OBJECT DETECTION USING SOFTWARE

**A MINI PROJECT REPORT**

***Submitted by***

**Dhinesh Raj M (212222060048)**

**Ganapathy K (212222060056)**

**Gladsin Paul J (212222060061)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**SAVEETHA ENGINEERING COLLEGE(AUTONOMOUS),**

**CHENNAI,**

**ANNA UNIVERSITY, CHENNAI 600 025**

**NOVEMBER –2023**

**SAVEETHA ENGINEERING COLLEGE (AUTONOMOUS),CHENNAI**

ANNA UNIVERSITY, CHENNAI – 600 025

**BONAFIDE CERTIFICATE**

Certified that this project report **“OBJECT DETECTION USING SOFTWARE ”** is the bonafide work of **“Dhinesh Raj M, Ganapathy K, Gladsin Paul J”** who carried out the project work under my/our supervision.

**SIGNATURE SIGNATURE**

Dr. C. Sheeba Joice, M.E.,M.B.A.,Ph.D. DR.AmitKumar Varshney

M.E.,Ph.D.

**HEAD OF THE DEPARTMENT SUPERVISOR**

Department of Electronics and Department of Electronics and

communication engineering, communication engineering,

Saveetha Nagar, Saveetha Nagar,

Thandalam , Thandalam,

Chennai- 602 105 Chennai- 602 105

Submitted for the project viva-voce examination held on \_

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**MINI PROJECT APPROVAL SHEET**

The Mini project sheet of **“OBJECT DETECTION USING SOFTWARE”** is submitted by ‘Dhinesh Raj M(212222060048), Ganapathy K (212222060056), Gladsin Paul J (212222060061)’ is approved for submission, as partial requirement for the award of the Degree of Bachelor of Engineering in Electronics and Communication, Anna University during the academic year 2023- 2024.

Submitted for the ‘University Mini Project Viva Voce’ examination held on

.

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ABSTRACT**

Embarking on a journey into the realm of object detection, this project harnesses the power of deep learning to construct and assess object detection models. The project begins by reviewing the state-of-the-art object detection algorithms, including two-stage detectors like R-CNN and Faster-RCNN, and single-stage detectors like SSD and YOLO. These algorithms are evaluated on benchmark datasets, such as COCO and PASCAL VOC, to assess their performance in terms of accuracy and speed It meticulously examines the influence of various factors on performance, explores the potential of transfer learning, and confronts the challenges of real-time object detection. By placing a strong emphasis on experimental evaluation and comparison, this project sheds light on the intricacies of designing, implementing, and evaluating high-performance object detection systems using deep learning techniques. This project also explores the application of transfer learning in object detection. Pre-trained models on large-scale datasets, such as ImageNet, are fine-tuned on specific object detection tasks to improve performance. The effectiveness of transfer learning is evaluated for different object detection algorithms and datasets.

**ACKNOWLEDGEMENT**

We convey our sincere thanks to **Dr.N.M.Veeraiyan -** President(SMET) and Chancellor-SIMATS, Saveetha Amaravathi University, **Dr.S.Rajesh,** Director- Saveetha Engineering College and **Dr. V. Saveetha Rajesh –** Director, Saveetha Medical College and Hospital for providing us with the facilities for the completion of our project. We are grateful to our Principal, **Dr.N.Duraipandian** for his continuous support and encouragement in carrying out our project work. We are deeply indebted to our beloved Head of the Department, **Dr.Sheeba Joice. C, M.E.,M.B.A., Ph.D.,** Department of Electronics and Communication, for giving us the opportunity to display our professional skills through this project.

We are greatly thankful to our Mini Project Coordinator, **Mr. M. Praveen** and our Mini Project Guide, **Dr. P. Baraneedharan** for their valuable guidance and motivation which helped to complete our project on time.

We thank all our teaching and non- teaching faculty members of the Department of Electronics and Communication for their passionate support, for helping us to identify our mistakes and for the appreciation they gave us. We heartily thank our library staff and the management for their extensive support in providing the resources and information that helped us to complete the project successfully. Also, we would like to record our deepest gratitude to our parents for their constant encouragement and support, which motivated us a lot to complete our project work.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | **4** |
|  | **ACKNOWLEDGEMENT** | **5** |
|  | **LIST OF TABLES** | **8** |
|  | **LIST OF FIGURES LIST OF SYMBOLS** | **8**  **-** |
| **1.** | | **INTRODUCTION** |

1.1 Background 11

1.2 Methodology 12

1.2.1 SqueezeNet 12

1.3 Summary 14

**2. LITERATURE SURVEY**

2.1 Object Detection 15

2.1.1 Shape Bases 16

2.1.2 Motion Based 16

2.1.3 Color Based 16

2.1.4 Texture Bases 16

2.2 Background Subtraction 17

2.3 Template Matching 18

**3. PROPOSED METHOD**

3.1 ResNet 19

3.2 R-CNN 19

3.2.1 Problems with R-CNN 21

3.3 Fast R-CNN 21

3.4 YOLO 23

3.5 SSD 24

**4. RESULT**

4.1 Input & Output 26

4.2 Detection Speed 28

4.3 Summary 29

**5. CONCLUSION**

5.1 Conclusion 30

5.2 Application 31

5.2.1 Face Detection 31

5.2.2 Counting Objects 31

5.2.3 Vehicle Detection 31

5.2.4 Industries 32

5.2.5 Security 32

5.2.6 Biometric recognition 32

5.2.7 Survelliance 32

5.2.8 Medical anlaysis 32

**REFERENCE**

**PROGRAM**

|  |  |  |
| --- | --- | --- |
|  | **LIST OF FIGURES** |  |
| **FIGURE NO.** | **TITLE** | **PAGE NO.** |

1. **Object detection background diagram 12**
2. **Background subtraction 17**
3. **Template matching 18**
4. **R-CNN Regions 20**
5. **Fast R-CNN Diagram 21**
6. **R-CNN timing bar graph 22**
7. **YOLO Diagram 23**
8. **SSD Diagram 24**
9. **SSD object detection 25**
10. **SSD Output 25**
11. **Input 26**
12. **Output 27**

**LIST OF SYMBOLS AND ABBREVIATION**

**CNN - Convolution Neural Network**

**DNN - Deep Neural Network**

**VGG - Visual Geometry Group**

**DSSD - Deconvolution Single Shot Detector**

**MLP - Multilayer perceptrons**

**YOLO - You Only Look Once**

**R-CNN - Region Based Convolution Neural Network**

**SSD - Single Shot Detector**

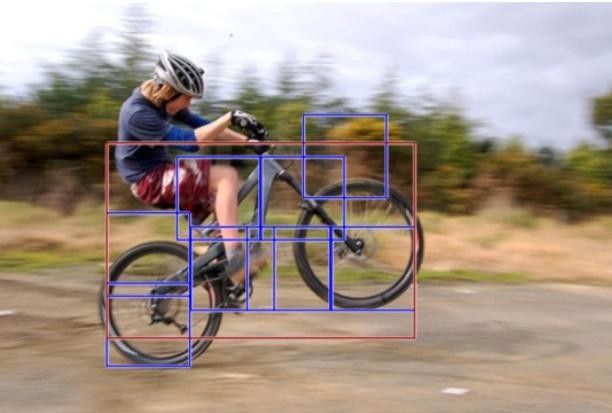
**CHAPTER 1**

# INTRODUCTION

A few years ago, the creation of the software and hardware image processing systems was mainly limited to the development of the user interface, which most of the programmers of each firm were engaged in. The situation has been significantly changed with the advent of the Windows operating system when the majority of the developers switched to solving the problems of image processing itself. However, this has not yet led to the cardinal progress in solving typical tasks of recognizing faces, car numbers, road signs, analyzing remote and medical images, etc. Each of these "eternal" problems is solved by trial and error by the efforts of numerous groups of the engineers and scientists. As modern technical solutions are turn out to be excessively expensive, the task of automating the creation of the software tools for solving intellectual problems is formulated and intensively solved abroad. In the field of image processing, the required tool kit should be supporting the analysis and recognition of images of previously unknown content and ensure the effective development of applications by ordinary programmers. Just as the Windows toolkit supports the creation of interfaces for solving various applied problems. Object recognition refers to a collection of related tasks for identifying objects in digital photographs. Region-based Convolutional Neural Networks, or R-CNNs, is a family of techniques for addressing object localization and recognition tasks, designed for model performance.

* 1. **Background**

The aim of object detection is to detect all instances of objects from a known class, such as people, cars or faces in an image. Generally, 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 need to somehow be explored. Each detection of the image is reported with some form of pose information. This is as simple as the location of the object, a location and scale, or the extent of the object defined in terms of a bounding box. In some other situations, the pose information is more detailed and contains the parameters of a linear or non-linear transformation. For example for face detection in a face detector may compute the locations of the eyes, nose and mouth, in addition to the bounding box of the face. An example of a bicycle detection in an image that specifies the locations of certain parts is shown in Figure 1. The pose can also be defined by a three-dimensional transformation specifying the location of the object relative to the camera. Object detection systems always construct a model for an object class from a set of training examples. In the case of a fixed rigid object in an image, only one example may be needed, but more generally multiple training examples are necessary to capture certain aspects of class variability. Convolutional implementation of the sliding windows Before we discuss the implementation of the sliding window using convents, let us analyze how we can convert the fully connected layers of the network into convolutional layers.



**Figure 1**

* 1. **Methodology**

This methodology ensures the development of robust and accurate object detection software. SqueezeNet's methodology revolves around designing a lightweight architecture with an emphasis on parameter efficiency, enabling its use in resource-constrained environments without sacrificing performance.

* + 1. **SqueezeNet**

SqueezeNet is name of a DNN for computer vision. SqueezeNet is developed by researchers at DeepScale, University of California, Berkeley, and Stanford University together. In SqueezeNet design, the authors goal is to create a smaller neural network with few parameters that can more easily fit into memory of computer and can more easily be transmitted over a computer network. SqueezeNet is originally released in 2016. This original version of SqueezeNet was implemented on top of the Caffe deep learning software framework.

The open-source research community ported SqueezeNet to a number of other deep learning frameworks. And is released in additions,in 2016, Eddie Bell released a part of SqueezeNet for the Chainer deep learning framework. in 2016, Guo Haria released a part of SqueezeNet for the Apache MXNet framework. 2016, Tammy Yang released a port of SqueezeNet for the Keras framework. In 2017, companies including Baidu, Xilnx, Imagination Technologies, and Synopsys demostrated SqueezedNet running on low-power processing platforms such as smartphones, FPGAs, and custom processors

SqueezeNet ships as part of the source code of a number of deep learning frameworks such as PyTorch, Apache MXNet, and Apple CoreML.In addition, 3rd party developers have created implementation of SqueezeNet that are compatible with frameworks such as TensorFlow. Below is summary of frameworks that support SqueezeNet.

SqueezeNet, a pioneering neural network architecture, is meticulously crafted for efficiency and compactness. At its core are "Fire Modules" incorporating squeeze and expand layers, intelligently capturing both local and global features. Leveraging 1x1 convolutions as bottlenecks allows for model depth without inflating parameters. The paradigm shift to global average pooling replaces fully connected layers, curbing overfitting and further reducing complexity.

SqueezeNet's paramount focus on parameter efficiency renders it adept for applications with constrained computational resources, offering a lightweight yet potent solution for real-time scenarios.

.

**1.3 Summary**

Object detection software is a crucial component in computer vision, enabling machines to identify and locate objects within images

or videos. Leveraging deep learning algorithms, these programs analyze visual data, recognizing patterns and shapes to detect and classify various objects. Popular frameworks like YOLO (You Only Look Once) and Faster R-CNN have gained prominence for their accuracy and efficiency. These tools find applications in diverse fields, from autonomous vehicles and surveillance systems to medical imaging and retail. The continuous evolution of object detection software contributes significantly to advancing artificial intelligence, enhancing automation, and improving real-world problem-solving capabilities.

**CHAPTER 2**

**LITERATURE SURVEY**

In various fields, there is a necessity to detect the target object and also track them effectively while handling occlusions and other included complexities. Many researchers (Almeida and Guting 2004, Hsiao-Ping Tsai 2011, Nicolas Papadakis and Aure lie Bugeau 2010 ) attempted for various approaches in object tracking. The nature of the techniques largely depends on the application domain. Some of the research works which made the evolution to proposed work in the field of object tracking are depicted as follows.

**2.1. OBJECT DETECTION**

Object detection is an important task, yet challenging vision task. It is a critical part of many applications such as image search, image auto-annotation and scene understanding, object tracking. Moving object tracking of video image sequences was one of the most important subjects in computer vision. It had already been applied in many computer vision fields, such as smart video surveillance (Arun Hampapur 2005), artificial intelligence, military guidance, safety detection and robot navigation, medical and biological application. In recent years, a number of successful single-object tracking system appeared, but in the presence of several objects, object detection becomes difficult and when objects are fully or partially occluded, they are obtruded from the human vision which further increases the problem of detection.. The proposed MLP based object tracking system is made robust by an optimum selection of unique features and also by implementing the Adaboost strong classification method.

**2.1.1 Shape based**

A mixture of image-based and scene-based object parameters such as image blob (binary large object) area, the aspect ratio of blob bounding box and camera zoom is given as input to this detection system. Classification is performed on the basis of the blob at each and every frame. The results are kept in the histogram.

**2.1.2 Motion Based**

When a simple image is given as an input with no objects in motion, this classification is not needed. In general, non-rigid articulated human motion shows a periodic property, hence this has been used as a strong clue for classification of moving objects. Based on this useful clue, human motion can be distinguished from other objects motion.

**2.1.3 Color Based**

Though color is not an appropriate measure alone for detecting and tracking objects, but the low computational cost of the color based algorithms makes the color a very good feature to be exploited. For example, the color-histogrambased technique is used for detection of vehicles in real-time. Color histogram describes the color distribution in a given region, which is robust against partial occlusions.

**2.1.4 Texture-Based**

The texture-based approaches with the help of texture pattern recognition work similar to motion-based approaches. It provides better accuracy, by using overlapping local contrast normalization but may require more time, which can be improved using some fast techniques.

**2.2. Background Subtraction**

The background subtraction method by Horprasert et al (1999), was able to cope with local illumination changes, such as shadows and highlights, even globe illumination changes. In this method, the background model was statistically modelled on each pixel. Computational colour mode, include the brightness distortion and the chromaticity distortion which was used to distinguish shading background from the ordinary background or moving foreground objects. The background and foreground subtraction method used the following approach. A pixel was modelled by a 4-tuple [Ei, si, ai, bi], where Ei- a vector with expected colour value, si - a vector with the standard deviation of colour value, ai - the variation of the brightness distortion and bi was the variation of the chromaticity distortion of the ith pixel. In the next step, the difference between the background image and the current image was evaluated



**Figure 2**

**2.3. Template Matching**

Template Matching is the technique of finding small parts of an image which match a template image. It slides the template from the top left to the bottom right of the image and compares for the best match with the template. The template dimension should be equal to the reference image or smaller than the reference image. It recognizes the segment with the highest correlation as the target. Given an image S and an image T, where the dimension of S was both larger than T, output whether S contains a subset image I where I and T are suitably similar in pattern and if such I exists, output the location of I in S as in Hager and Bellhumear (1998). Schweitzer et al (2011), derived an algorithm which used both upper and lowers bound to detect ‘k’ best matches. Euclidean distance and Walsh transform kernels are used to calculate match measure. The positive things included the usage of priority queue improved quality of decision as to which bound-improved and when good matches exist inherent cost was dominant and it improved performance. But there were constraints like the absence of good matches that lead to queue cost and the arithmetic operation cost was higher.



**Figure**

# CHAPTER 3

**PROPOSED METHOD**

# (Design and Implementation)

**3.1 ResNet**

To train the network model in a more effective manner, we herein adopt the same strategy as that used for DSSD (the performance of the residual network is better than that of the VGG network). The goal is to improve accuracy. However, the first implemented for the modification was the replacement of the VGG network which is used in the original SSD with ResNet. We will also add a series of convolution feature layers at the end of the underlying network. These feature layers will gradually be reduced in size that allowed prediction of the detection results on multiple scales. When the input size is given as 300 and 320, although the ResNet–101 layer is deeper than the VGG–16 layer, it is experimentally known that it replaces the SSD’s underlying convolution network with a residual network, and it does not improve its accuracy but rather decreases it.

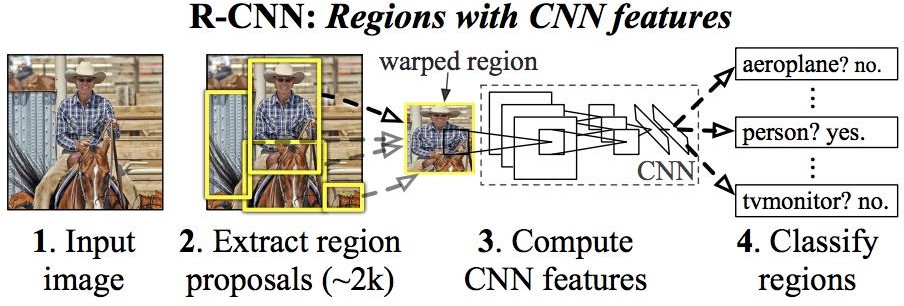
**3.2 R-CNN**

To circumvent the problem of selecting a huge number of regions, Ross Girshick et al. proposed a method where we use the selective search for extract just 2000 regions from the image and he called them region proposals. Therefore, instead of trying to classify the huge number of regions, you can just work with 2000 regions.

These 2000 region proposals are generated by using the selective search algorithm which is written below.

Selective Search:

1. Generate the initial sub-segmentation, we generate many candidate regions
2. Use the greedy algorithm to recursively combine similar regions into larger ones
3. Use generated regions to produce the final candidate region proposals

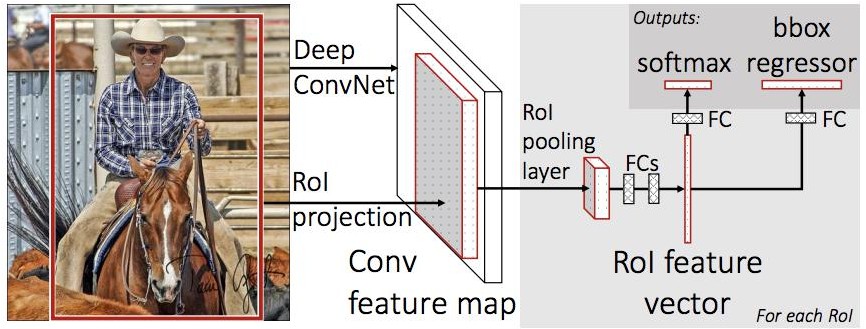


**Figure 4**

These 2000 candidate regions which are proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. The CNN plays a role of feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM for the classify the presence of the object within that candidate region proposal.

* + 1. **Problems with R-CNN**
* It cannot be implemented real time as it takes around 47 seconds for each test image.
* It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
* The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

**3.3 Fast R-CNN**

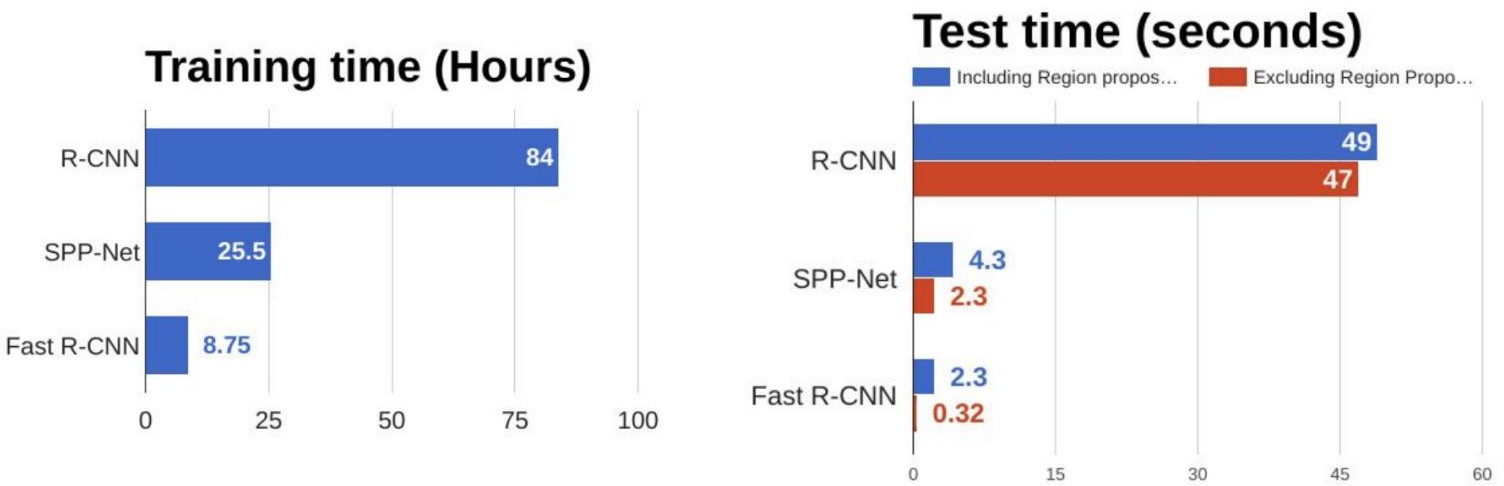


**Figure 5**

The same author of the previous paper(R-CNN) solved some of the drawbacks of R-CNN to build a faster object detection algorithm and it was called Fast R-CNN. The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map.

From the convolutional feature map, we can identify the region of the proposals and warp them into the squares and by using an RoI pooling layer we reshape them into the fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, we can use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box.

The reason “Fast R-CNN” is faster than R-CNN is because you don’t have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is always done only once per image and a feature map is generated from it.

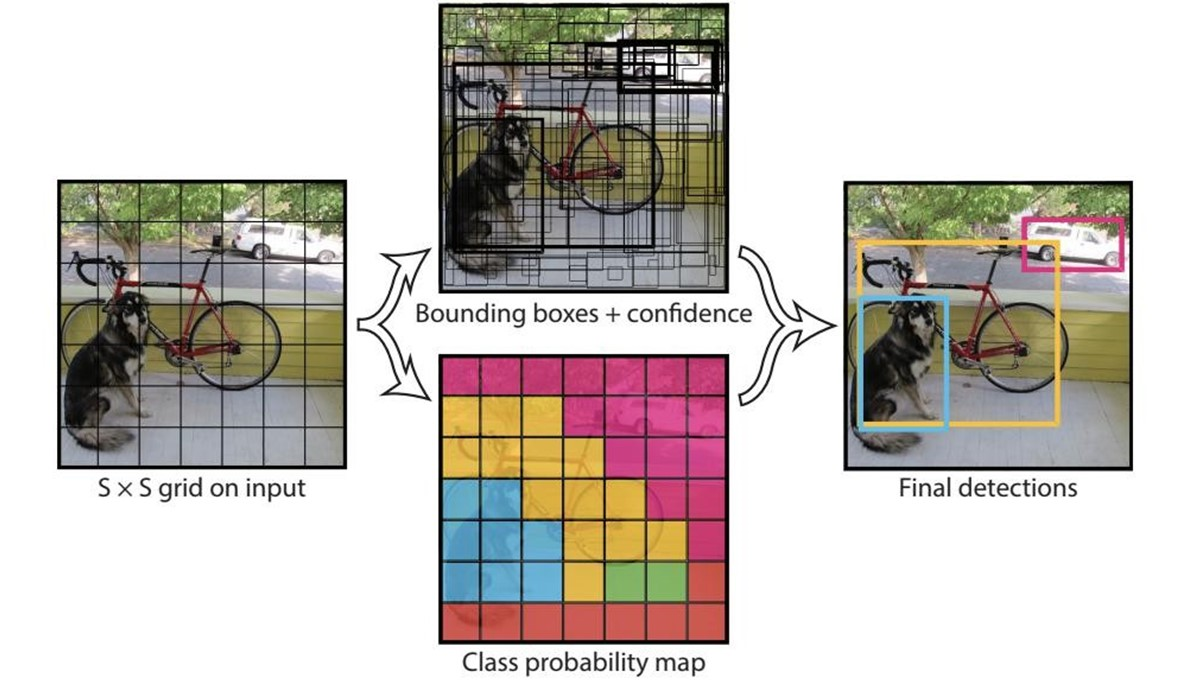


**Figure 6**

From the above graphs, you can infer that Fast R-CNN is significantly faster in training and testing sessions over R-CNN. When you look at the performance of Fast R-CNN during testing time, including region proposals slows down the algorithm significantly when compared to not using region proposals. Therefore, the region which is proposals become bottlenecks in Fast R-CNN algorithm affecting its performance.

**3.4 YOLO – You Only Look Once**

All the previous object detection algorithms have used regions to localize the object within the image. The network does not look at the complete image. Instead, parts of the image which has high probabilities of containing the object. YOLO or You Only Look Once is an object detection algorithm much is different from the region based algorithms which seen above. In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.



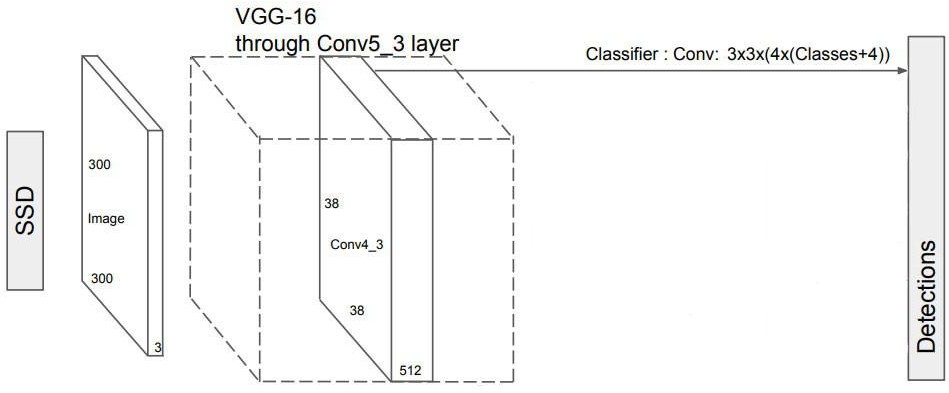
**Figure 7**

YOLO works by taking an image and split it into an SxS grid, within each of the grid we take m bounding boxes. For each of the bounding box, the network gives an output a class probability and offset values for the bounding box. The bounding boxes have the class probability above a threshold value is selected and used to locate the object within the image.

**3.5. SSD**

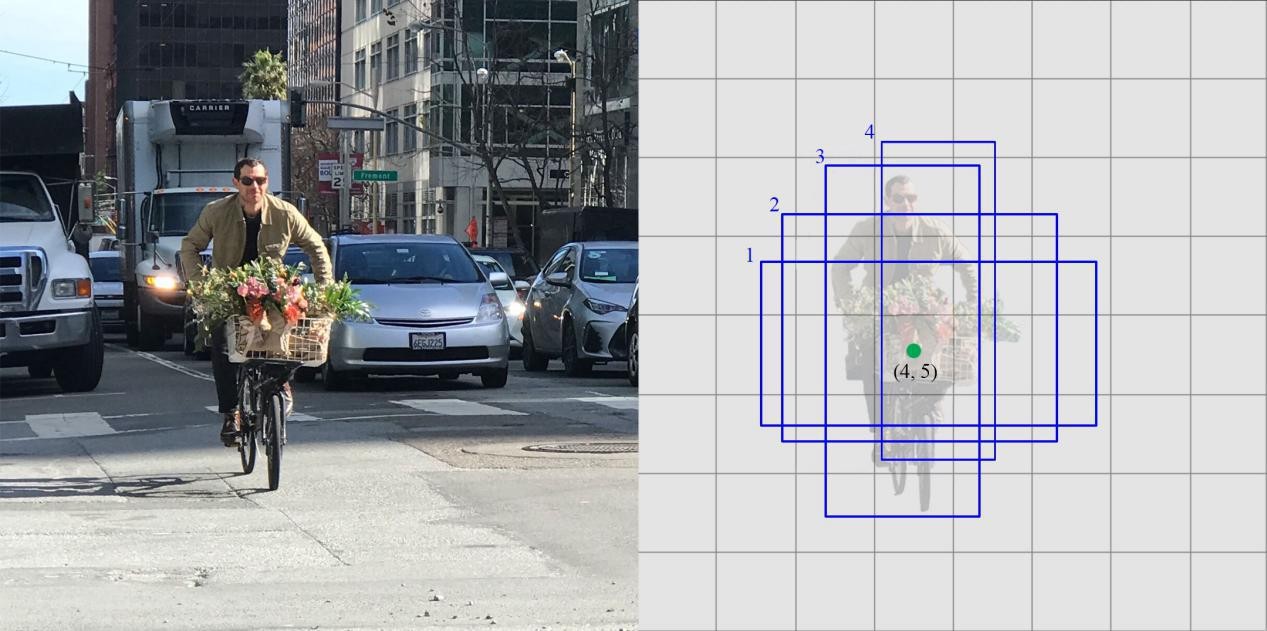
The SSD object detection composes of 2 parts:

1. Extract feature maps, and
2. Apply convolution filters to detect objects.



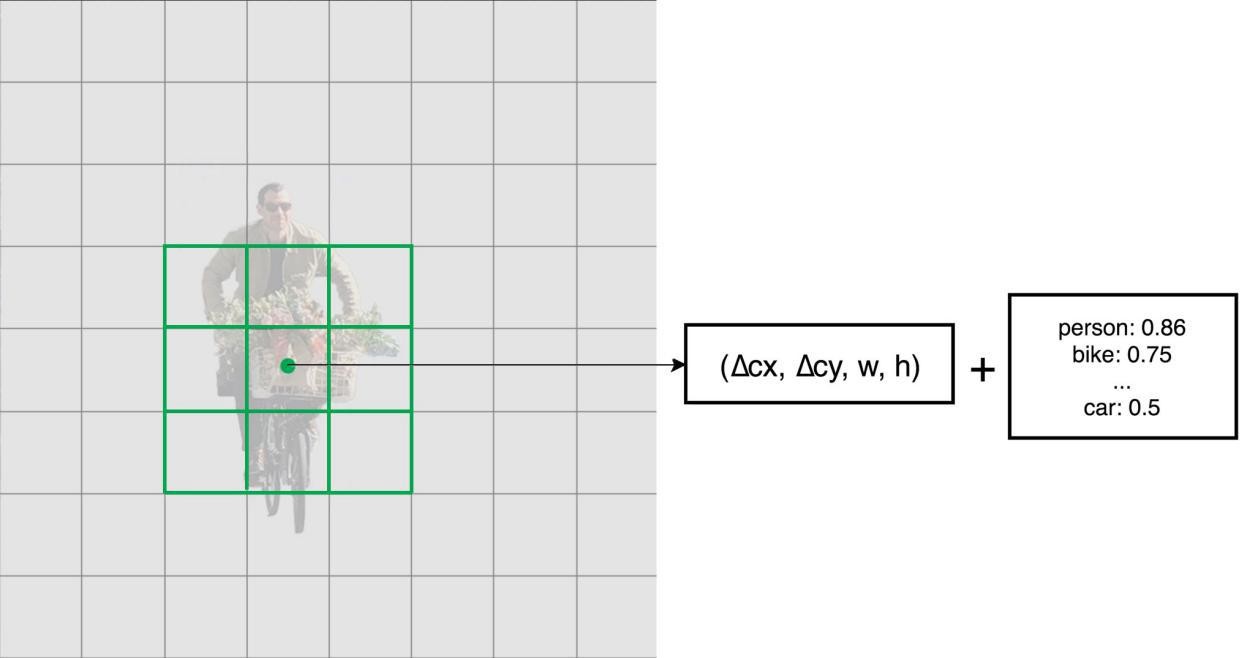
**Figure 8**

SSD uses VGG16 to extract feature maps. Then it detects objects using the Conv4\_3 layer. For illustration, we draw the Conv4\_3 to be 8 × 8 spatially (it should be 38 × 38). For each cell in the image(also called location), it makes 4 object predictions.This is a sample image we feed to the algorithm and expect our algorithm to detect and identify objects in the image and label them according to the class assigned to it. Conv4\_3 makes total of 38 × 38 × 4 predictions: four predictions per cell regardless of the depth of featuremaps. A expected, many predictions contain no object. SSD reserves a class “0” to indicate.



**Figure 9**

SSD does not use the delegated region proposal network. Instead, it resolves to a very simple method. It computes both the location and class scores using small convolution filters. After extraction the feature maps, SSD applies 3 × 3 convolution filters for each cell to make predictions. (These filters compute the results just like the regular CNN filters.) Each filter gives outputs as 25 channels: 21 scores for each class plus one boundary box.



**Figure 10**

**CHAPTER 4**

**RESULT**

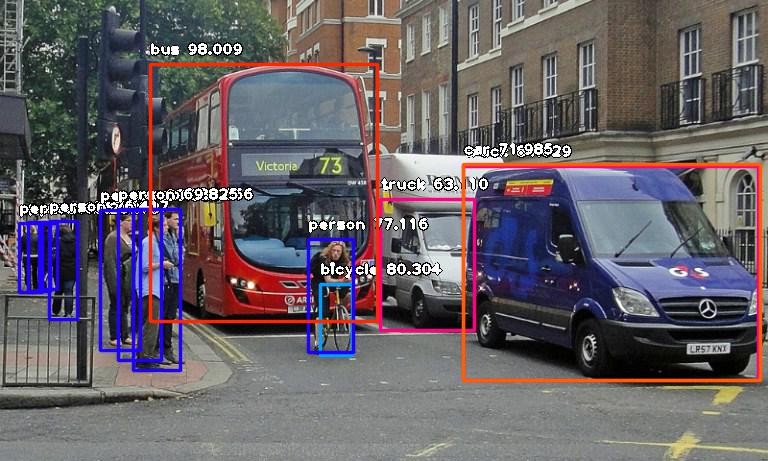
**4.1. Input & Output**

This is a sample image we feed to the algorithm and expect our algorithm to detect and identify objects in the image and label them according to the class assigned to it.



**Figure 11 : Before Detection**

As expected our algorithm identifies the objects by its classes ans assigns each object by its tag and has dimensions on detected image.



**Figure 12 : After Detection**

ImageAI provides many more features useful for customization and production capable deployments for object detection tasks. Some of the features supported are :

* Adjusting Minimum Probability: By default, objects detected with a probability percentage of less than 50 will not be shown or reported. You can increase this value for high certainty cases or reduce the value for cases where all possible objects are needed to be detected
* Custom Objects Detection: Using a provided CustomObject class, you can tell the detection class to report detections on one or a few number of unique objects
* Detection Speeds: You can reduce the time it takes to detect an image by setting the speed of detection speed to “fast”, “faster” and “fastest”.
* Input Types: You can specify and parse in file path to an image, Numpy array or file stream of an image as the input image .
* Output Types: You can specify that the detectObjectsFromImage function should return the image in the form of a file or Numpy array .

**4.2. Detection Speed**

ImageAI now provides detection speeds for all object detection tasks. The detection speeds allow you to reduce the time of detection at a rate between 20% - 80%, and yet having just slight changes but accurate detection results. Coupled with lowering the minimum-percentage-probability parameter, detections can match the normal speed and yet reduce detection time drastically. The available detection speeds are **"normal"**(default), **"fast"**, **"faster"** , **"fastest"** and **"flash"**. All you need to do is to state the speed mode you desire when loading the modelin the code.

Image AI provides options to hide the name of objects detected and / or the probability from being shown on the saved / returned detected image.

Using the detect Objects From Image () and detectCustomObjectsFrom image() functions, the parameters display-object-name and display-percentage-probability can be set to true or false individually.

**4.3. Summary**

The object detection report encompasses various stages, starting with data collection and concluding with model deployment and maintenance. The process involves gathering a diverse dataset, annotating images, and preprocessing data for optimal performance. Model selection, training, validation, and testing follow, ensuring a robust and accurate detection model. Post-processing techniques refine results, and integration into the target system precedes optimization for efficiency. Deployment in the intended environment marks a crucial stage, with ongoing monitoring and maintenance to adapt to evolving data patterns. The report underscores the comprehensive methodology, emphasizing the significance of each step in achieving successful object detection.

**CHAPTER 5**

**CONCLUSION AND FUTURE SCOPE**

**5.1. Conclusion**

The possibilities of using computer vision to solve real world problems are immense. The basics of object detection along with various ways of achieving it and its scope has been discussed. Python has been preferred over MATLAB for integrating with OpenCV because when a Matlab program is run on a computer, it gets busy trying to interpret all that Matlab code as Matlab code is built on Java. OpenCV is basically a library of functions written in C\C++. Additionally, OpenCV is easier to use for someone with little programming background. So, it is better to start researching on any concept of object detection using OpenCV-Python.

Feature understanding and matching are the major steps in object detection and should be performed well and with high accuracy. Deep Face is the most effective face detection method that is preferred over Haar-Cascade by most of the social applications like facebook, snap chat, Instagram etc.

In the coming days, OpenCV will be immensely popular among the coders and will also be the prime requirement of IT companies. To improve the performance of object detection IOU measures are used.

**5.2. Application and Future Scope**

Computer vision is still a developing discipline, it has not been matured to that level where it can be applied directly to real life problems.

After few years‟ computer vision and particularly the object detection would not be any more futuristic and will be ubiquitous. For now, we can consider object detection as a sub-branch of machine learning.

Some common and widely used application of object detection are:

**5.2.1. Face Detection**

Have you ever wondered how Facebook detects your face when you upload a photo? Not only it detects, it remembers the face too. This is a simple application of object detection that we see in our daily life.

**5.2.2. Counting objects/peoples**

Object detection can be also used for counting purpose, it is used for keeping a count of particular or all objects in an image or a frame. For e.g. from a group photograph it can count the number of persons and if implemented smartly you may also find out different people with different dresses.

**5.2.3. Vehicle detection**

Similarly, when the object is a vehicle, object detection along with tracking can be used for finding the type of vehicle, this application may be extended to even make a traffic calculator.

**5.2.4. Industries**

Object detection is also used in industrial processes for the identification of different products. Say you want your machine to only detect objects of a particular shape, you can achieve it very easily. For e.g. Hough circle detection transform [6] can be used for detecting circular objects.

**5.2.5 Security**

Identification of unwanted or suspicious objects in any particular area or more specifically object detection techniques are used for detecting bombs/explosives. It is also even used for personal security purpose.

**5.2.6. Biometric recognition**

Biometric recognition uses physical or behavioral traits of humans to recognize any individuals for security and authentication purpose [1]. It uses distinct biological traits like fingerprints, hand geometry, retina and iris patterns etc.

**5.2.7. Surveillance**

Objects can be recognized and tracked in videos for security purpose. Object recognition is required so that the suspected person or vehicle can be tracked.

**5.2.8. Medical analysis**

Object detection is used to detect diseases like a tumor, stones, cancer in MRI images.

**REFERENCES**

1. Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images via a sparse, part-based representation. IEEE Trans. Pattern Anal. Mach. Intell. 26,1475–1490. doi:10.1109/TPAMI.2004.108
2. Alexe, B., Deselaers, T., and Ferrari, V. (2010). “What is an object?,” in ComputerVision and Pattern Recognition (CVPR), 2010 IEEE Conference on (San Francisco,CA: IEEE), 73–80. doi:10.1109/CVPR.2010.5540226
3. Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988). Active vision. Int. J.Comput. Vis. 1, 333–356. doi:10.1007/BF00133571
4. Andreopoulos, A., and Tsotsos, J. K. (2013). 50 years of object recognition: direc-tions forward. Comput. Vis. Image Underst. 117, 827–891. doi:10.1016/j.cviu.2013.04.005
5. Azizpour, H., and Laptev, I. (2012). “Object detection using strongly-superviseddeformable part models,” in Computer Vision-ECCV 2012 (Florence: Springer),836–849.
6. Azzopardi, G., and Petkov, N. (2013). Trainable cosfire filters for keypoint detectionand pattern recognition. IEEE Trans. Pattern Anal. Mach. Intell. 35, 490–503.doi:10.1109/TPAMI.2012.106
7. Azzopardi, G., and Petkov, N. (2014). Ventral-stream-like shape representation:from pixel intensity values to trainable object-selective cosfire models. Front.Comput. Neurosci. 8:80. doi:10.3389/fncom.2014.00080
8. Benbouzid, D., Busa-Fekete, R., and Kegl, B. (2012). “Fast classification using sparsedecision dags,” in Proceedings of the 29th International Conference on MachineLearning (ICML-12), ICML ‘12, eds

J. Langford and J. Pineau (New York, NY:Omnipress), 951–958.

**PROGRAM CODE:**

import cv2

import numpy as np

import time

np.random.seed(20)

class Detector:

    def \_\_init\_\_(self, videoPath, configPath, modelPath, classesPath):

        self.videoPath = videoPath

        self.configPath = configPath

        self.modelPath =modelPath

        self.classesPath = classesPath

        self.net = cv2.dnn.DetectionModel(self.modelPath, self.configPath)

        self.net.setInputSize(320,320)

        self.net.setInputScale(1.0/127.5)

        self.net.setInputMean((127.5, 127.5, 127.5))

        self.net.setInputSwapRB(True)

        self.readClasses()

    def readClasses(self):

       with open(self.classesPath, 'r') as f:

           self.classesList = f.read().splitlines()

       self.classesList.insert(0, '\_\_Background\_\_')

       self.colorList = np.random.uniform(low=0, high=255, size=(len(self.classesList), 3))

       # print(self.classesList)

    def onVideo(self):

        cap = cv2.VideoCapture(self.videoPath)

        if (cap.isOpened()==False):

            print("Error opening file...")

            return

        (success, image) = cap.read()

        startTime = 0

        while success:

            currentTime = time.time()

            fps = 1/(currentTime - startTime)

            startTime = currentTime

            classLabelIDs, confidences, bboxs = self.net.detect(image, confThreshold = 0.4)

            bboxs = list(bboxs)

            confidences = list(np.array(confidences).reshape(1,-1)[0])

            confidences = list(map(float, confidences))

            bboxIdx = cv2.dnn.NMSBoxes(bboxs, confidences, score\_threshold = 0.5, nms\_threshold = 0.2)

            if len(bboxIdx) !=0:

                for i in range(0, len(bboxIdx)):

                    bbox = bboxs[np.squeeze(bboxIdx[i])]

                    classConfidence = confidences[np.squeeze(bboxIdx[i])]

                    classLabelID = np.squeeze(classLabelIDs[np.squeeze(bboxIdx[i])])

                    classLabel = self.classesList[classLabelID]

                    classcolor = [int(c) for c in self.colorList[classLabelID]]

                    displayText = "{}: {:.2f}".format(classLabel, classConfidence)

                    x,y,w,h = bbox

                    cv2.rectangle(image, (x,y), (x+w, y+h), color=classcolor, thickness=1)

                    cv2.putText(image, displayText, (x,y-10), cv2.FONT\_HERSHEY\_PLAIN, 1, classcolor, 2)

                    lineWidth = min(int(w \* 0.3), int(h \* 0.3))

                    cv2.line(image, (x,y), (x +lineWidth, y), classcolor, thickness= 5)

                    cv2.line(image, (x,y), (x, y + lineWidth), classcolor, thickness= 5)

                    cv2.line(image, (x + w,y), (x + w - lineWidth, y), classcolor, thickness= 5)

                    cv2.line(image, (x + w,y), (x + w, y + lineWidth), classcolor, thickness= 5)

                    cv2.line(image, (x,y + h), (x +lineWidth, y + h), classcolor, thickness= 5)

                    cv2.line(image, (x,y + h), (x, y + h - lineWidth), classcolor, thickness= 5)

                    cv2.line(image, (x + w,y + h), (x + w - lineWidth, y + h), classcolor, thickness= 5)

                    cv2.line(image, (x + w,y + h), (x + w, y + h - lineWidth), classcolor, thickness= 5)

            cv2.putText(image, "FPS: " + str(int(fps)), (20,70), cv2.FONT\_HERSHEY\_PLAIN, 2, (0,255,0), 2)

            cv2.imshow("Result", image)

            key = cv2.waitKey(1) & 0xFF

            if key == ord("q"):

                break

            (success,image) = cap.read()

        cv2.destroyAllWindows()

**MAIN PYTHON CODE:**

from Detector import \*

import os

def main():

    videoPath =  0

    configPath = os.path.join("model\_data", "ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt")

    modelPath = os.path.join("model\_data", "frozen\_inference\_graph.pb")

    classesPath = os.path.join("model\_data", "coco.names")

    detector = Detector(videoPath, configPath, modelPath, classesPath)

    detector.onVideo()

if \_\_name\_\_ == '\_\_main\_\_':

    main()