ECE 685D HW3

Deliverability: Please upload your Jupyter Notebook file (.ipynb file) with all outputs and necessary derivations exported as a PDF or HTML to Sakai. All necessary coding documentation can be found in `https://pytorch.org/docs/stable/index.html`. **We highly encourage you to submit your derivation in LaTeX. If you choose to submit a hand-written derivation, please make sure your derivation is legible. If you have hand-written solutions, please scan/take a photo and insert them into their designated space.** You can follow this tutorial to insert images to .ipynb files: `https://www.geeksforgeeks.org/insert-image-in-a-jupyter-notebook/`

1 Problem 1: Object detection with Convolutional Neural Networks (CNNs)

In this problem, you will train a classification model and (most part of) an object detection model based on the VOC 2012 train/val dataset. You may download the data at `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar`. Please split all eligible data into a training and validation set with size 4:1 for training and evaluation purpose respectively.

1.1 Data Preprocessing

1.1.1 Extracting bounding box (15 pts)

In most of the cases, the bounding box annotation for each image came in the format of .xml files or .json files. VOC2012 uses .xml files to save the bounding box for each annotated image. The .xml files are located in a directory named 'VOCdevkit/VOC2012/Annotations'. Please follow the examples in `https://docs.python.org/3/library/xml.etree.elementtree.html` to extract the class and coordinates of all object-level bounding boxes (enclosed by tag ¡object¿...¡/object¿) in each image.

Specifically, your code should **contain a function that takes the path of the .xml file as input and outputs a NumPy array with dimension** $N \times 5$. Here, N represents the number of target objects in the image; 5 means [class_id, x,y,w,h] (p40 of lecture slides)

1.1.2 Bounding box visualization(10 pts)

Now, let us overlay the bounding box and source images to verify the bounding boxes are correctly placed.

Your code should contain a function that **takes an image path and its corresponding NumPy array for the bounding boxes as input and plots a visualization of the bounding box on the original image**.

1.2 Classifier for "the presence of object" (bonus)

Then, we want to train a classifier whose task is to tell "whether there exist objects with "class_id = cls" in the image. The trained classifier would later be used for bounding box generation.

1.2.1 Multi-hot encoding (bonus 5pts)

Before we start the training, we have to prepare the ground truth label for training. When we train a classification model with just one correct class, one-hot encoding is used. However, since images in VOC2012 frequently contain obejcts from different classes, mult-hot encoding would be a more reasonable option. Please extract and prepare the multi-hot encoding for each image and subsequent training.

1.2.2 Model for the classification of presence (bonus 10 pts)

Now, given the multi-hot encoding, please train a CNN classifier of arbitrary architecture of your choice with reasonable classification performance. We will use this model for the subsequent object detection task.

1.3 RPN from scratch (40 pts)

Region Proposal Network (RPN) is an essential module in faster R-CNN. It is responsible for proposing candidate bounding boxes. Assuming you have access to a pre-trained CNN feature extractor, train an RPN that achieves discriminative region-proposal capability. You may use the Jupyter Notebook here as a reference: `https://colab.research.google.com/drive/1W1s7Xwe1MbEImNdMzSt6keNhLAdgOJwR?usp=sharing`. You may also use the resources link on the last page of the lecture slide in your implementation. Please make comments on your code. Your code should also include the Non-maximum Suppression (NMS) as a post-processing for the region proposals.

For the visualization of each image, please plot the image, the ground-truth box in green, and the positive region proposals post-processed with NMS in red in the same figure.

***Note: If you choose to do the optional section 1.2, the feature extractor should come from the model trained by you. Otherwise, it should come from some network pre-trained on ImageNet: `https://pytorch.org/vision/0.15/models.html`

2 Problem 2: Denoising AutoEncoders (35pts)

In this problem, you will implement a denoising autoencoder to restore some corrupted images. You may keep using the images from VOC2012 for this problem. You should split your data into the training and validation set for training and inference respectively.

Before the training, you should first generate corrupted images so that you have the input-output pairs looking like $(X_{corrupted}, X_{clean})$. You may apply image corruptions with the imagecorruptions library downloadable from `https://pypi.org/project/imagecorruptions/#description` for your convenience.

After training your model, please apply your model to the validation set, and pick a few output examples for visualization. The visualization should be triplets of (corrupted image, restored image, and original image) placed side-by-side for easier comparison.