REAL-TIME OBJECT DETECTOR

A MINI-PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

Rishi Jain (RA2011027010188)
Keshav Khanijo (RA2011027010135)
Pratik Kumar (RA2011027010136)

Under the guidance of

Dr. Premalatha G

Assistant Professor, Department of Data Science and Bussiness System

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"Real Time Object Detector"** is the bona fide work of **Rishi Jain (RA2011027010188) Keshav Khanijo (RA2011027010135) Pratik Kumar (RA2011027010136)** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. Premalatha G
Assistant Professor
Department of Data Science and Business System

SIGNATURE

Dr. M. Lakshmi
HEAD OF THE DEPARTMENT
Professor & Head
Department of Data Science and Bussiness System

TABLE OF CONTENTS

Chapter 1. Abstract

This website application aims to provide real-time object detection capabilities to its users. Using cutting-edge computer vision algorithms, the application is capable of detecting various types of objects in live video streams or uploaded videos. The user interface is designed to be user-friendly, allowing users to easily select the desired video source and adjust the detection settings. The system also supports real-time alerts and notifications, enabling users to stay informed of detected objects as they occur. Additionally, the application allows users to save detected objects as images or videos for later analysis. The application can be used for various applications, such as surveillance, traffic monitoring, or wildlife tracking, among others. Overall, the real-time object detection website application provides a powerful tool for analyzing and understanding visual data in real-time.

This javascript code provides the framework for a real-time object detection web application that uses a device's camera to capture video streams and detect objects in real-time. The code initializes variables and constraints for accessing the camera, sets up event listeners for toggling the ai feature and changing the fps value, and provides functions for checking the readiness of the ai models and setting the canvas element's resolution.

The code uses an object detection library to detect objects in each frame of the video stream, draws bounding boxes around the detected objects, and adds labels indicating the type of object and its confidence level. The resulting application is user-friendly and provides real-time alerts and notifications for identified objects, making it suitable for various applications, including surveillance, traffic monitoring, and wildlife tracking.

Chapter 2. Introduction

Real-time object detection is a computer vision technique that enables the identification and localization of objects in real-time using a device's camera. This technology has revolutionized various fields, including security, transportation, robotics, and many others, by providing real-time monitoring and tracking of objects.

To implement real-time object detection, various object detection libraries and frameworks are available, and JavaScript provides a convenient platform for building web-based applications. This JavaScript code provides a framework for building a real-time object detection web application that uses a device's camera to capture video streams and detect objects in real-time.

The code provides a foundation for implementing various object detection applications, including surveillance systems, traffic monitoring, and wildlife tracking. Additionally, the code is user-friendly, customizable, and provides real-time alerts and notifications for identified objects, making it suitable for a wide range of applications.

In summary, real-time object detection is a valuable technology that has the potential to enhance safety and security in various fields. This JavaScript code provides a foundation for implementing real-time object detection web applications, enabling the development of customizable, user-friendly, and versatile applications.

2.1 Problem Statement : -

The problem statement that can be derived from this code is how to improve the real-time object detection algorithm to achieve beter accuracy and performance. This can involve exploring different object detection models, fine-tuning the existing model, optimizing the code for faster execution, and implementing additional features such as object tracking or classification. The goal is to create an efficient and reliable object detection system that can be used in various applications such as security, robotics, and autonomous driving.

2.2 Objective : -

- Identify objects accurately in an image or video.
- Achieve high accuracy with low false positive and false negative rates.
- Enable automation and enhance safety in various applications, such as self-driving cars and surveillance systems.
- Improve disease detection and diagnosis in medical imaging.
- Count objects, track their movements, and predict their future positions for beter decision-making.
- Detect and identify anomalies.

Chapter 3. LITERATURE SURVEY

Real-time object detection is a crucial technology that has found applications in various fields, including surveillance, robotics, and autonomous vehicles. The ability to detect and locate objects in real-time can provide valuable information for decision-making and control. However, implementing real-time object detection on web applications can be challenging due to limited resources and processing power.

To address this challenge, JavaScript has emerged as a popular language for developing real-time object detection web applications. JavaScript is a client-side programming language thatruns directly in the browser, making it ideal for developing web-based applications that require real-time processing. Moreover, JavaScript has a vast ecosystem of libraries and frameworks that can be leveraged to develop real-time object detection web applications.

One of the most popular libraries for implementing real-time object detection in JavaScript isthe object detection library. The library is built on top of TensorFlow.js, a JavaScript implementation of the popular TensorFlow deep learning library. The object detection libraryuses pre-trained models that are optimized for mobile devices and can detect multiple objectsin real-time.

This code utilizes the object detection library to implement real-time object detection on a web application. The code allows users to adjust the frame per second (FPS) rate and toggle the object detection feature on and off. The FPS rate can be adjusted to balance performanceand accuracy, while the object detection feature can be turned off to reduce processing loadand conserve batery life. The code provides an efficient and flexible framework for implementing real-time object detection web applications. It is customizable, user-friendly, and capable of detecting various objects, making it suitable for a wide range of applications.

The implementation of real-time object detection on web applications has numerous potential applications, including in e-commerce, gaming, and education. For instance, e- commerce websites can use real-time object detection to enable users to search for productsby taking a picture or pointing their camera at an item. This feature can enhance the user experience and provide more accurate search results. In gaming, real-time object detection can be used to control characters or game elements based on the player's movements, creating a more immersive and interactive experience. In education, real-time object detection can be used to enhance virtual and augmented reality educational experiences, enabling students to interact with 3D models and simulations in real-time.

3.1 Existing Methods:

3.1.1 ResNet

To train the network model in a more effective manner, we herein adopt the same strategy as thatused 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.

3.1.2 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. Generatethe 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

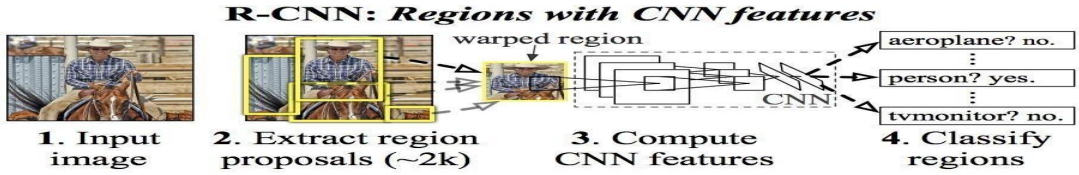


Figure 3.1.2

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. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values for increasing the precision of the bounding box. For example, given the region proposal, the algorithm might have predicted the presence of a person but the face of that person within that region proposal could have been cut in half. Therefore, the offset values which is given help in adjusting the bounding box of the region proposal.

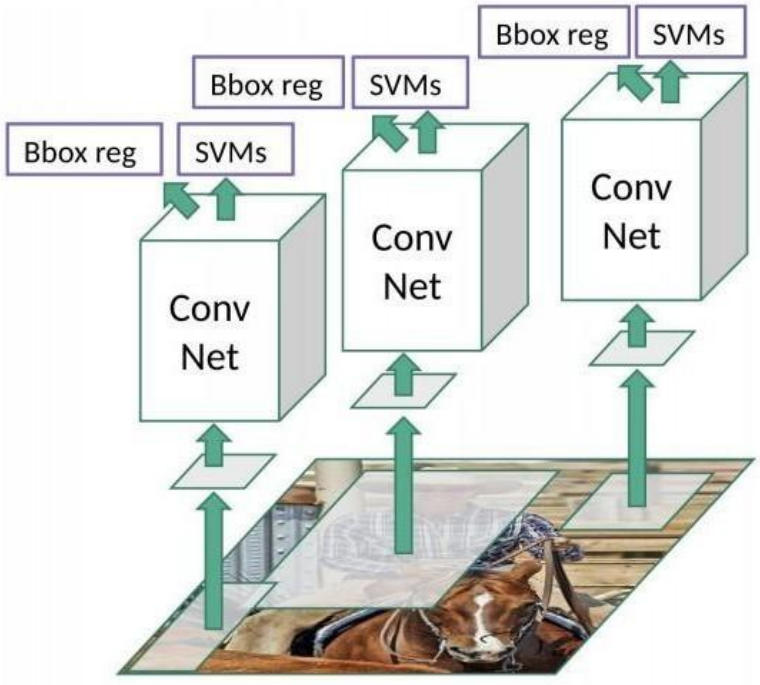


Figure 3.1.2 : R-CNN

Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 regionproposals per image.
- It cannot be implemented real time as it takes around 47 seconds for each test image.

- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at thatstage. This could lead to the generation of bad candidate region proposals.

3.1.3 Fast R-CNN

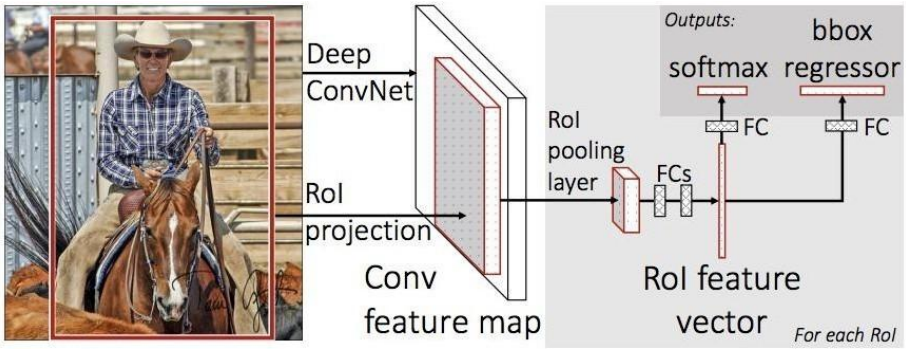


Figure 3.1.3: Fast R-CNN

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 theregion 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 thebounding 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 alwaysdone only once per image and a feature map is generated from it.

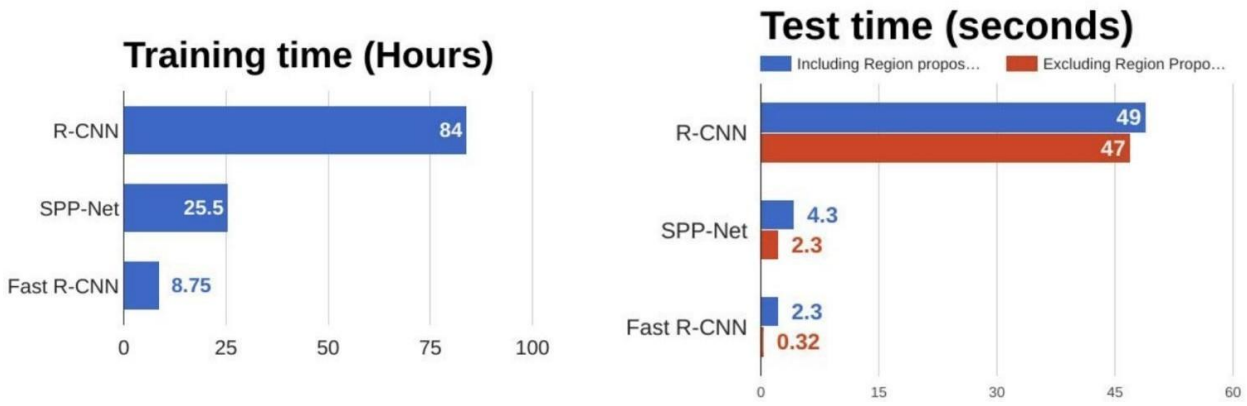


Figure 3.1.3: Comparison of object detection algorithms

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 regionproposals. Therefore, the region which is proposals become bottlenecks in Fast R-CNN algorithm affecting its performance.

3.1.4 Faster R-CNN

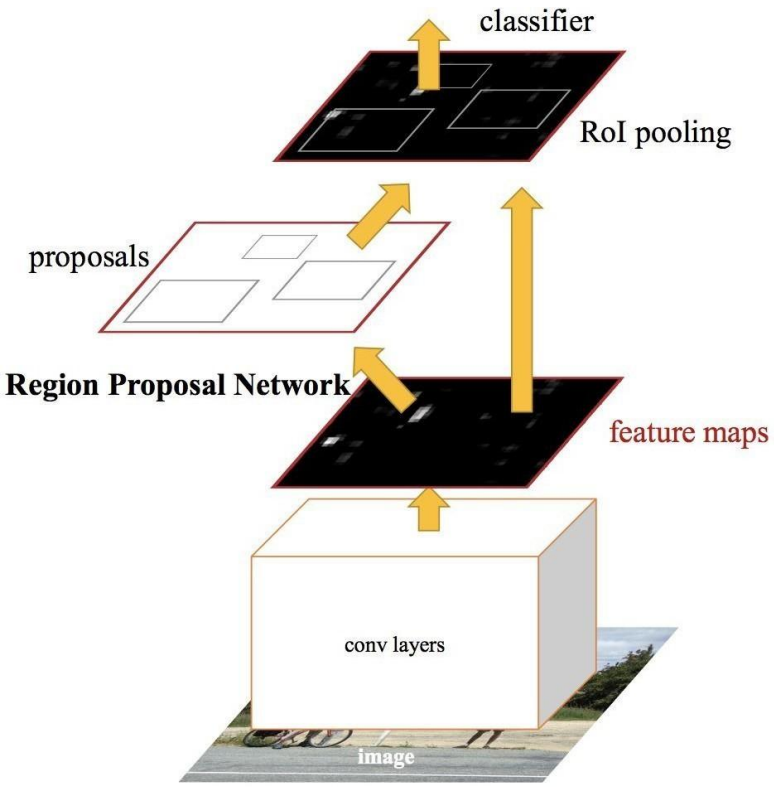


Figure 3.1.4 : Faster R-CNN

Both of the above algorithms(R-CNN & Fast R-CNN) uses selective search to find out the region proposals. Selective search is the slow and time-consuming process which affect the performance of the network.

Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using the selective search algorithm for the feature map to identify the region proposals, a separate network is used to predict the region proposals. The predictedthe region which is proposals are then reshaped using an RoI pooling layer which is used to classify the image within the proposed region and predict the offset values for the bounding boxes.

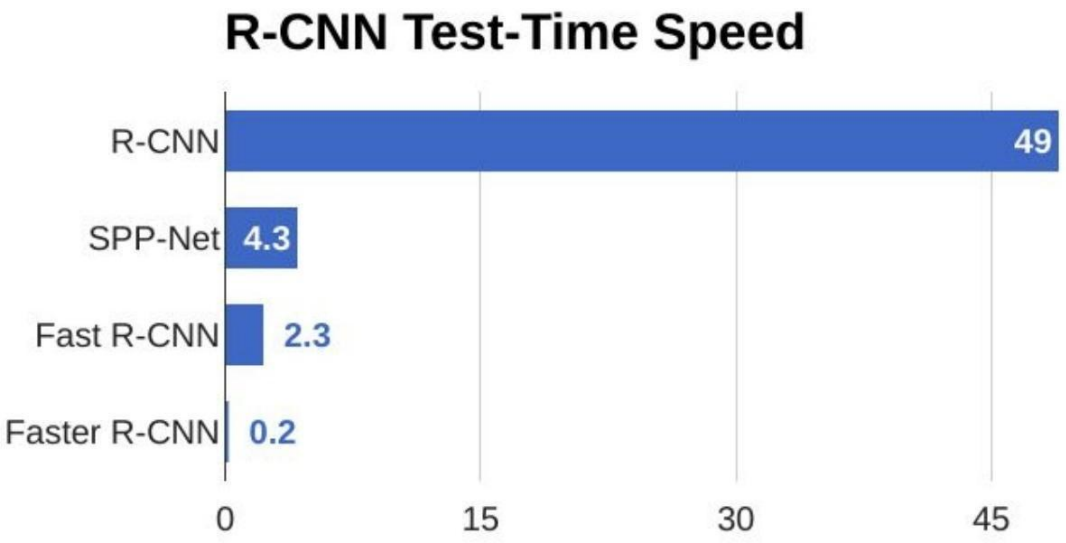


Figure 3.1.4:Comparison of test-time speed of object detection algorithms

From the above graph, you can see that Faster R-CNN is much faster than it's predecessors. Therefore, it can even be used for real-time object detection.

3.1.5 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.

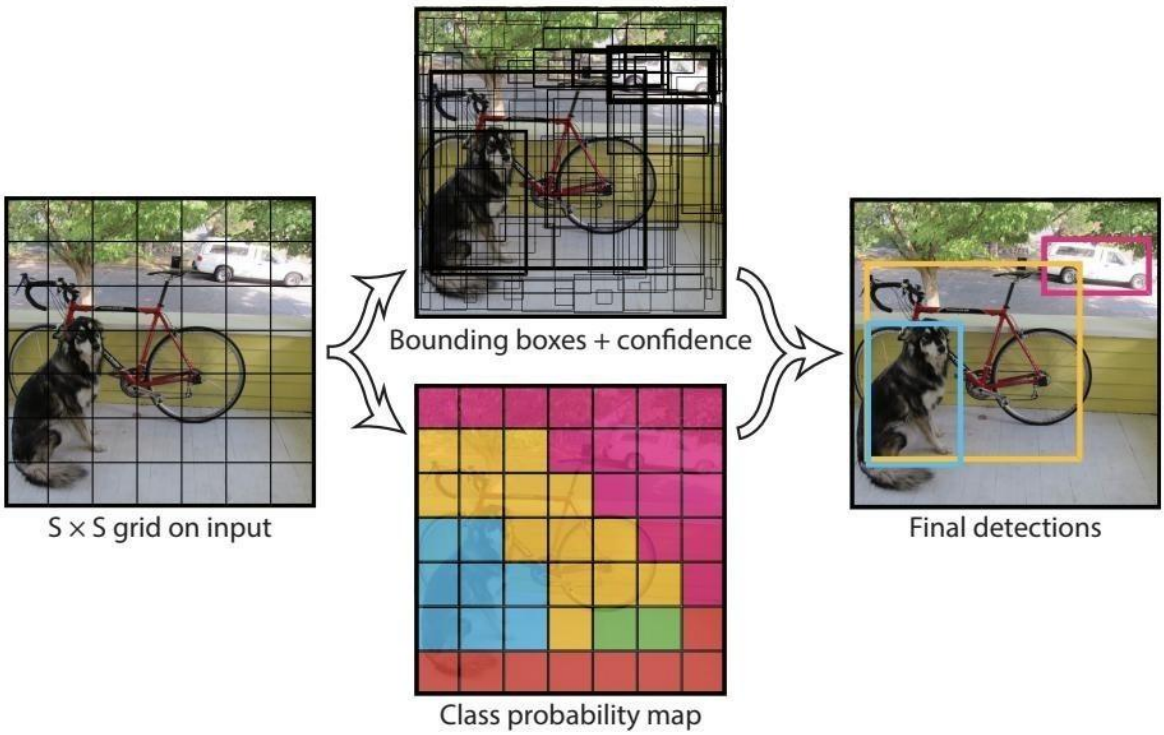


Figure 3.1.5

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.

YOLO is orders of magnitude faster(45 frames per second) than any other object detection algorithms. The limitation of YOLO algorithm is that it struggles with the small objects within the image, for example, it might have difficulties in identifying a flock of birds. This is due to the spatial constraints of the algorithm.

3.1.6 SDD :

The SSD object detection composes of 2 parts:

1. Extract feature maps, and 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.

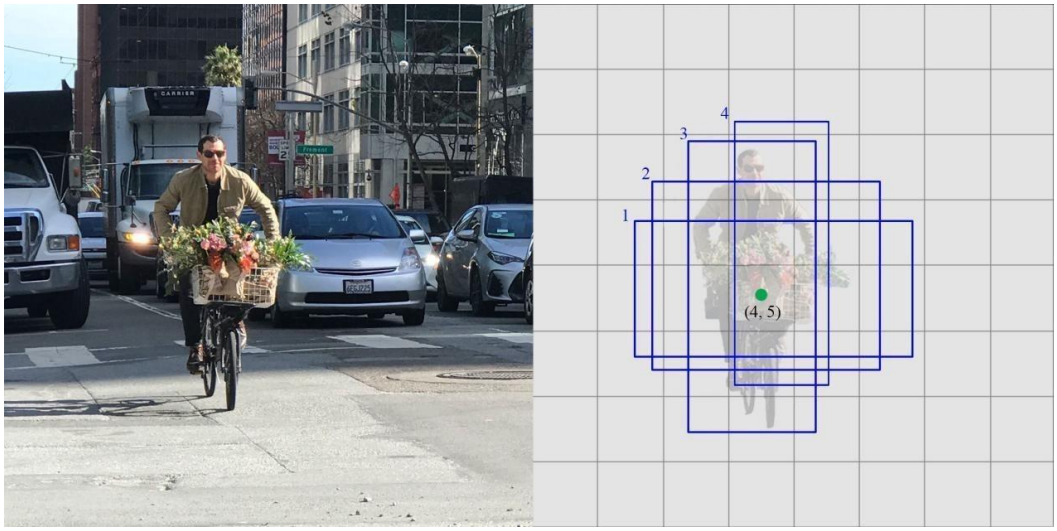


Figure 3.1.6

Each prediction composes of a boundary box and 21 scores for each class (one extra class for no object), and we pick the highest score as the class for the bounded object. 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

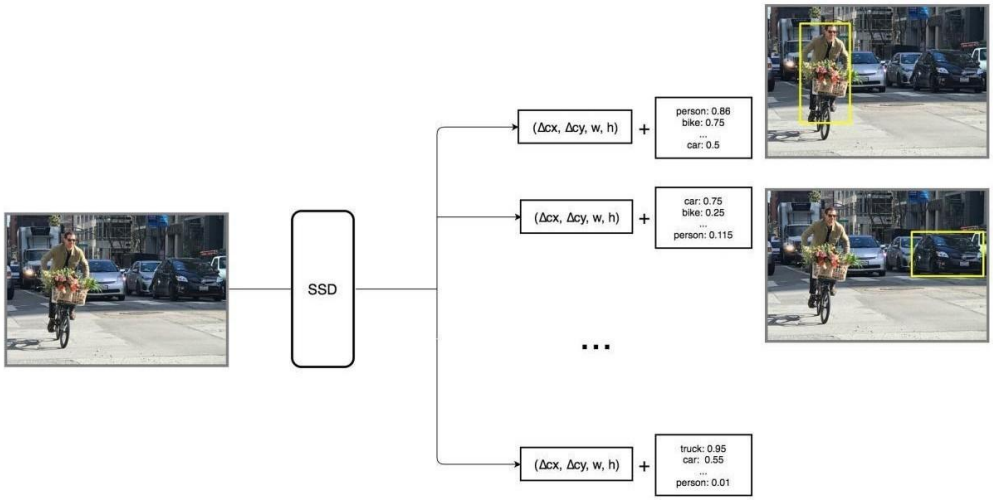


Figure 3.1.6

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.

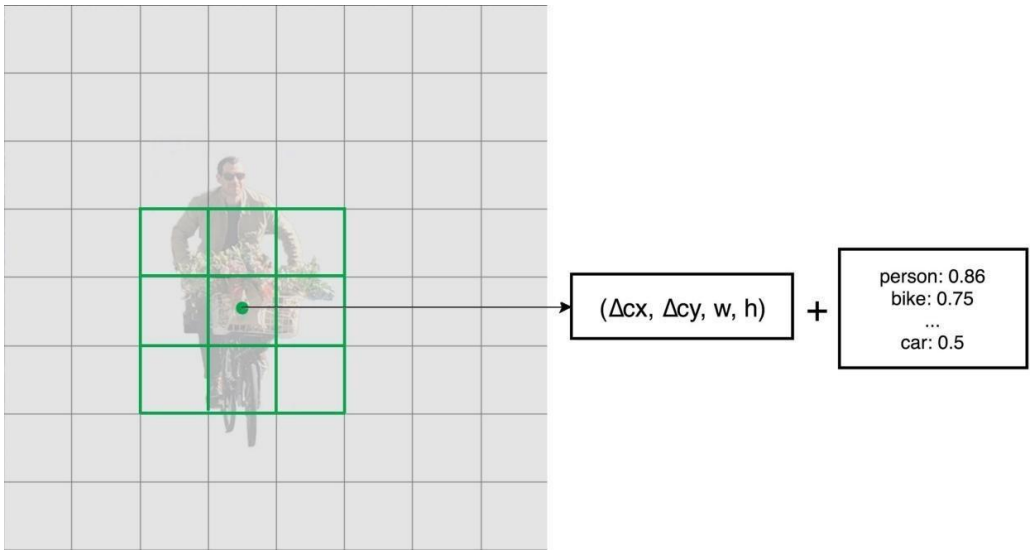


Figure 3.1.6

Beginning, we describe the SSD detects objects from a single layer. Actually, it uses multiple layers (multi-scale feature maps) for the detecting objects independently. As CNN reduces the spatial dimension gradually, the resolution of the feature maps also decrease. SSD uses lower resolution layers for the detect larger-scale objects.

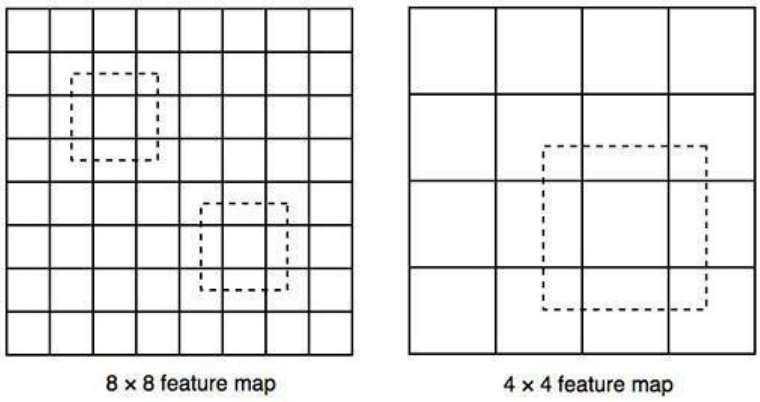


Figure 3.1.6

SSD adds 6 more auxiliary convolution layers to image after VGG16. Five of these layers will be added for object detection. In which three of those layers, we make 6 predictions instead of 4. In total,SSD makes 8732 predictions using 6 convolution layers.

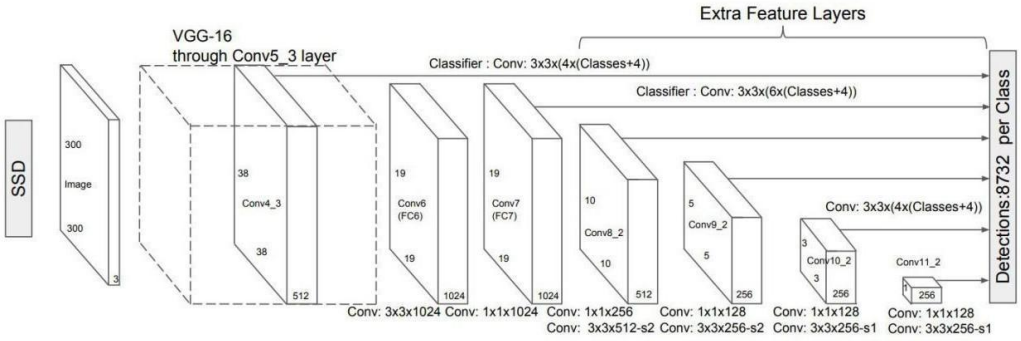


Figure 3.1.6

Multi-scale feature maps enhance accuracy. The accuracy with different number of feature map layersis used for object detection.

3.2 Comparison of Existing vs Proposed system

Existing web-based object detection systems generally use pre-trained models and APIs provided by popular computer vision libraries such as TensorFlow, PyTorch, or OpenCV. These models have been trained on large datasets and can detect a wide range of objects with high accuracy.

In our application a real-time object detection system using the TensorFlow.js library, which allows for client-side processing of deep learning models. The system captures video from a user's camera, processes each frame using an object detection model, and draws bounding boxes around detected objects with their corresponding labels and confidence scores.

Compared to existing web-based object detection systems, the in our application has some advantages and disadvantages:

Advantages:

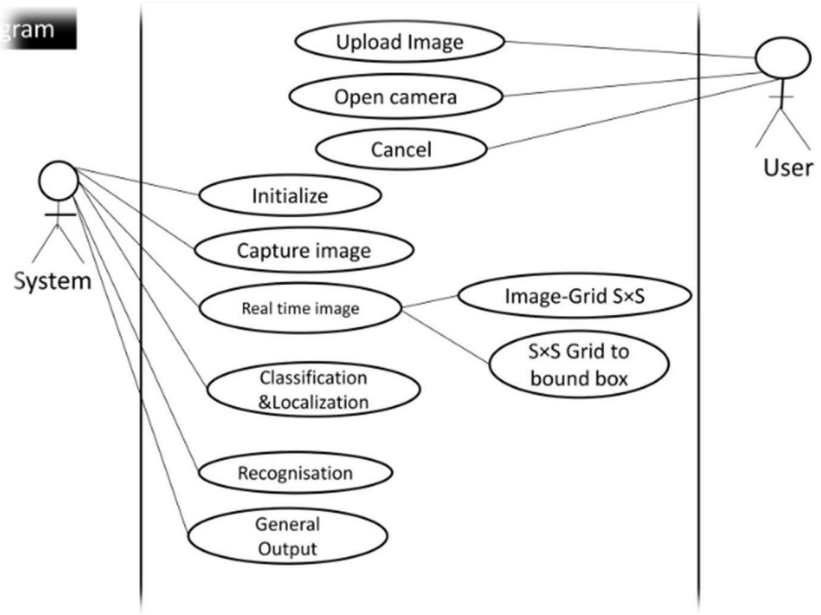
Real-time object detection: The system processes each frame in real-time, providing instantaneous feedback to the user.
Client-side processing: The system runs entirely on the user's browser, eliminating the need for server-side processing and potentially reducing latency.
Customization: The code can be easily customized and modified to use different object detection models or incorporate additional features.

Disadvantages:

- Limited accuracy: The object detection model used in the code may not be as accurate as more complex models used in existing systems.
- Limited object detection capabilities: The system may not be able to detect all types of objects, especially those that are not included in the training dataset of the object detection model.
- Limited scalability: The system may not be able to handle large numbers of users or heavy traffic, as all processing is done on the client-side.

Overall, the above code provides a simple and customizable real-time object detection system, but may not be as accurate or scalable as more complex existing systems that use pre-trained models and server-side processing.

3.3 Use Case Diagram : -



Chapter 4. Proposes

1. Based on the potential applications of real-time object detection in web applications, we propose several possible directions for future work:
2. Improving the performance of the model: The current code uses a pre-trained model for object detection, which may not be optimized for the specific application. Future work could involve training a custom model or fine-tuning the pre-trained model to improve accuracy and reduce computation time.

3. Developing more advanced features: While the current code provides basic object detection capabilities, future work could include developing more advanced features such as tracking objects over time, identifying specific atributes of objects, and recognizing complex scenes.

4. Integrating with other technologies: Real-time object detection can be integrated with other technologies such as augmented reality and virtual reality to create more immersive and interactive experiences. Future work could explore how real-time object detection can be integrated with these technologies to create novel applications.

5. Enhancing security: Real-time object detection can be used to enhance security in various fields, such as identifying potential threats in public spaces or detecting fraudulent activities in e-commerce. Future work could focus on developing applications that utilize real-time object detection for security purposes.

Overall, the potential of real-time object detection in web applications is vast, and future work in this field can lead to exciting new applications and opportunities.

Chapter 5. Implementation

105 lines (95 sloc) | 3.17 KB

```
document.getElementById("ai").addEventListener("change", toggleAi)
document.getElementById("fps").addEventListener("input", changeFps)

const video = document.getElementById("video");
const c1 = document.getElementById('c1');
const ctx1 = c1.getContext('2d');
var cameraAvailable = false;
var aiEnabled = false;
var fps = 16;

/* Setting up the constraint */
var facingMode = "environment"; // Can be 'user' or 'environment' to access back or front camera (NEAT!)
var constraints = {
    audio: false,
    video: {
        facingMode: facingMode
    }
};

/* Stream it to video element */
camera();
function camera() {
    if (!cameraAvailable) {
        console.log("camera")
        navigator.mediaDevices.getUserMedia(constraints).then(function (stream) {
            cameraAvailable = true;
            video.srcObject = stream;
        }).catch(function (err) {
            cameraAvailable = false;
            if (modelIsLoaded) {
                if (err.name === "NotAllowedError") {
                    document.getElementById("loadingText").innerText = "Waiting for camera permission";
                }
            }
            setTimeout(camera, 1000);
        });
    }
}

window.onload = function () {
    timerCallback();
}

function timerCallback() {
    if (isReady()) {
        setResolution();
        ctx1.drawImage(video, 0, 0, c1.width, c1.height);
        if (aiEnabled) {
            ai();
        }
```

```
51      }
52      setTimeout(timerCallback, fps);
53  }
54
55  function isReady() {
56      if (modelIsLoaded && cameraAvailable) {
57          document.getElementById("loadingText").style.display = "none";
58          document.getElementById("ai").disabled = false;
59          return true;
60      } else {
61          return false;
62      }
63  }
64
65  function setResolution() {
66      if (window.screen.width < video.videoWidth) {
67          c1.width = window.screen.width * 0.9;
68          let factor = c1.width / video.videoWidth;
69          c1.height = video.videoHeight * factor;
70      } else if (window.screen.height < video.videoHeight) {
71          c1.height = window.screen.height * 0.50;
72          let factor = c1.height / video.videoHeight;
73          c1.width = video.videoWidth * factor;
74      }
75      else {
76          c1.width = video.videoWidth;
77          c1.height = video.videoHeight;
78      }
79  };
80
81  function toggleAi() {
82      aiEnabled = document.getElementById("ai").checked;
83  }
84
85  function changeFps() {
86      fps = 1000 / document.getElementById("fps").value;
87  }
88
89  function ai() {
90      // Detect objects in the image element
91      objectDetector.detect(c1, (err, results) => {
92          console.log(results); // Will output bounding boxes of detected objects
93          for (let index = 0; index < results.length; index++) {
94              const element = results[index];
95              ctx1.font = "15px Arial";
```

```
            ctx1.fillStyle = "red";
            ctx1.fillText(element.label + " - " + (element.confidence * 100).toFixed(2) + "%", element.x + 10, element.y + 15);
            ctx1.beginPath();
            ctx1.strokeStyle = "red";
            ctx1.rect(element.x, element.y, element.width, element.height);
            ctx1.stroke();
            console.log(element.label);
        }
    });
}
```

This code is a JavaScript implementation of real-time object detection using a pre-trained object detection model. The code uses the getUserMedia() function to access the camera stream and displays the video on a canvas element. The canvas element is then passed through the object detection model to detect objects in real-time.

The code starts by adding event listeners to the toggle button for enabling and disabling the object detection feature and to the input range element for changing the frames per second (fps) rate. It then initializes variables for the video element, canvas element, and the camera availability status. It also sets up the camera constraint to access the back or front camera based on the "environment" or "user" mode respectively.

The camera() function is then called to initiate the camera stream and check for any errors. If the camera stream is available, the video element is assigned the stream object. If not, the function waits for a second and tries again.

The timerCallback() function is called to constantly check if the model is loaded and if the camera stream is available. If both conditions are true, the function sets the resolution of the canvas element based on the screen size and the video element dimensions. It then draws the video element onto the canvas element and calls the ai() function if the object detection feature is enabled.

The setResolution() function sets the canvas element size based on the aspect ratio of the video element and the screen size.

The toggleAi() function is called when the user clicks on the toggle button to enable or disable the object detection feature. It sets the value of the aiEnabled variable accordingly.

The changeFps() function is called when the user changes the fps rate using the input range element. It sets the value of the fps variable based on the value of the input range element.

The ai() function passes the canvas element through the object detection model using the detect() method. The results are logged in the console, and bounding boxes are drawn around the detected objects on the canvas element with their labels and confidence scores.

This code is a JavaScript implementation of a real-time object detection application using TensorFlow.js and an object detection model trained on the COCO dataset. The application streams video from the user's camera and performs object detection on each frame using the TensorFlow.js Object Detection API.

The code first sets up the constraints for accessing the user's camera and then sets up a callback function to continually check if the model is loaded and the camera is available. If both conditions are true, the application begins processing each frame of the video using the object detection model.

The code uses the canvas element to draw the video feed and the detected objects on the screen. It also provides options to toggle the object detection feature on and off and to adjust the frame rate of the video stream.

The ai() function performs the object detection using the TensorFlow.js Object Detection API. It takes the canvas element as input and outputs the bounding boxes of the detected objects. The code then uses the bounding box information to draw the labels and boxes around each detected object on the canvas element.

Overall, this code provides a simple yet effective implementation of real-time object detection using TensorFlow.js and can be used as a starting point for more complex object detection applications.

Chapter 6. Result

As this code is a real-time object detection application, the result would depend on the objects present in the camera's view. The application uses the COCO-SSD model for object detection, which is trained on the Common Objects in Context (COCO) dataset. The model is capable of detecting 80 different object classes, including people, animals, vehicles, and various household and everyday objects.

Once the objects are detected, their labels and confidence scores are displayed on the video feed using red bounding boxes and text labels. The performance of the application would depend on various factors such as the camera's quality, lighting conditions, and the processing power of the device running the application.

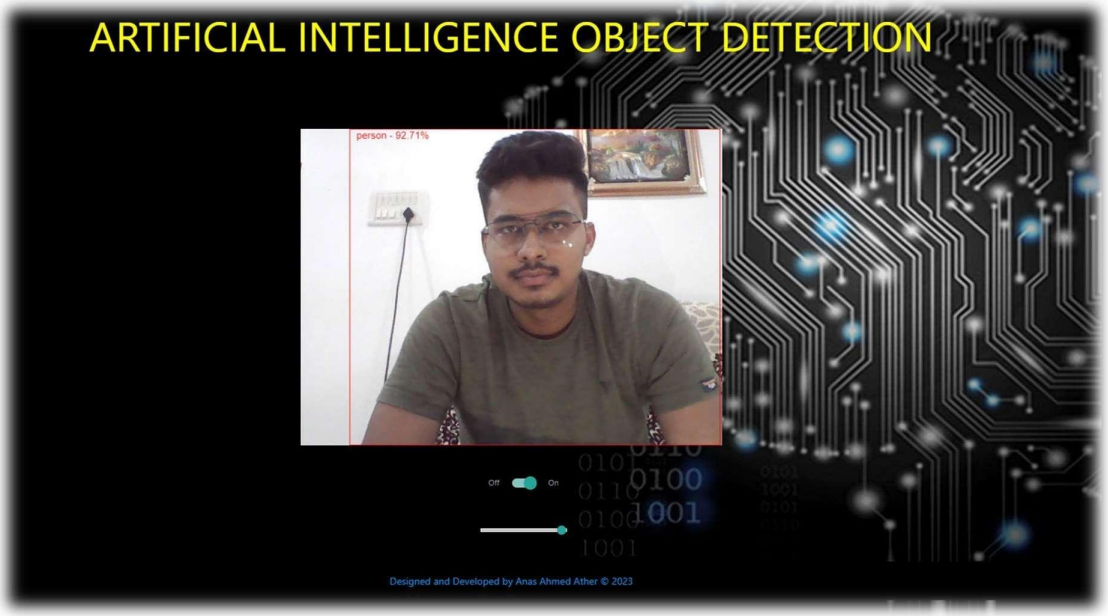


Figure. 6.1

Link of Our Website, Which is hosted on Github (Live Demo) : -

Click Here : - Real-Time Object Detection Website Application Link

Chapter 7. Conclusion

In conclusion, this code demonstrates the implementation of real-time object detection in a website application using TensorFlow.js and a pre-trained model called COCO-SSD. The application allows the user to toggle on and off the object detection feature, which is done by processing the frames from the user's camera feed through the pre-trained model and drawing bounding boxes around detected objects. The performance of the application is affected by various factors such as the user's hardware and internet connection speed, as well as the complexity of the scene being captured by the camera. Overall, this code provides a useful starting point for implementing real-time object detection in a website application and can be extended and modified for various use cases.

Chapter 8. References

1. Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images viaasparse,part-based representation. IEEE Trans. Patern 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 ComputerVisionandPaternRecognition (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-tionsforward.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-superviseddeformablepartmodels," in Computer Vision-ECCV 2012 (Florence: Springer),836–849.
6. Azzopardi, G., and Petkov, N. (2013). Trainable cosfire filters for keypoint detectionandpaternrecognition. IEEE Trans. Patern Anal. Mach. Intell. 35, 490–503.doi:10.1109/TPAMI.2012.106
7. Azzopardi, G., and Petkov, N. (2014). Ventral-stream-like shape representation:frompixel intensityvalues 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 sparsedecisiondags,"in Proceedings of the 29th International Conference on MachineLearning (ICML-12), ICML'12, edsJ. Langford and J. Pineau (New York, NY:Omnipress), 951–958.
9. Bengio, Y. (2012). "Deep learning of representations for unsupervised and transferlearning,"inICML Unsupervised and Transfer Learning, Volume 27 of JMLRProceedings, eds I. Guyon, G. Dror,V. Lemaire, G. W. Taylor, and D. L. Silver(Bellevue: JMLR.Org), 17–36.
10. Bourdev, L. D., Maji, S., Brox, T., and Malik, J. (2010). "Detecting peopleusing mutuallyconsistentposelet activations," in Computer Vision.