Table of Contents

[Introduction/Challenge Description 2](#_Toc25930404)

[Proposed solution 2](#_Toc25930405)

[Network Architecture(RetinaNet) 3](#_Toc25930406)

[Feature Pyramid Network 3](#_Toc25930407)

[Focal Loss 4](#_Toc25930408)

[The Backbone network 5](#_Toc25930409)

[classification subnet 8](#_Toc25930410)

[Regression Subnet 9](#_Toc25930411)

[a closer look at the subnet 9](#_Toc25930412)

[Matching prediction with ground truth 11](#_Toc25930413)

[The Loss Function 12](#_Toc25930414)

[Regression loss 12](#_Toc25930415)

[Classification Loss 13](#_Toc25930416)

[prediction 14](#_Toc25930417)

[installation 15](#_Toc25930418)

[building dataset 16](#_Toc25930419)

[Anchors parameters 21](#_Toc25930420)

[Some Hyperparameters 22](#_Toc25930421)

[Image Augmentation 23](#_Toc25930422)

[Visualize augmentations 27](#_Toc25930423)

[More Hyperparameters 27](#_Toc25930424)

[Callbacks 28](#_Toc25930425)

[Download pretrained model 28](#_Toc25930426)

[Loading the model 29](#_Toc25930427)

[Training the Model 29](#_Toc25930428)

[Inference/Prediction 29](#_Toc25930429)

[Predict Image 33](#_Toc25930430)

[Total Map Achieved from Training Model is 39% 35](#_Toc25930431)

[Experiment and Results 36](#_Toc25930432)

[Future Work 37](#_Toc25930433)

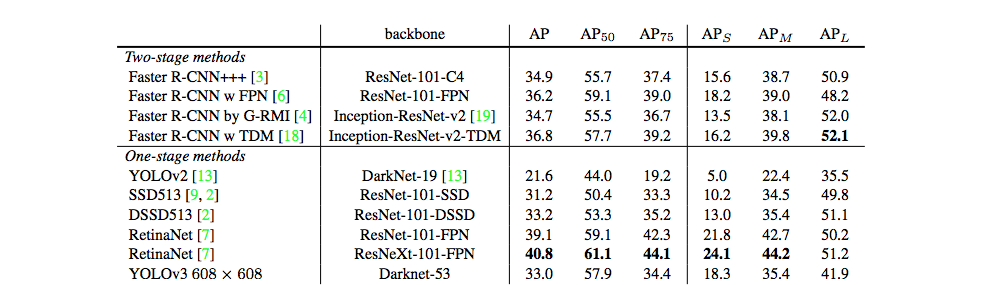
[References 37](#_Toc25930434)

# Introduction/Challenge Description

Development of object detection model to detect a smaller object/s in a given image.

# Proposed solution

I found several popular detectors including: OverFeat (Sermanet et al. 2013), R-CNN (Girshick et al. 2013), Fast R-CNN (Girshick 2015), SSD (Liu et al. 2016), R-FCN (Dai et al. 2016), YOLO (Redmon et al. 2016), Faster R-CNN (Ren et al. 2017) and RetinaNet (Lin, Goyal, et al. 2017). According to the paper, RetinaNet showed both ideal accuracy and speed compared to other detectors for small object detection while still keeping a very simple construct; plus, there is an [opensource implementation](https://github.com/fizyr/keras-retinanet) by Gaiser et al. (2018). Therefore, RetinaNet appears to be an ideal candidate for this project.



Retinanet Results on MS COCO Dataset

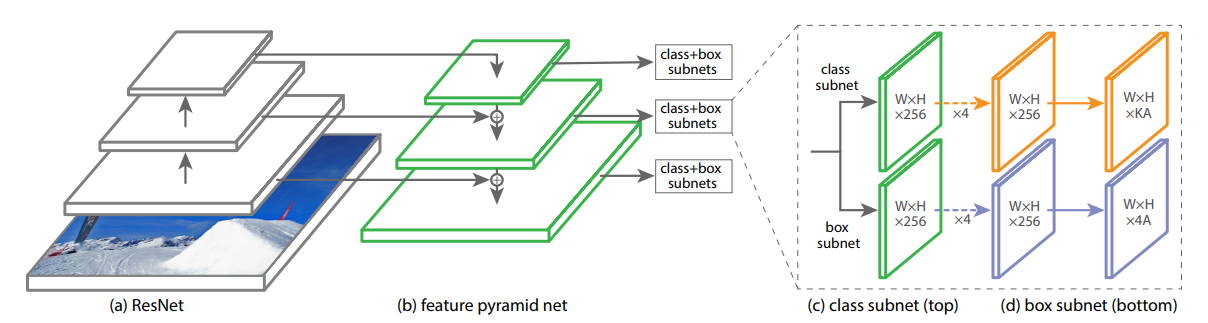
# Network Architecture(RetinaNet)

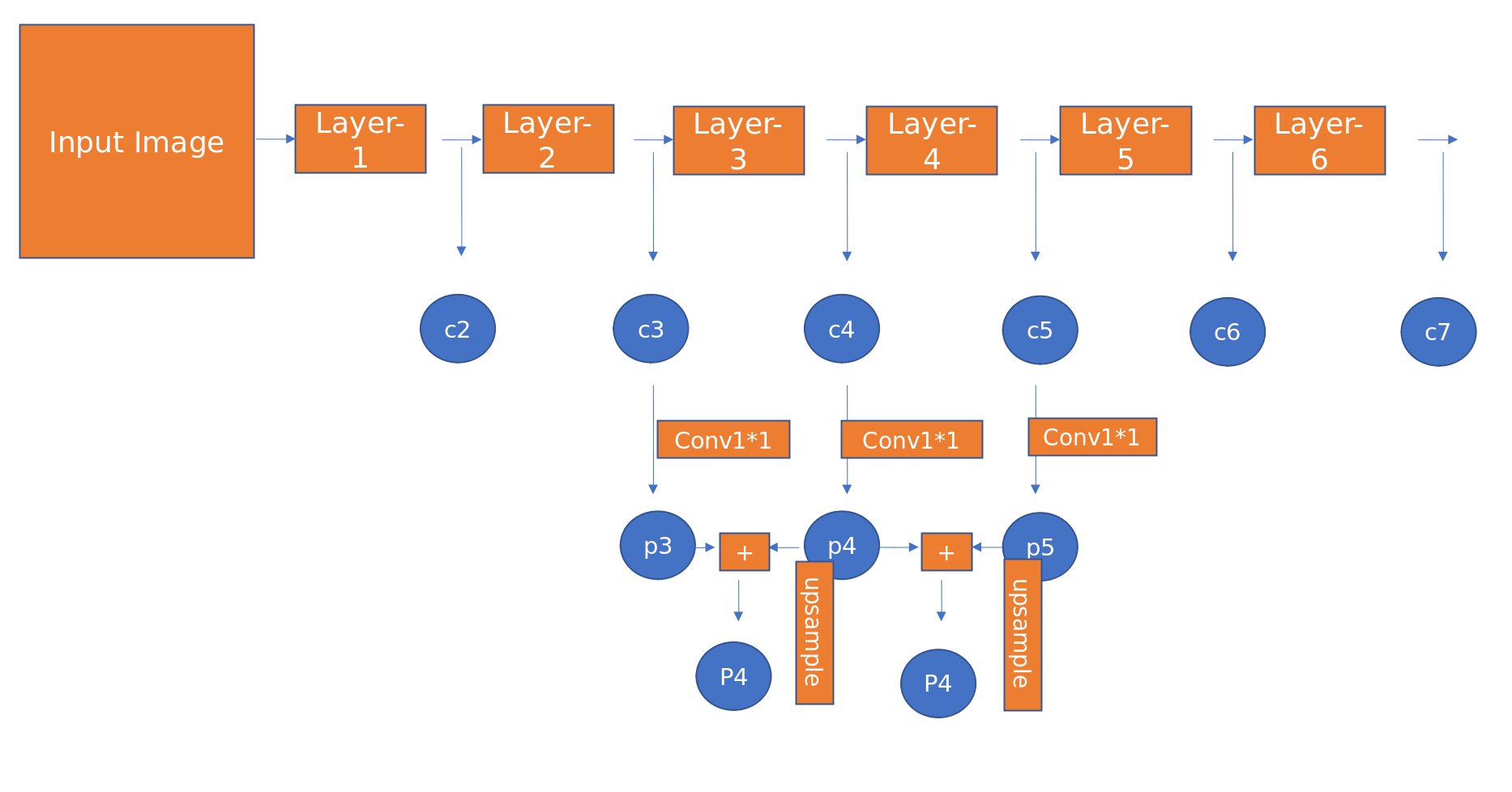
RetinaNet has been formed by making two improvements over existing single stage object detection models (like YOLO and SSD):

1. [Feature Pyramid Networks for Object Detection](https://arxiv.org/abs/1612.03144)
2. [Focal Loss for Dense Object Detection](https://arxiv.org/abs/1708.02002)

## Feature Pyramid Network

Pyramid networks have been used conventionally to identify objects at different scales. A Feature Pyramid Network (FPN) makes use of the inherent multi-scale pyramidal hierarchy of deep CNNs to create feature pyramids.





The one-stage RetinaNet network architecture uses a Feature Pyramid Network (FPN) backbone on top of a feedforward ResNet architecture (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). The network design is intentionally simple, which enables this work to focus on a novel focal loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN while running at faster speeds.

## Focal Loss

Focal Loss is an improvement on cross-entropy loss that helps to reduce the relative loss for well-classified examples and putting more focus on hard, misclassified examples.

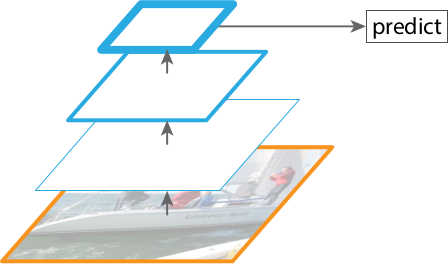
The focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.



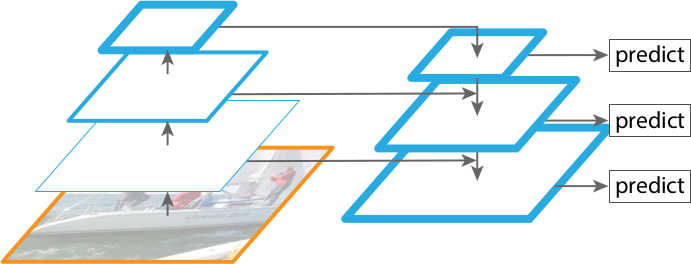
Focal Loss Function

# The Backbone network

* **Bottom-up pathway**. In ResNet, some consecutive layers may output feature maps of the same scale; but generally, feature maps of deeper layers have smaller scales/resolutions. The bottom-up pathway of building FPN is accomplished by choosing the last feature map of each group of consecutive layers[2](https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/#fn2) that output feature maps of the same scale. These chosen feature maps will be used as the foundation of the feature pyramid. The bottom-up pathway is visualized below



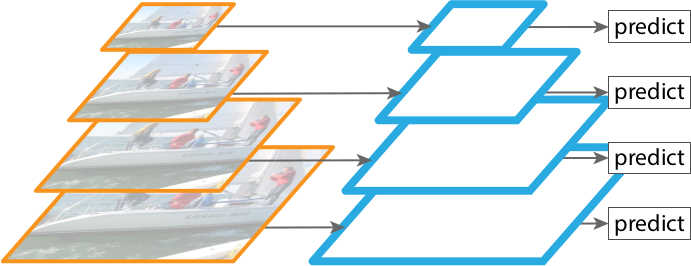
* **Top-down pathway and lateral connections**. Using nearest neighbour up sampling, the last feature map from the bottom-up pathway is expanded to the same scale as the second-to-last feature map. These two feature maps are then merged[3](https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/#fn3) by element-wise addition to form a new feature map. This process is iterated until each feature map from the bottom-up pathway has a corresponding new feature map connected with lateral connections. The top-down pathway and lateral connections are visualized in below figure.



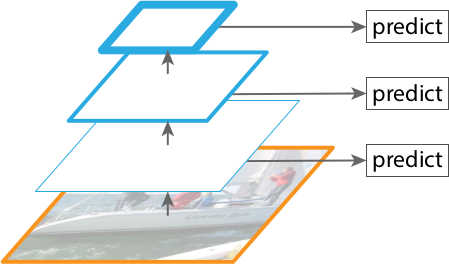
There are altogether five levels in the pyramid (the figures only shows three) denoted as P3,...,P7P3,...,P7, where l2 has resolution 2times lower than the input. The intuition behind FPN are described as below.

In real-world object detection, objects from the same class may be presented in a wide range of scales in images. This leads to some decrease in detection accuracy, especially for small objects. This is because feature maps from higher levels of the pyramid are spatially coarser, though semantically stronger. Therefore, only using the last feature map of a network to make the prediction is less ideal.

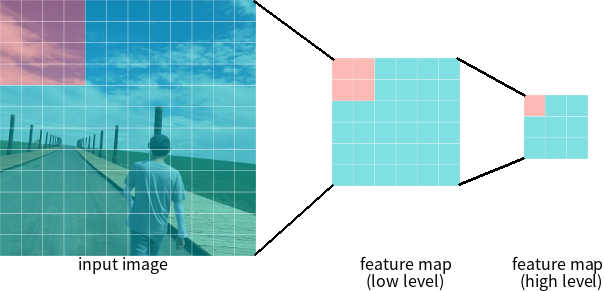
One solution would be to generate different scales of an image and feed them to the network separately for prediction This approach is termed “feature pyramids built upon image pyramids” and was widely adopted before the era of deep learning. However, since each image needs to be fed into the network multiple times, this approach also introduces a significant increase in test time, making it impractical for real-time applications.



Another solution would be to simply use multiple feature maps generated by a ConvNet for prediction (Fig. 5), and each feature map would be used to detect objects of different scales. This is an approach adopted by some detectors like SSD. However, although the approch requires little extra cost in computation, it is still sub-optimal since the lower feature maps cannot sufficiently obtain sementical features from the higher ones.

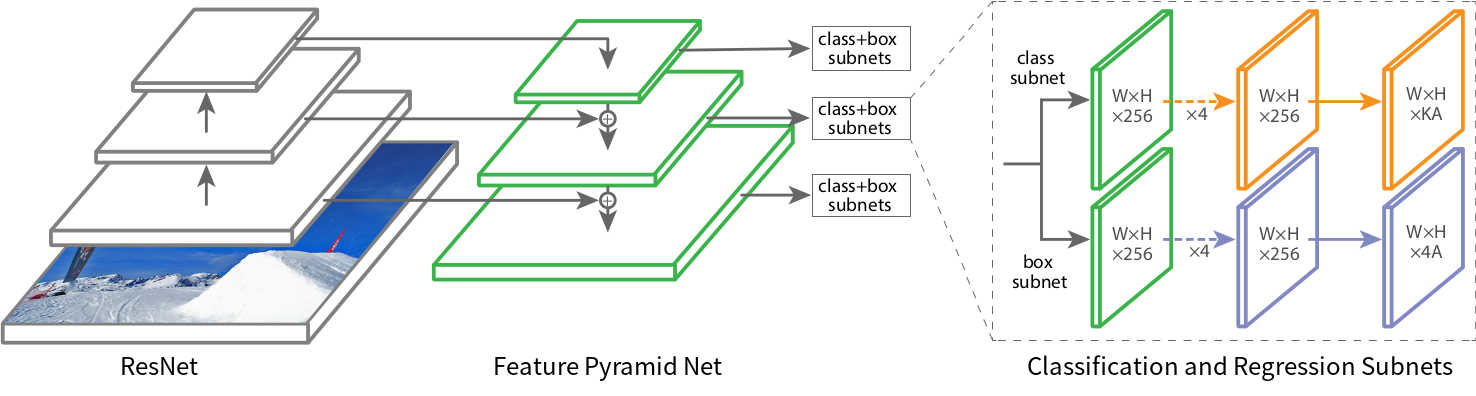


Finally, we turn to FPN. As mentioned, FPN is built in a fully convolutional fashion which can take an image of an arbitrary size and output proportionally sized feature maps at multiple levels. Higher level feature maps contain grid cells that cover larger regions of the image and is therefore more suitable for detecting larger objects; on the contrary, grid cells from lower level feature maps are better at detecting smaller objects (see below figure). With the help of the top-down pathway and lateral connections, which do not require much extra computation, every level of the resulting feature maps can be both semantically and spatially strong. These feature maps can be used independently to make predictions and thus contributes to a model that is scale-invariant and can provide better performance both in terms of speed and accuracy.



# classification subnet

The classification subnet is a fully convolutional network (FCN) attached to each FPN level. The subnet consists of four 3×33×3 convolutional layers with 256 filters, followed by RELU activations. Then, another 3×33×3 convolutional layer with K×A filters are applied, followed by sigmoid activation (instead of softmax activation)[5](https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/#fn5). The subnet has shared parameters across all levels. The shape of the output feature map would be (W,H,KA), where W and H are proportional to the width and height of the input feature map, K and A are the numbers of object class and anchor box (see below figure), which will be explain later.



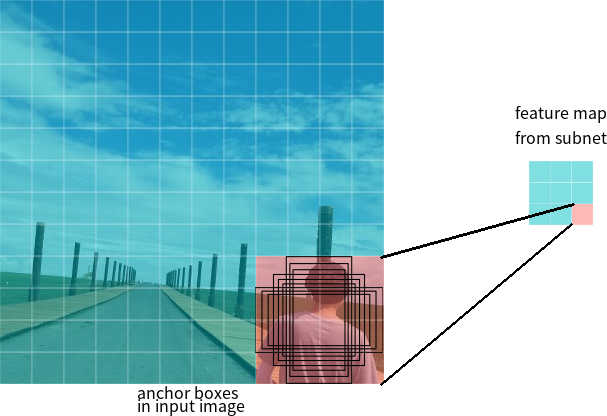
# Regression Subnet

The regression subnet is attached to each feature map of the FPN in parallel to the classification subnet. The design of the regression subnet is identical to that of the classification subnet, except that the last convolutional layer is 3×33×3 with 4A filters. Therefore, the shape of the output feature map would be (W,H,4A)

# a closer look at the subnet

Both the classification subnet and the regression subnet have output feature maps with width W and height H. As mentioned, each of the W×H slices corresponds to a region in the input image but what about the channels? Why does the classification subnet output KA channels while the regression subnet outputs 4A channels, and what do these channels respectively correspond to? To answer these questions, we first need to introduce a concept called anchor box, which was first proposed by Ren et al. (2017).

Let’s suppose that, given an input image, the width and height of a feature map output by the FPN is 3×33×3. Then for each one of these nine grid cells, the RetinaNet defines A=9 boxes called anchor boxes, each having different sizes and aspect ratios and covering an area in the input image (below Figure). Each anchor box is responsible for detecting the existence of objects from K classes in the area that it covers. Therefore, each anchor box corresponds to K numbers indicating the class probabilities. And since there are A bounding boxes per grid, the output feature map of the classification subnet has KA channels.



In addition to detecting the existence/class of objects, each anchor box is also responsible for detecting the size/shape of objects (if any). This is done through the regression subnetwork, which outputs 4 numbers for each anchor box that predict the relative offset (in terms of centre coordinates, width and height) between the anchor box and the ground truth box. Therefore, the output feature map of the regression subnet has 4A channels.

By now, we can see that the subnets actually generates many numbers (K from the classification subnet, 4 from the regression subnet) for a large number (∑7l=3Wl×Hl; where l denotes the level of pyramid; W and H are the width and height of the output feature map of the subnet.) of anchor boxes. Using these numbers to refine the anchor boxes, we get bounding box predictions. To calculate the loss for training, we need to compare the predictions with the ground-truths. However, how do we determine which bounding box should be compared with which ground-truth box, and what loss functions should be used? The goal of the authors is to naturally leverage the pyramidal shape of a Convnet feature hierarchy while creating a feature pyramid that has strong semantics at all scales. To achieve this goal, the authors relayed on a architecture that combines low-resolution, semantically strong features with high-resolution, semantically strong features via top-down pathway and lateral connection as shown in the diagram below.

# Matching prediction with ground truth

Note that the predictions output by the subnets are stored in **output tensors**. To calculate the loss, we would also need to create **target tensors**, each with the same shape as its corresponding output tensor and fill them with ground-truth labels at matching positions.

Also note that the match is actually performed between each anchor box and a ground-truth box. But since each anchor box has a one-to-one relationship with the bounding box prediction, the match naturally extends to the prediction and the ground-truth.

An anchor box is matched to a ground-truth box if their intersection-over-union (IoU) is greater than 0.5[8](https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/#fn8). When a match is found, the ground-truth labels will be assigned to the target tensor in the same positions as the corresponding predictions in the output tensor. In case of classification, a ground-truth label is a length KK one-hot encoded vector with a value of 1 in the corresponding class entry, while all the remaining class entries would be 0. In case of regression, the ground-truth label is a length 44 vector indicating the offset between the anchor box and the ground-truth box.

An anchor box is considered to be a background and has no matching ground-truth if its IoU with any ground-truth box is below 0.4. In this case, the target would be a length KK vector with all 0s[9](https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/#fn9). If the anchor box predicts an object, it will be penalized by the loss function. The regression target could be a vector of any values (typically zeros), but they will be ignored by the loss function.

Finally, an anchor box will also be considered to have no match if its IoU with any ground-truth box is between 0.4 and 0.5. However, unlike the previous case, both the labels for classification and regression will be ignored by the loss function.

# The Loss Function

The loss of RetinaNet is a multi-task loss that contains two terms: one for localization (denoted as Lloc below) and the other for classification (denoted as Lcls below). The multi-task loss can be written as:

L=λLloc+Lcls

where λ is a hyper-parameter that controls the balance between the two task losses. Next, let’s dive into more details on the two losses.

# Regression loss

Recall from previous sections how an anchor box is matched with a ground-truth box; the regression loss (as well as the classification loss) is calculated based on the match. Let’s denote these matching pairs as (Ai,Gi)i=1,...N where A represents an anchor, G represents a ground-truth, and N is the number of matches.

As mentioned, for each anchor with a match, the regression subnet predicts four numbers, which we denote as Pi=(Pix,Piy,Piw,Pih). The first two numbers specify the offset between the centres of anchor Ai and ground-truth Gi, while the last two numbers specify the offset between the width/height of the anchor and the ground-truth. Correspondingly, for each of these predictions, there is a regression target Ti computed as the offset between the anchor and the ground-truth:

Tix=(Gix−Aix)/Aiw(2)

Tiy=(Giy−Aiy)/Aih(3)

Tiw=log(Giw/Aiw)(4)

Tih=log(Gih/Aih) (5)

With the above notations, the regression loss can be defined as:

Lloc=∑j∈{x,y,w,h}smoothL1(Pij−Tij)

where smoothL1(x)smoothL1(x) is smooth L1 loss which can be defined as:

smoothL1(x)={0.5x2|x|<1

|x|−0.5|x|≥1}

It is worth noting that the smooth L1 loss is less sensitive to outliers than the L2 loss, which is adopted by some detectors like R-CNN. The L2 loss may require careful tuning of learning rates to prevent exploding gradients when the regression targets are unbounded.

# Classification Loss

The classification loss adopted by RetinaNet is a variant of the focal loss, which is the most innovative design of the detector. The loss for each anchor can be defined as[10](https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/#fn10):

Lcls=−K∑i=1(yilog(pi)(1−pi)γαi+(1−yi)log(1−pi)pγi(1−αi))

where KK denotes the number of classes; yi equals 1 if the ground-truth belongs to the i-th class and 0 otherwise; pi is the predicted probability for the i-th class; γ∈(0,+∞)is a focusing parameter; αi∈[0,1] is a weighting parameter for the i-th class. The loss is similar to categorical cross entropy, and they would be equivalent if γ=0 and αi=1. So, what are the purposes of these two additional parameters?

As the paper (Lin, Goyal, et al. 2017) points out, class imbalance is a very problematic issue that limits the performance of detectors in practice. This is because most locations in an image are easy negatives (meaning that they can be easily classified by the detector as background) and contribute no useful learning signal; worse still, since they account for a large portion of inputs, they can overwhelm the loss and computed gradients and lead to degenerated models. To address this problem, the focal loss introduces the focusing parameter γ to down-weight the loss assigned to easily classified examples. This effect increases as value of γ increases and makes the network focus more on hard examples.

The balancing parameter αα is also useful for addressing class imbalance. It may be set by inverse class frequency (or as a hyper-parameter) so that the loss assigned to examples of the background class can be down-weighted.

Note that since the two parameters interact with each other, they should be selected together. Generally speaking, as γ is increased, α should be decreased slightly.

# prediction

Finally, let’s see how RetinaNet generates predictions once it’s trained. Recall that, for each input image: there are ∑7l=3Wl×Hl×A anchor boxes from all FPN levels; for each anchor box, the classification subnet predicts K numbers indicating the probability distribution of object classes, while the regression subnet predicts 4 numbers indicating the offset between each anchor box and the corresponding bounding box.

For performance considerations, RetinaNet selects at most 1k anchor boxes that has the highest confidence score (i.e., predicted probability for each class) from each FPN level, after thresholding the score at 0.05. Only these anchors will be included in the following steps.

At this stage, an object in the image may be predicted by multiple anchor boxes. To remove redundancy, non-maximum-suppression (NMS) is applied to each class independently, which iteratively chooses an anchor box with the highest confidence and removes any overlapping anchor boxes with an IoU greater than 0.5.

In the last stage, for each remaining anchor, the regression subnet gives offset predictions that can used to refine the anchor to get a bounding box prediction.

# installation

* pip install imgaug
* pip install numpy scipy h5py
* pip install scikit-learn
* pip install tensorflow-gpu
* pip install opencv-contrib-python
* pip install keras-resnet
* pip install keras==2.1.3
* pip install tqdm

# Install Retinanet

* Download Link :https://github.com/fizyr/keras-retinanet
* cd keras-retinanet
* python setup.py build\_ext --inplace

# Install cudnn and cuda

* https://www.codingforentrepreneurs.com/blog/install-tensorflow-gpu-windows-cuda-cudnn(cudn)

# building dataset

* Download Bosch Small Traffic Lights Dataset <https://hci.iwr.uni-heidelberg.de/node/6132>
* Download contains Images and Yaml file.
* Convert Bosch Small Traffic Lights Dataset Annotations to Pascal VOC Format as image detail is given in Yaml file.
* python BoschTrafficYamlFileToXMLConverter.py train.yaml train\_out\_folder
* python BoschTrafficYamlFileToXMLConverter.py test.yaml test\_out\_folder

|  |
| --- |
| """  This script Converts Yaml annotations to Pascal .xml Files  of the Bosch Small Traffic Lights Dataset.  Example usage:  python bosch\_to\_pascal.py input\_yaml out\_folder  """  import os  import sys  import yaml  from lxml import etree  import os.path  import xml.etree.cElementTree as ET  def write\_xml(savedir, image, imgWidth, imgHeight,  depth=3, pose="Unspecified"):  boxes = image['boxes']  impath = image['path']  imagename = impath.split('/')[-1]  print("imagename&&&&&",imagename)  print("imagpath&&&&&",impath)  currentfolder = savedir.split("\\")[-1]  annotation = ET.Element("annotaion")  ET.SubElement(annotation, 'folder').text = str(currentfolder)  ET.SubElement(annotation, 'filename').text = str(impath)  imagename = imagename.split('.')[0]  size = ET.SubElement(annotation, 'size')  ET.SubElement(size, 'width').text = str(imgWidth)  ET.SubElement(size, 'height').text = str(imgHeight)  ET.SubElement(size, 'depth').text = str(depth)  ET.SubElement(annotation, 'segmented').text = '0'  for box in boxes:  obj = ET.SubElement(annotation, 'object')  ET.SubElement(obj, 'name').text = str(box['label'])  ET.SubElement(obj, 'pose').text = str(pose)  ET.SubElement(obj, 'occluded').text = str(box['occluded'])  ET.SubElement(obj, 'difficult').text = '0'  bbox = ET.SubElement(obj, 'bndbox')  ET.SubElement(bbox, 'xmin').text = str(box['x\_min'])  ET.SubElement(bbox, 'ymin').text = str(box['y\_min'])  ET.SubElement(bbox, 'xmax').text = str(box['x\_max'])  ET.SubElement(bbox, 'ymax').text = str(box['y\_max'])  xml\_str = ET.tostring(annotation)  root = etree.fromstring(xml\_str)  xml\_str = etree.tostring(root, pretty\_print=True)      save\_path = os.path.join(savedir, imagename + ".xml")    print("savepath is:::::::",save\_path)  print("xml string is::::::",xml\_str)  with open(save\_path, 'wb') as temp\_xml:  print("in writing the file")  temp\_xml.write(xml\_str)  if \_\_name\_\_ == '\_\_main\_\_':  if len(sys.argv) < 3:  print(\_\_doc\_\_)  sys.exit(-1)  yaml\_path = sys.argv[1]  out\_dir = sys.argv[2]  print("output directory is:::::",out\_dir)  images = yaml.load(open(yaml\_path, 'rb').read())  for image in images:  write\_xml(out\_dir, image, 1280, 720, depth=3, pose="Unspecified") |

* Perform above steps for Test and Train Data. Executing above code creates train and test xml file .
* Next, let’s write a Python script that will read all the image paths and annotations and output the three CSVs that are required during training and evaluating the model.
* train.csv — This file will hold all the annotations for training in the format: <path/to/image>,<xmin>,<ymin>,<xmax>,<ymax>,<label>  
  Each row will represent one bounding box, therefore, one image can be present in multiple rows depending on how many objects have been annotated in that image.
* test.csv — Similar to train.csv in format, this file will hold all the annotations for testing the model.
* classes.csv — A file with all unique class labels in the dataset with index assignments (starting from 0 and ignoring the background)
* For every image, find all the objects and iterate over each one of them. Then, find the bounding box (xmin, ymin, xmax, ymax) and the class label (name) for each object in the annotation. Do a cleanup by truncating any bounding box coordinate that falls outside the boundaries of the image. Also, do a sanity check if, by error, any minimum value is larger than the maximum value and vice-versa. If we find such values, we will ignore these objects and continue to the next one.

|  |
| --- |
| # -\*- coding: utf-8 -\*-  """  Created on Thu Oct 31 23:06:04 2019  @author: Pawan  """  import math  import sys  import os  import xml.etree.ElementTree as ET  import csv  ANNOTATIONS\_FILE = "C:\\Users\\Pawan\\Documents\\ML\\annotations\_train\_modified2.csv"  CLASSES\_FILE = "C:\\Users\\Pawan\\Documents\\ML\\classes\_train\_modified2.csv"  DATASET\_DIR="C:\\Users\\Pawan\\Documents\\ML\\TRAIN\_XML\_FILE"  annotations = []  classes = set([])  for xml\_file in [f for f in os.listdir(DATASET\_DIR) if f.endswith(".xml")]:  tree = ET.parse(os.path.join(DATASET\_DIR, xml\_file))  root = tree.getroot()  file\_name = None  print("after root################")  for elem in root:  if elem.tag == 'filename':  fileSplit=elem.text.split("/")  ModifiedFileName="C:\\Users\\Pawan\\Downloads\\dataset\_train\_rgb\\rgb\\train"+"\\"+fileSplit[3]+"\\"+fileSplit[4]  file\_name = ModifiedFileName  if elem.tag == 'object':  obj\_name = None  coords = []  for subelem in elem:  if subelem.tag == 'name':  obj\_name = subelem.text  if subelem.tag == 'bndbox':  for subsubelem in subelem:    if((str(subsubelem)).find('min')>0):  coords.append(math.floor(float(subsubelem.text)))  else:  coords.append(math.ceil(float(subsubelem.text)))    if coords[0]>=coords[2] or coords[1] >= coords[3] :  continue    if coords[2]<=coords[0] or coords[3]<=coords[1] :  continue    item = [file\_name] + coords + [obj\_name]  annotations.append(item)  classes.add(obj\_name)  with open(ANNOTATIONS\_FILE, 'w') as f:  writer = csv.writer(f)  writer.writerows(annotations)  with open(CLASSES\_FILE, 'w') as f:  for i, line in enumerate(classes):  f.write('{},{}\n'.format(line,i)) |

# Anchors parameters

* Anchor parameters are used to decide how anchor boxes will be generated for the model.
* As we're dealing mostly small boxes with can be highly elongated, we'll change ratios and scales to fit our needs.

|  |
| --- |
| with open('config.ini','w') as f: f.write('[anchor\_parameters]\nsizes = 16 32 64 128 256 512\nstrides = 8 16 32 64 128\nratios = 0.001 0.1 0.442 0.476 0.5 1.0 2.102 2.26 3 4\nscales =0.1 0.2 0.3 0.4 0.498 0.506 0.625 0.639 1 1.2 1.6 1.8\n') |

# Some Hyperparameters

* We will rescale our images to 672x672 for better precision

|  |
| --- |
| b = backbone('resnet50')  class args:  batch\_size =2  #64  config = read\_config\_file('config1111.ini')  #config=True  random\_transform = True # Image augmentation  annotations = "C:\\Users\\Pawan\\Documents\\ML\\annotations\_train\_modified2.csv"  val\_annotations = "C:\\Users\\Pawan\\Documents\\ML\\annotations\_test\_modified2.csv"  no\_resize=False  classes = "C:\\Users\\Pawan\\Documents\\ML\\classes\_train\_modified2.csv"  image\_min\_side = 672  image\_max\_side = 672  dataset\_type = 'csv'  tensorboard\_dir = 'C:\\Users\\Pawan\\Documents\\Tensorboard'  evaluation = True  snapshots = True  snapshot\_path = "C:\\Users\\Pawan\\Documents\\ML\\snapshots12"  backbone = 'resnet50'  #epochs = 70  epochs = 70  steps = 10755//(batch\_size)  weighted\_average = True  gpu=0  resize=True      train\_gen,valid\_gen = create\_generators(args,b.preprocess\_image) |

# Image Augmentation

* In addition to augmentations already done by keras-retinanet [here](https://github.com/fizyr/keras-retinanet/blob/master/keras_retinanet/bin/train.py#L227) , we'll use a package called imgaug to further augment the data.

|  |
| --- |
| sometimes = lambda aug: iaa.Sometimes(0.5, aug)  # Define our sequence of augmentation steps that will be applied to every image.  seq = iaa.Sequential(  [  #  # Execute 1 to 9 of the following (less important) augmenters per  # image. Don't execute all of them, as that would often be way too  # strong.  #  iaa.SomeOf((1, 9),  [  # Blur each image with varying strength using  # gaussian blur (sigma between 0 and .5),  # average/uniform blur (kernel size 1x1)  # median blur (kernel size 1x1).  iaa.OneOf([  iaa.GaussianBlur((0,0.5)),  iaa.AverageBlur(k=(1)),  iaa.MedianBlur(k=(1)),  ]),  # Sharpen each image, overlay the result with the original  # image using an alpha between 0 (no sharpening) and 1  # (full sharpening effect).  iaa.Sharpen(alpha=(0, 0.25), lightness=(0.75, 1.5)),  # Add gaussian noise to some images.  # In 50% of these cases, the noise is randomly sampled per  # channel and pixel.  # In the other 50% of all cases it is sampled once per  # pixel (i.e. brightness change).  iaa.AdditiveGaussianNoise(  loc=0, scale=(0.0, 0.01\*255), per\_channel=0.5  ),  # Either drop randomly 1 to 10% of all pixels (i.e. set  # them to black) or drop them on an image with 2-5% percent  # of the original size, leading to large dropped  # rectangles.  iaa.OneOf([  iaa.Dropout((0.01, 0.1), per\_channel=0.5),  iaa.CoarseDropout(  (0.03, 0.15), size\_percent=(0.02, 0.05),  per\_channel=0.2  ),  ]),  # Add a value of -5 to 5 to each pixel.  iaa.Add((-5, 5), per\_channel=0.5),  # Change brightness of images (85-115% of original value).  iaa.Multiply((0.85, 1.15), per\_channel=0.5),  # Improve or worsen the contrast of images.  iaa.ContrastNormalization((0.75, 1.25), per\_channel=0.5),  # Convert each image to grayscale and then overlay the  # result with the original with random alpha. I.e. remove  # colors with varying strengths.  iaa.Grayscale(alpha=(0.0, 0.25)),  # In some images distort local areas with varying strength.  sometimes(iaa.PiecewiseAffine(scale=(0.001, 0.01)))  ],  # do all of the above augmentations in random order  random\_order=True  )  ],  # do all of the above augmentations in random order  random\_order=True  )  #######################################################################################  def augment\_train\_gen(train\_gen,visualize=False):  '''  Creates a generator using another generator with applied image augmentation.  Args  train\_gen : keras-retinanet generator object.  visualize : Boolean; False will convert bounding boxes to their anchor box targets for the model.  '''  imgs = []  boxes = []  targets = []  size = train\_gen.size()  idx = 0  while True:  while len(imgs) < args.batch\_size:  image = train\_gen.load\_image(idx % size)  annotations = train\_gen.load\_annotations(idx % size)  image,annotations = train\_gen.random\_transform\_group\_entry(image,annotations)  imgs.append(image)  boxes.append(annotations['bboxes'])  targets.append(annotations)  idx += 1  if visualize:  imgs = seq.augment\_images(imgs)  imgs = np.array(imgs)  boxes = np.array(boxes)  yield imgs,boxes  else:  imgs = seq.augment\_images(imgs)  imgs,targets = train\_gen.preprocess\_group(imgs,targets)  imgs = train\_gen.compute\_inputs(imgs)  targets = train\_gen.compute\_targets(imgs,targets)  imgs = np.array(imgs)  yield imgs,targets  imgs = []  boxes = []  targets = [] |

# Visualize augmentations

|  |
| --- |
| i = 0  for imgs,boxes in augment\_train\_gen(train\_gen,visualize=True):  if i > skip\_batches:  fig=plt.figure(figsize=(24,96))  columns = 2  rows = 8  for i in range(1, columns\*rows + 1):  draw\_boxes(imgs[i], boxes[i], (0, 255, 0), thickness=1)  fig.add\_subplot(rows, columns, i)  plt.imshow(cv2.cvtColor(imgs[i],cv2.COLOR\_BGR2RGB))  plt.show()    else:  i += 1 |

# More Hyperparameters

* we'll use learning rate of 0.001 and freeze weights for the backbone

|  |
| --- |
| model, training\_model, prediction\_model = create\_models(  backbone\_retinanet=b.retinanet,  num\_classes=train\_gen.num\_classes(),  weights=None,  multi\_gpu=True,  freeze\_backbone=True,  lr=1e-3,  config=args.config  ) |

# Callbacks

|  |
| --- |
| callbacks = create\_callbacks(  model,  training\_model,  prediction\_model,  valid\_gen,  args,  ) |

# Download pretrained model

* We download a pretrained model on COCO dataset and load it's weights
* Download Link: <https://github.com/fizyr/keras-retinanet/releases/download/0.5.1/resnet50_coco_best_v2.1.0.h5>

# Loading the model

|  |
| --- |
| training\_model.load\_weights('C:\\Users\\Pawan\\Documents\\ML\\snapshots11\\resnet50\_csv\_01.h5',  skip\_mismatch=True,by\_name=True) |

# Training the Model

|  |
| --- |
| training\_model.fit\_generator(generator=augment\_train\_gen(train\_gen),  steps\_per\_epoch=args.steps,  epochs=args.epochs,  verbose=1,  callbacks=callbacks,) |

# Inference/Prediction

|  |
| --- |
| import os  from os import listdir, walk  from os.path import join  import numpy as np  import keras  import math  import tensorflow as tf  from keras\_retinanet.utils.visualization import draw\_boxes  from sklearn.model\_selection import train\_test\_split  from imgaug import augmenters as iaa  import matplotlib.pyplot as plt  from keras\_retinanet.utils.gpu import setup\_gpu  from keras\_retinanet.utils.image import read\_image\_bgr, preprocess\_image, resize\_image  from tqdm import tqdm  from keras\_retinanet.bin.train import create\_generators,create\_models,create\_callbacks  from keras\_retinanet.models import backbone,load\_model,convert\_model  from keras\_retinanet.utils.config import read\_config\_file,parse\_anchor\_parameters  from keras\_retinanet.utils.visualization import draw\_boxes  from sklearn.model\_selection import train\_test\_split  from imgaug import augmenters as iaa  import matplotlib.pyplot as plt  from keras\_retinanet.utils.gpu import setup\_gpu  import tensorflow.compat.v1 as tf  tf.disable\_v2\_behavior()  config = tf.ConfigProto()  config.gpu\_options.allow\_growth = True  sess = tf.Session(config=config)  b = backbone('resnet50')  class args:  batch\_size =4  config = read\_config\_file('config.ini')  random\_transform = True # Image augmentation  annotations = "C:\\Users\\Pawan\\Documents\\ML\\annotations\_train\_modified2.csv"  val\_annotations = "C:\\Users\\Pawan\\Documents\\ML\\annotations\_test\_modified2.csv"  no\_resize=False  classes = "C:\\Users\\Pawan\\Documents\\ML\\classes\_train\_modified2.csv"  image\_min\_side = 672  image\_max\_side = 672  dataset\_type = 'csv'  tensorboard\_dir = 'C:\\Users\\Pawan\\Documents\\Tensorboard'  evaluation = True  snapshots = True  snapshot\_path = "C:\\Users\\Pawan\\Documents\\ML\\snapshots12"  backbone = 'resnet50'  epochs = 100  steps = 10755//(batch\_size)  gpu=0  resize=True    train\_gen,valid\_gen = create\_generators(args,b.preprocess\_image)  model, training\_model, prediction\_model = create\_models(  backbone\_retinanet=b.retinanet,  num\_classes=train\_gen.num\_classes(),  weights=None,  multi\_gpu=True,  freeze\_backbone=True,  lr=1e-9,  config=args.config  )    training\_model.load\_weights("C:\\Users\\Pawan\\Documents\\ML\\snapshots12\\resnet50\_csv\_07.h5")  infer\_model = convert\_model(training\_model,anchor\_params=parse\_anchor\_parameters(read\_config\_file('C:\\Users\\Pawan\\Documents\\config.ini')))  def test\_gen(image\_ids, bs = 2, size=672,test = True):  imgs = []  scale = None  idx = 0  if test:  path = 'C:\\Users\\Pawan\\Downloads\\dataset\_test\_rgb\\rgb\\test\\'  else:  path = 'C:\\Users\\Pawan\\Downloads\\dataset\_test\_rgb\\rgb\\test\\'    while idx < len(image\_ids):  if len(imgs) < bs:  imgs.append(resize\_image(preprocess\_image(read\_image\_bgr(path + image\_ids[idx] + '.png')),min\_side=size,max\_side=size)[0])  if scale is None:  scale = resize\_image(preprocess\_image(read\_image\_bgr(path + image\_ids[idx] + '.png')),min\_side=size,max\_side=size)[1]  idx += 1  else:  yield np.array(imgs),scale  imgs = []      if len(imgs) > 0:  yield np.array(imgs),scale  print("###########################################################################################")  \_,\_,image\_ids = next(walk('C:\\Users\\Pawan\\Downloads\\dataset\_test\_rgb\\rgb\\test\\'))  image\_ids = [i[:-4] for i in image\_ids]  image\_ids = sorted(image\_ids)  #print("image\_ids",image\_ids)  iter\_num = 0  test\_bs = 2  for imgs,scale in tqdm(test\_gen(image\_ids,bs=test\_bs),total=math.ceil(len(image\_ids)/test\_bs)):  boxes, scores, labels = infer\_model.predict\_on\_batch(imgs)  boxes /= scale  for img\_num in range(len(imgs)):  with open('C:\\Users\\Pawan\\Music\\Paku5\\' + image\_ids[(iter\_num\*test\_bs) + img\_num] + '.txt', 'w') as f:  for box, score, label in zip(boxes[img\_num], scores[img\_num], labels[img\_num]):  # scores are sorted so we can break  if score < 0:  break  f.write(f'{label + 1} {score} {int((box[1]))} {int((box[0]))} {int((box[3]))} {int((box[2]))} \n')  iter\_num += 1 |

# Predict Image

|  |
| --- |
| from keras\_retinanet.utils.visualization import draw\_box, draw\_caption  from keras\_retinanet.utils.colors import label\_color  from keras\_retinanet.utils.image import read\_image\_bgr, preprocess\_image, resize\_image  import cv2  from os import walk  import matplotlib.pyplot as plt  \_,\_,image\_ids = next(walk('C:\\Users\\Pawan\\Downloads\\dataset\_test\_rgb\\rgb\\test\\'))  image\_ids = [i[:-4] for i in image\_ids]  image\_ids = sorted(image\_ids)  image\_ids=["28164"]  idx = 0  image\_id = 1  score\_thres = 0.1  for id in image\_ids:  # load image  #idx += 1  #if idx == image\_id:  image = read\_image\_bgr('C:\\Users\\Pawan\\Downloads\\dataset\_test\_rgb\\rgb\\test\\' + id + '.png')  # copy to draw on  draw = image.copy()  draw = cv2.cvtColor(draw, cv2.COLOR\_BGR2RGB)  # process image  boxes = [list(map(int,(line.split()[3],line.split()[2],line.split()[5],line.split()[4]))) for line in open('C:\\Users\\Pawan\\Music\\Paku5\\' + id + '.txt','r').readlines()]  scores = [float(line.split()[1]) for line in open('C:\\Users\\Pawan\\Music\\Paku5\\' + id + '.txt','r').readlines()]  labels = [int(line.split()[0]) - 1 for line in open('C:\\Users\\Pawan\\Music\\Paku5\\' + id + '.txt','r').readlines()]  print("scores is :::::",scores)  for box, score, label in zip(boxes, scores, labels):  if score < score\_thres:  break  color = label\_color(label)  draw\_box(draw, box, color=color,thickness=1)  caption = "{:.3f}".format(score)  draw\_caption(draw, box, caption)  plt.figure(figsize=(15, 15))  plt.axis('off')  plt.imshow(draw)  plt.show()  break |

# Total Map Achieved from Training Model is 39%

Running network: 100% (7147 of 7147) |###| Elapsed Time: 0:12:00 Time: 0:12:00

Parsing annotations: 100% (7147 of 7147) || Elapsed Time: 0:00:00 Time: 0:00:00

0 instances of class GreenStraightRight with average precision: 0.0000

5321 instances of class Red with average precision: 0.3694

442 instances of class off with average precision: 0.0005

0 instances of class GreenStraightLeft with average precision: 0.0000

0 instances of class GreenStraight with average precision: 0.0000

0 instances of class RedRight with average precision: 0.0000

0 instances of class GreenRight with average precision: 0.0000

154 instances of class Yellow with average precision: 0.0445

0 instances of class RedLeft with average precision: 0.0000

0 instances of class RedStraight with average precision: 0.0000

7569 instances of class Green with average precision: 0.4249

0 instances of class GreenLeft with average precision: 0.0000

0 instances of class RedStraightLeft with average precision: 0.0000

**mAP: 0.3847**

# Experiment and Results

|  |
| --- |
| scores is ::::: [0.6621619462966919, 0.6152707934379578, 0.4665617048740387, 0.3548182249069214, 0.3443726897239685, 0.26342302560806274, 0.26012301445007324, 0.2334878146648407, 0.2268904149532318, 0.21646422147750854, 0.20138224959373474, 0.18319416046142578, 0.17737096548080444, 0.15093502402305603, 0.1430239975452423, 0.141590416431427, 0.13504180312156677, 0.13482537865638733, 0.12520945072174072, 0.11978760361671448, 0.1185823380947113, 0.11579084396362305, 0.11218792200088501, 0.1117866039276123, 0.11095410585403442, 0.10869008302688599, 0.10500943660736084, 0.10437360405921936, 0.10064387321472168, 0.09914103150367737, 0.09482219815254211, 0.0884065330028534, 0.08746984601020813, 0.0851631760597229, 0.0846567153930664, 0.08421489596366882, 0.08210834860801697, 0.08201262354850769, 0.0819520354270935, 0.08170577883720398, 0.07812950015068054, 0.07706010341644287, 0.07694727182388306, 0.076772540807724, 0.0761944055557251, 0.07222086191177368, 0.07214677333831787, 0.07160621881484985, 0.07092833518981934, 0.06951013207435608, 0.06949323415756226, 0.06923791766166687, 0.06859856843948364, 0.06818825006484985, 0.06683939695358276, 0.06659549474716187, 0.06614750623703003, 0.06598401069641113, 0.06588268280029297, 0.06526273488998413, 0.06508141756057739, 0.0640210509300232, 0.06284156441688538, 0.06282418966293335, 0.06262487173080444, 0.0624944269657135, 0.062422096729278564, 0.062124550342559814, 0.06200599670410156, 0.06168997287750244, 0.061205655336380005, 0.06106477975845337, 0.060545414686203, 0.059911131858825684, 0.059479743242263794, 0.05855002999305725, 0.05843770503997803, 0.05840784311294556, 0.05816096067428589, 0.05755653977394104, 0.05727052688598633, 0.05708572268486023, 0.056704163551330566, 0.05621013045310974, 0.05610126256942749, 0.056080013513565063, 0.055916547775268555, 0.05585181713104248, 0.0557064414024353, 0.05564594268798828, 0.05547612905502319, 0.054994404315948486, 0.0548056960105896, 0.054744333028793335, 0.05458986759185791, 0.05406603217124939, 0.05399686098098755, 0.05392506718635559, 0.053876638412475586, 0.0534062385559082, 0.053102195262908936, 0.05308029055595398, 0.053034424781799316, 0.05271118879318237, 0.05267989635467529, 0.052247315645217896, 0.051478177309036255, 0.05141270160675049, 0.05139085650444031, 0.05136466026306152, 0.050578951835632324, 0.050404131412506104] |



# Future Work

To Improve MAP

# References

* <https://github.com/fizyr/keras-retinanet>
* <https://github.com/kunwar31/ESRI_Object_Detection>
* <https://medium.com/@14prakash/the-intuition-behind-retinanet-eb636755607d>
* <https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/>