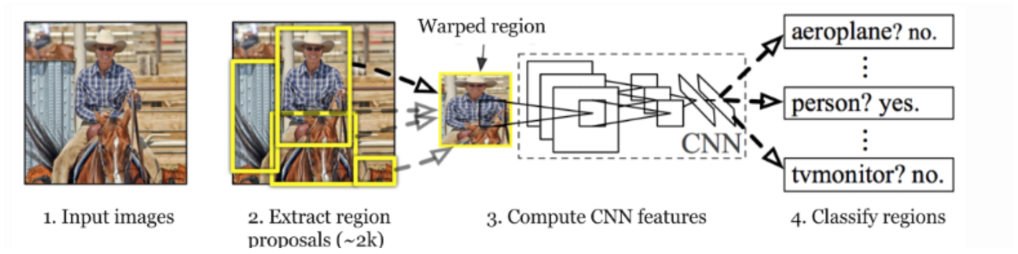
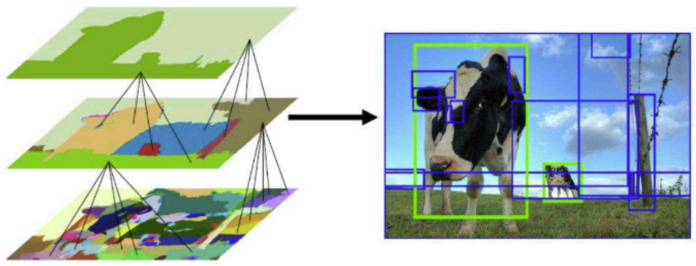
Experiment No 13

Aim: Implementation of RCNN

Ross Girshick et al in 2013 proposed an architecture called R-CNN (Region-based CNN) to deal with this challenge of object detection. This R-CNN architecture uses the selective search algorithm that generates approximately 2000 region proposals. These 2000 region proposals are then provided to CNN architecture that computes CNN features. These features are then passed in an SVM model to classify the object present in the region proposal. An extra step is to perform a bounding box regressor to localize the objects present in the image more precisely.



Region Proposals
Region proposals are simply the smaller regions of the image that possibly contains the objects we are searching for in the input image. To reduce the region proposals in the R-CNN uses a greedy algorithm called selective search.



Selective Search
Selective search is a greedy algorithm that combines smaller segmented regions to generate region proposals. This algorithm takes an image as input and output generates region proposals on it. This algorithm has the advantage over random proposal generation in that it limits the number of proposals to approximately *2000* and these region proposals have a high recall.

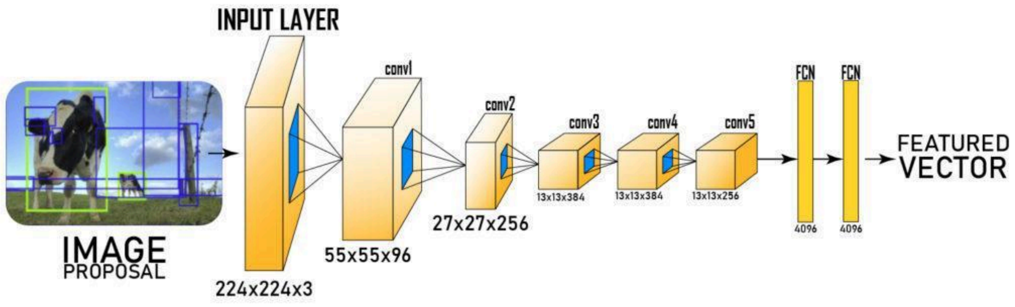
Algorithm

1. Generate initial sub-segmentation of the input image.
2. Combine similar bounding boxes into larger ones recursively
3. Use these larger boxes to generate region proposals for object detection.

In Step 2 similarities are considered based on color similarity, texture similarity, region size, etc.

CNN architecture of R-CNN
After that these regions are warped into a single square of regions of dimension as required by the CNN model. The CNN model that we used here is a pre-trained AlexNet model, which is the

state-of-the-art CNN model at that time for image classification Let's look at AlexNet architecture here.Here the input of AlexNet is *(227, 227, 3)*. So, if the region proposals are small and large then we need to resize that region proposal to given dimensions.
From the above architecture, we remove the last softmax layer to get the *(1, 4096)* feature vector. We pass this feature vector into SVM and bounding box regressor.

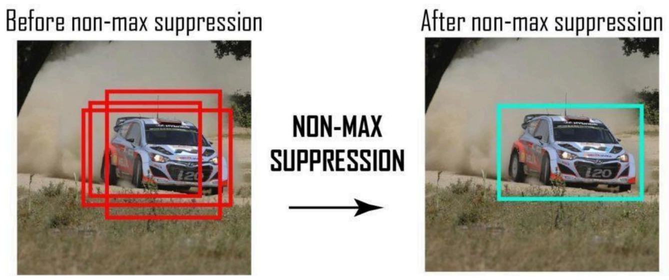


SVM (Support Vector Machine)
The feature vector generated by CNN is then consumed by the binary SVM which is trained on each class independently. This SVM model takes the feature vector generated in previous CNN architecture and outputs a confidence score of the presence of an object in that region. However, there is an issue with training with SVM is that we required AlexNet feature vectors for training the SVM class. So, we could not train AlexNet and SVM independently in paralleled manner. This challenge is resolved in future versions of R-CNN (Fast R-CNN, Faster R-CNN, etc.).

Bounding Box Regressor
In order to precisely locate the bounding box in the image., we used a scale-invariant linear regression model called bounding box regressor. For training this model we take as predicted and Ground truth pairs of four dimensions of localization. These dimensions are *(x, y, w, h)* where *x* and *y* are the pixel coordinates of the center of the bounding box respectively. w and

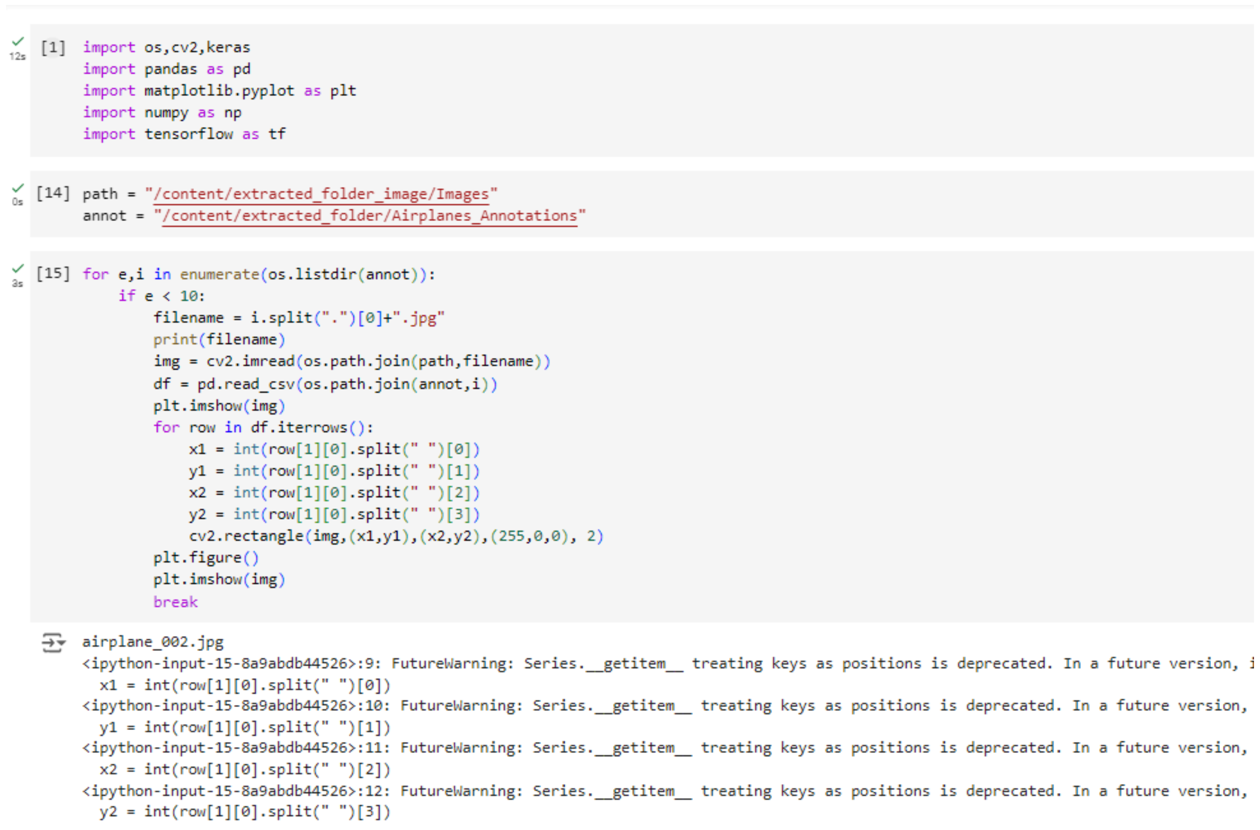
represent the width and height of bounding boxes. This method increases the Mean Average precision (mAP) of the result by *3-4%*.**Output:**
Now we have region proposals that are classified for every class label. In order to deal with the extra bounding box generated by the above model in the image, we use an algorithm called Non-maximum suppression**.** It works in 3 steps:

- Discard those objects where the confidence score is less than a certain threshold value*(say 0.5)*.
- Select the region which has the highest probability among candidates regions for the object as the predicted region.
- In the final step, we discard those regions which have IoU (intersection Over Union) with the predicted region over *0.5*.



After that, we can obtain output by plotting these bounding boxes on the input image and labeling objects that are present in bounding boxes.

Implementation

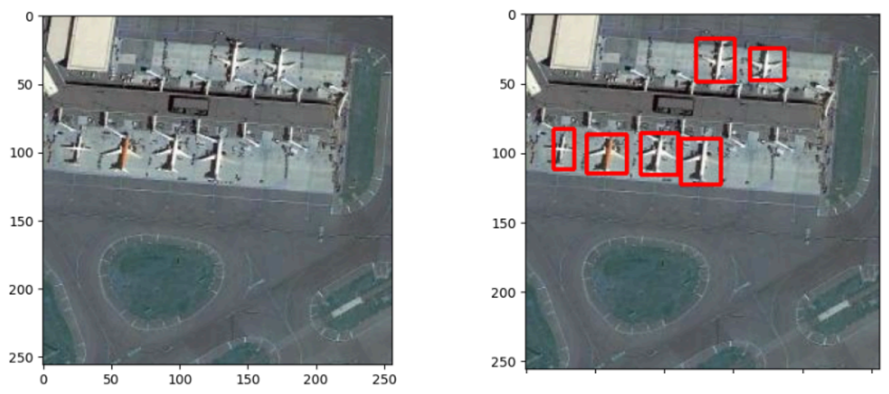


```
[1]  import os,cv2,keras
     import pandas as pd
     import matplotlib.pyplot as plt
     import numpy as np
     import tensorflow as tf
```

```
[14] path = "/content/extracted_folder_image/Images"
     annot = "/content/extracted_folder/Airplanes_Annotations"
```

```
[15] for e,i in enumerate(os.listdir(annot)):
         if e < 10:
             filename = i.split(".")[0]+".jpg"
             print(filename)
             img = cv2.imread(os.path.join(path,filename))
             df = pd.read_csv(os.path.join(annot,i))
             plt.imshow(img)
             for row in df.iterrows():
                 x1 = int(row[1][0].split(" ")[0])
                 y1 = int(row[1][0].split(" ")[1])
                 x2 = int(row[1][0].split(" ")[2])
                 y2 = int(row[1][0].split(" ")[3])
                 cv2.rectangle(img,(x1,y1),(x2,y2),(255,0,0), 2)
             plt.figure()
             plt.imshow(img)
             break
```

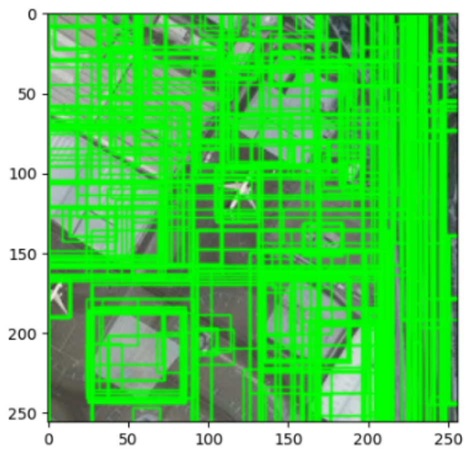
```
airplane_002.jpg
<ipython-input-15-8a9abdb44526>:9: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, i
  x1 = int(row[1][0].split(" ")[0])
<ipython-input-15-8a9abdb44526>:10: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version,
  y1 = int(row[1][0].split(" ")[1])
<ipython-input-15-8a9abdb44526>:11: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version,
  x2 = int(row[1][0].split(" ")[2])
<ipython-input-15-8a9abdb44526>:12: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version,
  y2 = int(row[1][0].split(" ")[3])
```



```
[16] ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
```

```
[17] im = cv2.imread(os.path.join(path,"42850.jpg"))
     ss.setBaseImage(im)
     ss.switchToSelectiveSearchFast()
     rects = ss.process()
     imOut = im.copy()
     for i, rect in (enumerate(rects)):
         x, y, w, h = rect
     #     print(x,y,w,h)
     #     imOut = imOut[x:x+w,y:y+h]
         cv2.rectangle(imOut, (x, y), (x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
     # plt.figure()
     plt.imshow(imOut)
```

```
<matplotlib.image.AxesImage at 0x7d2a15470280>
```



Advantages of RCNN

1. High Accuracy: Provides improved detection accuracy using deep learning for feature extraction.

2. Rich Feature Representation: Captures complex patterns, making it effective for various object types.

3. Flexibility: Adaptable to different datasets and object detection tasks.

4. End-to-End Learning: Optimizes feature extraction and classification simultaneously.

Disadvantages of RCNN

1. Slow Inference Time: Longer processing time makes it less suitable for real-time applications.

2. High Computational Cost: Requires significant resources, often needing GPUs for training.

3. Storage Requirements: Large storage needed for features of proposed regions.

4. Complex Training Process: Involves multiple steps, complicating implementation and tuning.

Conclusion

In this implementation of RCNN for object detection, we demonstrated its effectiveness in accurately identifying and localizing objects within images. Despite its computational intensity, the model's robust feature extraction capabilities lead to improved detection performance.