Auto Number Plate Recognition

Dipali Baviskar
Department of Computer Science and Engineering
Dr. Vishwanath Karad MIT World Peace University
Survey No, 124, Paud Rd, Kothrud
Pune, India
dipali.baviskar@mitwpu.edu.in

Shweta Choudhary
Department of Computer Science and Engineering
Dr. Vishwanath Karad MIT World Peace University
Survey No, 124, Paud Rd, Kothrud
Pune, India
shwetac074@gmail.com

Riya Danve
Department of Computer Science and Engineering
Dr. Vishwanath Karad MIT World Peace University
Survey No, 124, Paud Rd, Kothrud
Pune, India
riyardanve@gmail.com

Ketki Kinkar
Department of Computer Science and Engineering
Dr. Vishwanath Karad MIT World Peace University
Survey No, 124, Paud Rd, Kothrud
Pune, India
ketki.kinkar2000@gmail.com

Shivraj Patil
Department of Computer Science and Engineering
Dr. Vishwanath Karad MIT World Peace University
Survey No, 124, Paud Rd, Kothrud
Pune, India
shivrajrameshpatil@gmail.com

2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST) | 978-1-6654-9902-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/AIST55798.2022.10064725

***Abstract*—The development of an Auto Number Plate Recognition (ANPR) system using Yolo version 5 and ocr is shown in this work. Vehicle License Plate Recognition (VLPR) is a type of optical character recognition (OCR) that allows computer systems to read the registration numbers of vehicles from digital images for various applications like traffic control, security, access control to restricted areas, vehicle tracking, tracing of stolen vehicles, and identifying dangerous and reckless drivers on the road. Vehicle license plate detection and License Plate character recognition are the two main components of this system. There are various obstacles in vehicle license plate detection, such as complicated plate backgrounds, lighting inconsistencies, vehicle motion, and distance variations, for which edge detection analysis was investigated. 4165 images from a dataset were used in this project. Some license plates had legible text, while others had blurry characters and unclean spots. Yolo version 5 detected the plates with a precision of 0.996. The accuracy of character feature extraction and plate recognition was determined. The plates with the least amount of blur and discoloration were the most reliably retrieved and recognized, although the others yielded good results as well.**

Keywords—Number plate, YOLO V5, tesseract, OCR

I. Introduction

Automatic number plate recognition is developed to recognize car license plates and provide us with the output as vehicle number plate data using "optical character recognition" on photos. License plate recognition is utilized in traffic video monitoring. When compared to manual scenarios, our method improves license plate detection effectiveness [1] with a greater speed of recognition. The main goal of our approach is to train a deep learning model to extract a vehicle's license plate with an effective speed and greater precision.

The model was designed to work in any season, which could cause issues with data extraction and illumination. The suggested application is a robust and flexible solution with built-in alert capabilities that notify users of illegal detections in residential areas and roadways. The previous study looked at grayscale license plate detection and localization, recognition, and segmentation using computer vision [2].

Surveillance cameras are used to determine the frame quality or sharpness. It's difficult to view the number plates in dim light. The angle at which the camera is placed and its distance from the number plate are other key elements in detecting the object in the video. When the distance between the camera and the vehicle is not optimal, the quality of the number plate image gets affected, and so does the accuracy of number plate detection.

Following the introduction, in section II, we performed a thorough literature review, looking at a variety of articles and their methodologies. The project's approach, workflow, as well as data gathering, deep learning model training, and OCR engine selection, are covered in section III. In part IV, we'll talk about the results of the model used. Section V concludes the paper and looks ahead to future possibilities.

II. Literature Review

Numerous innovative approaches and AI algorithms have been discovered concerning automatic number plate identification and recognition. For instance, frame segmentation and image interpolation techniques to obtain high-quality images from a surveillance camera that records continuous footage of the road is developed by researchers [1]. For number plate detection, Faster R-CNN is a very successful algorithm. The contents of the number plates are also obtained via OCR. The suggested system can identify the number plate and display the vehicle's owner information with an overall accuracy of 99.1%.

A. Sasi et. al [2] suggest a web application for an intrusion detection system. They employed YOLOv5 for the object detection module to detect a number plate and then compared the tesseract OCR engine and a Microsoft Vision API to determine the character recognition technique to recognize the contents of the plate. The API was found to be the most

978-1-6654-9902-6/22/$31.00 ©2022 IEEE

effective in achieving improved results. To increase the performance of the entire ANPR process, the model's online deployment leverages asynchronous function calls and multi-threading.

A previous study [3] proposes a solution consisting of three steps: plate detection, character segmentation, and recognition. They employed edge clustering, the Maximally Stable Extreme Region (MSER) detector, and the LDA classifier for each step.

By pre-processing the photos into gray-scale, P. Prabhakar et al. [4] proposed a highly effective technique for localizing the license plate from the image, segmentation of the image to extract the plate and recognizing the characters inside that localized plate. For edge detection, Hough lines are determined using the Hough transform, and then character recognition is conducted using OCR resulting in a reliable real-time system.

A modified version of the Ant colony optimization (ACO) is developed [5] using an algorithm for edge detection, followed by an algorithm for extraction of characters and segmentation that uses the Kohonen neural network for identification based on position and dimensions of characters, and finally, a character recognition algorithm that uses learning-based classification and SVM. They could detect number plates with a 94% accuracy using the approaches indicated above.

A smart car service application is included [6], which uses ANPD to recognize cars from their photos and then extract and save servicing data based on the car's recognized number plate. The PHP Laravel 5.4 framework and PHPMyAdmin are used as backend services, while JSON is used as a communication protocol in this deployment architecture. The article is useful for studying the deployment and end-to-end procedure of the system.

Another work [7] proposes Gabor filtering for character recognition in grayscale images. In addition to the standard procedures of edge detection, character segmentation, and OCR, the researchers applied template matching for recognition acknowledgment with satisfactory results.

M. Samantaray et al. [8] worked on the OCR of car number plates utilizing python programming and the OpenCV package in particular. Upload image, pre-process image, number plate identification, character segmentation, and finally, OCR model to recognize car number are the steps in the system workflow. They were effective in greatly decreasing noise in the output.

The research of N. Duan et. al [9] describes a complete system for number plate recognition that uses CNN algorithms. The dataset used contains approximately 6000 photos of different quality, yielding a 99% overall accuracy. In comparison to standard SVM, Multi-AlexNet showed to be the most successful for OCR. In both dim and fuzzy backgrounds, the end-to-end model for recognition using CNN can successfully recognize the license plate. This technology offers quick execution as well as a reduction in overall output noise.

Another interesting study [10] shows the use of Gradient-based segmentation, which analyzes changes in image intensity levels to detect where the license plate is located. To acquire edge points and identify license plate location, Sobel magnitude is utilized as an edge detection technique. The gradient-based segmentation technique provided more accuracy and could handle photos shot in shadows or low light, as well as those with incorrect orientation.

A. B. Mohammad et. al [11] examined ANPR applications in the context of vehicle security. The following is the process workflow: camera input image, pre-process, number plate centralization, character segmentation, character recognition, and output characters. This was done using 600 different photos of varied quality. Python's pytesseract module was used to do the processing.

This research helped us to go through various algorithms and that's when we came across the latest version of yolo. The main advantage of using yolo v5 was that it did not contain the use of darknet and it gave us results faster than any other algorithm for a larger dataset with greater precision in recognizing the custom objects defined.

III. Research Methodology

The suggested methodology makes use of YOLO version 5, which was released on May 18, 2020, shortly after YOLOv4 by Glenn Jocher utilizing the Pytorch framework. YOLOv5 is a set of object identification architectures and models that have already been trained on the COCO dataset.

YOLO (You Only Look Once) is a state-of-the-art object detector that can accurately and quickly identify objects. Instead of being a branch of the original darknet, YOLOv5 is constructed in Pytorch, which sets it apart from earlier editions. Similar to the YOLO v4, the YOLO v5 has a CSP backbone and a PA-NET neck. YOLO v5 is significantly rapid and more efficient than YOLO v4, with accuracy comparable to the YOLO v4 test. Two significant improvements include mosaic data augmentation and auto-learning bounding box anchors. Fig. 1 shows the flow we followed for using yolov5 as the approach for detection of number plate.

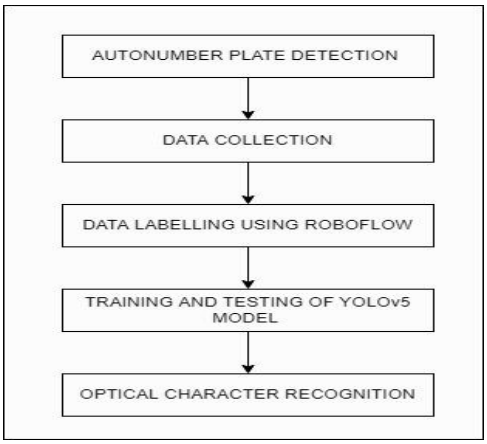

Fig.1. Flow diagram prepared for proposed algorithm

A. Preparing the dataset

With the advent of YOLOv5 by Ultralytics, the YOLO family of object detection models has become even stronger. The quick and accurate deployment of unique computer vision models is made possible by Roboflow.

Object detection and classification models are supported by Roboflow. It can be used to: annotate photos or upload existing annotations; existing VOC XML annotations can be converted to COCO JSON annotations; check for in-frame labels; modify resize, grayscale, auto-orientation, and contrast. Enhances your training data by clipping, shearing, blurring, and adding random noise to images; develops custom YOLOv3 implementations and annotation formats and quickly analyzes the quality of your dataset with your team. Share version-controlled datasets, easily and train models with a single click.

After creating an account in roboflow, we created a workspace. Fig. 2 is a screenshot from roboflow interface which shows how to create a new project in roboflow.

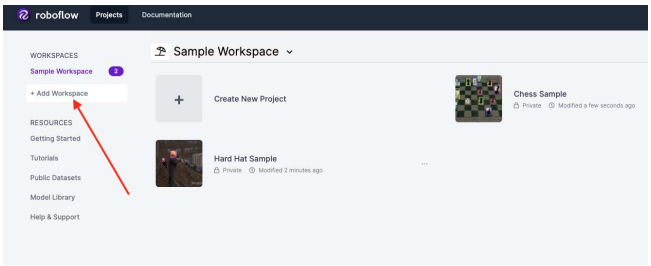

Fig.2. Roboflow interface screenshot to create new project

After creating a new Workspace, we chose a plan for the Workspace. Fig. 3 is a screenshot from roboflow interface which shows all plans available and we went with sandbox as it was freely available.

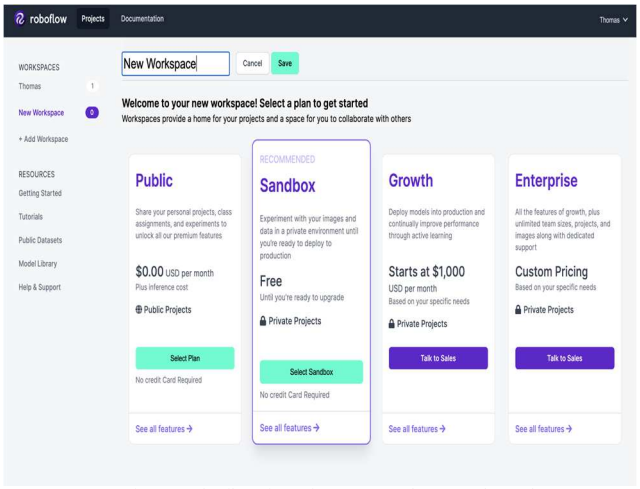

Fig.3. Roboflow interface screenshot to select plan

After choosing the plan, we upload our dataset and choice of algorithm for annotation and labels. The dataset for Auto Number Plate detection was uploaded in the following steps. The dataset consists of a total of 4165 images which were split into train, test, and validate sets.

Fig. 4 is the representation of distribution of 4165 images in training, validation and testing set.

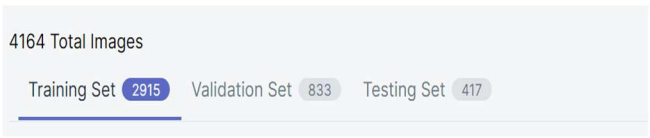

Fig.4. Dataset division into training, validation and testing set

Roboflow also helps in annotating the dataset and creating labels of the data for training and testing purposes. Fig. 5 shows how the annotations look in roboflow.

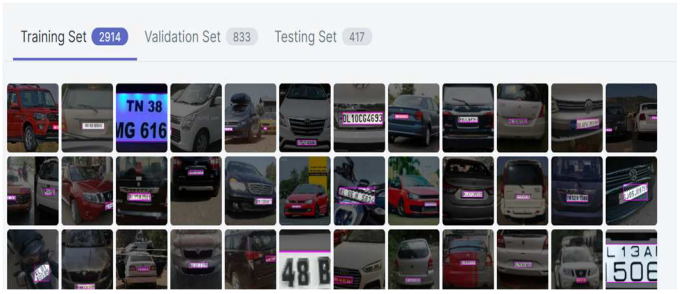

Fig.5. Roboflow labeling and annotating the dataset

Once the annotation is done we exported the annotated data and choose the algorithm, here Yolov5 pytorch is selected and you can download the data and export it to your notebook for training and testing purposes. Fig. 6 is a screenshot from roboflow which pops up after annotation is done and it is used to export the annotated data into a format as soon in the figure.

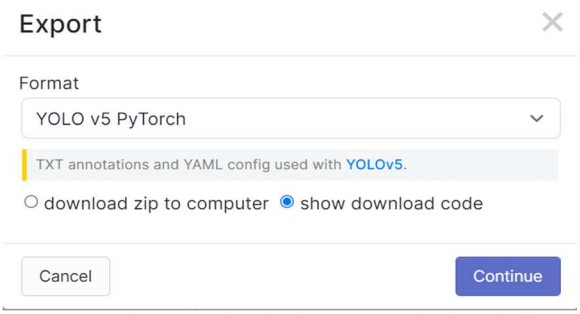

Fig.6. Exporting the dataset generating in Roboflow

B. *Cloning Yolov5 Model*

Cloning the yolov5 model using the export option from roboflow and then using that code in the ipython notebook.

C. *Installing Roboflow in your notebook*

Installation of roboflow packages and API keys required for it using the exported code in a notebook.

D. *Installing the dataset into the notebook*

Once we have exported and installed roboflow in the notebook, the next step is to install the created dataset in the notebook using roboflow credentials.

E. *Checking the labels created in the notebook*

Fig. 7 shows the labels created after annotation for the model yolov5

```
names:
- license_plate
- plate
nc: 2
train: YOLO-1/train/images
val: YOLO-1/valid/images
```

Fig.7. Labels identified by python from the dataset annotations

F. Check the loaded model

Loading the yolov5 model and its architectural structure. Fig. 8 shows the architecture of yolov5 used in number plate recognition.

```
# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],  #
9
  ]
```

```
# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]],  # 13
   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]],  # 17 (P3/8-small)
   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, BottleneckCSP, [512, False]],  # 20 (P4/16-medium)
   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]],  # 23 (P5/32-large)
   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4,
P5)  ]
```

Fig.8. Yolov5 architecture

G. Train the Model

Once the model is imported in the notebook, the training of the model is required using the train data and weights from the custom yolov5 model generated. As mentioned in fig. __ it took 2.456 hours to run 100 epochs using Cuda in collaboration with google collab for yolov5 which is faster than the other algorithms that were discussed in the above literature survey. Fig. 9 shows the labels recognized by yolov5 during the training of the model on train data along with its precision, recall and mAP values.

```
Class       Images  Targets P     R     mAP@.5  mAP@.5:.95:100% 27/27[00:11<00:00,2.38it/s]
all         833     963     0.99  0.918 0.975   0.794
license_plate 833   900     0.999 0.995 0.997   0.959
plate       833     63      0.981 0.841 0.953   0.629
Optimizer stripped from runs/train/yolov5s_results/weights/last.pt, 14.7MB
Optimizer stripped from runs/train/yolov5s_results/weights/best.pt, 14.7MB
100 epochs completed in 2.456 hours.
CPU times: user 1min 48s, sys: 16.3 s, total: 2min 4s
Wall time: 2h 27min 56s
```

Fig.9. Training of data in python using yolov5 and cuda

H. Evaluating the model performance

With the help of the tensorboard, we were able to get the metrics of the training model in a visual format. Fig. 10 shows a graphical representation generated while training the yolov5 model using tensorboard to showcase it.

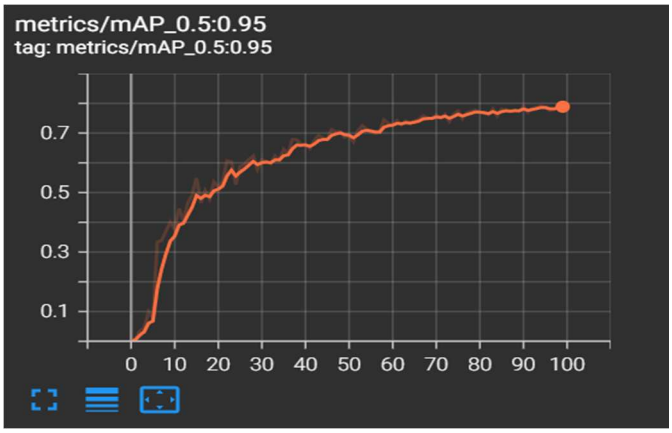

Fig.10. Graphical representation of the training model

I. Detecting the model

Using the detect.py from the yolov5 model we detect the plates and recognize the plates by changing some codes of detect.py for custom boundaries. This was executed successfully in 16.715s.

J. Checking the output generated from the model

Fig. 11 shows the recognized number plates identified.

Fig.11. Plates identified by yolov5 after using detection model

K. Testing the Model

Using test.py to test the plates detected from the yolov5 model. Fig. 12 shows the results of precision, recall and mAP on the labels while testing the dataset using test data.

```
Class       Images Targets P     R     mAP@.5 mAP@.5:.95: 100% 27/27[00:25<00:00,1.04it/s]
all         833    963     0.919 0.872 0.919  0.741
license_plate 833  900     0.983 0.998 0.996  0.95
plate       833    63      0.854 0.745 0.841  0.532
Speed: 23.5/2.0/25.5 ms inference/NMS/total per 640x640 image at batch-size 32
Results saved to runs/test/exp
```

Fig.12. Testing the model on test data and saving results

L. Using OCR on the plates detected

For this paper we used two OCR algorithms, namely pytesseract and easy ocr and we saw that easyocr was giving a faster result than pytesseract and better accuracy.

IV. Results

After training the yolov5 model we look at the performance matrix and graphs generated while training it.

Fig. 13 shows the metrix generated after testing the data using tensorboard for visualization.

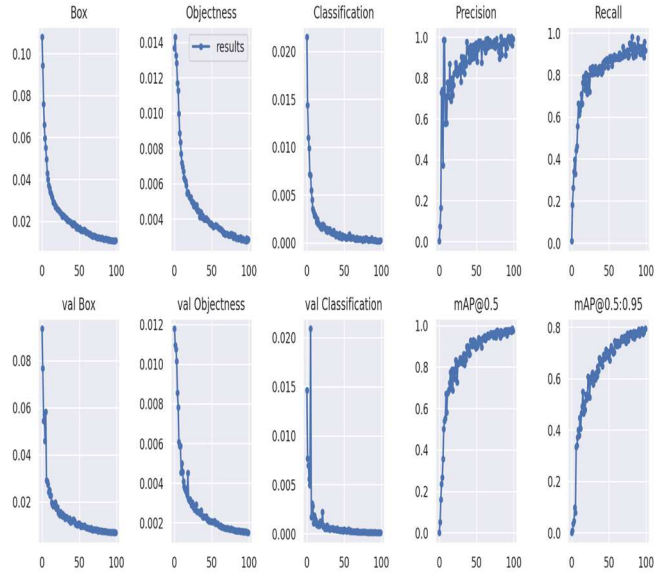

Fig.13. Metrix on object identification after testing the data

Fig. 14 shows the license plate identified on the test data along with the confidence of identification along the boundaries.

Fig.14. Identified plates with the confidence of identification

The OCR results were fairly reliable which is shown in the Fig. 15 where the comparison of pytesseract and easyocr with the original text from the plates in the three columns.

1	104_jpg.rf.6479f1f707612d1e06374f2b17f19ad8.jpg	MH 02BM5048	MH 02 BM5048	VH02BM5048
2	106_jpg.rf.03c8a7e0eb73ab4b6841a817f23eb68e.jpg	HR26DK6475	HR26DK6475	HR26DK6475
3	116_jpg.rf.25074e5c713642d83d8fc86d9f20c426.jpg	GJ05H2501	GIO5 JH750 Fi	GIO5JH750Fi

Fig.15. Comparison of 2 OCR models - pytesseract and easyocr

The testing model generated the following graph results. Fig. 16, Fig. 17, Fig. 18 and Fig. 19 represent recall, confidence, F1 score and prediction, respectively, generated while testing the data on yolov5 model.

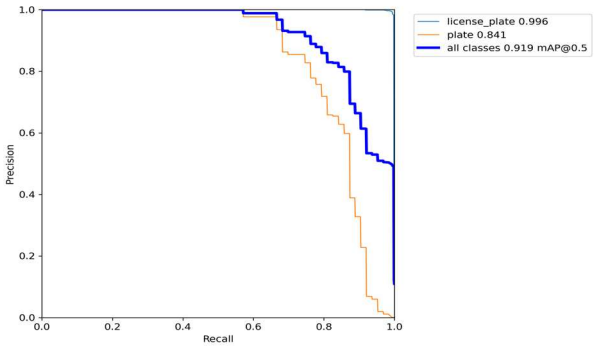

Fig.16. Recall of the results from testing model

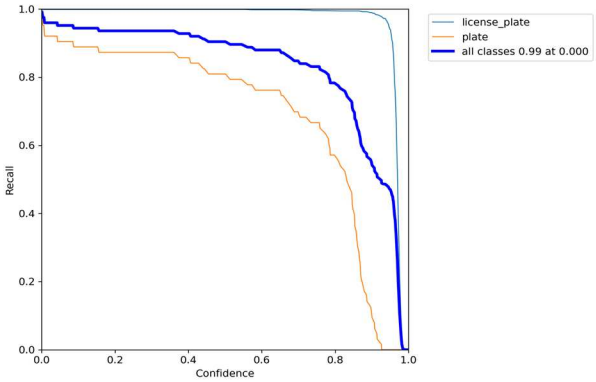

Fig.17. Confidence of the results from testing model

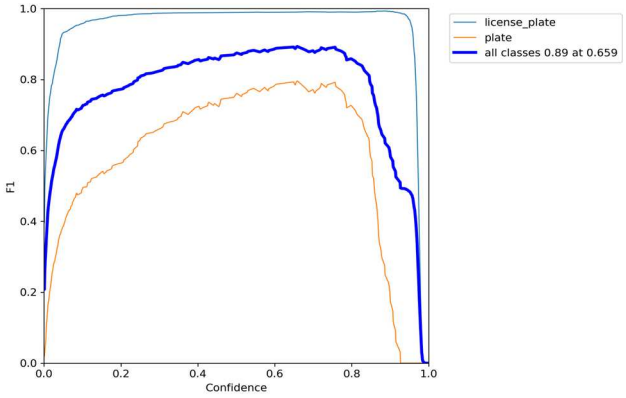

Fig.18. F1 score of model after testing

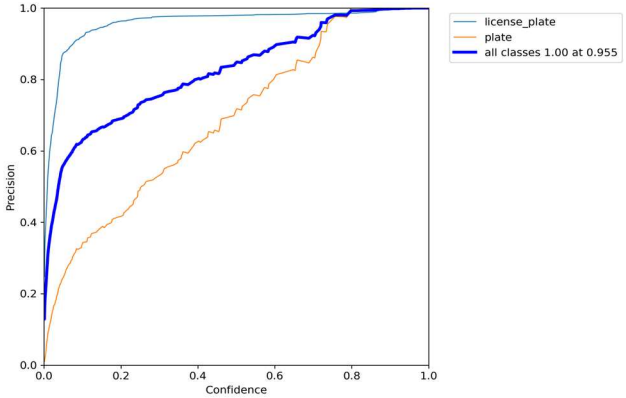

Fig.19. Precision of the model after testing

V. Conclusion

Due to the size of the dataset, after some considerations, we used yolov5 which is the most recent version of Yolo. It provided more accuracy, faster processing power, and shorter execution times because it uses Cuda in conjunction with google collab. It is known to work best for object detection, and it is easier to train bespoke models on it than on YOLO's prior versions because it doesn't require a darknet or any support model to run. Additionally comparing Yolov5 with yolov4, we found that Yolov5 is 88% smaller and 180% faster than yolov4.

Easy-ocr is the best OCR algorithm since it operates on GPU and is faster and more accurate than pytesseract. With a precision-recall of 0.996 and confidence of 0.89, this model was able to predict and crop the detected class - plate.

VI. Future Work

The above model was built on a dataset with photographs, but it can also be applied to live images acquired by the camera in the future. Its applications range from traffic surveillance to insurance claims. Insurance firms can use the results to recognize the license plate and match it to the owner using their database. It can be used to record traffic or the vehicle that breached the speed limit. We assumed that all number plates in different nations have the same format which may cause the OCR model to perform poorly. The ocr model characters are sequenced, thus it won't recognize the format of other countries, for the time being. Future implementations might also involve increasing the resolution of the photos using different deep learning models to improve OCR outcomes. However, this can be adjusted in the future to improve the solution.

References

[1] N. P. Ap, T. Vigneshwaran, M. S. Arappradhan and R. Madhanraj, "Automatic Number Plate Detection in Vehicles using Faster R-CNN," 2020 International Conference on System, Computation, Automation and Networking (ICSCAN), 2020, pp. 1-6, doi: 10.1109/ICSCAN49426.2020.9262400.

[2] D. U. Nayak, A. P. Mohite, P. P. Nair and P. J. Bide, "SecurePark: Vehicle Intrusion Detection System," 2021 Asian Conference on Innovation in Technology (ASIANCON), 2021, pp. 1-6, doi: 10.1109/ASIANCON51346.2021.9544923.

[3] G. Hsu, J. Chen and Y. Chung, "Application-Oriented License Plate Recognition," in IEEE Transactions on Vehicular Technology, vol. 62, no. 2, pp. 552-561, Feb. 2013, doi: 10.1109/TVT.2012.2226218.

[4] P. Prabhakar, P. Anupama and S. R. Resmi, "Automatic vehicle number plate detection and recognition," 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014, pp. 185-190, doi: 10.1109/ICCICCT.2014.6992954.

[5] A. Sasi, S. Sharma and A. N. Cheeran, "Automatic car number plate recognition," 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), 2017, pp. 1-6, doi: 10.1109/ICIIECS.2017.8275893.

[6] R. M. Sferle and E. V. Moisi, "Automatic Number Plate Recognition for a Smart Service Auto," 2019 15th International Conference on Engineering of Modern Electric Systems (EMES), 2019, pp. 57-60, doi: 10.1109/EMES.2019.8795201.

[7] B. Pechiammal and J. A. Renjith, "An efficient approach for automatic license plate recognition system," 2017 Third International Conference on Science Technology Engineering & Management (ICONSTEM), 2017, pp. 121-129, doi: 10.1109/ICONSTEM.2017.8261267.

[8] M. Samantaray, A. K. Biswal, D. Singh, D. Samanta, M. Karuppiah and N. P. Joseph, "Optical Character Recognition (OCR) based Vehicle's License Plate Recognition System Using Python and OpenCV," 2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA), 2021, pp. 849-853, doi: 10.1109/ICECA52323.2021.9676015.

[9] N. Duan, J. Cui, L. Liu and L. Zheng, "An End to End Recognition for License Plates Using Convolutional Neural Networks," in IEEE Intelligent Transportation Systems Magazine, vol. 13, no. 2, pp. 177-188, Summer 2021, doi: 10.1109/MITS.2019.2898967.

[10] G. Kumar, A. Barman and M. Pal, "License Plate Tracking using Gradient based Segmentation," TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), 2019, pp. 1737-1740, doi: 10.1109/TENCON.2019.8929688.

[11] A. B. Mohammad, M. Suneetha and M. A. Muqeet, "An Efficient Method for Vehicle theft and Parking rule Violators Detection using Automatic Number Plate Recognition," 2022 2nd International Conference on Artificial Intelligence and Signal Processing (AISP), 2022, pp. 1-4, doi: 10.1109/AISP53593.2022.9760556.