**README v0.0 01 AUG 2019**

**Title: Text Detection using MobileNet SSD.**

**# Introduction/Overview:**

* This task is to detect the text present in the image.
* This is achieved by using TensorFlow's Object detection API.
* The architectural model used is MobileNet Single Shot detector.
* Purpose behind the task is to detect/locate the text in the images with proper bounding box showing the confidence score for text being detected.

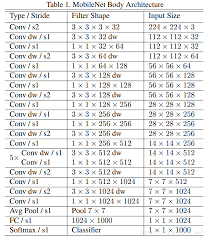
**# Why MobileNet SSD?**

* Comparing with all other models like YOLO, Inception, Resnet, AlexNet and even small models like Squeezenet, we chose the MobileNet architecture over other models as it is reducing time and platform resource constraints.
* Moreover, in real time applications such as Robotics and Self-Driving cars, we need the object identification and localization keeping the speed in mind.
* MobileNet architecture has core layers based on "Depthwise separable convolutions" and has total 28 layers.
* Depthwise separable convolutions is a form of factorized convolutions which factorizes the standard convolutions into two operations as mentioned below:

>> Depthwise convolution.

>> Pointwise convolution.

* This factorization of standard convolution to depthwise and pointwise convolution drastically leads to reduction in factors like computation cost and model size.
* Adding depth multiplier and resolution multiplier as the main factors that affects the model size.
* As there is always been a trade-off between speed and accuracy, MobileNet SSD makes the model to increase the speed factor and by compromising the accuracy.
* And thus, used in application where speed is the relevant factor like in Self-Driving cars and on platform limited to resources like mobile phones.



**# Difference between Standard and Depthwise separable convolution:**

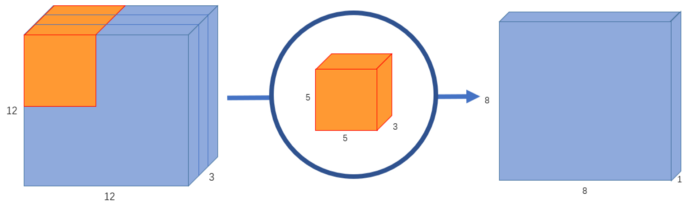
* First, we will understand how the standard convolution works, standard convolution does filter and combining in single step as explained below.

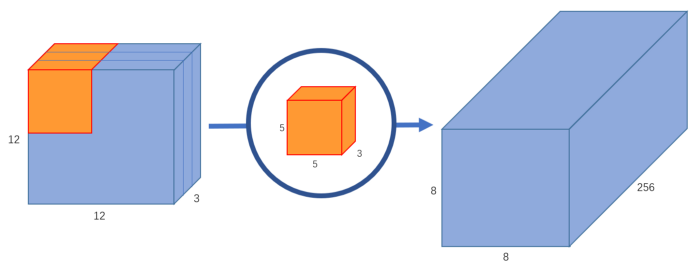
**# Standard Convolution:**

* Consider an RGB image of size (12 x 12 x 3), which is being convolved with (5 x 5 x 3) filter giving the output of (8 x 8 x 1) dimension.

(12 x 12 x 3) ----> (5 x 5 x 3) ----> (8 x 8 x 1).

* But what if I want the output dimension as (8 x 8 x 256), then this makes the reverse that means we will need the (5 x 5 x 3) size filters 256 times with input image of (12 x 12 x 3 x 256), which is something makes change in input dimension.so we need operation of (x 256) to be in kernel multiplication itself.
* This leads to split/factorizes the kernel into 2 parts which leads to introduction of depthwise separable convolution.



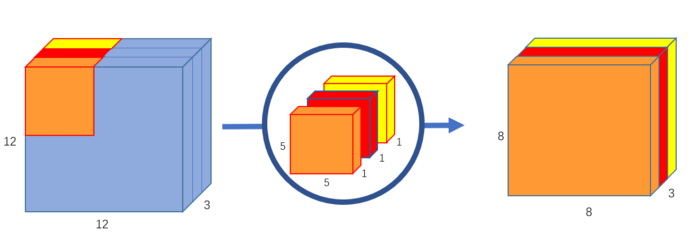


**# Depthwise separable Convolution:**

* Depthwise separable Convolution is achieved in two different steps where filtering step is done in depthwise and combining + elongating is done in pointwise.
* Consider the same as that of the previous one, with (12 x 12 x 3) RGB image.

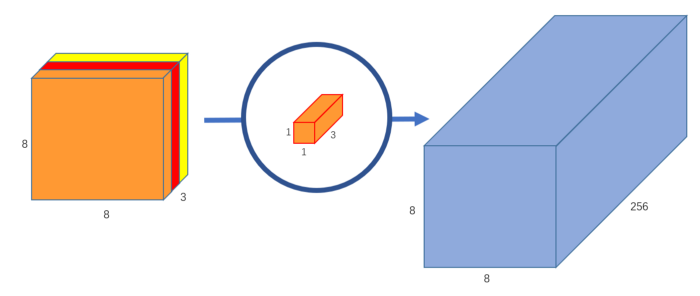
**# Depthwise:**

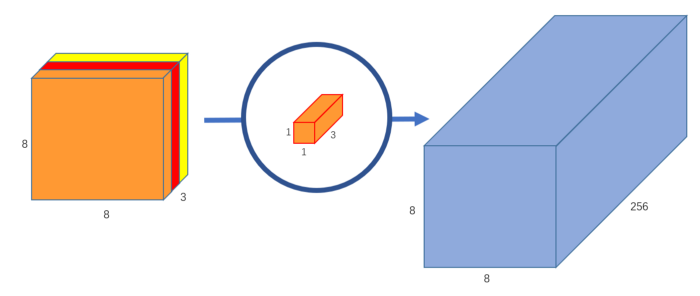
* This (12 x 12 x 3) input image being convolved with [(5 x 5 x 1 x 1)] \* 3 (each for R, G, B filter) which results in output of (8 x 8 x 3) dimension.



**# Pointwise:**

* Now even if we want the final dimension to be (8 x 8 x 256), we can combine the previous depthwise results and elongate to get the one.
* This (8 x 8 x 3) is being multiplied with [(1 x 1 x 3)] \* 256 to get output of (8 x 8 x 256).
* In general (8 x 8 x 3) is being multiplied with [(1 x 1 x 3)] \* mf to get output of (8 x 8 x mf), where mf is multiplying factor.
* In pointwise convolution the kernel dimension is [(1 x 1)] \* 3(for each R, G, B channels).





* More abstractly, we can say that in the normal convolution, we are transforming the image 256 times. And every transformation uses up 5x5x3x8x8=4800 multiplications. In the separable convolution, we only really transform the image once â€” in the depthwise convolution. Then, we take the transformed image and simply elongate it to 256 channels. Without having to transform the image repeatedly, we can save up on computational power.

**# But how it reduces the computational power? ...remains a question??**

**1)** **Standard convolution:**

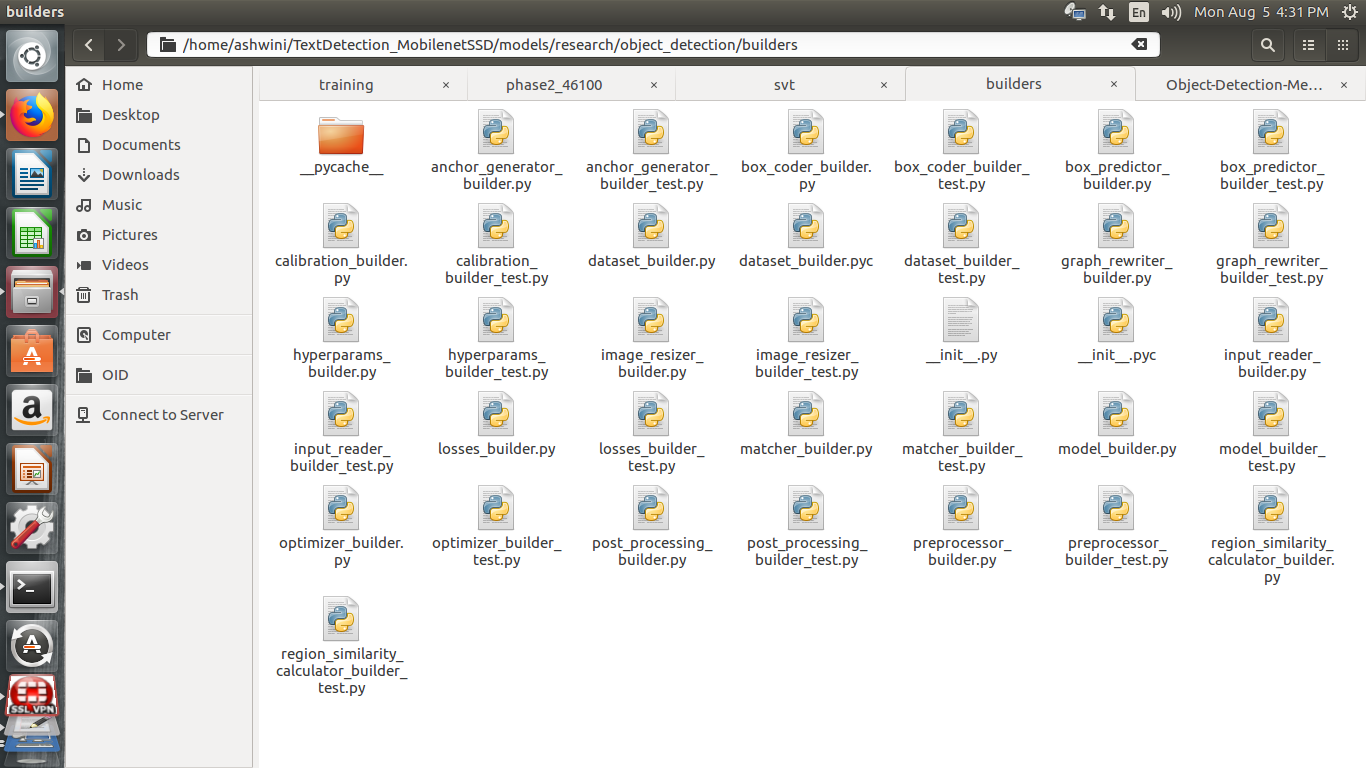
* The number of multiplications is standard convolution is calculated as:
* (5 x 5 x 3) kernel is being multiplied to get output 256 times i.e. (5 x 5 x 3 x 256) is moved (8 x 8) times on image
* Thus, 5 x 5 x 3 x 256 x 8 x 8 = **12,28,800 multiplication ops.**

**2) Depthwise convolution:**

* The number of multiplications is depthwise convolution is calculated as:
* (5 x 5 x 1) kernel is being multiplied to get output 3 times i.e. (5 x 5 x 1 x 3) is moved (8 x 8) times on image
* Thus, 5 x 5 x 1 x 3 x 8 x 8 = 4,800 multiplication ops.
* The number of multiplications is Pointwise conv. is calculated as:
* (1 x 1 x 3) kernel is being multiplied to get output 256 times i.e. (1 x 1 x 3 x 256) is moved (8 x 8) times on image
* Thus, 1 x 1 x 3 x 256 x 8 x 8 = 49,152 multiplication ops.
* Totalling makes it to **52,952 multiplication ops**, which is about 22 times less operations as compared to std. Convolutions.

**# Text detection using TensorFlow's Object detection API:**

* One can download or just git clone this repo. <https://github.com/tensorflow/models.git>.
* To train the model using transfer learning, we need to clone ssd\_mobilenet\_v1\_coco checkpoints from <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md>.
* In train.py, the configurations like model\_config, train\_config and input\_config is read/taken from ssd\_mobilenet\_v1\_coco.config file.
* model\_fn is being called and detection model is initialized i.e. meta architecture and feature extractor used here is SSD, and the model used is MobileNet.
* create\_input\_dict\_fn takes responsibility of taking images for training, generating the next iteration and pre-processing it before training.
* For actual training with this prepared model and input, trainer.py script is called to train the model.
* Folders such as utils, box\_coders, anchor\_generators, core, etc are used from train.py by calling each function implicitly from train script to build the model and the dataset.
* For example, the building parameters for the model are initialized from the scripts in builder folder as shown below:



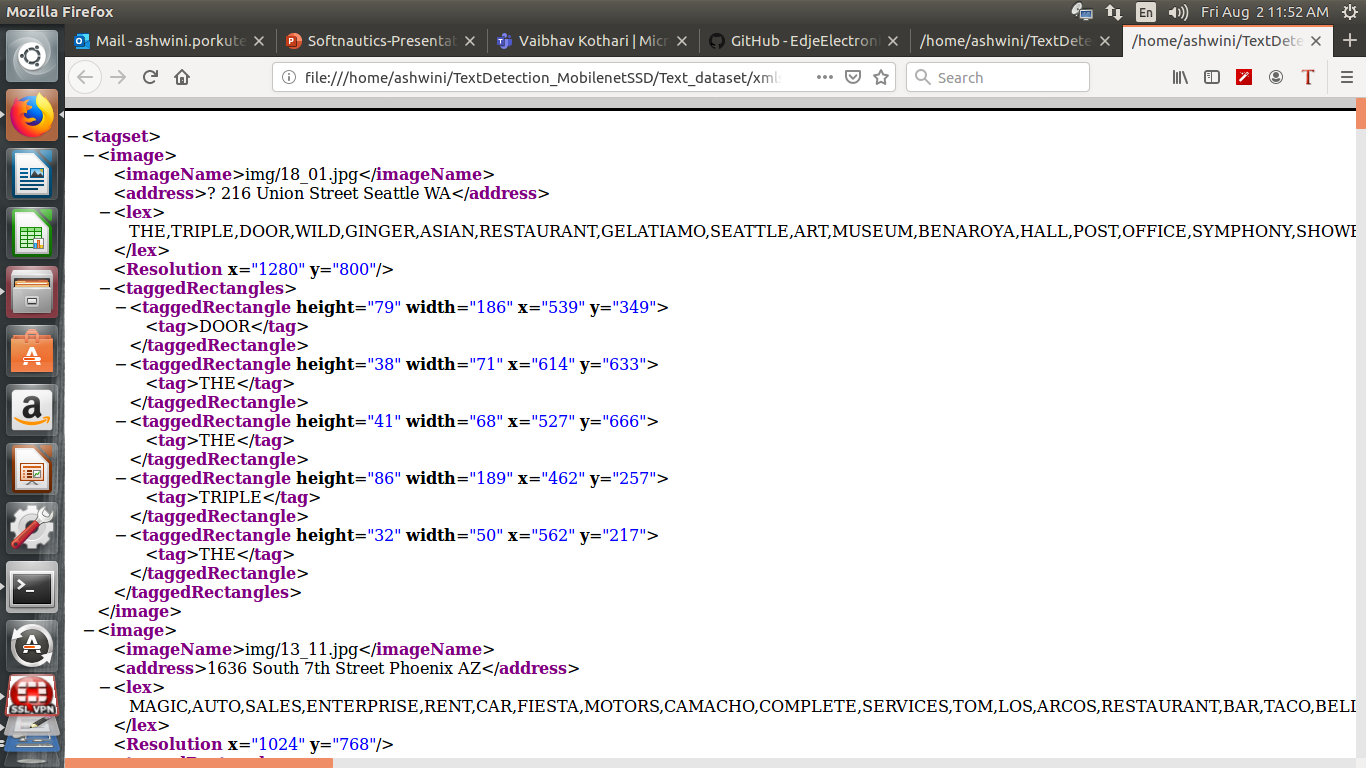
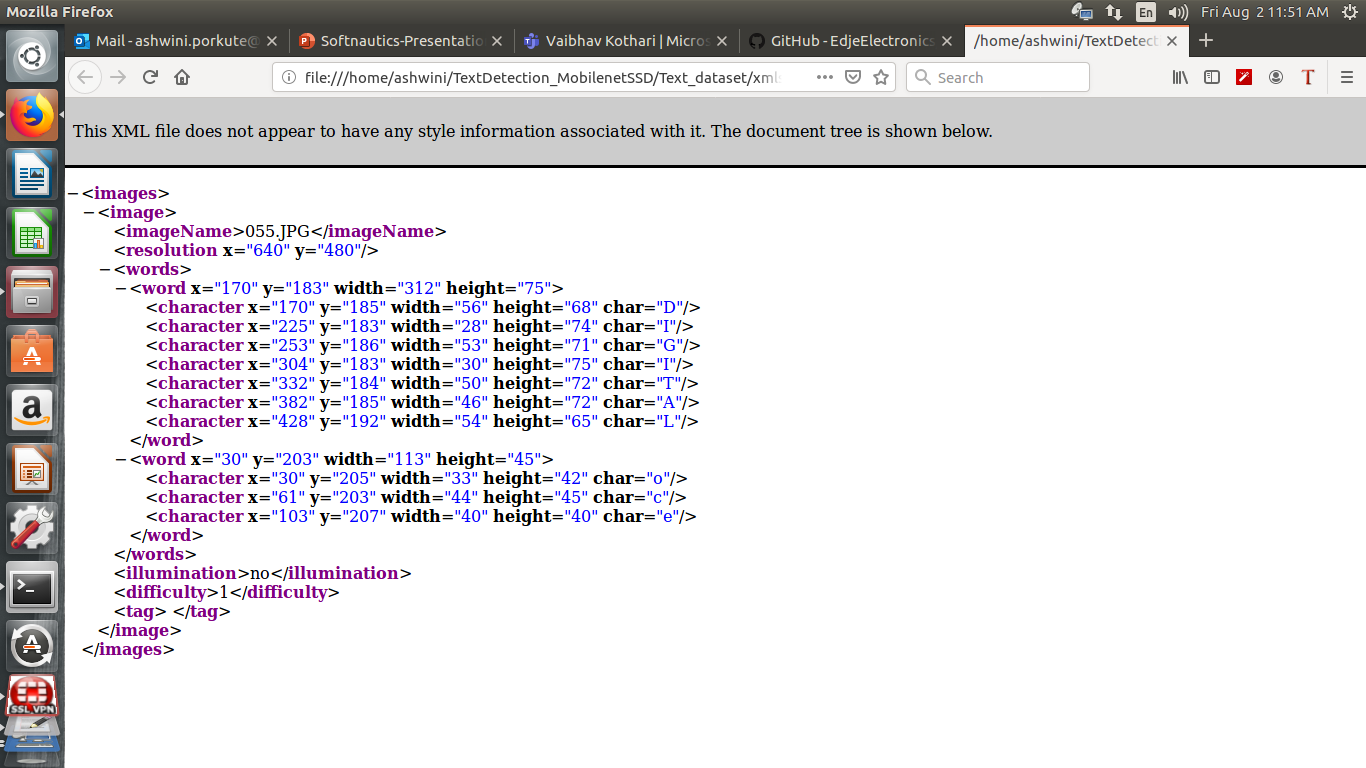
* To start training we need to make changes in config and label\_map file according the requirements.

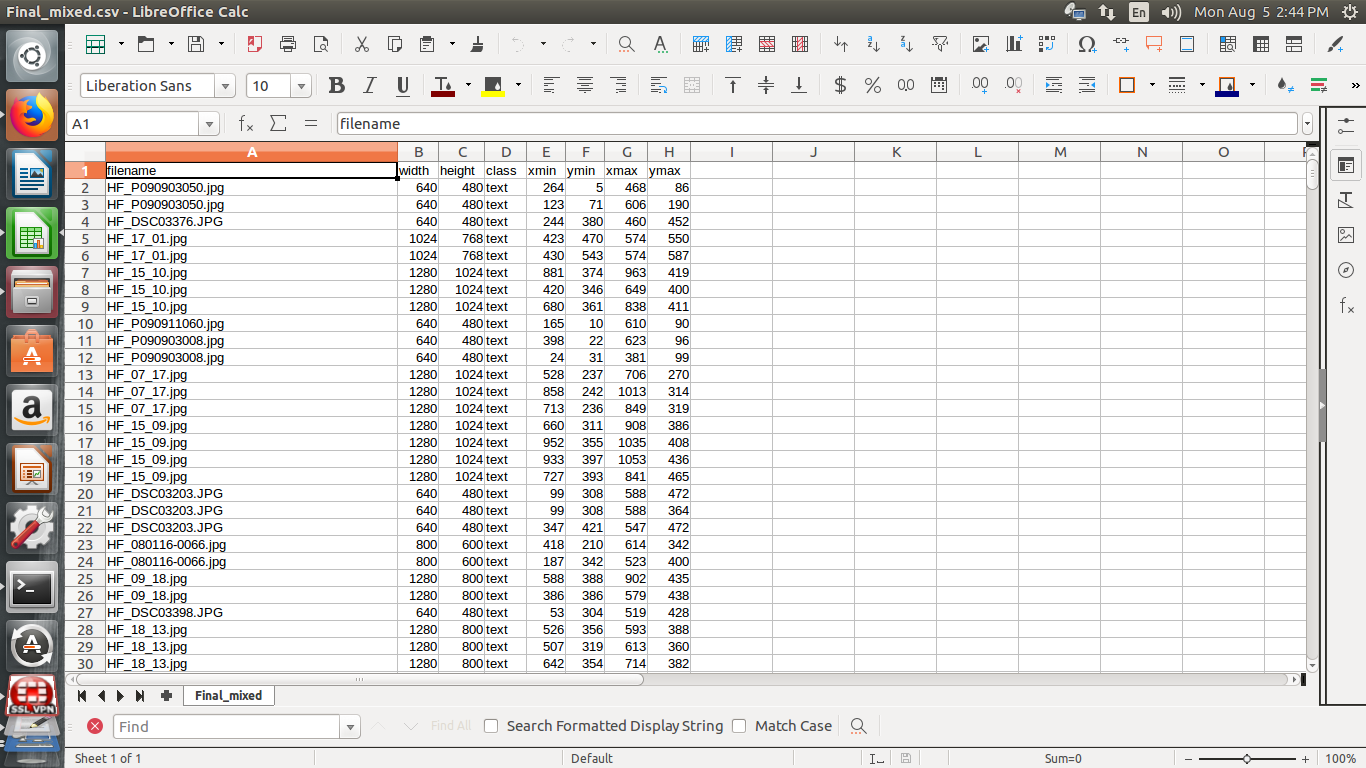
**# Dataset Preparation and Augmentation:**

* Now the relevant and time taking part comes into picture where we need to collect the dataset according to the requirement.
* One can found the list for available Text dataset by visiting <https://lionbridge.ai/datasets/15-best-ocr-handwriting-datasets/>.
* In Object detection API, rather than passing the images for the training we are converting the images and its annotations into TensorFlow's binary Tf\_record format.
* TF\_record is preferred as it takes less disk space and reads the data quickly.
* So, the main goal for dataset preparation is to convert dataset to tf\_record format for text detection in object detection API.
* For text detection purpose, chosen SVT (Street View Text) containing nearly 350 images and KAIST dataset containing about 350+ images with annotations in xml format (different xml format for SVT and KAIST).

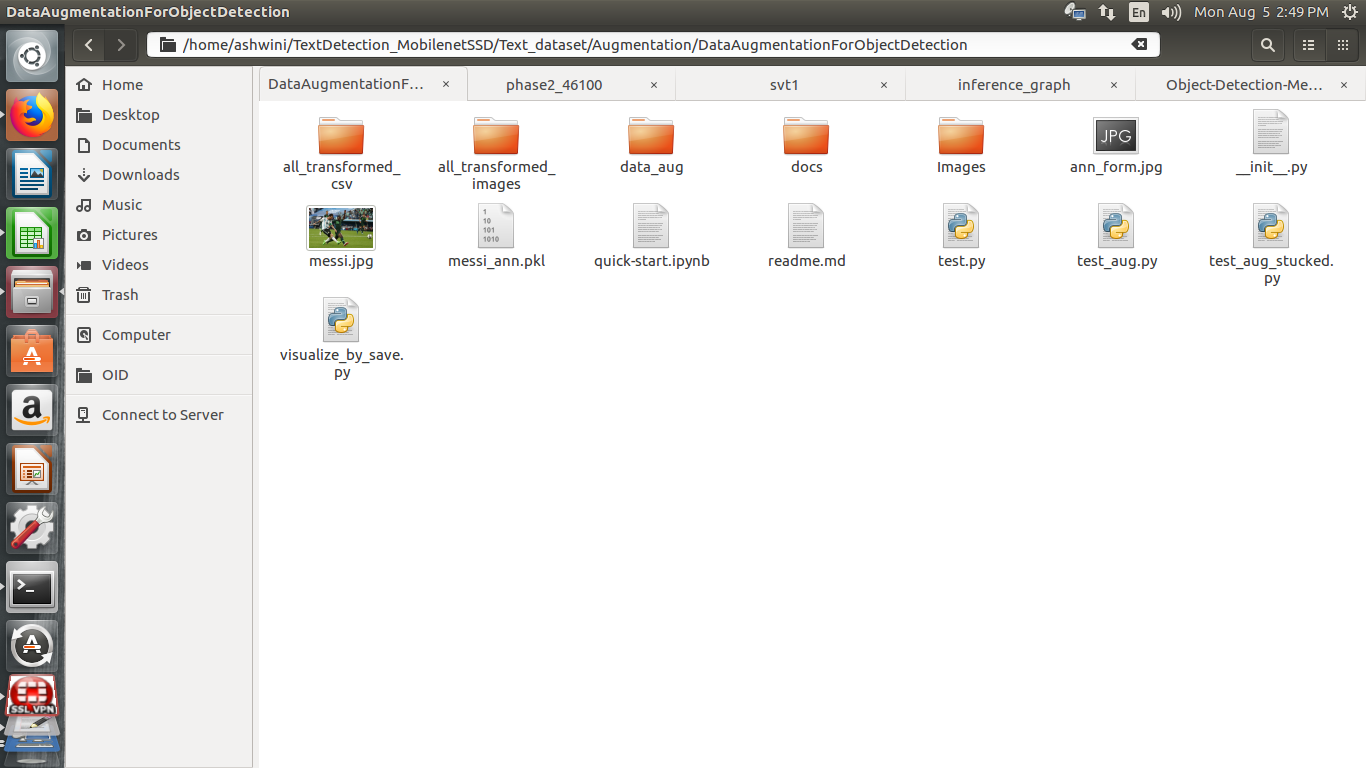
* Converted each xml format to csv using the script **xml\_to\_csv\_kaist.py** and **xml\_to\_csv\_svt.py** and later both csv is merged to get total dataset in single csv using **merging\_csv.py** script.





* Augmentation is used to increase the dataset and level up the quality of the dataset.
* Augmentation can be referred for object detection on <https://github.com/Paperspace/DataAugmentationForObjectDetection>.
* Example for single operation Horizontal Flip:





* Keep images in a folder and need to run below command to run script which uses the merged csv and generate augmented images, new csv containing the new bounding box co-ordinates after augmentation.
* Need to change the paths to images folder and merged csv folder in the test\_aug.py script.

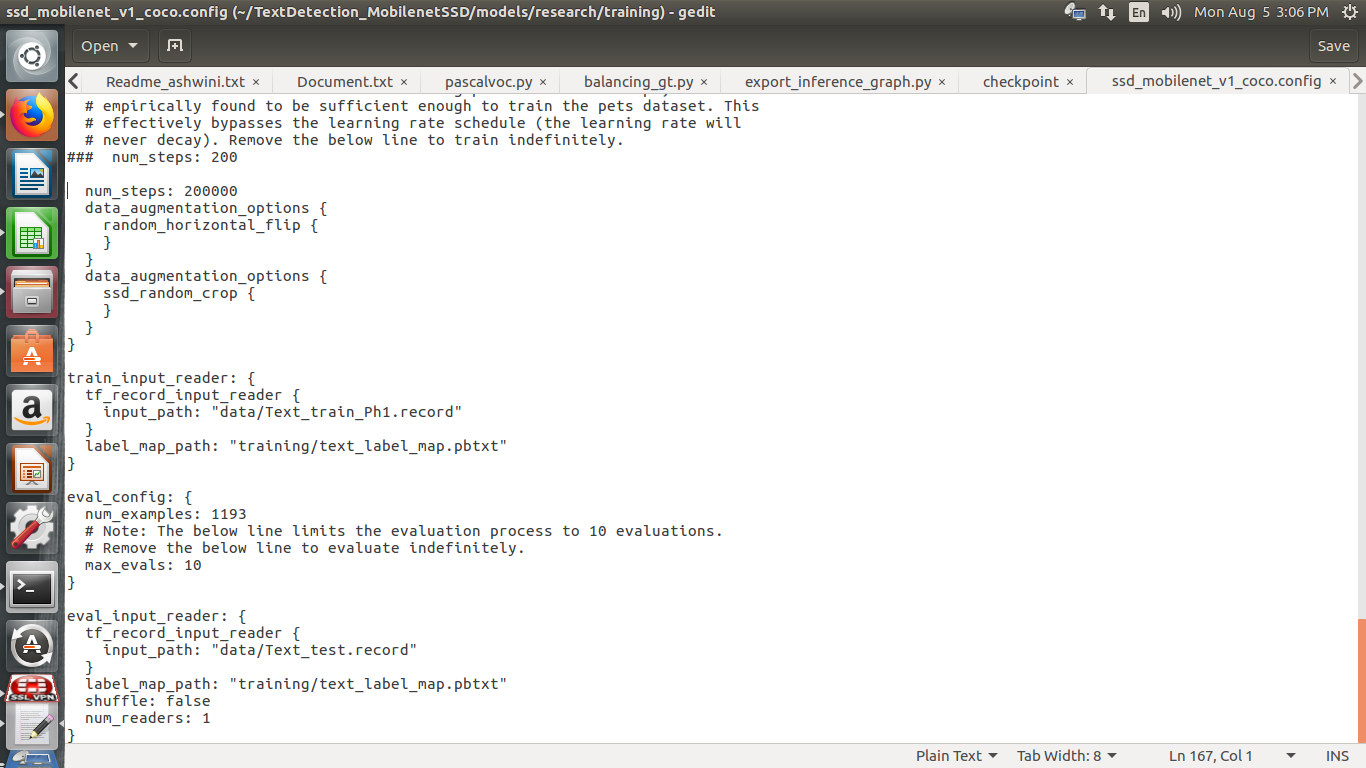
**$ python test\_aug.py**

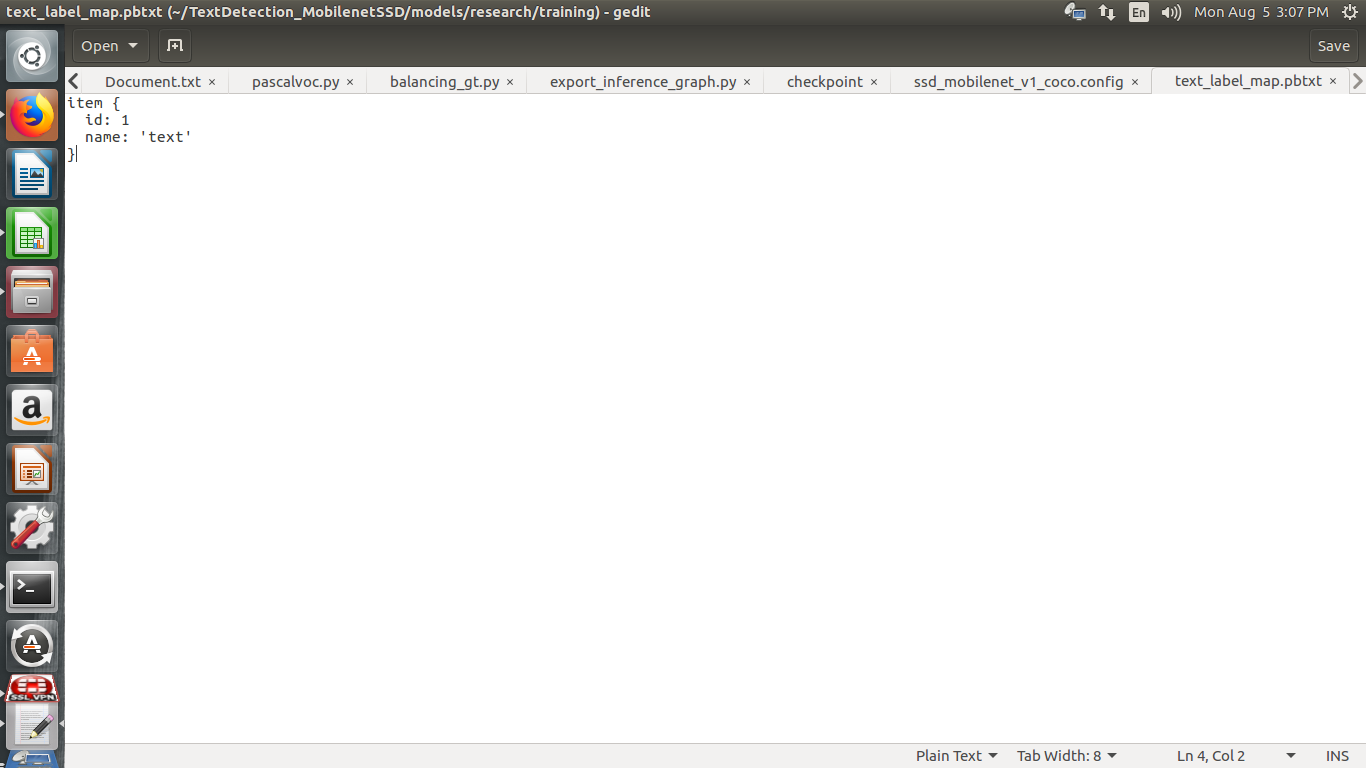
* Run for different operations. Here, for this task we used horizontal flip, rotate with different angles, scale, shear, translate, HSV. (For each operation new generated csv will be created so again need to merge)
* After augmentation we need to split the dataset into train and test dataset. (5943 total images. Split to 4750 images for training and 1193 images for test set).
* Converted the csv to tf\_record format by running the below command:

**$ python generate\_tfrecord.py**

**# Training the model using transfer learning:**

* To train the model using transfer learning one need to modify the ssd\_mobilenet\_coco.config where we change the terms like - num\_classes, fine\_tune\_checkpoint(for transfer learning), input\_path(path to tf\_records of the dataset), label\_map\_path(for labels) according to the structure of your repo.
* For transfer learning we need to pass the pretrained model (model.ckpt) file taken from <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md> to “fine\_tune\_checkpoint” parameter.





* And, we need to create the label\_map.txt file according to the labels that we want to detect.
* Now we are all set to run command to trigger the training process. Command used to train is:

**$ python train.py --logtostderr --train\_dir=logs/ --pipeline\_config\_path=training/ssd\_mobilenet\_v1\_coco.config**

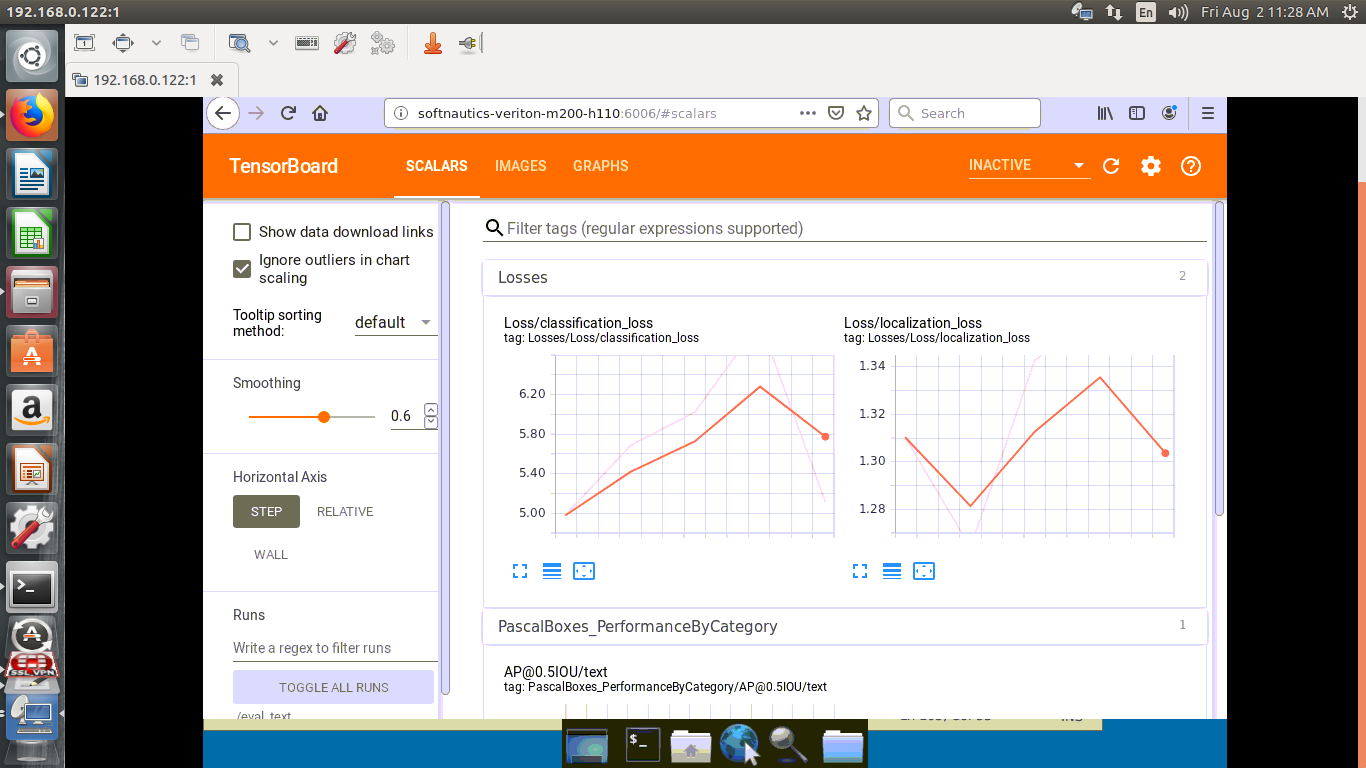
- We can evaluate our model is trained by running eval.py using command:

**$ python object\_detection/eval.py --logtostderr --pipeline\_config\_path=training/ssd\_mobilenet\_v1\_pets.config --checkpoint\_dir=training/ --eval\_dir=./eval\_text**

**# Visualization on Tensorboard:**

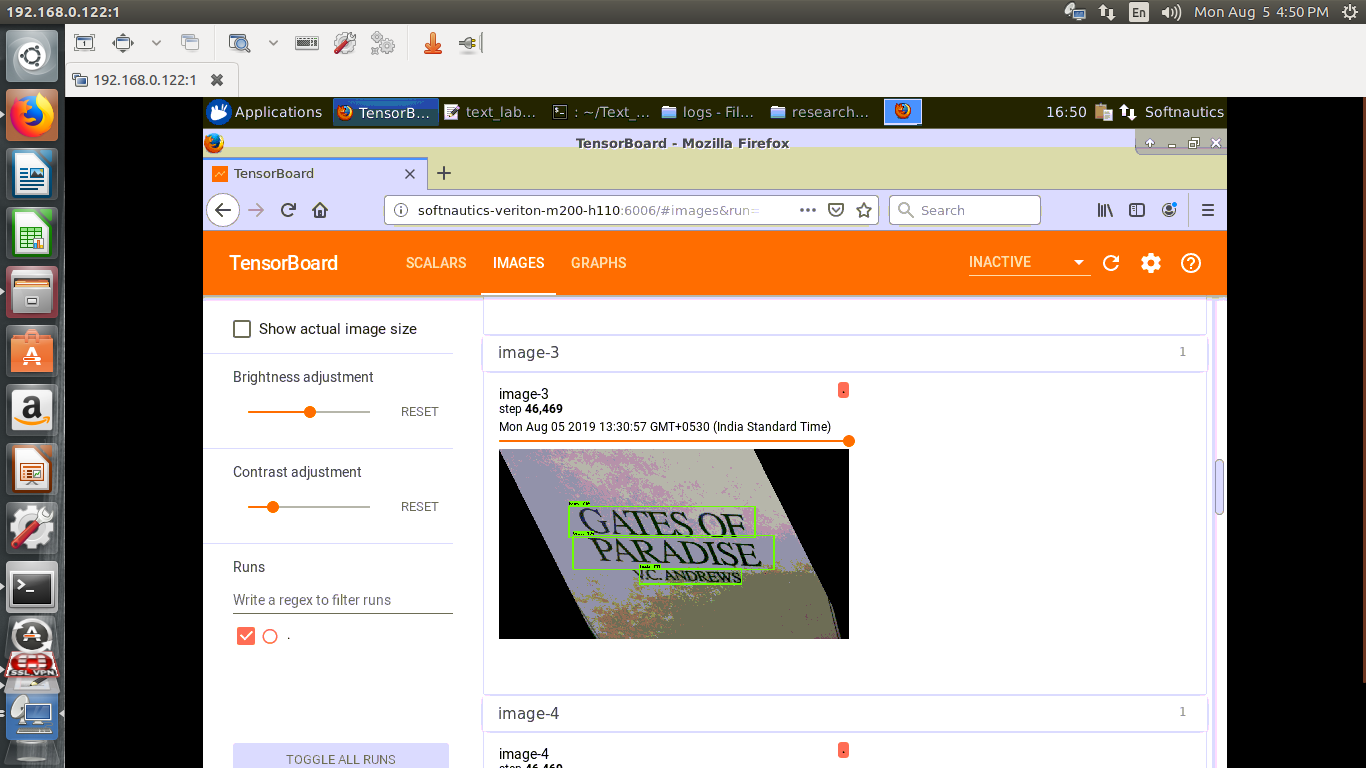
* We can visualize the training parameters like classification and localization loss, with the graph using the command below:

**$ tensorboard --logdir=./logs/**



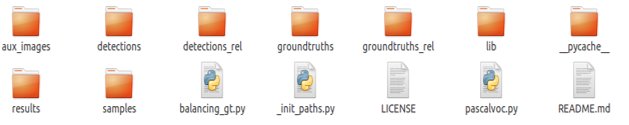
* We can visualize the images are properly trained or not with the detection graph with the command:

**$ tensorboard --logdir=./eval\_text/**



**# Checking the performance metrics of the model:**

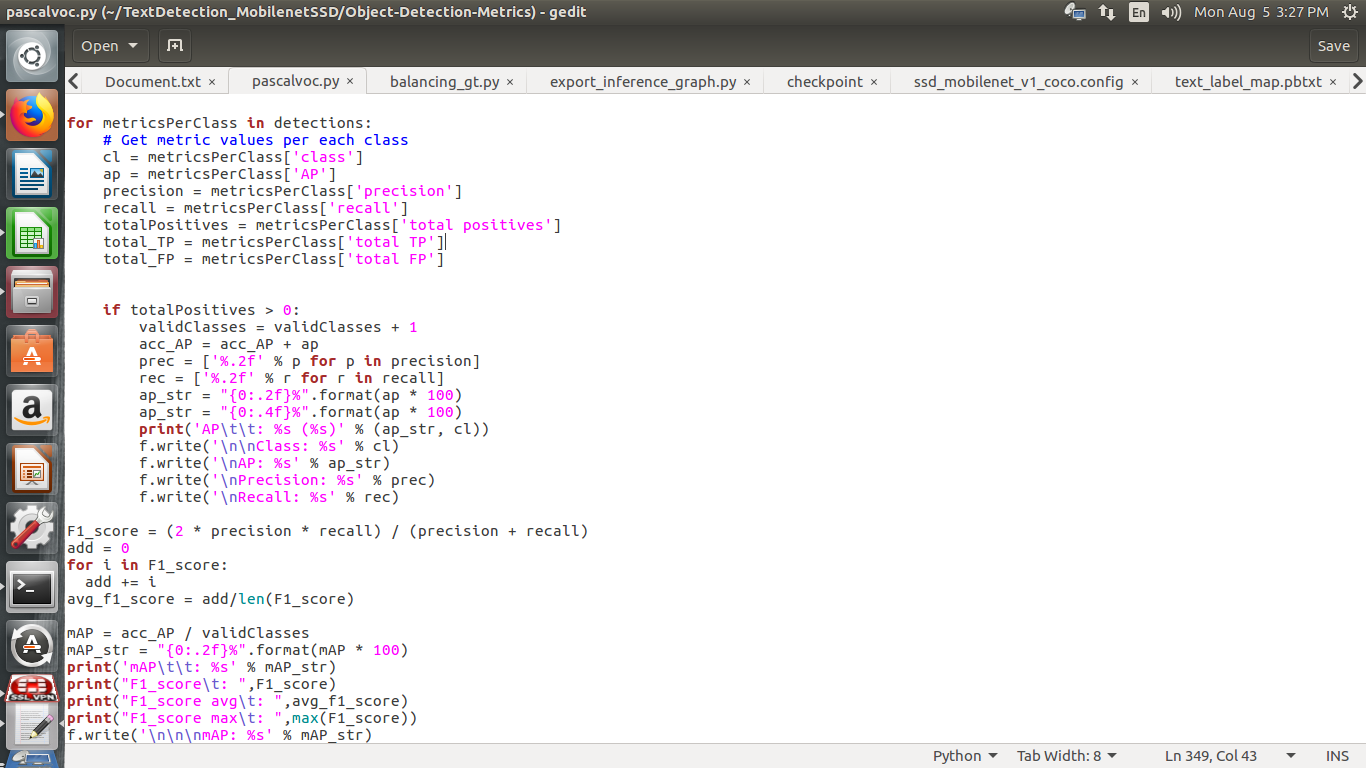
* To check the performance metrics, you can download or git clone <https://github.com/rafaelpadilla/Object-Detection-Metrics>.
* The directory structure for Object-Detection-metrics is shown below:



* balancing\_gt.py script is to make the detection and ground truth files count same in respective directories.
* To check the mAP, one can run the pascalvoc.py to get the output as shown:

**$ python pascalvoc.py**

* Some changes have been made in the pascalvoc.py script to get the f1-score metrics along with the other metrics as shown below:



* Result of above command can be found below:

