Edge Computing Problem Statement –Solution

By

Team :-ThinkNow

Name	DT No.	Email Ids	Role
Himanshu Sharma	DT20184814912	ansh.sharma.3181998@gmail.com	Team Leader
Shiv Shankar Prasad	DT20195110831	shivshankarkumar.pusa@gmail.com	Team Member

Contents

Links:-

1. YouTube Video Link:- https://youtu.be/wwdRqwhyOes

2. GitHub Links:-
    - Kirana web app:- https://github.com/sharmaansh/kiranawebapp.git
    - Darknet Repo:- https://github.com/sharmaansh/darknet.git
    - Bill Desk Website:- https://github.com/sharmaansh/serverweb.git

1) Problem Statement:-

Kirana is a retail store looking for more digitalized way of expanding their business.They want to use a more systematic way of checkout system and system that allows automatic detection of product using camera.The detection of product must be wrt the size of the product, type of product and automatically take the cost of product to make a bill of materials at checkout.This has to be done in real-time without sending the data to cloud for processing as some of these stores can be in remote areas with intermittent connectivity and Kirana do not want customers to wait due to latency issues for connectivity.

2) Solution Brief Description:-

As edge server is not compute intensive we will be hosting our web app on edge server with Yolov3_tiny_model for object detection as it is light weight and also less compute intensive.We will be using a temporary database to store information regarding of bill and cart item details and the tabels get truncated after bill is recived to the customer and bill record database in cloud.We will be using cloud server for trainning and storing the bill in database and web site to check the bills stored in bill record database.

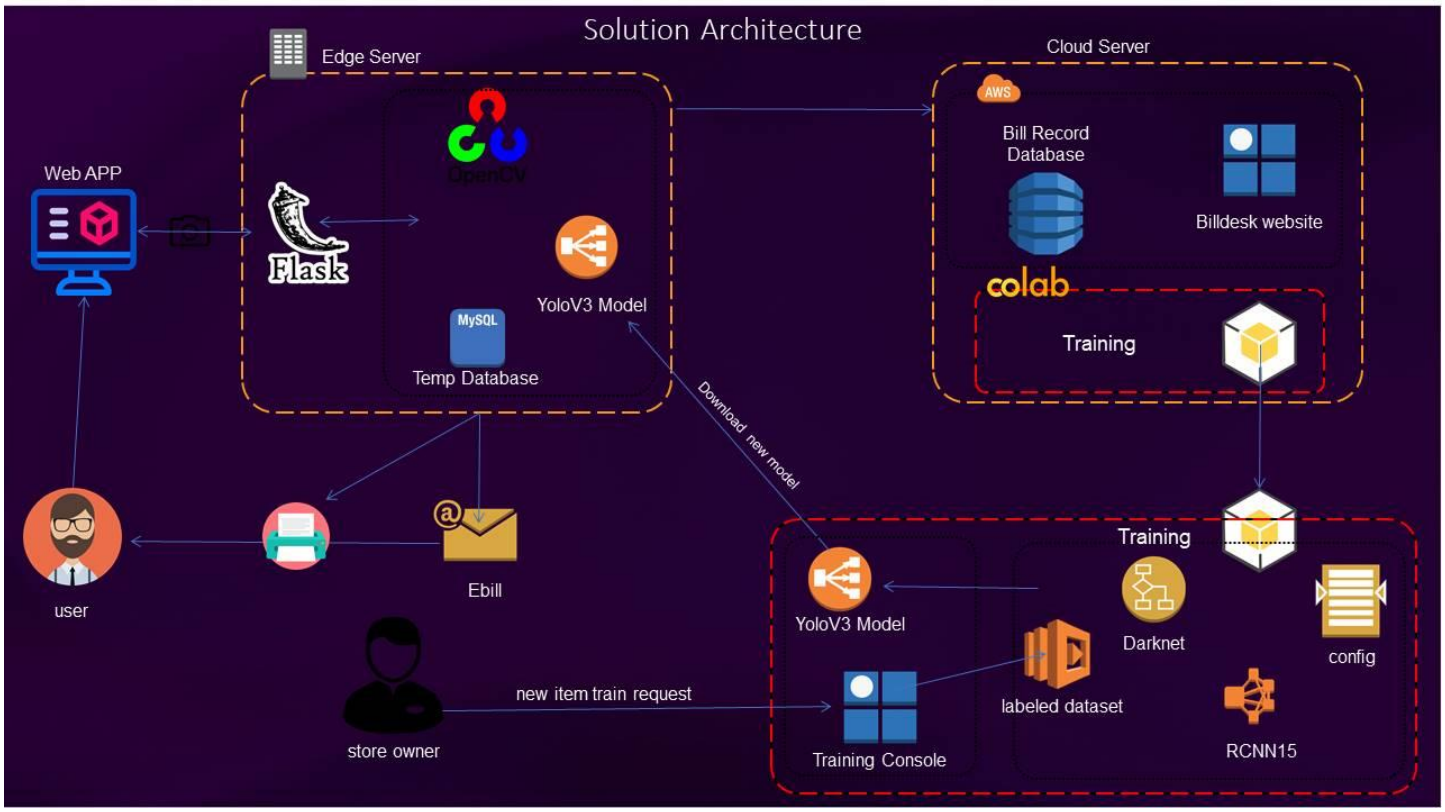
The trainning is done in Google colab as it provide 12hrs of jupyter notebook like enviornment to run your scripts with GPU and 12gb RAM support,our model take less time to train and have a good accuracy in detecting objects

We can alternately use different services to train our model also

3) **Requirments,Services,Tools Used:-**

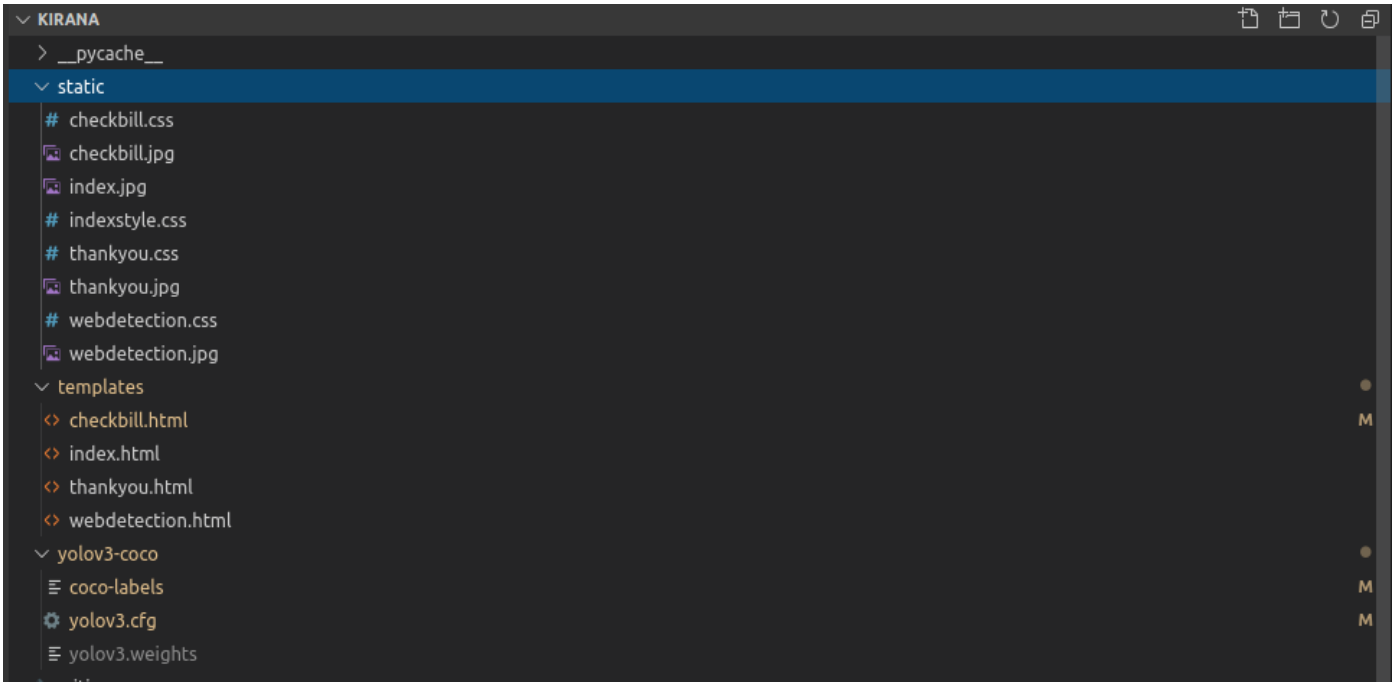
- Edge Server
  - Webapp
    - Python3
    - Opencv
    - Yolov3_tiny_model
    - Camera(hardware)
- Cloud server
  - Training
    - Google colab
    - Darknet
    - Google Drive
    - CUDA
    - Github
  - Billdedesk/trainning console
    - AWS
    - PHP

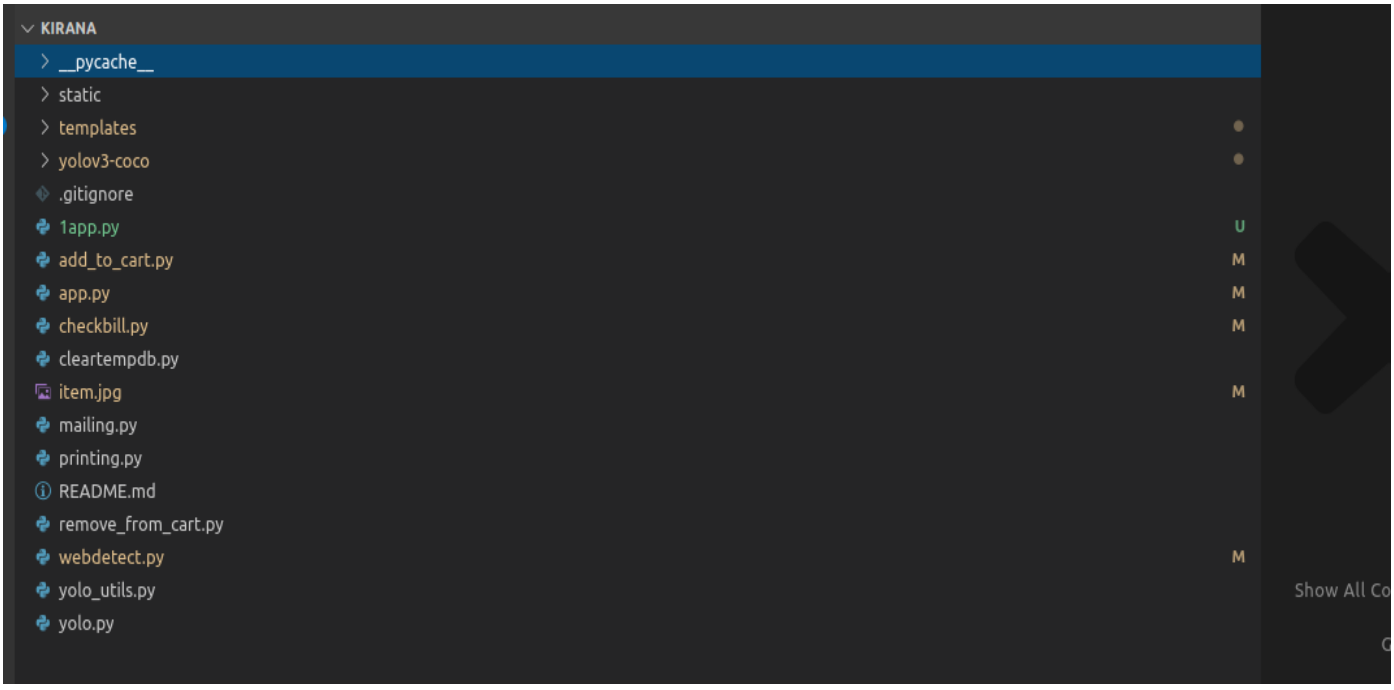
4) Architecture of solution:-



5) Edge Server:-

- **Web App:-**
    - **i. Directory Structure:-**



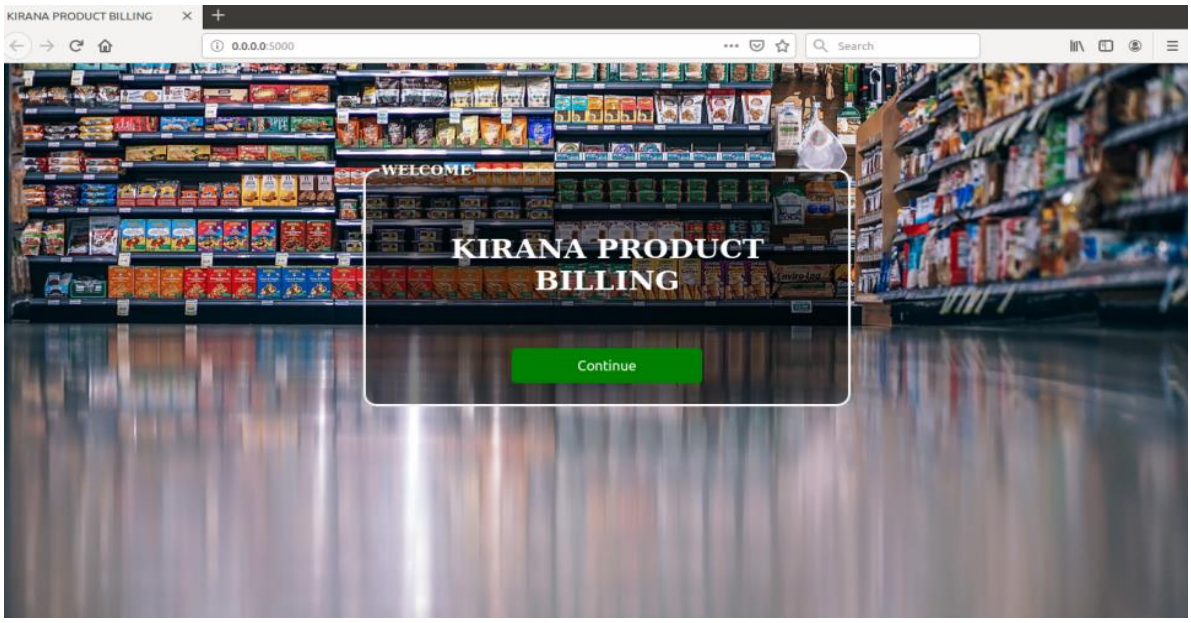


ii. **Running the web application:-**

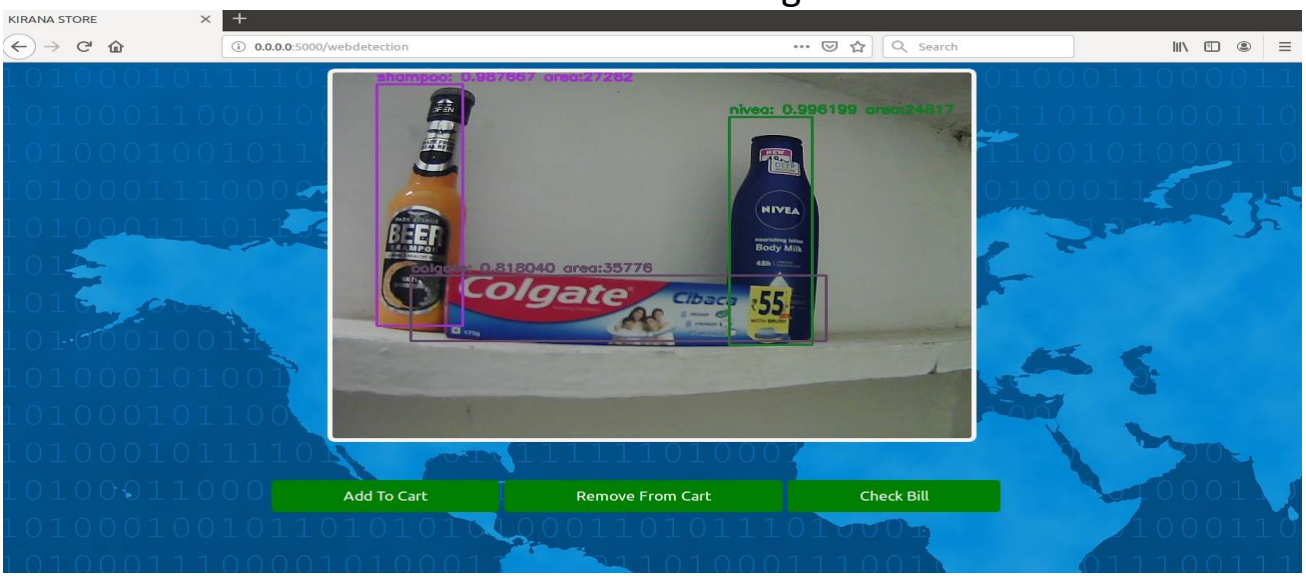
1. Clone the Kirana web App Repository.
2. Install all the dependencies mentioned in requirement file.
3. Run the **app.py** script.
4. Open browser go to the server IP shown by running the python script.

iii. **Description:-**

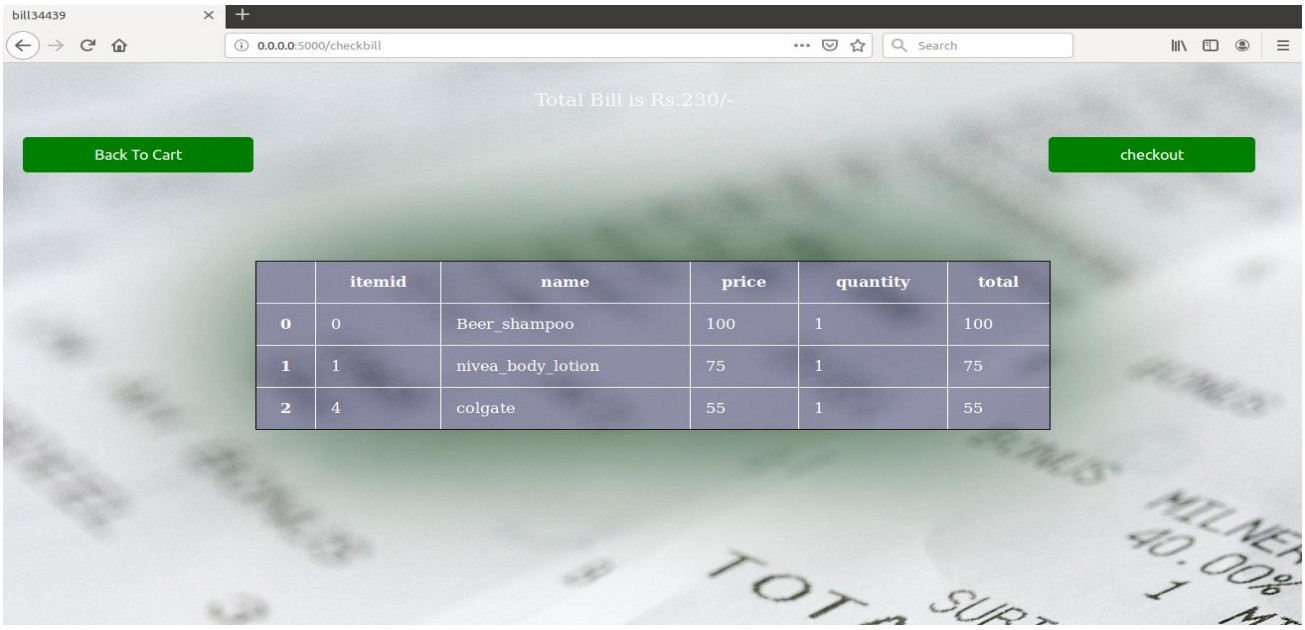
1. The customer at first see the home page of the web app when going for checkout .



2. On continue he will be directed to main page where product detection will be performed.
3. Customer can add item to cart and can also remove them and can check the bill using check bill button**.**

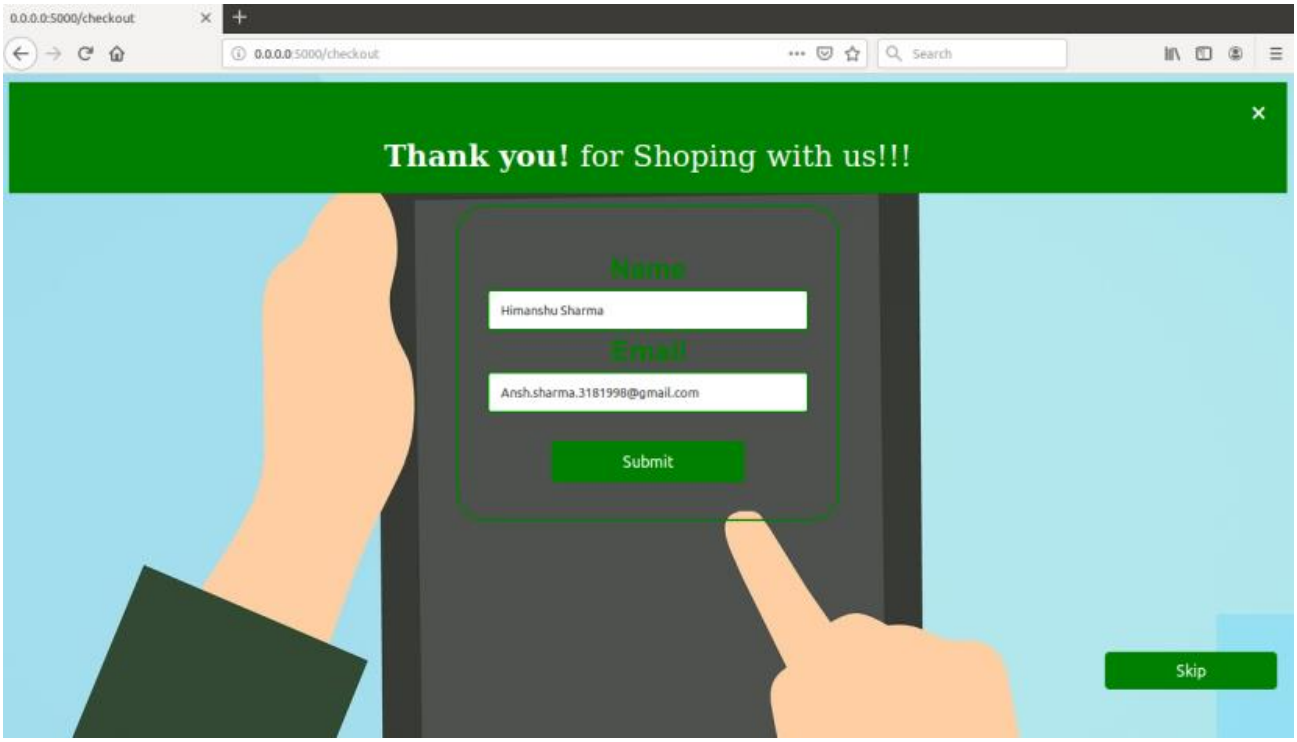


4. After checking bill he can go back and add/remove more items or can proceed to checkout.



	itemid	name	price	quantity	total
0	0	Beer_shampoo	100	1	100
1	1	nivea_body_lotion	75	1	75
2	4	colgate	55	1	55

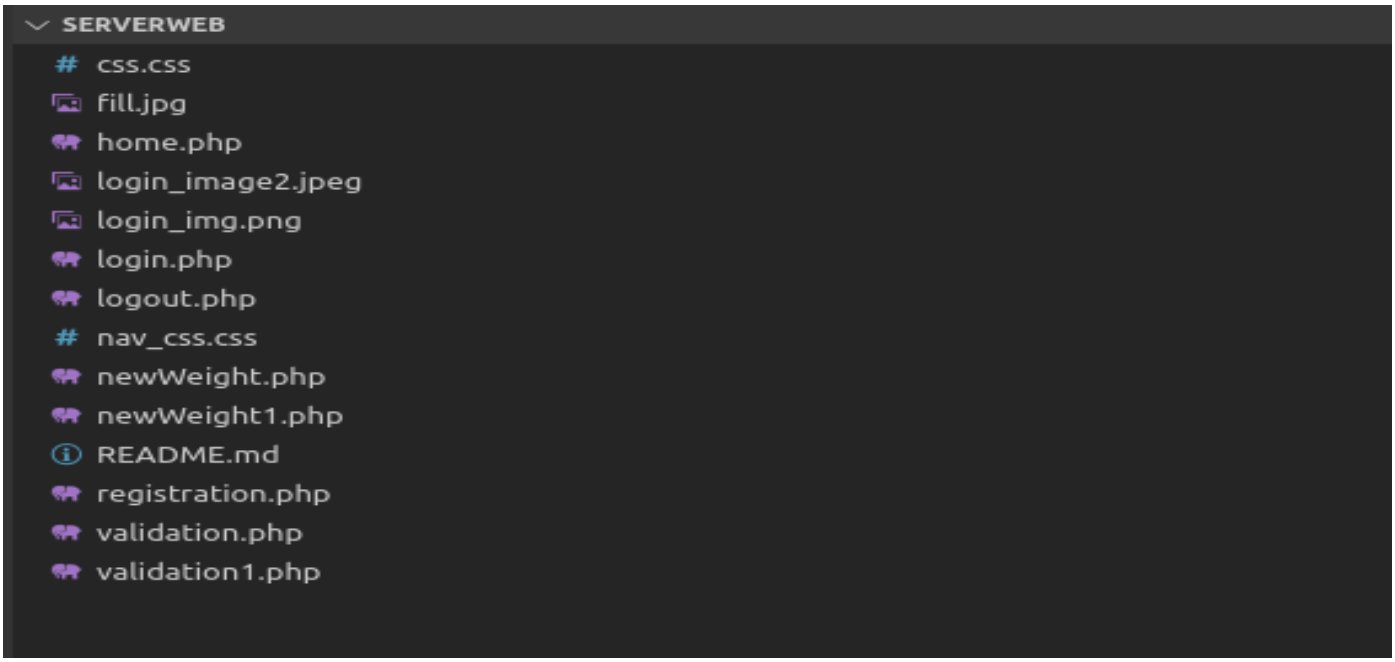
5. here customer can choose for E-Bill or skip and get a printed bill and web app is redirected back to home page.



6) Cloud Server:-

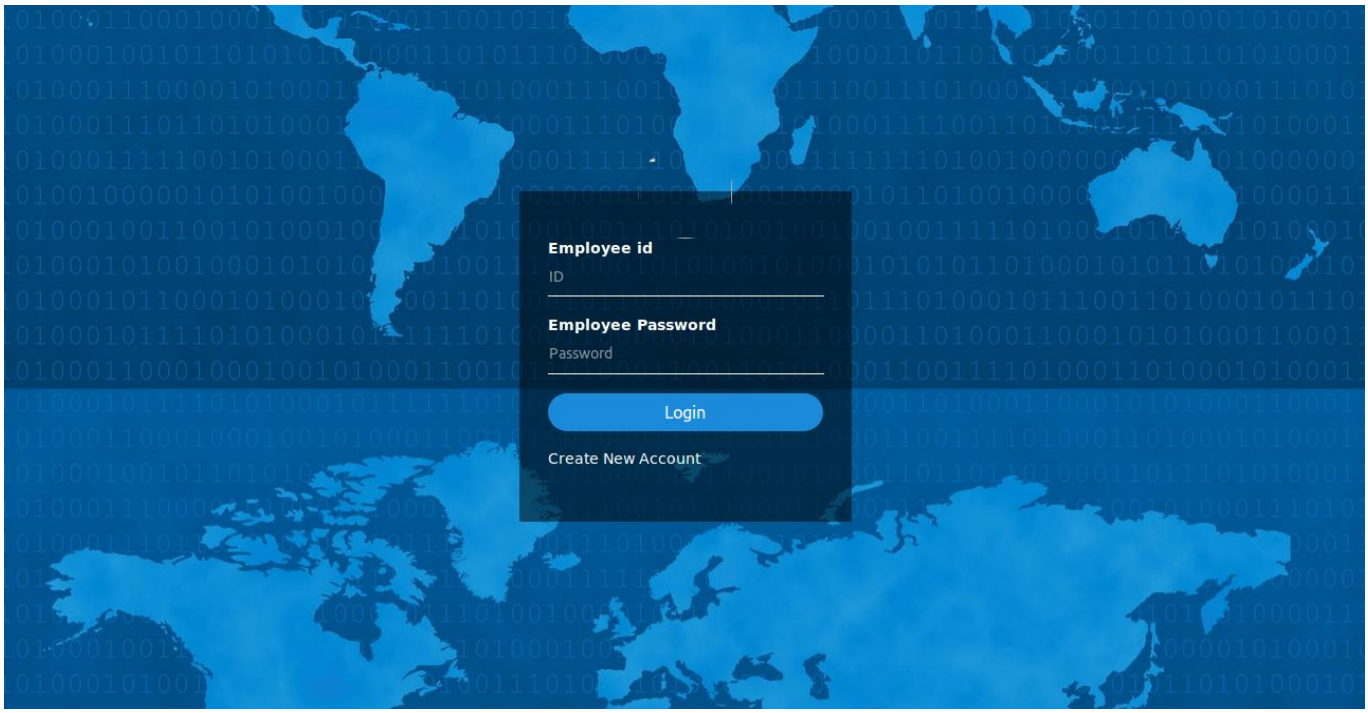
- **Bill Desk/Training console (Serverweb):-**

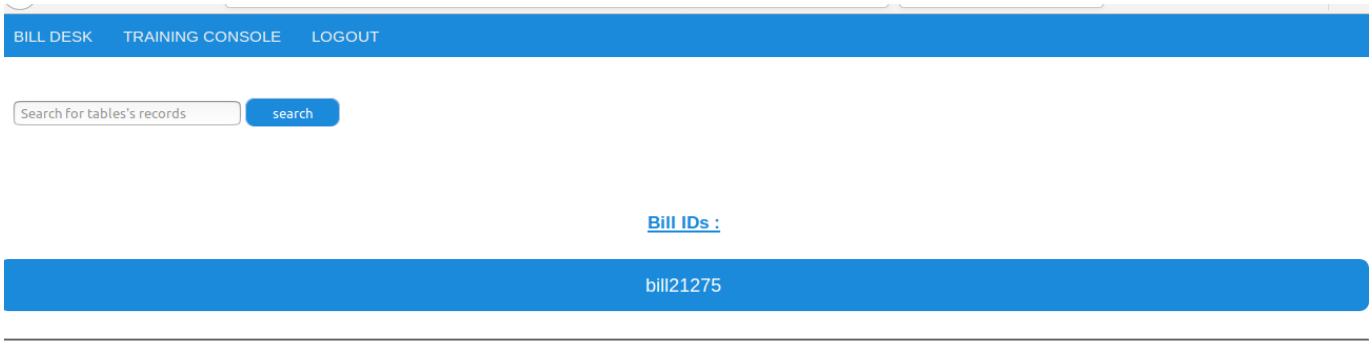
i. Directory Structure:-



ii. Discription(Bill Desk):-

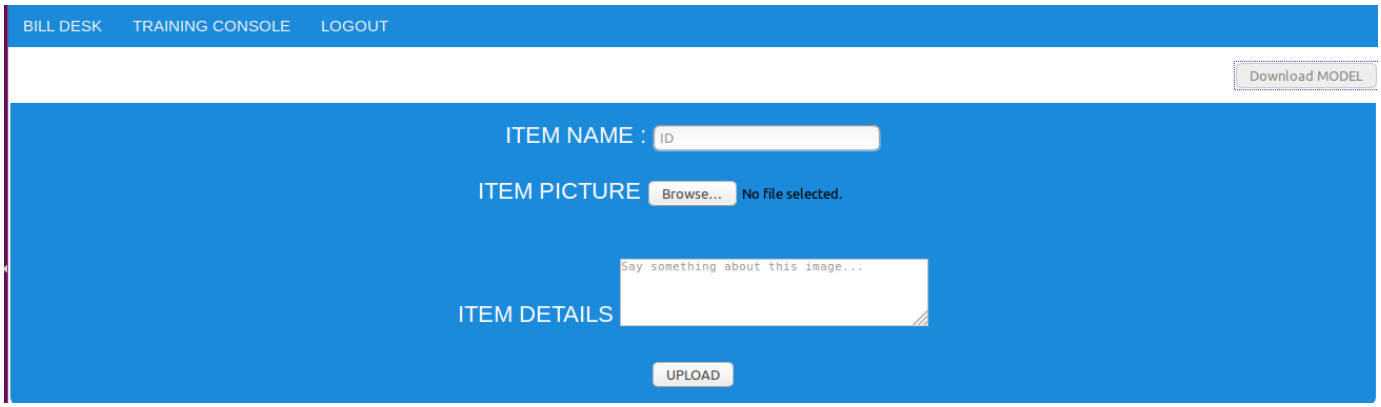
1. The bill generated when a customer checkout is sent to the bill record database and store employee can check the bill records using the bill desk website.





iii. Discription(Training console):-

1. When a new item for which our model is not trained and store owner wants to train the model for new item can send a new item train request using training console.



2. After training for new item when the model is ready can be downloaded using download model button

- **Training: -**

i. Requirments,Services,Tools Used:-

1. LabelImg.
2. Darknet
3. Yolov3_Tiny_R-CNN (Convolutional Neural Network) With 15 (Neurons)
4. Google Colab
5. CUDA

ii. **LabelImg** is a graphical image annotation tool and label object bounding boxes in images.
It is completely written in python and we have used it for labeling our images. While labeling we have to draw a boundary box around the object and save it. After saving, it generates a text file with the same name as of the file which contains the detail about the class and the x-coordinate, y-coordinate, height of the bounding box and the width of the bounding box. Similarly, labeled for around 600 images for 5 classes.

iii. **CUDA** (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by Nvidia. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.
We used CUDA for the direct use of GPU for the training of our model at a faster pace.

iv. **Darknet** is a framework to train neural networks, it is open source and written in C/CUDA and serves as the basis for YOLO. Darknet is used as the framework for training **YOLO**, meaning it sets the architecture of the network. We used darknet to train our RCNN-15(Region bound Convolutional Neural Network) on our created dataset.

v. **YOLOv3** is a powerful neural net that tell you what is in your image giving the bounding box around the detected objects.YOLO, “You Look Only Once”, is a neural network capable of detecting what is in an image and where stuff is, in one pass. It gives the bounding boxes around the detected objects, and it can detect multiple objects at a time.

1. There are many different versions of YOLO. We used YOLO V3 tiny because of the following characteristics.
   a. It is very fast as compared to any other YOLO model.
   b. It is more precise as compared to any other model.

c. It is very light weighted as compared to the any of the other YOLO model.

vi. **R-CNN 15**(Region bound Convolutional Neural Network) - It is a convolutional neural network having 15 layers. The purpose of R-CNNs (Region Based Convolution Neural Network) is to solve the problem of object detection. Given a certain image, we want to be able to draw bounding boxes over all of the objects.
It consist of 15 layers.**i**t consists of three modules.

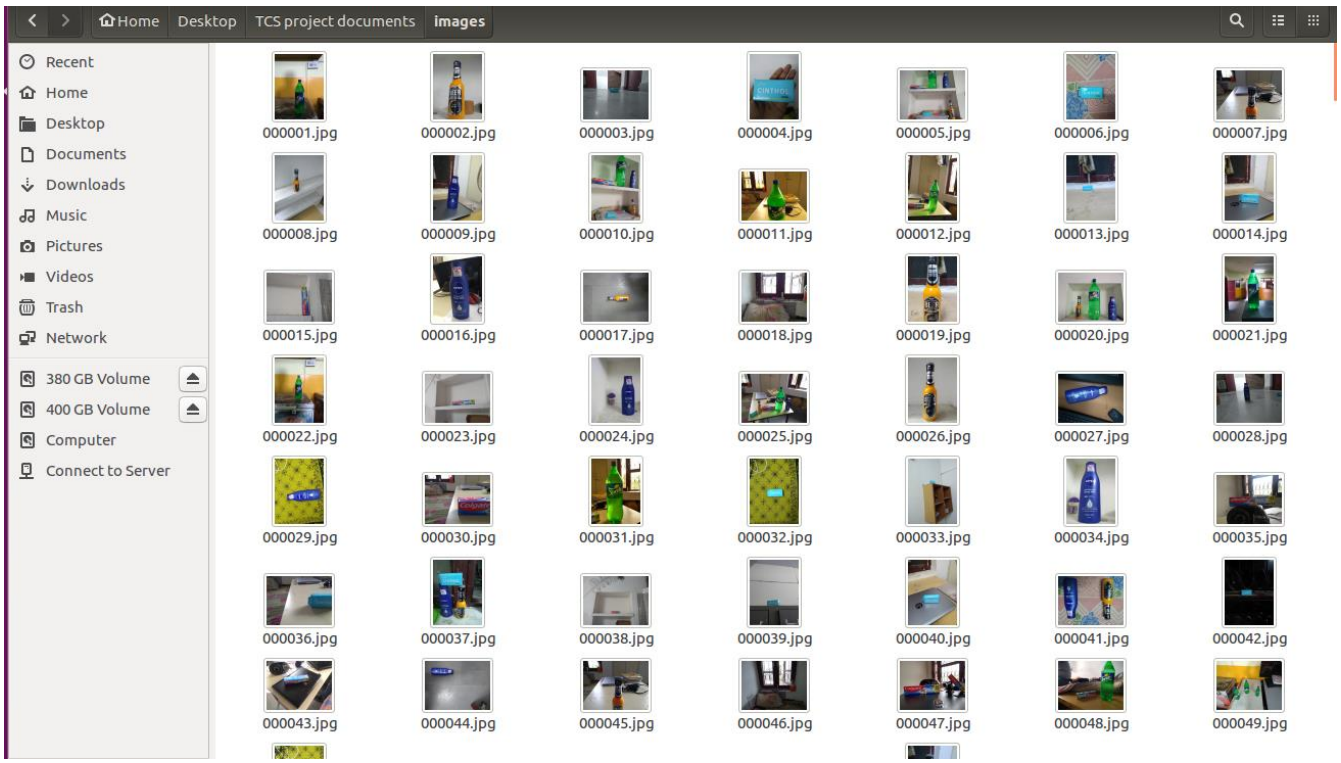
1. The first generates category-independent region proposals. These proposals define the set of candidate detection available to detector.
2. The second module is a large convolutional neural network that extracts a fixed-length feature vector from each region.
3. The third module is a set of class- specific linear SVMs.

vii. ***Training Steps*** *:-*

1. Gather Image
2. Resize the Image
3. Label Image
4. Generate Training File
5. Generate the Anchors
6. Change the Configuration File
7. Download the Convolutional File(Convolutional Neural Network)
8. Providing paths in obj.data file
9. Clone Everything to Google colab
10. Mount the Drive for Backup
11. Install the Required Dependencies
12. Start the Training
13. Finish Training(output)

Training Processs:

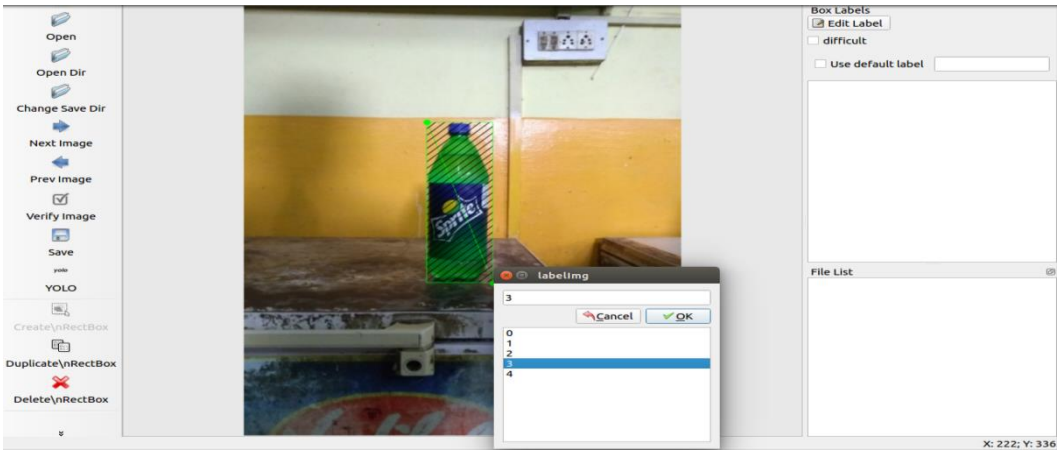
1. Gathering Image dataset simply consist of clicking pictures of our object in different angles and in different light conditions.



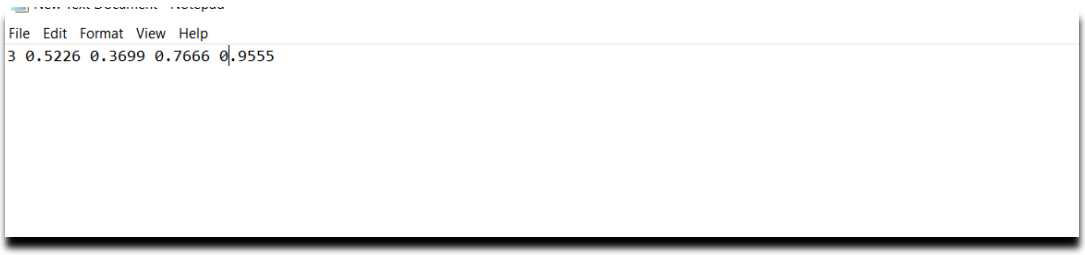
2. The picture clicked were of very high quality so we had to resize it as per our requirement that was 416x416. We used a python script to resize it.

```
import numpy as np
import cv2
import os

dir_path = os.getcwd()

for filename in os.listdir(dir_path):
    # If the images are not .JPG images, change the line below to match the image type.
    if filename.endswith(".JPG"):
        image = cv2.imread(filename)
        resized = cv2.resize(image,None,fx=0.125, fy=0.125, interpolation=cv2.INTER_AREA)
        cv2.imwrite(filename,resized)
```

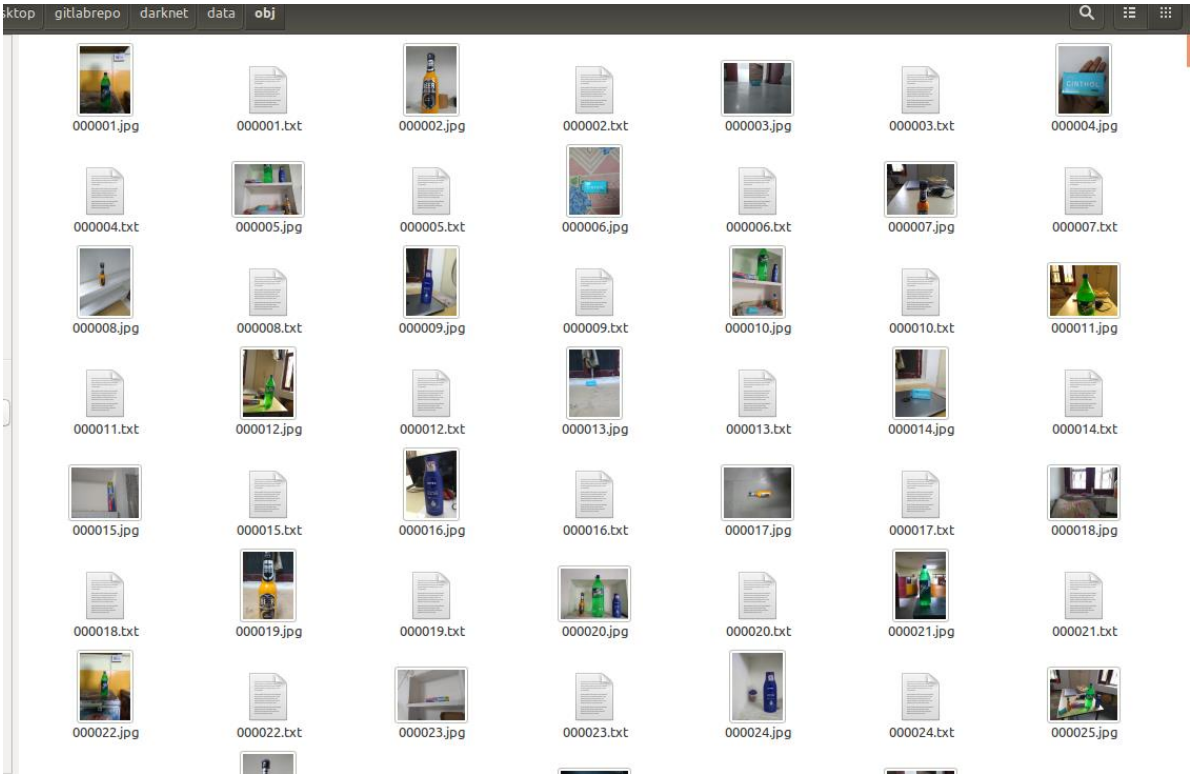
3. After that we have to label our images using the LabelImg , First we just drawn the boundary box around the object then provided the class and the saved it.



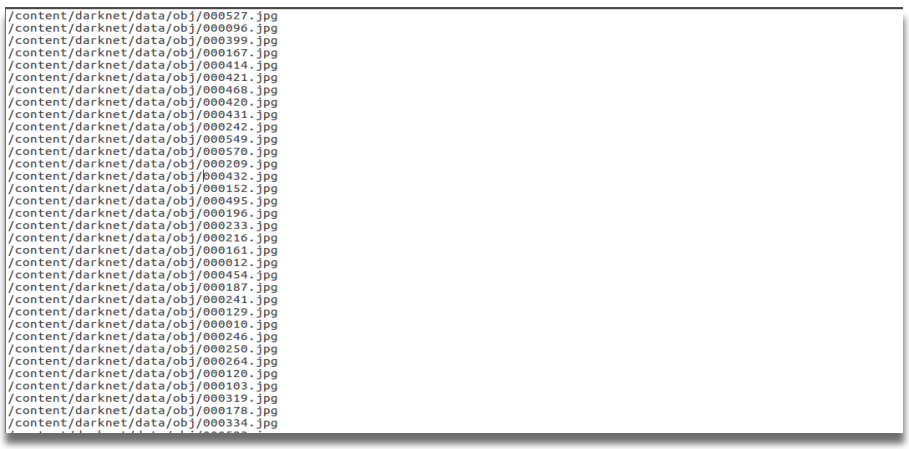
After saving the file, a text file was created with the class and the x-coordinate, y-coordinate, the width of the object and the height of the object.



Similarly we labelled for around 600 images.



4. After that we generated the training file and the test file having the path of the images of the dataset. In training file we provided the path to the 90% of our dataset and in test file we provided the path to the left 10% of the image dataset to test and validate the model.



5. After generating the training file and the test file we generated the anchors for the training of the model.

```
./darknet detector calc_anchors data/obj.data -num_of_clusters 6 -width 416 -height 416
```

```
num_of_clusters = 6, width = 416, height = 416
read labels from 56 images
loaded          image: 56       box: 88
all loaded.

calculating k-means++ ...

iterations = 5


avg IoU = 66.75 %

Saving anchors to the file: anchors.txt
anchors =  68, 29,  41,147, 142, 58,  87,256, 243,103, 142,380
```

6. Configuration file details: -

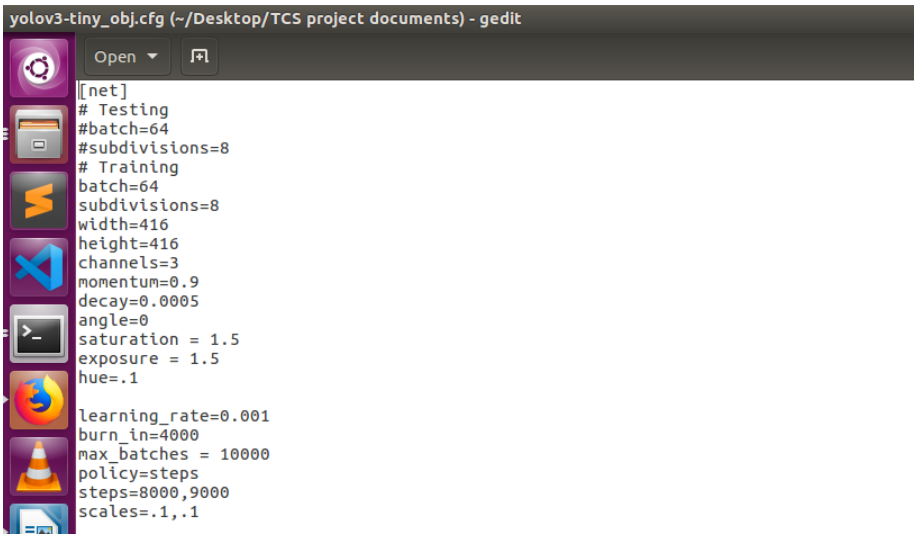
   Batch in set to 64 : It is the maximum no. of images which will be taken at once for training.

   Width & Height : When the training begins , images will be resized according to this.

   Learning rate : It is the parameter to control the adjustment of weight of our net work with respect to loss.

   Burn_in : It is the minimum no. iteration in which first best weights will be saved.

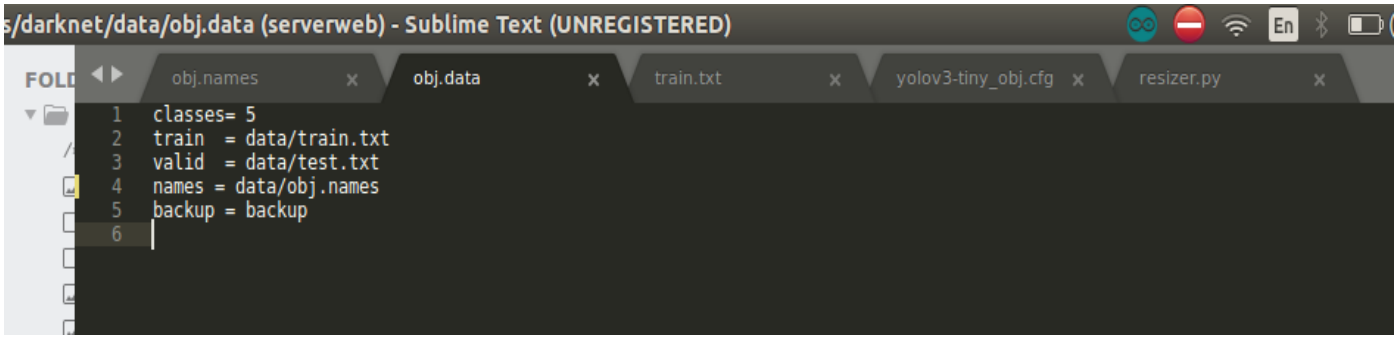
   Max_batches : It is the maximum no. of iteration for which our model will be trained.



7. Anchors : In anchor parameter we provide the anchor box values which are the sizes of objects on the image that is resized to the network size.
   Num is set to 6. It is the number of the anchor pair and in the YOLO tiny model number of anchor pairs is 6.
   We set the random parameter to 1 to increase the precision of the model which just selects the random image batches from the dataset.

```
[yolo]
mask = 0,1,2
anchors =  27, 60,  93,225, 128,231, 117,302, 143,304, 171,288
classes=5
num=6
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

8. In this obj.data file we have defined the number of class, the path to train file, the path to test file, the path to names file(the names of the object is defined in this file) and the path to the backup folder where the model will be stored after the training.



9. We cloned our project files to the google colab.

```
[ ]  #Cloning the required files

     !git clone https://github.com/sharmaansh/darknet.git
```

10 . we mounted the google drive for the backup of our trained model after the model is trained.

```
[ ]  from google.colab import drive
     drive.mount('/content/drive') #to back up the model in the drive
```

11 . After that we installed the required dependencies and compile the darknet files and update and upgrade packages to latest versions.

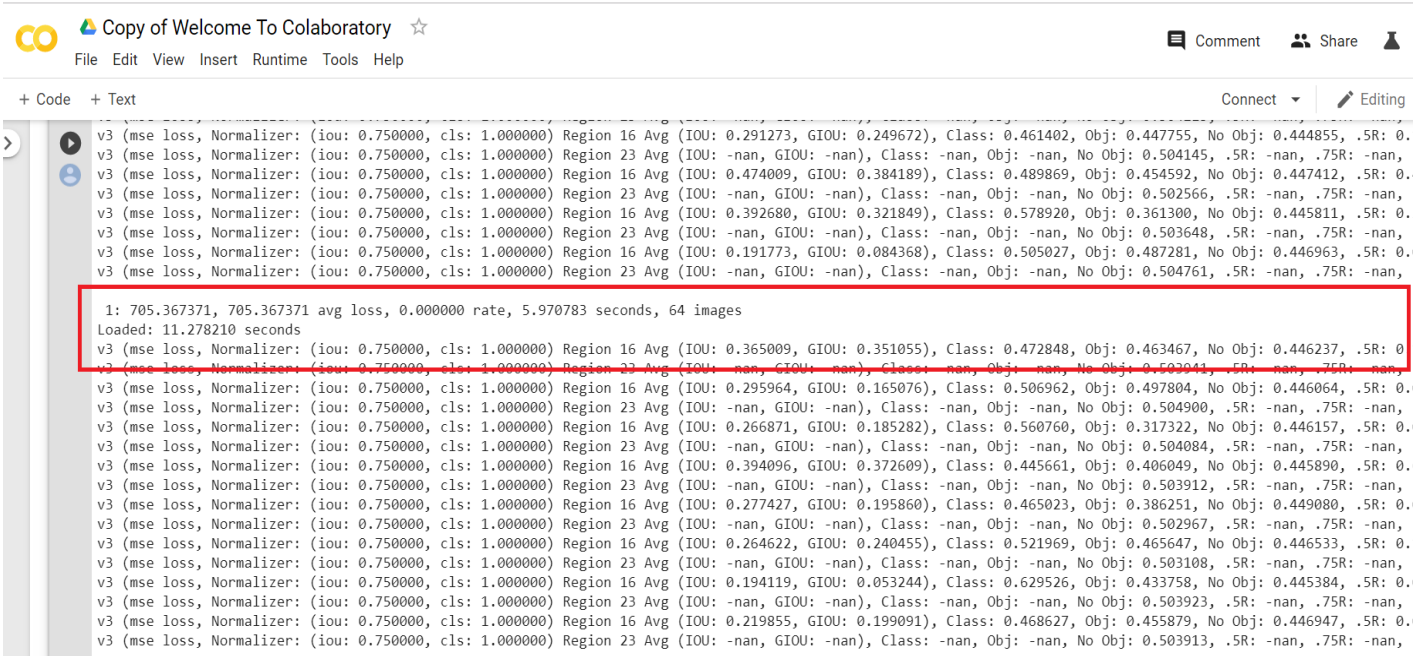
```
[ ]  # installing the required dependencies

     !apt-get update
     !apt-get upgrade
     !apt-get install build-essential
     !apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
     !apt-get -y install cmake
     !which cmake
     !cmake --version
     !apt-get install vim
```

```
[ ]  %cd darknet/
     !ls
     !sed -i 's/OPENCV=0/OPENCV=1/g' Makefile
     !sed -i 's/GPU=0/GPU=1/g' Makefile # to commpile with opencv and GPU
     !ls
     %cd ../
     !ls
     !apt install g++-5
     !apt install gcc-5
     !update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-5 10
     !update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-5 20
     !update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-5 10
     !update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-5 20
     !update-alternatives --install /usr/bin/cc cc /usr/bin/gcc 30
     !update-alternatives --set cc /usr/bin/gcc
     !update-alternatives --install /usr/bin/c++ c++ /usr/bin/g++ 30
     !update-alternatives --set c++ /usr/bin/g++
     !apt update -qq;
     !wget https://developer.nvidia.com/compute/cuda/8.0/Prod2/local_installers/cuda-repo-ubuntu1604-8-0-loc
     !dpkg -i cuda-repo-ubuntu1604-8-0-local-ga2_8.0.61-1_amd64-deb
     !apt-get update -qq
     !apt-get install cuda -y -qq #gcc-5 g++-5
     !apt update
     !apt upgrade
     !apt install cuda-8.0 -y # Installing CUDA
     %cd darknet
     !ls
     !make #Compilig the Darknet with GPU and openCV
```

12 . Now this is the command to start the training having the path to obj.data fil, the configuration file and the path to the R-CNN file.

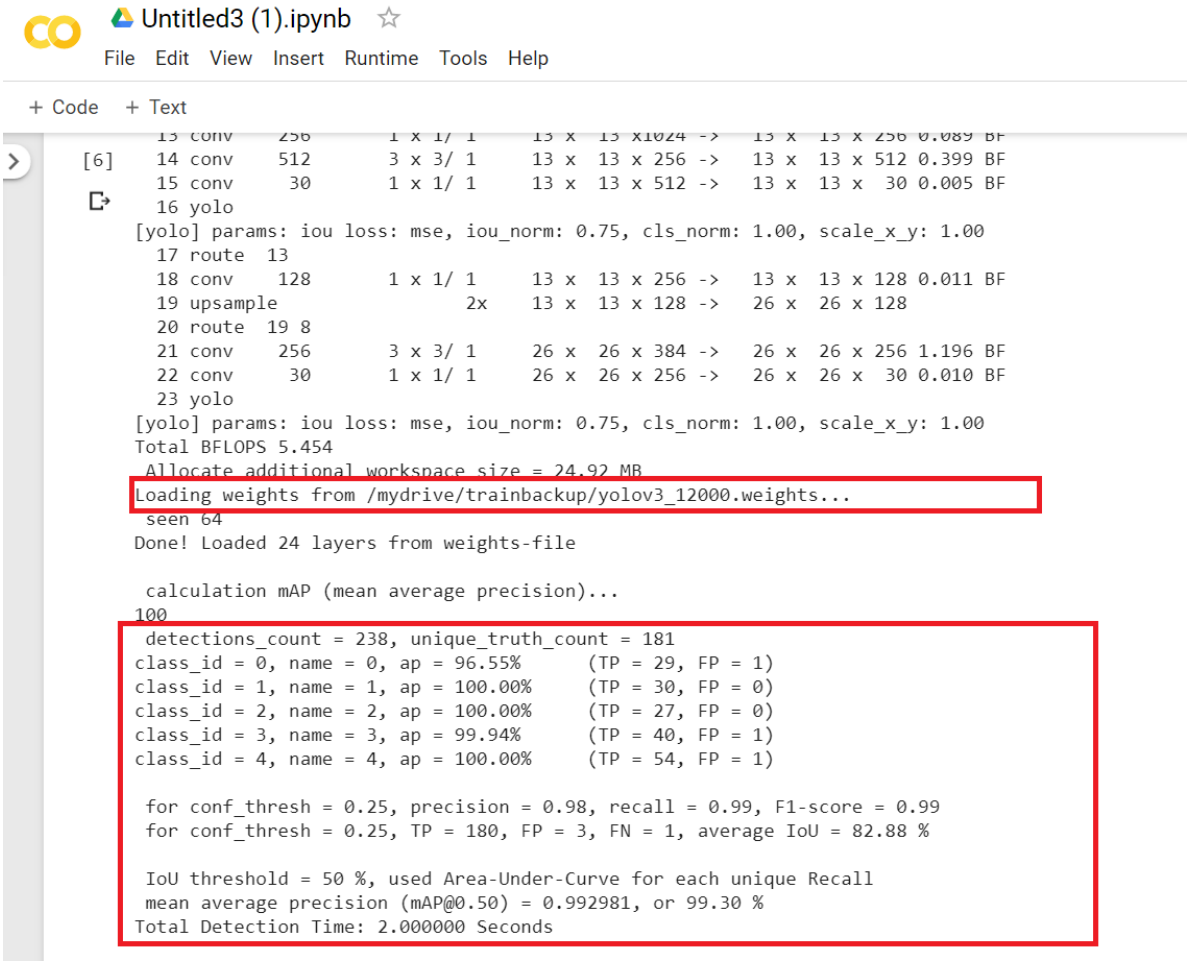
```
#Start the training

!./content/darknet/darknet detector train /content/darknet/data/obj.data /content/darknet/cfg/yolov3-tiny_obj.cfg \
/content/darknet/yolov3-tiny.conv.15 -dont_show
```

13 . This is the sample output which was generated at the very beginning of the training. These were the value of the learning rate, class detection percent, average loss , IOU, object detection percent and no object detection percent.



14 . As more iteration passed the average loss increased and decreased and after 12000 iterations the final values of our trained model were:-

Average Precision for each class were 96,100,100,99 and 100.
The final precision was .98
The F1 score was .99
The average IOU was 82%.
And the final mAP (mean Average Precision) was 99.2%.



7) Scope of Automation

- Usually if a dedicated hardware is maintained for processing the data require a lot of initial and maintenance cost
- in our case its just a camera and simple edge server.
- Cloud services also offer training machine learning Models which save time as well as cost.
- Hence, use of such services can reduce the cost a lot.

8) Conclusion

- Edge computing give ability to analyze data closer to the source, will minimize latency, reduce the load on the internet, improve privacy and security, and lower data management costs.