International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

An Efficient Parking Bill System using Deep Learning

Anmol Gupta[1], Dhruv Munish Pandey[2]
Department of Information Technology[12]
SRM Institute of Science and Technology, NCR[12]

***Abstract*:- Sometimes the futile process at the parking gates exasperates the customers waiting in long queues to enter. What if parking spaces were fully automated and human interaction were tapered off making the process brisk? The exposition aims at solving the snags stated above and assist the organization with the fully automated Parking Bill System. The implementation of Artificial Intelligence in the parking system could assuage the tortuous process and provide a hassle-free parking experience. With this project, we give incentives to people who finish their work faster and exit the parking lot sooner. The project will help the parking space to be automated and provide a new user experience.**

Keywords: Automatic Number Plate Detection, YOLO, Firebase, Google Tesseract

I. INTRODUCTION

This is the system which could potentially be used by any parking system in the globe. The idea we want to display is that instead of a designated employee working on every entry and exit gate of any institution, be it malls, airports or railway stations, we would put our system to use where it minimizes the workload of employees who can then be effectively assigned a different task. We would create a Deep Learning model known as YOLO (You Only Look Once)[1] which will help us in detecting specifically the number plate from the image or video feed. Then we make use of Google Tesseract to extract text which here is the license plate number and start to calculate the duration. This will ensure safe entry and no corruption in between by a human. Whenever a customer decides to depart from the institution, the customer before going to the parking area will need to use a simple software and after the submission process is done, the timer will be stopped. The customer will now need to make the payment (cash, card, UPI, etc.). As soon as the payment is received, the machine will generate a physical token and this will be our key to exit. The token will be submitted to the machine which will raise the barricade or bollard and ensure the exit.

In this paper, the main focus stays on how to create the system which can be easily implemented and set up in any parking system. The method will involve the algorithm used and the reason for using it. The dataset used, gathering technique and manipulating file structure to prepare the dataset that can be fed to train the model. The paper will also show the data storage technique for storing the license plate number and time of the entry. The database used is cloud storage and real time database option provided by Google Firebase.

II. LITERATURE SURVEY

The early method used was to simply feed the license plate picture and create a dataset with labelling done for each and every plate. The process was slow and inefficient as the model can only work when the number plate is the only constituent of the image and another drawback was that it could only detect one number plate instead of detecting more than one number plates.

The later method didn't detect the number plate directly from the image. The image needed to go through several segmentation processes. The process was inefficient again for the same reasons and the accuracy wasn't decent enough. The earlier research[2] involved the usage of OpenCV library for extracting features out of the image such as contours and perform edge detection. This method was marginally better as it was providing good accuracy but had a few drawbacks. Methods have been gauged at different preconditions such as input image dimensions, distance between camera and vehicle, different albedos, types of number plate depending upon the country. Methods assessed for character segmentation work under several different pre-conditions such as input image dimensions, orientation of number plate. There is no consistency in the techniques used for character segmentation. Hence, to overtly arrive at conclusion which method actually demonstrates the maximum performance will be inappropriate. The introduction of YOLO gained more popularity and gave flexibility to use the model not only for images but for videos as well. It also helped us in identifying more than one number plates in one frame or image.

III. DATASET

Google Open Image Database was used as the dataset. Open Images is a collection of over 9 million images that have been annotated with image-level identifiers, bounding boxes, segmentation masks, and localised narratives. It has over 15 million bounding boxes for about 600 different items, making it the largest existing dataset.

The boxes have been manually drawn and bounded by professional annotators under Google to ensure accuracy and consistency. The videos are highly varied and have convolution scenes of a wide range of artefacts (8.3 per image on average). The dataset is divided into three parts: a training dataset (about 9 million images), a validation dataset (about 41 thousand images), and a test dataset (125 thousand images).[3] The images are annotated with image-level labels, bounding boxes for the image, segmentation masks, etc. The OIDv4 Toolkit[4] was used to extract the

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

images and bounding box information required for license plate detection and the YOLO Object Detection format for the image. The script created a dataset of specified amount of data. We used 1500 for training and 300 for testing. Once the execution of script was completed, we had pictures fetched from the database and corresponding bounding box information was stored in a text file. Figure 1[5] shows the example of the image and bounding boxinformation.

Figure 1: Sample image with bounding box information

IV. TECHNOLOGY STACK AND ALGORITHMS

ALGORITHM - YOLO

YOLO (You Just Look One) is a deep learning object recognition model built on an automated attribute extraction neural network. This is accomplished by applying a single neural network to the acquired complete image. The algorithm divides the image or frames captured from the video into bounding box regions and predicts bounding box and probability values. The weight of the bounding boxes is determined by the preference. The estimated probability is used to weight the bounding boxes. The model outperforms standard classifier- based structures in a variety of ways. At test time, it examines the whole image or the frames of the video, so its projections are guided by the image's global meaning. It also attempts to make predictions for a single network evaluation, as opposed to traditional systems like R-CNN, which require thousands of network evaluations for a single image[6]. This is the reason why it is over 1000 times faster than R-CNN and100 times faster than Fast R-CNN[6].

The YOLO is used as when there are multiple objects or multiple instances of objects and we want to detect each image, then approaches like sliding window or grid division won't work. The method is fast and accurate as we ONLY LOOK ONCE.

We usually divide the image we get into grids, but use other technique to improve accuracy. So, initially we decide the number of grids we want and now we do classification on each grid. The values are in the order [object present (1)/object absent (0), x_min, x_max, y_min, y_max] where x_min is in [0, 1] where 0 is the left most pixel, and 1 is the rightmost pixel in the image. Y coordinates go from the top pixelvalued at 0 to the bottom pixel viz. valued at 1. Now, for a grid containing object, we consider, top left of grid as (0, 0) and bottom right as (1, 1) for any grid we look at. Then using the canvas or convention, we find the co-ordinates. So, here we normalize that for each grid top left is (0, 0) and right is (1, 1). The question arises is what if the object is cut across multiple grids. In this case we first find the left of the image and assume it to be on the bottom grid, so the first grid will have vector with value 1 in object present/absent category. Now, we mark the left co-ordinate relative to that grid and in totality we look at the complete image instead of one grid and this is the magic of YOLO. Now, the left value can be greater than 1 as well and this will indicate that image is in 2 grids. Once, we train the model, model will understand that we don't have to look at part of object and with the convention we also need to look at the adjacent grids and will make the guess that if it sees the part of the object and should also look for the other part of the object. This is what we call tagging. The other problem which YOLO solved was the problem in prior researches, i.e., if detecting multiple objects. There is let's say more than one object in a single grid. One of the ways is that we reduce the size of the grid or increase the number of grids. Other way is specifying number of objects we can identify in the grid. So, if there are 2 objects, we could double the vector size. First half of the vector is for one object and another half for the second object. But, in order to be consistent, we increase the number of objects in a grid and also need to increase vector size for every grid. But here problem arises that which order is followed so you never know which is object 1 and which is object 2. Here we use concept of the anchor boxes. In anchor boxes we make assumptions to solve the problem in which we assume maximum number of objects in a grid we can have. So, we specify for example 2 as maximum occurrence. Now, we define what might be the shape of objects. Now, we allow rectangular and square boxes lets' say as anchor boxes. Now, person who trains the data and provides label should figure out which object fits into the anchor boxes. We can have anchor boxes of different aspect ratio and have convention on how to create vectors in which order to specify.

The advantageous part of the Google Dataset is that we don't have to actually care about all these details as the labels are set and in order to YOLO format. So, using the toolkit we can actually extract images and the information. But, internally here for multiple objects it does through colour of the boxes.

ALGORITHM - SEGMENTATION

Now we need to perform segmentation operation on the image we detected using the YOLO Model. The input is the image of the plate, we need to recognise the uni- character now. The outcome of this step serves as an input to the recognition process, so it is critical. Segmentation is one of the most critical processes for identification in any method of automatic reading of licence plates, since any other phase is either focused on it or contingent on it. The identification process would be wrong if the segmentation fails. Preliminary screening is needed to ensure proper segmentation. Now, we perform adaptive thresholding on the plate's value image to binarize it and reveal the

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

characters. The picture of a plate may have varying lighting conditions and albedos in different areas; in this situation, adaptive thresholding may be more appropriate for binarization. Following binarization, we must perform a bitwise not operation on the image to locate the connected components from which we can remove the characters (numbers and alphabets) from the licence plate. Figure 2 shows an example of the segmentation process done on the license plate and how each character in the number plate is extracted and how exactly the image is received and converted to the desired format so that recognizing the characters can be easy for the Google Tesseract.

Figure 2: License plate characters segmentation

ALGORITHM - CHARACTERRECOGNITION

The recognition phase is the last step in the development of the automatic license plate detection and recognition step. The recognition must make predictions of the character it tried recognizing from the output of the segmentation phase. The learning model that will be used for the recognition must be able to read the image and successfully recognize the characters in the number plate. The application used here for character recognition is Google Tesseract. Google Tesseract is implemented using PyTesseract[7] for Python implementation and hence by the implementation of the application, characters were recognized.

TECHNOLOGY- GOOGLEFIREBASE

Once the Deep Learning phase was completed, we performed storage of the data acquired and fetching the data for the exit operation. The data is in the form of text and timestamp when car entered is noted and stored in the database. The next time when the same car number is recognised, the timeris stopped and based on the price rate the bill is calculated and a token number is generated. The front-end layer for entering the car detail is entered and once selected the car detail is popped out of the database and based on exit time the price is calculated. The database used here was Google Firebase for real time database.

V. METHODOLOGY

The process starts with enabling the GPU environment so that your YOLOv4 system will be able to process detections over 100 times faster than CPU/TPU. Then we will need to clone darknet from AlexeyAB's repository, adjust the Makefile to enable OPENCV and GPU for darknet and then build darknet. YOLOv4 has been trained already on the COCO dataset which has 80 classes.[1] We will make use of these pretrained weights to run YOLOv4 on the pretrained classes and get detections. Once the darknet is trained, customisation according to the object is done.

In order to create a custom YOLOv4 detector we will need the Labeled Custom Dataset Custom .cfg file, obj.data and obj.names files and train.txt file and test.txtis optional. The change we found useful in .cfg file is having batch set to 64 and subdivisions set to 16 for ultimate results. If you run into any issues then you can change the subdivisions to 32.

The rest of the changes to the cfg is based on how many classes we are training our detector on which in our case is one, i.e., Vehicle registration plate. We set our max_batches to 6000 and steps = 4800, 5400. Since, we have only one class, we changed the classes = 1 in the three YOLO layers and filters = 18 in the three conv layers before the YOLO layers.

Variable Configuration:

width and height can be set to 416. These can be any multiple of 32, 416 is standard,

you can sometimes improve results by making value larger like 608 but can slow down the training process.

max_batches = (number of classes) * 2000. The max_batches cannot be less than 6000, so if you are training for 1, 2, or 3 classes it will be 6000, however detector for 5 classeswould need the max_batches be set to 10,000.

steps = 80%- 90% of max_batches

filters = (number of class/es + 5) * 3. So, if you are training for one class then your filters will be 18, but if you are training for4 classes then your filters should be 27)

Optional change: If you run into memory issues or your training process takes way more time, then in each of the three yolo layers in the cfg, you could change random from 1 to 0 to speed up training but this will slightly reduce accuracy of model. It will also help save memory if you run intoany memory issues as well.

Once the changes are done, we run the script with our custom labels and weights and we can now detect the license plate. Once, the license plate is detected, the recognition is done through GoogleTesseract.

After the recognition, the data operations are performed for further access of the data and the generation of token number for exit and the bill amount that has to be calculated is done.

VI. RESULTS

The accuracy which was recorded for the license plate detection was 90.06%.

```
for conf_thresh = 0.25, precision = 0.92, recall = 0.87, F1-sc
for conf_thresh = 0.25, TP = 224, FP = 20, FN = 34, average Io

IoU threshold = 50 %, used Area-Under-Curve for each unique Re
mean average precision (mAP@0.50) = 0.900615, or 90.06 %
Total Detection Time: 15 Seconds
```

Figure 3: Accuracy of the model

The YOLO Model was able to successfully detect the number plate and recognise the characters from the model. Figure 4 shows the model detecting number plate from the image and video and extracting the characters and returning the characters of the number plate.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

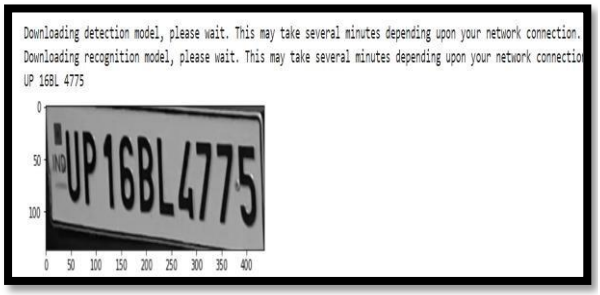

Figure 4: Number plate detection results

Figure 5 shows the database image where the timestamp and car number is shown.

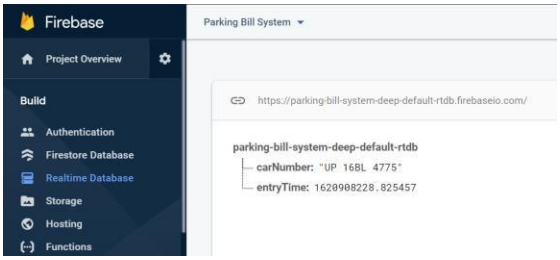

Figure 5: Database updation of the number platedetected

VII. CONCLUSION

The parking process in parking spaces are really time consuming and tiring and human interactions make it more time consuming. The research provided an architecture and solution to the utopia by introducing Artificial Intelligence and making the process swift and easy. It eliminates discrepancies. Instead, the system will make out the fare in proportion to the time and hence save accidental pay by customers. The queues are eliminated since you can access it, but won't be allowed to leave till the dues are paid.

Since there are no personnel at the gates, you don't spend on their salaries. It is a win-win for the organizations and customers. You could divert the workforce, using them as ushers for bettermanagement of the parking system.

Utilising the research for the parking space can save lot of money for the organisation and customers too. The use of Artificial Intelligence will give users a better experience than conventional parking methods.

VIII. FUTURE SCOPE

The future scope involves using the same methodology to get the stolen car information or get hold of defaulters who skip traffic signals. The project can be used to detect the number plate if a car crosses a certain point while the signal is red and even detect the license plate for stolen vehicles. Whenever there is a need to scale up, say parking for stadiums where thousand people will come at once, we can speed up the model using the cluster computing[8].

IX. REFERENCES

[1] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. You Only Look Once: Unified Real-Time Object Detection. In CVPR 2016, Pages 1-8, Year 2016

[2] Narote, P.R. Sanap and SP. License Plate Recognition Survey. Pages 2-8, Year 2010

[3] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasing, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, Vittorio Ferrari. The Open Images Dataset v4: Unified Image Classification, Object Detection and Visual Relationship Detection at Scale. IJSV 2020, Year 2020.

[4] Vittorio, Angelo, Toolkit to download and visualize single or multiple classes from the huge Open Images v4 Dataset. Github, Year 2018 [5]Google Open Images Dataset Picture, Google

[6] Shaoqing Ren, Kaiming HeRoss, B. Girshick, J. SunFaster. R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. Year 2015

[7] Anthony Kay. Tesseract an Open-Source Optical Character Recognition Engine. Linux Journal, Volume 2007, Issue 159, Year 2007

[8] Adarsh Chaturvedee, Karan Kantharia, Manan Nikam, Rujul Shringarpure, Disha Bhosle. Supercomputer using cluster computing. International Journal of Engineering Research & Technology (IJERT). ICIATE Special Issue-2017. ISSN: 2278-0181. Pages 1-3. Year 2017.