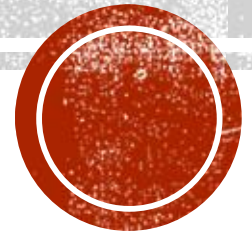


OBJECT DETECTION(AIRCRAFT) & BOUNDING BOX PROBLEM [BTP2]

By – Ujjwal & Geordi

Supervisor – Prof. J Indu





Objective

- Aircraft detection in the input image and creating bounding box along its location
- Creating a UI platform to perform testing on any random images



BTP-1 SUMMARY

- In our previous project we both worked on two different aspects of the same problem
 - Classifying If an image contains an aircraft or not
 - Locating an aircraft inside in an image via a bounding box

Both these tasks couldn't be integrated efficiently, as the computational load involved was pretty high, and the efficiency in locating aircrafts inside the image was not very good.

Therefore we moved on to work on employing a more powerful single solution to both these tasks using a deep learning approach

Methodology





Finding the right dataset

- For any machine learning model to work, it needs a proper dataset to train from.
- Finding a proper dataset was the first challenge and the most important aspect of our solution
- Actual satellite images with aircrafts in them were very difficult to access from any open source platform
- We therefore used a CGI dataset, consisting of random satellite images with aircrafts placed randomly inside the images digitally.

Methodology – CGI Dataset



- 400 computer generated image + 100 for testing
- csv file containing location of aircraft in images



Methodology - Augmentation

- 10 different augmentation was performed such as flipping, scaling, rotating, shearing, translating and a few combination of these
- From 400 training dataset we increased the number to 4400
- Annotation.txt file was created to containing location of each training data, its name and min-max of x,y coordinate of aircrafts in those images to feed to the model for training

Methodology – Augmentation(2)

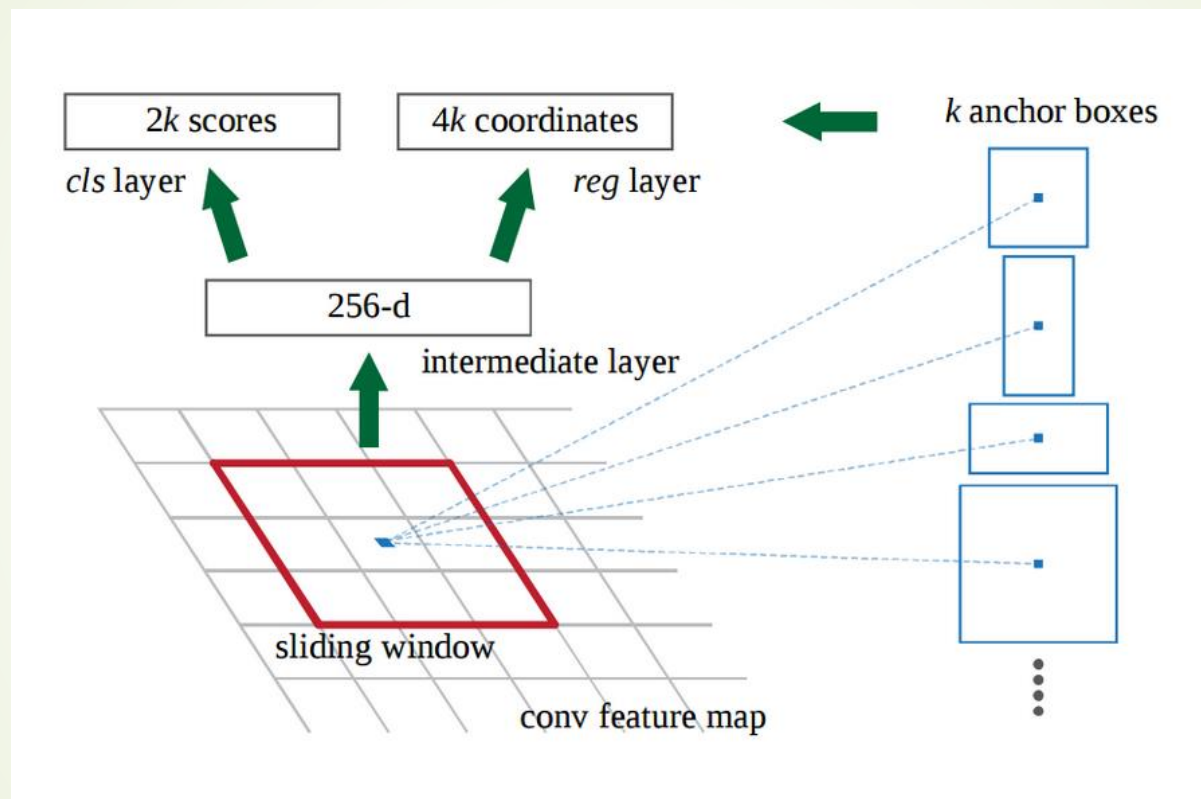
train_labels.csv - Excel

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	filename	xmin	ymin	xmax	ymax	label							
2	img1.png	410	356	469	419	plane							
3	img1.png	635	276	738	380	plane							
4	img1.png	830	46	902	119	plane							
5	img1.png	669	80	715	120	plane							
6	img1.png	58	225	108	272	plane							
7	img1.png	526	279	579	332	plane							
8	img1.png	822	446	870	493	plane							
9	img1.png	591	322	664	400	plane							
10	img1.png	838	227	920	309	plane							
11	img1.png	605	353	661	415	plane							
12	img1.png	775	434	827	484	plane							
13	img1.png	159	467	235	541	plane							
14	img10.png	439	537	475	574	plane							
15	img10.png	156	277	251	376	plane							
16	img10.png	480	480	524	526	plane							
17	img10.png	763	39	860	136	plane							
18	img10.png	50	318	123	392	plane							
19	img10.png	568	324	621	376	plane							
20	img10.png	375	134	420	182	plane							
21	img10.png	490	486	531	528	plane							
22	img10.png	245	493	317	567	plane							
23	img10.png	859	59	909	111	plane							
24	img10.png	839	152	905	221	plane							
25	img100.pn	537	69	595	127	plane							
26	img100.pn	307	203	343	241	plane							
27	img100.pn	820	162	888	229	plane							
28	img100.pn	490	97	586	190	plane							
29	img100.pn	118	30	193	104	plane							

train_labels

csv file
containing
object
coordinates

RPN – The architecture

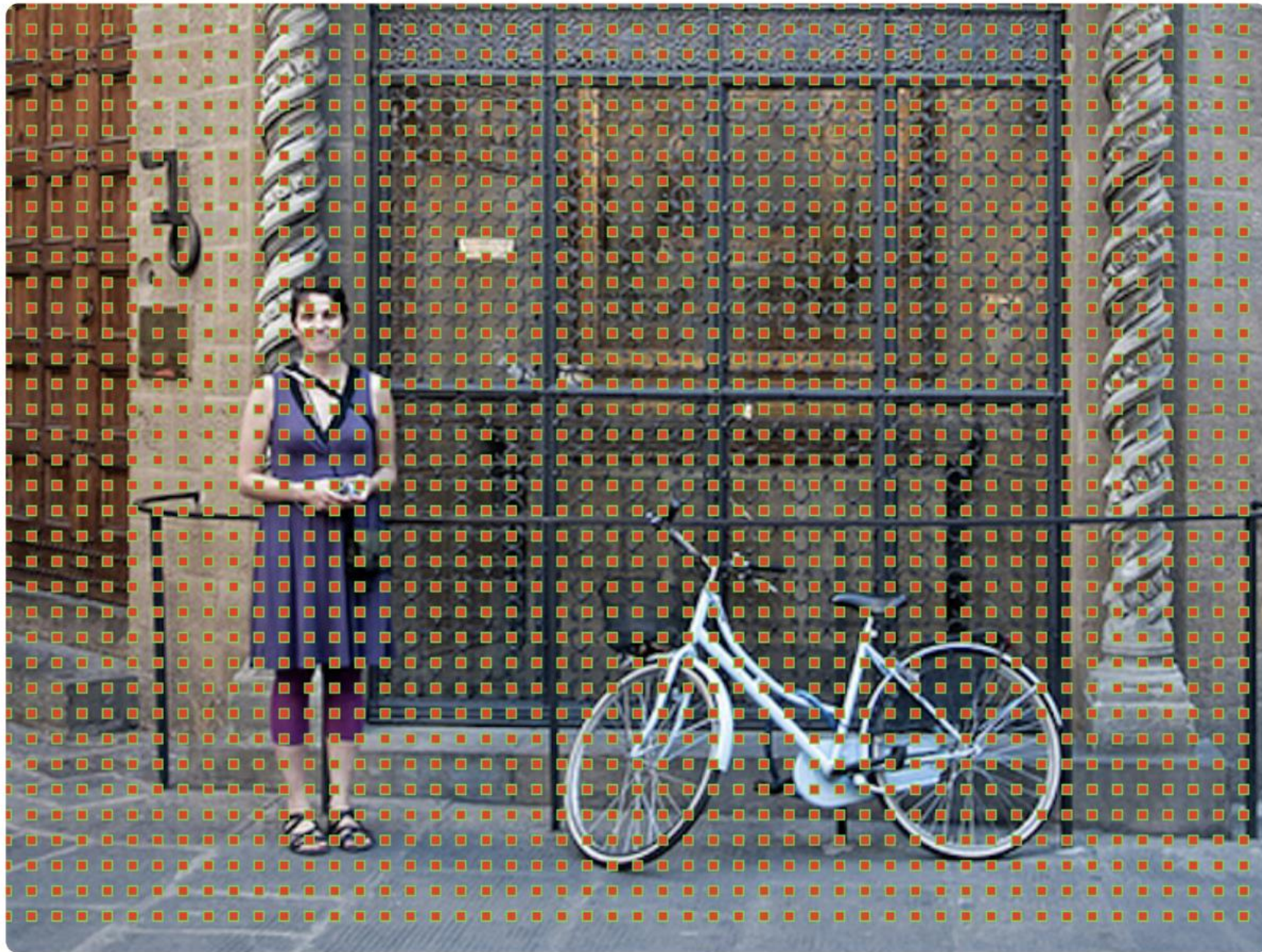


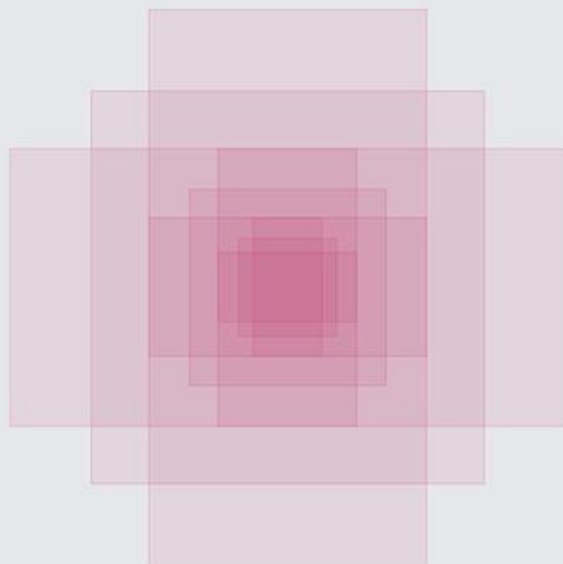
- Faster RCNN uses region proposal network (RPN)



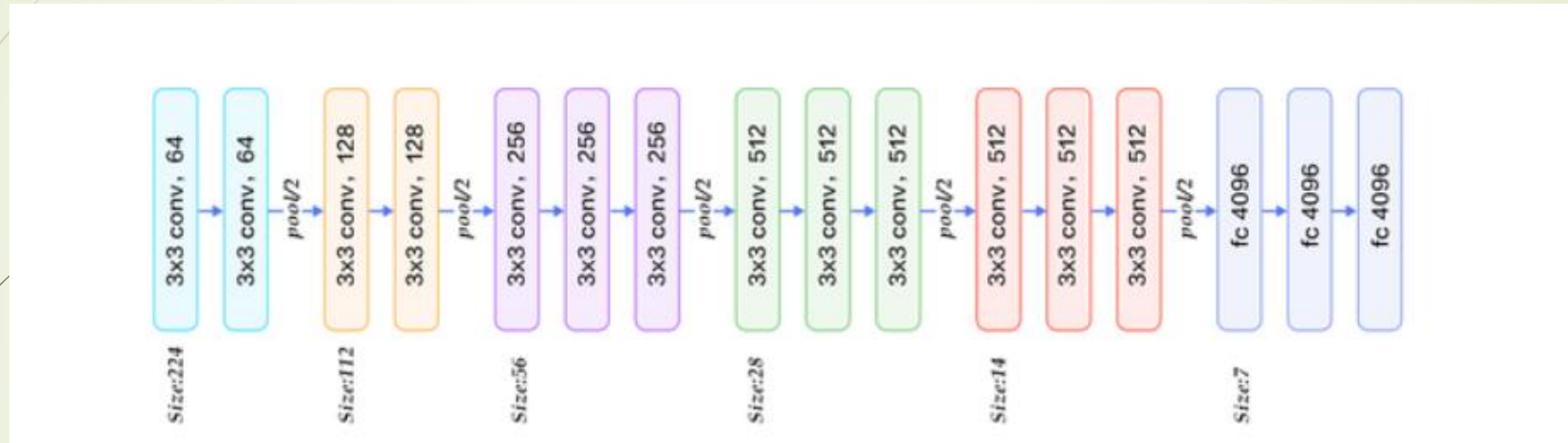
Working of RPN

- At last layer of last CNN, a 3×3 sliding window moves across feature map and maps it to lower dimension
- For each sliding window location, it generates multiple possible regions based on k fixed-ratio bounding boxes
- Each region proposal consists of a) an “objectness” score for that region and b) 4 coordinates representing the bounding box of the region
- For each of those boxes, we output whether or not we think it contains an object, and what the coordinates for that box are.
- If an anchor box has an “objectness” score above a certain threshold, that box’s coordinates get passed forward as a region proposal.
- Once we have our region proposals, we feed them straight into what is essentially a Fast R-CNN.





Faster RCNN (The VGG Network)



- Faster R-CNN = RPN + Fast R-CNN.
- The final step is a softmax function for classification and linear regression to fix the boxes' location.

VGG Network

```
# Block 1
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv1')(img_input)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv1')(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

# Block 3
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv1')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv2')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)

# Block 4
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

# Block 5
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv3')(x)
# x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)
```

- A part of this network is converted to trainable form and then keras package is used to define the properties

Methodology – Building Model

- The input data is from annotation.txt file which contains a bunch of images with their bounding boxes information. We used RPN method to create proposed bboxes. It creates final layer's feature map according to input image size
- Calculate rpn for each image: returns whether box is valid or not and whether box has an object(aircraft) or not
- Calculate region of interest from RPN: It returns box from non-maximum suppression and coordinates for bbox based on the anchor value that we set
- ROI pooling layer and classifier layer: ROI pooling layer is a function to process ROI to a specific size output by max pooling. Classifier layer is the final layer of the whole model and just behind the RoIPooling layer. It's used to predict the class name for each input anchor and the regression of their bounding box.
- Finally, there are two output layers.
 - # out_class: softmax activation function for classifying the class name of the object
 - # out_regr: linear activation function for bboxes coordinates regression



Training



- We carried out a total of 1000 epochs
- Training was done on Google colab server with GPU network
- It took around 22hrs for training
- Google drive was mounted with the server which kept on saving weights whenever there was an overall decrease in loss function

Training(2)

```
1000/1000 [=====] - 753s 753ms/step - rpn_cls: 0.8438 - rpn_regr: 0.2302 -
final_cls: 0.4520 - final_regr: 0.2268
Mean number of bounding boxes from RPN overlapping ground truth boxes: 12.26321036889332
Classifier accuracy for bounding boxes from RPN: 0.807
Loss RPN classifier: 0.8722642721683797
Loss RPN regression: 0.23128215429116972
Loss Detector classifier: 0.446441008906977
Loss Detector regression: 0.23175630393996835
Total loss: 1.7817437393064948
Elapsed time: 932.6565265655518
Epoch 88/126
1000/1000 [=====] - 785s 785ms/step - rpn_cls: 0.9270 - rpn_regr: 0.2315 -
final_cls: 0.4147 - final_regr: 0.2248
Mean number of bounding boxes from RPN overlapping ground truth boxes: 11.53227408142999
Classifier accuracy for bounding boxes from RPN: 0.81425
Loss RPN classifier: 0.87854375148419
Loss RPN regression: 0.23205739994451868
Loss Detector classifier: 0.4183896440564631
Loss Detector regression: 0.23319237146084196
Total loss: 1.7621831669460137
Elapsed time: 785.0421531200409
Epoch 89/126
1000/1000 [=====] - 737s 737ms/step - rpn_cls: 0.9438 - rpn_regr: 0.2379 -
final_cls: 0.4652 - final_regr: 0.2445
Mean number of bounding boxes from RPN overlapping ground truth boxes: 12.395812562313061
Classifier accuracy for bounding boxes from RPN: 0.80775
Loss RPN classifier: 0.9155260486609184
Loss RPN regression: 0.22743836374177046
Loss Detector classifier: 0.4338727388291154
Loss Detector regression: 0.22870131808705627
Total loss: 1.8055384693188605
Elapsed time: 737.3596696853638
Epoch 90/126
1000/1000 [=====] - 687s 687ms/step - rpn_cls: 1.0162 - rpn_regr: 0.2298 -
final_cls: 0.4525 - final_regr: 0.2309
Mean number of bounding boxes from RPN overlapping ground truth boxes: 11.679960119641077
Classifier accuracy for bounding boxes from RPN: 0.80575
Loss RPN classifier: 0.986764288362554
Loss RPN regression: 0.2323578881379217
Loss Detector classifier: 0.45484118838390714
Loss Detector regression: 0.23057722708582878
Total loss: 1.9045405919702116
Elapsed time: 686.8345861434937
Epoch 91/126
```

- Screenshot of verbose during training

Validation metric – mAP score

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



IoU of predicted BB (yellow) and GT BB (blue) > 0.5 with the correct classification



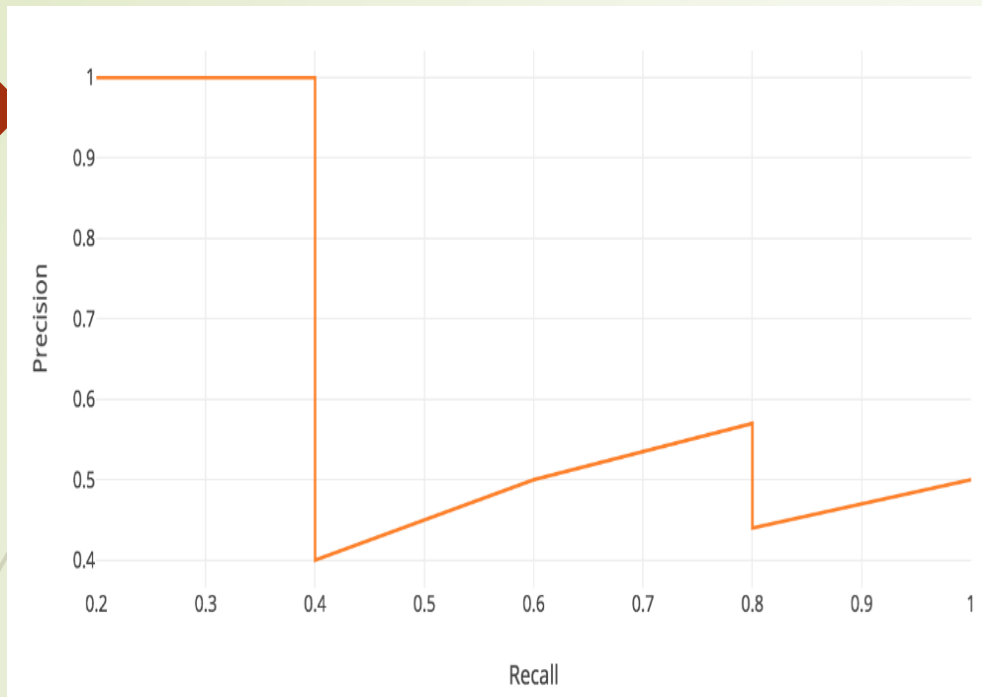
$\text{IoU} < 0.5$



Duplicate BB are considered as FP




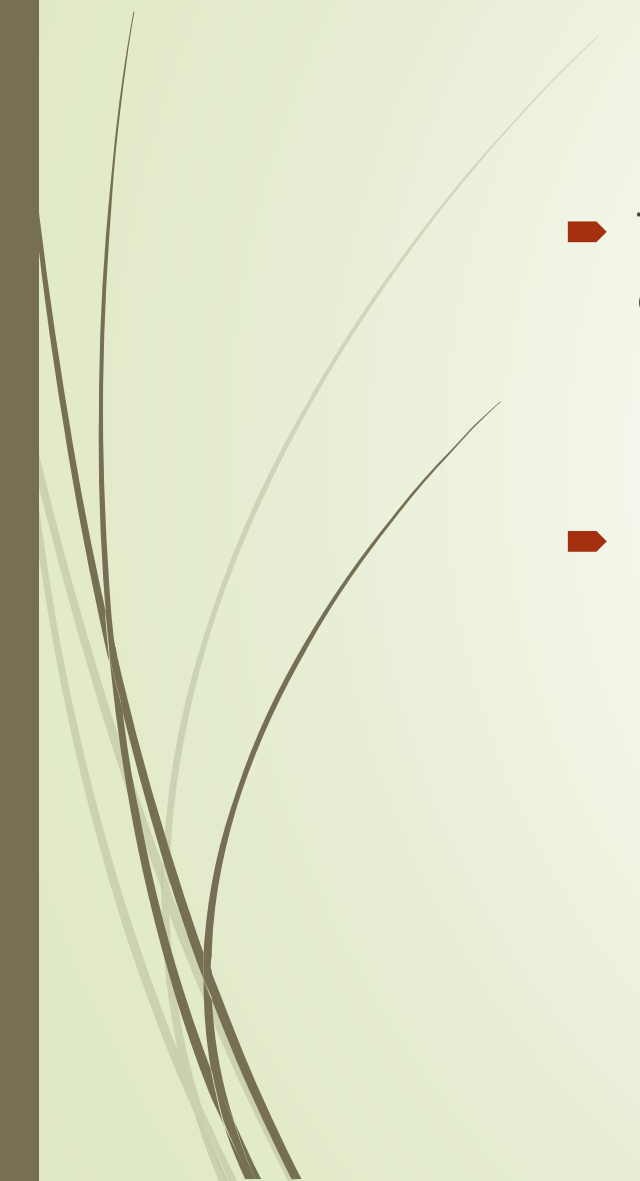
No IoU




Area under precision recall curve is mean average precision

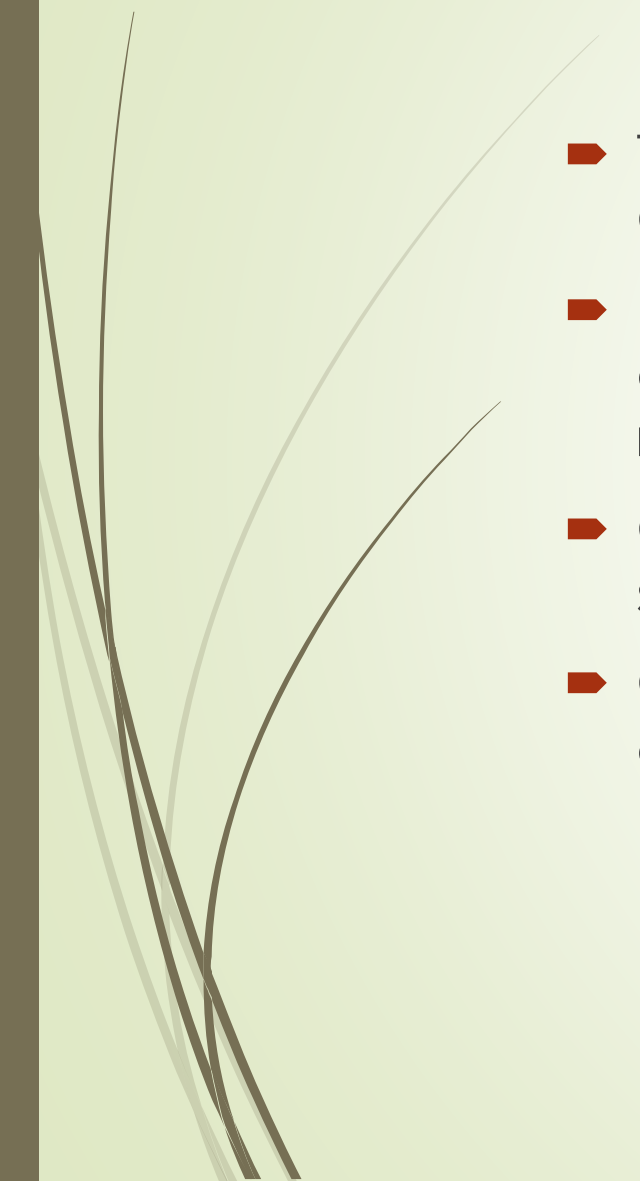
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

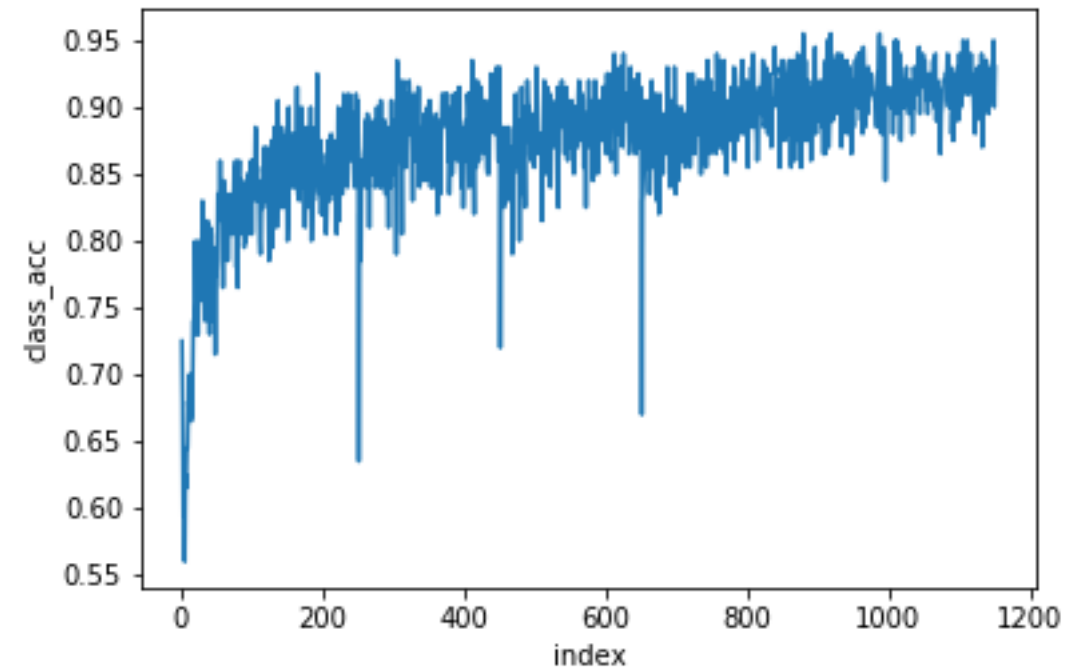
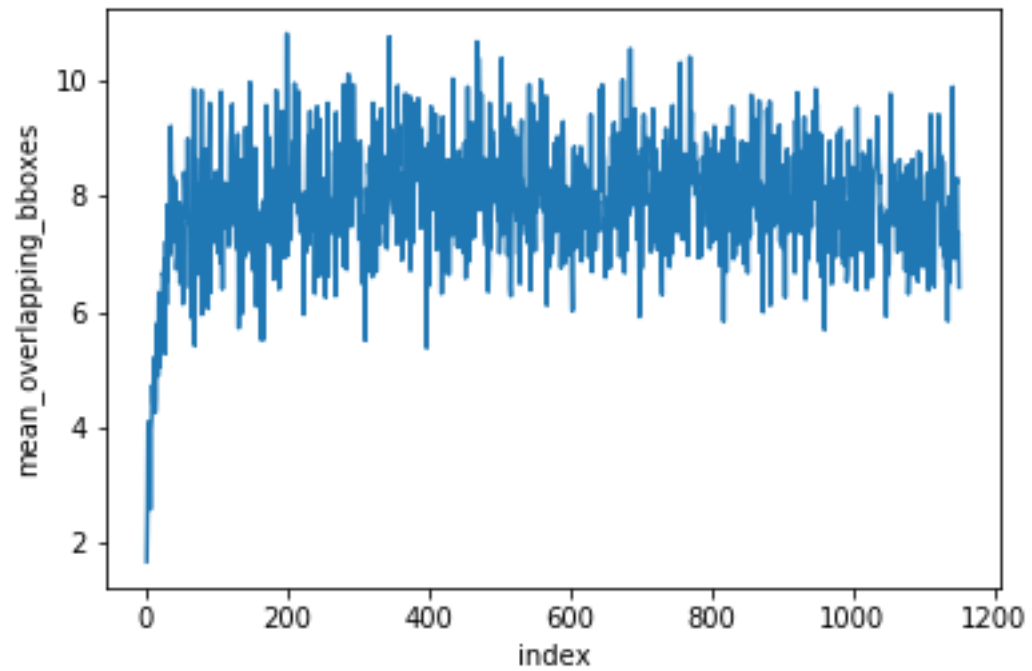
- 
- 
- The training hyperparameters like number of epochs, iterations per epoch, and detection threshold were determined based on the mAP we achieved.
 - Upon final validation our trained model was able to achieve an mAP score of 0.54 which is quite good given the limited dataset.



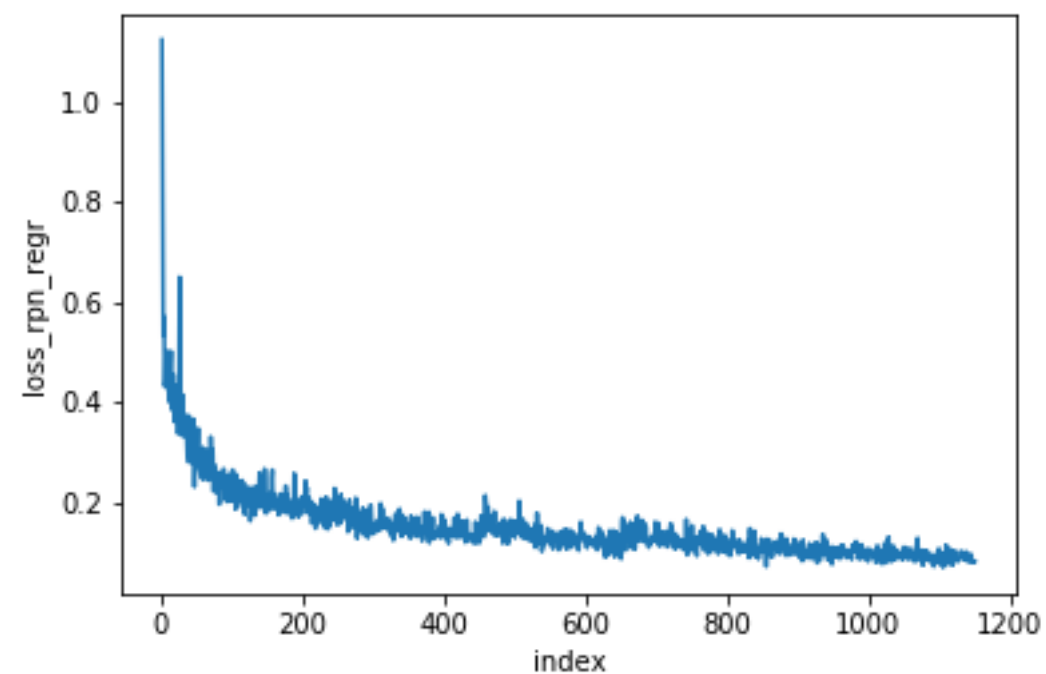
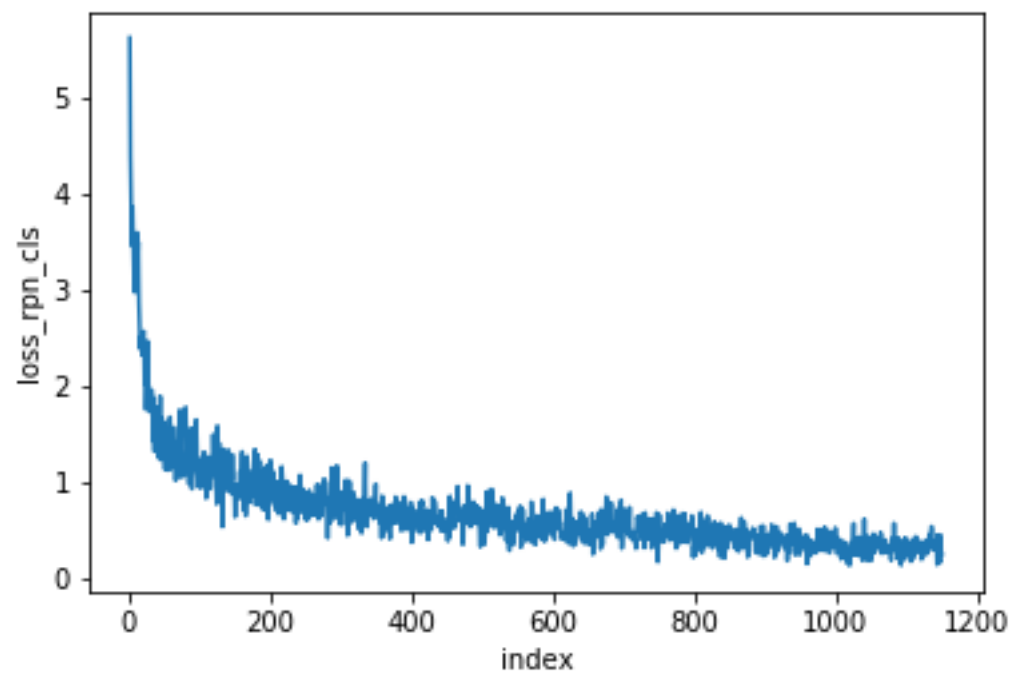
Results

- There are two loss functions we applied to both the RPN model and Classifier model.
 - RPN model has two output. One is for classifying whether it's an object and the other one is for bounding boxes' coordinates regression.
 - Compared with the two plots for bboxes regression, they show a similar tendency and even similar loss value.
 - Compared with two plots for classifying, we can see that predicting objectness is easier than predicting the class name of a bbox.
- 

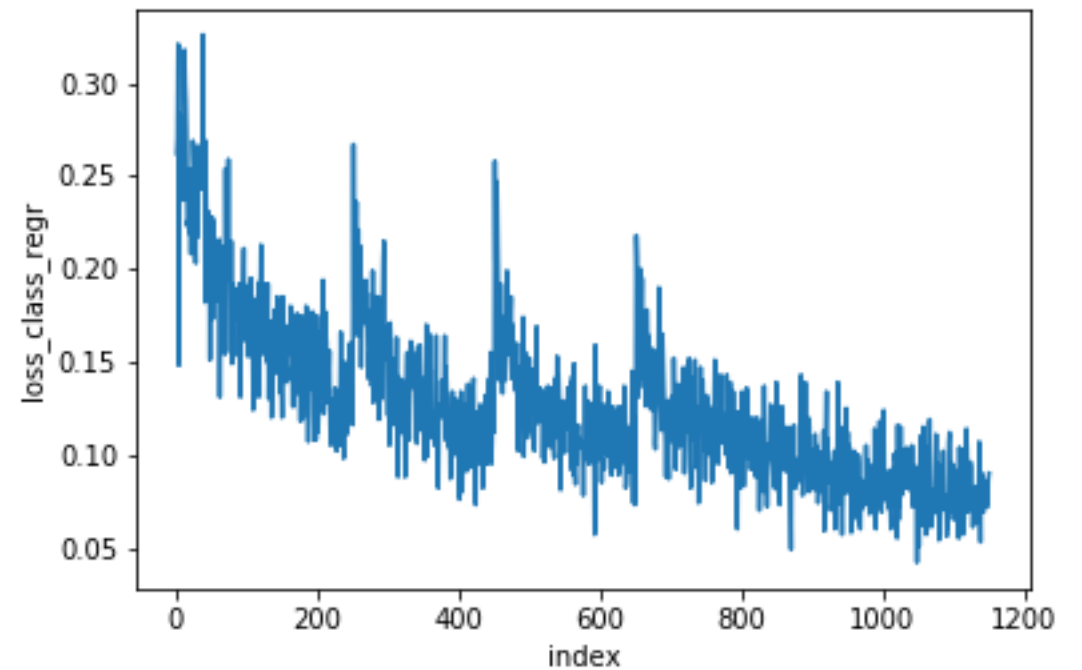
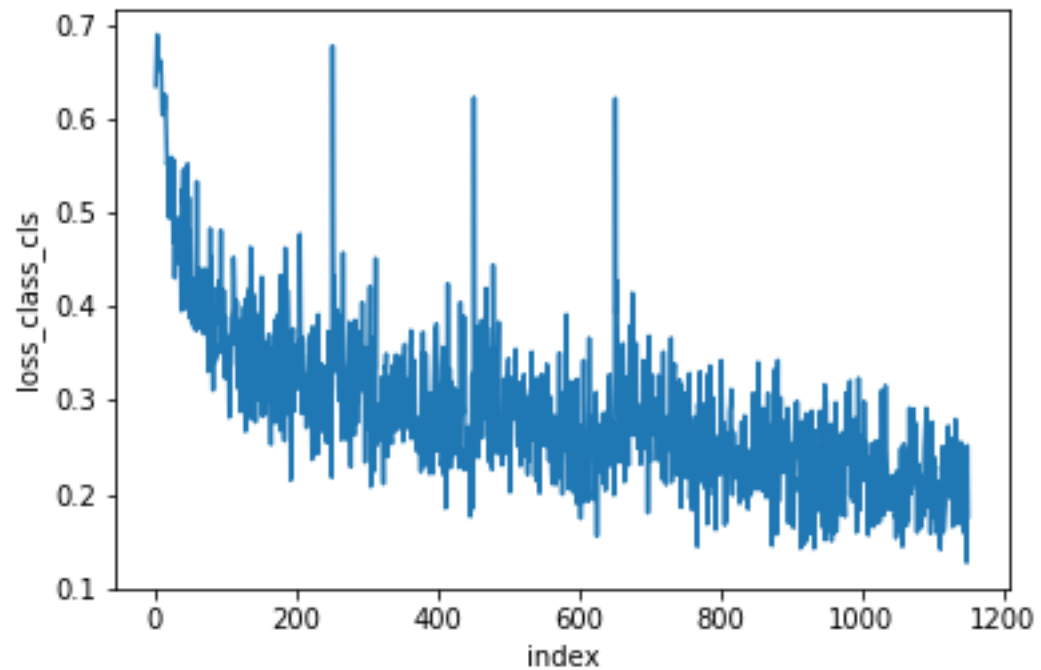
Results (1)



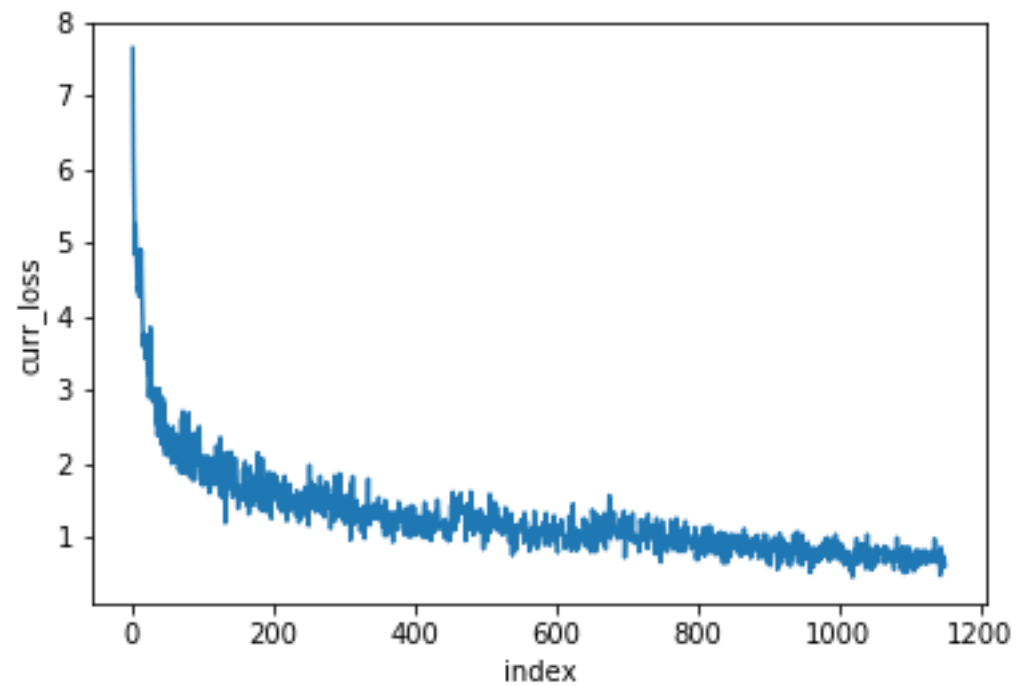
Results (2)



Results (3)



Results (4)





Deploying our solution on the web

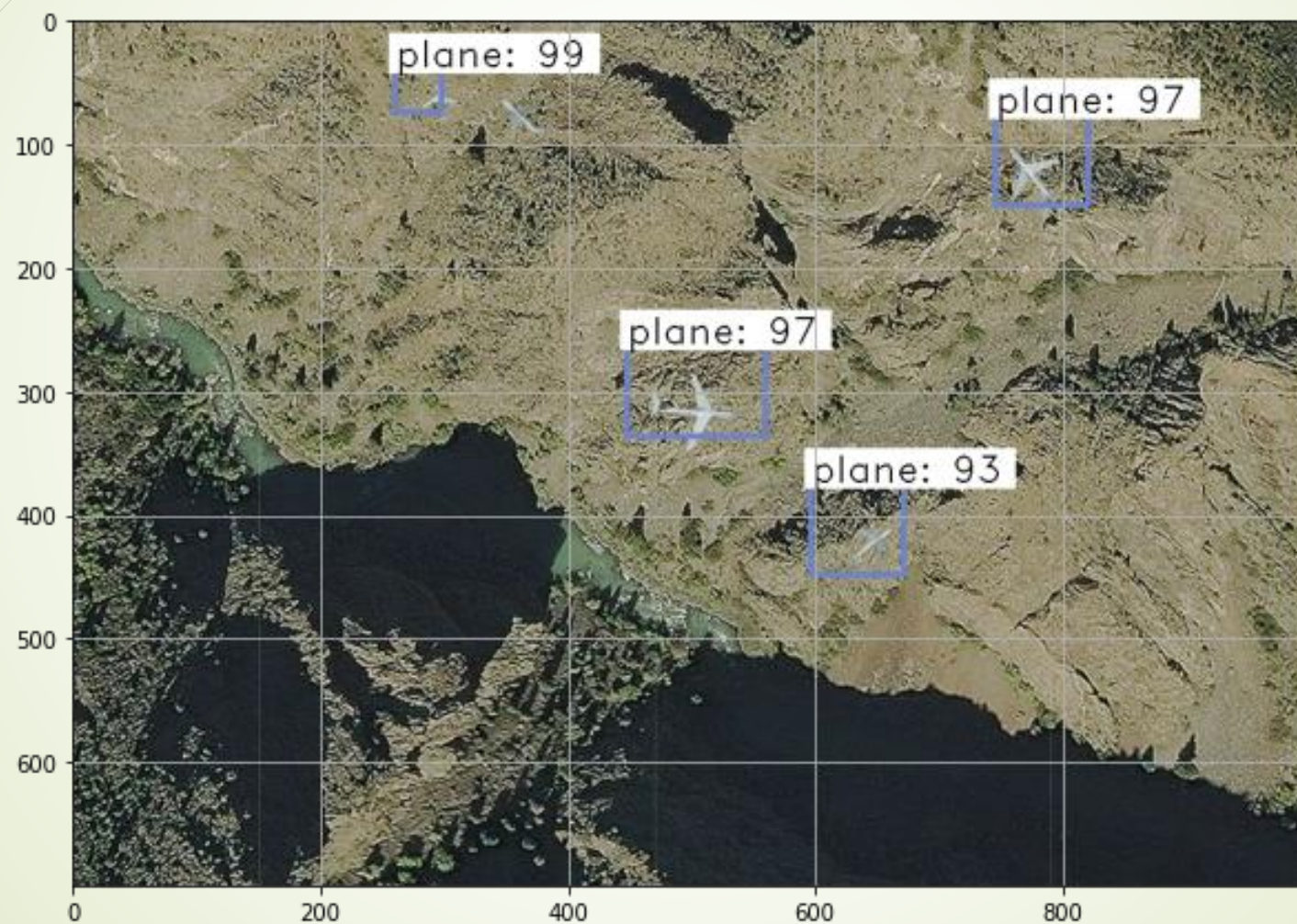
- Open source python web app tool **Anvil**
- The backend server script hosted on google Colab, with the webpage feeding any image input into the validation code of the model, and displaying the output images
- Demo of our web app follows – [Click here](#)

BTP-II Airplane detection using FRCNN

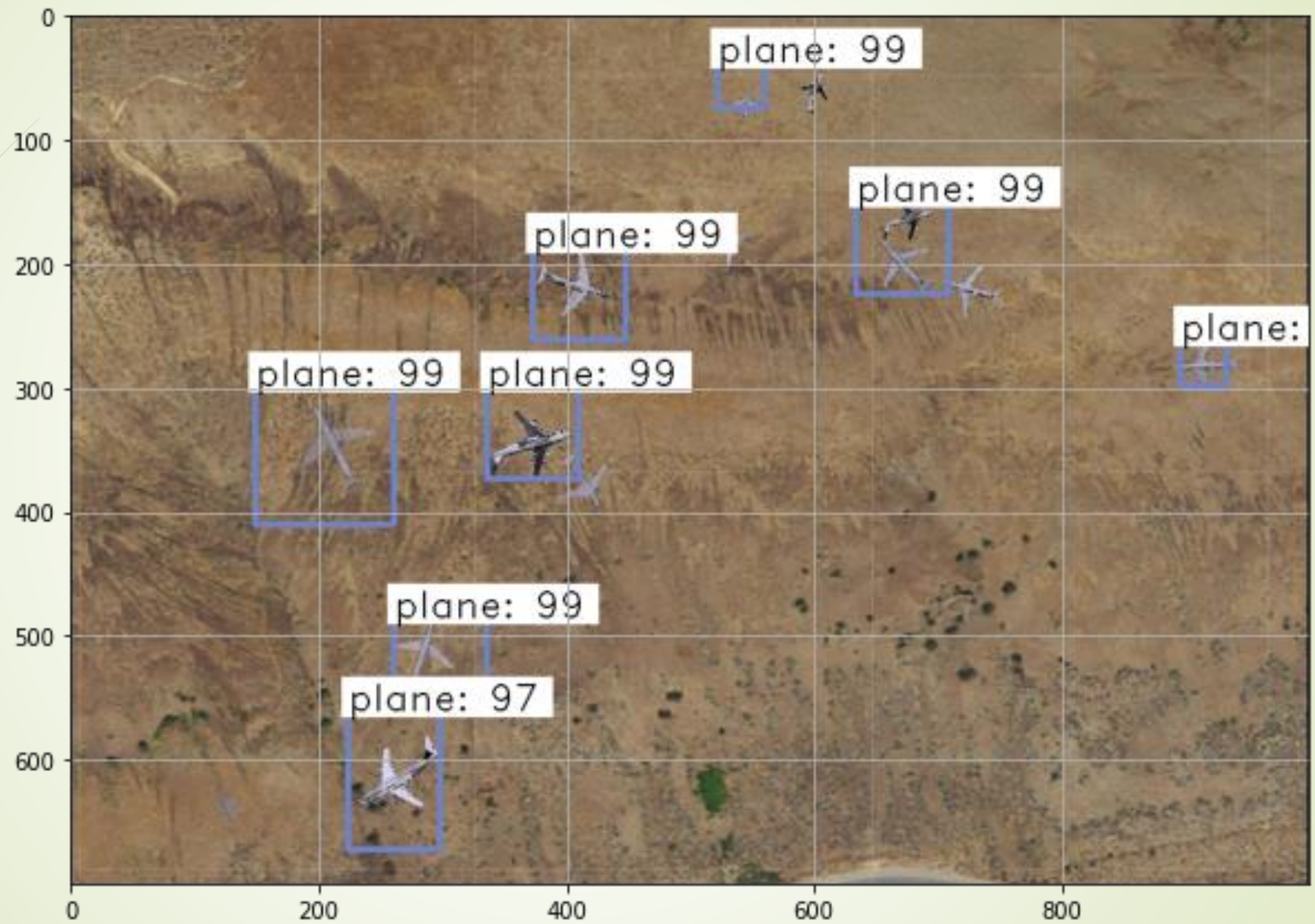
 [UPLOAD INPUT IMAGE](#)

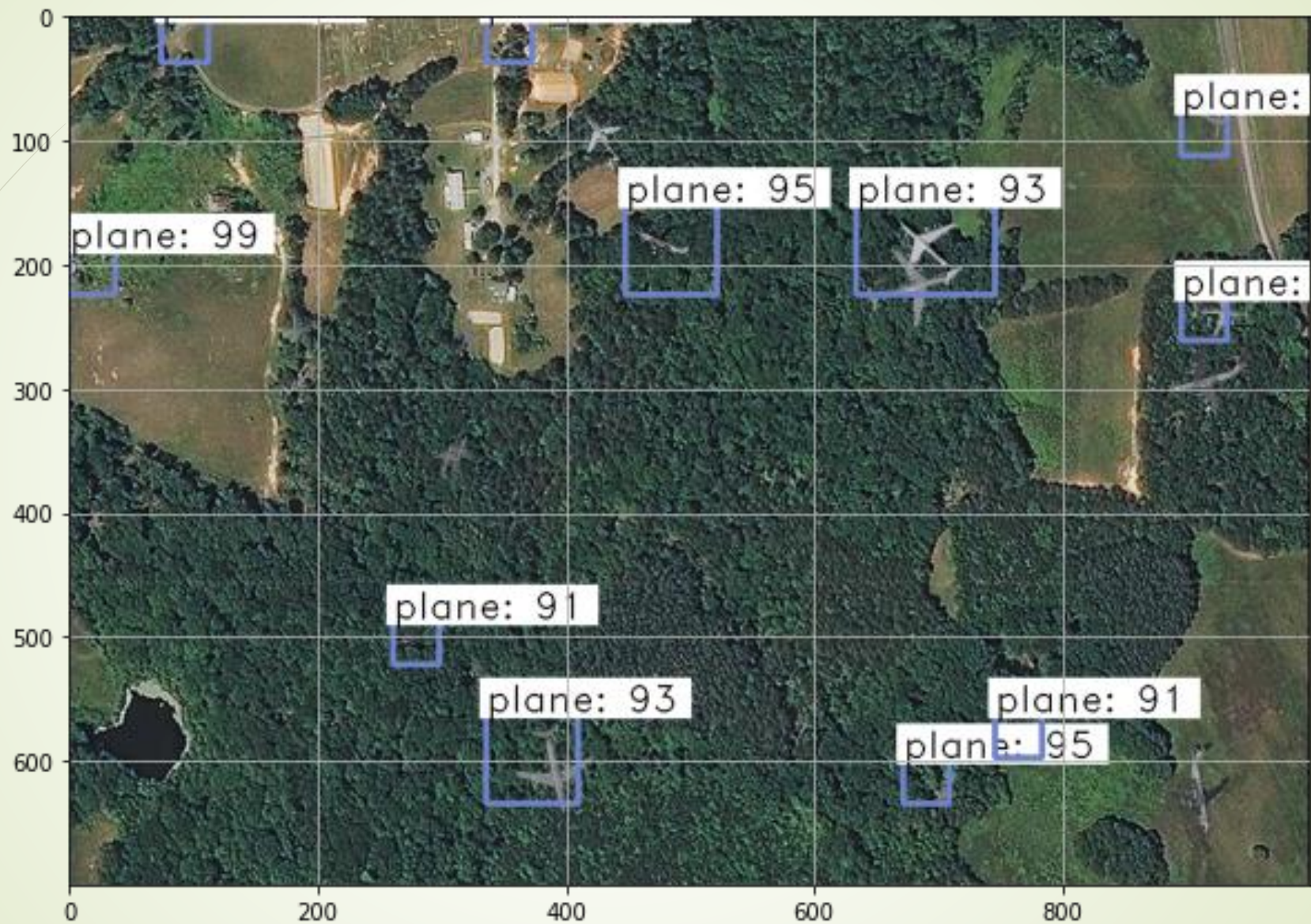
- Server based UI platform for any user to perform aircraft detection

Some Results from testing









Special Credit

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

Abstract—State-of-the-art object detection networks depend on region proposal algorithms to hypothesize object locations. Advances like SPPnet [1] and Fast R-CNN [2] have reduced the running time of these detection networks, exposing region proposal computation as a bottleneck. In this work, we introduce a *Region Proposal Network* (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We further merge RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with “attention” mechanisms, the RPN component tells the unified network where to look. For the very deep VGG-16 model [3], our detection system has a frame rate of 5fps (*including all steps*) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks. Code has been made publicly available.

Index Terms—Object Detection, Region Proposal, Convolutional Neural Network.

arXiv:1506.01497v3 [cs.CV] 6 Jan 2016

1 INTRODUCTION

Recent advances in object detection are driven by the success of region proposal methods (*e.g.*, [4]) and region-based convolutional neural networks (R-CNNs) [5]. Although region-based CNNs were computationally expensive as originally developed in [5], their cost has been drastically reduced thanks to sharing convolutions across proposals [1], [2]. The latest incarnation, Fast R-CNN [2], achieves near real-time rates using very deep networks [3], *when ignoring the time spent on region proposals*. Now, proposals are the test-time computational bottleneck in state-of-the-art detection systems.

Region proposal methods typically rely on inexpensive features and economical inference schemes. Selective Search [4], one of the most popular methods, greedily merges superpixels based on engineered low-level features. Yet when compared to efficient detection networks [2], Selective Search is an order of magnitude slower, at 2 seconds per image in a CPU implementation. EdgeBoxes [6] currently provides the best tradeoff between proposal quality and speed, at 0.2 seconds per image. Nevertheless, the region proposal step still consumes as much running time

One may note that fast region-based CNNs take advantage of GPUs, while the region proposal methods used in research are implemented on the CPU, making such runtime comparisons inequitable. An obvious way to accelerate proposal computation is to re-implement it for the GPU. This may be an effective engineering solution, but re-implementation ignores the down-stream detection network and therefore misses important opportunities for sharing computation.

In this paper, we show that an algorithmic change—computing proposals with a deep convolutional neural network—leads to an elegant and effective solution where proposal computation is nearly cost-free given the detection network’s computation. To this end, we introduce novel *Region Proposal Networks* (RPNs) that share convolutional layers with state-of-the-art object detection networks [1], [2]. By sharing convolutions at test-time, the marginal cost for computing proposals is small (*e.g.*, 10ms per image).

Our observation is that the convolutional feature maps used by region-based detectors, like Fast R-CNN, can also be used for generating region proposals. On top of these convolutional features, we construct an RPN by adding a few additional convolutional layers that simultaneously regress region

- The entire Faster RCNN was developed and published as open source by a team at Microsoft Research composed of Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun
- We have used a version of their development and tweaked as per our need.



Thank You