Object Localization and Recognition

CS484 Term Project

Atakan Serbes
Computer Engineering
Bilkent University
Ankara, Turkey
atakan.serbes@bilkent.edu.tr

Ana Peçini
Computer Engineering
Bilkent University
Ankara, Turkey
ana.pecini@ug.bilkent.edu.tr

Endi Merkuri
Computer Engineering
Bilkent University
Ankara, Turkey
endi.merkuri@ug.bilkent.edu.tr

***Abstract*—A method for image classification and object localization is developed. This method is similar to R-CNN (Region-based convolutional neural network) model. The images are padded and resized to the same size and fed into a pre-trained model for feature extraction. The features extracted are then used with the labels to train a 2-layer feed-forward neural network classifier. Candidate windows from test images are extracted using selective search method and the windows are classified and localized accordingly with evaluation criteria.**

***Index Terms*—object localization, recognition, feature extraction, candidate windows, classification, evaluation, selective search, neural network, deep learning, ResNet**

I. Introduction

The purpose of this project is to classify and localize animals from ImageNet [?] database. The objects to be detected are classified into 10 different classes: eagle, dog, cat, tiger, starfish, zebra, bison, antelope, chimpanzee, and elephant. The dataset is divided into 400 train images and 100 test images. Sample images from dataset are displayed in Figure 1.

This paper reports on the results obtained from using a 2-layer feed-forward neural network classifier that runs on object proposals on images, which is a similar architecture to R-CNN (region-based convolutional neural network) model.

We have organised the rest of this paper into 3 sections where we describe our methodology, present classification and localization results and metrics and finally discussing the performance of the system according to different implementation decisions and parameter settings.

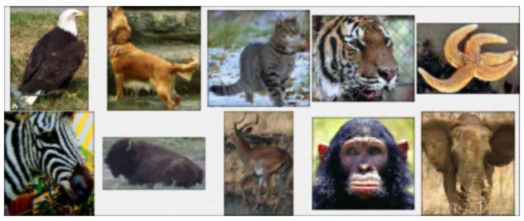


Fig. 1: Dataset Samples

II. Methodology

A. Pre-processing

In order for the image to become a proper input for the deep network classification model, it needs to go through two main pre- processing steps: data normalization and feature extraction. The pre-trained ResNet-50 model [?] by PyTorch in Python is used to extract the images' visual features.

1) Data Normalization: Images are loaded from file and converted to numpy array using PIL and Numpy libraries and converted to RGB. Since images have different aspect ratio and sizes, we pad them to a square shape. To pad, we identify the largest dimension and we concatenate the image array with a numpy matrix of zeros along the x- axis or y- axis accordingly using numpy.hstack and numpy.vstack [?] on both sides of the image. The ResNet-50 model expects as an input an image of size 224×224, so we resize all the images. The implementation of padding is depicted in Listing 1.

```
# Create pads to concatenate
hpad = np.zeros(( (int)(max_dim), (int)((max_dim -
    min_dim) /2) , 3))
vpad = np.zeros(( (int)((max_dim - min_dim) /2 ), (
    int)(max_dim), 3))

# Part 3: Pad the image
if max_dim == org_size[0]:
    res_im = np.hstack((hpad, image_arr, hpad))
else:
    res_im = np.vstack((vpad, image_arr, vpad))
```

Listing 1: Padding implementation

Next we normalize the image by performing the following sequence of operations:

- Dividing the image by 255 so all its values are in the range $[0, 1]$.
- Subtracting 0.485, 0.456, 0.406 from red, green and blue channels, respectively.
- Dividing red, green and blue channels by 0.229, 0.224, 0.225, respectively.

2) Feature Extraction: Once the image is converted to RGB, padded, resized and normalized, it can be used as an input to the pre- trained ResNet model. The model is loaded and its fully-connected layer is changed to identity to produce an output of 2048- dimensional feature vector of image data as follows:

```python
import torchvision
import torchvision.models as models
class Identity(nn.Module):
    def _init_(self):
        super()._init_()

    def forward(self, x):
        return x
model = models.resnet50(pretrained = True)
#Delete last layer of the model
#Set the models last layer to identity
model.fc = Identity()
```

Listing 2: ResNet-50 model

B. Classification Model

For classification, we use 2-layer feed-forward neural network classifier as outlined on the project description. We used PyTorch to create a custom Dataset object which can be used with the built-in DataLoader to feed our image data to the neural network. To customize the Dataset object in order to handle the ImageNet data, we need to override the __len__ function which returns the size of the dataset and the __getitem__ function which returns a sample from the dataset given an index. In our project, we have used a small batch size of 16 images to provide a regularizing effect and lower generalization error. The loss function used is Cross Entropy Loss and the optimizing method is Stochastic Gradient Descent. The number of neurons of the hidden layers(i.e the hidden size) is set to 500. The implementation of the Dataset object is depicted on Listing 3.

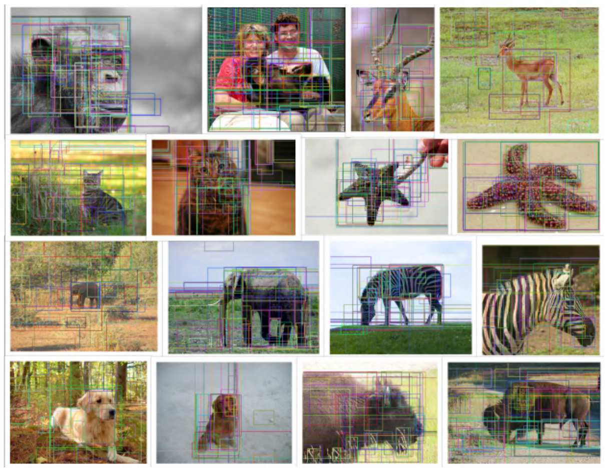
```python
from torch.utils.data.dataset import Dataset

class MyCustomDataset(Dataset):
    def __init__(self, a , b ):
        self.inputs = a
        self.labels = b

    def __getitem__(self, index):
        return (self.inputs[index], classesDict[self
    .labels[index]])

    def __len__(self):
        return len(self.inputs)

trainset = MyCustomDataset(a = input2 , b = labels)
trainloader = torch.utils.data.DataLoader(trainset,
    batch_size=16,
                                              shuffle=
    True, num_workers=2)
```

Listing 3: Dataset and DataLoader

C. Testing

For testing the system, image data is subject to Selective Search [?] algorithm to extract candidate windows (region proposals) for localizing the object. All candidate windows are fed to the classification model and the candidate window with the highest class score is obtained as the object localization result.

1) Candidate Windows: Candidate windows are extracted using the selective search algorithm [?] provided by OpenCV library. After experimenting with two modes of selective search: quality and fast, the algorithm is switched to fast, because there is not too much difference in performance between the two. We restrict the number of candidate windows by selecting only those windows whose area is larger than a threshold. By experimenting, we observed that an area of 5000 is a good threshold to select the most relevant windows, because generally the objects are big enough to cover an area larger than 5000. The result of selective search is stored into a numpy array where each entry consists of 4 values: the x and y coordinate of the lower left corner, the width and the height of the window. Figure 2 shows the results of applying selective search to 2 images out of each class.


(a) Set 1

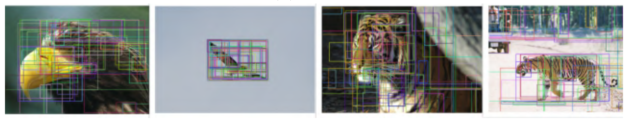

(b) Set 2

Fig. 2: Selective Search outputs

2) Localization: For each image, after preprocessing the candidate windows, extracting their feature vectors and feeding them into the classifier, we store in numpy arrays the window with the highest confidence value, its value and the respective class. The results of image classification and localization are presented in Figure 3 and 4. The green color represents the actual bounding box and class and the violet color represents the predicted bounding box and class.

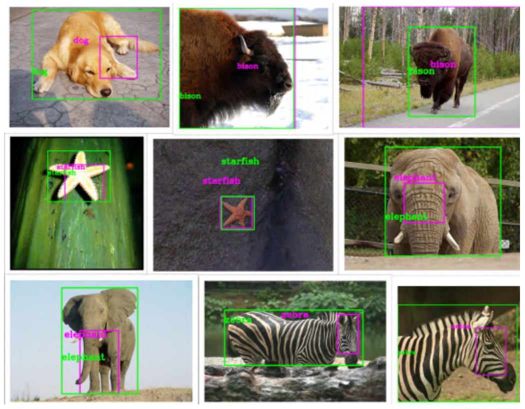


Fig. 3: Localization outputs

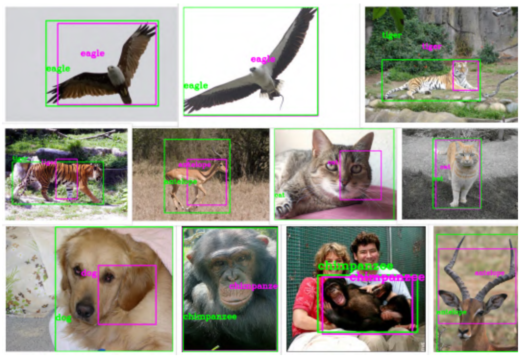


Fig. 4: Localization outputs

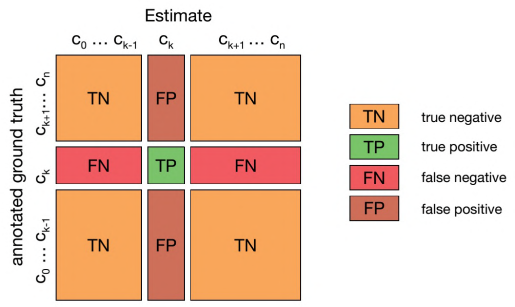


Fig. 5: Confusion Matrix for multiclass classification

D. Quantitative Performance Evaluation

We have used the following metrics to evaluate the accuracy of our model:

1) Classification Accuracy: To evaluate our classification accuracy we firstly computed the confusion matrix for our results of running the test data. To compute the confusion, for each pair of classes we counted the number of images that are predicted as one and are part of the other class as follows:

```
confusion_matrix = np.zeros((len(classes),len(classes)))
for i in range(len(actual_classes)):
    confusion_matrix[classesDict[pred_classes[i]], classesDict[actual_classes[i]]] += 1
```

Listing 4: Confusion Matrix Computation

After computing the confusion matrix we used it to find the number of true positives, true negatives, false positives and false negatives for each of classes using the following formula: As it is shown in the figure [?] in order to calculate the true positives for each class we have to get the values along the diagonal of the matrix. The false positives are calculated by computing the sum of the elements along the row or column of the predicted classes and then subtracting the number of true positives. The number of true negatives for a class is equal to the number of all images minus the sum of true positives, false positives and false negatives.

2) Localization Accuracy: In order to compute the localization accuracy of our model we have calculated the bounding box overlap ratio for each of the test images using the formula given in the project specification. The ratio was obtained by using the following code, where the function intersection and union are auxiliary functions and the function iou returns the ratio for one image given its predicted and ground truth bounding box:

```
def intersection(predicted_box, ground_truth):
    width_max = max(predicted_box[0], ground_truth[0])
    width_min = min(predicted_box[2], ground_truth[2])
    width_diff = width_min - width_max

    height_max = max(predicted_box[1], ground_truth[1])
    height_min = min(predicted_box[3], ground_truth[3])
    height_diff = height_min - height_max

    area = width_diff * height_diff
    if area <= 0:
        return 0
    else:
        return area

def union(predicted_box, ground_truth):
    predicted_area = (predicted_box[2]-predicted_box[0])*(predicted_box[3]-predicted_box[1])
    ground_truth_area = (ground_truth[2]-ground_truth[0])*(ground_truth[3]-ground_truth[1])
    total_area = predicted_area + ground_truth_area
    union = total_area - intersection(predicted_box, ground_truth)
    return union

def iou(predicted_box, ground_truth):
    return intersection(predicted_box, ground_truth) / union(predicted_box, ground_truth)
```

Listing 5: Bounding box overlap ratio

While computing the bounding box overlap ratio for each of the images we count the ones with a higher ratio than 0.5 and we use that to calculate the overall percentage of the localization accuracy.

III. Results

In this section we present the confusion matrix and the performance evaluation metrics for the classification results and the average accuracy for localization for each class.

TABLE I: Confusion matrix

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	10	0	0	0	0	0	0	0	0	0
2	0	10	0	0	0	0	0	0	0	0
3	0	0	10	0	0	0	0	0	0	0
4	0	0	0	10	0	0	0	0	0	0
5	0	0	0	0	10	0	0	0	0	0
6	0	0	0	0	0	10	0	0	0	0
7	0	0	0	0	0	0	10	0	0	0
8	0	0	0	0	0	0	0	10	0	0
9	0	0	0	0	0	0	0	0	10	0
10	0	0	0	0	0	0	0	0	0	10

TABLE II: Performance Evaluation Metrics

	TP	**FP**	**FN**	**TN**	**Precision**	**Recall**
eagle	10	0	0	90	1	1
dog	10	0	0	90	1	1
cat	10	0	0	90	1	1
tiger	10	0	0	90	1	1
starfish	10	0	0	90	1	1
zebra	10	0	0	90	1	1
bison	10	0	0	90	1	1
antelope	10	0	0	90	1	1
chimpanzee	10	0	0	90	1	1
elephant	10	0	0	90	1	1

TABLE III: Average localization accuracy per class

	Average localization accuracy
eagle	0.589
dog	0.390
cat	0.413
tiger	0.234
starfish	0.626
zebra	0.252
bison	0.507
antelope	0.362
chimpanzee	0.543
elephant	0.325

IV. Discussion

As it can be seen from our results, we have **100%** overall classification accuracy and **39%** overall localization accuracy. The high classification accuracy might be accounted to the ResNet model, since it may extract features in such a way that the classification is linearly separable. Another reason might be the small number of classes. There also exists a possibility that the model might be overfitting due to small number and diversity of training samples. On the other hand, the low localization accuracy might be because of several reasons:

- The training data is biased with pictures of close-up heads of animals. Because of this, the classifier learns to detect heads better than the whole body causing it to output a greater confidence for the said bounding boxes. This effect is more prominent in the pictures of tigers and zebras especially.
- The localization result depends on the selective search output. If the selective search algorithm does not produce a window close to the ground truth, the result is not optimal. An example of such a case is depicted on Figure 11.

A. Different Implementation Decisions

- We experimented with the hidden size, which did not give a considerable improvement in result. The final decision was keeping the hidden size to 500.
- We experimented with the number of epochs and batch size for the training of the classifier in order to check if these results were the effect of overfitting. We tried to reduce the number of epochs to avoid overfitting. We decided to keep 30 epochs. However, there was not a huge difference in results when training for a smaller number of epochs.
- We tried two different versions of selective search namely "Fast" and "Quality". Although "Quality" settings provide more windows and computationally heavy in our case, the "Fast" setting did outperform the "Quality" setting in general.

B. Examples

By observing our results, we can conclude that easiest objects to detect were starfish and eagle, with an overall accuracy of 63% and 59%, respectively. The most difficult objects to detect were tiger and zebra with an overall accuracy 23% and 25%, respectively. Starfish and eagles were easier to detect because of their easily separable background in test images. In figure 6, there are shown good results of localization from different classes.

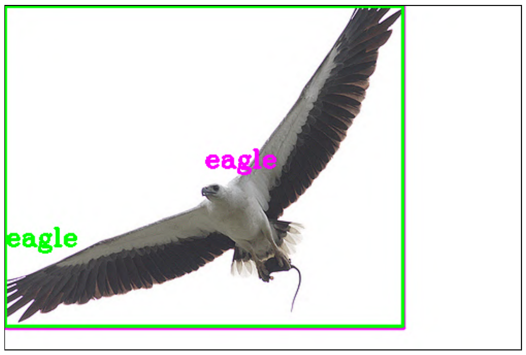


Fig. 6: Localization result with 99.1 % accuracy

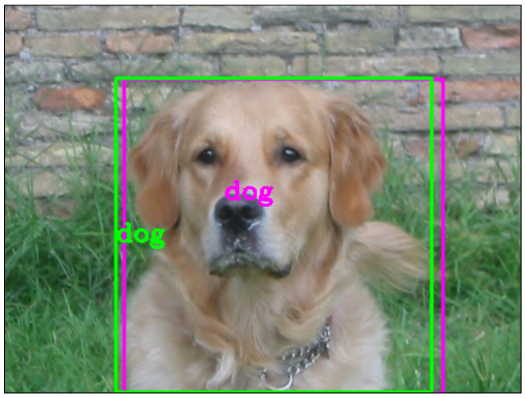
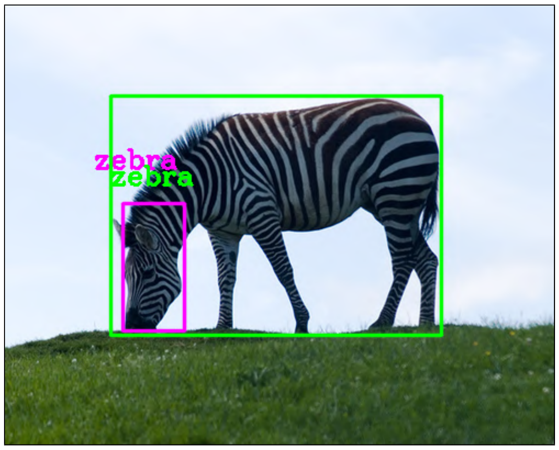
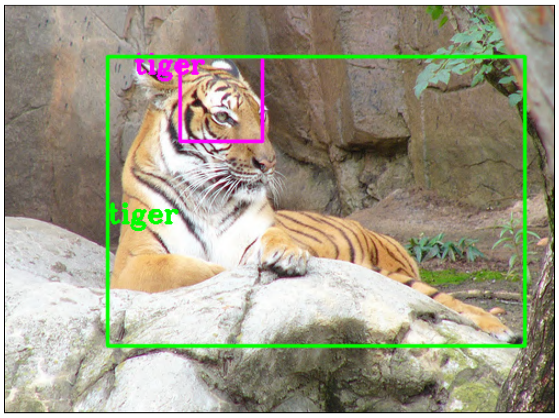


Fig. 7: Localization result with 93.40 % accuracy

In the case of tigers and zebras, there are a lot of training images containing only their cropped faces. Another reason for the incorrect localization might be that the features of the animals are similar to those of the surrounding background, which makes them harder to distinguish from the environment. In figure 7, bad results are illustrated from different classes.



(a) Localization result with 9.8 % accuracy



(b) Localization result with 5.77 % accuracy

Fig. 8: Some bad localization results

We have some of the results as in Figure 8 where the localization result covers only a small part of the object. The output with the highest classification score coming from the selective search in this case is the forehead of the elephant. And by going back and checking the training set, we can easily find cropped elephant foreheads as shown in Figure 9. As a result, the features extracted from the ResNet model gives the highest relevance to the forehead output of selective search.

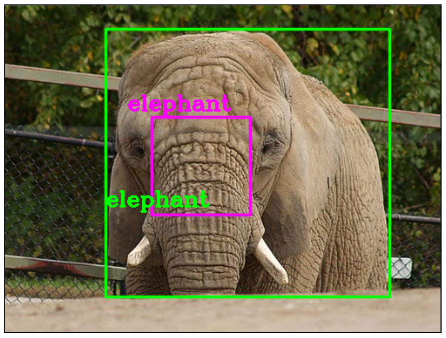


Fig. 9: Elephant forehead detected in a test set

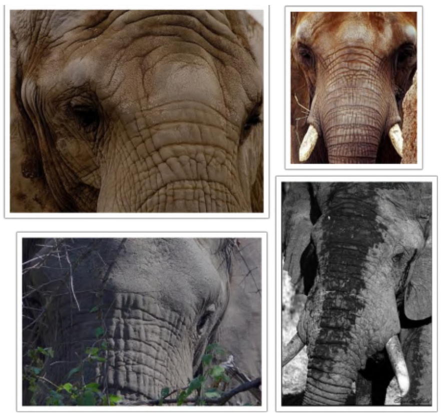


Fig. 10: Elephant forehead examples in training set

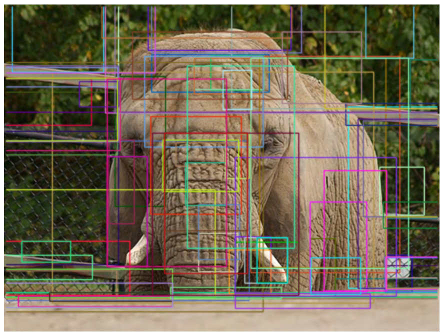


Fig. 11: Selective search output examples for Figure 8

C. Suggestions for Improvement

The prediction could be improved if we had a bigger training dataset with different angles and scales of animals, dropout layers and batch normalization. A suggestion to chech whether the model is learning would be having a validation set.

APPENDIX

D. Division of Labor

- **Atakan Serbes**. Pre-processing image, loading and storing images, removing the last layer of the pretrained ResNet model, creating the custom dataset class, training the classifier, selective search, finding the top-scoring window for object localization, classification accuracy, localization accuracy.
- **Ana Peçini**. Pre-processing image, loading and storing images, training the classifier, finding the top-scoring window for object localization, classification accuracy, localization accuracy.
- **Endi Merkuri**. Pre-processing image, creating the custom dataset class, training the classifier, finding the top-scoring window for object localization, classification accuracy, localization accuracy.

In addition, all of the group members participated equally in writing the report.

REFERENCES

[1] L. Fei-Fei, J. Deng and K. Li, ”ImageNet: Constructing a large-scale image database”, Journal of Vision , vol. 9, no. 8, pp. 1037-1037, 2010. Available: 10.1167/9.8.1037.

[2] K. He, X. Zhang, S. Ren, J. Sun, ”Deep Residual Learning for Image Recognition,” IEEE Conference on Computer Vision and Pattern Recognition, 770-778, 2016.

[3] ”numpy.hstack - NumPy v1.17 Manual”, Docs.scipy.org, 2019. [Online]. Available: https://docs.scipy.org/doc/numpy/reference/generated/numpy.hstack.html. [Accessed: 09- Jan- 2020].

[4] J. R. Uijlings, et al. ”Selective search for object recognition,” International Journal of Computer Vision, pp. 154- 171, 2013.

[5] Devopedia. 2019. ”Confusion Matrix.” Version 6, August 20. Accessed 2020-01-09.