UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY
PANJAB UNIVERSITY, CHANDIGARH - 160014, INDIA

Project report on

Accident Site detection using deep learning

Submitted By :

Name	Roll Number	Contact No.
Nikhil Garg	UE173067	9915238133
Sumit Kumar	UE173107	8360797550
Udhay Nath	UE173112	7696567658

Under the Supervision of
Dr. Sarabjeet Singh

Department of Computer Science and Engineering,
University Institute of Engineering and Technology,
Panjab University, Chandigarh-160014, India
November 2020

CONTENTS

CERTIFICATE ……3

ACKNOWLEDGEMENT……..4

1. INTRODUCTION

1.1. Accident Site detection…….5
1.2. Objective……...6
1.3. Challenges…….7
1.4. Object Detection Techniques……....8
1.5. Model Outcome……10

2. BACKGROUND

2.1. Object Detection Using Traditional Computer Vision Technique……..12
2.1.1. Standard representation of workflow of Computer Vision System……12
2.1.2. Advantage of Computer Vision System……..12
2.2. Object Detection Using Deep Learning……..…..13
2.2.1. Example Of Deep Learning ……14
2.3. YOLO……..15
2.3.1. YOLO v1……...15
2.3.2. YOLO v2……...18
2.3.3. YOLO v3……..20

3. IMPLEMENTATION OF OBJECT DETECTION MODEL

3.1. Implementation Approach……...24
3.2. Dataset……..24
3.3. Model……28
3.4. Object Detection Process……..30
3.4.1. Dataset Preparation……..31
3.4.2. Setup Colab Environment……..….31
3.4.3. Installing Darknet……...32
3.4.4. Modify Configuration File…….……32
3.4.5. Split train/test dataset……....33
3.4.6. Training the Model……....34

3.4.7. Predict With YOLOv3……....34
3.5. Results and Output……..….34
4. CONCLUSION……...…..37
5. REFERENCES……..…..38

CERTIFICATE

This is to certify that the project entitled “ACCIDENT DETECTION USING DEEP LEARNING” is a bonafide work done by **Nikhil Garg** (Roll Number: UE173067) and **Sumit Kumar** (Roll Number: UE173107) and **Udhay Nath** (Roll Number: UE173112) as a part of MINOR PROJECT in the Department of Computer Science and Engineering at **University Institute of Engineering and Technology, Panjab University**. This project was carried out by them under my guidance and has not been submitted elsewhere.

Dr.Sarabjeet Singh
Department of Computer Science and Engineering
Panjab university , Chandigarh

Acknowledgement

We sincerely thank our supervisor DR. Sarabjeet Singh for giving us the opportunity to work on this project and also his constant supervision and valuable guidance during the course of the thesis. We also express our sincere gratitude to the Department of Computer Science and Engineering for providing us with all the necessary facilities required for the completion of this project. We also take help from the udhemy course given by our mentor and for that we are very grateful.

Nikhil Garg

Sumit Kumar

Udhay Nath

CHAPTER 1

INTRODUCTION

A total of 1, 51,113 people were killed in 4,80,652 road accidents across India in 2019. An average of 414 a day or 17 an hour, according to a report by the transport research wing of road transport and highway. Road accidents are major cause of the deaths in India. And many of road accidents occur because of over speeding and rash driving which also include the minor cases. While in terms of vehicles Two-wheelers were involved in most road fatalities.Road accidents cost a lot of lives and property damage. In order to avoid accidents, everyone should be more careful and vigilant on the streets. As a driver, one must be extra cautious because one mistake from the driver can cost the lives of the innocent ones.The main problem and the cause of the accident is the sheer carelessness of the drivers and their driving style which may cause trouble to themselves and also to the other persons on road. One must follow all the rules and regulations which are advised by the government.

Emergency response to roadway crashes is very important for traffic management. On the one hand, people injured in a crash need to be sent to the nearest hospital in the first place to prevent their health condition from being worsened.On the other hand, serious crashes often cause congestion, if emergency response or clearance is not carried out in time. In order to mitigate those negative impacts, roadway crashes need to be quickly detected.So this project mainly focuses on detection of accidents on site. In real time it will be helpful as we are able to identify the location of an accident, we can ensure on time delivery of healthcare facilities. The collection and use of accurate and comprehensive data related to road accidents is very important to road safety management.

1.1 Accident Site Detection : Overview

The model is to detect the accident at the site and will identify the objects in terms of vehicle class whether the object is car, bus, van etc. and also detect the accident that is whether the accident happened or not. Proper labeling will be there which make sure to give whole information from the site we examine.This model will also run on videos and in real time which will detect the accident in a fraction of seconds. Mainly our focus is on the far moving objects which must be captured by cctv cams as they are located on some height and it becomes some challenging task to detect objects from far images captured and classify them accordingly. We train our model in such a way that it will do the task of accident detection from far sites and cctv footages.

This model is based on the technique of Object detection, which is one of the challenging features to artificial neural networking.Every object class has its own special features that helps in classifying the class And the platform we are using is YOLO v3(You look Only Once version 3). Using Darknet -53 framework. Darknet -53 comprises 53 convolutional layers and is stacked with 53 more layers. So, basically this model consists of 106 layers which will read the image and give the output.Simply this model is capable of classifying vehicles and detecting the accident. YOLO runs 45fps which means it can run 45 frames per second and there is a very large number of frames to read and identify classes. This will help in detecting accidents with less loss. Main problems occur in the cases of high speed moving vehicles where it may confuse the model to detect the accident in which the contact period of vehicles are of very much short period. And The YOLO model is consummate in this task as it performs function on 45 fps and it is easy for the model to render the short period of accident.

1.2 Objective

The main objective of the model is to tell whether the accident took place or not and also detecting the accident on site.Mainly our focus is on the far moving objects which must be captured by cctv cams as they are located on some height and it becomes some challenging task to detect objects from far images captured and classify them accordingly. We train our model in such a way that it will do the task of accident detection from far sites and cctv footages. The objective to design a model which will tell in run-time about the accident according to inputs fed to the model. Accident detection model is an object detection model which works in run-time and YOLO performs this work very well whose speed is 25 fps and is very much fast. There are different types of vehicles that can be found running on roads and their shapes are also different from each other. And we also see that accidents can occur anywhere and at any time and these accidents are of different types. Maybe it can be face to face collision of vehicles or it may be ramming vehicle from sidewise anyhow. So this model is successful in detecting the accident site with much less loss.

As the main goal for a object detection model is to classify objects whether it is large or small in size.Typically only a small number of instances of the object are present in the image, but there is a very large number of possible locations and scales at which they can occur and that needed to be explored. Each detection is reported with some form of pose information. This could be as simple as the location of the object, or the extent of the object defined in terms of a bounding box. For example an accident detector may compute the locations of the Vehicle and accident in addition to the bounding box of the face.Object detection systems construct a model for an object class from a set of training examples.

1.3 Challenges

As it is mentioned earlier that there are different types of vehicles that are running on the road so it is a challenging task for the model to classify the vehicles. Let's take an example of teaching a kid alphabets and numbers. As it takes time for a kid to learn all this similarly for an artificial neural network it is a challenging task to train a model like this.

We can classify vehicles on ground level as two wheelers and four wheelers. And there are also the sub-types in vehicles like two wheelers may be a bike, scooter, cycle etc. and in case of a four-wheeler there are a lot of cases.

So first to train the model to classify the vehicle is a challenging phase. After this the challenging task is to detect the accident. Mainly we call road accidents when any of two vehicles collide with each other and this is a very short period interval when the accident occurred. Training the model by feeding input of accidents also takes much more time.

Another challenging task is the quality. Different videos have different qualities. And the model has to detect all objects small as well as large. It is easy in high resolution videos but problems occur in small pixel videos and photos.

Viewpoint variation is also one of the challenges for object detection because from different views and angles vehicles appear to be different and accidents may also not be determined. So it is also one of the main problems and challenges for the model to recognise the accident and identify the class of vehicle as the model is trained from a different view point and is very much difficult to take all the aspects of accidents and vehicles.

Deformation also causes problems as Many objects of interest are not rigid bodies and can be deformed in extreme ways. We can take an example of trucks. There are also different types of trucks we can see such as lorry , tipper and dumper, same as in case of two wheelers.

Illumination condition also plays a vital role as the amount of light in an image because effects of illumination are drastic on pixel level. Objects exhibit different

colors under different illumination conditions. For example, an outdoor surveillance camera is exposed to different lighting conditions throughout the day, bright daylight, evening, and night light. An image of a pedestrian looks different in these varying illuminations.

Inter - class variation is also the main challenge. Because if we take the case of vehicles many of the things resemble each other. For example if you take an example of a bike, motorcycle and scooter they resemble each other, as they all are two wheelers and their body and shapes also resemble each other. Same in case of cars and vans.

Problems also occur while training the model as it can take a very long time to train a model on high memory data and on high epoc. And sometimes we face a problem in maintaining the graph once training model get struck we have to start training from the beginning and graph also get destructed.

1.4 Object Detection Technique : YOLO V3

YOLO- You only look once, real time object detection

A new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.
Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is far less likely to predict false detections where nothing exists. Finally, YOLO learns very general representations of objects. Our base YOLO model processes images in real-time at 45 frames per second. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is far less likely to predict false detections where nothing exists. Finally, YOLO learns very general representations of objects.

*DARKNET -53-:*YOLO uses convolutional layers. YOO v3 consists of 53 Convolutional layers that are also called DARKNET-53.

	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			

For detection layer original architecture is stacked with 53 more layers that give us 106 layers of YOLO and they can efficiently perform their task with less loss. Essential elements of this model are Residual block, skip connections, up-sampling. This mode takes input as a batch of image (n{no. Of images},416{width},416{height},3{RGB channel}).We can load a pre trained version of the network trained on more than a million images from the ImageNet database [1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 256-by-256.

1.5 Model Outcome

This model will label the vehicles and detect the accident site. As our model is defined to detect the accident and classify vehicles it is working for the same. We fed a video having several clips in which there are accidents. Our model works properly by classifying vehicles and accidents clearly even in the video.

Link for the video is listed below

https://drive.google.com/drive/folders/1hqm_HXNET3NgOvHF4S71tSPXT6AOBnYw?usp=sharing

Our main goal is to detect the accident which our model is successfully doing. It classifies vehicles also. A sample image is also shown below with confidence level which proves that the object resembles that particular object as the model is trained.

CHAPTER 2

BACKGROUND

Accident Detection problem is a subset of Object detection. This section presents an overview of related work done in this DomainMethods for object detection generally fall into either machine learning-based approaches or deep learning-based approaches. For Machine Learning approaches, it becomes necessary to first define features using one of the methods below, then using a technique such as support vector machine (SVM) to do the classification. On the other hand, deep learning techniques are able to do end-to-end object detection without specifically defining features, and are typically based on convolutional neural networks (CNN)

- Machine learning approaches:
    - Viola–Jones object detection framework based on Haar features
    - Scale-invariant feature transform (SIFT)
    - Histogram of oriented gradients (HOG) features.
- Deep learning approaches:
    - Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN, cascade R-CNN.)
    - Single Shot MultiBox Detector (SSD)
    - You Only Look Once (YOLO)
    - Single-Shot Refinement Neural Network for Object Detection (RefineDet)
    - Retina-Net
    - Deformable convolutional networks

Deep Learning has pushed the limits of what was possible in the domain of Digital Image Processing. However, that is not to say that the traditional computer vision techniques which had been undergoing progressive development in years prior to the rise of DL have become obsolete. Now these two sides of object detection have been combined to form new hybrid technologies. Several recent hybrid methodologies are reviewed which have demonstrated the ability to improve computer vision performance and to tackle problems not suited to Deep Learning. For example, combining traditional computer vision techniques with Deep Learning has been popular in emerging domains such as Panoramic Vision and 3D vision for which Deep Learning models have not yet been fully optimised.

2.1 Object Detection Using Traditional Computer Vision Techniques

The field of Computer Vision started gaining traction dating as far back as the late 1950s till the late 1960s when researchers wanted to teach computers "*to be…human*". It was around this time when researchers tried to mimic the human visual system in order to achieve a new stepping stone to endow machines with human intelligence. Thanks to the extensive research being done back then, Traditional CV techniques like Edge Detection, Motion Estimation, Optical Flow were developed.

Visual object detection has become popular in the past decade because it has finally reached the stage where many practical applications appear to be within reach.. To adapt to these difficulties, researchers have developed increasingly informative descriptors and powerful classifiers and training mechanisms. Visual descriptors are a basic component of any object detection system. They must capture sufficient information to distinguish class instances from non-class ones while being resistant to photometric and geometric changes. People have tried many different kinds of descriptors including: edges and gradients [4,5,6], convolutional net filters During the detection process, the classifier is scanned across the image at multiple scales and positions. For a give real class instance, this often produces several candidate detections that overlap closely in scale and space.

2.1.1 A standard representation of the workflow of a Computer Vision system is:

1. A set of images enters the system.
2. A Feature Extractor is used in order to pre-process and extract features from these images.
3. A Machine Learning system makes use of the feature extracted in order to train a model and make predictions.

2.1.2 Advantages of Traditional Computer Vision Techniques

• Scale Invariant Feature Transform (SIFT)

• Speeded Up Robust Features (SURF)

• Features from Accelerated Segment Test (FAST)

• Hough transforms

• Geometric hashing

2.2 Object detection Using Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

Deep Learning (DL) is used in the domain of digital image processing to solve difficult problems (e.g. image colourization, classification, segmentation and detection). DL methods such as Convolutional Neural Networks (CNNs) mostly improve prediction performance using big data and plentiful computing resources and have pushed the boundaries of what was possible. Problems which were assumed to be unsolvable are now being solved with superhuman accuracy. Image classification is a prime example of this. Since being reignited by Krizhevsky, Sutskever and Hinton in 2012 [1], DL has dominated the domain ever since due to a substantially better performance compared to traditional methods .Probably the most well-known problem in computer vision. It consists of classifying an image into one of many different categories.Regions with CNN features or R-CNN boasted an almost 50% improvement on the object detection challenge. they proposed was a three stage approach:

- Extract possible objects using a region proposal method (the most popular one being Selective Search).
- Extract features from each region using a CNN.
- Classify each region with SVMs.

2.2.1 Examples of Deep Learning at Work

Deep learning applications are used in industries from automated driving to medical devices.

Automated Driving: Automotive researchers are using deep learning to automatically detect objects such as stop signs and traffic lights. In addition, deep learning is used to detect pedestrians, which helps decrease accidents.

Aerospace and Defense: Deep learning is used to identify objects from satellites that locate areas of interest, and identify safe or unsafe zones for troops.

Medical Research: Cancer researchers are using deep learning to automatically detect cancer cells. Teams at UCLA built an advanced microscope that yields a high-dimensional data set used to train a deep learning application to accurately identify cancer cells.

Industrial Automation: Deep learning is helping to improve worker safety around heavy machinery by automatically detecting when people or objects are within an unsafe distance of machines.

Electronics: Deep learning is being used in automated hearing and speech translation. For example, home assistance devices that respond to your voice and know your preferences are powered by deep learning applications.

2.3 YOLO

YOLO came on the computer vision scene with the seminal 2015 paper by Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection,” and immediately got a lot of attention by fellow computer vision researchers.

There are three main variations of the approach, at the time of writing; they are YOLOv1, YOLOv2, and YOLOv3. The first version proposed the general architecture, whereas the second version refined the design and made use of predefined anchor boxes to improve bounding box proposal, and version three further refined the model architecture and training process.

Although the accuracy of the models is close but not as good as Region-Based Convolutional Neural Networks (R-CNNs), they are popular for object detection because of their detection speed, often demonstrated in real-time on video or with camera feed input.

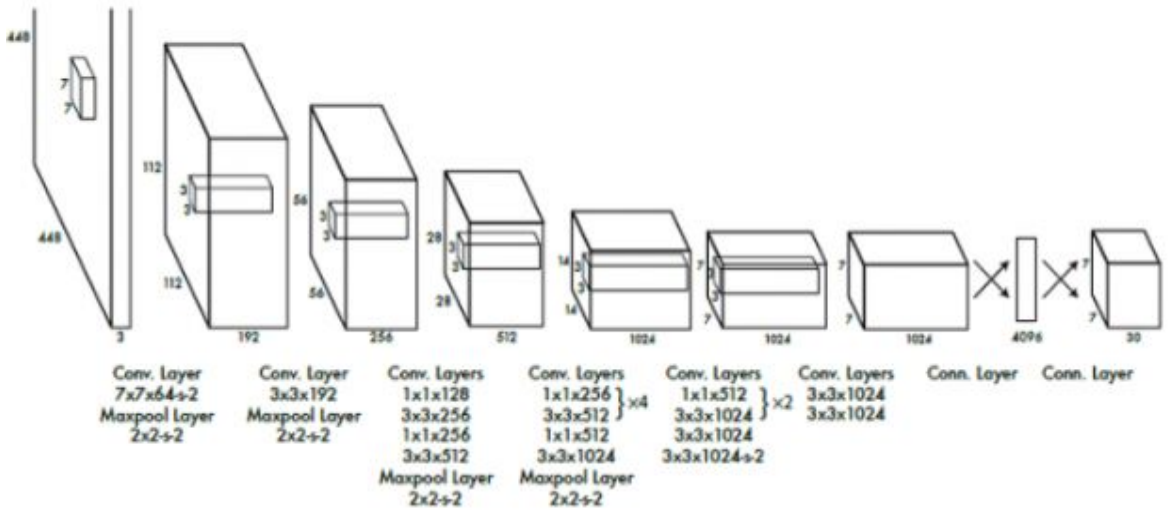
DIfferent versions of YOLO are present we will discuss each of them -:

2.3.1 YOLO v1

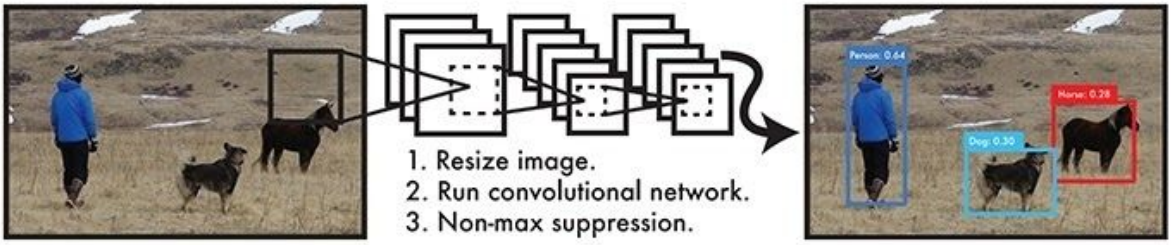
YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm “only looks once” at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes.

Further research was conducted resulting in the December 2016 paper “YOLO9000: Better, Faster, Stronger,” by Redmon and Farhadi, both from the University of Washington, that provided a number of improvements to the YOLO detection method including the detection of over 9,000 object categories by jointly optimizing detection and classification.

Even more recently, the same researchers wrote another paper in April 2018 on their progress with evolving YOLO even further, “YOLOv3: An Incremental Improvement”



You Only Look Once: Unified, Real-Time Object Detection (YOLO) .YOLO proposed a simple convolutional neural network approach which has both great results and high speed, allowing for the first time real time object detection.



Compared to other region proposal classification networks (fast RCNN) which perform detection on various region proposals and thus end up performing prediction multiple times for various regions in a image, Yolo architecture is more like FCNN (fully convolutional neural network) and passes the image (nxn) once through the FCNN and output is (mxm) prediction. This the architecture is splitting the input image in mxm grid and for each grid generation 2 bounding boxes and class probabilities for those bounding boxes. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This model has several benefits over traditional object detection model;

First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency.

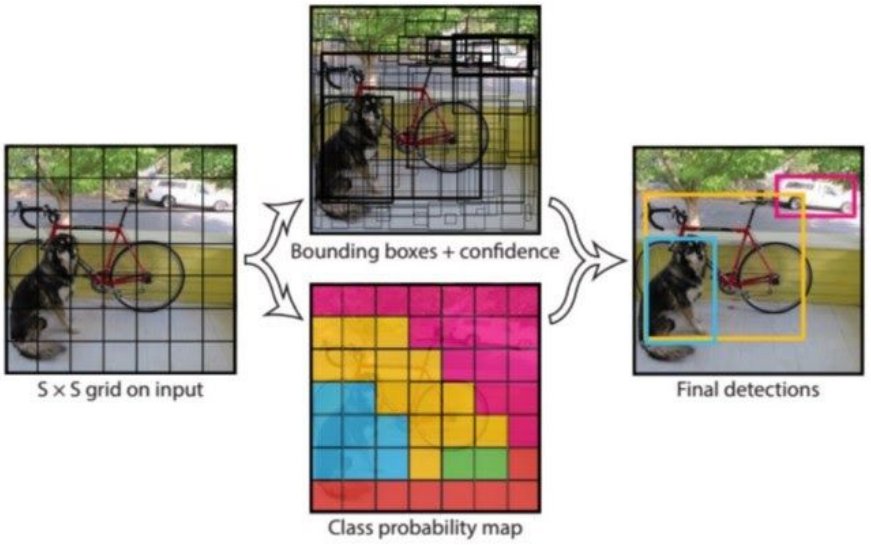
Second, YOLO reasons globally about the image when making predictions. YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. YOLO makes less than half the number of background errors compared to Fast R-CNN.

Third, YOLO learns representations of objects. When trained on natural images and tested on inputs, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is less likely to break down when applied to new domains or unexpected inputs.

YOLO network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the

image. The YOLO design enables end-to-end training and real time speeds while maintaining high average precision.

The YOLO system divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts i.e. if the model results as a car it will gave the confidence score as it resembles the car.Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.



2.3.2 YOLO v2

YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO.Multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster RCNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the

COCO detection dataset and the ImageNet classification dataset. Joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.

General purpose object detection should be fast, accurate, and able to recognize a wide variety of objects. Since the introduction of neural networks, detection frameworks have become increasingly fast and accurate. However, most detection methods are still constrained to a small set of objects. The most common detection datasets contain thousands to hundreds of thousands of images with dozens to hundreds of tags. Classification datasets have millions of images with tens or hundreds of thousands of categories.
Using this method we train YOLO9000, a real-time object detector that can detect over 9000 different object categories. First we improve upon the base YOLO detection system to produce YOLOv2, a state-of-the-art, real-time detector. YOLO's convolutional layers downsample the image by a factor of 32 so by using an input image of 416 we get an output feature map of 13 × 13. When we move to anchor boxes we also decouple the class prediction mechanism from the spatial location and instead predict class and objectness for every anchor box. Using anchor boxes we get a small decrease in accuracy. YOLO only predicts 98 boxes per image but with anchor boxes our model predicts more than a thousand. Without anchor boxes our intermediate model gets 69.5 mAP with a recall of 81%. With anchor boxes our model gets 69.2 mAP with a recall of 88%. Even though the mAP decreases, the increase in recall means that our model has more room to improve.

We want detection to be accurate but we also want it to be fast. Most applications for detection, like robotics or self driving cars, rely on low latency predictions. In order to maximize performance we design YOLOv2 to be fast from the ground up. Most detection frameworks rely on VGG-16 as the base feature extractor. VGG-16 is a powerful, accurate classification network but it is needlessly complex. The convolutional layers of VGG-16 require 30.69 billion floating point operations for a single pass over a single image at 224 × 224 resolution. The YOLO framework uses a custom network. This network is faster than VGG-16. However, it's accuracy is slightly worse than VGG16. For single-crop, top-5 accuracy at 224 × 224, YOLO's custom model gets 88.0%.

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Fig : Path from YOLO to YOLO v2

We introduce YOLOv2 and YOLO9000, real-time detection systems. YOLOv2 is state-of-the-art and faster than other detection systems across a variety of detection datasets. Furthermore, it can be run at a variety of image sizes to provide a smooth tradeoff between speed and accuracy

2.3.3 YOLO v3

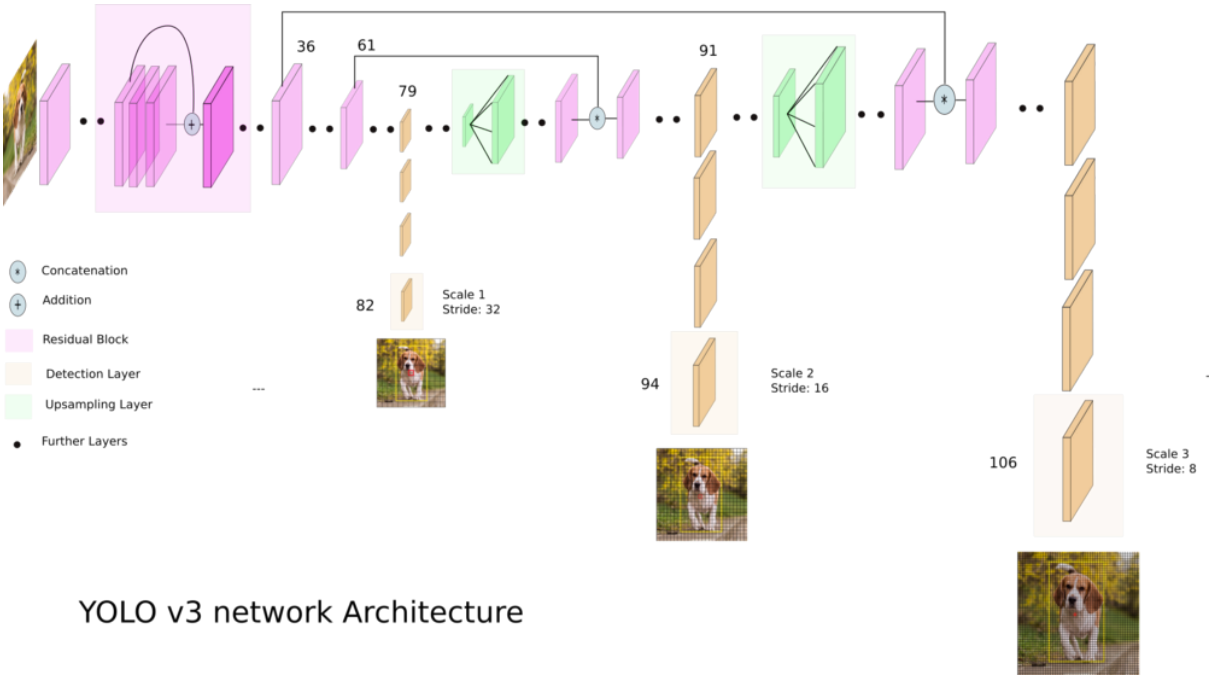
YOLO v3: Better, not Faster, Stronger
For it's time YOLO 9000 was the fastest, and also one of the most accurate algorithms.But that speed has been traded off for boosts in accuracy in YOLO v3. While the earlier variant ran on 45 FPS on a Titan X, the current version clocks about 30 FPS. This has to do with the increase in complexity of underlying architecture called Darknet.

Darknet-53

YOLO v2 used a custom deep architecture darknet-19, an originally 19-layer network supplemented with 11 more layers for object detection. With a 30-layer architecture, YOLO v2 often struggled with small object detections. This was attributed to loss of fine-grained features as the layers downsampled the input. To remedy this, YOLO v2 used an identity mapping, concatenating feature maps from a previous layer to capture low level features.
However, YOLO v2's architecture was still lacking some of the most important elements that are now staple in most state-of-the-art algorithms. No residual blocks, no skip connections and no upsampling. YOLO v3 incorporates all of these.

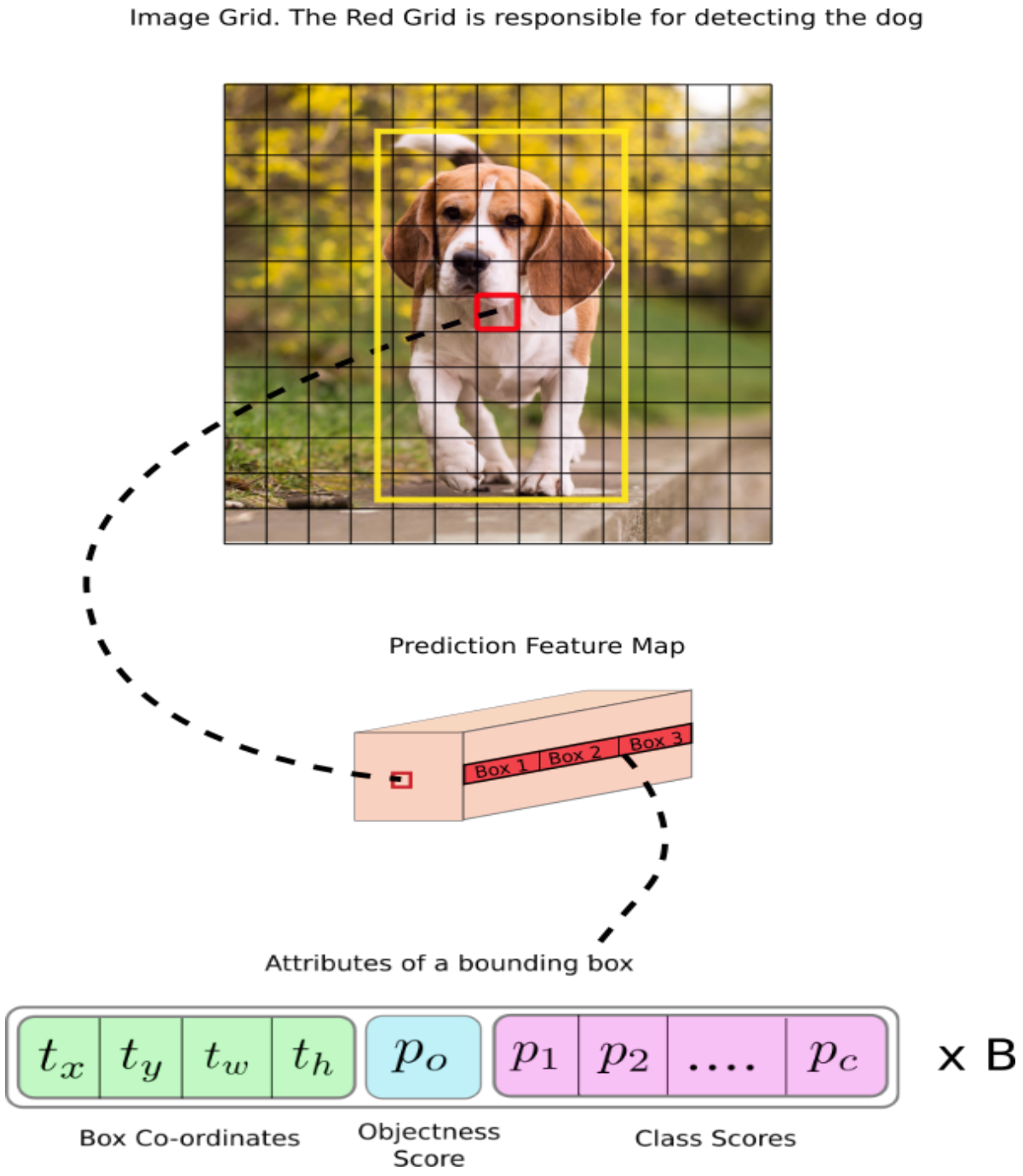
First, YOLO v3 uses a variant of Darknet, which originally has a 53 layer network trained on Imagenet. For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3. This is

the reason behind the slowness of YOLO v3 compared to YOLO v2. Here is how the architecture of YOLO now looks like.



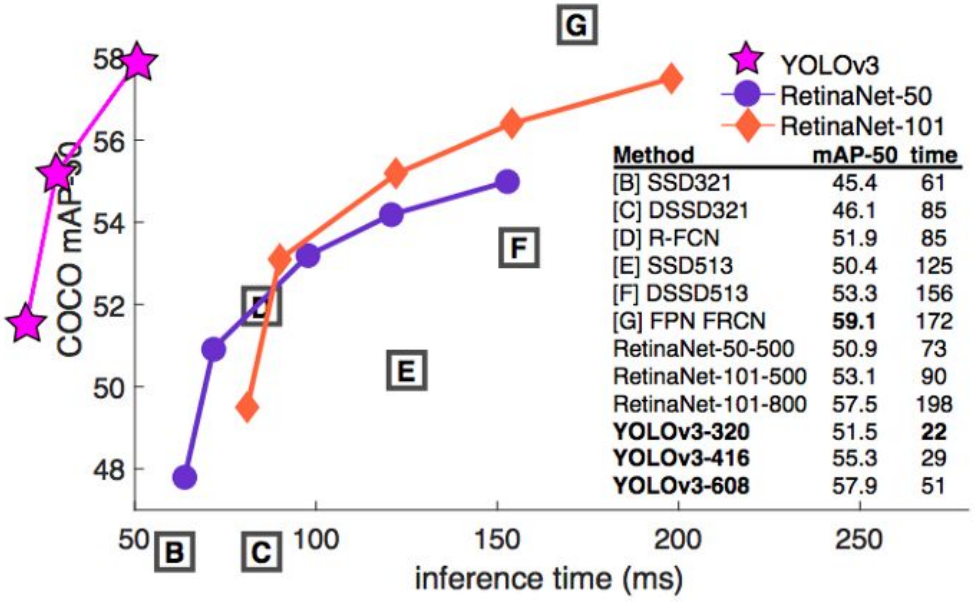
YOLO v3 network Architecture

Detection at three Scales-: The newer architecture boasts of residual skip connections, and upsampling. The most salient feature of v3 is that it makes detections at three different scales. YOLO is a fully convolutional network and its eventual output is generated by applying a 1 x 1 kernel on a feature map. In YOLO v3, the detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network.
The shape of the detection kernel is 1 x 1 x (B x (5 + C)). Here B is the number of bounding boxes a cell on the feature map can predict, “5” is for the 4 bounding box attributes and one object confidence, and C is the number of classes. In YOLO v3 trained on COCO, B = 3 and C = 80, so the kernel size is 1 x 1 x 255. The feature map produced by this kernel has identical height and width of the previous feature map, and has detection attributes along the depth as described above.



YOLO v3 makes predictions at three scales, which are precisely given by downsampling the dimensions of the input image by 32, 16 and 8 respectively.
The first detection is made by the 82nd layer. For the first 81 layers, the image is down sampled by the network, such that the 81st layer has a stride of 32. If we have an image of 416 x 416, the resultant feature map would be of size 13 x 13. One detection is made here using the 1 x 1 detection kernel, giving us a detection feature map of 13 x 13 x 255.
Then, the feature map from layer 79 is subjected to a few convolutional layers before being sampled by 2x to dimensions of 26 x 26. This feature map is then depth concatenated with the feature map from layer 61. Then the combined feature map is

again subjected to a few 1 x 1 convolutional layers to fuse the features from the earlier layer (61). Then, the second detection is made by the 94th layer, yielding a detection feature map of 26 x 26 x 255.
A similar procedure is followed again, where the feature map from layer 91 is subjected to few convolutional layers before being depth concatenated with a feature map from layer 36. Like before, a few 1 x 1 convolutional layers follow to fuse the information from the previous layer (36). We make the final of the 3 at 106th layer, yielding a feature map of size 52 x 52 x 255.
Better at detecting smaller objects-: Detections at different layers helps address the issue of detecting small objects, a frequent complaint with YOLO v2. The upsampled layers concatenated with the previous layers help preserve the fine grained features which help in detecting small objects. The 13 x 13 layer is responsible for detecting large objects, whereas the 52 x 52 layer detects the smaller objects, with the 26 x 26 layer detecting medium objects. Here is a comparative analysis of different objects picked in the same object by different layers.
More bounding boxes per image-: For an input image of the same size, YOLO v3 predicts more bounding boxes than YOLO v2. For instance, at its native resolution of 416 x 416, YOLO v2 predicted 13 x 13 x 5 = 845 boxes. At each grid cell, 5 boxes were detected using 5 anchors.
On the other hand YOLO v3 predicts boxes at 3 different scales. For the same image of 416 x 416, the number of predicted boxes are 10,647. This means that YOLO v3 predicts 10x the number of boxes predicted by YOLO v2. You could easily imagine why it's slower than YOLO v2. At each scale, every grid can predict 3 boxes using 3 anchors. Since there are three scales, the number of anchor boxes used in total are 9, 3 for each scale.
YOLO v3 performs at par with other state of art detectors like RetinaNet, while being considerably faster, at COCO mAP 50 benchmark. It is also better than SSD and it's variants



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Chapter 3

Implementation Of Object Detection Module

This chapter deals with the design and implementation of the object detection model. Section 3.1 tells about the implementation approach of the model, section 3.2 tells about the dataset and how its created, section 3.3 tells about the structure and architecture of the model, section 3.4 explains various steps of implementation and section 3.5 represents the results and output we receive.

3.1 Implementation Approach

This section covers the overall implementation approach of the model.

Implementation of the complete project is done on google platform, i.e Google drive and Google Colab. YOLOv3 is the object detection model used in this project. The algorithm is implemented in Python by using OpenCV.
YOLOv3 is the latest variant of a popular object detection algorithm YOLO – You Only Look Once. YOLO applies a single neural network on the whole image and this neural network divides the image into grid cells and produces probabilities for every region. YOLO predicts the number of bounding boxes that cover some region on the image and chooses the best ones according to the probabilities.
In this project, we'll make use of the darknet framework for making the model easy to implement and efficient to use.

3.2 Dataset

For the purpose of training and testing of the model, a database of images of required classes is created. There are images with car, bus, accident, bike, van and truck classes. Images of car, bus, bike, van and truck were downloaded using OIDv4 Toolkit on github downloading multiple classes at same time limiting upto 600 images of each class. There are about 600+ accident images combined with the downloaded dataset. Each image has an annotation file with the same name in yolo format, i.e. .txt format.

DATASET	NO. OF IMAGES
CAR	400
VAN	400
BUS	400
BIKE	400
TRUCK	400

First, We train our model by inputting the image of each class for example 400 of each. There are about 600+ accident images combined with the downloaded dataset. Each image has an annotation file with the same name in yolo format, i.e. .txt format, which contains labeling of the picture.

3.2.1 OIDv4 Toolkit

OIDv4 toolkit is an open source toolkit available on github. With the help of this toolkit, we can download huge amounts of images with less effort. In this project we used this toolkit to download a dataset of multiple classes, i.e. car, van, truck, bike and bus.

Installing OIDv4 Toolkit

OIDv4 Toolkit is easily available on github. We just need to implement a few steps to install the toolkit.

Step 1: Cloning Repository

Use this simple command in your terminal or Anaconda prompt:

```
git clone https://github.com/EscVM/OIDv4_ToolKit.git
```

Step 2: Go inside the directory by using command:

```
cd OIDv4_ToolKit
```

Then, run following command in Terminal:

```
pip install -r requirements.txt
```

Step 3: Verify

In order to verify installation, launch OIDv4 toolkit:

```
python main.py
```

For more details:

```
python main.py -h
```

Downloading Images from OIDv4 Toolkit

Firstly go to the OIDv4 Toolkit directory by using command:
`cd <path to OIDv4 Toolkit directory>`

To start downloading images, run the following command in the Terminal(or Anaconda prompt):
`python main.py downloader --classes Car_Van_Truck_Motorcycle_Bus --type_csv train -- multiclasses 1 --limit 600`

Arguments used:

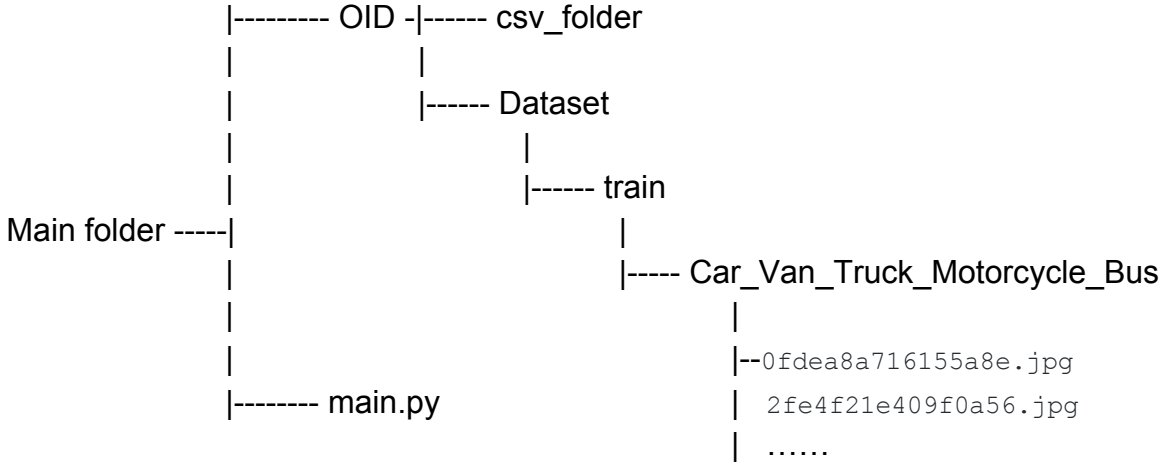
- **--classes Car_Van_Truck_Motorcycle_Bus**
  Name of the classes
- **--type_csv train**
  Specifies the type of dataset
- **--multiclasses 1**
  Specifies that all classes should be downloaded together in one folder
- **--limit 600**
  Specifies the number of images that will be downloaded for every class

Verify by Visualizer

Use the command to run visualizer:
`python main.py visualizer`

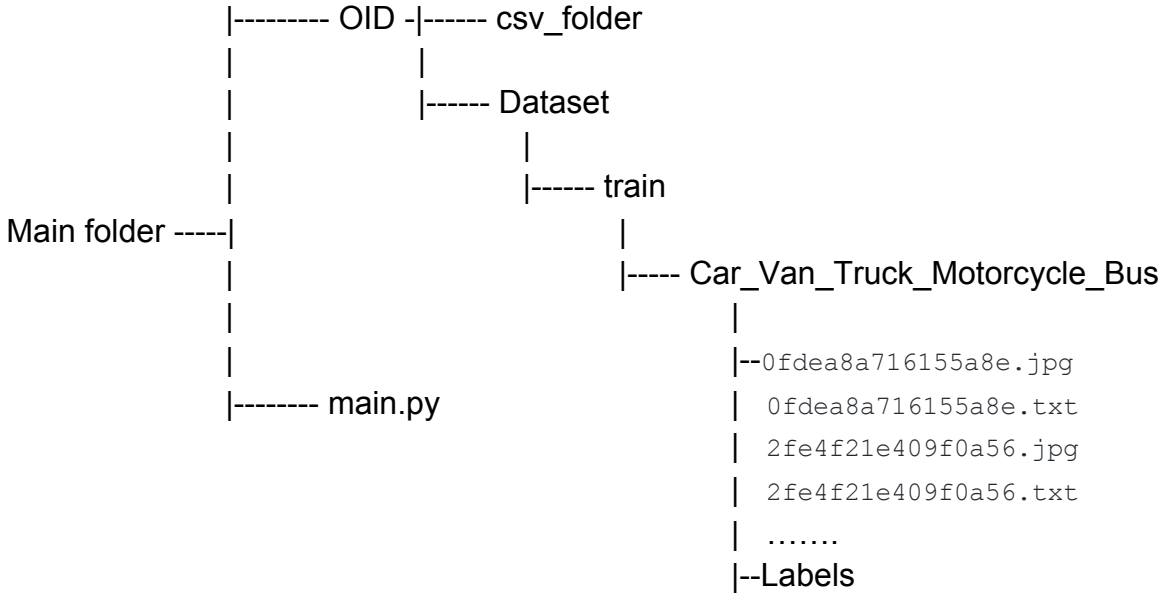
Use **d** and **a** to go next and previous between images
Use **q** to exit from visualizer

From the above process we get a dataset of images and csv file of annotations. Now our task is to convert the annotations from csv format to yolo format. For the purpose we use a python script that will read the csv file and create a .txt file of annotation for each image of the dataset.



|--Labels

After converting annotations,



Now, we have gathered dataset of 5 classes, i.e. Car, Van, Truck, Motorcycle and Van. For accident images we have to apply slightly more effort. We have to look across several websites to find images of accidents. Accidents are not as such a physical entity, we need to be specific about what we choose. In this project, we are considering only vehicle collision to be an accident.
Therefore, accident images in the dataset are of either of 2 vehicles colliding with each other or of a damaged vehicle like a damaged car or bus. There are 500+ images of accidents in the dataset. All these images are either downloaded from the web or extracted from video frames of Youtube accident videos.

Labelling of downloaded images is done using a labelling tool. There are variety of labelling tools available but we'll be using **LabelImg Tool** in this project.

LabelImg Tool:

LabelImg is an open source tool used for labelling images either in PASCAL VOC format(saved as XML files) or YOLO format(saved as .txt files). LabelImg is a graphical image annotation tool. It is written in python and easy to use.

Use of LabelImg tool in this project is to label accident images in YOLO format.

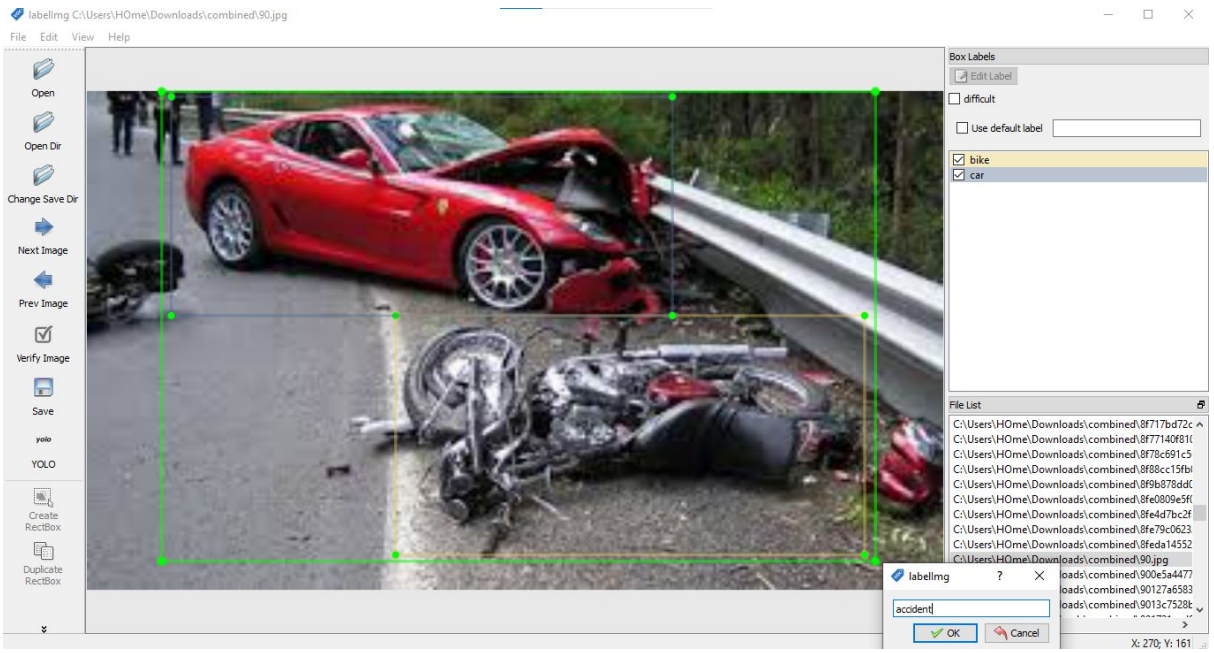


Fig. Labelling accident image using LabelImg Tool

Above image is an example of how we label an image using LabelImg. The annotation format on the left hand side clearly shows “YOLO”. On the right hand side, there is a list with all the classes labelled on the images.
Every time after drawing the RectBox, the tool asks for the name of the class. We can either give a new name or from the existing ones. After labelling the image, click on the save button so that a txt file with labelling information is created and saved in the saving directory. The same process is repeated for each image and once all the accident images are labelled, we are free to close the tool. When we go to the saving directory(or images directory), we can see that a new “classes.txt” file is created with names of classes.
Further, we have to combine the OIDv4 toolkit downloaded dataset and the dataset we created using LabelImg tool. To do so, we’ll make use of a python script that will read files from both the memory location and combine them in one folder. This script will take care of “classes.txt” in both the datasets and create a new “classes.txt” with all the 6 classes, i.e. car, van, truck, bike, bus and accident. The code will also be responsible for updating the annotation files corresponding to the new class numbers.

After doing all this, we get a combined dataset with all the images along with their annotation files and a classes.txt file containing all the 6 labels with each label in a new line.

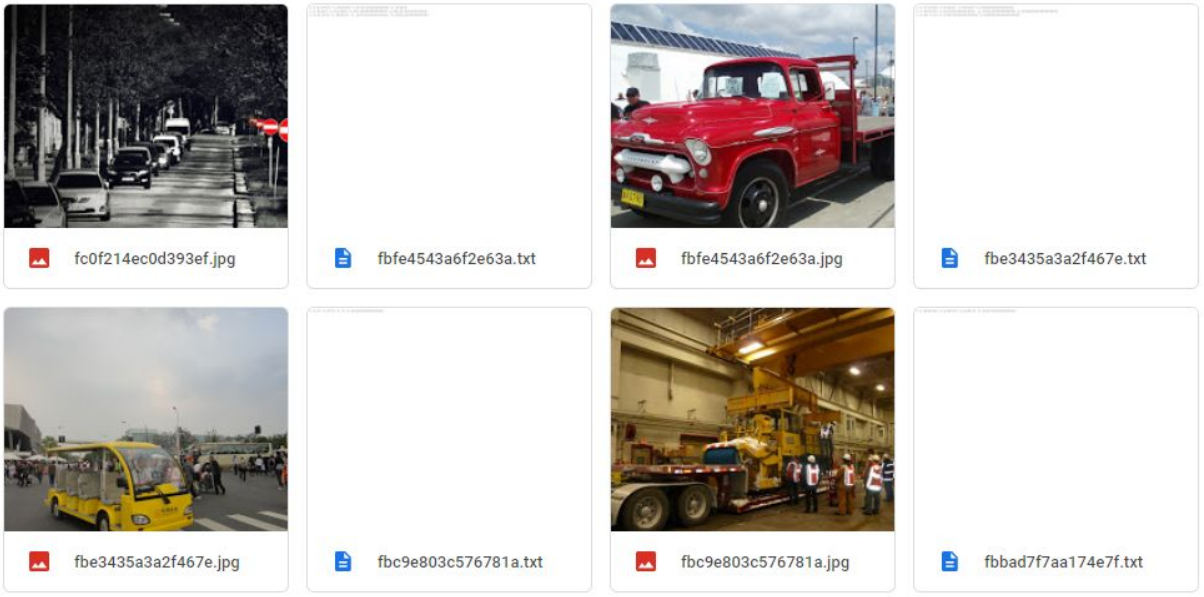


Fig. Dataset

Total number of images with annotations = 3345
Number of classes = 6 (car; van; truck; bike; bus; accident)

3.3 Model

This section deals with the architecture of the model and explains the ongoing processes in the model.

The YOLOv3 algorithm is an improvement on YOLOv1 and YOLOv2 because it has the advantages of high detection accuracy, accurate positioning, and fast speed. Especially when the multi-scale prediction methods are introduced, it can achieve the detection of small targets and has good robustness to environmental scenes, therefore, it has become a current research hotspot. The network structure of the YOLOv3 algorithm is shown in Figure below. The residual network is mainly used to upgrade the feature extraction network, and the basic backbone network is updated from Darknet-19 to Darknet-53 to extract features and obtain deeper feature information.

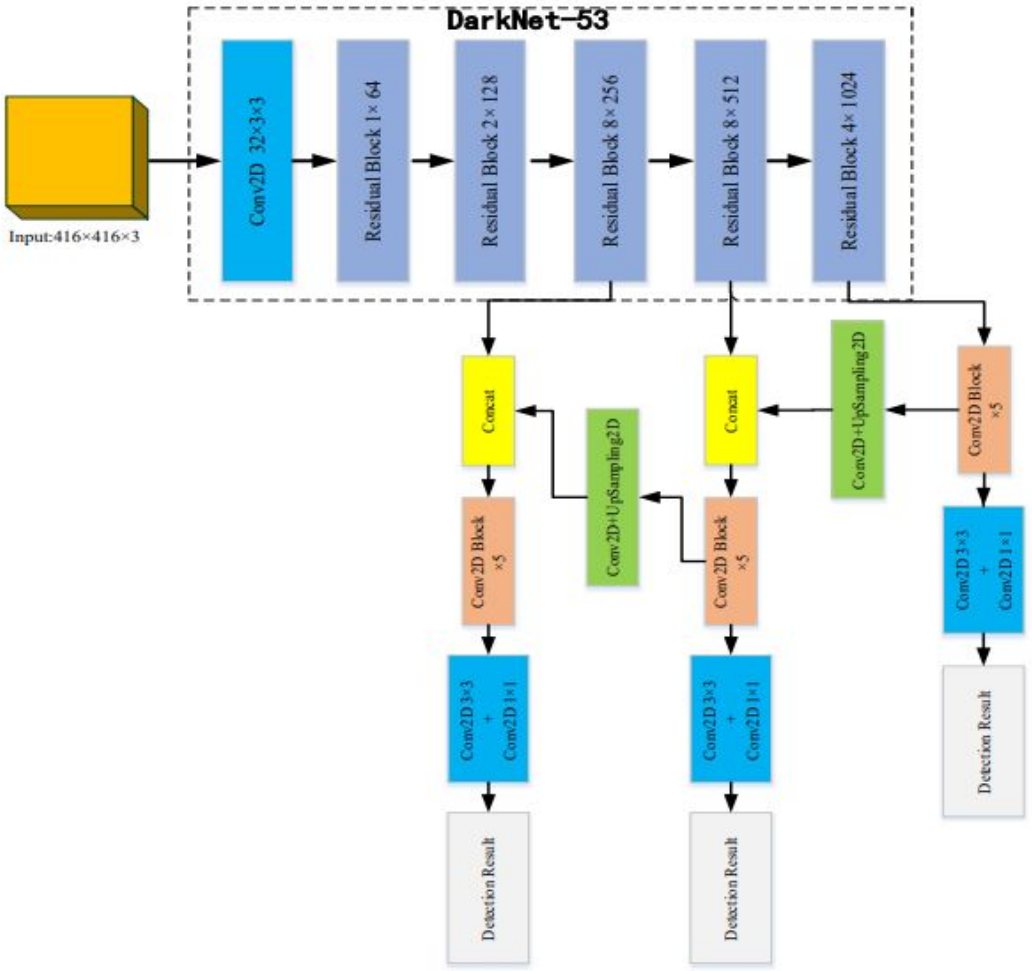


Fig. YOLOv3 network structure diagram

The Darknet-53 network in YOLOv3 uses a large number of 1 × 1 and 3 × 3 convolutional layers in order to connect local feature interactions and its function is equivalent to the global connection of the full feature layer and the addition of shortcut connections. This operation enables us to obtain more meaningful semantic information from up-sampled features and finer-grained information from earlier feature mappings. This feature extraction network has 53 convolutional layers, which is called Darknet-53, and its structure is shown at the top in Figure 2. In Darknet-53, 1 × 1 and 3 × 3 alternating convolution kernels are used, and after each layer of convolution, the BN layer is used for normalization. The Leaky Relu function is used as the activation function, the pooling layer is discarded, and the step size of the convolution kernel is enlarged to reduce the size of the feature map. As the network structure is deeper, its ability to extract features is also enhanced. The function of the sampling layer (upsample) is to generate the small-size images by interpolation of small-size feature maps and other methods. When short connections are set up between some layers to connect low-level features with high-level ones, the fine-grained information of high-level abstract features is enhanced, and they can be used for class prediction and bounding box regression.
The YOLOv3 network prediction process is listed below:

1. First, the images of size 416 × 416 are input into the Darknet-53 network. After performing many convolutions, a feature map of size 13 × 13 is obtained, and then 7 times by 1 × 1 and 3 × 3 convolution kernels are processed to realize the first class and regression bounding box prediction.
2. The feature map with size 13 × 13 is processed 5 times by 1 × 1 and 3 × 3 convolution kernels, and then the convolution operation is performed by using 1 × 1 convolution kernel, followed by 2 times the upsampling layer, and stitching to the size on the 26 × 26 feature map. The new feature map of size 26 × 26 is then processed 7 times using 1 × 1 and 3 × 3 convolution kernels to achieve the second category and regression bounding box prediction.
3. A new feature map has a size of 26 × 26. Firstly, we use 1 × 1 and 3 × 3 convolution kernels to process 5 times, perform a double upsampling operation, and stitch it onto the feature map of size 52 × 52. Then, the feature map is processed 7 times using 1 × 1 and 3 × 3 convolution kernels to achieve the third category and regression bounding box prediction.

It can be seen from the above results, that YOLOv3 can output three feature maps of different sizes at the same time, which are 13 × 13, 26 × 26, and 52 × 52. In this way, the feature maps of different sizes are optimized for the detections of small targets, but at the same time, the detections of large targets are weakened. Each feature map predicts three regression bounding boxes at each position, each bounding box contains a target confidence value, four coordinate values, and the probability of C different edges. There are (52 × 52 + 26 × 26 + 13 × 13) × 3 = 10647 regression bounding boxes.

3.4 Object Detection Process

This section explains various steps involved in the detection process.

Training YOLOv3 model on Google Colab involves following steps:

- Dataset Preparation
- Set up Colab environment
- Installing Darknet
- Modify configuration files
- Split train/test dataset
- Training the model
- Predict with YOLOv3 using OpenCV

3.4.1 Dataset Preparation

Dataset is the same combined dataset mentioned above.

The dataset required to train a detector with YOLOv3 contains 2 components: images and labels. Each image will be associated with a label file (normally a txt file) which defines the object class and coordinates of object in the image following this syntax: <object-class> <x_center> <y_center> <width> <height>

3.4.2 Set Up Colab Environment

Configure Runtime to work with GPU -
We want to use the 12GB-RAM GPU hardware acceleration!
Go to > Menu > Runtime > Configure Runtime Type And select GPU From the Hardware accelerator drop down menu

Connect files from Google drive -

Upload the dataset onto your google drive and connect the Colab notebook to google drive by using the below script:

```
from google.colab import drive
drive.mount('/content/drive')
```

3.4.3 Installing Darknet

Darknet is an open source neural network framework easily available on github. We'll be installing darknet in google drive itself

Go to desired location using command "%cd <path>" and run the script below to download darknet folder -

```
!git clone https://github.com/pjreddie/darknet
%cd darknet
```

To compile darknet -

```
!make
```

3.4.4 Modifying Configuration files

In directory darknet\cfg, creating 2 copies of "yolov3.cfg" in the same folder and renaming it to "yolov3_accident_train.cfg" and "yolov3_accident_test.cfg.

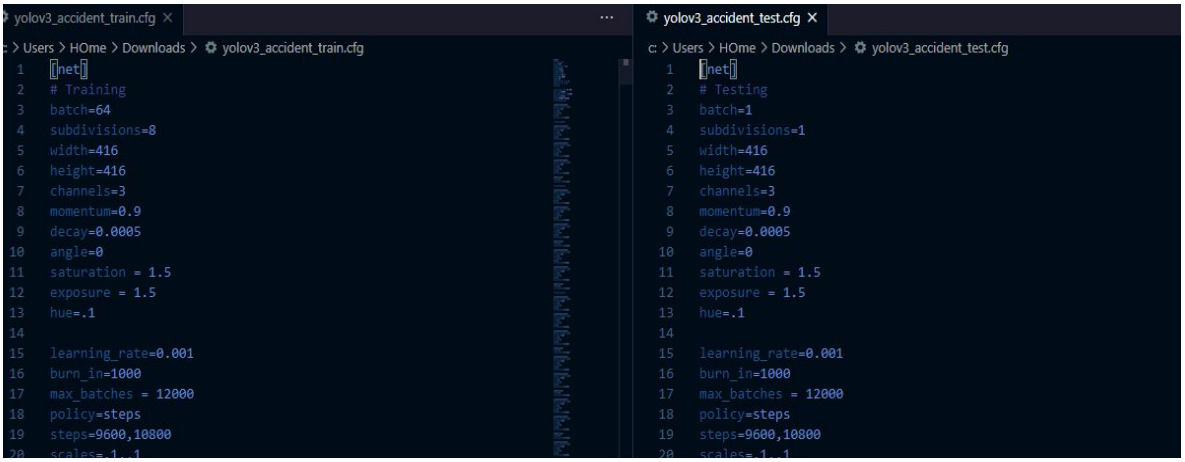


Fig. Modified configuration files

In here we apply this formula: (filters = (classes + 5)*3) to calculate the number of filters. For example, if you need to detect 6 objects, then your filter will be: filters=(6+5)*3 = 33 and classes = 6.

```
769  [convolutional]
770  size=1
771  stride=1
772  pad=1
773  filters=33
774  activation=linear
775
776
777  [yolo]
778  mask = 0,1,2
779  anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90
780  classes=6
781  num=9
782  jitter=.3
783  ignore_thresh = .7
784  truth_thresh = 1
785  random=1
```

Fig. Filters and classes in configuration file

3.4.5 Split train/test dataset

To avoid over-fitting and achieved an objective evaluation regarding our model, we need to split our dataset into training set and testing set.
Using Python script we read all file paths and split them in a ratio that 85% of the paths are added to train.txt and remaining 15% are added to test.txt. Both these files are stored in the dataset folder.

In the next step, we need to create a file “classes.names” in the dataset directory, with object names as each new line. In our case, we have

```
truck
accident
van
car
bike
Bus
```

Finally, create file “joined_data.data” in the directory darknet\cfg, containing classes = 6

```
train = /gdrive/MyDrive/combined/train.txt
valid = /gdrive/MyDrive/combined/test.txt
names = /gdrive/MyDrive/combined/classes.names
backup = backup
```

Here, “combined” is the name of the dataset directory.

3.4.6 Training the model

To train the model, we need to use the script below:

```
%cd /content/gdrive/MyDrive/darknet
!./darknet detector train cfg/joined_data.data cfg/yolov3_accident_train.cfg
darknet53.conv.74
```

To retrain yolo model with saved weight we use the script below:

```
!./darknet detector train cfg/joined_data.data cfg/yolov3_accident_train.cfg
backup/yolov3_accident_train_last.weights
```

3.4.7 Predict with YOLOv3 using OpenCV

In Google Colab, we can predict on images as well as videos.
For images we use :

```
!./darknet detector test cfg/joined_data.data cfg/yolov3_accident_test.cfg
backup/yolov3_accident_train_last.weights data/accident.jpg -dont_show
```

For videos we use :

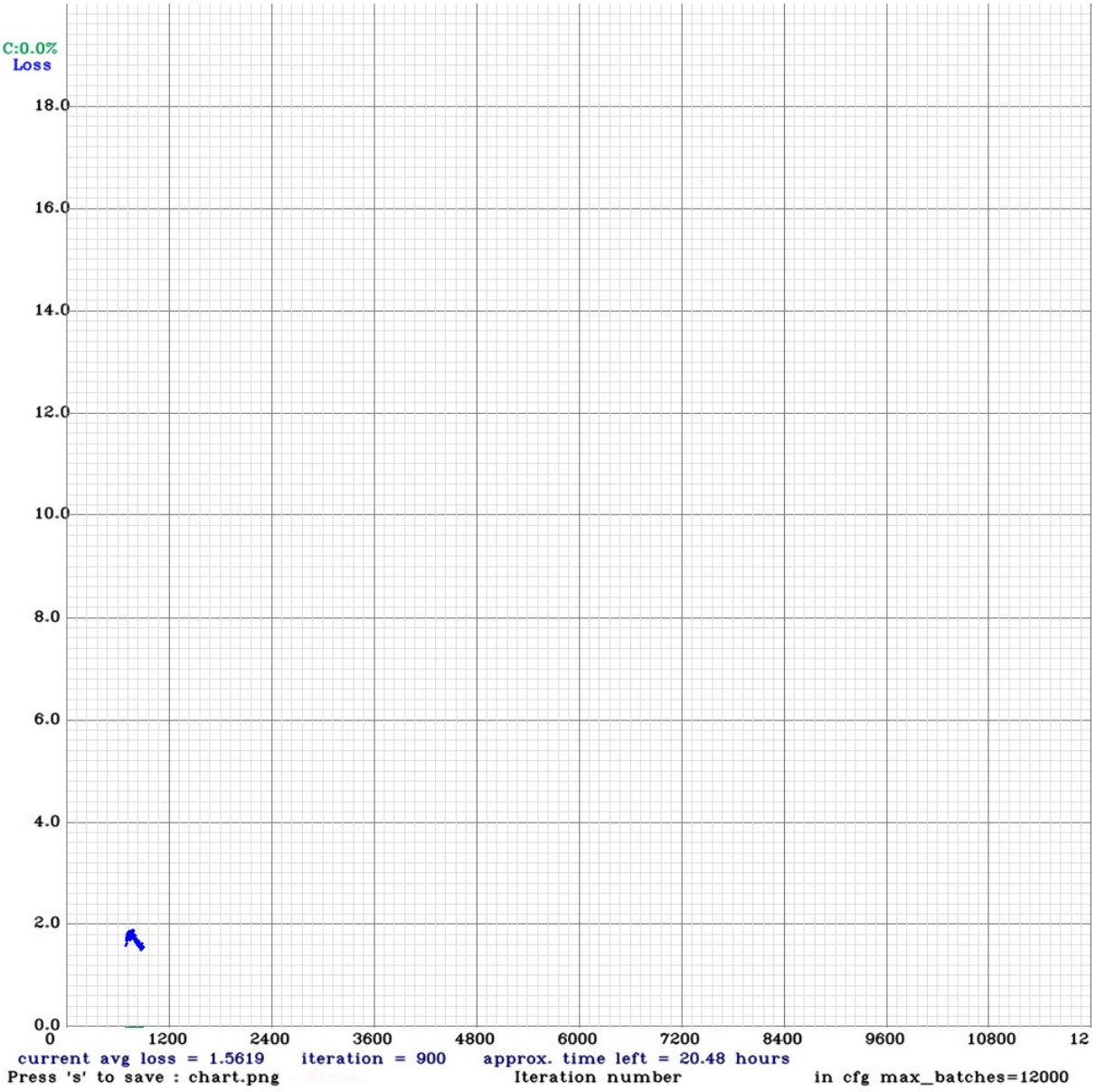
```
!./darknet detector demo cfg/joined_data.data cfg/yolov3_accident_test.cfg
backup/yolov3_accident_train_last.weights data/test-video.mp4 -out_filename
result.avi -dont_show
```

Result files are stored in the darknet directory.

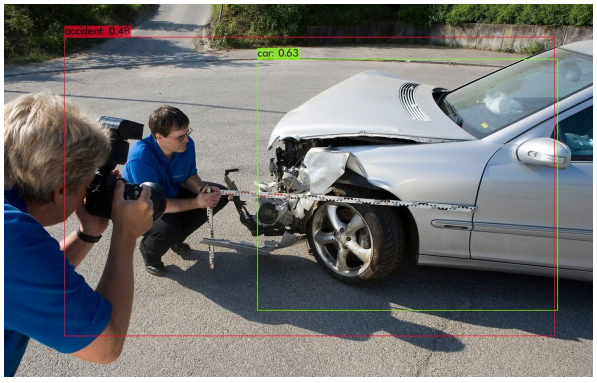
3.5 Results and Output

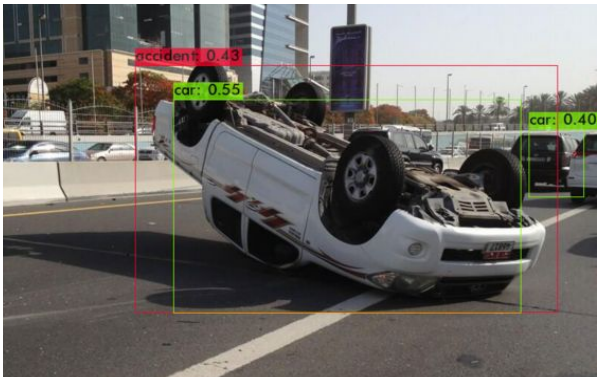
Result

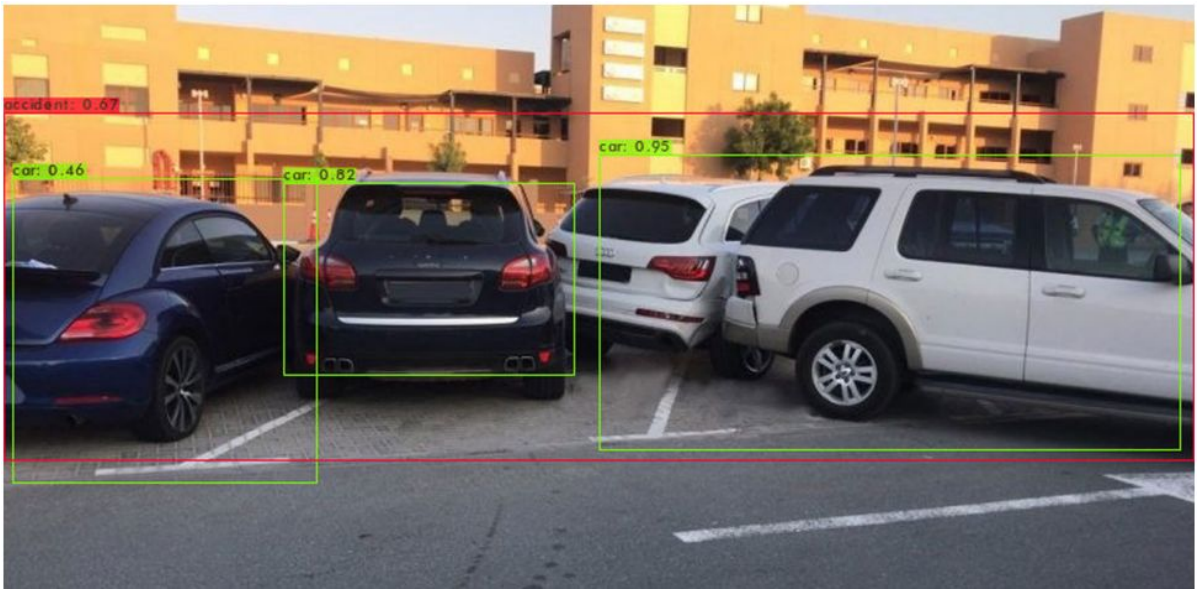
The Model predicts the accident correctly but with a low probability. We can improve this by increasing the number of images in the dataset and by increasing the epochs. As the graph shows as we increase the iteration number loss decreases and our results are better. Basically, more time the model read and gets training more reliable it performs. As shown in the below picture it shows labeling in rectangle along with confidence level according to which our model is train.



Output


accident: 0.48
car: 0.63


accident: 0.43
car: 0.55
car: 0.40


accident: 0.67
car: 0.46
car: 0.82
car: 0.95

Chapter 5
Conclusion

We made our project to detect the accident site by classifying vehicles and detecting whether an accident took place or not. We fed cctv input and the model is trained jointly firstly to extract photos from the videos and then, YOLO, a model for object detection.It can be trained directly on full images.YOLO is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly. Fast YOLO is the fastest general-purpose object detector in the literature and YOLO pushes the state-of-the-art in real-time object detection. YOLO also generalizes well to new domains making it ideal for applications that rely on fast, robust object detection. We connect YOLO to cctv webcams and it maintains its performance very well. It should be noted that object detection has not been used much in many areas where it could be of great help. As mobile robots, and in general autonomous machines, are starting to be more widely deployed (e.g., quad-copters, drones and soon service robots), the need of object detection systems is gaining more importance.we need to consider that we will need object detection systems for nano-robots or for robots that will explore areas that have not been seen by humans, such as depth parts of the sea or other planets, and the detection systems will have to learn to new object classes as they are encountered. In such cases, a real-time open-world learning ability will be critical.

REFERENCES

1. OpenCV: http://opencv.org/opencv-3-1.html.
2. Darknet https://github.com/pjreddie/darknet
3. Labelimg https://github.com/tzutalin/labelImg
4. https://www.udemy.com/share/102CsN/
5. https://www.frontiersin.org/articles/10.3389/frobt.2015.00029/full#h7
6. https://pjreddie.com/darknet/yolo/
7. https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006
8. https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/
9. https://github.com/EscVM/OIDv4_ToolKit – official resource of OIDv4 Toolkit
10. https://pjreddie.com/media/files/papers/yolo_1.pdf
11. https://pjreddie.com/media/files/papers/YOLOv3.pdf
12. https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras