CS 484, Fall 2019
Term Project: Object Localization and Recognition

Barış Can - 21501886 Faruk Oruç - 21601844

I. Introduction

In this project, there are 498 images (398 for the train and 100 for the test) from different classes such as starfish, bison, dog, etc. While the background of the train images is eliminated by default, the background of the test images is cropped during the pre-processing step. There are two aims of this project, localization and recognition. Localization is only applied on test images, whereas recognition is applied both train images and test images. Feature extraction for both training and testing is done using a pre-trained deep network. There are several steps of this project, data normalization to get better results and use them in PyTorch library, feature extraction from the pre-trained deep network by using the normalized images, training the system, extracting the candidate windows for the localization step, classification and localization, and evaluation. Details of these step will be explained in the Details of the Approach and Implementation Strategies sections.

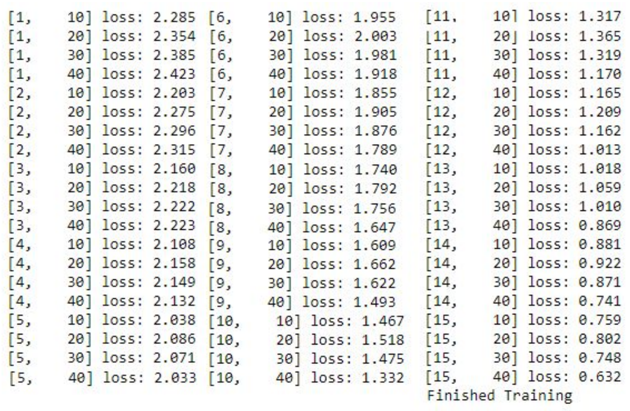
II. Details of the Approach

Since all the images have different aspect ratios and sizes, the first step was pre-processing. All images were converted to RGB since some images may have different channels, resized to 224x224 and applies some normalization in order to use them in PyTorch feature extraction methods. Then feature extraction was done for the training by using pre-trained ResNet-50 weights. Since the feature vectors will be used in the 2-layer feed-forward neural network classifier later, the last layer of the ResNet-50 model was removed. ReLU function was used in the 2-layer feed-forward neural network classifier since ReLU function reduces the likelihood of the gradient to vanish and more efficient than the other activation functions. After several experiments, 512 was used as a hidden size value since it is in the middle between 2048 ResNet-50 feature vectors and 10 output class size. Cross-Entropy Loss was used as loss function since it minimizes the distance between two probability distributions, predicted and actual classes and it is good for categorical classification. In addition, Stochastic Gradient Descent (SGD) algorithm was used as the optimization method. 0.001 was used as the learning rate after some experiments, although it increases the running time of the training step, it gave the best results for the model. After finishing the training, candidate windows were extracted using Selective Search region proposal algorithm to calculate the localization accuracy on the test samples [1]. Then, the same pre-processing with the training data is applied to each candidate window. Since Selective Search algorithm gives many candidate windows, some thresholds were experimented. Further discussion on the threshold values can be found in the Discussion section. Moreover, the feature vector of each candidate windows is extracted as training samples and the results were evaluated by putting the feature vectors to the same 2-layer feed-forward neural network classifier. Later, the obtained results were evaluated and confusion matrices and tables were created.

III. Implementation Strategies

We started the project with pre-processing the provided images since all of them had different aspect ratios and sizes. We used glob library to access all elements in the folders and directory, Python Imaging Library (PIL) to load our images and convert them to Numpy arrays in order to apply the pre-processing [2][3][4]. As mentioned before, the images were converted to 224x224 and they were converted to RGB colorspace as some of them might have had alpha channels. Since the conversion to the size of 224x224 would be problematic for some of the images due to their mismatch in their aspect ratio, the images were padded with zeros (black spaces) regarding their wider aspect (height/width). This process ensured that every image can be resized without disturbing the shape of the object of interest within them. Then, from every image were subtracted with 0.485, 0.456, 0.406 and divided by 0.229, 0.224, 0.225 from red, green and blue channels in order to make the images suitable to be used with PyTorch library [5]. Lastly, we converted each image to float32 type to be able to use them in training.

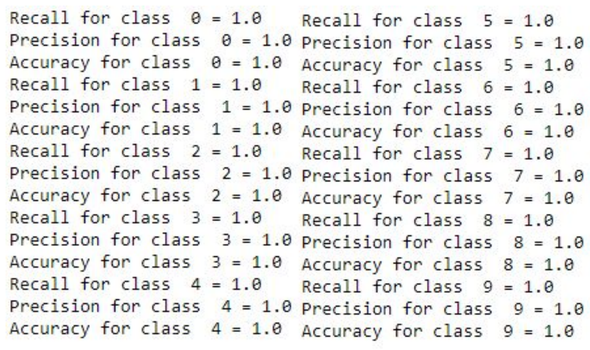
We used torchvision.models library to access the Pytorch's pre-trained models. We used PyTorch's ResNet50 model and removed the last layer of it to use it with our 2-layer feed-forward neural network classifier [6]. We used several normalizations during the extraction step by using Numpy. We first reshape each image to 1x224x224x3 format (we decided 1 as our batch size in this step), then took the transpose of it and finally convert our Numpy image arrays into torch tensor. After the normalization for the feature vectors, we extracted the vectors, converted them Numpy arrays and appended them to a list. We also created a list that includes the labels of the images we took from the folder's names in order to use them as target values in the Dataloader. After extracting the feature vectors, we created our 2-layer feed-forward neural network classifier using PyTorch's nn library [7]. We used ReLU function as our activation function and 512 as our hidden size value as mentioned in the Details of the Approach section. We also used Cross-Entropy Loss as our loss function and Stochastic Gradient Descent (SGD) using torch.optim library for the reasons mentioned in the Details of the Approach section [8]. We then squeeze our feature vectors because the output of the model was 1x2048x1x1 and we needed just 2048. After this conversion, we converted our input, feature vectors, and target, labels, to tensors in order to create a Dataset. The shapes of the inputs were [398, 2048] and the shape of the target was [398]. We used torch.utils to create the Dataset and Dataloader variables [9]. We also converted the labels to long type because of the Dataset's input type. We chose 10 as our batch size since it increases the speed of the process. After creating the Dataloader, we continued with the training step. We chose 15 as our epoch number as mentioned in the Discussion section. Change of localization and classification accuracy with different epoch numbers can be seen in Appendix Figure 28, Table 1. Then, for the training, we separated our data and labels from the Dataset, used the optimizer, and pass the feature vectors to 2-layer feed-forward neural network classifier. We got 10 outputs from the classifier, put the outputs into the loss function, and printed our loss rates. The loss rates for our training can be found below (Figure 1).



```
[1,    10] loss: 2.285 [6,    10] loss: 1.955   [11,    10] loss: 1.317
[1,    20] loss: 2.354 [6,    20] loss: 2.003   [11,    20] loss: 1.365
[1,    30] loss: 2.385 [6,    30] loss: 1.981   [11,    30] loss: 1.319
[1,    40] loss: 2.423 [6,    40] loss: 1.918   [11,    40] loss: 1.170
[2,    10] loss: 2.203 [7,    10] loss: 1.855   [12,    10] loss: 1.165
[2,    20] loss: 2.275 [7,    20] loss: 1.905   [12,    20] loss: 1.209
[2,    30] loss: 2.296 [7,    30] loss: 1.876   [12,    30] loss: 1.162
[2,    40] loss: 2.315 [7,    40] loss: 1.789   [12,    40] loss: 1.013
[3,    10] loss: 2.160 [8,    10] loss: 1.740   [13,    10] loss: 1.018
[3,    20] loss: 2.218 [8,    20] loss: 1.792   [13,    20] loss: 1.059
[3,    30] loss: 2.222 [8,    30] loss: 1.756   [13,    30] loss: 1.010
[3,    40] loss: 2.223 [8,    40] loss: 1.647   [13,    40] loss: 0.869
[4,    10] loss: 2.108 [9,    10] loss: 1.609   [14,    10] loss: 0.881
[4,    20] loss: 2.158 [9,    20] loss: 1.662   [14,    20] loss: 0.922
[4,    30] loss: 2.149 [9,    30] loss: 1.622   [14,    30] loss: 0.871
[4,    40] loss: 2.132 [9,    40] loss: 1.493   [14,    40] loss: 0.741
[5,    10] loss: 2.038 [10,    10] loss: 1.467  [15,    10] loss: 0.759
[5,    20] loss: 2.086 [10,    20] loss: 1.518  [15,    20] loss: 0.802
[5,    30] loss: 2.071 [10,    30] loss: 1.475  [15,    30] loss: 0.748
[5,    40] loss: 2.033 [10,    40] loss: 1.332  [15,    40] loss: 0.632
                                                Finished Training
```

Fig. 1. Loss Rates of the Training.

After the training is done, we went on with the Selective Search algorithm and extracting the candidate windows. We extracted the test images as we did with the training samples, and we applied the Selective Search algorithm for each image [1]. Results of the Selective Search algorithm with different classes can be found in Figures from 8 to 17. Since the Selective Search algorithm gives more than 1000 candidate windows, we decided to put a threshold to the windows. If the size of the candidate windows is less than 5% size of the actual image, we removed that candidate windows since it is too small to include an object in it. For example for an image that has the sizes 500x500, we removed the candidate windows that have sizes less than 111.8x111.8. In this way, we observed that even if we decrease the accuracy of some images that includes some small object, we increased the overall localization accuracy as mentioned in the Discussion section. Change of localization and classification accuracy with different thresholds also can be seen in Appendix Figure 28, Table 1. After extracting the candidate windows, we cropped the windows from the original image, we pasted the image to a black background to make it padded and applied the same pre-processing with the training samples. Extracting the feature vectors and creating the Dataset and Dataloader was the same with the training, however, since there are candidate windows in this step, the shapes were[100, # of candidate windows, 2048] instead of [398, 2048]. Therefore we changed some parts of the methods, but the algorithms were the same. Lastly, we evaluated our results by calculating the classification and localization accuracy. For the classification accuracy, we compared the actual classes of the images with our predictions and we summed the number of correctly labelled images. For the localization accuracy, we calculated true-positives, true-negatives, false-positives, false-negatives and created confusion matrices that can be found from Figure 3 to Figure 7 for each class. We also calculated precision and recall values for each class which is 1.0 for every class which can be seen below.



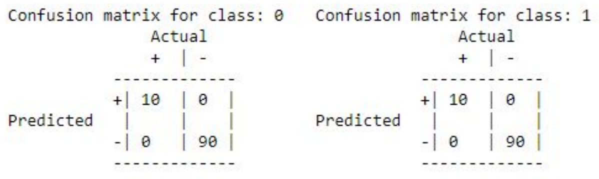
```
Recall for class  0 = 1.0       Recall for class  5 = 1.0
Precision for class  0 = 1.0    Precision for class  5 = 1.0
Accuracy for class  0 = 1.0     Accuracy for class  5 = 1.0
Recall for class  1 = 1.0       Recall for class  6 = 1.0
Precision for class  1 = 1.0    Precision for class  6 = 1.0
Accuracy for class  1 = 1.0     Accuracy for class  6 = 1.0
Recall for class  2 = 1.0       Recall for class  7 = 1.0
Precision for class  2 = 1.0    Precision for class  7 = 1.0
Accuracy for class  2 = 1.0     Accuracy for class  7 = 1.0
Recall for class  3 = 1.0       Recall for class  8 = 1.0
Precision for class  3 = 1.0    Precision for class  8 = 1.0
Accuracy for class  3 = 1.0     Accuracy for class  8 = 1.0
Recall for class  4 = 1.0       Recall for class  9 = 1.0
Precision for class  4 = 1.0    Precision for class  9 = 1.0
Accuracy for class  4 = 1.0     Accuracy for class  9 = 1.0
```

Fig. 2. Recall, Precision and Accuracy Values for Each Class.

IV. Results Obtained

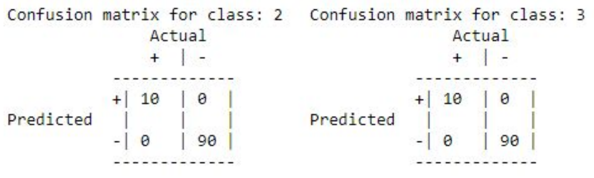
In this section, project results will be shown.

A. Confusion Matrix



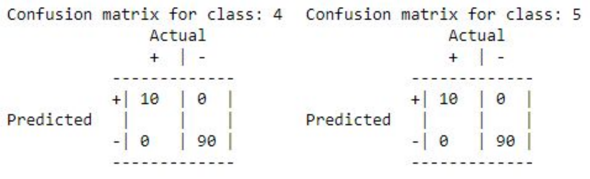
```
Confusion matrix for class: 0   Confusion matrix for class: 1
              Actual                          Actual
             +  | -                          +  | -
          ------------                    ------------
          +| 10  | 0  |                   +| 10  | 0  |
Predicted  |     |    |         Predicted  |     |    |
          -| 0   | 90 |                   -| 0   | 90 |
          ------------                    ------------
```

Fig. 3. Confusion Matrix for Classes 0 and 1.



```
Confusion matrix for class: 2   Confusion matrix for class: 3
              Actual                          Actual
             +  | -                          +  | -
          ------------                    ------------
          +| 10  | 0  |                   +| 10  | 0  |
Predicted  |     |    |         Predicted  |     |    |
          -| 0   | 90 |                   -| 0   | 90 |
          ------------                    ------------
```

Fig. 4. Confusion Matrix for Classes 2 and 3.



```
Confusion matrix for class: 4   Confusion matrix for class: 5
              Actual                          Actual
             +  | -                          +  | -
          ------------                    ------------
          +| 10  | 0  |                   +| 10  | 0  |
Predicted  |     |    |         Predicted  |     |    |
          -| 0   | 90 |                   -| 0   | 90 |
          ------------                    ------------
```

Fig.5. Confusion Matrix for Classes 4 and 5.

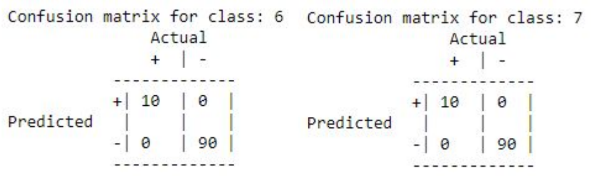

```
Confusion matrix for class: 6    Confusion matrix for class: 7
              Actual                           Actual
              +  | -                           +  | -
           -------------                    -------------
Predicted  +| 10  | 0  |         Predicted  +| 10  | 0  |
            |     |    |                     |     |    |
           -| 0   | 90 |                    -| 0   | 90 |
           -------------                    -------------
```

Fig. 6. Confusion Matrix for Classes 6 and 7.

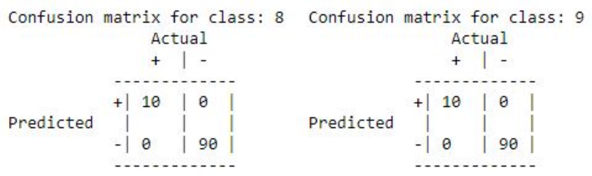

```
Confusion matrix for class: 8    Confusion matrix for class: 9
              Actual                           Actual
              +  | -                           +  | -
           -------------                    -------------
Predicted  +| 10  | 0  |         Predicted  +| 10  | 0  |
            |     |    |                     |     |    |
           -| 0   | 90 |                    -| 0   | 90 |
           -------------                    -------------
```

Fig. 7. Confusion Matrix for Classes 8 and 9.

B. Examples for Candidate Windows

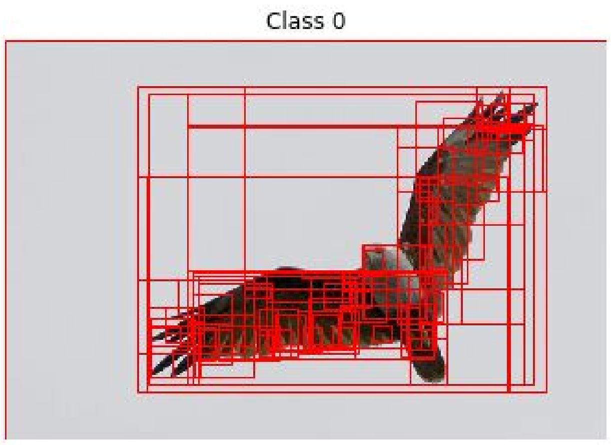


Fig. 8. Candidate Windows for Class 0

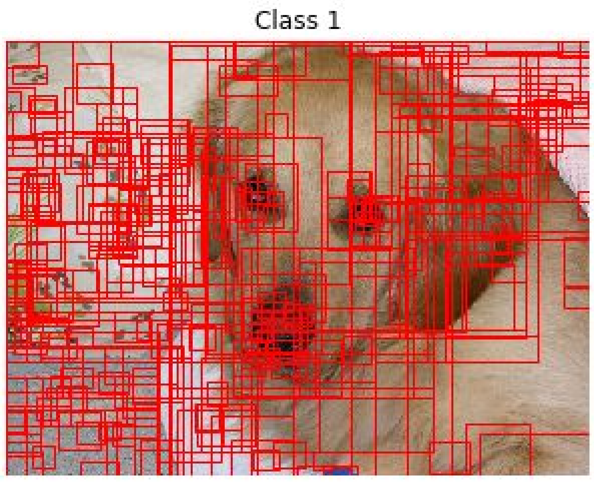


Fig. 9. Candidate Windows for Class 1

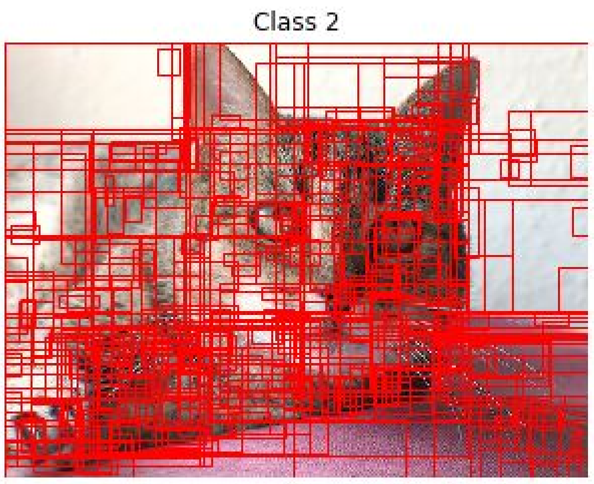


Fig. 10. Candidate Windows for Class 2

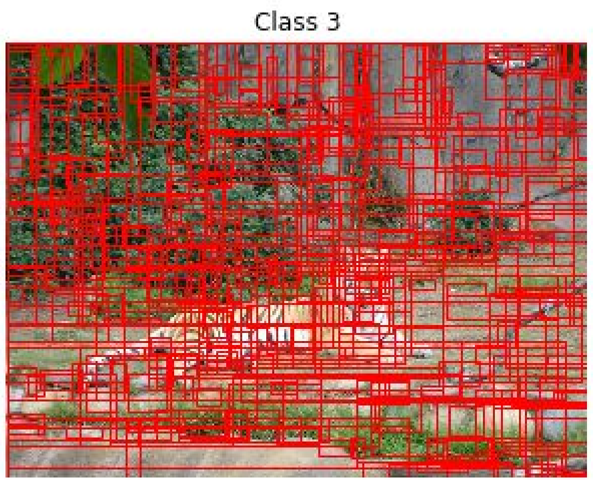


Fig. 11. Candidate Windows for Class 3

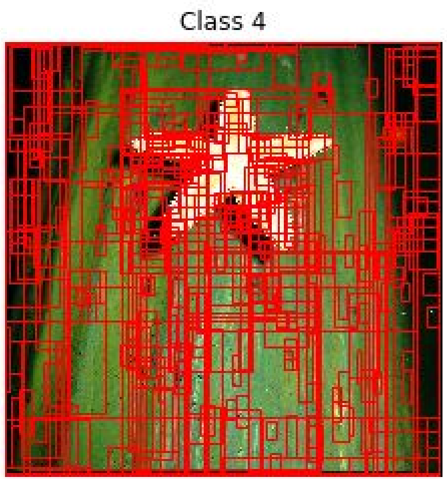


Fig. 12. Candidate Windows for Class 4

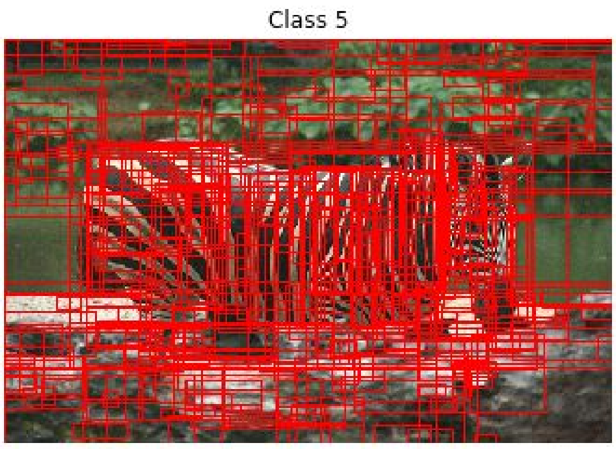


Fig. 13. Candidate Windows for Class 5

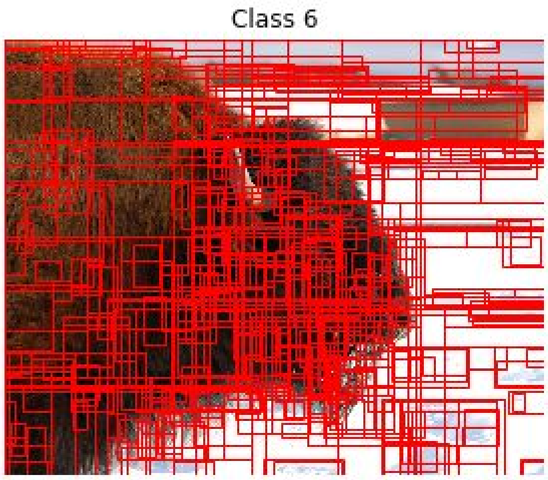


Fig. 14. Candidate Windows for Class 6

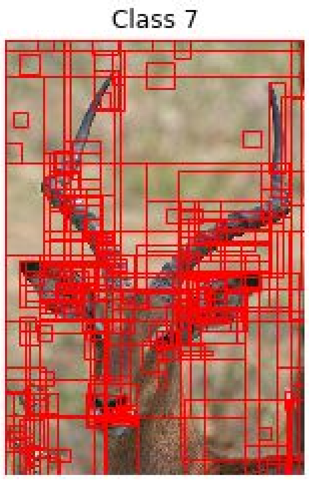


Fig. 15. Candidate Windows for Class 7

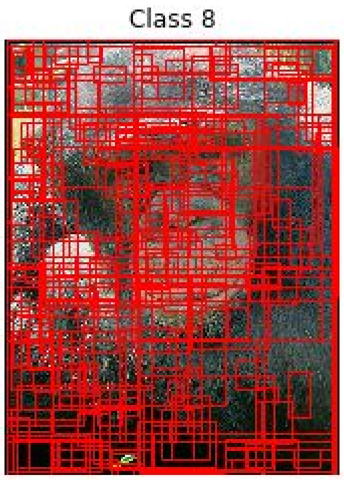


Fig. 16. Candidate Windows for Class 8

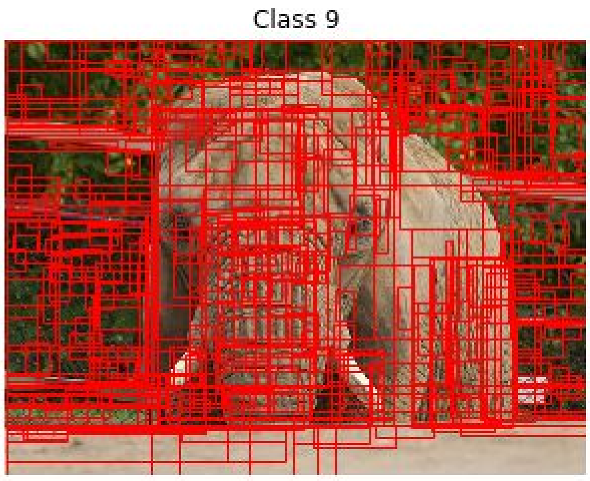


Fig. 17. Candidate Windows for Class 9

C. Localization Results

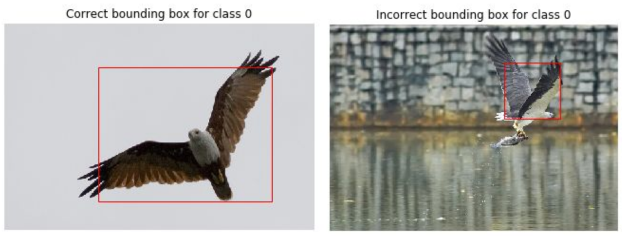

Correct bounding box for class 0
Incorrect bounding box for class 0

Fig. 18. Localization Results for Class 0

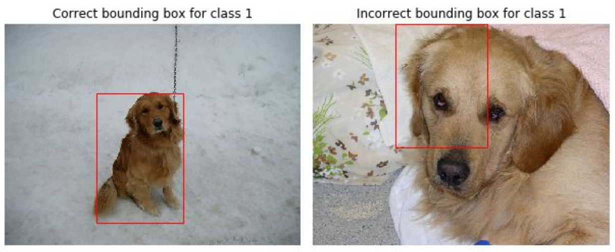

Correct bounding box for class 1
Incorrect bounding box for class 1

Fig. 19. Localization Results for Class 1

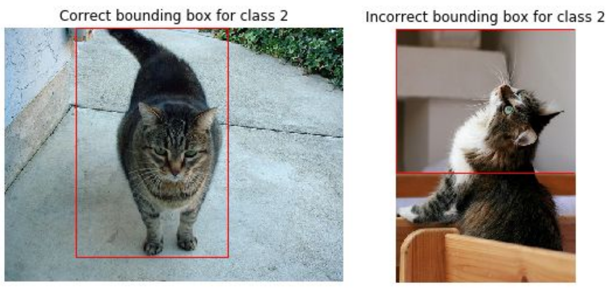

Correct bounding box for class 2
Incorrect bounding box for class 2

Fig. 20. Localization Results for Class 2

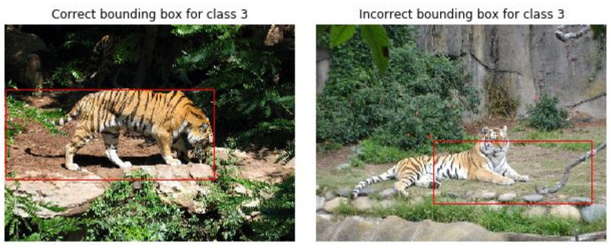

Correct bounding box for class 3
Incorrect bounding box for class 3

Fig. 21. Localization Results for Class 3

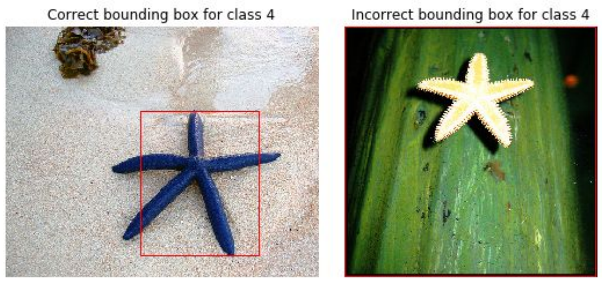

Correct bounding box for class 4
Incorrect bounding box for class 4

Fig. 22. Localization Results for Class 4

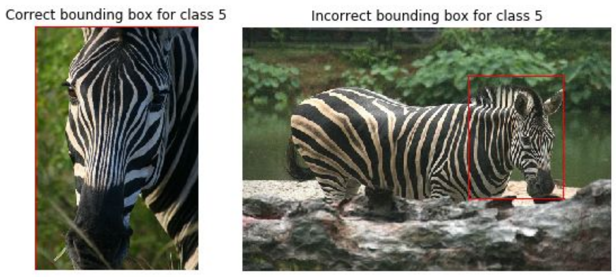

Correct bounding box for class 5
Incorrect bounding box for class 5

Fig. 23. Localization Results for Class 5

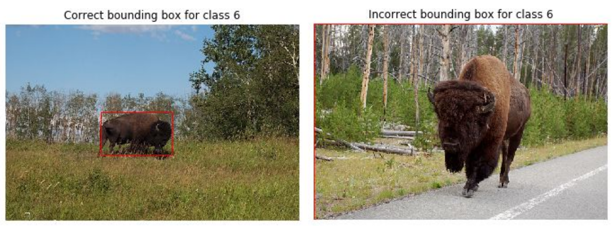

Correct bounding box for class 6
Incorrect bounding box for class 6

Fig. 24. Localization Results for Class 6

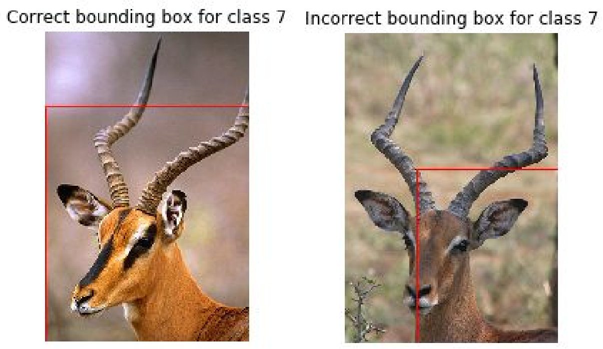

Correct bounding box for class 7
Incorrect bounding box for class 7

Fig. 25. Localization Results for Class 7

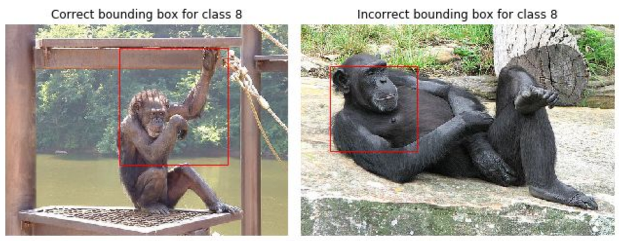

Correct bounding box for class 8
Incorrect bounding box for class 8

Fig. 26. Localization Results for Class 8

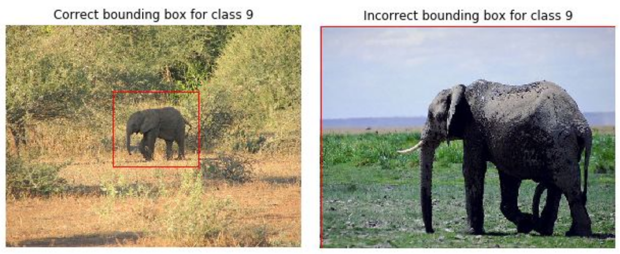

Correct bounding box for class 9
Incorrect bounding box for class 9

Fig. 27. Localization Results for Class 9

V. Discussion

Since there was no problem in the pre-processing step, the implementation was performed as mentioned above without any trial and errors. For the feature extraction step, since we did not know the general principles of pre-trained models beforehand, we first did not remove the last layer of the model, obtained 1000 vectors and worked on them some time. We realized we need to remove the last layer by using nn.Sequential method while searching on the Internet for another error. Moreover, we also did not we need to use .eval function after the deleting the last layer of the model. We were stuck in this step for again some time before realizing we need to use .eval from some research on the Internet. We did not encounter further errors for this step. However, we did not know we need to convert the feature vectors to tensors and use them with Dataset and Dataloader. We first tried to use them in the train without using dataset and Dataloader, but we got several errors about the dimension size or batch sizes. After doing some research we realized that almost all examples on the Internet on Pytorch training use Datasets and Dataloaders. Then, we tried to find a way to convert our Numpy feature vectors to datasets. After looking for many examples on the Internet we realized we first need to convert Numpy arrays to tensors, then pass tensors to a Dataset and pass the Dataset into a Dataloader. In addition, we needed to put two variables inside the Dataloader, one for the input and one for the target to be used in the loss method of the training and we did not decide the type of the target. We did long searches on the Internet, but we could not find any answers to our problems. We found the answer just by trial and error method. We did several type conversions, casting and squeezing. Although we reached 100% training accuracy by the fifth epoch, the loss rate was still reduced. Therefore, we did several experiments deciding on the epoch number. We observed that after 20th epoch, there are some increases and decreases on the loss rate, so we thought we are doing some overtraining and decided the keep the epoch number around 15. The localization and classification accuracy with different epoch number can be found in Appendix Figure 28, Table 1. We continued with the Selective Search algorithm. Since we used the same algorithm mentioned in the description of the project, we did not encounter any problems while trying the algorithm. Since the images in the test folder were named like “0”, “1” and “10”, our glob method took number 10 after number 1 instead of 2. Therefore, we changed the first names from “0” to “00” to prevent this confusion. Since we solved many problems during the training step, we did not encounter many challenges on this step.

After making sure that the testing step worked without any problems, we decided to improve the acquired accuracies of our system. Since the classification accuracy was 100% by the first trial, we did not make any changes for our neural network, but in order to improve the localization results for every image, various implementations for the pre-processing the bounding boxes and the feature vectors were done. However, some of them yielded worse results than our expectations so most of them were discarded. For instance, the threshold for dropping out some of the proposed bounding boxes, which were provided by an external library, was first implemented by thresholding it to the half size of the provided bounding boxes and the accuracy of 72% was acquired for the bounding boxes [1]. However, this implementation would be problematic in real-life scenarios, as it would be impossible to determine the bounding boxes of real-life objects beforehand. Due to that, this idea was discarded. Afterwards, some random sizes for bounding boxes such as 70x70 or 85x85 were tested and it did not give good results, 19% and 23% when compared to the first implementation so they discarded as well. This did not give expected results because of the fact that all of the images were different sized and this thresholding method would not be appropriate for smaller images or larger images. As a new approach, the threshold of the bounding boxes was decided by a ratio of the size of the image as mentioned in the Implementation Strategies section. Several thresholds were tested and the best result was obtained with 12% which was 49% localization accuracy and 100% classification accuracy. Then we changed the number of epochs in order to see whether it improved our results we could not observe an increase within the localization accuracy when we increased or decreased it (See Table 28 from Appendix). These results further ensured us that the best number of epochs for the network is 15. Afterwards, we decided to test with different modes of Selective Search other than the currently used single-mode. Firstly, we tested with quality mode, but this did not give any results due to lack of memory during the calculation of the bounding boxes. Then, we decided to test our algorithm with the fast mode but it gave too many results for each image and it also did not work due to memory problems so we decided to keep using the single-mode in the Selective Search. As previous results were not sufficient, and changing the mode of the selective search did not yield with any results, another approach was necessary. Every feature vector within both the training stage and the testing stage were normalized with its norm. The normalization formula that was used was $x / (||x|| + 0.0001)$. This trial caused a significant decrease in both the classification accuracy as well as the localization accuracy which were %16 for classification and %13 for localization. Afterwards, it was discovered that the problem was not setting normalized feature vectors to the 0-1 range. After ensuring that every value within the feature vectors was in 0 - 1 range, the accuracy for the classification increased back to 100% and the accuracy for the localization increased back to 40%. This was not as good as before normalization, so we thought that it might be because of the threshold value of 12% that was set beforehand. In order to improve the results, another threshold for the size of the bounding boxes was tested again. After testing with a threshold value of 5% the results improved back to 46% and we decided to further reduce the threshold value as we did not have any problems with the memory thanks to using single mode while doing the selective search [1]. After, decreasing the threshold to 2%, it decreased the locality accuracy to 45% again. After getting these results, we realized that we discarded some of the windows that contained the starfish even with the 2 % threshold so we decided to run the algorithm without any threshold, even though this increased the accuracy of the starfishes, the accuracy of the tigers decreased, thus resulting on a 45% again. Due to that, we decided to be satisfied with the results as we were able to perfectly classify every single test image correctly. Since we experimented with many variables, we put the examples of candidate windows without any threshold value and localization results for the best accuracy we got. As it can be seen from the examples of the localization results (from Figure 18 to 27), although the localization results of the incorrectly labelled classes were enough to make a correct labelling, they were not in the correct bounding boxes. The reason for this, for very small objects like in the Figure 18, we may have discarded the correct bounding boxes since the threshold is determined according to the images' actual size. However, we used this method since it gives better results overall as discussed in above. Moreover, the other problem that reduces our localization accuracy is that like in Figure 23, even if the head of the animals were localized correctly, the bounding box did not include the whole body of the animal. Also, as it can be seen from Figures 22, 24, 27, one common similarity of incorrectly labelled classes is that they tend to choose the whole image instead of some smaller parts of the animal and this also caused some reduction in the localization accuracy.

VI. References

[1] J. R. Uijlings, et al. "Selective search for object recognition," International Journal of Computer Vision, pp. 154–171, 2013
[2]"10.7. glob - Unix style pathname pattern expansion¶," 10.7. glob - Unix style pathname pattern expansion - Python 2.7.17 documentation. [Online]. Available: https://docs.python.org/2/library/glob.html. [Accessed: 07-Jan-2020].
[3]"Image Module¶," Image Module - Pillow (PIL Fork) 3.1.2 documentation. [Online]. Available: https://pillow.readthedocs.io/en/3.1.x/reference/Image.html. [Accessed: 07-Jan-2020].
[4]"NumPy¶," NumPy. [Online]. Available: https://numpy.org/. [Accessed: 07-Jan-2020].
[5]"Training a Classifier¶," Training a Classifier - PyTorch Tutorials 1.3.1 documentation. [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#train-the-network. [Accessed: 07-Jan-2020].
[6]"torchvision.models¶," torchvision.models - PyTorch master documentation. [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html. [Accessed: 07-Jan-2020].
[7]"torch.nn¶," torch.nn - PyTorch master documentation. [Online]. Available: https://pytorch.org/docs/stable/nn.html. [Accessed: 07-Jan-2020].
[8]"torch.optim¶," torch.optim - PyTorch master documentation. [Online]. Available: https://pytorch.org/docs/stable/optim.html. [Accessed: 07-Jan-2020].

VII. Appendix

A. Contributions

- Every part included code and report were shared and done equally.

B. Tables

Epoch	Threshold (%)	Locality Accuracy(%)	Classification Accuracy (%)
10	12	49	100
15	12	47	100
20	12	47	100
15	14	48	100
15	13	46	100
15	10	48	100

Fig. 28. Accuracies before normalizing vectors