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

The goal of this project is to develop a method for image classification and object localization. The method will use a framework that is similar to the R-CNN (region-based convolutional neural network) model. You are expected to work in groups of three. Specifications for the components of the approach are given below.

## 1 Data

ImageNet (`http://www.image-net.org`) is a very large collection of images belonging to a large number of different classes (called synset). Currently there are 14,197,122 images from 21841 synsets in the data set. In this project, you will be working on a small subset of ImageNet. You will be provided around 500 images (400 of them for training and the rest for testing) from 10 different classes: eagle, dog, cat, tiger, starfish, zebra, bison, antelope, chimpanzee, and elephant.

The training images contain only the object from the corresponding class. That is, each training image is tightly cropped around the object as shown in Figure 1. In contrast, the test images contain background information. For instance, you will see people on top of an elephant or a forest behind a tiger or an antelope. Dealing with such background clutter and scale variations is part of the project. Still, each test image contains exactly a single instance of the target object class.

Overall, there are two steps in the pipeline: *localization* that will be performed only on the test images, and *recognition* that will use a learning procedure on the training data and a classification stage on the test data. Feature extraction for both training and testing will be done by using a pre-trained deep network. Your model will have two outputs: predicting a single object class and the corresponding object bounding box in each test image, where a bounding box is denoted by four coordinates: `x1`, `y1` (upper left corner), `x2`, `y2`, (lower right corner).

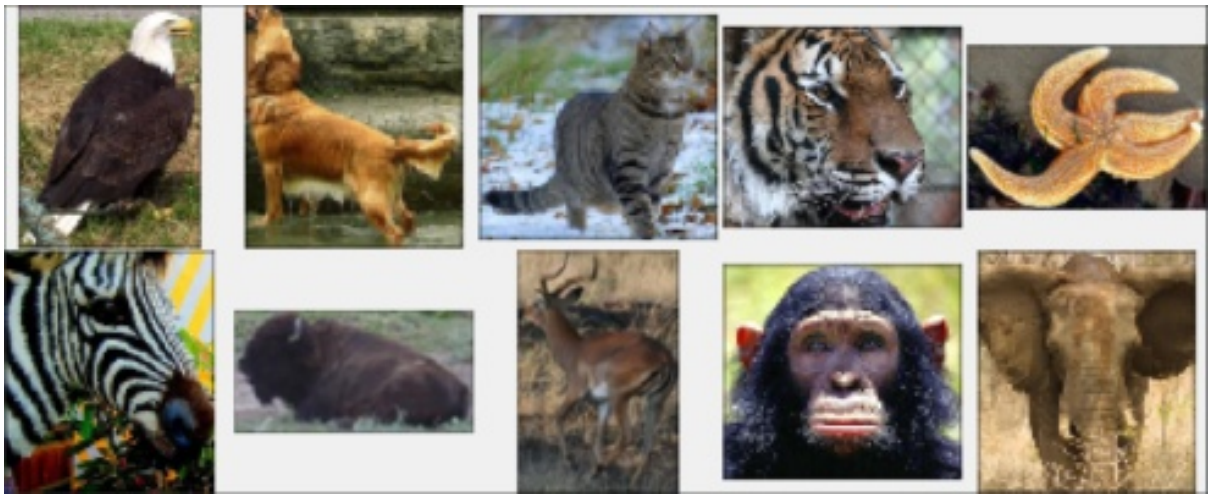

Figure 1: Example images from the training set.

## 2 Background

In this project, you will use a framework that is similar to the *R-CNN* model proposed by Girshick et al. It is discussed during the lectures but also read the following papers to learn more about the model:

- R. Girshick, J. Donahue, T. Darrell, J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," IEEE Conference on Computer Vision and Pattern Recognition, 580-587, June 23-28, 2014.
- R. Girshick, J. Donahue, T. Darrell, J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(1):142-158, January 2016.

While implementing the model, you will use a *ResNet-50* network that was pre-trained on the ImageNet data set in order to extract visual features. Read the following paper to learn more about this network:

- 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.

At this point, it is up to you to decide which deep learning framework to use in order to extract the visual features from a pre-trained ResNet-50. In this document, we encourage you to develop your algorithm with Python, and we provide examples that use PyTorch. See `https://pytorch.org` to learn how to install it on your computer.

## 3 Pre-processing

### 3.1 Data normalization

When you look at the images in the training set, you will notice that each image has a different aspect ratio and size. So, the first step is to apply the following pre-processing sequence to the images in the training set so that they will all have a size of $224 \times 224 \times 3$ to become a proper input for the pre-trained deep network.

1. Load an image.
2. Convert it to RGB (some images may contain an alpha channel).
3. Pad the image with minimum number of zeros so that it becomes a square image.
4. Resize the image to $224 \times 224$.
5. Apply normalization, if necessary. Depending on the deep learning framework you use, you may have to apply a particular normalization scheme. For instance, if you are using PyTorch, you have to

   (a) Divide an image by 255 so that its values are in the range $[0, 1]$,

   (b) Subtract $0.485, 0.456, 0.406$ from red, green and blue channels, respectively,

   (c) Divide red, green and blue channels by $0.229, 0.224, 0.225$, respectively,

   as described at `https://pytorch.org/docs/stable/torchvision/models.html`.

(a) (b)

Figure 2: Visualization of the padding scheme. For each example, (left) the raw image, (right) its padded version.

In Figure 2, you can see two examples for the padding scheme described above. If you work with Python, you may use the PIL and Numpy libraries to read and manipulate the images as follows:

```
import numpy as np
from PIL import Image
image = Image.open(image_path).convert('RGB') # load an image
image = np.asarray(image) # convert to a numpy array
... # do the padding, rescaling and normalization
# at this point make sure that the image is of type np.float32
```

### 3.2 Feature extraction

Once you have the $224 \times 224$ images, you can extract 2048-dimensional representations from ResNet-50. Luckily, PyTorch provides a way to easily load the pre-trained weights of ResNet-50. Once you load the model, you can convert the Numpy image (you obtained above) to `torch.FloatTensor`, extract its feature vector, and finally convert it back to a Numpy array as follows:

```
import torch
# we append an augmented dimension to indicate batch_size, which is one
image = np.reshape(image, [1, 224, 224, 3])
# model takes as input images of size [batch_size, 3, im_height, im_width]
image = np.transpose(image, [0, 3, 1, 2]])
# convert the Numpy image to torch.FloatTensor
image = torch.from_numpy(image)
# extract features
feature_vector = model(image)
# convert the features of type torch.FloatTensor to a Numpy array
# so that you can either work with them within the sklearn environment
# or save them as .mat files
feature_vector = feature_vector.numpy()
```

## 4 Training the system

First, extract the feature vectors for each training image as described above. Optionally, you can $\ell_2$ normalize each feature vector by dividing the vector by its norm in order to make the

representations more robust. That is, for a feature vector $x$, its $\ell_2$-normalized version is given by $x/(\|x\| + 0.0001)$. (When you use $\ell_2$ normalization at this stage, you should also normalize each feature vector at test time as well.)

Then, using the ResNet-50 features and the object labels, train a 2-layer feed-forward neural network classifier. The following code defines such a network in PyTorch:

```
from torch import nn
class Feedforward(nn.Module):
    def __init__(self, hidden_size):
        super().__init__()
        self.fc1 = nn.Linear(2048, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, 10)
    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

The input size of the first layer is 2048, which is the length of ResNet-50 features. The output size of the last layer is 10, which is the number of classes. You should experiment with the hidden size parameter to build an effective classifier. Optionally, you can experiment with different activation functions and additional normalization (e.g., nn.BatchNorm1d) and dropout (nn.Dropout) layers.

After defining the model, you can train it using an optimization algorithm. For PyTorch, this process is described at `https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html` and `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#train-the-network`. You can use the Stochastic Gradient Descent (SGD) algorithm. You must find a suitable learning rate parameter for your optimizer through experimentation, since a high rate will result in divergence from the minima, and a low rate will make the training slow. Cross-entropy loss function (nn.CrossEntropyLoss) is suitable for your multi-class classification task. You can stop the training of the network after a carefully-chosen number of iterations, or when the loss value stops improving (delta loss is less than a threshold).

# 5 Testing the system

Since test images contain natural scenes where target objects appear together with their backgrounds, you need to implement both object localization and classification for the testing stage.

## 5.1 Extracting candidate windows

In order to predict the bounding boxes of targets in test images, you will first extract candidate windows (region proposals) in which objects are expected. This will be done by using the *Selective Search* region proposal algorithm:

- J. R. Uijlings, et al. "Selective search for object recognition," International Journal of Computer Vision, pp. 154–171, 2013.

Selective Search works similarly to your Homework 3 assignment. First, the image is over-segmented. Then, a hierarchy of segments is constructed by iteratively merging two neighboring segments that are similar to each other based on color, texture, size, and shape compatibility,

until all segments are merged into a single segment that is the whole image. At each iteration, the bounding box of the newly-formed segment is added to the list of region proposals.

You can extend your Homework 3 implementation to perform Selective Search, or you can use code from other sources. Selective search algorithm has a very high recall, and you can expect it to output more than a thousand proposals. If the large number of proposals creates a bottleneck in your system, you can restrict it by discarding boxes that are too small to contain an object using a threshold.

### 5.2 Classification and localization

Once you extract the candidate windows in a test image, for each candidate window, apply the pre-processing step described above and extract the ResNet-50 features. Once you have the representation for each candidate window, you can obtain the class scores by applying your classification model to the feature vectors. Finally, find the maximum classification score among all candidate windows and classes for a test image, assign the image to the class with the highest score among all windows, and use the top-scoring candidate window as the object localization result.

## 6 Evaluation

Quantitative performance evaluation will be done in two stages:

1. Classification accuracy: Compute the confusion matrix, precision and recall for each object type, and the overall accuracy in terms of the percentage of correctly classified test images. Present these results as tables.

2. Localization accuracy: Compute the percentage of correctly classified and localized test images. Here, a test image is considered as correctly classified and localized if it is assigned to the correct class, and (if correctly classified) the localization output has a *bounding box overlap ratio* above 50%.

. The bounding box overlap ratio between a ground truth box $g$ and a candidate window $b$ is measured by:

$$\frac{g \cap b \,(\text{intersection area})}{g \cup b \,(\text{union area})}. \tag{1}$$

The class id and the bounding box of each test image is provided in the file `bounding_box.txt`. You should not use this file for any purpose other than evaluating the classification and localization results on the test images.

## 7 Submit

You must submit the final report and the developed code through the online form by the deadline given on the course web page. Each team is required to submit a single copy (not once for each team member).

1. Report

   - Must be readable and well-organized. Format and content will be subject to grading.
   - Must provide proper explanation of the details of the approach, the implementation strategies, the results obtained, and the observations made.

- Must report the performance metrics as defined in Section 6.
- Must provide examples for candidate windows (object proposals) for at least one test sample from each object type. You should overlay the candidate bounding boxes on the corresponding image with different colors.
- Must provide examples for localization results obtained for at least two test samples from each object type. Overlay the localization output (window and class name) on the corresponding image. Make sure that some of the examples correspond to correct localization results, and some to incorrect ones (unless you have 100% localization accuracy).
- Must include a thorough discussion where you comment on the performance of your system according to different implementation decisions and parameter settings. You should also analyze the results for individual object types, i.e., which object types were easy and which ones were more difficult, and why you think they were easy/difficult. You should discuss your observations in your report. You should also provide some suggestions for improvement.
- Must include all figures with proper captions. The figures also have to be properly referenced in the text.
- Must follow the IEEE Computer Society two-column format as shown on the course web page.
- Each team member should also provide a written description of her/his own specific contributions to the project, and should include this information as an appendix of the report.

2. Code

   - Provide well-documented source code that is ready to re-run giving the same results as presented in the report.
   - You can use code from other sources only for
     - extracting features (ResNet-50),
     - training the classifier (2-layer Neural Network),
     - extracting candidate windows (Selective Search).

     Other than that, you have to implement the pre-processing, training, testing, and evaluation pipeline on your own. Using code from other students who are in this class or took the same class in previous semesters is not allowed.