Controlling Cursor using Hand Gesture Recognition MLP-11

Rishikesh Dhurde (B-48), Ritik Ragit (B-49)

April 15, 2021

1 Abstract

Our main objective of doing this project is to build an application which can recognize hand gestures and we are trying to give a more personal and interactive touch to interface by using the hand gesture control and it can be used to perform functions of mouse cursor to virtually control a computer.

2 Introduction

Computer technology has grown tremendously over the past few years and has became an important part of our everyday life. The basic computer accessory for Human to Computer Interaction is the mouse. The regular mouse is not appropriate for Human to Computer Interaction in some real life situations, such as with the Human Robot Interaction. There have been many researches on alternative methods to the computer mouse for Human to Computer Interaction. The most natural and intuitive technique for Human to Computer Interaction, that is a viable replacement for the computer mouse is with the use of hand gestures. This project is therefore aimed at investigating and developing a Computer Control system using hand gestures.
Most laptops today are equipped with webcams, which have recently been used insecurity applications utilizing face recognition. In order to harness the full potential of a webcam, it can be used for vision based Computer Control, which would effectively eliminate the need for a computer mouse or mouse pad. The usefulness of a webcam can also be greatly extended to other Human to Computer Interaction application such as a sign language database or motion controller.[2]

3 Related Work

1.Hand Recognition and Gesture Control Using a Laptop Web Camera.
`https://web.stanford.edu/class/cs231a/prev_projects_2016/CS231A_Project_Final.pdf`
In this paper the authors made a hand recognition and gesture control system using a laptop web-camera. They tried three hand segmentation methods-
1.Canny Edge detection(This allowed for color contrasts between the hand and the background, which could be interpreted as edges).

2.Background Subtraction(this involves having the program read multiple frames and subtracting the aggregate values from future frames).
3.Calibration and Thresholding(this method has the potential to be fairly accurate in segmenting the hand, it turned out to be very sensitive to the environment).

2.OBJECT DETECTION AND IDENTIFICATION A Project Report
`https://www.researchgate.net/publication337464355`
In this paper the author has briefly discussed about the object detection concepts. This paper covers about R-CNN, as it's a family of techniques for addressing object recognition tasks and designed for model performances. YOLO is another class of technique for object recognition implemented for real-time use. This paper also covers comparison of speed and accuracy between R-CNN, Fast R-CNN, Faster R-CNN, YOLO and SSD.

3.Real-time Hand-Detection using Neural Networks (SSD) on Tensorflow.
`https://github.com/victordibia/handtracking`
In this repository a model is used to train a hand detector using Tensorflow (Object Detection API). The goal of this repo/post is to demonstrate how neural networks can be applied to the (hard) problem of tracking hands (egocentric and other views). In this model single shot detector is used, SSD only needs an input image and ground truth boxes for each object during training.[1]

4 Dataset and Features

We have made our own data set of images. First we clicked about 600 different images of multiple hand gestures in the different lighting conditions and in daily life environment. After this we labelled all the images using the labelImg tool. After the labelling we get an XML file for each image, this XML file contains details about the bounding box, the class that we have created with in that image. Here we have divided the images in the ratio of 80:20. 80 percent of images for training and 20 percent for the testing of the model. Then we generated the csv file from the obtained xml files, for training and testing of the dataset. Then we used tensorflow object detection model and configured it with its particular configuration file to train this model. After this we started training up the model, using Google Collab by utilising it's GPU. we trained this model for more than 100000 iterations and having very low loss of about 0.1. Then we generated the inference graph for using the model. That's the trained model.

```
INFO:tensorflow:Step 107400 per-step time 0.411s loss=0.085
I0415 07:47:56.274852 140343605712768 model_lib_v2.py:682] Step 107400 per-step time 0.411s loss=0.085
INFO:tensorflow:Step 107500 per-step time 0.404s loss=0.082
I0415 07:48:35.880851 140343605712768 model_lib_v2.py:682] Step 107500 per-step time 0.404s loss=0.082
INFO:tensorflow:Step 107600 per-step time 0.408s loss=0.097
I0415 07:49:15.496038 140343605712768 model_lib_v2.py:682] Step 107600 per-step time 0.408s loss=0.097
INFO:tensorflow:Step 107700 per-step time 0.403s loss=0.109
I0415 07:49:55.191343 140343605712768 model_lib_v2.py:682] Step 107700 per-step time 0.403s loss=0.109
INFO:tensorflow:Step 107800 per-step time 0.410s loss=0.078
I0415 07:50:35.045073 140343605712768 model_lib_v2.py:682] Step 107800 per-step time 0.410s loss=0.078
INFO:tensorflow:Step 107900 per-step time 0.385s loss=0.094
I0415 07:51:14.543937 140343605712768 model_lib_v2.py:682] Step 107900 per-step time 0.385s loss=0.094
INFO:tensorflow:Step 108000 per-step time 0.409s loss=0.085
I0415 07:51:54.132712 140343605712768 model_lib_v2.py:682] Step 108000 per-step time 0.409s loss=0.085
INFO:tensorflow:Step 108100 per-step time 0.415s loss=0.071
I0415 07:52:35.237444 140343605712768 model_lib_v2.py:682] Step 108100 per-step time 0.415s loss=0.071
INFO:tensorflow:Step 108200 per-step time 0.372s loss=0.074
I0415 07:53:15.266545 140343605712768 model_lib_v2.py:682] Step 108200 per-step time 0.372s loss=0.074
INFO:tensorflow:Step 108300 per-step time 0.434s loss=0.098
I0415 07:53:55.016966 140343605712768 model_lib_v2.py:682] Step 108300 per-step time 0.434s loss=0.098
INFO:tensorflow:Step 108400 per-step time 0.414s loss=0.062
I0415 07:54:34.428008 140343605712768 model_lib_v2.py:682] Step 108400 per-step time 0.414s loss=0.062
```

Figure 1: Training of Model

5 Methods

We are trying deep learning to implement this recognition system. By using convolutional neural network, we will train the system by adding multiple different images of hand and different gestures and set the output as per the gestures to perform the basic tasks of the mouse.

We have used deep learning because the data we are using to train the algorithm is already labeled with correct answers(operations). And we will try to identify different hand gestures, which are already trained on a dataset of images that are properly labeled with the hand sign operations.

So for this project we have chosen the Object Detection Algorithm i.e. Single Shot Detector(SSD). SSD has a decent balance between the accuracy and speed. SSD runs a convolutional network on input image only once and calculates it's feature map(like in our case different fingers). VGG-16 Network is used to extract feature maps in the SSD. An SSD makes 8732 predictions(bounding boxes) for every single object. SSD willcheck the confidence score of each box and will pick the top 200 predictions per image. For training of SSD we should have images with ground truth boxes, for each object. Apart from these multiple default boxes of different sizes and aspect ratios are across the image. This process will help us finding the box that overlap the ground truth bounding box. We will check this by finding Intersection over Union(IOU) i.e. (Area of Overlap)/(Area of Union), with the help of IOU we are finding a box having highest overlap and that box is finalized as output out of the 8732 boxes.

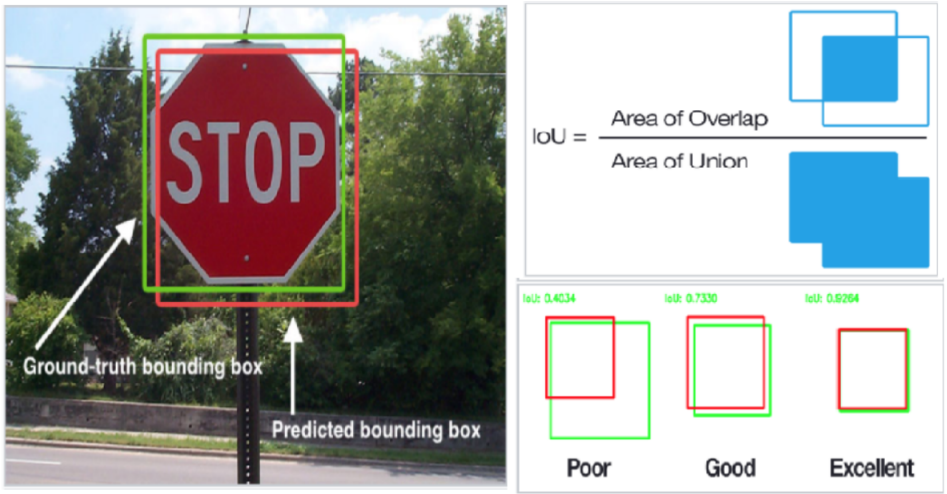

Figure 2: Bounding box process
[5]

6 Experiments/Results/Discussion

We have chosen the state of art model from tensorflow zoo. For this model we have chosen the learning rate to be 0.0001 and number of epochs are more than 100000, we have chosen such small learning rate because we are having less number of images. Batch size of 4 is

selected for this model, because we are not having GPU on our system, so we performed the training on Google Collab.

On SSD model we got the maximum accuracy. Before SSD we have tried the YOLOv3, but here we are not getting the proper accuracy, and sometime it simply did not recognized the hand gestures. For YOLO detection is like a regression problem which takes input image and learns about the class probabilities and bounding box coordinates. If accuracy is not a concern and have to train the model super fast then YOLO is the best.[4]

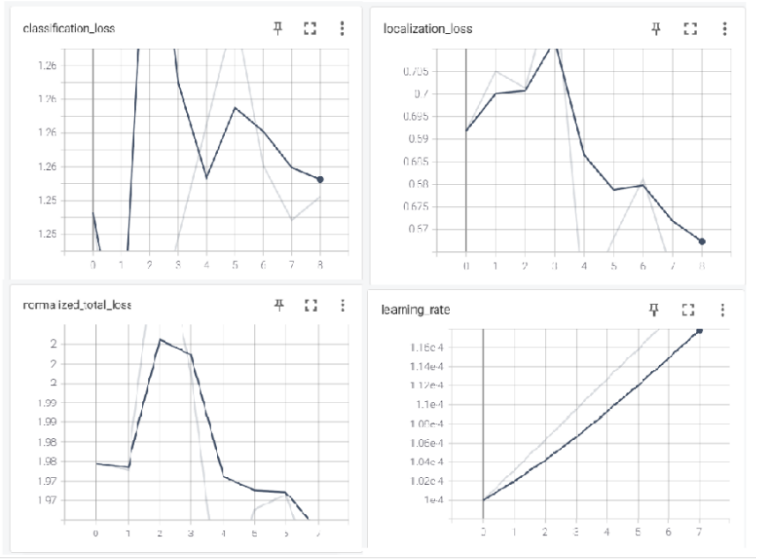

Figure 3: Graph of Accuracy and Loss

n=125	Predicted:NO	Predicted:Yes
Actual:No	1	2
Actual:Yes	3	119

Table 1: Confusion Matrix

Accuracy = (No. of Correct Prediction)/(Total No. of input Samples)
Accuracy = 120/125 = 96 percent.

Precision = (Total Positive)/(Total Positive+False Positive)
Precision = 119/121 = 98.34 percent.

We have overviewed other object detection algorithms also:

1.Region-based Convolutional Neural Network(R-CNN): CNN are slow and computationally expensive, so R-CNN solves this problem by using an object proposal algorithm called selective search which reduces number of bounding boxes that are fed to the classifier.[4]

2.Fast R-CNN: It uses a simple back propagation calculation. Fast-RCNN added the bounding box regression to the neural network training itself. So the network has two heads classification head and bounding box regression head.[4]

3.Faster R-CNN: Well Faster RCNN is similar to Fast R-CNN, but it replaces selective search with a very small convolutional network called Region Proposal Network to generate regions of Interests.[4]

Algorithm	KITTI		Caltech	
	mAP (%)	Speed (ms)	mAP (%)	Speed (ms)
Fast RCNN [25]	49.87	2271	53.13	2140
Faster RCNN [26]	58.78	122	61.73	149
Sliding window & CNN [23]	68.98	79,000	71.26	42,000
SSD [28]	75.73	29.3	77.39	25.6
Context & RCNN [45]	79.26	197	81.75	136
Yolo v1 ($S \times S$) [27]	72.21	44.7	73.92	45.2
T-S Yolo v1 ($S \times 2S$)	74.67	45.1	75.69	45.4
Yolo v2 ($S \times S$) [29]	81.64	59.1	82.81	58.5
T-S Yolo v2 ($S \times 2S$)	83.16	59.6	84.07	59.2
Yolo v3 ($S \times S$) [31]	87.42	24.8	88.44	24.3
T-S Yolo v3 ($S \times 2S$)	88.39	25.2	89.32	24.7

Figure 4: Comparison between different object detection models [6]

7 Conclusion/Future Work

We have tested our dataset using two object detection algorithm i.e. YOLO object detection model and SSD, and came to conclusion that SSD is suitable for our problem and dataset. YOLO was fast to train but we did not achieved the required accuracy. With SSD it took a lot of time to train the dataset but the accuracy was also appreciable.

While implementing this project we faced many issues for dependencies of libraries, at the time of training as we don't have GPU in our device. But overall it was fun in implementing a hand gesture gestures recognition system from our own dataset.

If we had more time, more members and more computational resources try to implement this on more object detection algorithms. We will try to increase accuracy and precision of the model, to smoothly control the cursor using hand gestures. And also try to increase our dataset for better optimization.

8 Contributions

RISHIKESH DHURDE (B-48)- Making of dataset, Writing and compiling report, Code writing and experimentation.

RITIK RAGIT (B-49)- Code research, writing and debugging, Finding relative work and content.

"For this project we had taken help and advise from my friend MR. TANUJ KAMDE."

9 References/Bibliography

References

[1] https://github.com/victordibia/handtracking

[2] https://michigansciencecenter.net/computer-technology-has-tremendously-grown-over-

[3] https://www.researchgate.net/figure/SSD-framework-a-SSD-only-needs-an-input-image-
fig1_286513835

[4] https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

[5] https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-de

[6] https://www.researchgate.net/figure/Detection-accuracy-and-speed-of-all-lane-detec
tbl1_329475183

10 Git Hub Link

https://github.com/RishikeshDhurde/MLP-11.git