Report: Red Round Sign Detection

Part 2: Training:

In this section, I will outline the **Faster RCNN** based architecture that I implemented for the given task. Due to lack of sufficient data, we use all the training instances for training the model. We use two types of **base neural network** to extract the feature map:

- A very shallow trainable network called FC-net with two convolutional and two max pool layers, with output number of features being 64 with RPN stride = 4.

- An similar implementation of VGG-net along with pre-trained weights downloaded from Keras. The output number of features is 512, with RPN stride = 8.

Next, after we obtain the the feature map from base network, we feed it along with number of anchors to an RPN network, which outputs the region proposals with probability of having an object (classification) and bounding boxes (regression). The **RPN** is implemented in a fully convolutional way:

- A convolutional layer with 3 x 3 filters and output channel = 512 that operates on the feature map produced by base network
- The output of convolutional layer is fed into two separate networks:
  - A 1 x 1 convolutional layer with (2 * number of anchors) as output channel (classification of each anchor as foreground or background)
  - A 1 x 1 convolutional layer with (4 * number of anchors) as output channel (regression to 4 bounding box coordinates)

Finally we implemented the classifier (detector) network on top of the RPN model, which takes into the feature map obtained from the base neural network, the Region of Interests (RoIs) proposed by RPN after Non maximum suppression and number of RoIs to process at a time. It gives the output prediction class (red round sign or background) along with refined bounding box coordinates of the proposals generated from RPN. If the IoU of the proposals with ground truth is above 0.5, we assign it to be positive class i.e. red round sign. We reject any proposals whose IoU is below 0.1, and assign the proposals with IoU between 0.1 and 0.5 as background. I believe we can modify these hard criteria in a better way to get good results, if I had enough time.

Training hyper-parameters:

- learning rate = 0.00001
- optimizer = Adam
- number of steps in each epoch = 600
- number of epochs = 10
- Anchor box sizes = [8, 16, 32, 64]
- Anchor box aspect ratios = [1:1, 1:2, 2:1]
- Number of RoIs for classifier to process at a time = 32
- RPN stride = 4 (FC-net) or 8 (VGG-net)

Here, I am training separately the two models (Base network + RPN) and the classifier, with the latter working on the output of RPN. With enough time, I could have experimented by training in an **end-to-end manner**, which usually give much better performances according to the original paper. I am monitoring the following attributes after each training epoch:

- Mean number of bounding boxes from RPN overlapping ground truth boxes
- Classifier accuracy for bounding boxes from RPN (foreground or background)
- Loss RPN classifier
- Loss RPN regression
- Loss Detector classifier
- Loss Detector regression
- Elapsed time

Pros:

- It is one of the best state of art object detector model existing in literature.
- It provides an interface for end-to-end learning, and the multi-task loss (cross entropy for classification + Smooth L1 for regression) helps in the training procedure.

Cons:

- Depending on the number of proposals we keep after RPN output, the running time varies a lot. The training and testing speed is a major concern for implementing Faster RCNN. In this context, SSD or YOLO can be a better option as their GPU time for training is much less.

Due to computational burden, the model with either FC-net or pretrained VGG-net couldn't be trained on my Mac, therefore, I used the free Tesla K80 GPU in Google

Colab for training purposes. Training for 10 epochs of the entire Faster RCNN model with FC-net takes more than 12 hours (1 epoch is around 7 hours). Since Colab has a hard restriction of 12 hours, I was able to train only 1 epoch of the models, with the following result:

```
597/600 [============================>.] - ETA: 1:18 - rpn_cls: 0.5331 - rpn_regr: -0.0531 - detector_cls: 0.0743 - de
598/600 [============================>.] - ETA: 52s - rpn_cls: 0.5328 - rpn_regr: -0.0531 - detector_cls: 0.0743 - det
599/600 [============================>.] - ETA: 26s - rpn_cls: 0.5325 - rpn_regr: -0.0531 - detector_cls: 0.0743 - det
 img 892: dataset/png_TrainIJCNN2013/00396.png
 img 893: dataset/png_TrainIJCNN2013/00124.png
600/600 [==============================] - 15675s 26s/step - rpn_cls: 0.5322 - rpn_regr: -0.0531 - detector_cls: 0.074
Mean number of bounding boxes from RPN overlapping ground truth boxes: 0.270996640538
Classifier accuracy for bounding boxes from RPN: 0.986041666667
Loss RPN classifier: 0.358274214444
Loss RPN regression: -0.060478361166
Loss Detector classifier: 0.0643924599764
Loss Detector regression: 0.0480640479375
Elapsed time: 15674.858593
Total loss decreased from inf to 0.410252361191, saving weights
Epoch 2/10
  0/600 [..............................] - ETA: 0s img 1: dataset/png_TrainIJCNN2013/00239.png
 img 2: dataset/png_TrainIJCNN2013/00264.png
 img 3: dataset/png_TrainIJCNN2013/00517.png
  1/600 [..............................] - ETA: 8:52:17 - rpn_cls: 0.2997 - rpn_regr: -0.2052 - detector_cls: 6.3517e-
  2/600 [..............................] - ETA: 5:59:49 - rpn_cls: 0.3338 - rpn_regr: -0.2177 - detector_cls: 5.8194e-
 img 6: dataset/png_TrainIJCNN2013/00281.png
  3/600 [..............................] - ETA: 5:53:49 - rpn_cls: 0.3054 - rpn_regr: -0.1982 - detector_cls: 0.0083 -
  4/600 [..............................] - ETA: 5:02:36 - rpn_cls: 0.2788 - rpn_regr: -0.1805 - detector_cls: 0.0202 -
```

As we can see, since the FC-net is a very shallow model, the mean number of bounding boxes from RPN overlapping with ground truth is much less than that we obtained using the pretrained VGG-net. During testing, we didn't get good performance at all. I tested with a few test images, and the model failed to detect any round traffic signs in those images. Maybe the performance would have got better if we trained more epochs.

With the pretrained VGG-net, however I was able to train for entire 10 epochs within 12 hours in Colab. The following is a snapshot after the training for 10 epochs:

```
598/600 [============================>.] - ETA: 10s - rpn_cls: 9.4719e-04 - rpn_regr: -0.2139 - detector_cls: 0.0173 -
599/600 [============================>.] - ETA: 5s - rpn_cls: 9.4721e-04 - rpn_regr: -0.2139 - detector_cls: 0.0173 -
600/600 [==============================] - 3283s 5s/step - rpn_cls: 9.4723e-04 - rpn_regr: -0.2139 - detector_cls: 0.(
Mean number of bounding boxes from RPN overlapping ground truth boxes: 6.745
Classifier accuracy for bounding boxes from RPN: 0.994114583333
Loss RPN classifier: 0.000960719006923
Loss RPN regression: -0.215013930742
Loss Detector classifier: 0.0185927108617
Loss Detector regression: -0.386346782632
Elapsed time: 3290.84462905
Total loss decreased from -0.565707834316 to -0.581807283505, saving weights
Training complete, exiting.
Trained on 10 epochs in 8 hr 41 min
```

To evaluate the model performance, I calculated the mean average precision (mAP) over all classes (red round sign or background) across all training images. However, for prediction, each image takes around 18-20 seconds, and due to time constraint I am

unable to report the final mAP over all training images. Below, I have attached some of the good detections obtained using VGG-net on the test images which do not have ground truth prediction boxes. The predictions are annotated with the class label (1 for red round sign) along with prediction probability.

We clearly see the effect of lighting conditions on the predictions, with the probability of prediction being lower in the second image. Also, we see that the model is able to distinguish between red round signs and blue round signs, as evident from the first image. Given more time, I could investigate more qualitative insights on the results obtained.

If I had more time, I would consider experimenting with the hyper-parameters such as learning rate, anchor box sizes and RPN strides and monitor the performance of the model. Experimenting with different optimizers such as Adam, Adagrad etc can also be explored. Since, I didn't have time to design other model architectures, I would try to outline other base neural network structures and inspect their performance on the training data. Unless we try to change the problem formulation and detect more than 1 class, we do not need to go back to Part 1 for performance improvement.