# Training Visible Body,Head detection on CrowdHuman Dataset

Using TensorFlow Object Detection API

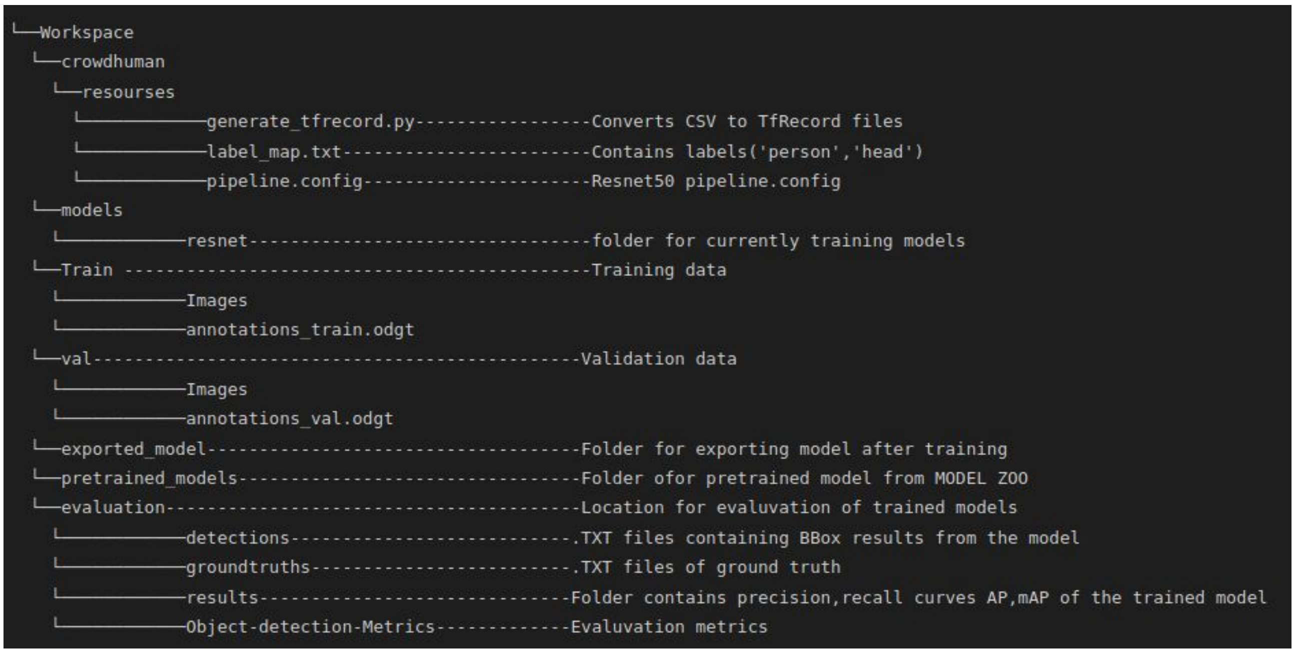
https://github.com/ManojKesani/crowdhuman

## Table of content:

## 1.Installing Tensorflow 2.2.0 and setting up Object Detection API

We start with pip install tensorflow==2.2.0 and then clone Tensorflow object detection api from github.

## 2. Preparing workspace



```
└─Workspace
  └─crowdhuman
    └─resourses
      └────────────generate_tfrecord.py-----------------Converts CSV to TfRecord files
      └────────────label_map.txt------------------------Contains labels('person','head')
      └────────────pipeline.config----------------------Resnet50 pipeline.config
  └─models
    └────────────resnet---------------------------------folder for currently training models
  └─Train ----------------------------------------------Training data
    └────────────Images
    └────────────annotations_train.odgt
  └─val------------------------------------------------Validation data
    └────────────Images
    └────────────annotations_val.odgt
  └─exported_model-------------------------------------Folder for exporting model after training
  └─pretrained_models----------------------------------Folder ofor pretrained model from MODEL ZOO
  └─evaluation-----------------------------------------Location for evaluvation of trained models
    └────────────detections---------------------------.TXT files containing BBox results from the model
    └────────────groundtruths-------------------------.TXT files of ground truth
    └────────────results------------------------------Folder contains precision,recall curves AP,mAP of the trained model
    └────────────Object-detection-Metrics-------------Evaluvation metrics
```

Now we create the file structure requried.

The main folder we are working under is work space. It contains :

crowdhuman – my github repo location containing CSV_to_tfrecord converter,labelmap.pbtxt,Pipeline.config for training resnet

- Pipeline.config has Batch size,learning rate,path to train and validation data, model architecture.

Train – Folder for training data
val – Folder for validation data
models – Folder to save currently training structure
pretrained_models – Folder for pretrained weights used in Transfer learning
exported_models – Folder for exporting trained models
evaluation – Folder for evaluating trained models

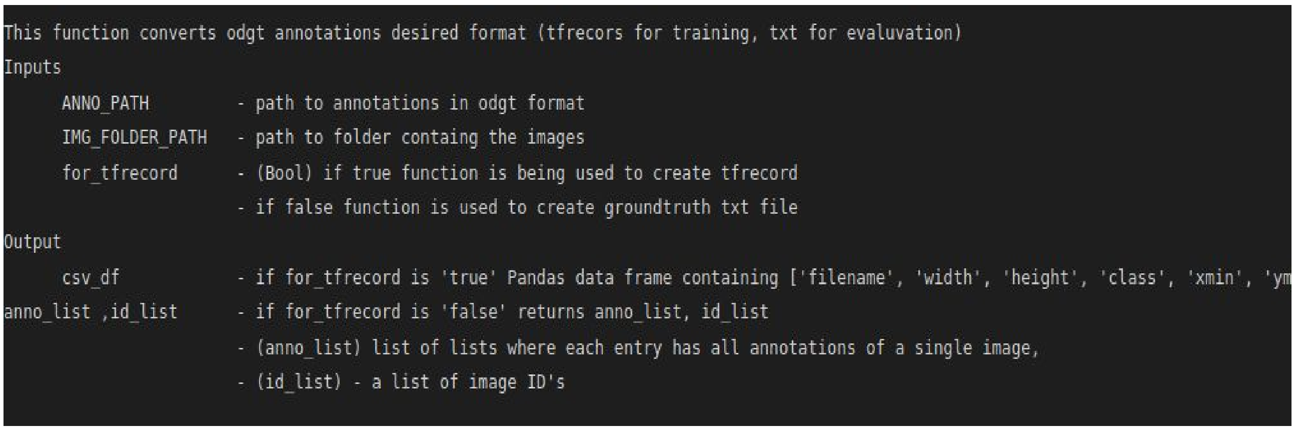
we unzip training data,validation data and annotations into corresponding folders
we download Resnet pretrained model weights and clone my github repo into corresponding folders.

## 3. ODGT_to_CSV

A function used to parse .odgt annotations into requried format.

This is used to parse annotations for training and evaluation.

For training we need a csv file containing [path to image, image width,height,class,Bbox]. Each annotation is a row in the output csv ie., a single image can occupy many rows.
For evaluation we need a .txt file for each image which contains information of all the annotations. Each image has a single .txt file correspondingly

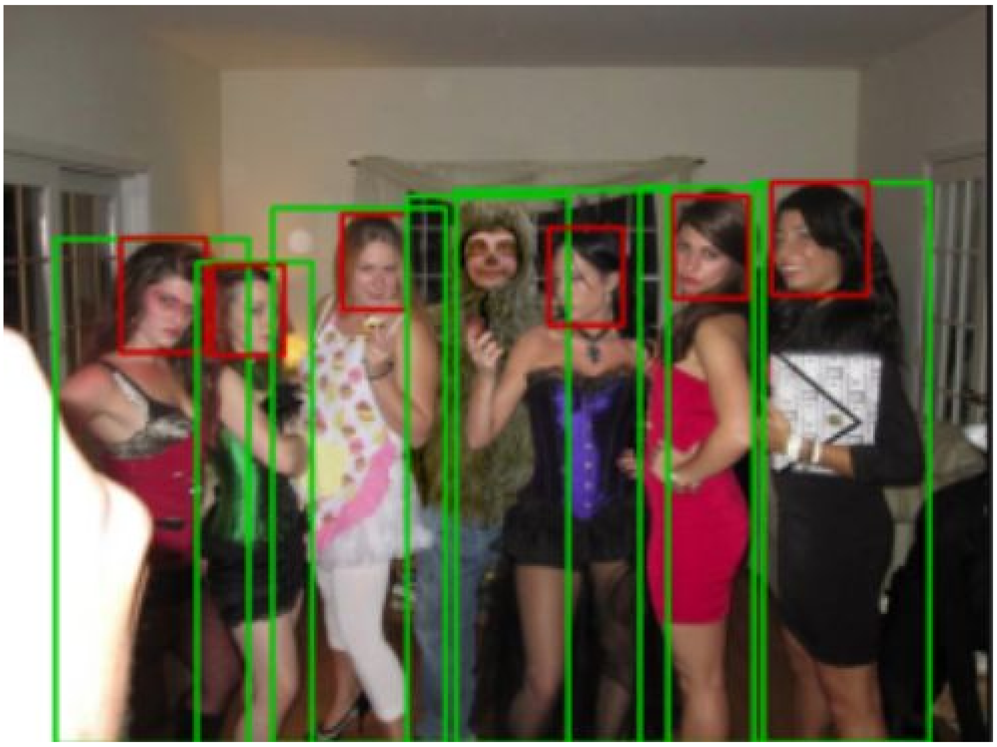


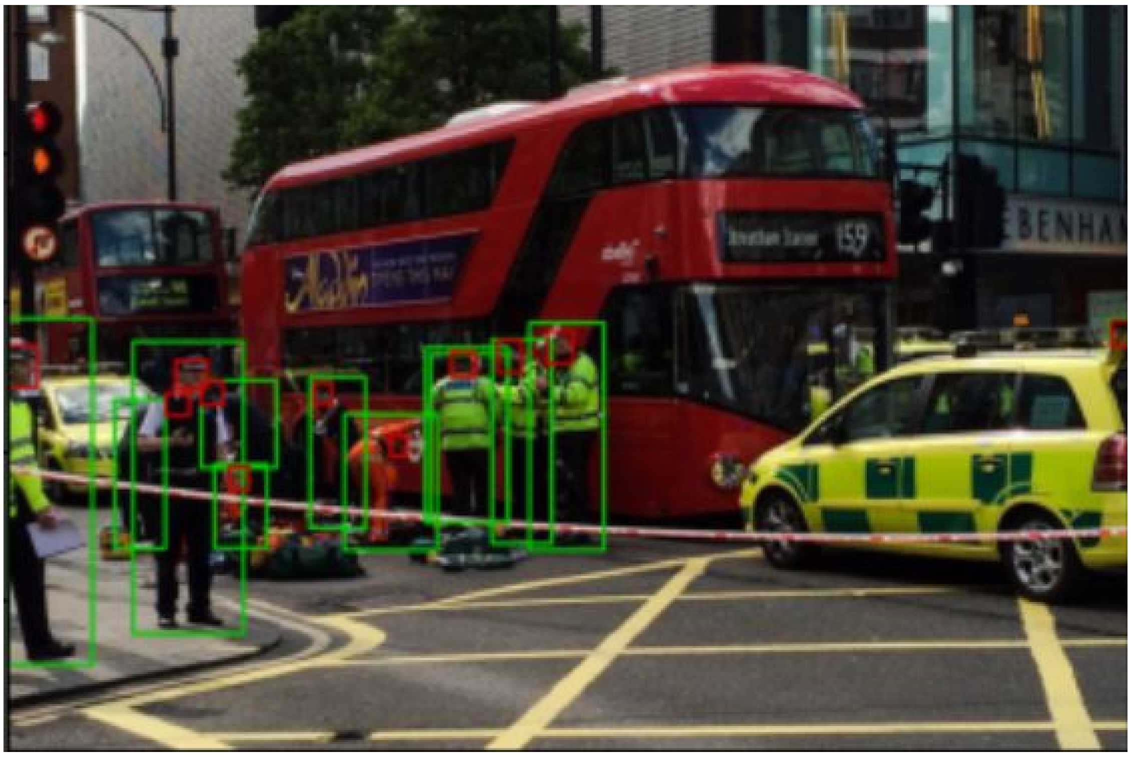
```
This function converts odgt annotations desired format (tfrecors for training, txt for evaluvation)
Inputs
      ANNO_PATH         - path to annotations in odgt format
      IMG_FOLDER_PATH   - path to folder containg the images
      for_tfrecord      - (Bool) if true function is being used to create tfrecord
                        - if false function is used to create groundtruth txt file
Output
      csv_df            - if for_tfrecord is 'true' Pandas data frame containing ['filename', 'width', 'height', 'class', 'xmin', 'ym
anno_list ,id_list      - if for_tfrecord is 'false' returns anno_list, id_list
                        - (anno_list) list of lists where each entry has all annotations of a single image,
                        - (id_list) - a list of image ID's
```

We check if the tag is person and discard the masks. We check if the head attribute exists and isn’t ignored. Based on this we convert the annotations from XYWH to XYRB format, and parse according to the flag ‘for_TFRecord’. If this is set the function is being used to parse to CSV file which thenis converted into tfrecord by generate_tfrecord.py. Else the function is being used to generate .txt files for evaluation.

## 4. Sanity check.

In this we plot few of the annotations on the images.





## 5.Preparing data for training

Now we have all the data and functions ready, we start to convert .odgt to a csv file using odgt_to_csv which is then converted to tfrecord by the generate_tfrecord script.

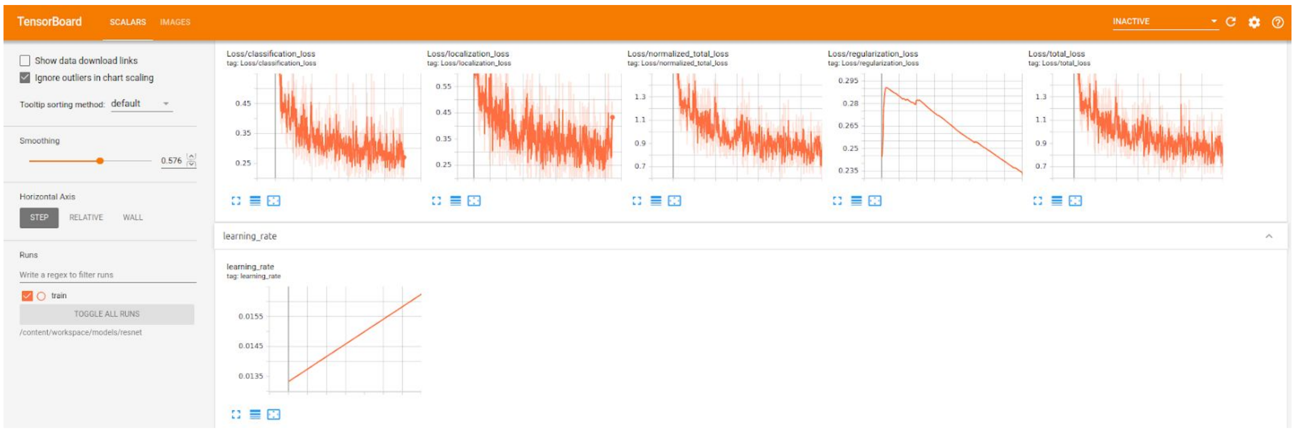
## 6. Training Resnet50 model

Training is done using the script model_main_tf2.py from the object detection api.
It requires model_dir, a place to store the checkpoints,weights, train history events and a pipeline config containing all the details for training. This is in my github repo and can be copied into the folder.

## 7.My Experiments

Exp 1

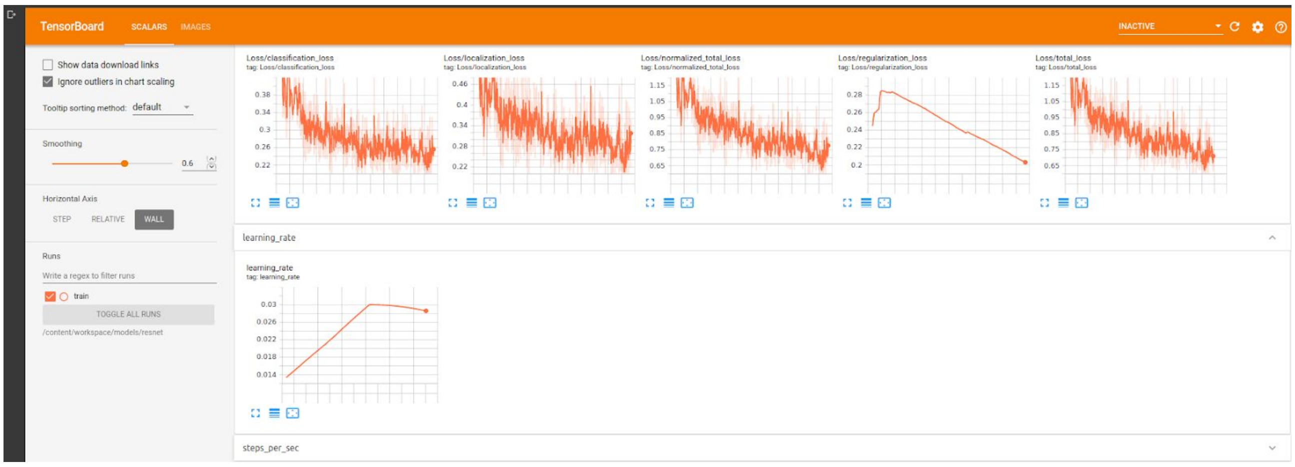
I started off with a 0.0013 learning rate and a monotonically increasing curve to judge the loss terrain.



Tooltip sorting method: default
Smoothing 0.576

Exp 2

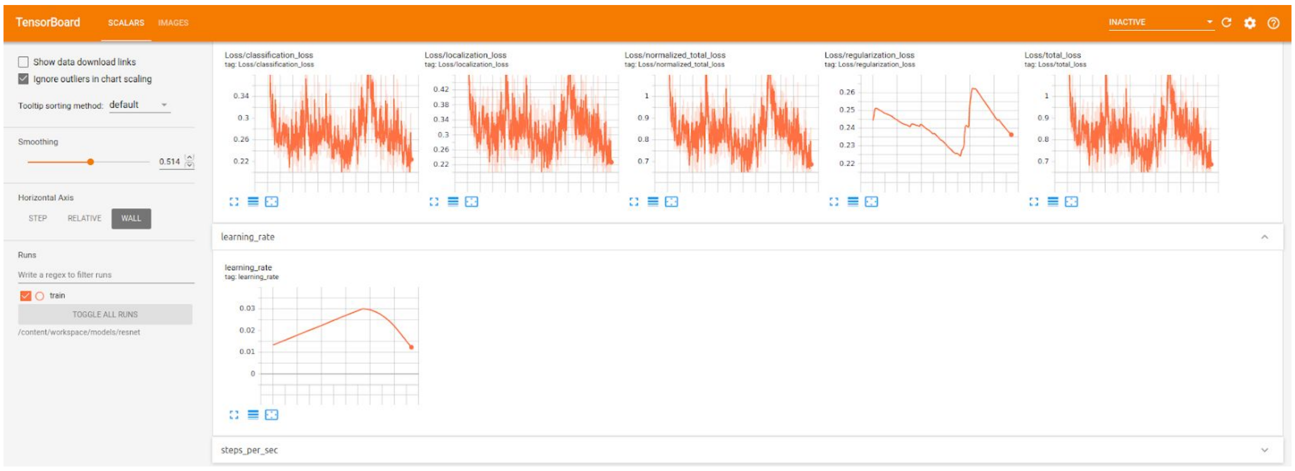
Then with the same initial learning rate that goes to 0.003 monotonically and then slowly decreases.



Tooltip sorting method: default
Smoothing 0.6

Exp 3

Similar to the above but the reduction is steeper.



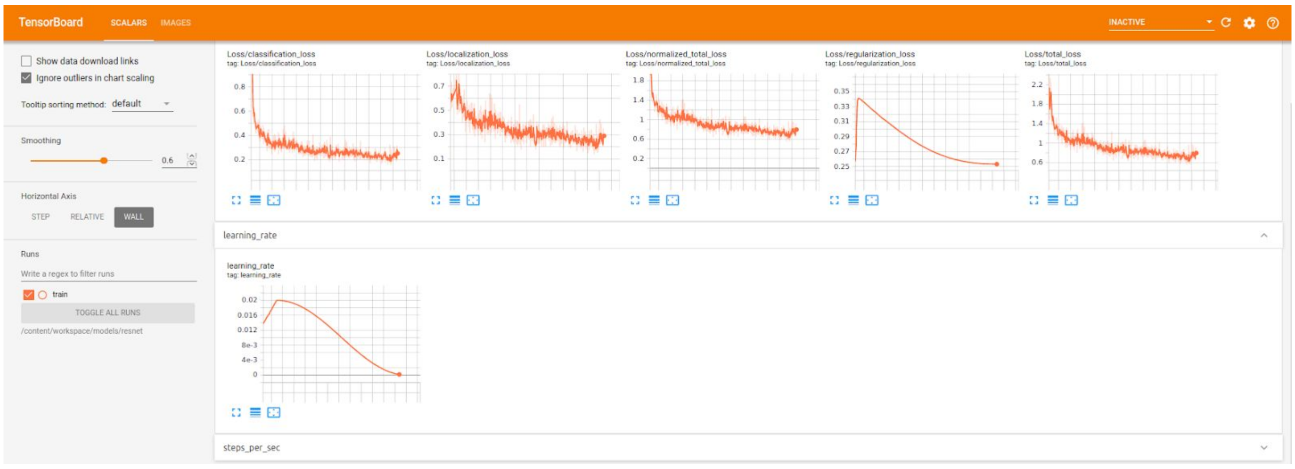
Finally Exp 4 :

We start off with 0.0013 and increase to 0.002 within 500 steps and then drop off to 0 by 5000th step.

The Idea here is that :

Batch size = 4, total number of images = 15000.

so by (15000/4) = 3750 steps a single Epoch is completed and the rest 1250 setp the model revisits the initial images. So we keep the learning rate during this small and decreasing.



## 8. Exporting the trained model

The script exporter_main_v2.py from the API is used to export the trained model

## 9. Inference

I have shared the saved model.Downloading this or the file from export we can perform inference.

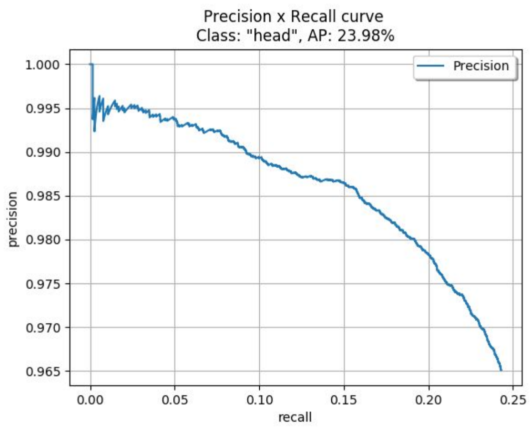
## 10. Model Evaluation

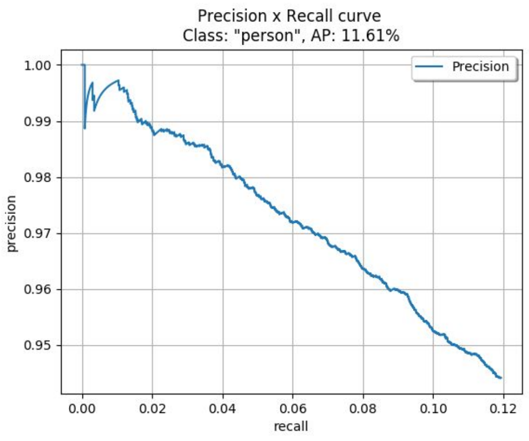
For evaluation we need ground truth labels and predicted labels in a .txt for every single image.

We can now use odgt_to_cvs with flag for_tfrecord = False and parse the validation annotations to get the groundtruth.

For prediction we run the detector and save the outputs with confidence > 0.5. into corresponding .txt files

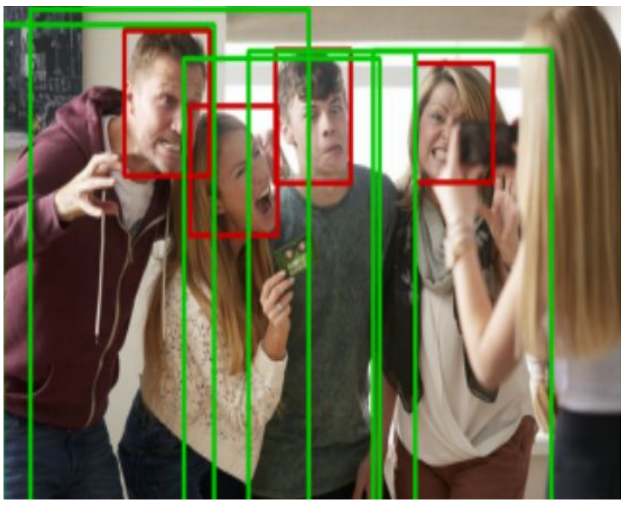
Clone the Object-Detection-Metrics repo and run the pascalvoc.py

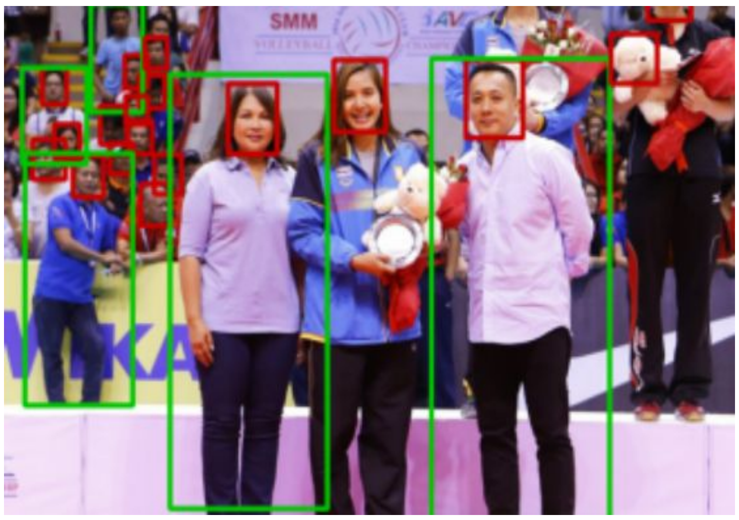




***<u>Results for 4th experiment:</u>***

Head AP =23.98
person AP = 11.61
mAP = 17.79





## 11. Further Improvements

*Monitoring the validation loss* : Tensorflow object detection api the validation is done by opening a new terminal and running the same training script with checkpoint flag. This then runs on the validation data whenever a new checkpoint is saved by the training script. As this isn't feasible on Google Collab I judged the model by the plateau. While training on a local system we can stop training the model when the validation loss diverges from the training loss.

*Model evaluation metrics*: Jaccard Index can be used in place of pascal_voc as it is a better judge for crowded scenarios .Reference

*Model Quantization, Model Pruning:* The trained model can further be optimised to increase the throughput with a tradeoff in accuracy.

By pruning we remove the small weights that aren't much contributing to the total accuracy. In general the weights are in a normal distribution around 0. so by removing them we can decrease the inference time with a small tradeoff with accuracy.

By Quantization we convert float32 weights into int4 or int8 which then can be run on embedded devices.

## 12.References

https://github.com/datitran/raccoon_dataset

https://github.com/rafaelpadilla/Object-Detection-Metrics

https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/

https://developer.nvidia.com/blog/improving-int8-accuracy-using-quantization-aware-training-and-the-transfer-learning-toolkit/