Pedestrian Detection for Drone Surveillance

Prashant Kumar Sharma (173059007) Debanjan Das (173050069)

Acknowledgment

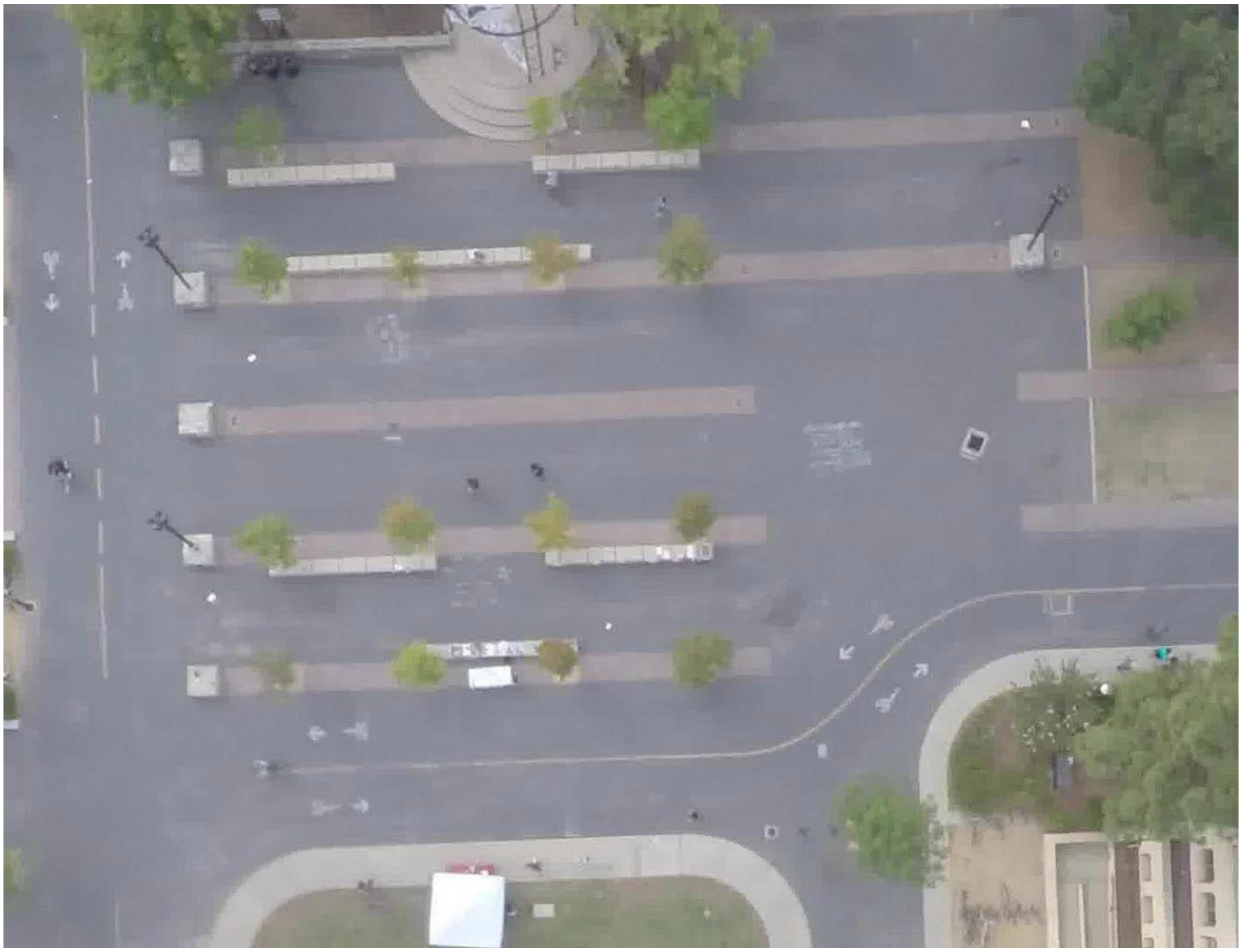
We would like to express our special thanks and gratitude to our guide, Prof. Manoj Gopalkrishnan and Mr. Dhiraj Dhule (Director of FlytBase) who guided and provided us with the necessary insights and materials on the given problem statement “Pedestrian Detection for Drone Surveillance”.

Abstract

Pedestrian detection from aerial view is a challenging task because as the height increases, it becomes a challenge to detect an object occupying less no. of pixels. Our goal was to have a model which can perform well at this task. And investigate model compression so that it is easy to deploy the model on a quadcopter. In this project, we have finally used RetinaNet architecture on Stanford Drone Dataset. Our approach achieved an F1 score of 78% on the final dataset.

Introduction

Why pedestrian is a challenge for surveillance from an aerial view?
Pedestrian detection is one of the major tasks in the field of automated surveillance. It can be defined as the detection of pedestrians present in a particular scene. The images of scenes are mostly taken from a drone or CCTV cameras. In our project, we have focused on the scenes taken from drones at a height of nearly 50 meters from the ground. At this height, the pedestrians look like a small dark object as described in the below figure.



The pedestrians in the above image occupy very less number of pixels. This makes the task of pedestrian detection challenging from drone cameras where the height may vary with time.

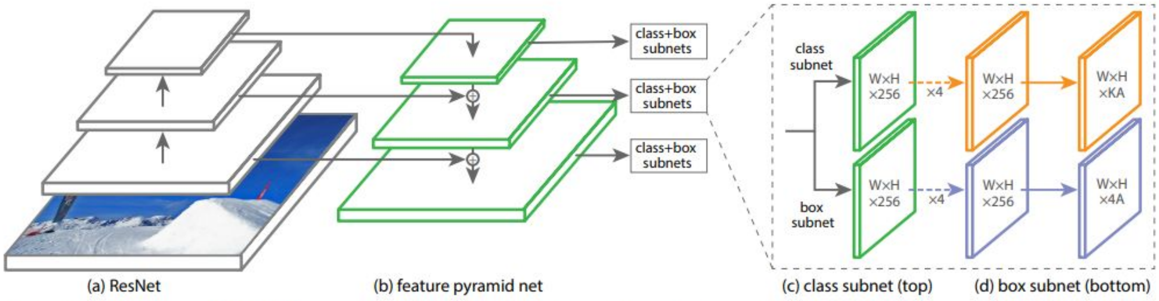
There are several object detection methods present which perform very good on normal images (i.e, taken on the ground), e.g, YOLO, Singleshot Multibox Detector, etc. But, they generally fail on detecting small objects and densely populated objects in an image. Hence, we went for RetinaNet as this network is mainly built for detecting hard-to-detect objects. RetinaNet redefined their loss function, termed as focal loss which basically helps to detect these objects. As pedestrian occupies a very low number of pixels in an image thus it becomes a reasonable choice to directly experiment with RetinaNet.

This project was kind of a collaboration project with FlytBase. According to the Director of FlytBase, the pedestrian detection from aerial images taken from drones is still an unsolved problem at the production level in India. Only Japan is actively working on solving this problem at the industry level.

Approach

Pedestrian detection: Why retinanet is better than previous ? Why we chose retinanet is explained below:

- The architecture of RetinaNet is shown below.



(a) ResNet (b) feature pyramid net (c) class subnet (top) (d) box subnet (bottom)

Figure 3. The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (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 [20] while running at faster speeds.

RetinaNet was published in 2018 by Facebook research.

Mainly object detection architecture are split into two categories: single-stage and two-stage

Two-stage architectures try to categorize objects in the images as foreground or background. Then at later stage objects are further classified into fine-grained classes: pedestrian, biker, car, etc. Example: is Faster-RCNN.

Single-stage architecture doesn't split into foreground and background. They are faster than two-stage architecture but less accurate. RetinaNet is an exception, it has two-stage performance while having single-stage speed.

The main components of Retinanet architecture are :

- CNN
  ResNet-50 is used as a backbone network. And input image is fed and processed via convolution kernels to generate output feature map. First few feature map capture lines, edges, etc. and later feature maps capture higher level feature eg. structure of the face, car, etc.
- Feature Pyramid network
  After retinanet is passed through Resnet, a feature pyramid network is added to the architecture by picking feature maps at various layers from ResNet, to get rich and multi-scale features.

  We first upsample the low-level feature maps then add to higher level feature maps. Upsampling is done using nearest neighbor method.
- Anchors

For every FPN level, various anchors are moved over FPN's feature maps. Anchors could be of different sizes and ratios. These anchors are base position of potential object. They are scales according to the dimension of FPN level.

- Fully Convolutional Networks (FCN)
  Every FPN level is input to FCN , which are neural networks with convolutions and pooling operations.

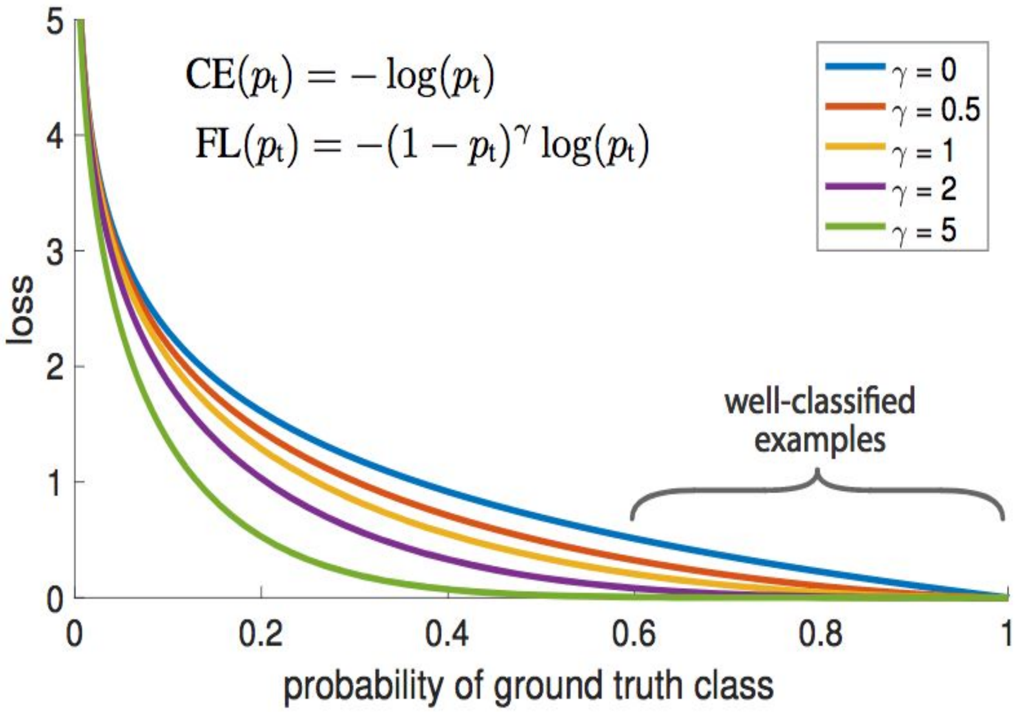
  The first FCN is for regression it predicts 4 values x1, y1, x2, y2 for each anchor.

  The second FCN is for classification. The classifier predict N classes using softmax.

<u>Removing duplicates</u>

There could be multiple box prediction for an object. Thus for each class non-max max supression is applied
tf.image.non_max_supression(boxes, scores, max_output_size, iou_threshold)

- Focal loss
  This is the key point of retinanet:



As it can be seen that the the cross entropy loss is weight by a factor of (1 - Pt)^λ. This means that the loss for well classified example is very low, and forces the model to learn on harder examples.

Neural Network compression

Two ways to do it :

1) By making architectural changes: eg. Squeezenet (https://arxiv.org/abs/1602.07360)

    a) Inspired from VGGNet, GoogleNet, ResNet
        i) Use of small sized kernel
        ii) Information flow at multiple scales
        iii) Use of bypass connectoins
    b) Down sample later in network to maintain large activation maps

2) Deep comprssion
    a) Network pruning: all weights below some threshold are removed from the network
    b) Quantization and weight sharing: weights are quantized into bins and are referred by index values
    c) Huffman encoding: distribution of weights are biased so encoding them save memory.

Observation:

- Maxpool over average pool
- Decrease in number of feature maps deep in the architeacture
- Number of feature maps per layers depends upon the type of dataset chosen
- Fire models should be implemented later in the network
- Replacing 5x5 with two 3x3 results in slight drop in accuracy

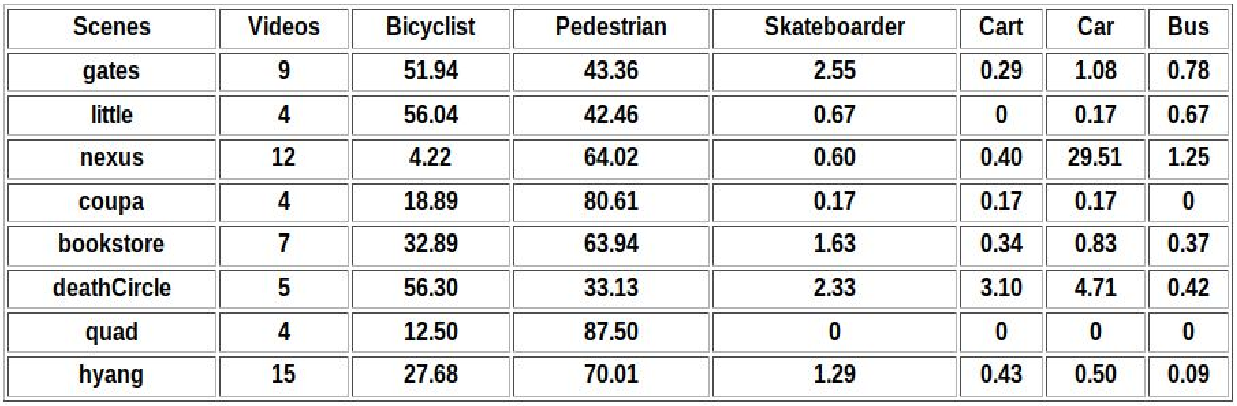
Experiment

Pedestrian detection

In this section, we will first describe the dataset, our experiment pipeline: consisting of data preprocessing, model architecture, inference and results. All the codes related to our experimentations in compression and pedestrian detection can be found on GitHub here: https://github.com/prashantksharma/ee763

Dataset

We used the Stanford Drone Dataset for our training purpose (http://cvgl.stanford.edu/projects/uav_data/). Statistics of the dataset is as follows



Scenes	Videos	Bicyclist	Pedestrian	Skateboarder	Cart	Car	Bus
gates	9	51.94	43.36	2.55	0.29	1.08	0.78
little	4	56.04	42.46	0.67	0	0.17	0.67
nexus	12	4.22	64.02	0.60	0.40	29.51	1.25
coupa	4	18.89	80.61	0.17	0.17	0.17	0
bookstore	7	32.89	63.94	1.63	0.34	0.83	0.37
deathCircle	5	56.30	33.13	2.33	3.10	4.71	0.42
quad	4	12.50	87.50	0	0	0	0
hyang	15	27.68	70.01	1.29	0.43	0.50	0.09

For our purpose, we focussed on detecting just two classes: pedestrian and biker (i.e bicyclist).

From the videos given in the dataset, we prepare datasets of the following type

- ~100 image samples: consisting of ~2144 objects in total across all images
- ~1k image samples: consisting of ~15545 objects in total across all images
- ~ 2k image samples: consisting of ~19926 objects in total across all images

Pipeline

Coming to the pipeline, we have data preprocessing, model architecture implementation in Keras, inference on video input

Data pre-processing

Data pre-processing is the most important part of the pipeline as it deals with capturing the variation of the real world into the training sample that we prepare. The process of preparing the training set is as follows.

1. Take the input raw videos
2. Decompose the video into frames using https://github.com/prashantksharma/ee763/blob/master/notebooks/video2frame.ipynb
3. Sample non-duplicate frames across various categories of videos using https://github.com/prashantksharma/ee763/blob/master/notebooks/mapping_final.ipynb for both training and test sample generation. One thing to note is that we have relied on heuristics to capture non-duplicate frames. For example, we have capture frames every second. This could be automated further is if incorporate methods to calculate the similarity between two frames.
4. Before we train our mode we need to pre-process the annotations given in the Stanford Drone Dataset using https://github.com/prashantksharma/ee763/blob/master/notebooks/test_train_split.py

Some statistics for data pre-processing are:

1 raw input videos of 7-10 minutes, we get 12-14k get frames after decomposition, approximately getting frames at the rate of ~30 FPS, out of which we sample 200 frames at the interval of 25 frames. This is done for both train and test set.

This is done for videos related to 8 scenes (http://cvgl.stanford.edu/projects/uav_data/)

Mode architecture

There are 3 key components to retinanet architecture:

1. ResNet50 as backbone
2. Feature Pyramid Network
3. Regression and classification using Fully connected network

The activation function used is ReLU. The learning rate we used is 1e-5, with Adam optimization technique. Used it focal loss as proposed in the retinanet paper:
https://arxiv.org/pdf/1708.02002.pdf

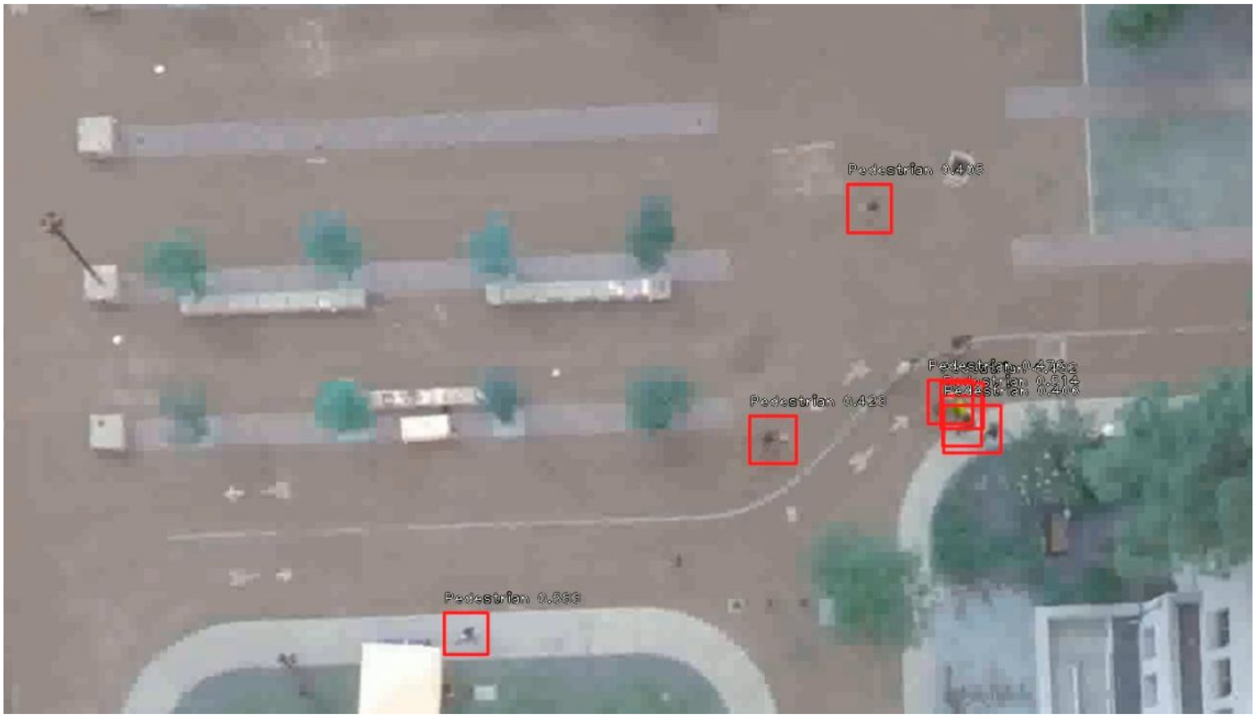
Also, for regression on bounding boxes: L1 loss has been used, and for classification task focal loss has been used.

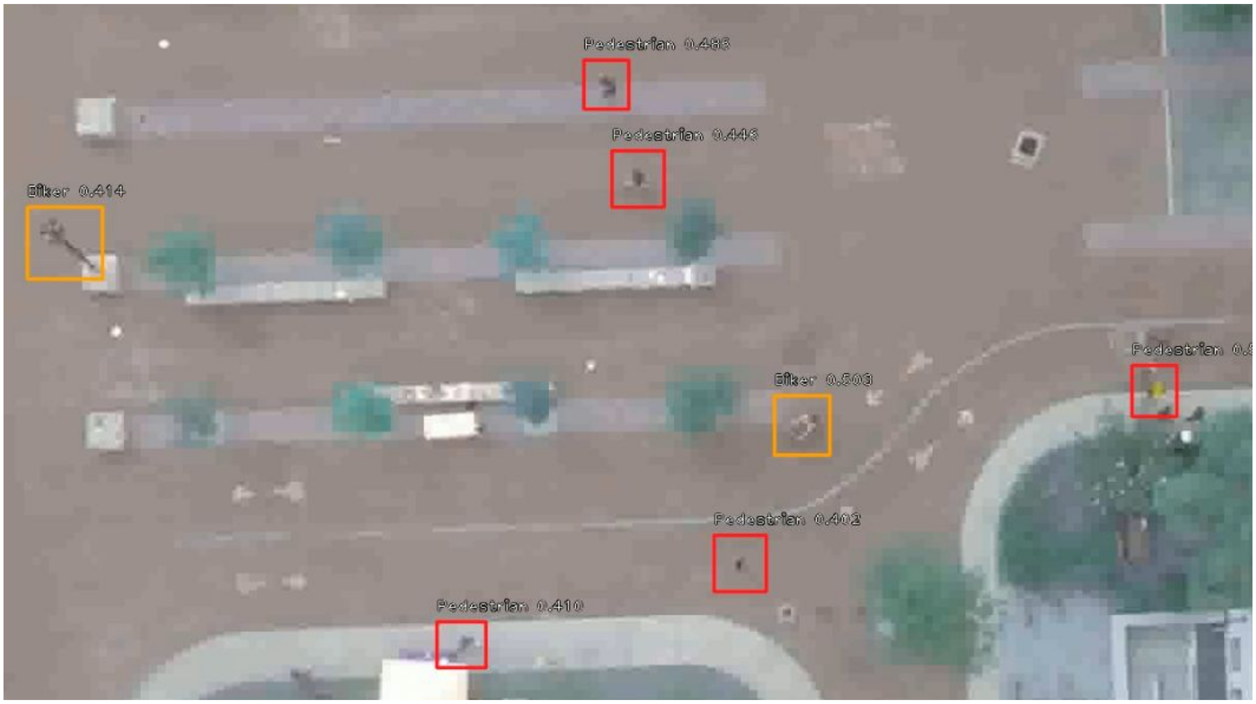
The model has been implemented using Keras and Tensorflow framework.

- Training:
  We trained the mode for two datasets consisting of 1k and 2k data samples
  The model was trained for 50 epochs. The results consisting of loss, F1 scores can be seen in the *result* section below.

- Inference:
  For making inference on test videos, first, they need to be decomposed into frames using `cv2.VideoCapture` object of OpenCV library. Calling `get()` methods on the object givens us number of frames. We iterated of reach frame using `read()` function on `cv2.VideoCapture` object we can read individual frames then make a prediction using the pre-trained network and after then stitch the inference frames into a video using `cv2.VideoWriter` object.
  Some stats are:
  For 1-minute test video, we get ~1800 frames of size (1424x1088) in ~250 seconds on GPU this will be about ~7-8 FPS.

  The output of the inference is shown below.

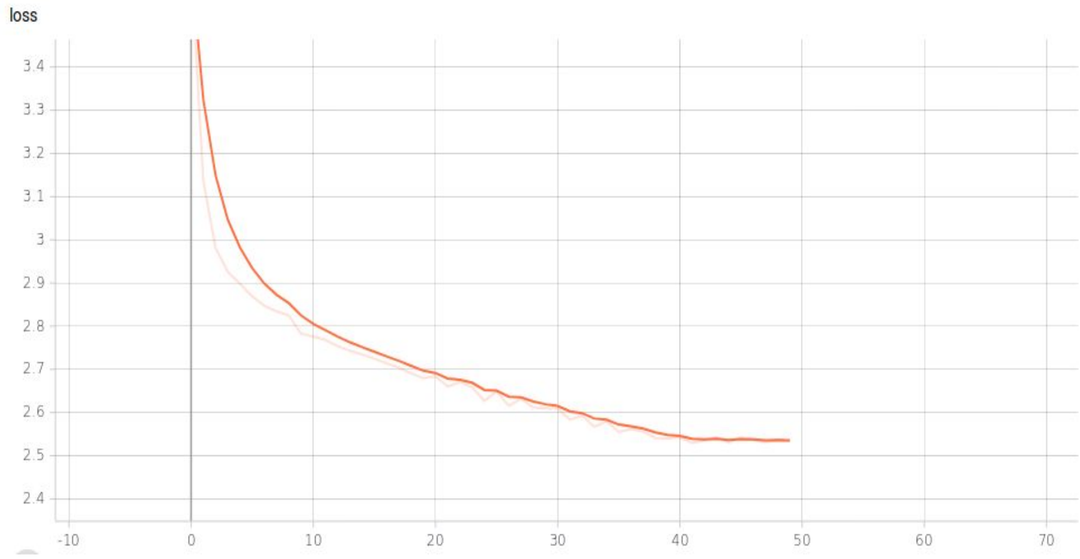


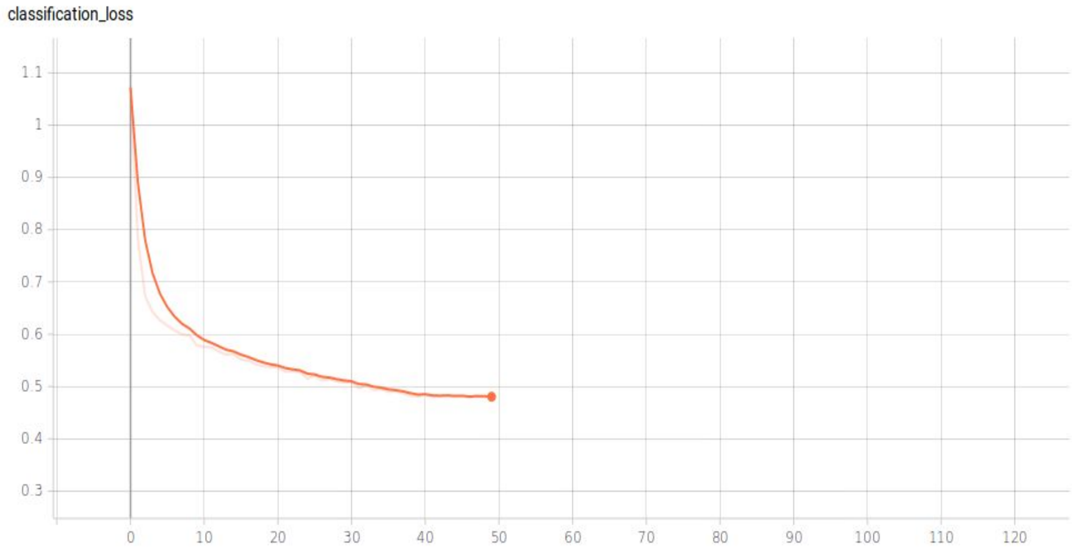


Results

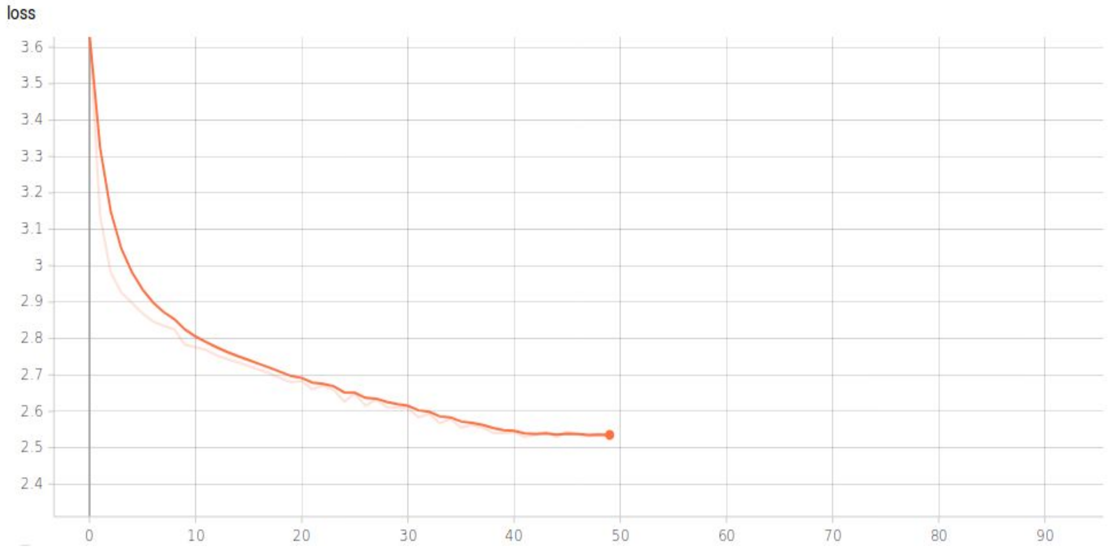
Loss

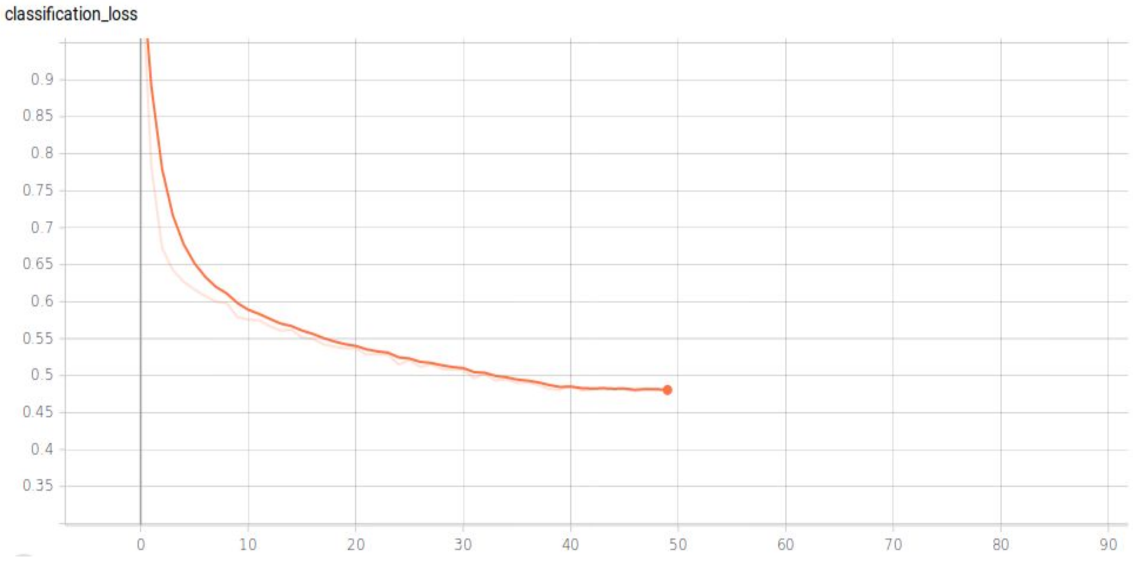
1000 data samples





2000 data samples





F1 scores

The scores are calculated on ~1k and ~2k validation data.

	Precision	**Recall**	**F1**
1k data (~ 1000 samples)	**71.9285**	**77.24773**	**74.4932**
2k data (~ 2000 samples)	**79.4097**	**77.831977**	**78.6288**

Neural network compression

Dataset

MNIST, CIFAR10

Results

- LeNet-5 over MNIST dataset
  Original architecture: The model size of 270KB, accuracy 98.9%
  Squeezenet: Model size 141.4 KB, accuracy 98.4%
  Squeezenet + our observation: The model size of 72 KB, the accuracy of 98%
- VGGNet over CIFAR10 Dataset
  Original architecture: models size of 36 MB, accuracy ~93.5 %
  Squeezenet architecture + our observation: Models size ~90.8 %

Conclusion and Future work

RetinaNet has worked quite well for detection from an aerial view and predicting pedestrian, occupying fewer pixels. Future work would entail, customizing our model for suiting Flytbase need by training on their custom dataset and deploying the model on their production pipeline. The experiments we performed are on small datasets. In fututre, we would also like to increase the dataset amount which will lead to a better F1 score.

In the context of neural neural compression we want to experiment with architecture changes suggested by Squeezenet results in significant compression of the model without much loss in accuracy. Going forward we would like to incorporate those changes for RetinaNet architecture. And also, try deep compression techniques for furture compression.