Perception

Q2 Pedestrian Detection

R-CNN model ?

Region based Convolutional Neural Network

Set of ML models for C.V..

- Takes an object & produces a bounding box which contains some info like type of object too.
- When you take all windows of all sizes & feed it to the model to classify object type it takes lots of time.
- We use RNN to tackle this. RCNN uses selective search.
- We use reading RCNN further

Faster RCNN

Improved version of RCNN which doesn't use selective search rather it uses Region Proposal Network. (RPN)

Basically Faster RCNN is faster than RCNN & Fast RCNN cuz it integrated the proposal generation step directly into the neural network. So region proposal & object classification both are done in the same model.

We pass the image into some convolutional layer & take the feature map & use sliding window of $n \times n$ size to go through the image to see if the region contains an object.

Working:

Extract features (making feature maps) → using pretrained CNN

↓

Use RPN to slide over the feature map & predict: ① Objectiness score
② Bounding box coordinates (refinements to anchors)

↓ predefined boxes of different scale & aspect ratios centered at each sliding window

↓

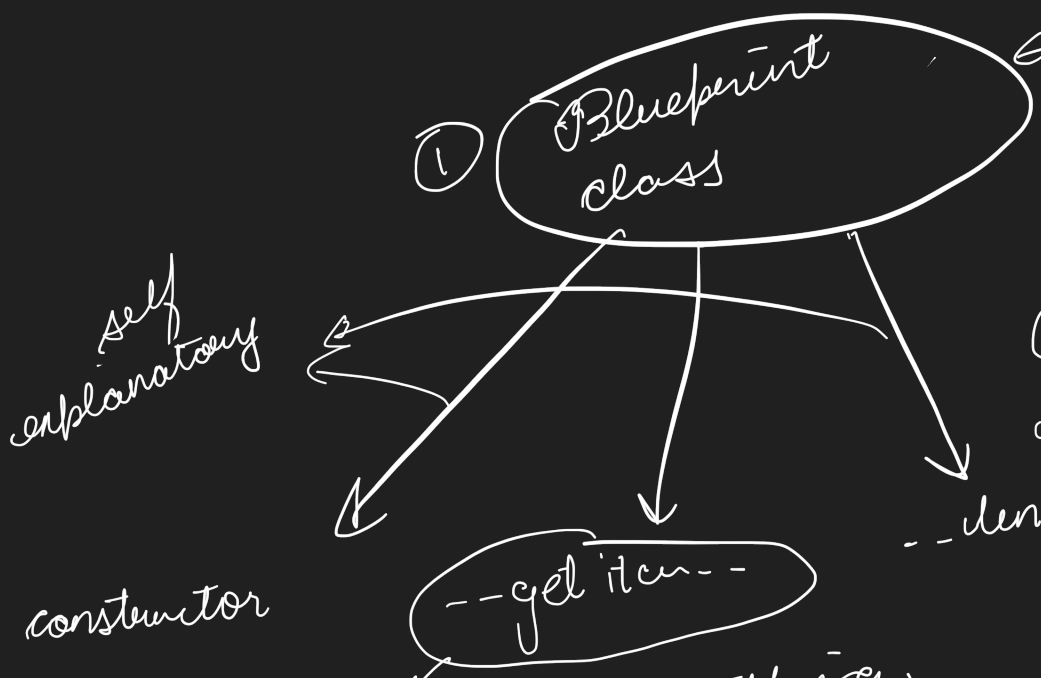
ROI pooling layer takes each proposal & corresponding section of feature map & convert it to fixed size feature map suz feature fully connected (FC) layers require fixed size input.

↓

Each proposed region is then classified into object categories using classification heads & bounding box's coordinates are refined using the bounding box regression head.

Implimentation

The model will be built using Python since that seems the common way.



most important thing.

tells PyTorch how to handle individual data. (in our case the pedestrian images)

every data has to be converted to Tensors before PyTorch can use it.

we return the input image & the target as tuples in this func^n.

2 Extra methods

for calling the model in main()

for grouping the data correctly

③ the main() method

- basically everything is done here
- we select device for training model
- divide dataset into → train (80%)
  → test (20%)
- use Dataloader to give data to model more easily
- set up the model training loop.
- call for evaluation & visualization functions

④ evaluating model

- the model has different modes (training & evaluation)
- MeanAverage Precision is the class which helps evaluate our model. Feed the test data results to it & the target & there you have it your precision values.
- map values are basically Intersection of Union (IOU) values

$$\frac{\text{area of overlap}}{\text{area of union}}$$

5) Visualizing predictions

- We calculate mAP scores for the model which basically tell how good the model is.
  - map, map_50 are most important
  - map_small for small objects
- We then display 5 random images using matplotlib & see how accurate the model is.