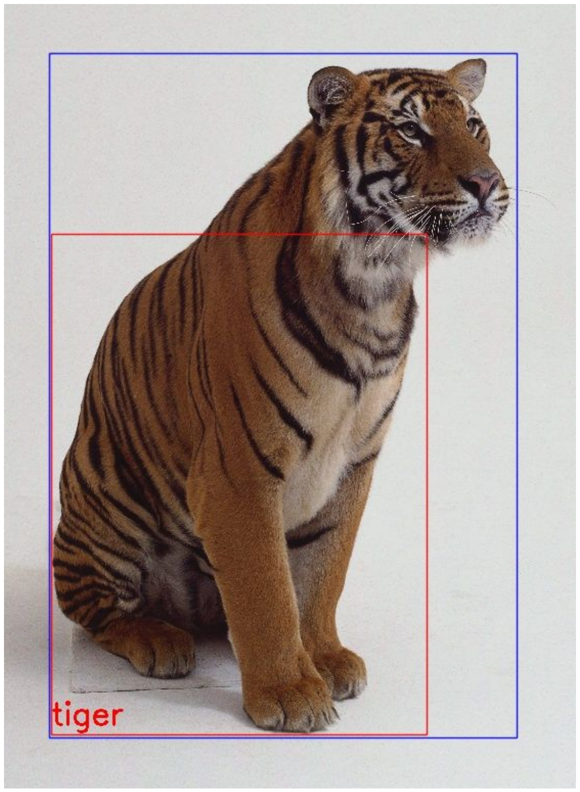


Fig. 30. Correct boundary box for the n02129604 class

IV. OBSERVATIONS

We observe here that some object types perform better than others . Starfish ,bison and eagle for example , have better localizations than other classes , whereas images with dogs perform much worse.

The observed classification accuracy was 95%, while the localization accuracy was 48%.

V. PERFORMANCE METRICS

Fig. 31 below shows the confusion matrix for each class.

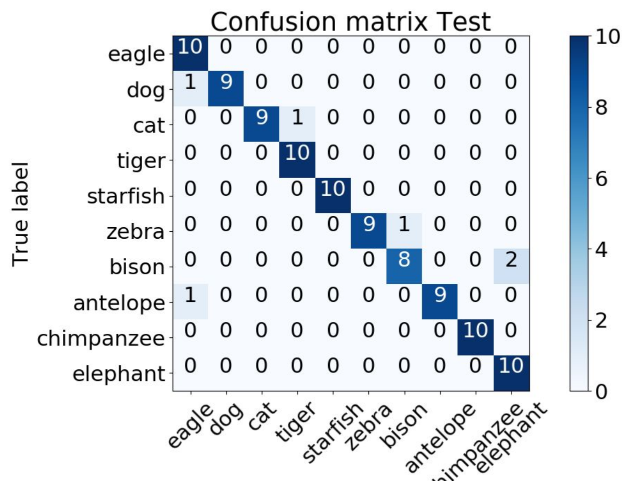


Fig. 31. Confusion matrix for each class

Figure 32 [1] shows the precision, recall, and localization accuracy for each object type.

VI. DISCUSSION

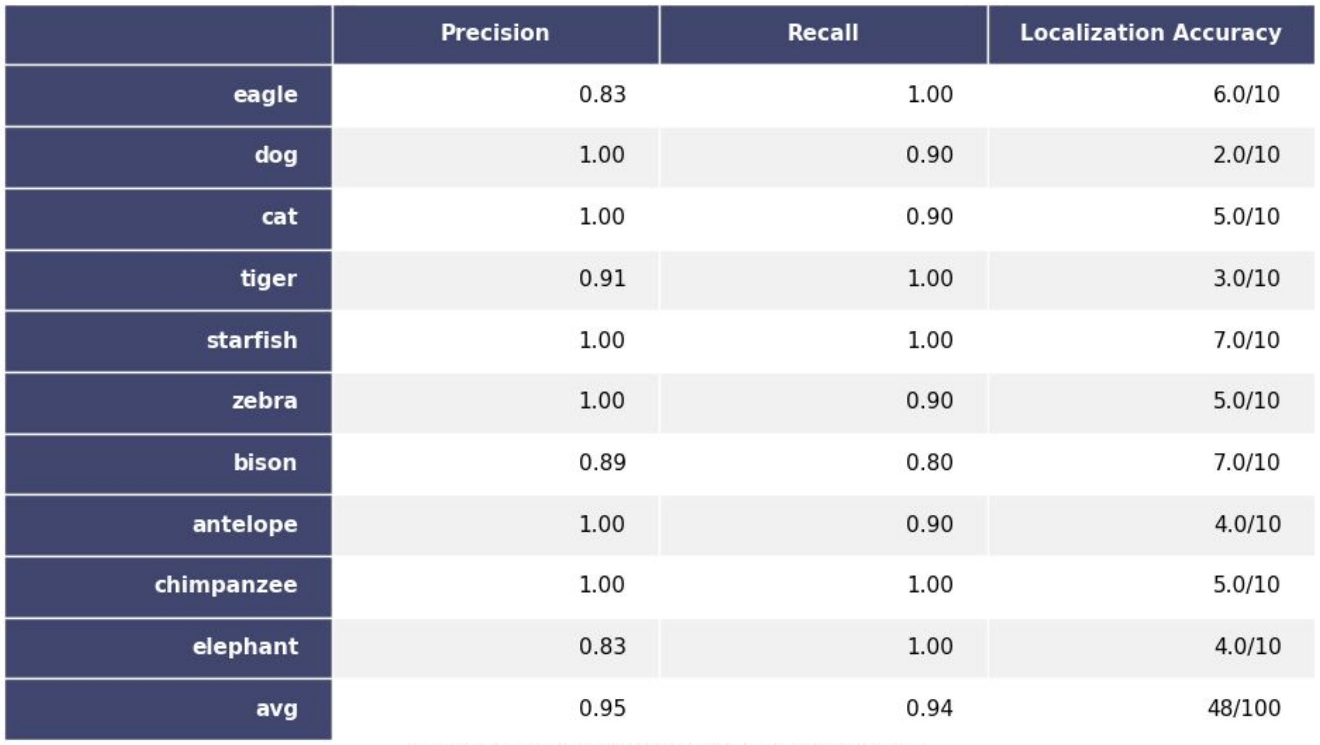
A. Performance Analysis

There were not many parameters available here subject to change.

The ResNet50 and SVM implementations automated most of the hyperparameter selections so we were not able to experiment with those.

For the ResNet50 model, we did select **include_top** and **pooling** parameters according to what the documentation said about them. Since giving them different settings would not work with our implementation, we did not try them.

For the SVM classifier, we selected the 'rbf' **kernel** option because it had the best accuracy out of all the options. We



	Precision	Recall	Localization Accuracy
eagle	0.83	1.00	6.0/10
dog	1.00	0.90	2.0/10
cat	1.00	0.90	5.0/10
tiger	0.91	1.00	3.0/10
starfish	1.00	1.00	7.0/10
zebra	1.00	0.90	5.0/10
bison	0.89	0.80	7.0/10
antelope	1.00	0.90	4.0/10
chimpanzee	1.00	1.00	5.0/10
elephant	0.83	1.00	4.0/10
avg	0.95	0.94	48/100

Fig 32. Precision, Recall, and Localization Accuracy by class

had already suspected this due to our high dimensionality

and 'rbf' using the kernel trick, and our guess was proven correct.

We opted to maximize classification accuracy at the expense of localization since we theorized that the bounding boxes were man-made and should not be relied on. Additionally, since the images will be classified into groups, we thought it better to make the classifications as accurate as possible.

We experimented with the number of edge boxes and found 30 boxes provided the much better accuracy than 50 or 100 . Since 30 boxes provided the better results and also performing significantly faster, we chose 30 edge boxes.

B. Results based on Object Types

The convolutional neural network implemented in our code detects not the objects themselves but some of the features of the objects to identify patterns/ traits of the animals shown in the pictures such as their skin, ear, nose, facial structures, and shapes, or their fur or horns. The CNN can focus on the small but specific portions of the pictures that characterize the general features of the objects and easily identify with the information. Such portions may include skin textures or patterns. In our case, tigers and zebras' stripes, and elephants' skin textures provided such information and the simulation resulted in higher classification accuracy for these animals. As a trade-off of high accuracy, the localization accuracy was lower.

However, objects with rather distinct characteristics for each individual, or with traits that aren't easy to generalize performed in an opposite manner. It included the hair colors of cats, the color of starfishes , bison , and feather colors of eagles. These objects resulted in higher localization accuracy and lower in classification.

C. Suggestions for Improvement

The performance is hampered by some of the constraints in the project.

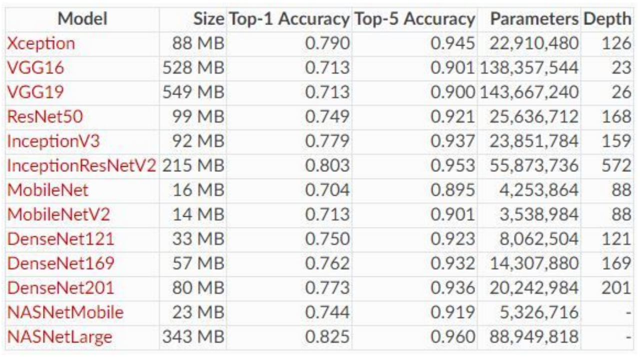
The small dataset does not give enough training instances for fully accurate predictions. Increasing the size of the training set would allow for further training of the model and result in a feature set which would deliver more accurate results.

In addition to using a larger training set, we could increase the utility of the current one by using data augmentation. Flipping, rotating, shearing, zooming, translating, altering the brightness, or even adding noise to the images would artificially create a larger dataset which would allow for more generalization of the object type and its features, leading to more accurate classifications.

Rather than using our own classification pipeline, we could use other pre-implemented pipelines like Faster R-CNN. These were made by the creators of our algorithms and thus might perform better.

Rather than prioritizing only a high classification accuracy, we could try to strike a balance between it and the localization accuracy, thus getting a better overall accuracy/

The ResNet50 model is good and provides a favorable balance between size, time, and accuracy. Other models, such as IneptionResNetV2, however, provide a better top-1 accuracy, usually at the expense of either size or time. Figure 33 provides statistics for some other models in the Keras library. According to its data, we would suggest using the Xception model for better results and a slightly lower size.



Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	99 MB	0.749	0.921	25,636,712	168
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

Fig. 33. Documentation of the performances of various algorithms [1]

VII. CONTRIBUTIONS

Each part of the project was done with all teammates together, so tasks were not individually assigned as much as they were done in conjunction. Sait had more knowledge of the Keras and scikit-learn libraries so he guided the direction of the code written and image preprocessing and also testing overall pipeline. Chulhyeong contributed in edge box forming , precision and recall part . Ali managed data loading , SVM, and created and edited most of the report.

VIII. REFERENCES

[1] "Applications - Keras Documentation", Keras.io, 2019. [Online]. Available: https://keras.io/applications/. [Accessed: 08- Jan- 2019].