Introduction to R-CNN and Implementation using PyTorch

R-CNN is a classic yet effective algorithm for object detection tasks. R-CNN is an old model architecture that was proposed in 2014, which relies on region proposal approach using selective search. If we apply the selective search algorithm on a RGB input image, it will segment the images into defined regions based on similarity and other properties. This algorithm is helpful for performing object detection task because it helps us to create defined number of boxes on a sample image, which actually can serve as proposed regions. The following image shows an example of selective search:

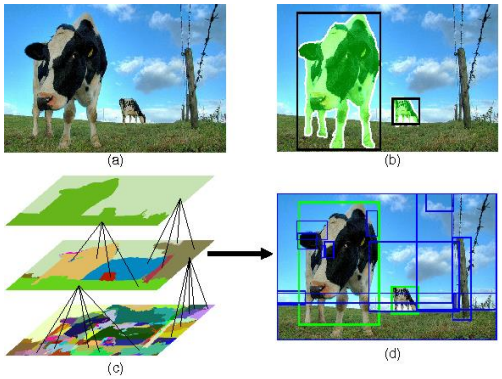


Figure-01: Visual illustration of selection search algorithms.

To start working with object detection, we need to gain basic concepts and intuitions of this method. There are many state-of-the-art models exist for performing this task, however, it is essential to get basic knowledge and developing models from scratch to solve object detection tasks. Here are some tutorials that can walk through the concepts of object detection and some architectures summary.

- Object Detection Part-1
- Object Detection Part-2
- Object Detection Part-3

After reading these blogs, we should start learning R-CNN to develop an object detection model algorithm from scratch. To get the overview of this model and internal architecture, we can read their original paper:

- Main paper

To implement this paper for developing R-CNN, we can follow this blog: link

Dataset

Dataset plays a crucial part in model training; therefore, we must be careful with dataset selection. Typically, benchmarked object detection datasets are larger in size. There are several benchmark datasets for this problem by which many state-of-the-art models were developed and evaluated. Some of them are:

- PASCAL VOC [Medium]
- COCO (Common Objects in Context) [Large]
- Open Image Dataset [Large]
- CityScape [Large]
- Roboflow Custom Datasets [Custom]

- iMaterialist (Product) Challenge [Large]
- Plant CLEF [Medium]
- Custom Datasets

We can also create our problem specific custom dataset by using **LabelImg** or **CVAT**, where we need manual annotation.

Implementation with PyTorch

After getting the theoretical intuition and architecture overview, we are ready to implement this model from scratch using PyTorch. Initially, we need to select a dataset for training R-CNN. I have utilized PASCAL VOC 2007 edition dataset, which is medium in size and beginner friendly. This dataset offers 9,963 images containing 24,640 annotated objects for 20 classes. Classes of this dataset is given below:

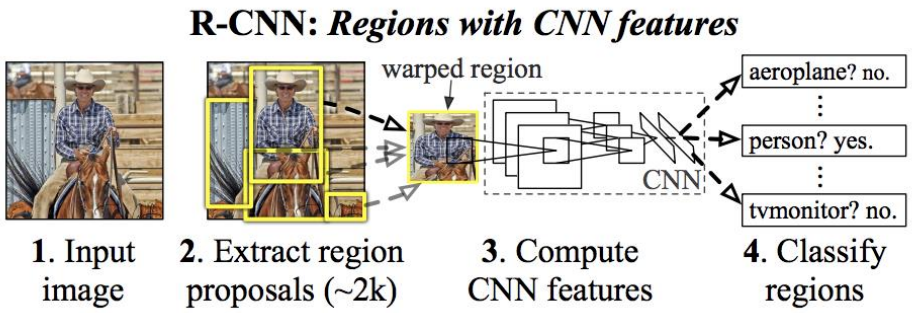
- *Person:* person
- *Animal:* bird, cat, cow, dog, horse, sheep
- *Vehicle:* aero plane, bicycle, boat, bus, car, motorbike, train
- *Indoor:* bottle, chair, dining table, potted plant, sofa, tv/monito

For each image, it has annotation like this,

```
(<PIL.Image.Image image mode=RGB size=500x333>,
 {'annotation': {'folder': 'VOC2007',
   'filename': '000012.jpg',
   'source': {'database': 'The VOC2007 Database',
    'annotation': 'PASCAL VOC2007',
    'image': 'flickr',
    'flickrid': '207539885'},
   'owner': {'flickrid': 'KevBow', 'name': '?'},
   'size': {'width': '500', 'height': '333', 'depth': '3'},
   'segmented': '0',
   'object': [{'name': 'car',
     'pose': 'Rear',
     'truncated': '0',
     'difficult': '0',
     'bndbox': {'xmin': '156', 'ymin': '97', 'xmax': '351', 'ymax': '270'}}]}
})
```

It contains image file and their annotation in a single tuple. This dataset designed in a way that we can detect multiple objects from the input image within these 20 annotated classes.

Model Architecture



Approaches

- Initially, we need to preprocess the dataset to train with a CNN model (i.e., vgg-16, resnet-50, densenet-169 or scratch). Because the dataset may contain many information but we need only images, objects and their classes to train with R-CNN network.
- Therefore, we will create custom datasets for both training and validation from the PASCAL VOC dataset. It provides 2501 training samples and 2510 validation samples.
- R-CNN network takes region proposals as input to train with CNN models. To create a custom dataset, we will apply selective search algorithm to each image to get proposed regions and collect the corresponding bounding boxes and labels of these images from annotation. We can save that information in pickle, json or yaml format.
- After that, we nee to train those proposals using a pre-trained CNN with fine tuning. This mode will be utilized for object detection from a test image. We need to perform inference to get the proper output.
- There is another way of training this network with bounding box regression. In this approach, first we need to extract a certain number of features from the images and train with a SVM classifier for classification. After that, we need to train a regression network using these features to learn the bounding box. We need to transform the bounding box values into box offsets according to the paper's formula for training the regressor. After training, we can use predicted class from the SVM classifier and the bounding box from the regressor.

Inference

- Pass the image and proposals to the model.
- Get the predictions (scores and offsets).
- Use NMS to filter out unnecessary boxes.
- Draw the bounding boxes on the input image.
- Display or save the final image with bounding boxes.

Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
- It cannot be implemented real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.
- Training is multistage pipeline: In R-CNN first a Convnet is fine-tuned on object proposals using log loss. Then, it fits SVM to Convnet features by replacing Soft Max. In third stage regressors are trained.
- In R-CNN, we need to forward and pass every region proposal through the Deep Convolution architecture (that's up to *~2000* region proposals per image). That explains the amount of time taken to train this model