Problem Description:

Build the data

Use the following links to locally download the data:
Training and validation:
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar
Testing data:
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-Nov-2007.tar

The dataset consists of images from 20 classes, with detection annotations included. The JPEGImages folder houses the images, and the Annotations folder has the object-wise labels for the objects in one xml file per image. You have to extract the object information, ie. the [xmin, ymin] (the top left x,y coordinates) and the [xmax, ymax] (the bottom right x,y coordinates) of only the objects belonging to the given 20 classes (aeroplane, bicycle, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, train, TV). For parsing the xml file, you can import xml.etree.ElementTree for you.

Organize the data as follows:
For every image in the dataset, extract/crop the object patch from the image one by one using their respective coordinates:[xmin, ymin, xmax, ymax], resize the image to resnet_input, and store it with its class label information. Do the same for training/validation and test datasets.

Important
You also have to collect data for an extra background class which stands for the class of an object which is not a part of any of the 20 classes. For this, you can crop and resize any random patches from an image. A good idea is to extract patches that have low "intersection over union" with any object present in the image frame from the 20 Pascal VOC classes. The number of background images should be roughly around those of other class objects' images. Hence the total classes turn out to be 21. This is important for applying the sliding window method later.

Train the network

You can train the network on the created dataset. This will yield a classification network on the 21 classes of the VOC dataset.
Use the pre-trained network to fine-tune the network in the following section:

Testing and Accuracy Calculation

For applying detection, use a sliding window method to test the above trained trained network on the detection task:
Take some windows of varying size and aspect ratios and slide it through the test image (considering some stride of pixels) from left to right, and top to bottom, detect the class scores for each of the windows, and keep only those which are above a certain threshold value. There

is a similar approach used in the paper -Faster RCNN by Ross Girshick, where he uses three different scales/sizes and three different aspect ratios, making a total of nine windows per pixel to slide. You need to write the code and use it in testing code to find the predicted boxes and their classes.

Result To Show:

Print/Display the results/graphs/figures wherever possible.

1. After reading the images, display some images from the batch
2. Plot the training curve
3. Display the result for detected objects before and after NMS suppression
4. Plot the Precision-Recall curve
5. Effect of sliding window strategy (stride, aspect ratio, size) on detection performance - Try to make a plot for each

These are some of the plots/graphs that come to my mind. Feel free to plot some more if you can.

Resources:

Precision-Recall Understanding:
https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173