ARTIFICIAL INTELLIGENCE AND COGNITIVE COMPUTING

OBJECT DETECTION & IMAGE CLASSIFICATION WITH FASTER R-CNN

Arena Gabriele, Corsaro Miriana, Ventimiglia Andrea Paolo

Cognitive Computing and Artificial Intelligence

University of Catania, Italy

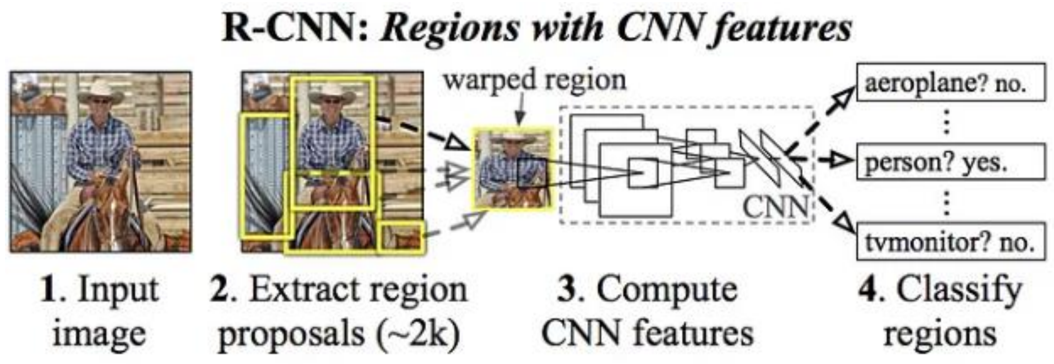
2022-2023

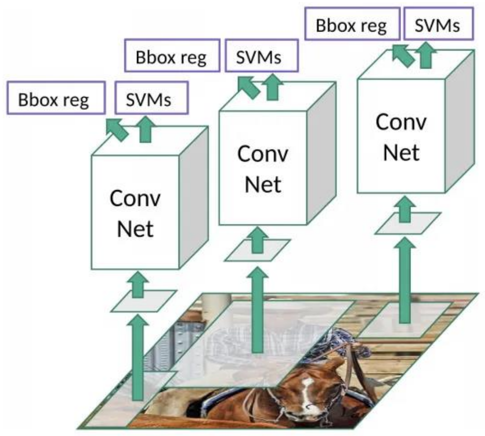
R-CNN

The R-CNN, Region-based Convolutional Neural Network, proposed in 2014, transforms an *object detection* problem into a classification problem.

It works as follows:

- About 2000 possible regions of interest (RoI) are extracted from an image using the *Selective Search* algorithm.
- The characteristics of each region are then extracted using a CNN.
- A SVM is applied for classification based on the extracted features.
- A linear regression is applied to narrow the bounding box of the object.





SELECTIVE SEARCH FOR OBJECT RECOGNITION: [LINK]

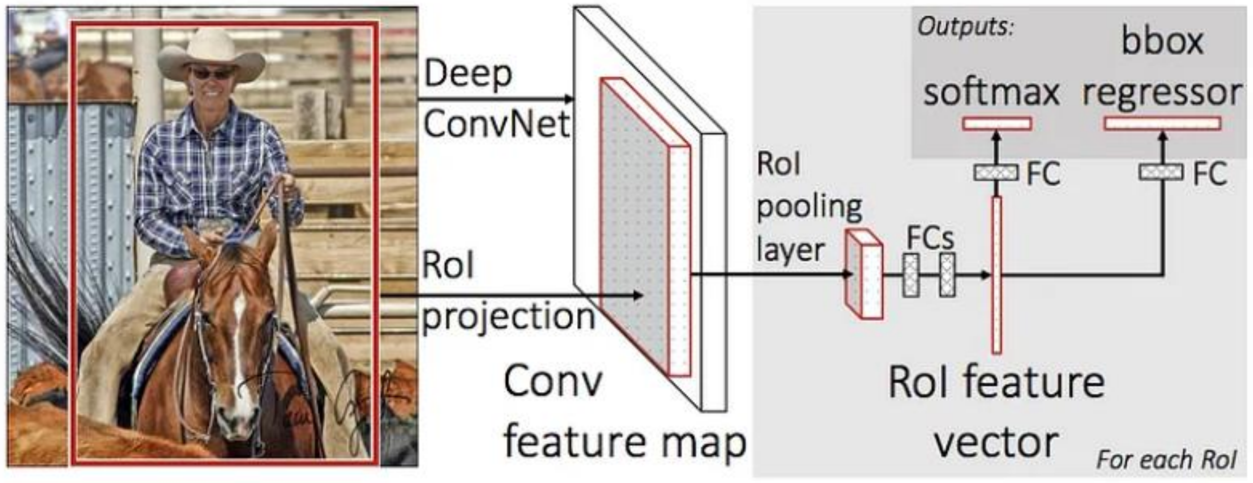
R-CNN

The main problem of R-CNN is due to the high computational cost:

- Each proposed region of an image must be classified by a separate convolutional network, which results in a large amount of computation since this operation must be performed for every single extracted region.
- Additionally, each bounding box regressor will be trained individually, without sharing of information.
- Furthermore, R-CNN uses a number of SVMs equivalent to the number of classes, which requires training each SVM individually during the training phase.

Fast RCNN

The approach of Fast RCNN is similar to the R-CNN algorithm but, instead of feeding the region proposals to the CNN, it feeds the input image to the CNN to generate a convolutional feature map.



Faster R-CNN

Faster R-CNN aims to improve the performance of Fast R-CNN by modifying the process of selecting regions of interest.

It works as follows:

- The early convolutional layers extracted the feature map to enable the extraction of regions of interest.
- Adds additional layers to extract regions of interest using a true convolutional network, the Region Proposal Network (RPN), which replaces the Selective Search algorithm.
- At an implementation level, Faster R-CNN can be seen as the composition of RPN and Fast R-CNN.

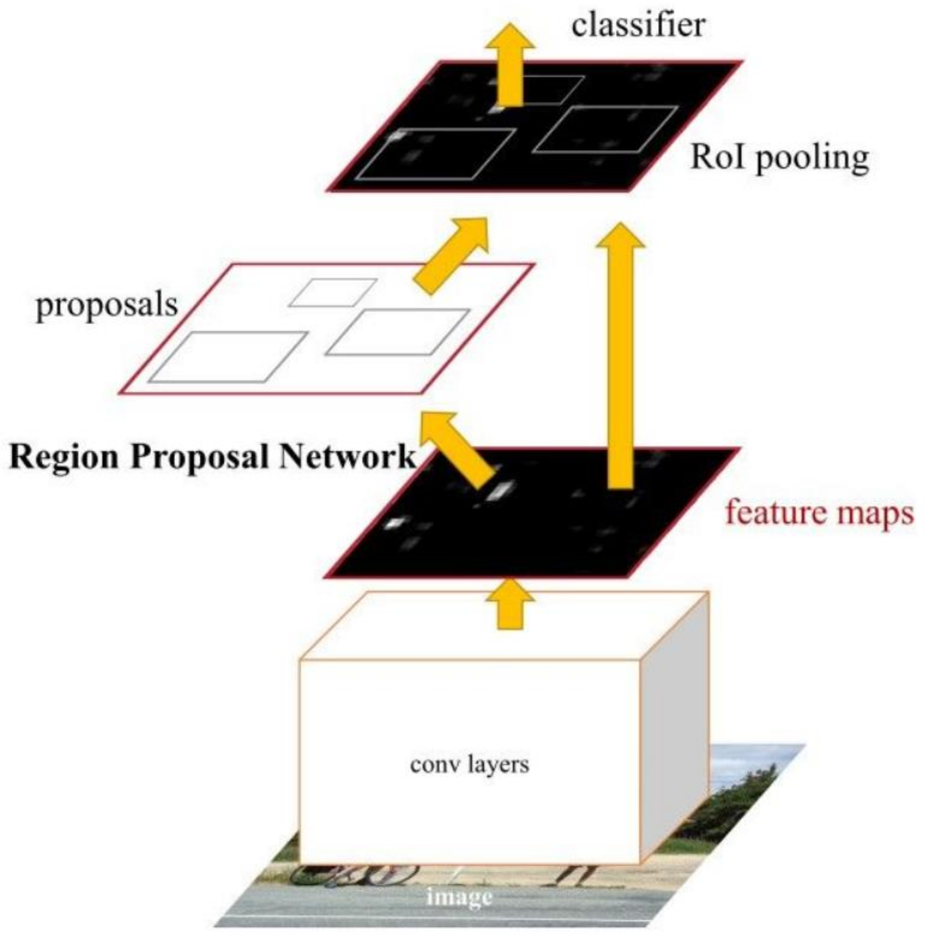
Faster R-CNN

The architecture of Faster R-CNN consists of 2 modules:

- **RPN - Region Proposal Network** : for generating region proposals.
- **Fast R-CNN**: for detecting objects in the proposed regions.

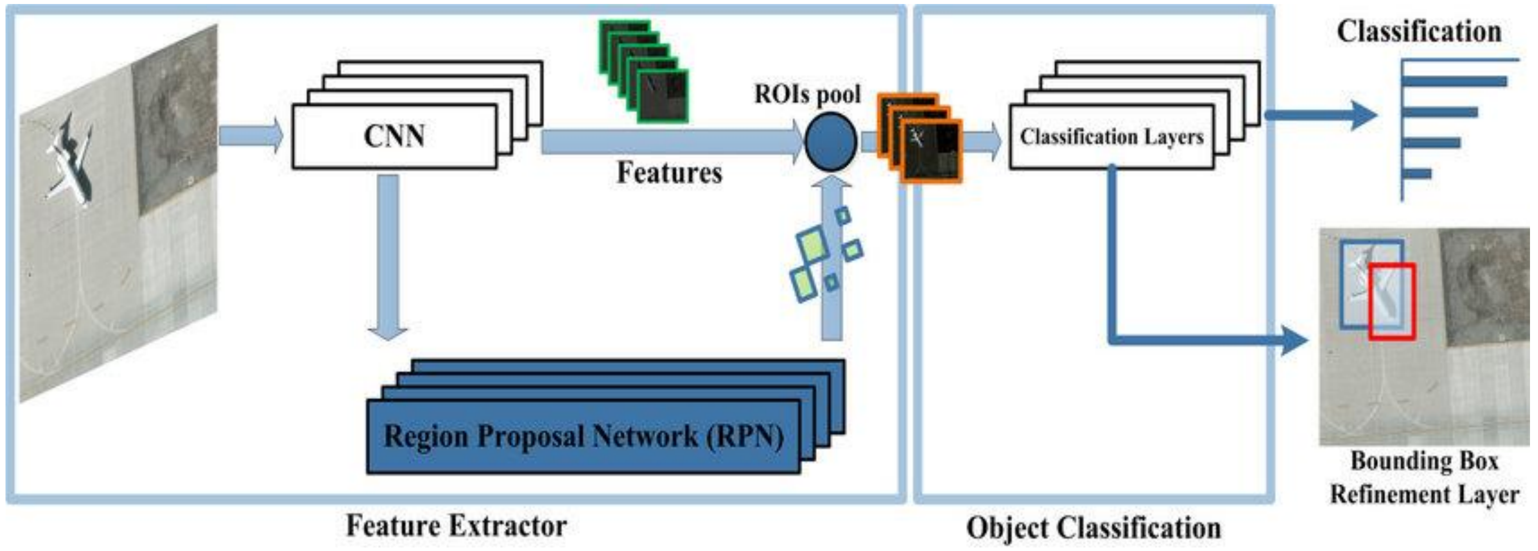
The RPN module guides the Fast R-CNN detection module to where to look for objects in the image.

<u>This allows us to reach our objective: perform image classification on a dataset characterized by a domain shift between training and test data.</u>



Faster R-CNN

We use Faster R-CNN model with a ResNet-50-FPN backbone (not pre-trained).



FASTERRCNN_RESNET50_FPN by Torchvision: link

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks paper: link

Dataset

During training, the model expects both the input tensors and a targets (list of dictionary), containing:

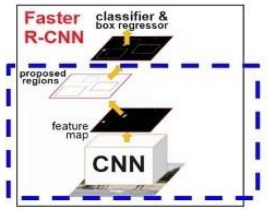
- boxes (FloatTensor[N, 4]): the ground-truth boxes in [x1, y1, x2, y2] format, with $0 <= x1 < x2 <= W$ and $0 <= y1 < y2 <= H$;
- labels (Int64Tensor[N]): the class label for each ground-truth box.

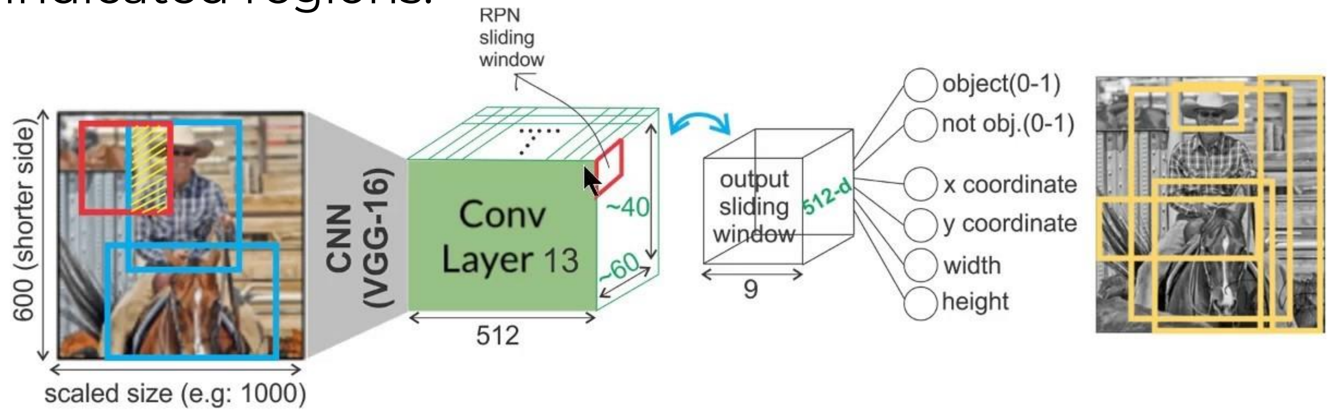
So, we implement from scratch `ObjectDataset` class – where target is a dictionary `{'boxes' : boxes, 'labels' :labels}`.

TORCHVISION TRANSFORMS V2 : link

Training

During training the model tries to achieve the objects in the images correctly thanks to the *ObjectDataset* which helps it to focus the learning in the indicated regions.





To train the model we used:

- The *SGD optimizer* with learning rate of 0.01
- *Dataset* that was split into *batches of size 16*
- *20 epochs*

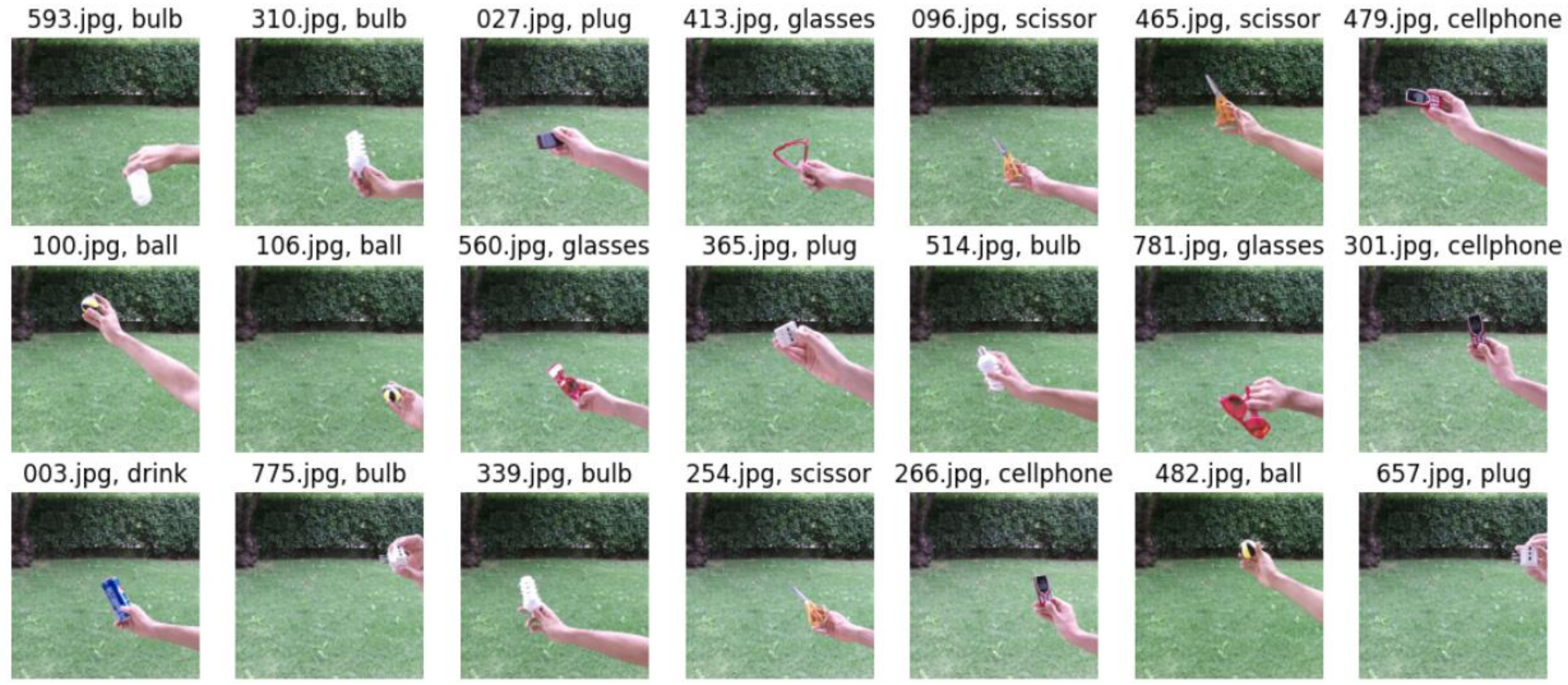
```
train_loader = DataLoader(train_dataset, batch_size = 16, num_workers = 0,
shuffle = True, drop_last = True, collate_fn = collate_fn)

# This is the line that start the train:
train(fasterrcnn, 20, dev, train_loader, 0.01)
```

Test

During inference, the model requires only the input tensors, and returns the post-processed predictions as a List[Dict[Tensor]], one for each input image.

The fields of the Dict are as follows, where N is the number of detections:

- boxes (FloatTensor[N, 4]): the predicted boxes in [x1, y1, x2, y2] format;
- labels (Int64Tensor[N]): the predicted labels for each detection;
- scores (Tensor[N]): the scores of each detection.



[3 scores of Kaggle]

Accuracy = 89.75 ± 0.88

THANK YOU!

Our work can be consulted at the following link.