Object Detection Using Deep Learning

Report submitted to **Indian Institute of Information Technology, Allahabad** for

Summer Internship, 2019.

Rutuja Umesh Madhure

Indian Institute of Information Technology, Pune

B.Tech: Electronics and Communication Engineering

CERTIFICATE

This is to certify that **Ms. Rutuja Umesh Madhure**, 2nd year student of Electronics & Communication Engineering Department has worked on a project entitled as **"Object Detection using Deep Learning"** under the supervision of **Dr. Rahul Kala**, Assistant Professor in Indian Institute of Information Technology, Allahabad during the period of May-July, 2019 in the fulfilment of summer internship programme.

Dr. Rahul Kala

(Assistant Professor)

ACKNOWLEDGEMENT

I would like to thank **Dr. Rahul Kala** for his valuable time and efforts for giving me this opportunity to work on the project. His guidance and support throughout the internship helped me to successfully complete the project.

I would like to thank the Director of IIIT Pune, Dr. Anupam Shukla for providing me this internship opportunity. I would also like to express my gratitude towards my parents for their support and encouragement.

CONTENTS:

1. INTRODUCTION:

Computer vision briefly means the ability of computers to analyze and process images and videos similar to the visual system in humans. Computer vision is a widely growing field with constant research done in various domains. Convolutional Neural Networks (CNN) basically use the convolution concepts to perform tasks similar to the neural networks in human visual systems. Deep Learning uses stack of several layers to extract valuable information about the input data which are known as features. It branches out into several types of learning majorly, Supervised Learning, Unsupervised Learning, Reinforcement Learning and Transfer Learning. The approach used in this project is Transfer Learning. Transfer Learning is a technique which uses pre-trained weights on a large dataset to fine-tune a new dataset. This is a significant process as the network has previously learnt a lot of features from the large dataset, hence, it can now easily adapt to the features of the new dataset. This approach gives good results and requires less learning time as well as small dataset. An important concept in deep learning is loss function. Loss function simply provides a measure of accuracy of the model by using a mathematical function relating the predicted value and the actual value. The most commonly used loss function is the mean squared error loss function.

2. PROBLEM STATEMENT:

Imagine that you are told to classify images of tables and chairs. This is plainly a 'Classification' problem. But you may wonder that it would be better if one could determine the position of table or chair in the given image. This is a 'Classification with Localization' problem. However, the major drawback of a 'Classification with Localization' problem is that it is unable to detect all the objects in the image since classification yields a single output class (e.g. Chair) . To overcome this drawback , we can use a more robust technique which is 'Object Detection'.

Figure 1: Classification

Figure 2: Classification with Localization

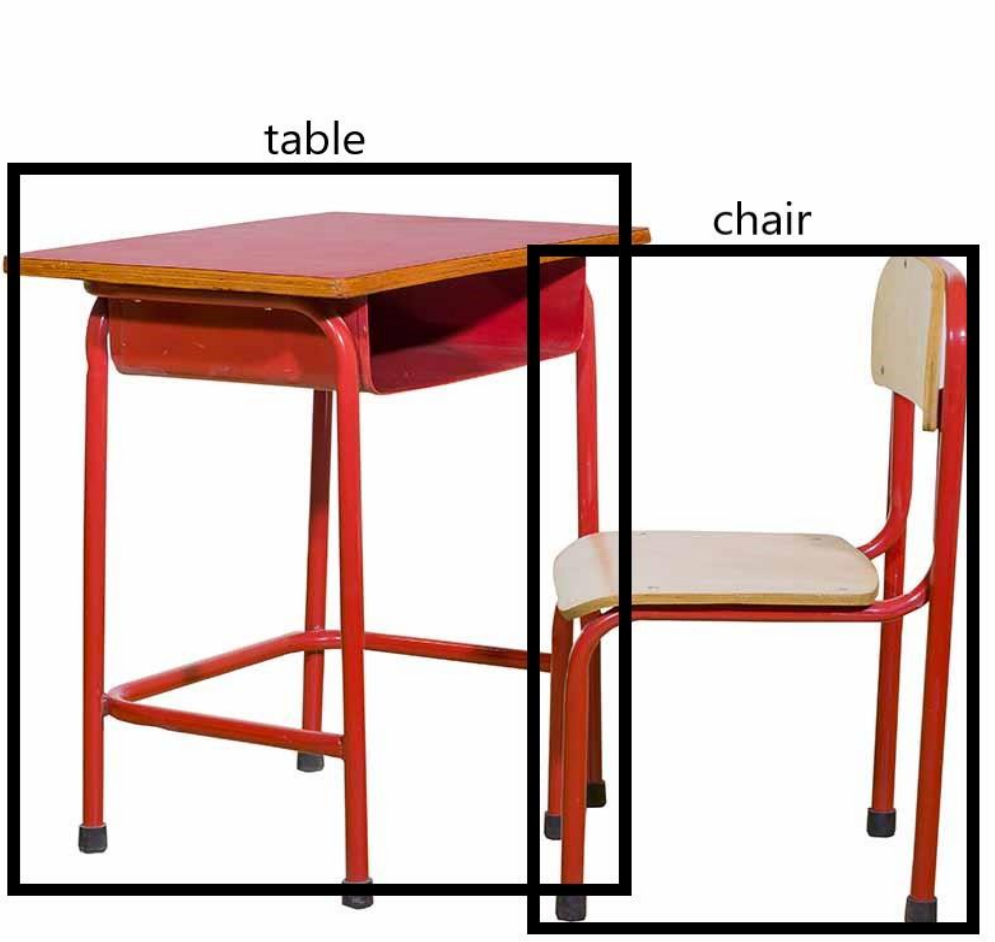


Figure 3: Object Detection

3. MODEL:

My project is based on the **YOLO(YOLOv3) Darknet Model** by **AlexeyAB** which is the forked version of the original darknet model by **Joseph Redmon**. YOLO means You Only Look Once. It has state of the art abilities and proves to be one of the most efficient models when it comes to real time object detection. Its performance on the COCO dataset can be seen on the graph below.

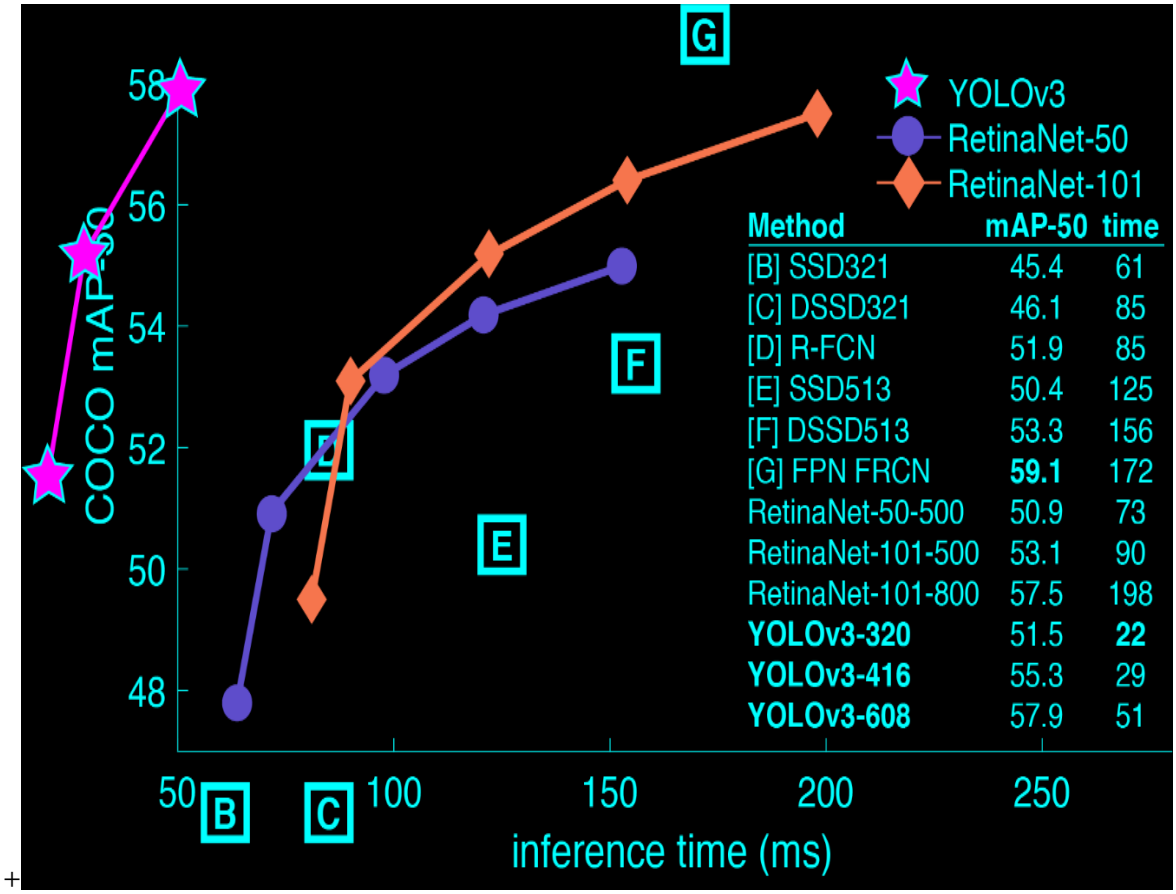


Figure 4: Performance of different networks on COCO dataset

(source of above image : https://pjreddie.com/darknet/yolo/)

A typical classifier can be modified to work as an object detector if image is analyzed in small parts. Every time we can run the classifier on a small part of the image and classify it. Further, we can localize the objects in these image segments and only retain the most confident predictions. However, this traditional approach is time consuming and not efficient. YOLO, on the other hand, exhibits a finer approach.

YOLO is a convolutional neural network (CNN). It is a stack made up of several units of convolution layers, leaky ReLU layers, max pool layers and fully connected layers. It uses the mean squared error loss function. The loss function takes into account classification loss (error in predicting class), localization loss(error in predicting bounding boxes) and object confidence loss(error in predicting presence or absence of an object).

The bounding box prediction of whether an object is present or not is done using Logistic Regression. For prediction of class, independent logistic classifiers are used instead of softmax layer.

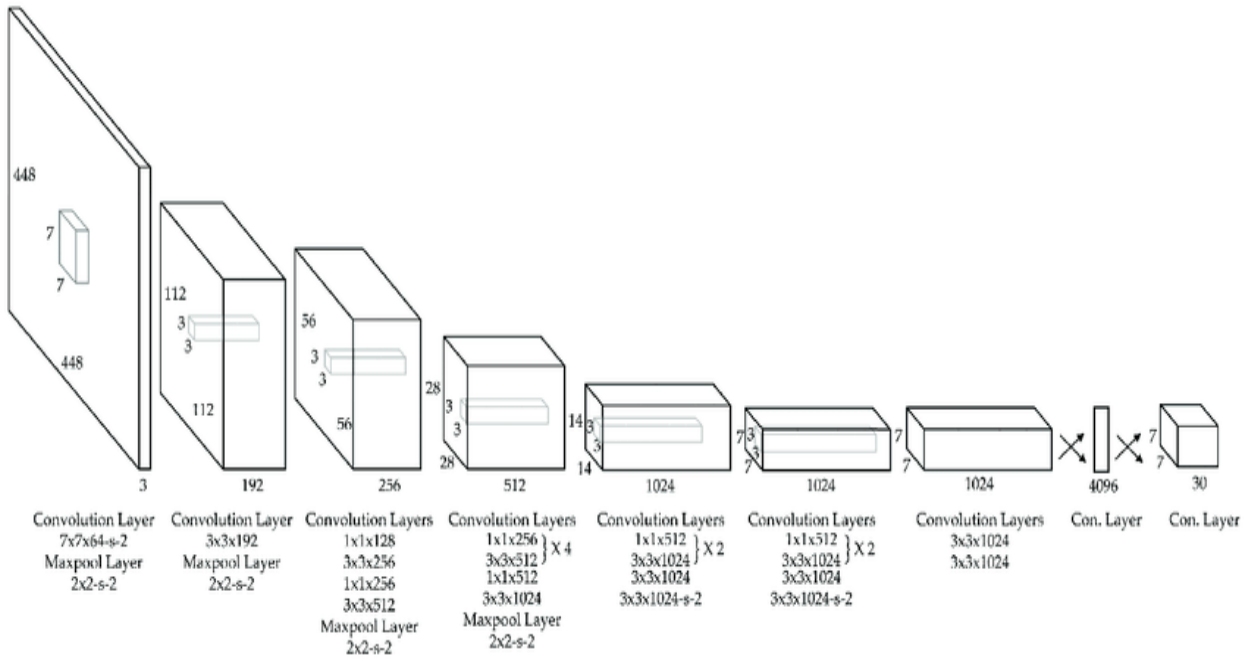


Figure 5: YOLO Architecture (source: Google image)

YOLO divides the image into specific number of grids 'g x g' (Commonly, number of grids is 19x19). Every grid now determines if any object(based on the classes used) is present or absent (p_c=0 or 1) where p_c is the probability that a class/object exists. Next, it identifies the class to which the object in the grid belongs. It outputs the confidence scores for each and every class (c_1,c_2,c_3.....). The algorithm creates 'm' number of bounding boxes(Usually, m=5) for every grid in the image. Each bounding box has the co-ordinates of the top-left corner (b_x,b_y) along with its height and width (b_h,b_w). Hence, the output 'Y' can be visualized as follows:

$$Y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Figure 6: Output ‘Y’

(source of above image:https://medium.com/machine-learning-bites/deeplearning-series-objection-detection-and-localization-yolo-algorithm-r-cnn-71d4dfd07d5f)

The class probabilities are considered as the weights for the bounding boxes. Finally, the model outputs the most confident predictions.

4. DARKNET 53:

Darknet 53 feature extractor is made up of 53 convolutional layers and is used in YOLOv3. Each of these layers is followed by batch normalization layer and leaky ReLU layer. The YOLOv2 model used Darknet 19. This project makes use of the pre-trained weights darknet53.conv.74 for transfer learning. It is a powerful feature extractor.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 7: Darknet 53

(source: https://towardsdatascience.com/review-yolov3-you-only-look-once-object-detection-eab75d7a1ba6)

5. DATASET:

My dataset includes the campus images of Indian Institute of Information Technology, Allahabad. I collected images that contain varying backgrounds for three classes- Bench, Dustbin and Boards. The dataset was prepared with utmost care to include images with varying

brightness, orientation, distance and background. For the training purpose, I have used 1,270 images uniformly distributed amongst three classes. The resolution of all images used is 640x480 px.

- **Important points while preparing dataset:**

1.) Try to bring variations in the background by collecting images from various locations.
2.) Change the brightness of images.
3.) Include images containing multiple objects as it enhances the training process.
4.) Images need to be uniformly distributed amongst all classes to avoid bias.
5.) Let the objects be present at different parts of the image and not simply in the center.
6.) You may use images with varying resolutions to create a more robust model

6. REQUIREMENTS:

Dataset has to be prepared by collecting images and annotating them correctly as per YOLO format. Google Colab has been used for the project which is a very popular framework for various machine learning and deep learning applications as it provides a free GPU and interactive user interface. The project is based on an efficient implementation of the YOLO algorithm which is Darknet model. Darknet is written in C and based on CUDA technology. Darknet model used for training has been downloaded from github - AlexeyAB darknet model. The project uses CUDA version 10.0 (CUDA is pre-installed on Colab). OpenCV library is used to carry out detections on videos. All tasks have been carried out on Ubuntu 16.04 LTS operating system.

Tool used for annotation of images: **labelImg** tool.
The labels generated are in YOLO format which is a .txt file. One .txt file per image is required with every object's bounding box description on a new line for all objects in that image.

Format : <class> <x> <y> <width> <height> ,where
<class> is an integer number from 0 to (number of classes – 1)
<x> <y> are center of rectangle
<x>=absolute-x/image-width and <y>=absolute-y/image-height
<width>=object-width/image-width and <height>=object-height/image-height

Other details: CPU used is Intel Core i3-3220 @ 3.3GHz x 4, OS : ubuntu 16.04 LTS, 64 bit

7. TRAINING:

The approach that I have used for training is transfer learning. I have used the pre-trained weights **darknet53.conv.74** for fine-tuning the model on three classes- Bench, Dustbin, Boards. All the images used for training are present in a folder(e.g. Obj) inside the data folder of darknet directory(AlexeyAB darknet -repository from github). The path of all these images has to be mentioned in a separate file(.txt) in data folder. The names of all classes used for training is inside a file present in data folder. There is one more file in data folder which contains the paths of the previously mentioned files(e.g. Obj.data) The configuration file(.cfg) is placed in the cfg

folder and contains model design and valuable parameters. Important parameters used during the training process are as below:

1.) Batch=64
2.) Subdivisions=32
3.) Learning rate=0.0001
4.) Steps=100,1500
5.) Scales=10,0.1
6.) Epochs=3000

Data augmentation is also used during the training process. Parameters such as hue, saturation in the configuration file allow the user to add variations to the dataset.

The command used for training is given below:

./darknet detector train your_path/obj.data your_path/yolo-obj.cfg darknet53.conv.74 (Linux)

darknet.exe detector train your_path/obj.data your_path/yolo-obj.cfg darknet53.conv.74 (Windows)

During training the weights are saved in the backup folder after every 100 iterations with the name yolo-obj_last.weights. After every 1000 iterations, the weights are stored as yolo-obj_1000.weights, yolo-obj_2000.weights and so on.

To restart the training using the backup weights, modify the train command as below:

./darknet detector train your_path/obj.data your_path/yolo-obj.cfg backup/yolo-obj_1000.weights (Linux)

darknet.exe detector train your_path/obj.data your_path/yolo-obj.cfg backup/yolo-obj_1000.weights (Windows)

8. DETECTION:

After the training process is completed, detection of the objects using the latest weights obtained in the backup folder is done as follows:

Detection on images:

darknet.exe detector test path/obj.data path/yolo-obj.cfg path/yolo-obj_3000.weights path/test_image.jpg (Windows)

./darknet detector test path/obj.data path/yolo-obj.cfg path/yolo-obj_3000.weights path/test_image.jpg (Linux)

Detection on videos:

The test video is test_video.mp4 and the output video with detections is stored in a file named result.avi.

darknet.exe detector demo path/obj.data path/yolo-obj.cfg path/yolo-obj_1000.weights

-dont_show path/test_video.mp4 -i 0 -out_filename result.avi (Windows)

./darknet detector demo path/obj.data path/yolo-obj.cfg path/yolo-obj_1000.weights -dont_show path/test_video.mp4 -i 0 -out_filename result.avi (Linux)

WAYS TO TRAIN MORE EFFICIENTLY:

- In the configuration file (.cfg), set parameter random=1 for better training process on various image resolutions.
- One may change the network parameters for height and width in .cfg file to a greater multiple of 32 in order to increase accuracy at the cost of increased time and space consumption during training.
- It is important to verify the labels in the training dataset else it might hamper the training process.
- Include many variations of a particular object class to enhance training. The model will not be able to detect an object in the extreme left/right if all training examples contain object at the centre.
- One may stop the training process prior to completion of all the epochs if average loss reaches 0.05 or becomes constant. This can help to prevent overfitting.
  (Overfitting is a condition where the model performs excellently on the training set and poorly on new, unseen images)

9. RESULTS:

The below results were obtained on some test images from the campus (these images were not included during training process).

1. **Domination of bigger object in the image:**

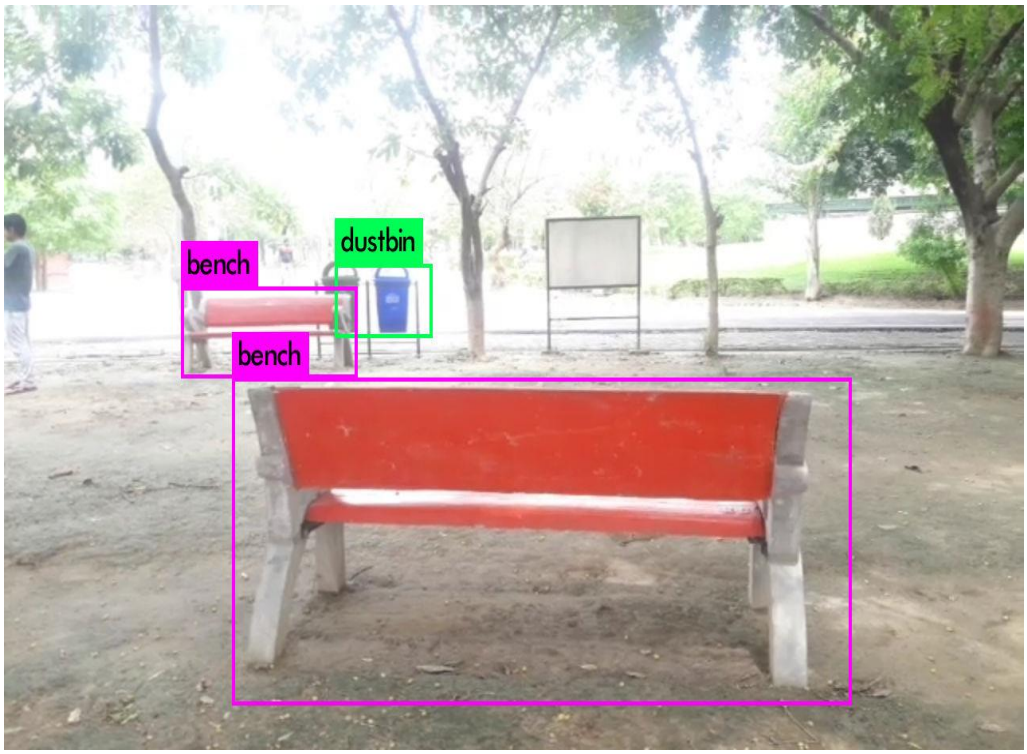
2. **Performance on slight side view of object:**

3. **Performance on front view of object:**

4. **Performance on varying object distance:**

5. **Performance on back view of object:**

6. **Performance on multiple objects:**



10. CONCLUSION:

The network proved to be efficient in object detection on videos and images, however, there is a lot of scope for improvement. It can effectively deal with several barriers that are faced during object detection in real time scenarios such as variations in illumination, angle, distances, etc. It can be used in real time applications where object detection is one of the component. YOLO is a high speed network and hence suitable for real time applications. It is currently one of the fastest models for object detection. It can be widely used in self driving cars, autonomous drones to identify people in remote places during natural disaster, to identify diseases by observing the plant leaves or other parts, identification of rare species of plants and animals in dangerous or unexplored habitats.

11. REFERENCES:

- @article{yolov3,
  title={YOLOv3: An Incremental Improvement},
  author={Redmon, Joseph and Farhadi, Ali}, journal = {arXiv}, year={2018} }
- https://github.com/AlexeyAB/darknet
- https://pjreddie.com/darknet/yolo/

APPENDIX A

(Training Procedure)

REQUIREMENTS:

- Linux GCC >=4.9 or **Windows MS Visual Studio 2015 (v140)**
- **CUDA 10.0**
- **OpenCV 3.3 or OpenCV 2.4.13**
- **GPU with CC >=3.0**

OVERVIEW:

All the above requirements are already satisfied on Google Colab. Hence, for this project, I have done the training using Colab on CPU: Intel Core i3-3220 @ 3.3GHz x 4 and OS : ubuntu 16.04 LTS, 64 bit. The project is based on the YOLO(yolov3) Darknet model and uses Transfer Learning to fine-tune the network speedily and efficiently.

PROCEDURE:

Step 1:

Preparation of Dataset

1. Collect the training images for various classes that you want to train, preferably 2000 images per class. Save images in the .jpg format.
2. Make sure that the images include variations in background, orientation, illumination and distance.
3. Include images with multiple objects belonging to different classes that you want to detect.

Step 2:

Labelling the Dataset

1. LabelImg tool allows the user to draw bounding boxes on images and saves the annotations in YOLO format which is a .txt file.
2. Run the following command on Linux to download the github repository of labelImg tool.
   Command: git clone https://github.com/tzutalin/labelImg.git
3. Create a 'classes.txt' file containing the names of all the classes that you want to train inside the folder containing all of the training images. Each class name has to be on a new line as below:
   Bench
   Dustbin
   Board

4. Now, run the following commands from the labelImg directory on your terminal as per the python version you use.

   Python 2

   sudo apt-get install python-qt4
   sudo apt-get install pyqt4-dev-tools
   sudo pip install lxml
   make qt4py2
   python labelImg.py

   Python 3

   sudo apt-get install python3-pyqt5
   sudo apt-get install pyqt5-dev-tools
   sudo pip3 install -r requirements/requirements-linux-python3.txt
   make qt5py3
   python3 labelImg.py
5. The GUI of labelImg tool will now appear on the Desktop. Click on the 'Open Dir' icon and select your images folder.
6. Use the following keyboard shortcuts:
   w : Create a rectangular box
   d : Next image
   a : Previous image
   ctrl + s: Save the image
7. It will create .txt-file for each .jpg-image-file - in the **same directory** and with the same name, but with .txt-extension.
8. The labels generated are in YOLO format which is a .txt file.

   Format : <class> <x> <y> <width> <height> ,where
   <class> is an integer number from 0 to (number of classes – 1)
   <x> <y> are center of rectangle
   <x>=absolute-x/image-width and <y>=absolute-y/image-height
   <width>=object-width/image-width and <height>=object-height/image-height

Step 3:

Download darknet model

We will prepare the complete Darknet directory on local computer and then upload the folder on Google Drive. After mounting Drive on Colab, we can access all the files present on our Drive. This is useful since the weights will appear in the folder present on your Drive in case Colab crashes or gets disconnected.

1. Open Linux Terminal.
2. Type the following command to download AlexeyAB's Darknet model:
   git clone https://github.com/AlexeyAB/darknet.git

(You may create a folder to place this darknet directory. I created a folder named '**project_last**'.)

Step 4:

Preparation of configuration file

1. Create an empty document named ‘yolo-obj.cfg’ in the /darknet/build/darknet/x64/cfg.
2. Copy the contents of ‘yolov3.cfg’ to ‘yolo-obj.cfg’.
3. Change the following parameters in ‘yolo-obj.cfg’
    - Change batch to batch=64
    - Change subdivisions to subdivisions=32
    - Change max_batches to (classes*2000) e.g. max_batches=6000 for 3 classes. However, if the number of images is less then the max_batches should be reduced to avoid overfitting.
    - Change steps to 80% and 90% of max_batches e.g. steps=4800,5400
    - Change scales to scales=10, 0.1 which alters the learning rate by this factor after ‘steps’ epochs.
    - Change learning_rate to learning_rate=0.0001
    - Change classes=80 to your number of classes e.g. classes=3 in each of the last 3 [yolo] layers. (Line numbers 610, 696 and 783 in .cfg file)
    - Change filters=255 to filters=(classes+5)x3 e.g. filters=24 in the 3 [convolutional] layer before each [yolo] layer. (Line numbers 603, 689 and 776 in .cfg file)

      (Do not write in the cfg-file: filters=(classes + 5)x3. It should be filters=24)

Step 5:

Preparation of ‘obj.names’ file

1. Create an empty document named ‘obj.names’ in the directory /darknet/build/darknet/x64/data.
2. Mention the class names each on a new line.
    e.g. bench
    dustbin
    board

Step 6:

Preparation of ‘obj.data’ file

1. Create an empty document named ‘obj.data’ in the directory /darknet/build/darknet/x64/data containing

```
classes= 3
train  =/myDrive/build/darknet/x64/data/train.txt
valid  =/myDrive/build/darknet/x64/data/test.txt
names = /myDrive/build/darknet/x64/data/obj.names
backup =/myDrive/build/darknet/x64/backup
```

(**NOTE:** The paths should be with respect to Colab after mounting your drive. I have created a symbolic link in Colab notebook which is as below:

!ln -s /content/drive/My\ Drive/project_last/darknet/ /myDrive

Now, I can refer '/content/drive/My\ Drive/project_last/darknet/' path simply as '/myDrive'.)

Step 7:

Placing images directory inside darknet

1. Create a new folder inside /darknet/build/darknet/x64/data named 'obj'.
2. Place all the image files (.jpg) and all the (.txt) files associated with them inside /darknet/build/darknet/x64/data/obj.

Step 8:

Creating 'train.txt' file

1. Create an empty document named 'train.txt' inside /darknet/build/darknet/x64/data.
2. The file must contain the full path of all the images (.jpg) present in the 'obj' folder. Every path should be on a new line. The names of .txt files should not be included.

(**NOTE:** The paths should be with respect to Colab after mounting your drive. I have created a symbolic link in Colab notebook which is as below:

!ln -s /content/drive/My\ Drive/project_last/darknet/ /myDrive

Now, I can refer '/content/drive/My\ Drive/project_last/darknet/' path simply as '/myDrive'.)

e.g. Few lines of train.txt

/myDrive/build/darknet/x64/data/obj/frame1194.jpg
/myDrive/build/darknet/x64/data/obj/frame329.jpg
/myDrive/build/darknet/x64/data/obj/frame1591__rot90.jpg
/myDrive/build/darknet/x64/data/obj/frame384.jpg

Step 9:
Downloading the pre-trained weights for transfer learning.

1. Download pre-trained weights for the convolutional layers (154 MB): https://pjreddie.com/media/files/darknet53.conv.74 and place in the directory /darknet/build/darknet/x64.
2. Since, we are using the transfer learning approach, we will be using these pre-trained weights.

Step 10:

Now, the 'project_last' which contains the darknet directory has to be uploaded on Google Drive.

Step 11:

Open Colab Notebook-Python 3. Select runtime as **GPU**. Run following commands on Colab Notebook

1. **# Run this to mount your Google Drive.**

   from google.colab import drive

   drive.mount('/content/drive')

2. **# To use Symbolic link**

   !ln -s /content/drive/My\ Drive/project_last/darknet/ /myDrive

   !ls /myDrive

3. **#To change directory**

   %cd /myDrive

4. **#Change the variables to include OpenCV and GPU in the Makefile**

   %cd /myDrive

   !sed -i 's/OPENCV=0/OPENCV=1/g' Makefile

   !sed -i 's/GPU=0/GPU=1/g' Makefile

5. **#To ensure that OpenCV runs without any errors.**

   !apt-get install libopencv-dev

6. **#make command**

   !make

7. **#Training command**

   %cd /myDrive

   !./darknet detector train /myDrive/build/darknet/x64/data/obj.data /myDrive/build/darknet/x64/cfg/yolo-obj.cfg /myDrive/build/darknet/x64/darknet53.conv.74 -dont_show

(**NOTE:**

- File yolo-obj_last.weights will be saved to the /myDrive/build/darknet/x64/ backup for each 100 iterations.
- File yolo-obj_xxxx.weights will be saved to the /myDrive/build/darknet/x64/backup for each 1000 iterations.

- After training is complete, final weights yolo-obj_final.weights will be stored in /myDrive/build/darknet/x64/backup.
- If error- Out of memory occurs then increase the number of subdivisions to 64.
- One can even stop the training and resume it again by using the last stored weights in the backup folder as follows:

  !./darknet detector train /myDrive/build/darknet/x64/data/obj.data /myDrive/build/darknet/x64/cfg/yolo-obj.cfg /myDrive/build/darknet/x64/backup/yolo-obj_last.weights -dont_show

- You can stop the training if average loss settles at 0.06 or less.)

Step 12:

Testing the model

1. **#To test on images**

   %cd /myDrive

   !./darknet detector test /myDrive/build/darknet/x64/data/obj.data /myDrive/build/darknet/x64/cfg/yolo-obj.cfg /myDrive/build/darknet/x64/backup/yolo-obj_final.weights /myDrive/build/darknet/x64/data/test_image.jpg

   (The result is stored inside a file named 'predictions.jpg' in darknet directory.)

2. **#To test on video**

   %cd /myDrive

   !./darknet detector demo /myDrive/build/darknet/x64/data/obj.data /myDrive/build/darknet/x64/cfg/yolo-obj.cfg /myDrive/build/darknet/x64/backup/yolo-obj_final.weights -dont_show /myDrive/test_video.mp4 -i 0 -out_filename res.avi

   (The result is stored as 'res.avi' inside the darknet directory.)
