Object Detection using Faster RCNN

Fast R-CNN passes the entire image to ConvNet which generates regions of interest (instead of passing the extracted regions from the image). Also, it uses a single model which extracts features from the regions, classifies them into different classes, and returns the bounding boxes. All these steps are done simultaneously, thus making it execute faster as compared to R-CNN. Fast R-CNN is, however, not fast enough when applied on a large dataset as it also uses selective search for extracting the regions.

Faster R-CNN fixes the problem of selective search by replacing it with Region Proposal Network (RPN). We first extract feature maps from the input image using ConvNet and then pass those maps through a RPN which returns object proposals. Finally, these maps are classified and the bounding boxes are predicted.

Below the steps followed by a Faster R-CNN algorithm to detect objects in an image:

1. Take an input image and pass it to the ConvNet which returns feature maps for the image
2. Apply Region Proposal Network (RPN) on these feature maps and get object proposals
3. Apply ROI pooling layer to bring down all the proposals to the same size
4. Finally, pass these proposals to a fully connected layer in order to classify any predict the bounding boxes for the image

Anchor boxes

Anchors play an important role in Faster R-CNN. An anchor is a box. In the default configuration of Faster R-CNN, there are 9 anchors at a position of an image. The following graph shows 9 anchors at the position (320, 320) of an image with size (600, 800). We choose self.anchor_box_scales = [320, 440, 550] and self.anchor_box_ratios = [[1, 1], [1./math.sqrt(2), 2./math.sqrt(2)], [2./math.sqrt(2), 1./math.sqrt(2)]].

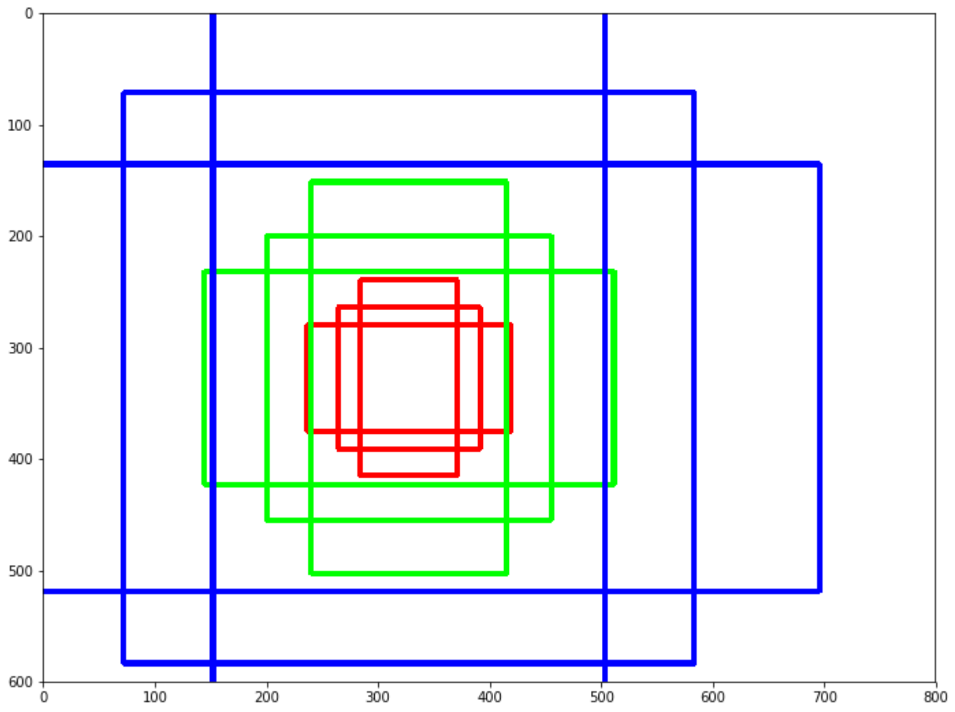


Figure: Anchors at (320, 320)

I have used the model present at https://github.com/kbardool/keras-frcnn to train Faster RCNN in our case. For the model to train we need to provide object's x-y coordinates (minimum and maximum). This is done by passing a .txt file.

There are 6 columns in the train file. Let's understand what each column represents:

1. image_names: contains the name of the image
2. cell_type: denotes the type of the cell
3. xmin: x-coordinate of the bottom left part of the image
4. xmax: x-coordinate of the top right part of the image
5. ymin: y-coordinate of the bottom left part of the image
6. ymax: y-coordinate of the top right part of the image

To get x-y coordinates, we used contour selection of open cv. I designed a function **get_canvas** which returns all these column values for each image. Function provided near approx estimate of object boundary but for few cases I had to manually present the x-y coordinates using http://www.robots.ox.ac.uk/~vgg/software/via/via-1.0.6.html.

```
def get_canvas(image_path):
    img = cv2.imread(image_path)
    if 'Healthy' in image_path:
        image_cls = 'Healthy'
    else:
        image_cls = 'Defective'

    img = cv2.resize(img, (img.shape[0]//2, img.shape[1]//2))
    if img.shape[0] > img.shape[1]:
        img = rotate_image(img)
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h,s,v = cv2.split(hsv)

    s = s*1.22
    s = np.clip(s,0,255).astype("uint8")

    ##(3) threshold the S channel using adaptive method(`THRESH_OTSU`) or fixed thresh
    _, threshed = cv2.threshold(s, 82, 140, cv2.THRESH_BINARY_INV)

    ##(4) find all the external contours on the threshed S
    cnts = cv2.findContours(threshed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]

    ## sort and choose the largest contour
    cnts = sorted(cnts, key = cv2.contourArea)
    cnt = cnts[-1]
    # print('contour area is : ', cv2.contourArea(cnt))
    # Contour Approximation: approx the contour, so the get the corner points
    epsilon = 0.003*cv2.arcLength(cnt,True)
    cnt = cv2.approxPolyDP(cnt,epsilon,True)

    xmin = max((cnt[cnt[:, :, 0].argmin()][0][0]) - 5, 0)
    ymin = max((cnt[cnt[:, :, 1].argmin()][0][1]) - 5, 0)
    xmax = min((cnt[cnt[:, :, 0].argmax()][0][0]) + 5, img.shape[0])
    ymax = min((cnt[cnt[:, :, 1].argmax()][0][1]) + 5, img.shape[1])
    cv2.imwrite('./TrainData/' + image_path.split('/')[-1], img)

    return './TrainData/' + image_path.split('/')[-1], xmin, ymin, xmax, ymax, image_cls

def rotate_image(image, angle=90):
    return imutils.rotate_bound(image, angle)
```

To train the model, we choose 70 epochs and 32 RoIs to process at once. Since, I did not have access to GPU, I used Google Colab to train the model. Each epoch was taking close to 8 minutes to complete and it took close to 10 hours to complete the training.

```
# start training

!python /content/RCNN-JBM/keras-frcnn/train_frcnn.py  -o simple --num_epochs 70 -n 32 -p RCNN-JBM/Fast\ RCNN/annotate1.txt

# -- input_weight_path gdrive/My Drive/ModelJBM/model_frcnn.hdf5 (if model is already trained and you want to train further, provide the saved model location)
# --num_epochs to set the number of epochs
```

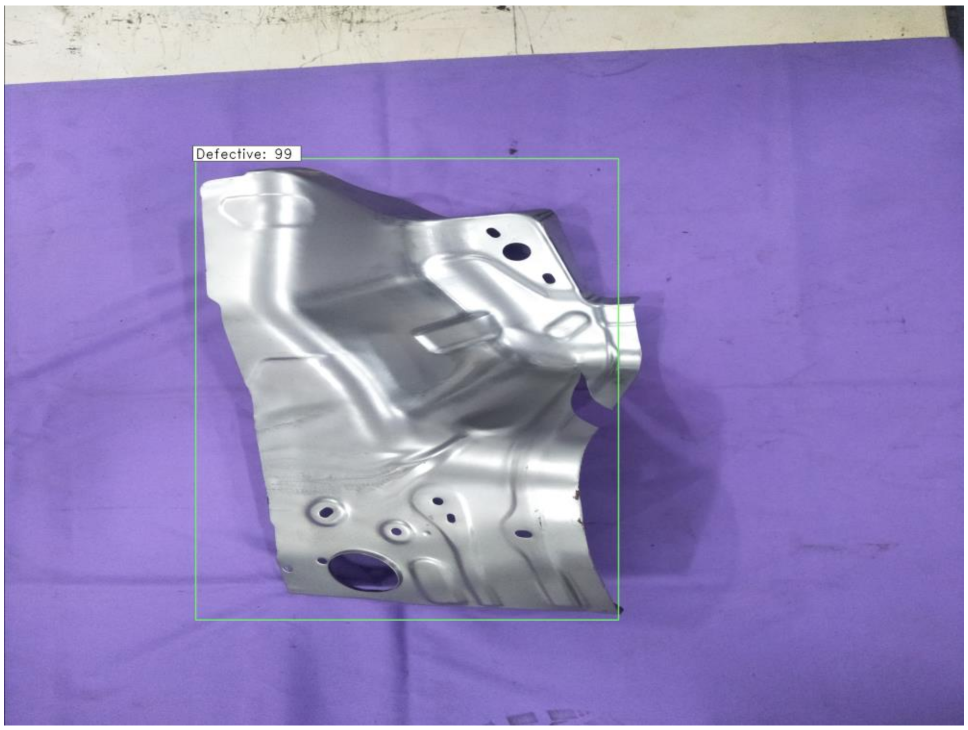
To test the model, I selected 9 images from each category to test.

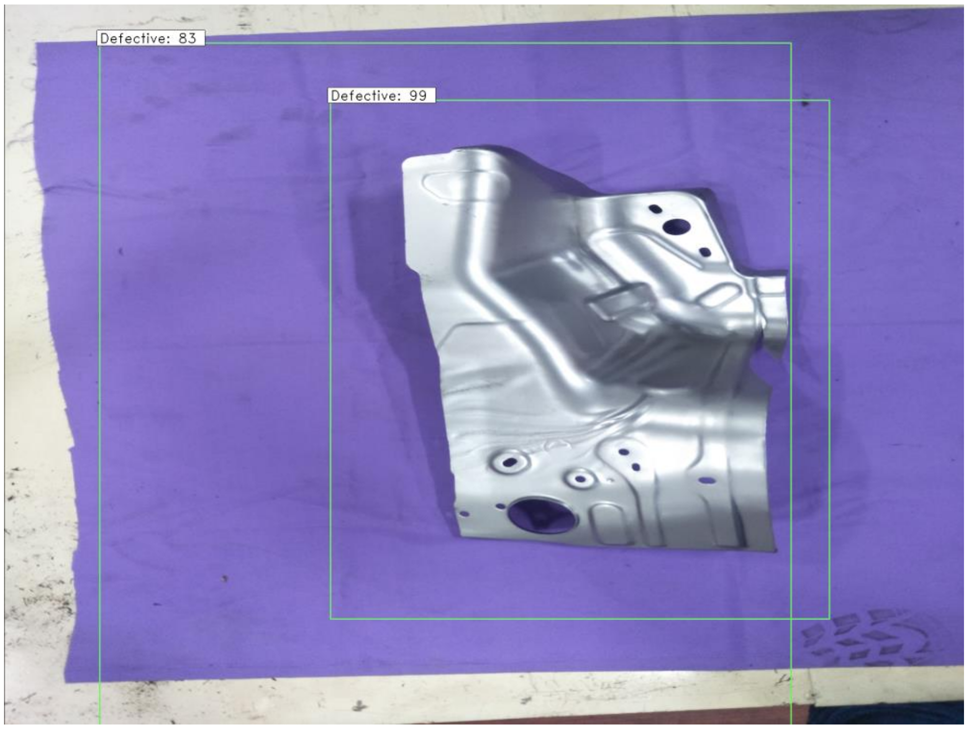
Model predicted 8 out of 9 defective images correctly and 7 out of 9 healthy images correctly.

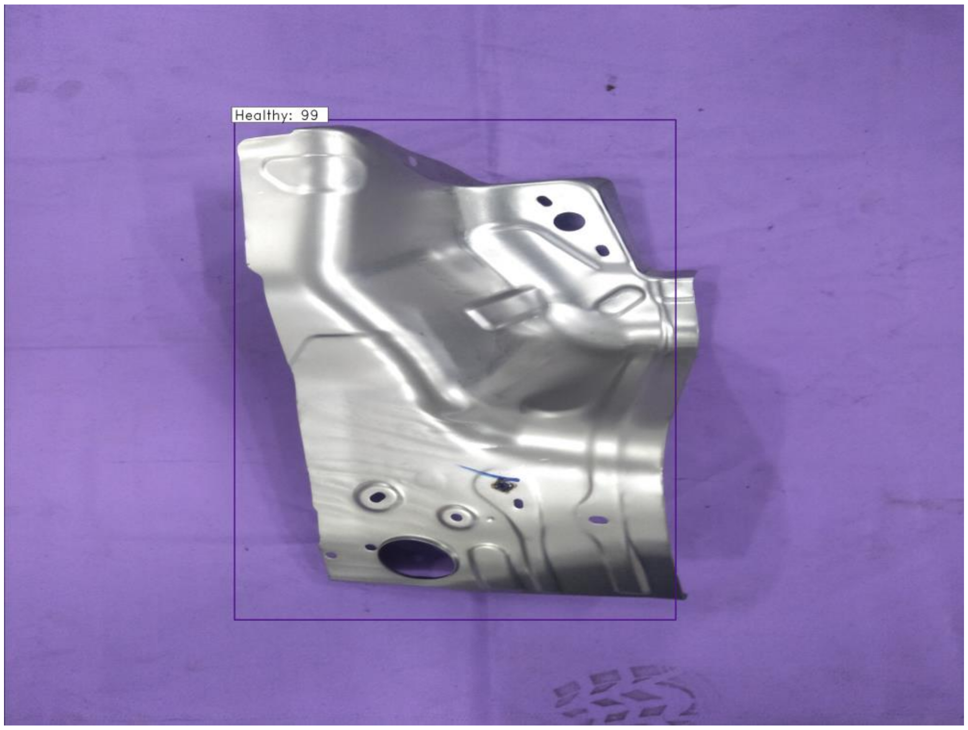
```
# start testing
!python /content/RCNN-JBM/keras-frcnn/test_frcnn.py -p /content/gdrive/My\
Drive/ModelJBM/Test/Healthy
```

Results: -

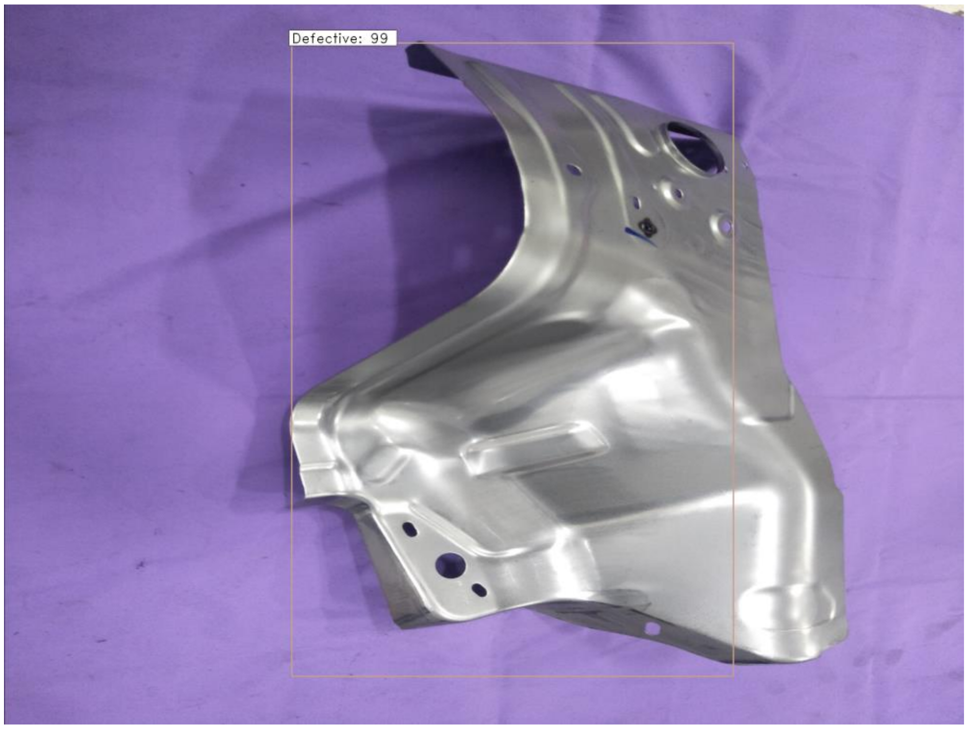
Correctly classified







Misclassified image example:



Faster RCNN does two things: Classification using SoftMax layer and bounding box coordinates prediction using regression.

As we can see from result images, model is handling classification at a very decent rate (17/20) but box surrounding the object is not completely covering the object. This should be next goal for model improvement. We will need to play more with the anchor box sizes, ratios, epoch numbers etc. hyperparameters.

Github Location: **https://github.com/ravijp/RCNN-JBM**

References:-

1. https://www.analyticsvidhya.com/blog/2018/11/implementation-faster-r-cnn-python-object-detection/
2. https://github.com/kbardool/keras-frcnn
3. https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8
4. https://www.analyticsvidhya.com/blog/2018/07/building-mask-r-cnn-model-detecting-damage-cars-python/