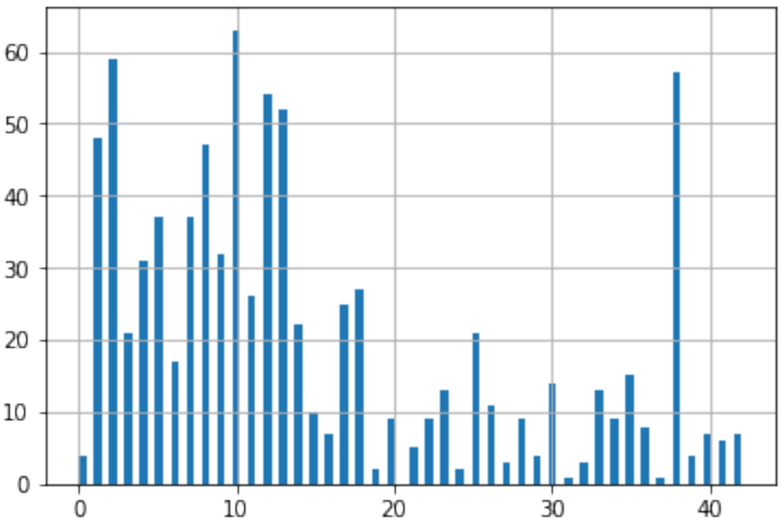
Report: Red Round Sign Detection

Part 1: Dataset:

The given dataset is the German Traffic Signs Detection Benchmark (GTSDB) dataset. The training dataset features:

- 600 total images, each of 1360 x 800 pixels in PPM format
- the image sections that contain the traffic signs
- a file in CSV format containing ground truth labels

The testing dataset features only 300 total images of same size and same format, with no ground truth labels.
There are a total 852 instances of different traffic signs across all 600 images in training data, with uneven distribution of 43 classes, shown below:



All images are read from the PPM format, and converted to PNG format, which is an extremely efficient lossless compression technique. The original dataset contains around 43 classes of different traffic signs, but our objective is to detect only red round signs in the images. The red round signs fall under the “prohibitory” class, and it corresponds to the following labels:

Red round signs = [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16]
Therefore, I changed the ids of all images that contain red round traffic signs to 1, and removed all rest of images. Now we have **396 positive class (red round sign) across**

264 unique training images. Now, the problem can be formulated as single class traffic sign detection problem, where the task to identify whether the traffic sign is a red round sign. We achieve this by implementing a **Faster RCNN** architecture, details of which are in Report 2.

Now, we parse the modified annotation text file (gt_train.txt) and store the data into various dictionaries, which we will later feed into data generators for training. Due to insufficient amount of data, we use all training instances to train our model. In the file parse_data_to_dict.py, we create a dictionary class_count that stores the number of training instances per class, and another dictionary named all_imgs that encodes all the information from each training image. For every image, it stores the filepath, width, height and list of bounding boxes with class id. Finally we add an instance of background class (bg) in both the dictionaries. Similarly, we modify the test images into a new dictionary, without the bounding boxes and class id.

We resize every image from 1360 x 800 to 1020 x 600, so as to reduce the computational burden of training the model. We also zero center each image by subtracting the channel means across 3 colour channels. In Faster RCNN, along with this, we need to generate ground truth anchors for the feature map computed using the base neural network. We use a RPN stride of 4 for FC-net and a stride of 8 for VGG-net as the base model. This means that for every pixel in the feature map where we generate the anchors, in the original image, the anchors are situated specified stride apart. The ground truth positive anchors are those that overlap with the ground truth bounding boxes above the specified RPN max overlap threshold (0.7), and the negative anchors are those whose overlaps with BB boxes are below RPN min overlap threshold (0.3). We do not include anchors in the objective that fall between 0.7 and 0.3. One issue is that the RPN has many more negative than positive regions, so we turn off some of the negative regions and limit it to 256 regions. We use different anchor sizes [8, 16, 32, 64] and different aspect ratios [1:1, 1:2, 2:1]. Therefore, total number of anchors we consider for every pixel in the feature map is 12.

To train the RPN, a multi-task loss function is used. The output of RPN consists of two parts, classification loss and regression loss. The classification loss is a log loss of two classes which encodes the confidence of whether the region proposal is an object or not. The regression loss is smooth L1 loss which represents the four coordinates of the proposed bounding box. I tried to experiment on various sets of configurations and models to explore and determine the performance on the dataset, but due to time constraint, I couldn't.

We notice that the total number of images is much less than that required for training a standard Faster RCNN model. One way to tackle this to perform Image Augmentation during training. We allow three types of **random augmentation** on every training image fed to the model:

- Horizontally flipping
- Vertically flipping
- Rotating the image randomly in 90, 180 and 270 degrees

This partially alleviates the problem of having less data for training purposes, and prevents early overfitting. Another way of tackling problems with very less data is to use the knowledge of Transfer learning. According to this technique, models trained on one task capture relations in the data type and can easily be reused for different problems in the same domain. With transfer learning, we can take a pretrained model, which was trained on a large readily available dataset. Then try to find layers which output reusable features. We use the output of that layer as input features to train a much smaller network that requires a smaller number of data points.

Unlike this GTSDB dataset, there exists a much bigger and better dataset known as LISA Traffic Sign dataset, that consists of set of videos and annotated frames containing US traffic signs. It consists of 7855 annotations of traffic signs in 6610 frames, with varying sizes. More interestingly, images are obtained from different cameras, with varying size, position, occlusion and illumination. Another interesting dataset that we can explore is the CURE-TSR dataset, which consists more than two million traffic sign images that are captured in controlled challenging conditions. I believe that working on these type of datasets will really help understand how to effectively tackle diverse images and traffic signs and come up with better deep learning algorithms to solve the detection / classification problem.

In unconstrained environment conditions, object detection / recognition becomes a challenging task. The huge variation in object scale, orientation, category, and complex backgrounds, as well as the different camera sensors pose great challenges for current algorithms. Most importantly, object detection and tracking at night remain very important problems for visual surveillance. If scene is completely dark, then it might be necessary to use a thermal infrared camera, which can prove to be extremely costly. The images captured by traditional car cameras have low brightness, low contrast, low signal to noise ratio (SNR) and nearly no color information. One way we can handle this is by detecting the object using local contrast computed over the entire image. This can be done by measuring sub-image inter-frame differences. Then we can track the detected objects and remove falsely detected objects with the help of a tracking algorithm, such as Kalman filter. Hand crafted feature extraction algorithms such as Local Binary Pattern (LBP) and Histogram of Oriented Gradients (HOG) are partially invariant to image distortions and illumination changes, thus can be explored under dark environment conditions.

Also, there are many state-of-art deep learning algorithms such as RCNN, Fast RCNN, Mask RCNN which were introduced over the years to tackle such problems. Additionally, YOLO and SSD are two popular frameworks that can handle complex object detection tasks. YOLO architecture is more like a fully connected CNN, which splits the input image into multiple fixed sized grids, and for each grid generates 2 bounding boxes and class probabilities for those bounding boxes. Another separate technique involves feature pyramid networks (FPN) that extracts feature maps through a feature pyramid, thus facilitating object detection in different scales, at a marginal extra cost.