TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING

Khwopa College of Engineering
Libali, Bhaktapur
Department of Computer Engineering

A REPORT ON
NEPALI NUMBER PLATE RECOGNITION SYSTEM USING YOLO AND CNN

Submitted in partial fulfillment of the requirements for the degree

BACHELOR OF COMPUTER ENGINEERING

Submitted by

Anish Shilpakar	KCE075BCT008
Anjaan Khadka	KCE075BCT009
Sachin Manandhar	KCE075BCT035
Sudip Shrestha	KCE075BCT046

Under the Supervision of
Er. Bindu Bhandari

Khwopa College of Engineering
Libali, Bhaktapur
March 09, 2022

Certificate of Approval

The undersigned certify that the sixth semester project entitled **"Nepali Number Plate Recognition System Using YOLO and CNN"** submitted by Anish Shilpakar, Anjaan Khadka, Sachin Manandhar, Sudip Shrestha to the **Department of Computer Engineering** in partial fulfillment of requirement for the degree of **Bachelor of Engineering in Computer Engineering**. The project was carried out under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled, bona fide and ready to undertake any commercial and industrial work related to their field of study and hence we recommend the award of Bachelor of Computer Engineering degree.

.....
Er. Bindu Bhandari
(Project Supervisor)

.....
Er. Dinesh Man Gothe
Head of Department
Department of Computer Engineering, KhCE

Copyright

The author has agreed that the library, Khwopa College of Engineering may make this report freely available for inspection. Moreover, the author has agreed that permission for the extensive copying of this project report for scholarly purpose may be granted by supervisor who supervised the project work recorded here in or, in absence the Head of The Department where in the project report was done. It is understood that the recognition will be given to the author of the report and to Department of Computer Engineering, KhCE in any use of the material of this project report. Copying or publication or other use of this report for financial gain without approval of the department and author's written permission is prohibited. Request for the permission to copy or to make any other use of material in this report in whole or in part should be addressed to:

Head of Department
Department of Computer Engineering
Khwopa College of Engineering
Liwali,
Bhaktapur, Nepal

Acknowledgement

Firstly, we would like to express our deep gratitude to our supervisor, Er. Bindu Bhandari for her supervision, encouragement, suggestions and encouragement during course of this project. We are also deeply indepted to Er. Dinesh Man Gothe, Er. Sajit Pyakurel and Er. Sandesh Lochan Shrestha for their valuable suggestions and encouragement.
Also, we are thankful to Niranjan Bekoju for this advice and guidance in this project. We are also grateful to Manisha Adhikari for her suggestions and help for collecting number plate dataset. We gratefully acknowledge help of Sushant Tha, Shyamron Dongol and friends during dataset collection.
We would also like to thank all our classmates who helped us during the course of the project. Finally, we would like to acknowledge everyone who played a role directly or indirectly in this project.

Anish Shilpakar	KCE075BCT008
Anjaan Khadka	KCE075BCT009
Sachin Manandhar	KCE075BCT035
Sudip Shrestha	KCE075BCT046

Abstract

Generally, every vehicle is identified by a number plate which is attached on rear and front side. Number Plate Recognition (NPR) is a practical application of deep learning and computer vision and, can be used to detect and recognize the number plate using static image Datasets. In this paper, we developed a Nepali Number Plate Recognition(NNPR) system which can extract the license plate number of vehicles passing through a given location using YOLO CNN algorithm.The working mechanism of Nepali Number Plate Recognition system involves four main stages, and they are Number Plate Detection, Character Detection and Segmentation, Character Recognition and Data recording.The plate detection performs cropping of LP and further fed to Character detection. The character detection and segmentation then segments the characters. The segmented characters are further passed to image processing and feed to Character recognition model where validation of proper number plate constraints are performed. Thus, the accepted recognized character are stored in database and further displayed on the user interface.The benefit of this project would be reduction in manual efforts in manual checking of each vehicle and also maintaining manual records of the same vehicle which can be further applicable in different sectors.All this is done in python because resources are easily available and provides better result in comparison to others. The number plate condition, camera quality, memory limitations, environmental conditions and software/hardware based constraints may undermine its performance but with all these constraints this project aims to detect the number plate and character recognition in real time with better efficiency and accuracy. Our system currently has mAP score of approximately 98% for number-plate detection for seen dataset and 70% for unseen data-set, while average accuracy for character recognition of approximately 95% is observed. Also, this system can give real-time results for images however for video real-time results may not be obtained due to lack of proper resources.

Keywords:*ANNPR, ALPR, YOLO, CNN, Image Processing, Intersection over union*

Table of Contents

Certificate of Approval i
Copyright ii
Acknowledgement iii
Abstract iv
Contents vii
List of Tables viii
List of Figures ix
List of Abbreviation x

1 Introduction **1**
1.1 Background Introduction 1
1.2 Motivation 1
1.3 Problem Definition 1
1.4 Objectives 2
1.5 Scope and Applications 2
1.6 Limitations 2

2 Literature Review **3**

3 Requirement Analysis **6**
3.1 Software Requirement 6
3.2 Hardware Requirement 6
3.2.1 Graphical Processing Unit 6
3.2.2 Computer 6
3.3 Functional Requirement 7
3.4 Non-Functional Requirement 7
3.4.1 Reliability 7
3.4.2 Maintainability 7
3.4.3 Performance and Accuracy 7
3.4.4 Frames Per Second 7
3.4.5 Portability 7
3.4.6 Availability 7

4 Feasibility Study **8**
4.0.1 Economic Feasibility 8
4.0.2 Technical Feasibility 8
4.0.3 Operational Feasibility 8
4.0.4 Time Feasibility 8

5 Methodology **9**
5.1 Software Development Approach 9
5.2 ClickUp as Project Management Tool 9
5.2.1 Product Backlog 10
5.2.2 Sprint 1 11
5.2.3 Sprint 2 11

5.2.4 Sprint 3 12
5.3 Overall Phase Followed 13
5.3.1 Planning Phase 13
5.3.2 Development Phase 13
5.3.3 Integration 13
5.4 Work Breakdown 14

6 System Design and Architecture 15
6.1 System Block diagram 15
6.2 Black Box View 16
6.3 Entity Relationship Diagram 16
6.4 Sequence Diagram 17
6.5 Activity Diagram 18
6.6 Description of Working Flow of System 19
6.6.1 Image Validation 20
6.6.2 YOLOv4 License Plate Detection 20
6.6.3 YOLOv4 Tiny Character Detection 20
6.6.4 License plate and Character Cropping/Segmentation 20
6.6.5 Image Processing 21
6.6.5.1 Gray Scale Conversion 21
6.6.5.2 Otsu's Binary Thresholding 21
6.6.5.3 Median Filter(Blur) 21
6.6.5.4 Image Resizing 21
6.6.6 CNN Character Recognition 22
6.6.7 Output Number Validation 22
6.6.8 Data Recording 22
6.7 Dataset 22
6.7.1 Dataset Collection 22
6.7.2 Data preprocessing 23
6.8 Model Creation 24
6.8.1 YOLOv4 models 24
6.8.2 CNN model 26
6.9 Model Evaluation 28
6.9.1 Precision 28
6.9.2 Recall 28
6.9.3 IoU - Intersection over Union 28
6.9.4 mAP - mean Average Precision 29
6.9.5 F1-score 29
6.9.6 Epoch, Iteration, Batch and Subdivisions 30
6.9.7 Max Batch 30
6.9.8 Dropout 30
6.9.9 Learning rate, Optimizer and Loss Function 31
6.9.10 Accuracy 31
6.9.11 Confusion Matrix 31

7 Result and Discussion **32**
7.1 Desktop Outcomes . . . 32
7.2 Unit Test . . . 32
7.2.1 YOLOv4 License Plate Detection(LPD) Unit . . . 32
7.2.2 YOLOv4-tiny Character Detection Unit . . . 33
7.2.3 CNN Character Recognition Unit . . . 34
7.2.4 Web App Unit . . . 34
7.3 Integration Testing . . . 35
7.4 Model Evaluation Result . . . 36
7.4.1 YOLO Model Evaluation . . . 36
7.4.2 CNN Model Evaluation . . . 37

8 Conclusion and Future Enhancements **39**
Bibliography . . . 40

Appendix **41**
A Assistance for Running the Project . . . 41
A.1 Automatic Nepali Number Plate Recognition . . . 41
A.1.1 Clone Git Project . . . 41
A.1.2 Installation Required . . . 41
A.1.3 Setting up in PC . . . 41
A.1.4 Running the Program . . . 41
B Snapshot . . . 42
B.1 ClickUp . . . 42
B.1.1 Create task . . . 42
B.1.2 Sprint-1 Chart . . . 43
B.1.3 Sprint-2 Chart . . . 43
B.1.4 Sprint-3 Chart . . . 43
B.2 Image Processing . . . 44
B.2.1 Different Outcomes in Image Processing Pipeline . 44
B.3 YOLO Model Visualization . . . 45
B.3.1 YOLOv4 Number-plate detection . . . 45
B.3.2 YOLOv4-tiny Character detection . . . 45
B.4 CNN Model Visualization . . . 46
B.4.1 Plot for Accuracy . . . 46
B.4.2 Plot of Loss . . . 46
B.5 Web App . . . 47
B.5.1 Home Page For Web App . . . 47
B.5.2 About Page For Web App . . . 47
B.5.3 Upload Form For Web App . . . 48
B.5.4 Detection Page For Web App . . . 48
B.6 Unit Testing . . . 49
B.6.1 License Plate Detection . . . 49
B.6.2 Character Segmentation . . . 50
B.6.3 Character Recognition . . . 51
B.7 Integration Testing . . . 51

List of Tables

2.1 Review Matrix with Research Papers, limitations and our improvements. 3

5.1 Work Breakdown Structure 14

6.1 Data For License Plate Detection using YOLOv4. 23
6.2 Data For Character Detection using YOLOv4 tiny. 23
6.3 Data For Character Recognition using CNN. 23

7.1 LPD Model test scenario and requirements 32
7.2 Unit Test Case for LPD Module 33
7.3 Character Detection Model test scenario and requirements 33
7.4 Unit Test Case for Character-detection Module 33
7.5 Character Recognition test scenario and requirements 34
7.6 Unit Test Case for Character Recognition Module 34
7.7 Web App test scenario and requirements 34
7.8 Unit Test Case for Web app 35
7.9 Integrated System Test scenario and requirements 35
7.10 Unit test case for complete/integrated model 35
7.11 YOLO number-plate detection model performance table 36
7.12 Average Accuracy for Individual Characters for unseen data 37
7.13 Confusion Matrix for CNN Character Recognition model for individual characters 37

List of Figures

5.1 Software Development Approach 9

6.1 Block Diagram of ANNPR 15
6.2 Black Box View of ANNPR 16
6.3 ER diagram of ANNPR 16
6.4 Sequence Diagram of ANNPR 17
6.5 Activity diagram of ANNPR 18
6.6 Flowchart showing working flow of system 19
6.7 Image Processing Pipeline 21
6.8 Generation of Custom YOLOv4 model for License Plate Detection 25
6.9 Generation of Custom YOLOv4 model for Character Detection . . 25
6.10 Generation if custom CNN model for Character Recognition 26
6.11 Architecture for CNN model 27
6.12 Diagram Showing Precison, Recall and IoU 29
6.13 Confusion Matrix 31

7.1 Confusion matrix for YOLO LPD model 36
7.2 Confusion Matrix for CNN character recognition model 38

8.1 Steps in Image Processing 44
8.2 Test case for Number-plate 49
8.3 Test case for character separation 50
8.4 Test case for character recognition 51
8.5 Test for image with clearly visible number plate 51
8.6 Test for image with clearly visible number plate of ration 4:3 52
8.7 Test for image with multiple number plates 52
8.8 Test for video with multiple cars passing 52

List of Abbreviation

Abbreviations	**Meaning**
ALPR	Automatic License Plate Recognition
ANPR	Automatic Number Plate Recognition
ANNPR	Automatic Nepali Number Plate Recognition
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DCNN	Deep Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
FCN	Fully Convolution Network
FN	False Negative
FP	False Positive
GPU	Graphical Processing Unit
IoU	Intersection over Union
ITS	Intelligent Transportation System
LP	License Plate
LPD	License Plate Detection
LPR	License Plate Recognition
LSTM	Long Short-Term Memory
mAP	mean Average Precision
ML	Machine Learning
NLP	Natural Language Processing
OKM	Optical K-Means
RDBMS	Relational Database Management System
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RPG	Rapid Population Growth
SDLC	Software Development Life Cycle
SQL	Structured Query Language
SSD	Single Shot Multibox Detector
STN	Spatial Transformer Network
SVM	Support Vector Machines
TP	True Positive
TPU	Tensor Processing Unit
WPOD-NET	Warped Planar Object Detection Network
YOLO	You Only Look Once

Chapter 1
Introduction

1.1 Background Introduction

Nepali Number Plate Recognition System is a system designed to detect number plates from vehicles and then to recognize the Nepali number written in those detected number plates. Number Plate Recognition (NPR) system was first in 1976 at the Police Scientific Development Branch in Britain [1].This system uses the Optical character Recognition (OCR) on image taken by camera. In the last few years, Number Plate Recognition (NPR) has been one of the useful approaches for vehicle monitoring and management.In Nepal, there is high increase in numbers of vehicles. Due to insufficient spaces and techniques, there is hazard in parking management system. This Number Plate Recognition (NPR) can be a useful system to eradicate those problems. Moreover, this system supports for huge traffic management in major cities of South Asia like Kathmandu in context of Nepal. Historically, This type of system were used in unidentified crimes but later, it was proposed in the traffic management along with road management. So, in this project we have come up with a system that uses object detection to detect license plates and then using character recognition to recognize the number which will later be stored.

1.2 Motivation

Currently, There has been huge problems with traffic management and control in major cities of Nepal. Besides, few project on Nepali Number Plate Detection has been done. This is challenging due to the insufficient datasets and resources like high-end computing device. Several problems regarding parking and huge traffic management are highly observed, so this system is able to diminish those problems as far as noticeable. Further, it will help in smart parking and Intelligent Transport System(ITS).

1.3 Problem Definition

With the rapid increment of vehicles in major cities of Nepal over years, it becomes impossible for vehicle management to be fully managed and monitored by humans.As traffic control and vehicle owner identification has become one of a major problem, we made an effort to combat these problems. It becomes difficult to identify vehicle owner who violate traffic rule, non-taxed and non-insurance vehicles and who drives too fast, it is not possible to catch and penalize them. Also, it's becoming difficult to manage parking area and parking charge for each vehicle manually, specially in places like shopping centers, cinemas and parking zones.In

recent years, there has been a rise in vehicle theft as well as illegal activities sing vehicle.Considering all these problems in the context of Nepal, we aim to develop the system which helps to identify vehicle owner and vehicle's detail, track stolen cars, monitor and control traffic and parking and hence, help in border security. According to the problems stated earlier, we have prepared a system in this paper which aid in recognizing and storing number plate details which can help in vehicle management.

1.4 Objectives

The main objectives of this project are:

- To identify Number Plate automatically.
- To recognize Devanagari characters in Nepali number plates.
- To record the identified number plates.
- To be robust enough to handle different scenarios.

1.5 Scope and Applications

The major scope of the project is to assist the user to record the number plates detected in an image or video, and this system can have the following applications:

- Traffic Management
- Security
- Parking Management

1.6 Limitations

Although, we tried to make our system as accurate as possible, we couldn't achieve some of our goals, mainly due to lack of resources like GPU, dataset and time. So our system currently has the following limitations:

- Our system can only detect number plates but cannot recognize number plates properly if the distance between camera and vehicle is very far.
- As we have only used 13 characters(0-9, ba, pa, cha) due to lack of proper dataset so our system cannot recognize if any other characters are present in the number plate.
- When testing our system in video, we achieved very low fps which takes more processing time even for a small length video, so our system is not real-time for video. This is mainly because we are using our system in CPU, as we don't have a proper GPU for running our system.

Chapter 2

Literature Review

Automatic Number Plate Recognition system is in state of research and development but in context of Nepal due to lack of proper dataset only small number of researches have been made. In the past, the systems proposed for ALPR were mostly based on image binarization or gray-scale analysis. However, with the rise of DL, Automatic Number Plate Recognition(ANPR) systems have been developed using deep learning techniques like Support Vector Machine(SVM), Convolutional Neural Network(CNN), Deep Neural Network(DNN) e.t.c. In this project, we have proposed a Nepali Number Plate Recognition System using YOLO and CNN.

Table 2.1: Review Matrix with Research Papers, limitations and our improvements.

S.N.	**Title**	**Limitations**	**Our Improvements**
1	Automatic Nepali Number Plate Recognition with Support Vector Machines [2]	Low accuracy, license plate detection using contours method which resulted in many false positives, use of SVM, less training data	Used YOLO for license plate detection which resulted in higher accuracy, character recognition using CNN which outperforms SVM, used more data for training.
2	Optical Character Recognition System for Nepali Language Using ConvNet [3]	Only tested for printed text and not on number plates, many errors for visually similar characters, fewer fonts support	Our system is made specially for detecting and recognizing number plates. Detection even works for video.
3	An Integrated Number Plate Recognition for Multi Language Fonts [4]	Template Matching used for character recognition, which requires large database for multiple fonts, may result in poor accuracy for video	We used YOLO for detection of License plates which achieved good accuracy both in case of images and videos.

4	Optical character recognition for Sanskrit using convolution neural networks [5]	System was designed to recognize texts in manuscripts and not for License plates	Our system is specially designed to detect and recognize Nepali Number Plates
5	A robust real-time automatic license plate recognition based on the YOLO detector [6]	Use of YOLOv2 which is slower and less accurate, also detection is not tested for Nepali number plates, Use of two different CNNs for recognizing characters and digits	We have used YOLOv4 which is faster and more accurate, our system uses a single CNN for recognizing characters as well as digits which works for Nepali number plates.
6	Number plate recognition on vehicle using YOLO-Darknet [7]	Used YOLOv3 both for License plate detection and recognition but not tested for Nepali number plates	We used YOLOv4 which is more accurate for detection and CNN for recognition because when we used YOLO for character recognition as well we didn't achieve good accuracy for video and smaller blur number plates
7	Evaluating performance of Nepali script OCR using Tesseract and artificial neural network [8]	They compared Tesseract OCR and an ANN for Nepali documents but didn't test for number plates. Also, they found that ANN resulted in higher accuracy than Tesseract OCR	Our system can detect License plates and recognizes numbers in number plate using CNN which resulted in better accuracy than Tesseract.

The paper "Automatic Nepali Number Plate recognition with Support Vector Machines" by Pant et al. [2] used support vector machines for character recognition and achieved overall accuracy of 75% with 87.59% average precision, 98.66% average recall and 92.79% average F-Score. However, they used very less dataset, so this system is not efficient.

Manish K. Sharma and Bidhan Bhattarai [3] proposed a system that performs optical character recognition for Devnagari characters in Nepali language by using

CNN(Convolutional Neural Network). This system translates images of handwritten or printed texts to machine editable electronic texts and achieved an accuracy of 96%. This system was not only able to recognize basic characters, but also characters obtained from a combination of vowels, consonants and special symbols. However, this system hasn't been tested for recognizing characters from a number plate and for recognizing different fonts.

The paper by Arun Vaishnav and Manju Mandot [4] shows the use of template matching algorithm for recognizing multi-language fonts. This sytem can recognize both English and Devnagari characters and achieved segmentation accuracy of 97% and recognition accuracy of 98%. However, this system is not tested for video.

Meduri Avadesh and Navneet Goyal [5] proposed a CNN based OCR system that accurately digitizes Sanskrit manuscripts.This system was robust to poor image quality, contrast, font style, font sizes and even worked with half letters and combined letters which achieved an accuracy of 93.32%.

The paper by Laroca et al. [6] proposed a robust real time automatic license plate recognition based on YOLOV2 and fast YOLO which achieved recognition rate of 93.53% and 47 FPS in SSIG dataset and 78.33% and 35 FPS for larger UFPR-ALPR dataset. This system used YOLO for License plate detection and two seperate networks for recognizing characters and digits.

The paper "Number plate recognition on vehicle using YOLO-Darknet" by Setiyono et al. [7] used YOLOv3 and Darknet-53 as feature extractor and used YOLO both for license plate detection and character recognition. This system achieved number plate recognition accuracy of 88% and character recognition accuracy of 98.2% with proper data preprocessing, but thy have used very fewer data for training and testing the model.

In the paper "Evaluating Performance of Nepali Script OCR using Tesseract and Artificial Neural Network" by Prajapati et al. [8] compared performance of an ANN and Tesseract OCR when testing on Nepali documents. They achieved training accuracy of 96% and testing accuracy of 69% using Tesseract OCR, and training accuracy of 98% and testing accuracy of 81% using ANN. They have used many font faces and concluded that ANN was more accurate than Tesseract.

Numerous methods have been used for number plate detection and recognition, but most of these approaches are not capable of recognizing LPs in real-time and for Nepali number plates. In this sense, we employ a deep learning approach using YOLOv4-darknet for license plate detection in vehicles and for character segmentation and a Lenet based CNN for character recognition to create a Nepali number plate recognition system. In addition, we used various algorithms for image preprocessing, data augmentation and image deskewing.

Chapter 3
Requirement Analysis

3.1 Software Requirement

Software requirement for our prepared system includes:

a. Python

b. OpenCV

c. Adobe XD

d. Star UML

e. Visual Studio Code

f. Google Colaboratory

g. CUDA

h. Gitlab

i. GitHub

j. ClickUp

k. OverLeaf

l. Microsoft Teams

m. Google Drive

n. SQLite

3.2 Hardware Requirement

3.2.1 Graphical Processing Unit

Basically, Google Colab gave the privilege to GPU '12 GB NVIDIA Tesla K80 GPU'. In general, the requirement for processing power is CUDA enabled GPU.

3.2.2 Computer

To operate this system, we generally need a laptop or desktop with Intel-i5 processor. However, the operational feasibility in CPU is quite slow so, recommendation is in use of Graphics Card with CUDA enabled.

3.3 Functional Requirement

These requirements describe the main functionality of the system and about the services provided to the user. Our system includes the following functional requirements.

a. The system should allow user to upload image and video.

b. It should save the uploaded image and video.

c. It should detect number plates in Image and Video.

d. It should recognize characters in the detected number plate.

e. It should record the recognized number in database.

3.4 Non-Functional Requirement

These are essential for the better performance of the system. The points below focus on the non-functional requirement of the system prepared.

3.4.1 Reliability

Different test metrics are conducted to test the accuracy of the system. Also, the system has been tested for various scenarios. Necessary error handling mechanism have also been added.

3.4.2 Maintainability

We have modularized our system by breaking it to smaller submodules. In case of any problems, we can easily check and maintain a particular submodule.

3.4.3 Performance and Accuracy

When using our system for an image, we got real-time results for images and proper accuracy. Also, using a proper hardware, we can get good performance in our system.

3.4.4 Frames Per Second

For the smoothness of the system, we required at least 4 to 7 frames per sec.

3.4.5 Portability

As we have developed a web app for our system with proper cloud hosting, we can deploy it, so it is portable across different OS.

3.4.6 Availability

User can upload the image or view in the web app hosted at any time to see the result so, it is available at anytime.

Chapter 4

Feasibility Study

4.0.1 Economic Feasibility

The expenditures for this project are camera and computation power. Camera is easily available and is comparatively cheaper. On the other hand, the processing power of this system is mainly for creating and training deep learning models, which requires the good GPU and memory. This was done in Google Colab.We were able to take the datasets of vehicle on our own and characters datasets were found from kaggle and GitHub sources.

4.0.2 Technical Feasibility

Technically, this project is challenging as we are working on the Devanagari character dataset for recognizing Nepali number plates. However, this project is technically feasible as we are mostly familiar with Python language, and we implemented CNN and YOLO models using Python. Some dataset was prepared by us and others were extracted from online.For this project we are using Intel i5 processor and GeForce 920M(CUDA enabled) GPU with Windows as main operating system. Also, we accessed GPUs like Tesla T4 and Tesla K80 from Google Colab for training YOLO models. However, deploying this project in cloud servers like Heroku may be difficult because the trained YOLO weights take more memory space, and we installed many dependencies for this project which may not be compatible in cloud. So for now we are using localhost for deploying our system in a web app.

4.0.3 Operational Feasibility

The system integration is done using PyTorch and deployed into web using Django as backend and bootstrap as frontend. Our system is operationally feasible due to very simple UI and UX. When a user enters our web app, s/he will have options either to detect for image or video. Based on their choice, the user will then upload an image or video in which they have to detect the number plate. The model runs in the backend once the user enters input and returns the required output numbers, which are then stored in a database and shown to the user in the frontend. So in this way a User can easily operate this system.

4.0.4 Time Feasibility

This system needs more than 3 months if we were to develop it properly with proper accuracy, but as we only got nearly 1.5 months to complete this project. Our system lacks proper accuracy and various other features. Also, documentation took some time. So this system is hardly time feasible to be developed in 1.5 months mainly due to lack of proper dataset and resources like GPU for local testing.

Chapter 5

Methodology

5.1 Software Development Approach

Building the overall project in python and then implementing in web app was a big challenge, so we decided to firstly develop individual components and then integrate the components.In this project, we have used Agile Methodology with SCRUM framework and developed different components in different sprints of project. In order to track progress of our project and follow SCRUM guidelines, we used B.1 software.We assigned task B.1.1 to members using clickup. Different tasks were assigned to different members and the project is divided into 3 sprints where each sprint lasted for 2 weeks. Our System development approach is shown in figure 5.1.We conducted weekly meetings and based on meetings we created and divided work and after completion of work we made weekly reports after some discussion.

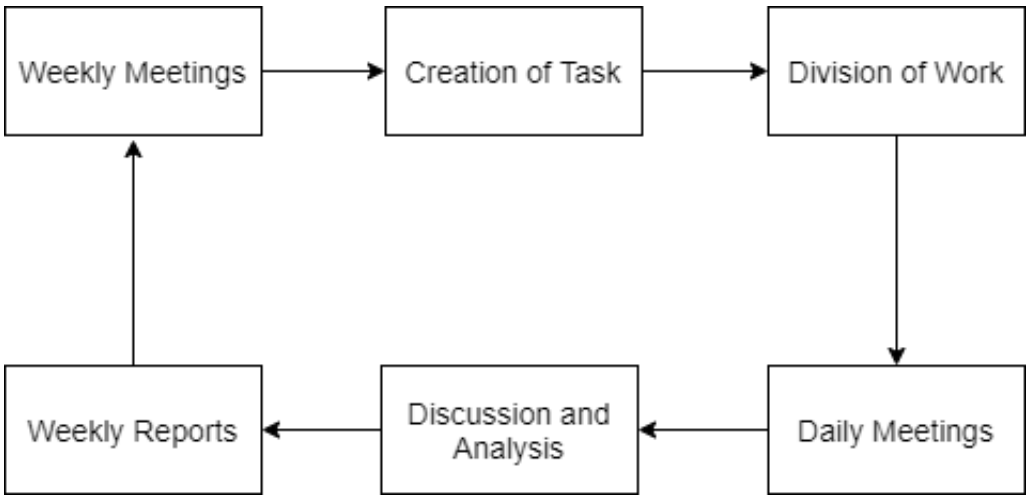


Figure 5.1: Software Development Approach

5.2 ClickUp as Project Management Tool

We used ClickUp as our project management tool to organize, distribute, manage and track all the works and to assign tasks to each member of our team as shown in figure B.1. We created tasks as shown in figure B.1.1. Firstly, we created a product backlog as shown below. Then we created 3 sprints of 2 weeks and completed tasks from backlog in these sprints.

5.2.1 Product Backlog

Product Backlog consists of new features, bug fixes and activities changes during the development phase of project. The features of product backlog is combined together to create release and those release are futher combined to form sprint backlog. Our product backlog includes the following features:

1. Data Collection for YOLOv4 LPD
2. Data Labelling for YOLOv4 LPD
3. Data Visualization
4. YOLO LPD Model Training
5. Testing YOLO LPD Model
6. Data Collection and Labelling For Character Segmentation
7. Character Segmentation using YOLOv4 Tiny
8. Image DeSkewing
9. Data Collection For CNN
10. Data Cleaning for OCR CNN
11. Image Processing
12. CNN Model Training
13. CNN Model Testing
14. UI design
15. Frontend Development
16. Backend Development
17. Error Handling
18. Number Plate validation
19. Integration of YOLO and CNN models
20. Model Evaluation and Visualization
21. Deployment in Web app

Different tasks we have done in different sprints during project development are as follows:

5.2.2 Sprint 1

After doing some research and proper planning, we started our first sprint. The Sprint 1 shown in B.1.2 consist of the following tasks taken from product backlog:

1. Data Collection for YOLOv4 LPD
2. Data Labelling for YOLOv4 LPD
3. YOLO LPD Model Training
4. Testing YOLO LPD Model

At the end of the first sprint we obtained a YOLOv4 LPD model that can take an image or video as input and detect number plates in that video. Also, this sprint consists only 4 tasks as data collection and data labelling had to be done manually which took about a week.

5.2.3 Sprint 2

We started the second sprint by taking the LPD model from sprint 1 and analyzing the output. We then made some changes in that model which improved the accuracy of that model. The sprint2 shown in B.1.3 consist of following tasks taken from product backlog:

1. Data Visualization
2. Data Collection and Labelling For Character Segmentation
3. Character Segmentation using YOLOv4 Tiny
4. Image DeSkewing
5. Data Collection For CNN
6. Data Cleaning for OCR CNN
7. Image Processing
8. CNN Model Training
9. CNN Model Testing

At the end of this second sprint we obtained a YOLOv4-tiny model that can perform character detection and segmentation from the input number plate images. Also, we developed an image processing pipeline and trained a CNN model. CNN model for character recognition was also tested, which gave good accuracy.

5.2.4 Sprint 3

At the beginning of the third sprint we have all the 3 models needed for LPD, character segmentation and character recognition. Some improvements were made in character recognition model which increased the accuracy. Also, some changes were made in the image processing and sorting algorithms. The sprint3 shown in B.1.4 consist of following tasks taken from product backlog:

1. UI design
2. Frontend Development
3. Backend Development
4. Error Handling
5. Number Plate validation
6. Integration of YOLO and CNN models
7. Model Evaluation and Visualization
8. Deployment in Web app

At the end of third sprint, we have all the models were integrated, and also the web app was developed. The web app was later linked with the integrated models and deployed in localhost. Some changes and improvements were also made with time.

5.3 Overall Phase Followed

The overall project has been completed in three main phases which are:

a. Planning Phase

b. Development Phase

c. Integration

5.3.1 Planning Phase

In planning phase, necessary works to be done and planning of overall project were discussed, and the decision were documented for further procedures. We divided the work and made future plans.
First, the project was divided into four parts:

a. Number Plate Detection

b. Cropping Number Plate

c. Character Segmentation

d. Character Recognition

These parts were then assigned to each project members, who then studied about the respective parts in detail.

5.3.2 Development Phase

In this phase, the divided parts were well studied and developed. Then each of those developed parts were tested separately.

5.3.3 Integration

In this phase, the separately developed parts were integrated to form a system and then integration testing was done.

5.4 Work Breakdown

Table 5.1: Work Breakdown Structure

Task	**Anish**	**Anjan**	**Sachin**	**Sudip**
Study about YOLOv4 and YOLOv4-tiny models	✓	✓	✓	✓
Study about Character Segmentation and OCR	✓	✓	✓	✓
Data Collection from various sources	✓	✓	✓	✓
Data Labelling for YOLO LPD	✓	✓	✓	✓
Training YOLO Model for LPD	✓	✓	✓	✓
Testing YOLO Model	✓	✓	✗	✓
Study about different image deskewing techniques	✗	✓	✗	✓
Implementing deskewing algorithm in number plates	✗	✓	✗	✓
Study about different character segmentation techniques	✗	✓	✗	✓
Implementing Character Segmentation	✗	✓	✗	✓
Interpretation and visualization of output from detection model	✗	✗	✓	✗
Study about different OCRs(tesseract-ocr,keras-ocr,easy-ocr)	✓	✓	✗	✓
Data Preparation and Cleaning for CNN	✓	✗	✗	✗
Study about different image processing algorithms	✓	✓	✗	✓
Implementing Image Processing algorithm	✓	✓	✗	✓
Study about different different CNN architectures	✓	✗	✗	✗
Model Training For CNN	✓	✗	✗	✗
Testing and Evaluation CNN Model	✓	✗	✗	✗
Study about Integrating different types of models	✗	✓	✗	✗
Integrate LPD models with Character segmentation and recognition model with other parts	✗	✓	✗	✗
Testing integrated models for Image	✗	✓	✗	✗
Testing integrated models for Video	✗	✓	✗	✗
UI and Frontend Design	✓	✗	✓	✗
Backend development using Django	✓	✗	✗	✓
Integrating Web app with Models	✓	✗	✗	✗
Implementing sorting algorithm	✓	✓	✗	✗
Implementing Error handling and Number Plate Validation algorithm	✓	✓	✗	✗
Documentation Preparation	✓	✓	✓	✓

Chapter 6
System Design and Architecture

6.1 System Block diagram

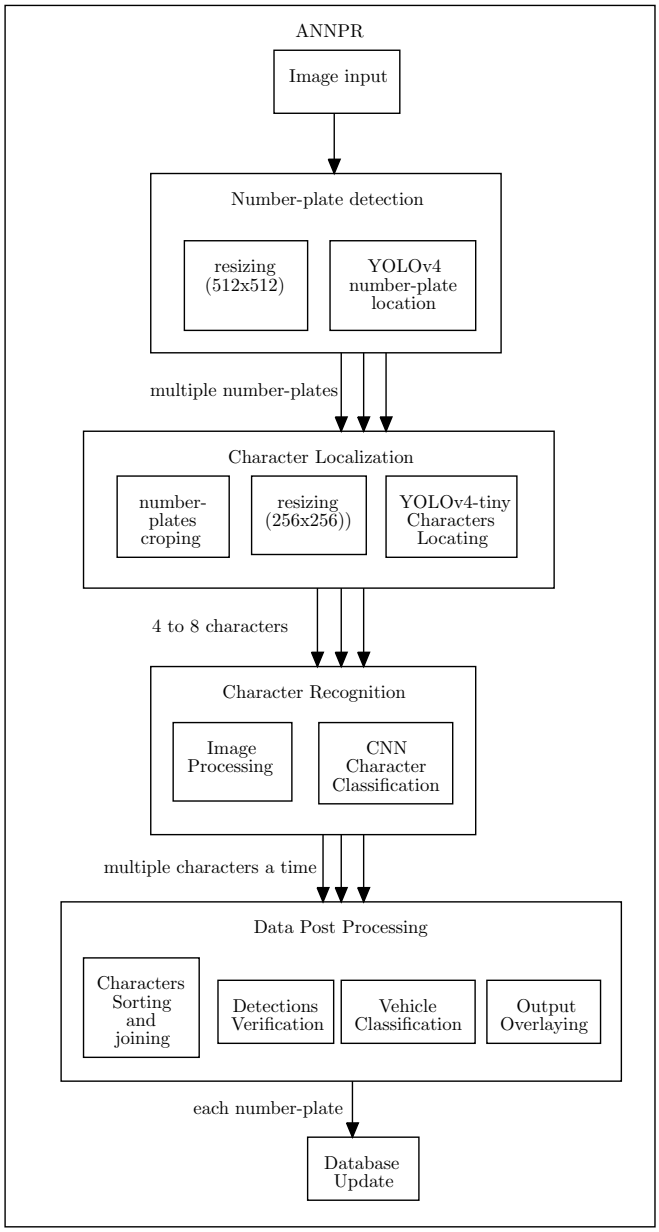


Figure 6.1: Block Diagram of ANNPR

6.2 Black Box View

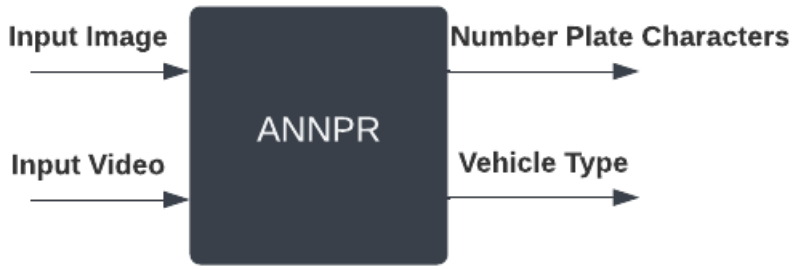


Figure 6.2: Black Box View of ANNPR

6.3 Entity Relationship Diagram

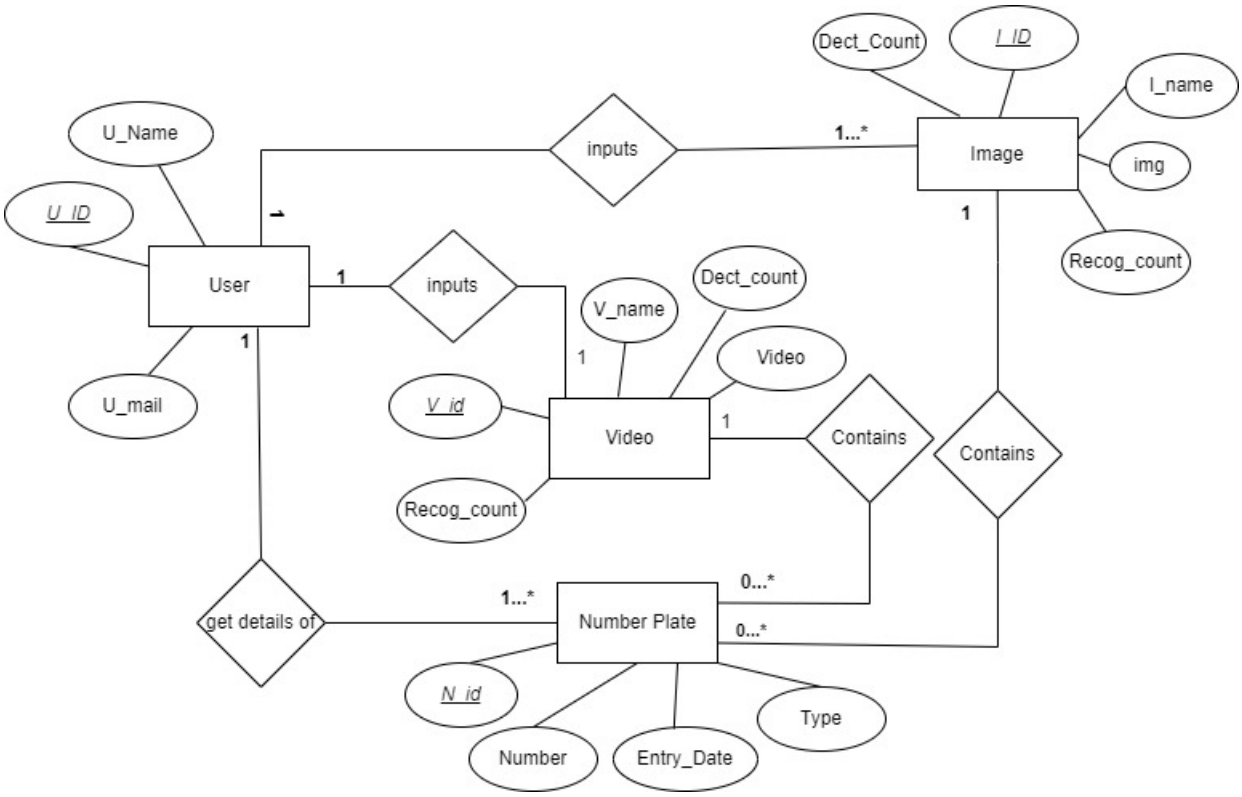


Figure 6.3: ER diagram of ANNPR

6.4 Sequence Diagram

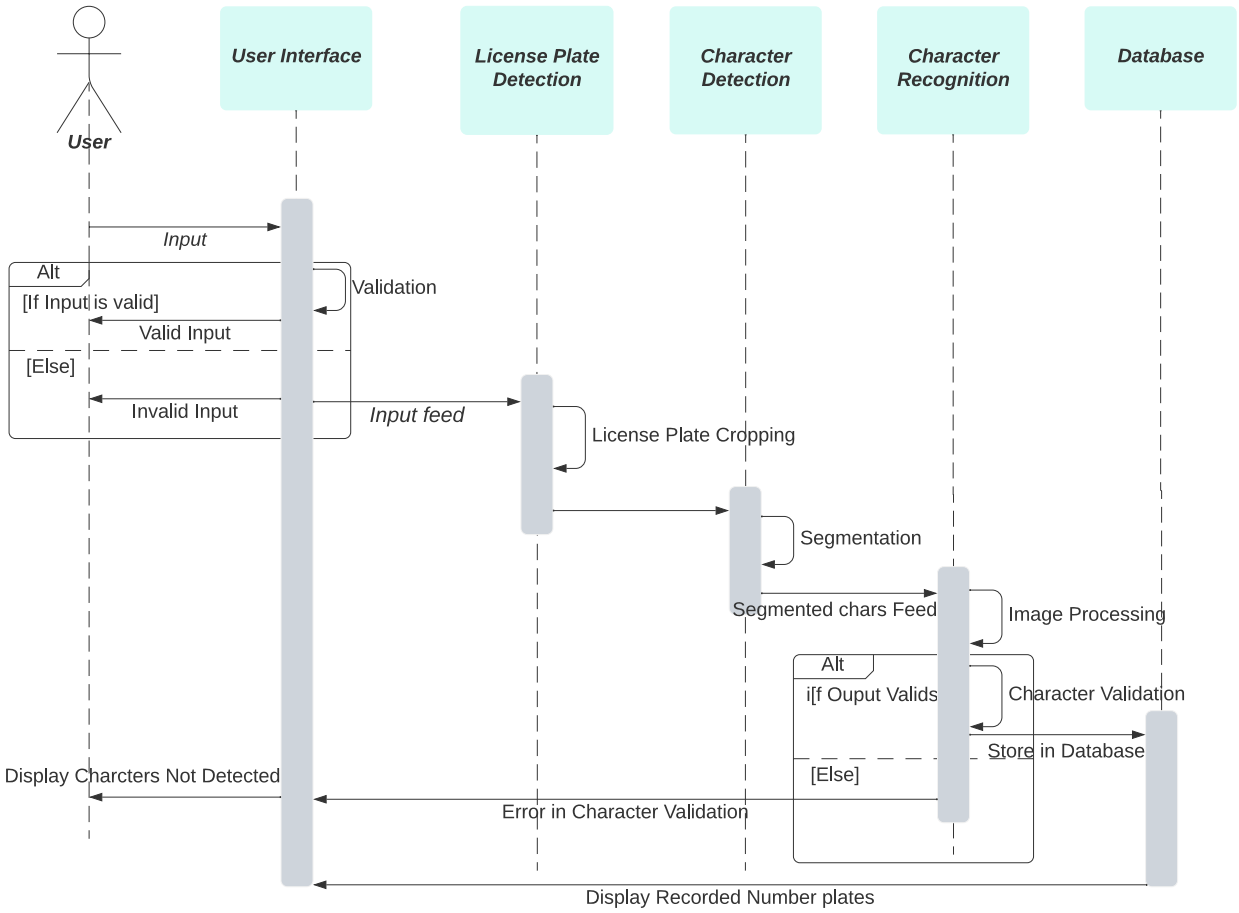


Figure 6.4: Sequence Diagram of ANNPR

6.5 Activity Diagram

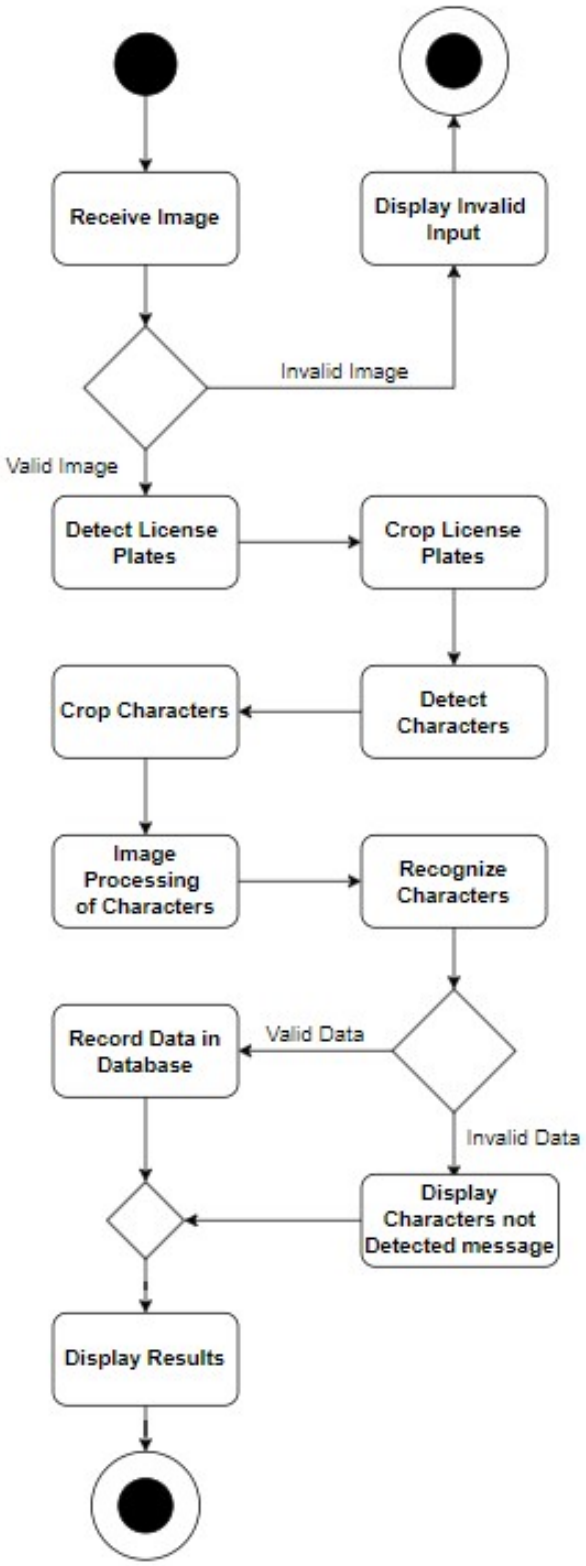


Figure 6.5: Activity diagram of ANNPR

6.6 Description of Working Flow of System

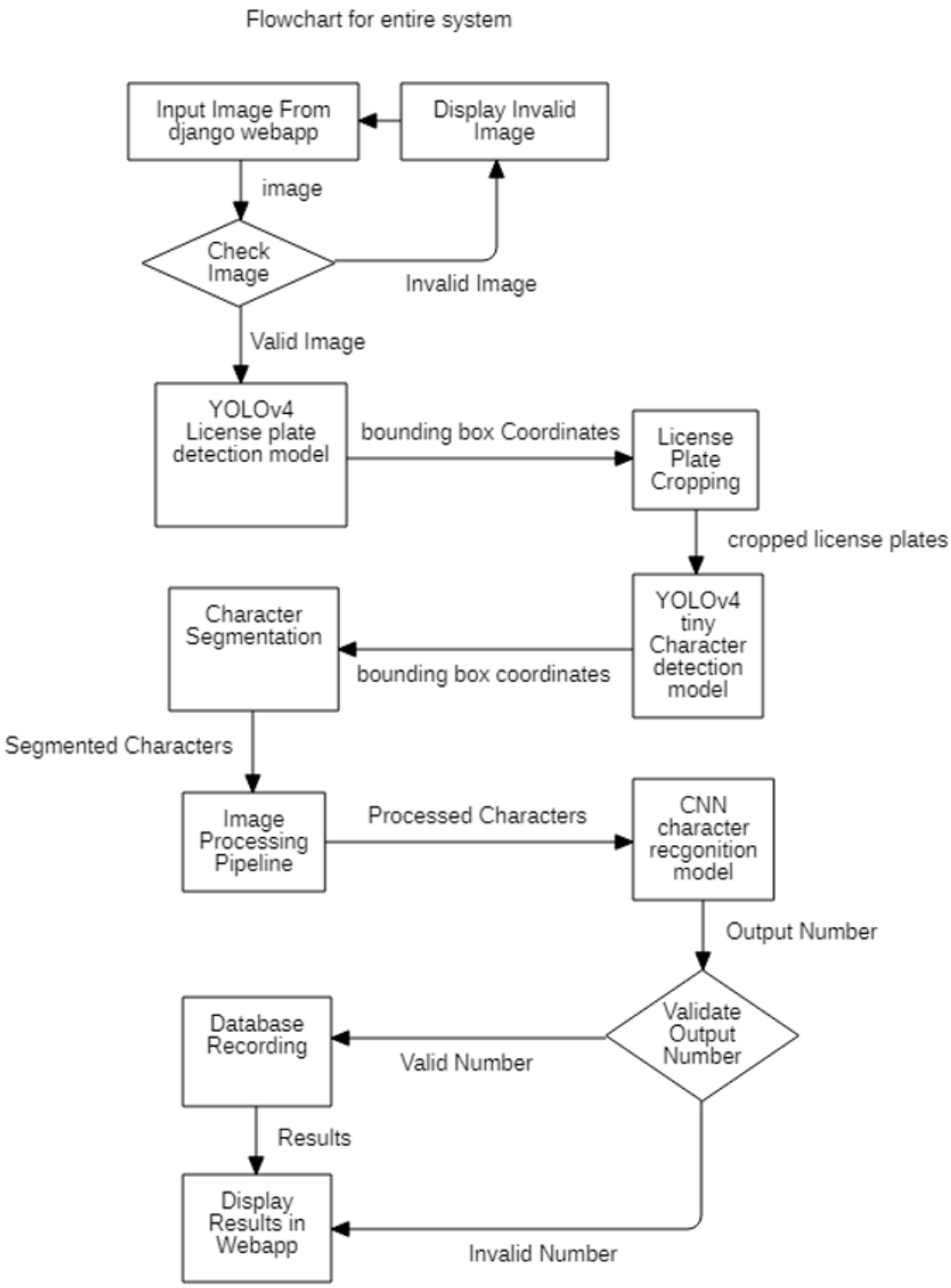


Figure 6.6: Flowchart showing working flow of system

Figure 6.6 shows the working flow of our system. The main steps in working of this system are as follows:

1. Image Validation
2. YOLOv4 License Plate Detection
3. YOLOv4 Tiny Character Detection
4. License Plate and Character Cropping/Segmentation
5. Image Processing
6. CNN Character Recognition
7. Output Number Validation
8. Data Recording

6.6.1 Image Validation

This step validates the input image from the user in Django web app. It checks whether the uploaded file is an image or not.

6.6.2 YOLOv4 License Plate Detection

The YOLOv4 model shown in 6.8 is used for detecting the license plates in input image. This step outputs bounding box coordinates for the detected license plate, which is later used for cropping the plate.

6.6.3 YOLOv4 Tiny Character Detection

The YOLOv4 Tiny model shown in 6.9 is used for detecting characters from the cropped license plates. Similar to LP detection model, it outputs bounding box coordinate for each character. However, this step includes an additional sorting step that will sort the characters from top to bottom (for 2 rowed LPs) comparing the height of the character with respect to y co-ordinate and then left to right so that the output from CNN will be in correct order.

6.6.4 License plate and Character Cropping/Segmentation

Based on the bounding box coordinates obtained from YOLO models, we crop the license plate and characters. Cropped LP is then used for character detection. Also, character segmentation(cropping) is performed using the coordinates received from YOLO. We preferred YOLOv4 tiny for character segmentation than Histogram and different image threshold based approach because they resulted in lower accuracy, specially in case of skewed images and 2 rowed license plates. The segmented characters are then passed to an image processing pipeline before sending to CNN model for character recognition.

6.6.5 Image Processing

Image Processing along with character segmentation are the most important steps in this system as it plays crucial role in character recognition. Image processing is done with help of OpenCV which involves various steps as shown in image processing pipeline in figure

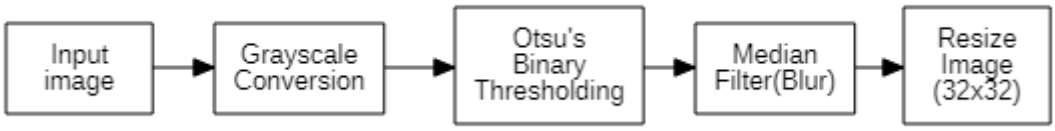


Figure 6.7: Image Processing Pipeline

6.6.5.1 Gray Scale Conversion

The input image by user is generally a colored image which is difficult to process, so we convert it to a grayscale image. A grayscale image consists of shades of gray whose pixel value ranges from 0 to 255. It basically converts a 3D image to 2D image, which makes the image suitable for other image processing algorithms.

6.6.5.2 Otsu's Binary Thresholding

Otsu's method of Thresholding is the simplest algorithm that returns a single intensity threshold that separate pixels into two classes, foreground and background. It is a global thresholding technique that uses variance based technique to perform binarization in an image. After reading different research papers, we found that this method has been widely used in different image processing projects and is efficient, so we decided to use this method for binary thresholding of the characters. We implemented it in OpenCV, taking 125 as threshold and 255 as max value.

6.6.5.3 Median Filter(Blur)

Our main purpose of using Median filter during image processing is to reduce the noise present in the image. It blurs(smooths) an image using median filter with the ksize x ksize aperture. This algorithm scans the entire image using a small matrix and then recalculate the value of the center pixel by taking the median of all values inside the matrix. We used a median filter with ksize of 3 using OpenCV after we performed Otsu's Binary Thresholding on the grayscale image.

6.6.5.4 Image Resizing

At the end of the image processing pipeline, we again perform resizing to convert each character image to 32x32 to make it suitable for input to CNN model. Image Resizing is also performed during number plate detection step when the number plate is resized to size 512x512 before sending to YOLOv4 model for detection. Also, we perform image resizing to resize cropped number plates to size 256x256 to detect characters using YOLOv4 tiny model. After resizing each character image, we reshape the image and add a depth of 1 to each image to make it ready to

pass to CNN model for character recognition. The Final Shape of the image looks like (32,32,1) where 32 represents width and height and 1 represents number of channels or depth.

Outcomes of all these steps are shown in appendix B.2.1

6.6.6 CNN Character Recognition

The CNN model shown in figure 6.10 is used for recognizing the character images sent from image processing pipeline. This model will then return the class ID of the image that has the highest percentage. After detecting the class of each cropped characters, output class names are joined and output number is returned from the CNN model. Our model currently used has an average accuracy of 94% for all characters. The accuracy of character recognition highly relies on character segmentation, image preprocessing and character orientation.

6.6.7 Output Number Validation

The output number returned from CNN model is then validated in view of a Django web app. Sometimes the CNN model may give false positive and falsely recognize some characters or sometimes the sorting algorithm may fail, and the numbers may come in incorrect order or sometimes the segmentation algorithm may fail to detect and segment all characters from the number plate. In such cases, to prevent the system to save invalid data to database, we use output number validation. If the output number is invalid, we return no recognition as output to the user in the frontend.

6.6.8 Data Recording

If the recognized number plate returns a valid output number, it is recorded in a database in this step. Our project uses SQLite database, and we have created a table in the database to store the output number, vehicle type and entry date. Also, classification of the vehicle is done in view before saving the data in database. Finally, recorded data is displayed to the user by frontend in web app.

6.7 Dataset

6.7.1 Dataset Collection

We needed to train two models using datasets. The two models include the number plate detection and characters recognition. We collected two types of dataset for model training. For number plate detection using YOLO, we did not find any dataset from online resources. we had to collect dataset on own and arranged them. We collected around 1850 images of vehicle which contained the images of private 2-wheeler and private 4-wheeler vehicle only for this project. We labelled those images using LabelImg [9]. This data was split into 70% train and 30% test and some unseen data was used for validation with single class name as 'license-plate' in YOLO format as shown in table 6.2. For character recognition model using CNN,

we collected dataset from online source. The online source includes GitHub [10] and kaggle [11]. The datasets contain the image of Devanagari characters: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, baa, pa, cha. However, the dataset was of very poor quality, and we had to manually clean the dataset, which again decreased the size of the dataset. We have total, 16540 images which were divided into dataset for each character as shown in table 6.3. Overall data was split as 70% training data, 15% testing data and 15% validation data. Also, 100 images of each class were used as unseen test data

Table 6.1: Data For License Plate Detection using YOLOv4.

Number Plate Dataset			
Type	**Training**	**Validation**	**Testing**
Vehicle Image	1200	150	500

Table 6.2: Data For Character Detection using YOLOv4 tiny.

Character Detection Dataset		
Type	**Training**	**Testing**
Character Image	250	110

Table 6.3: Data For Character Recognition using CNN.

Character Recognition Dataset				
Characters	**Training**	**Validation**	**Testing**	**Unseen**
0	1168	178	237	150
1	1047	154	212	150
2	855	167	180	150
3	854	162	180	150
4	850	157	178	150
5	844	145	175	150
6	930	186	197	150
7	757	143	159	150
8	941	167	196	150
9	921	164	191	150
ba	800	131	164	150
pa	1211	220	253	150
cha	772	134	160	150

6.7.2 Data preprocessing

The collected data were not all clean, and it required preprocessing before using it in model creation. For YOLO models we firstly renamed all the dataset images in a proper order (e.g. a1,a2) then only we used labelimg [9] tool to label each images. After labelling, we get labels of each class and a file called classes.txt with all the classes present. These were important for training. From classes.txt we

generated classes.names and also using python script we split our training dataset for 70% train and 30% test for YOLO models.

Similarly, for training CNN model, images were firstly separated into proper classes and unclear images were removed. This was done manually and took some time. After that, we used a python script to rename all the images serially. Further preprocessing was done with help of sklearn module. We split data to 70% train, 15% test and 15% validate as seen in table 6.3 then used data augmentation techniques like zoom, shift, shear, rotate to generate more dataset. Finally data was ready to be used in training CNN model.

6.8 Model Creation

In this project, we have two models, comprising YOLOv4 model and CNN model.Basically, The model for number plate detection and segmentation of characters from number plate model are trained and tested using DarkNet framework of YOLO. The character Recognition Model was trained and tested using CNN.

6.8.1 YOLOv4 models

We used DarkNet which is a convolutional neural network that is used as the backbone of YOLO.Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.Darknet is used for YOLOv4 model creation in our project.In short, DarkNet model is a typical CNN structure which has:

- a convolution layer that extracts features from the input with the filter applied on it.
- a pooling layer to reduce the size of computational performance.
- a fully connected layer which is combined one or more times.

In general, two YOLOv4 model for object detection were trained. Several models were created after training.We tried training for YOLOv3, YOLOv4, YOLOv4-tiny and test accuracy for different models. Among them, the shortest time for training was YOLOv4-tiny but has less accuracy so, we chose YOLOv4 over them because it is fast is detection. The model was basically trained with transfer learning phenomenon.

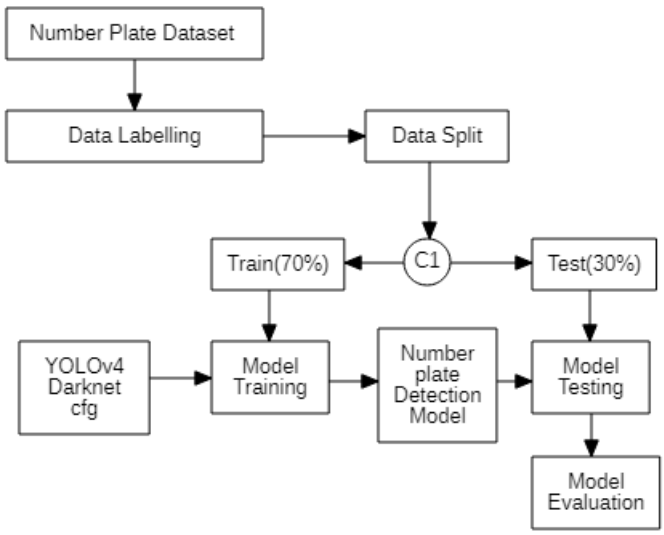


Figure 6.8: Generation of Custom YOLOv4 model for License Plate Detection

Number Plate Detection model was obtained by training the labelled dataset with split of 70% total. We configured the framework DarkNet CFG file for best accuracy and model was obtained. Further, it was tested with 30% of total data.

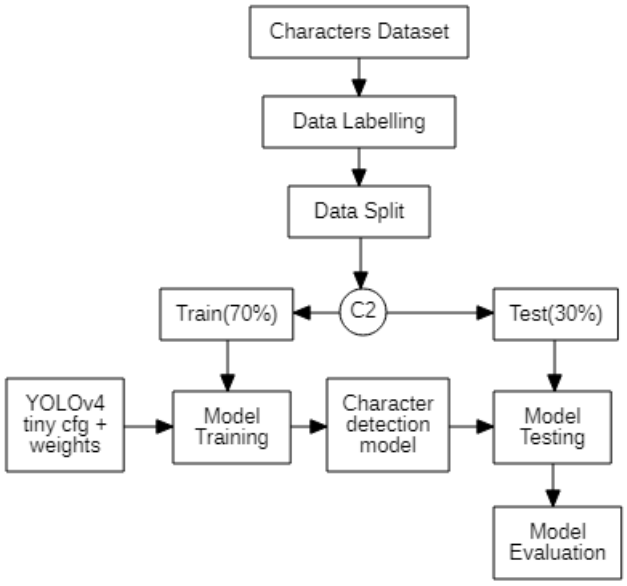


Figure 6.9: Generation of Custom YOLOv4 model for Character Detection

Character Detection Model was generated using yolov4 tiny with training data 70% of total data. The single class was created for data labeling. Besides, we even directly did the OCR using YOLOv4 with 19 classes, but its accuracy was lesser due to high number of classes so, we chose single class model over 19 classes model.

6.8.2 CNN model

In order to recognize the characters of a number plate, we trained a CNN model. Our CNN model was based on Lenet architecture, which was later modified according to our needs. Our CNN model was implemented using TensorFlow Keras.

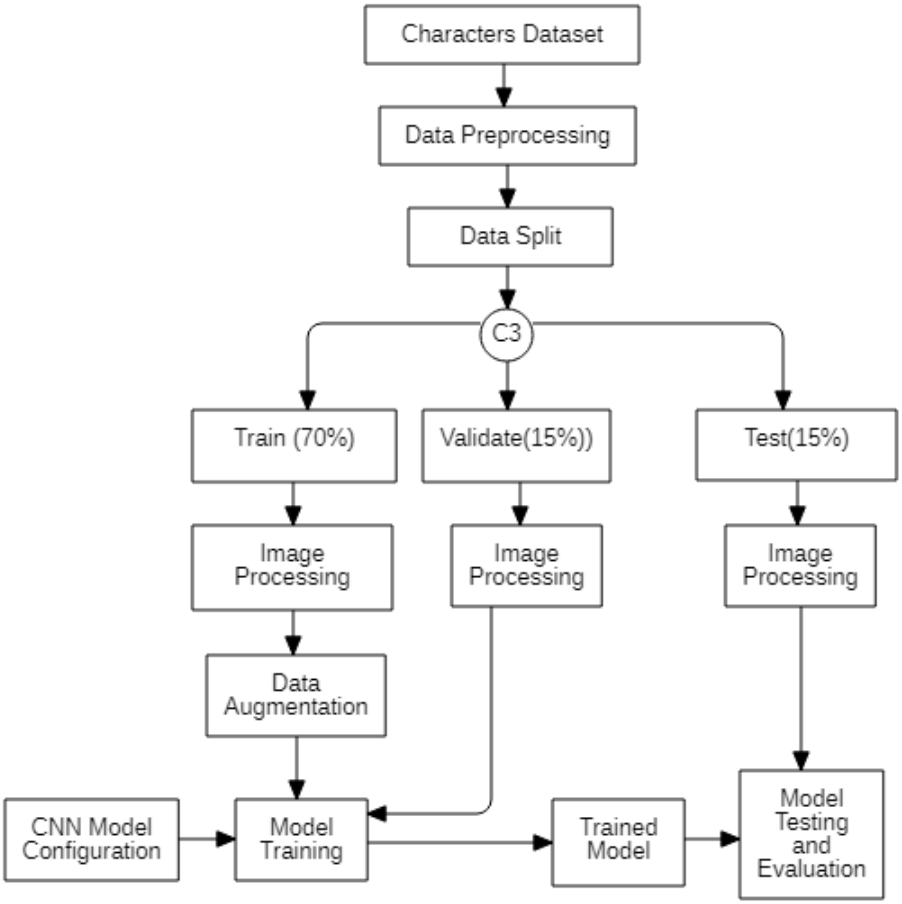


Figure 6.10: Generation if custom CNN model for Character Recognition

During data preprocessing for the model, each class ID was assigned to images in the dataset, as this CNN model is based on supervised learning. Later data was sent to image processing and data reshaping followed by data augmentation to make it suitable for model training. The architecture of the trained CNN model can be seen below in figure 6.11. In this model, convolutional layers act as feature extractors that will extract feature from data during training. As seen below, we have 11 layers in total.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 28, 28, 60)        1560

 conv2d_5 (Conv2D)           (None, 24, 24, 60)        90060

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 60)       0
 2D)

 conv2d_6 (Conv2D)           (None, 10, 10, 30)        16230

 conv2d_7 (Conv2D)           (None, 8, 8, 30)          8130

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 30)         0
 2D)

 dropout (Dropout)           (None, 4, 4, 30)          0

 flatten_1 (Flatten)         (None, 480)               0

 dense_2 (Dense)             (None, 500)               240500

 dropout_1 (Dropout)         (None, 500)               0

 dense_3 (Dense)             (None, 13)                6513

=================================================================
Total params: 362,993
Trainable params: 362,993
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 6.11: Architecture for CNN model

Input to this CNN model is a 32x32 sized image. Image is input firstly to two convolutional layers with filter size of 5x5 and number of filters 60 and its output is passed to a relu activation function. The resulting output is passed to a max pooling layer with pool size of 2x2, which performs max pooling on the feature map extracted by convolutional layers. Again, the output of this pooling layer is passed to two more convolutional layers with 60 filters and filter size of 3x3 and relu as activation function. These two layers also act as feature extractor in this CNN model. The extracted features output is then passed to another max pooling layer with the pool size of 2x2 which perform subsampling. Then, the output of this pooling layer is passed to a dropout layer with value 0.5 which helps to reduce overfitting and make the model more generic. We then flatten the output, which is converting the feature map to one dimensional array. Then the 1D array is passed to a dense layer with 500 nodes and relu activation function. A dense layer or fully connected layer then classifies the image based on the output feature map from the convolutional layer. The output of this layer is again passed to another dropout layer with value of 0.5 and finally to another dense layer with 13 nodes equivalent to number of classes and softmax as activation function. This layer provides classification percentage for each class of input image. During this project we trained CNN models with various configurations and for different approaches and best value was found for batch 50 and epochs 100 and on using

two dropout layers with value 0.5. Also, we then compiled our model using Adam as an optimizer with learning rate of 0.001 and categorical cross entropy as our loss function. Also, we have taken accuracy as our metrics for CNN. We have also used ReduceLROnPlateau class to reduce learning rate when our metric has stopped improving, and we reduce learning rate by a factor of 0.2.

6.9 Model Evaluation

In order to evaluate the trained model, we have tuned the different parameters. The different hyperparameters for YOLO and CNN models are given below:

6.9.1 Precision

Precision measures how accurate are your predictions. i.e. the percentage of your predictions are correct. In our YOLO model for LP detection, we obtained precision of 0.87 for 30% of original seen test data and 0.94 for new unseen data.

$$Precision = \frac{TP}{TP + FP}$$

6.9.2 Recall

Recall measures how good you find all the positives. Recall is a metric that quantifies the number of correct positive predictions made out of all positive predictions that could have been made. Unlike precision that only comments on the correct positive predictions out of all positive predictions, recall provides an indication of missed positive predictions. In our YOLOv4 model, we obtained recall of 0.98 for 30% test data and recall of 0.55 for unseen test data.

$$Recall = \frac{TP}{TP + FN}$$

6.9.3 IoU - Intersection over Union

IoU is the measure of closeness of detected bounding box to the actual number-plate. It is the ratio of intersecting area of detected bounding box and actual bounding box to the union of area of the same two bounding boxes. If the detected bounding box is really close to the actual bounding box, the value of IoU is close to 1 and if they are not intersecting, IoU is zero. Average IoU is the average of all IoUs of number-plate in certain number of test/validation data.

$$avg_IoU = \frac{IntersectingArea}{OverlapingArea}$$

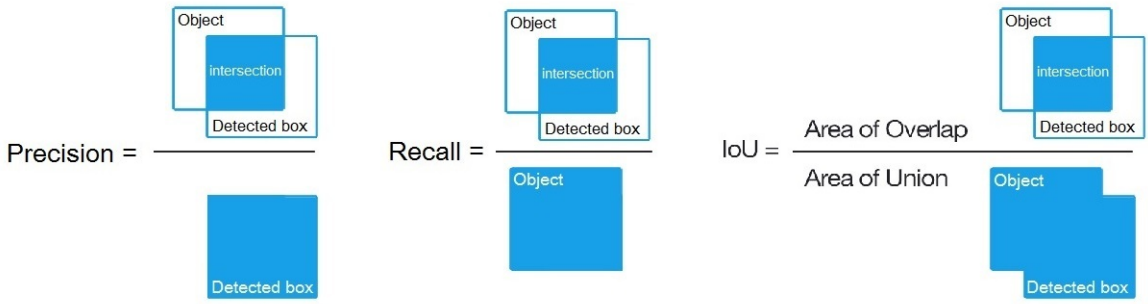


Figure 6.12: Diagram Showing Precison, Recall and IoU
Source: AlexeyAB/darknet [12]

6.9.4 mAP - mean Average Precision

To evaluate object detection models like R-CNN and YOLO, the mean average precision (mAP) is used. The mAP compares the ground-truth bounding box to the detected box and returns a score. The higher the score, the more accurate the model is in its detections.The general definition for the Average Precision (AP) is finding the area under the precision-recall curve above. AP falls within 0 and 1. We obtained mAP of 99.35% for 30% test data and 69.75% for unseen test data in our YOLO LP model.

$$Average\ Precision = \int_0^1 P(r)\,dr$$

Also, the plot showing mAP and average loss for different iterations is shown in B.3.1 and B.3.2 respectively.

6.9.5 F1-score

The F1-score combines the precision and recall of a classifier into a single metric by taking their harmonic mean. It is primarily used to compare the performance of two classifiers. Suppose that classifier A has a higher recall, and classifier B has higher precision. In this case, the F1-scores for both the classifiers can be used to determine which one produces better results. For our YOLO LP detection model, we obtained a F1 Score of 0.92 for 30% of the seen test data and 0.69 for unseen test data.

$$F1\ score = \frac{2(Precision * Recall)}{Precision + Recall}$$

6.9.6 Epoch, Iteration, Batch and Subdivisions

An epoch is a term used in machine learning and indicates the number of passes of the entire training dataset the machine learning algorithm has completed. Iterations is the number of batches needed to complete one epoch. we have 2000 training examples that we are going to use.We can divide the dataset of 2000 examples into batches of 500 then it will take 4 iterations to complete 1 epoch.
While training CNN model we have used 100 epochs and batch size of 50 for approximately, 12000 training images. And number of steps or iterations per epoch is calculated using the formula

$$No\ of\ Steps(Iterations)\ per\ epoch = \frac{Training\ data\ size}{Batch\ Size}$$

For YOLO model, Batch=32 gives loading 32 images for this "iteration". Subdivision=32 gives Split batch into 32 "mini-batches" so 32/32 = 1 images per "minibatch" and this get sent to the GPU for process.That will be repeated 32 times until the batch is completed, and a new iteration will start with 32 new images. Also, for YOLOv4 Model max iterations was 3500.
For convolution layer just ahead of YOLO layers, filters are determined with the help of number of classes using following formula:

$$filters = 3 * (number\ of\ classes + 5)$$

Here 5 indicates other 5 parameters of image : image height, image width, x position, y position, and confidence of the cell. Then it is multiplied by 3 because YOLO can detect 3 bounding boxes in each cell.

6.9.7 Max Batch

In our YOLO model the maximum batches that we can enter was found using the formula:

$$Max\ Batch = 2000 * number\ of\ classes$$

Also Max Batches determine the number of steps which is given as

$$Steps = 0.8 * max\ batches, 0.9 * max\ batches$$

6.9.8 Dropout

Dropout is a regularization technique to avoid overfitting (increase the validation accuracy) in CNN only, thus increasing the generalizing power.Generally, using a small dropout value of 20%-50% of neurons, with 20% providing a good starting point. A probability too low has minimal effect, and a value too high results in under-learning by the network.Using a larger network, we are likely to get better performance when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations. In our CNN model, we have used 2 dropout layers of values 0.5 with the purpose of preventing overfitting.

6.9.9 Learning rate, Optimizer and Loss Function

Learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function. In our CNN model, we have set our learning rate to 0.001. Also, we have used Adam optimizer as our optimization algorithm. Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. We have also used categorical cross entropy as our loss function in CNN, as it is used as a loss function for a multi-class classification model where there are two or more output labels.

6.9.10 Accuracy

Accuracy is the main metric used for performance evaluation in our CNN model. In our CNN model at the end of 100th epoch the training accuracy is 95.88%, validation accuracy is 99.15% and test accuracy is 99.51% with test score of 0.0214. When testing on unseen data for all characters, the accuracy is obtained as shown in table 7.12 below. Also, the average accuracy of our CNN model for all characters is found to be 96.51% for this unseen data. Also, the plot for accuracy and loss for our CNN model can be seen in B.4 and B.4.2 respectively.

6.9.11 Confusion Matrix

A confusion matrix is a summary of prediction results on a classification problem that describes the performance of a classification model. The number of correct and incorrect predictions are summarized with count values and broken down by each class. It tells about True Positives(TP), True Negatives(TN), False Positives(FP) and False Negatives(FN) when a model evaluates in a test data. Evaluation for output of YOLO detection model is determined from confusion matrix as:

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Figure 6.13: Confusion Matrix

True Positive (TP): Number of Correct Detections in each frame.

False Positive (FP): Number of Wrong Detections in each frame.

False Negative (FN): Number of Missed Detection in each frame.

Chapter 7
Result and Discussion

7.1 Desktop Outcomes

In this system, outcomes obtained are not exactly as we expected, specially for videos. Our system developed is very much similar to the UI first designed as shown in B.5.1. When the system was tested for image, the output obtained was fast and mostly accurate. But when the system was tested for video it took a lot of time for processing as our system worked on very low FPS due to lack of proper GPU. We then tweaked some changes in our integrated models and slightly improved processing.
For license plate detection, we provide user with options either to detect for image or detect for video and then based on choice we return appropriate outcome.

7.2 Unit Test

We developed separate units of our system and performed tests as these units were being developed. After inspecting results of these tests, we made necessary modifications and improvements in these units before integrating all the units. Different test cases for our different units are shown below:

7.2.1 YOLOv4 License Plate Detection(LPD) Unit

Table 7.1: LPD Model test scenario and requirements

Test Scenario ID	YOLO LPD Module
Test Case Description	Check LPD with Image and Video as Input
Pre-conditions	Valid Image and Video

Test execution steps:

A. Open module

B. Insert image name or video name

C. Run the module

Table 7.2: Unit Test Case for LPD Module

S.N	Input	Expected Outcome	Actual Output	Result
1	Image with a clear number-plate	one bounding box around number-plate	As Expected shown in 8.2b	pass
2	Image from a busy road in Kathmandu	Bunch of bounding boxes around number-plates with smaller ones missing	As Expected with some number-plates missing shown in 8.2a	pass
3	Grabbed frame from a video	bounding box around number-plate	As Expected shown in 8.2c	pass

7.2.2 YOLOv4-tiny Character Detection Unit

Table 7.3: Character Detection Model test scenario and requirements

Test Scenario ID	Character Detection Module
Test Case Description	Check character detection module for input image of cropped LP
Pre-conditions	Valid Image

Test execution steps:

A. Open module

B. Input image of cropped LP

C. Run the module to detect characters

Table 7.4: Unit Test Case for Character-detection Module

S.N	Input	Expected Outcome	Actual Output	Result
1	Number-plate in 4:1 ratio	bounding box around all characters	As Expected shown in 8.3a	pass
2	Number-plate in 4:3 ratio	bounding box around all characters	As Expected shown in 8.3b	pass
3	Skewed Number Plate	bounding box around all characters	As Expected shown in 8.3c	pass
4	Skewed Number Plate with large angle	No bounding box around any characters	Bounding box seen around some characters shown in 8.3d	fail

7.2.3 CNN Character Recognition Unit

Table 7.5: Character Recognition test scenario and requirements

Test Scenario ID	CNN Character Recognition Module
Test Case Description	Check CNN model with cropped character image as input
Pre-conditions	Valid and processed Image

Test execution steps:

A. Open module

B. Input image of cropped and processed image

C. Run the module to recognize characters

Table 7.6: Unit Test Case for Character Recognition Module

S.N	**Input**	**Expected Outcome**	**Actual Output**	**Result**
1	Processed Image of character included in dataset	Correct Class ID and name for character	As Expected as shown in 8.4a	pass
2	Unknown Character not included in dataset	Model shouldn't predict any class for such image	Not as expected shown in 8.4b	fail

7.2.4 Web App Unit

Table 7.7: Web App test scenario and requirements

Test Scenario ID	Django Web App
Test Case Description	Check if the web app frontend is well integrated with the model's backend and test if image/video can be uploaded or not
Pre-conditions	Function with all integrated models that takes image/video as input and returns list of detected numbers

Test execution steps:

A. Open webapp

B. Input image/video to test

C. Click detect button

Table 7.8: Unit Test Case for Web app

S.N	Input	Expected Outcome	Actual Output	Result
1	Upload Image/Video	Save Image in Media folder	As Expected	pass
2	Call the integrated function with any image	Model returns list of output numbers	as expected	pass

7.3 Integration Testing

Table 7.9: Integrated System Test scenario and requirements

Test Scenario ID	Integrated ANNPR system
Test Case Description	Check if the entire system works properly or not. Check if the user gets proper output for his input image/video and check if details are recorded in database
Pre-conditions	All models should be tested and working correctly

Test execution steps:

A. Open webapp

B. Input image/video to test

C. Click detect button

D. Check detection outputs and database

Table 7.10: Unit test case for complete/integrated model

S.N	Input	Expected Outcome	Actual Output	Result
1	Image with clearly visible number-plate and characters	ba17cha8674	As Expected as shown in 8.5	pass
2	Image with clearly visible number-plate and characters with number-plate ratio 4:3	ba11cha4898	As Expected shown in 8.6	pass
3	Image with multiple number-plates	Both number-plates recognized correctly	only one was recognized correctly as shown in 8.7	fail

4	Video with multiple cars passing	number plates valid number-plates recognized	some number-plates were left unrecognized shown in 8.8	pass

7.4 Model Evaluation Result

7.4.1 YOLO Model Evaluation

Table 7.11: YOLO number-plate detection model performance table

Data	number	TP	FP	FN	mAP	F1-score	avg_IoU
Test data	500	311	46	6	99.35%	0.92	75.8%
Validation data	150	188	13	154	69.76%	0.70	71.97%

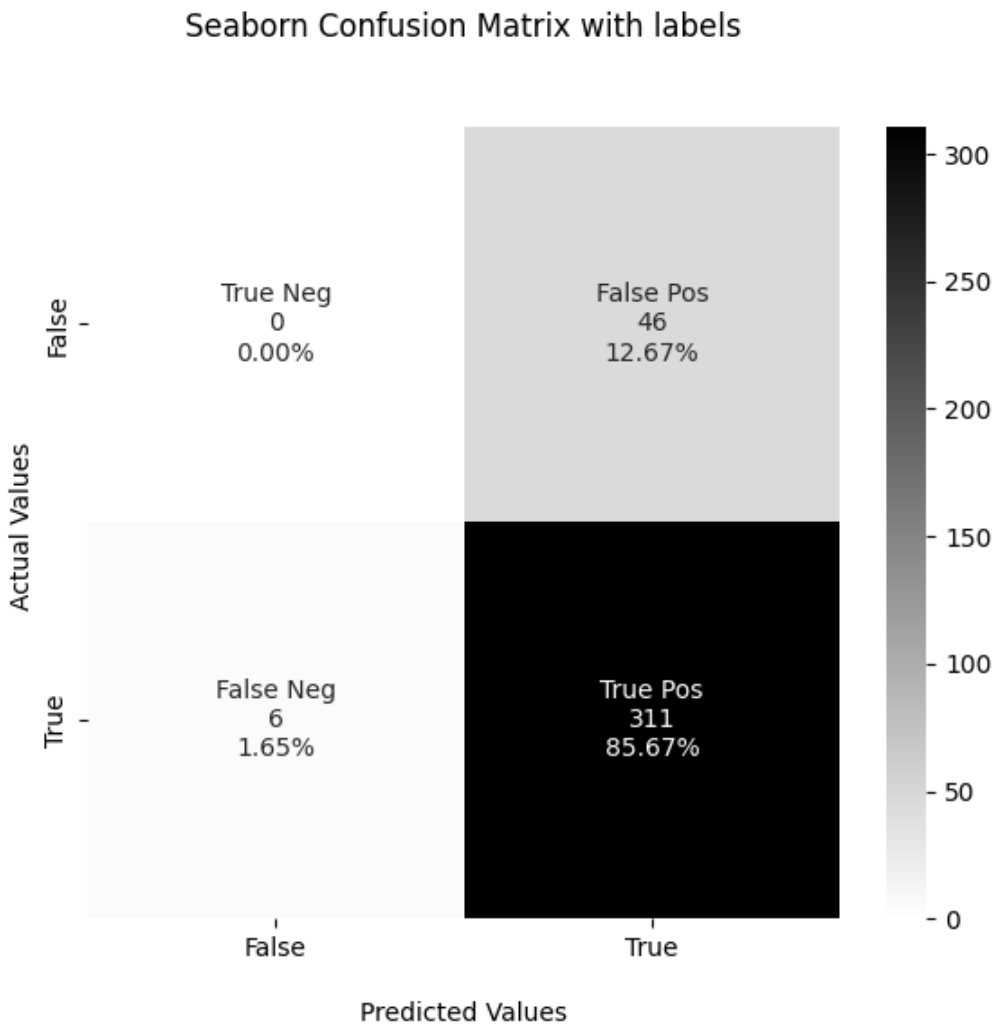


Figure 7.1: Confusion matrix for YOLO LPD model

7.4.2 CNN Model Evaluation

Table 7.12: Average Accuracy for Individual Characters for unseen data

Character	**Accuracy**
0	82.667%
1	97.334%
2	98.667%
3	96.667%
4	93.333%
5	98.667%
6	96.667%
7	99.3333%
8	98.0%
9	98.0%
ba	99.333%
pa	97.334%
cha	98.66%

Table 7.13: Confusion Matrix for CNN Character Recognition model for individual characters

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**ba**	**pa**	**cha**	**total**
0	124	1	1	0	0	0	0	2	3	0	2	17	0	150
1	0	146	1	1	0	1	0	1	0	0	0	0	0	150
2	0	0	148	0	0	2	0	0	0	0	0	0	0	150
3	0	0	0	145	0	0	3	2	0	0	0	0	0	150
4	0	0	0	0	140	4	1	0	0	3	0	0	2	150
5	0	0	0	0	0	148	0	1	0	0	0	1	0	150
6	0	0	0	1	0	1	145	2	0	1	0	0	0	150
7	0	0	0	0	0	0	0	149	1	0	0	0	0	150
8	0	0	1	0	2	0	0	0	147	0	0	0	0	150
9	0	0	2	0	1	0	0	0	0	147	0	0	0	150
ba	0	0	0	0	0	0	0	1	0	0	149	0	0	150
pa	2	0	0	0	0	0	1	0	0	0	0	146	1	150
cha	0	0	0	0	0	0	1	0	0	0	0	1	148	150
total	126	147	153	147	143	156	151	158	151	151	151	165	151	1950

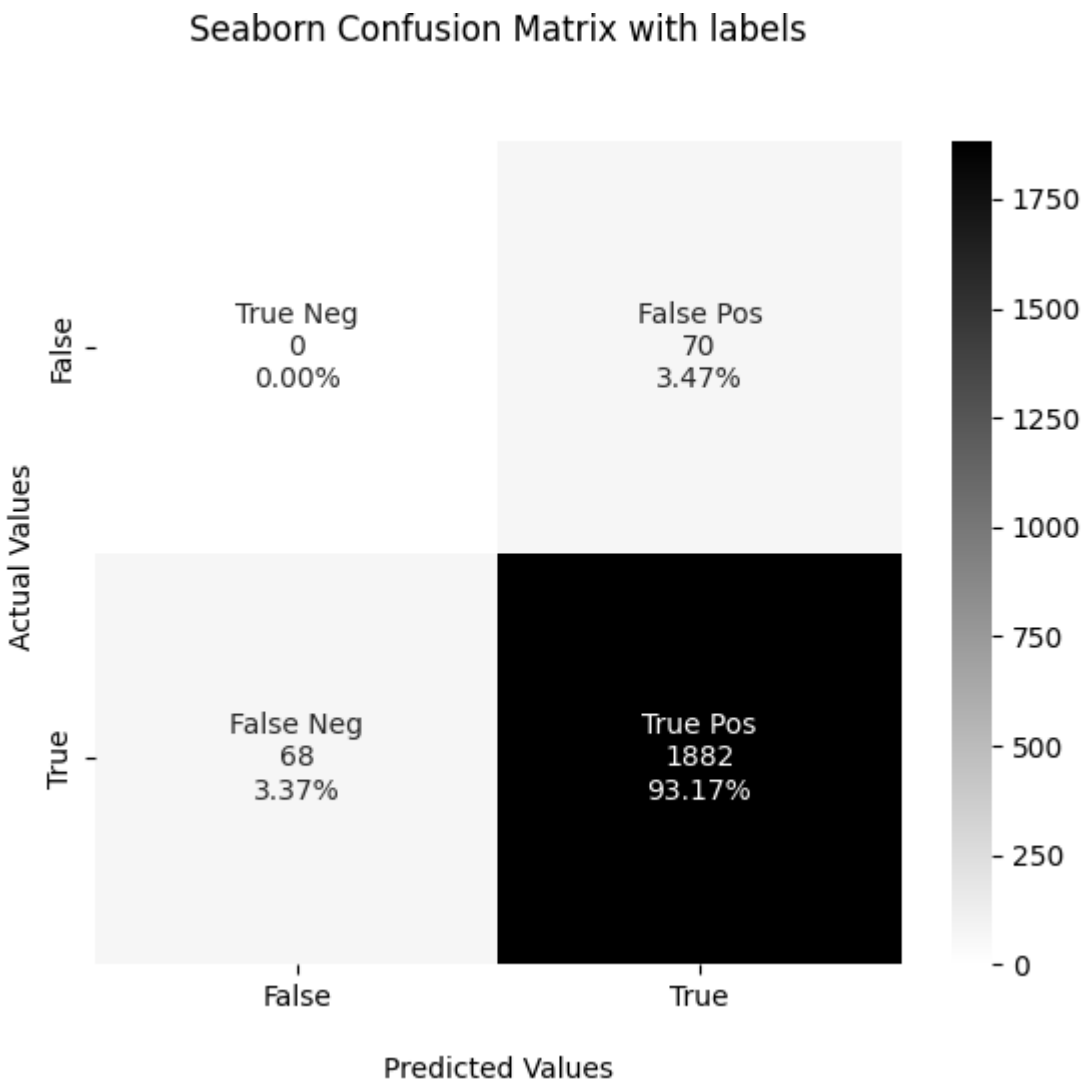


Figure 7.2: Confusion Matrix for CNN character recognition model

Chapter 8

Conclusion and Future Enhancements

Conclusion

To conclude, Automatic Nepali Number Plate Recognition system is completed successfully by making use of algorithms like YOLOv4 and CNN and deployed as a web app which will help user to detect and recognize number plates and store the details. Mean average precision of about 70% have been obtained for unseen data and 98% for seen data in YOLO. Also, the average character recognition accuracy of CNN is obtained to be approximately 95% and this accuracy mostly depends on the image processing technique used, so it may be increased by improving the image processing technique. The System has been developed which will give proper and accurate results for images but may need some improvements for videos due to long processing time in videos.

Future Enhancements

The ANNPR system is a system for number plate recognition, but it currently works for some limited number plates, mostly due to lack of proper dataset. So, In near future this system can be enhanced and some future enhancements that we can implement in this project are given below:

1. Achieving higher FPS when detecting for video
2. Model training on different zonal code Number Plate
3. Different Type Vehicle Classification
4. Training CNN model with more characters
5. Implementing system for embossed number plates
6. Implementing the system, so it works on live video feed
7. Improving the detection and character recognition accuracy

Bibliography

[1] "Automatic number-plate recognition," Nov 2021. [Online]. Available: https://en.wikipedia.org/wiki/Automatic_number-plate_recognition#In_mobile_systems

[2] A. K. Pant, P. K. Gyawali, and S. Acharya, "Automatic nepali number plate recognition with support vector machines," in *Proceedings of the 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, 2015, pp. 92–99.

[3] M. K. Sharma and B. Bhattarai, "Optical character recognition system for nepali language using convnet," in *Proceedings of the 9th International Conference on Machine Learning and Computing*, 2017, pp. 184–189.

[4] A. Vaishnav and M. Mandot, "An integrated automatic number plate recognition for recognizing multi language fonts," in *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2018, pp. 551–556.

[5] M. Avadesh and N. Goyal, "Optical character recognition for sanskrit using convolution neural networks," in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 2018, pp. 447–452.

[6] R. Laroca, E. Severo, L. A. Zanlorensi, L. S. Oliveira, G. R. Gonçalves, W. R. Schwartz, and D. Menotti, "A robust real-time automatic license plate recognition based on the yolo detector," in *2018 international joint conference on neural networks (ijcnn)*. IEEE, 2018, pp. 1–10.

[7] B. Setiyono, D. A. Amini, and D. R. Sulistyaningrum, "Number plate recognition on vehicle using yolo-darknet," in *Journal of Physics: Conference Series*, vol. 1821, no. 1. IOP Publishing, 2021, p. 012049.

[8] S. Prajapati, S. R. Joshi, A. Maharjan, and B. Balami, "Evaluating performance of nepali script ocr using tesseract and artificial neural network," in *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*. IEEE, 2018, pp. 104–107.

[9] "labelimg tool," 2022. [Online]. Available: https://github.com/tzutalin/labelImg

[10] "Nepali number plate dataset," 2021. [Online]. Available: https://github.com/Prasanna1991/LPR

[11] "Nepali number plate dataset 2," 2021. [Online]. Available: https://www.kaggle.com/pankajdawadi/nepali-lp-dataset

[12] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

Appendix

A Assistance for Running the Project

A.1 Automatic Nepali Number Plate Recognition

A.1.1 Clone Git Project

git clone "https://github.com/JuJu2181/annpr.git"

A.1.2 Installation Required

To run the project, pythonV3.8.5 is required.
To install the dependencies and libraries, run the following command:
pip install -r requirements.txt

A.1.3 Setting up in PC

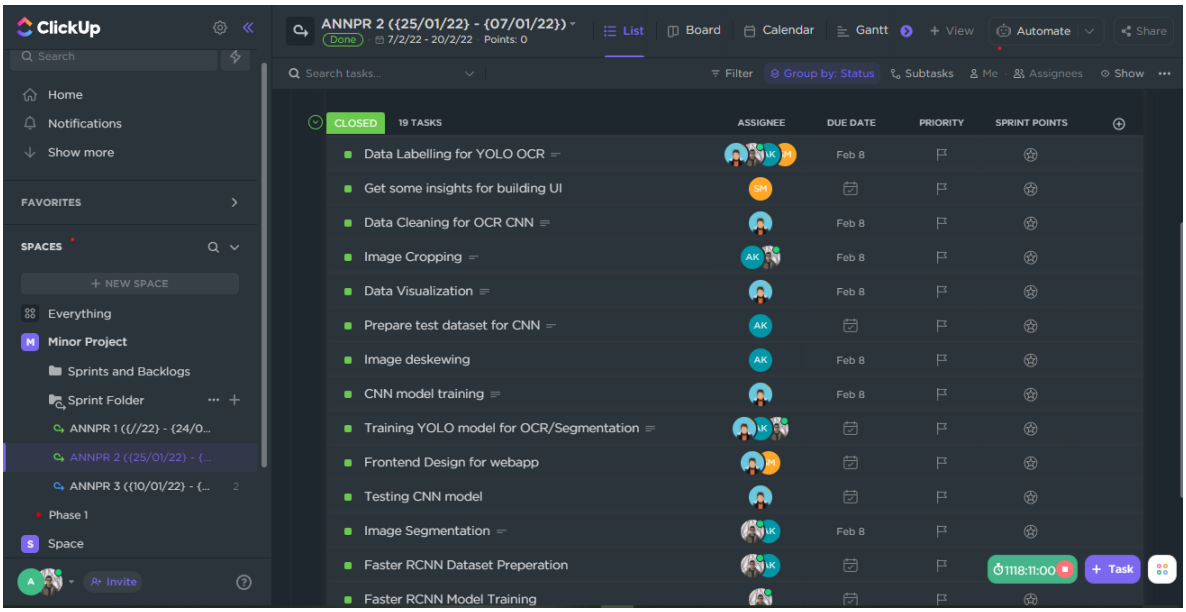
Download and add these two folders Own_cfg_and_weights and cnn_weights inside pytorch_YOLOv4 folder
Own_cfg_and_weights Link:**CFG AND WEIGHT**
CNN_weights Link:**CNN WEIGHT**

A.1.4 Running the Program

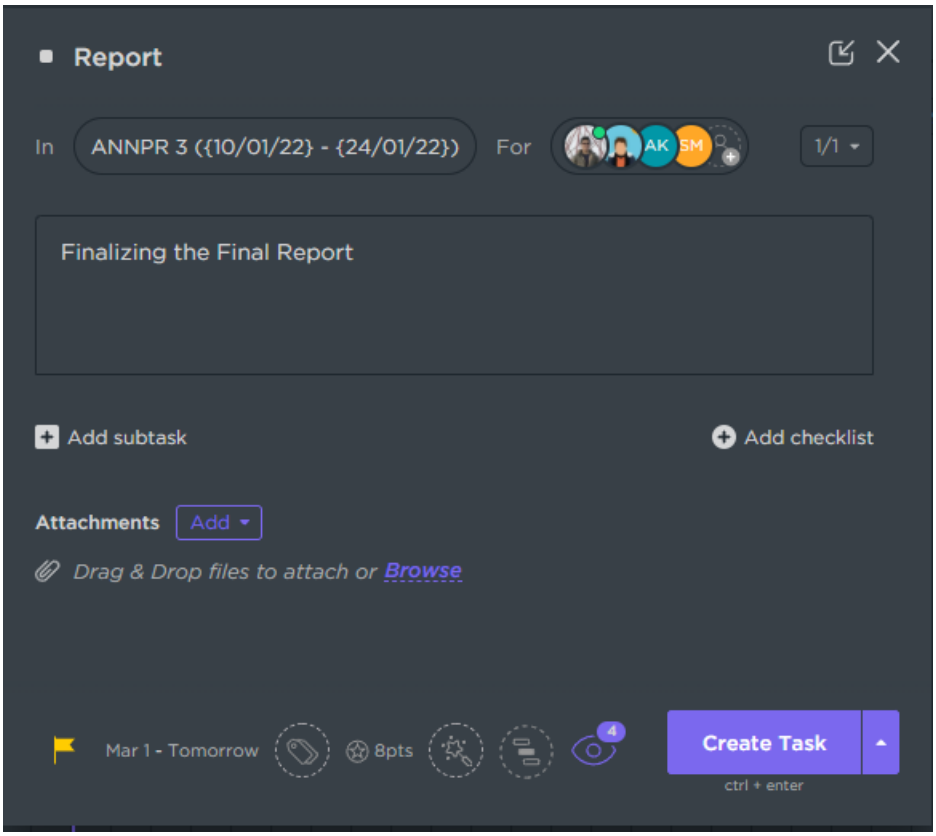
First, our working directory must be annpr. The following code helps to run this project:
cd annpr
python manage.py runserver

B Snapshot

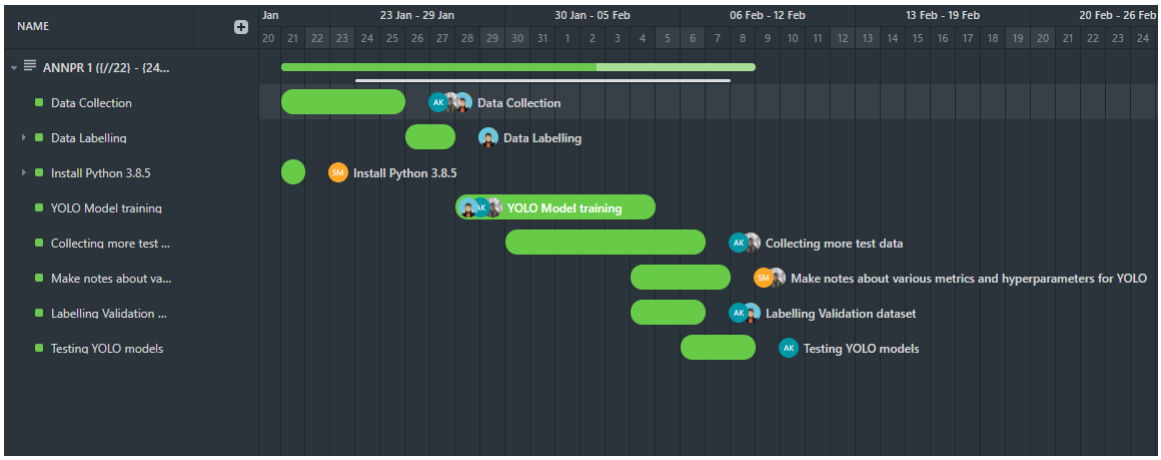
B.1 ClickUp



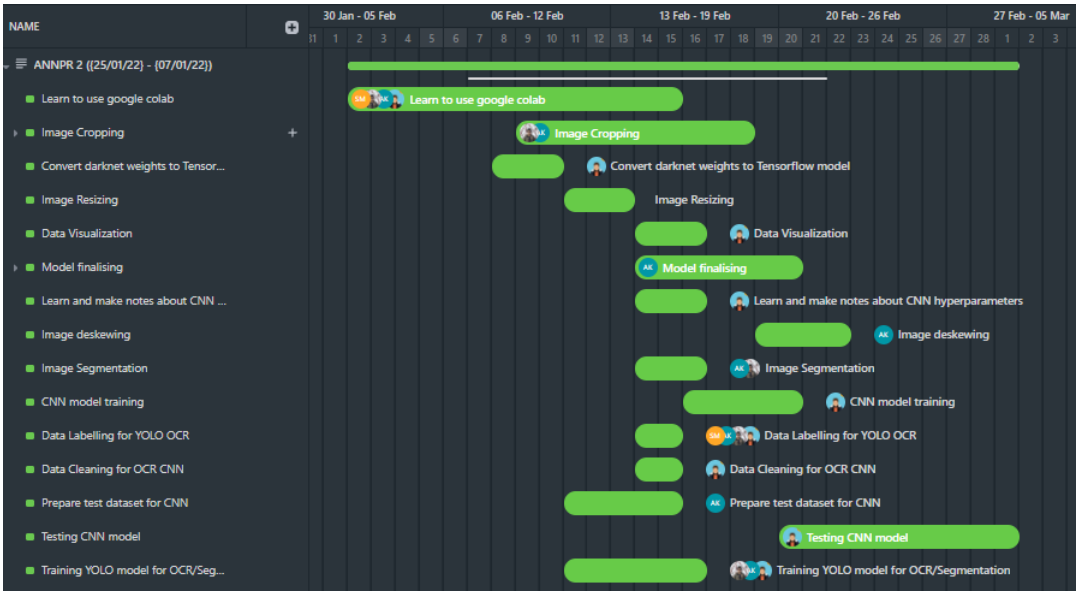
B.1.1 Create task



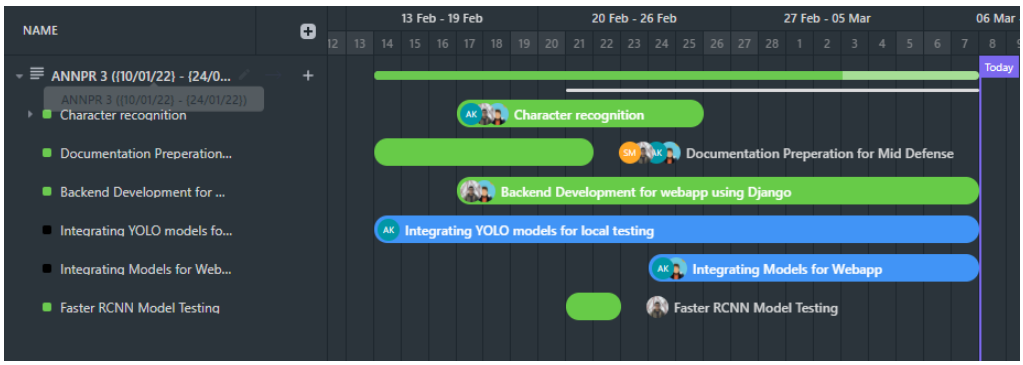
B.1.2 Sprint-1 Chart



B.1.3 Sprint-2 Chart



B.1.4 Sprint-3 Chart

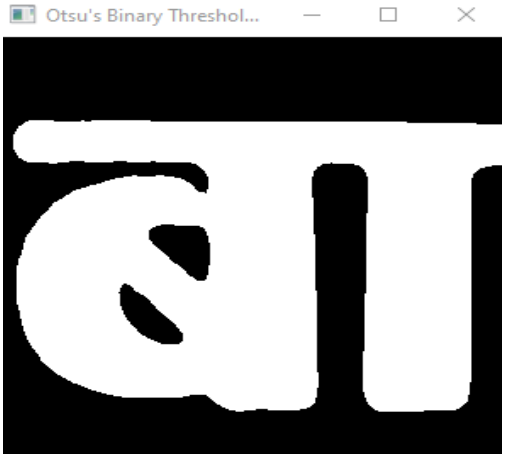


B.2 Image Processing

B.2.1 Different Outcomes in Image Processing Pipeline

(a) Original Input

(b) Grayscale Conversion



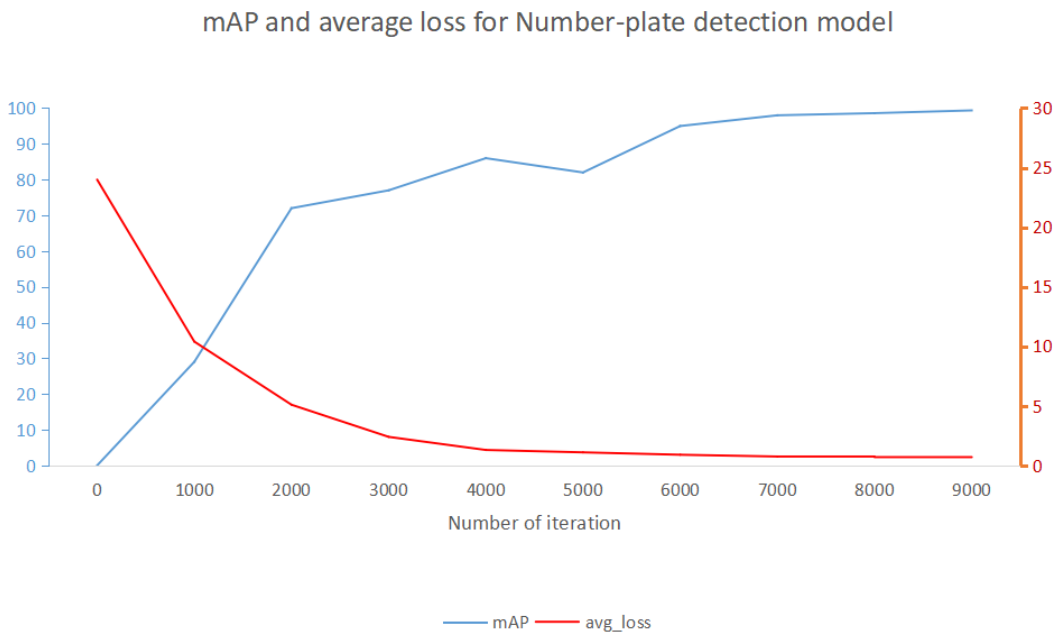
(c) Otsu's Binary Thresholding

(d) Median Filter

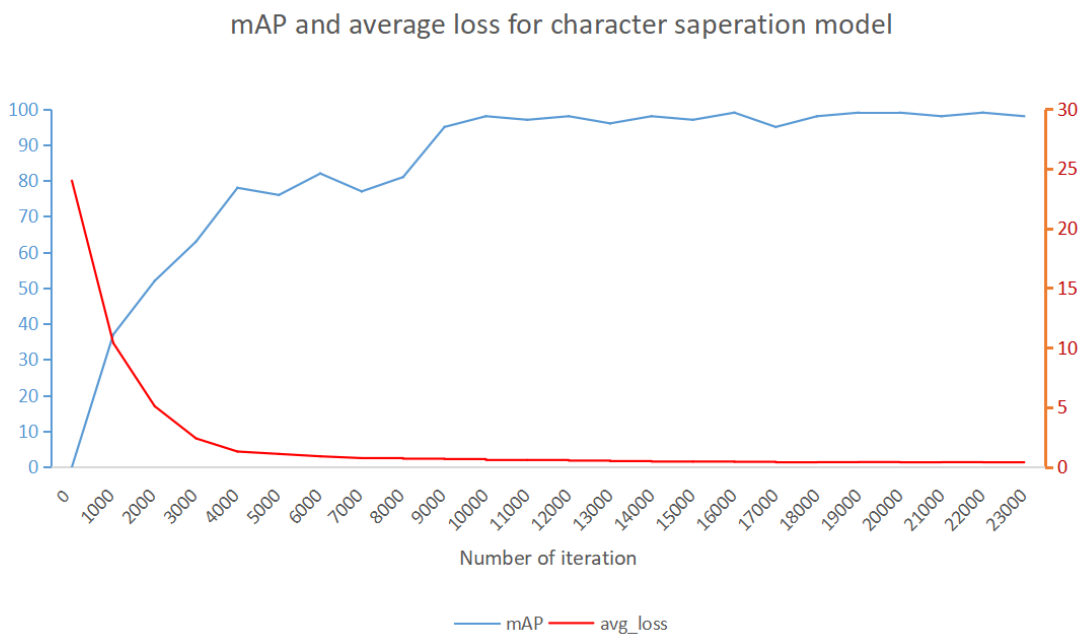
Figure 8.1: Steps in Image Processing

B.3 YOLO Model Visualization

B.3.1 YOLOv4 Number-plate detection

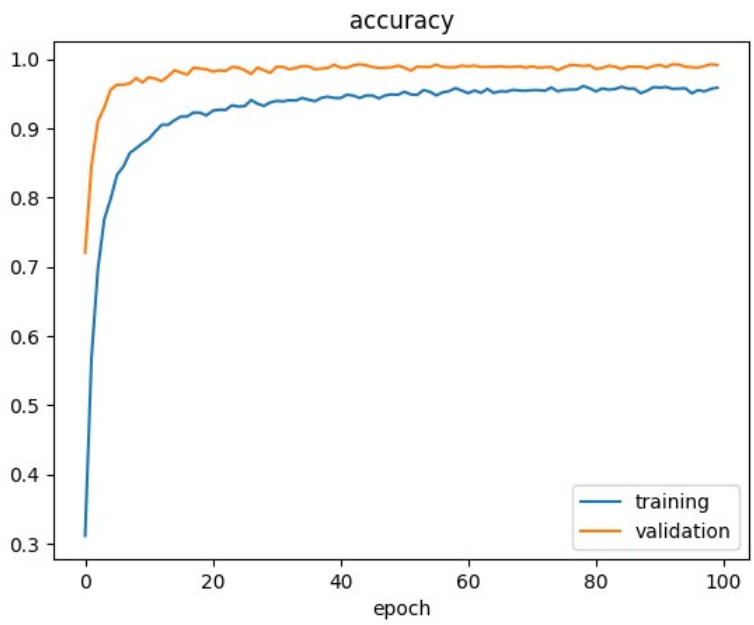


B.3.2 YOLOv4-tiny Character detection

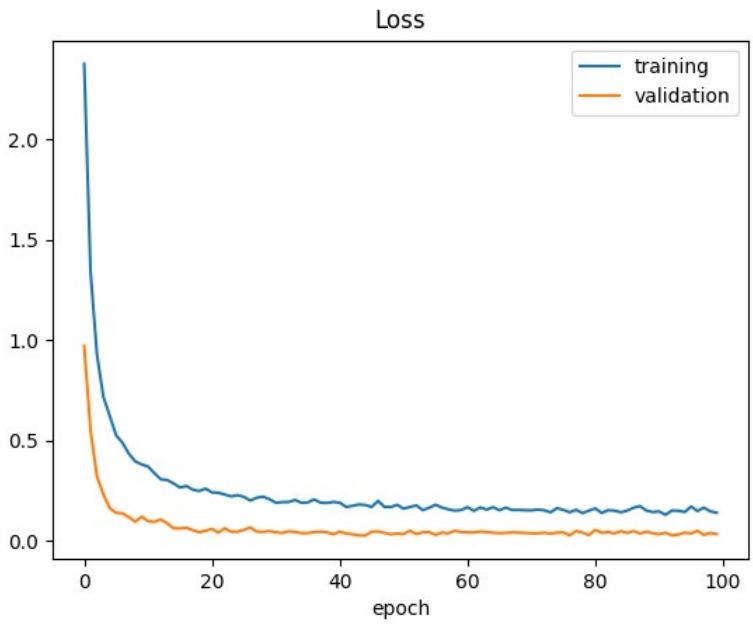


B.4 CNN Model Visualization

B.4.1 Plot for Accuracy



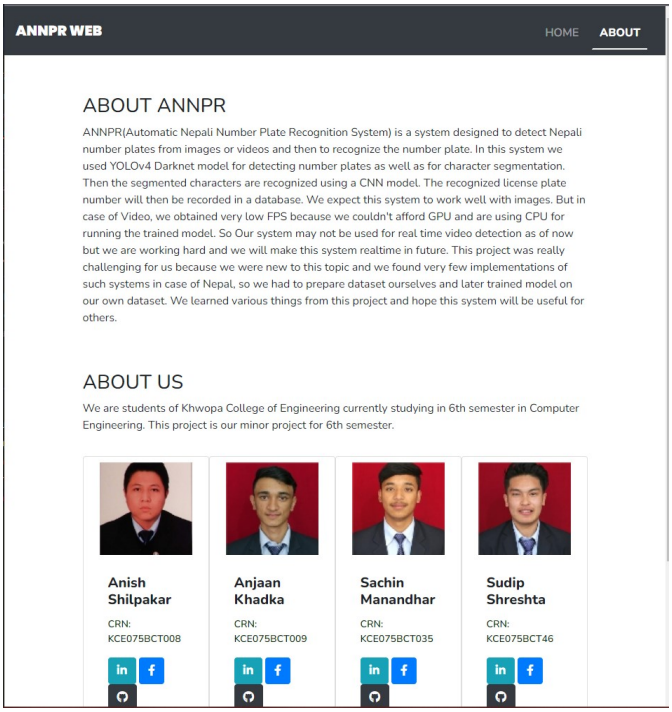
B.4.2 Plot of Loss



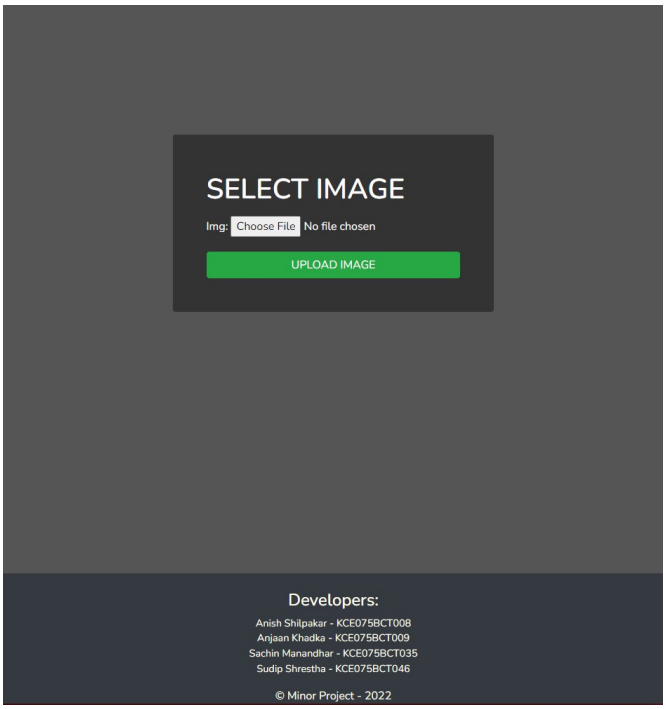
B.5 Web App

B.5.1 Home Page For Web App

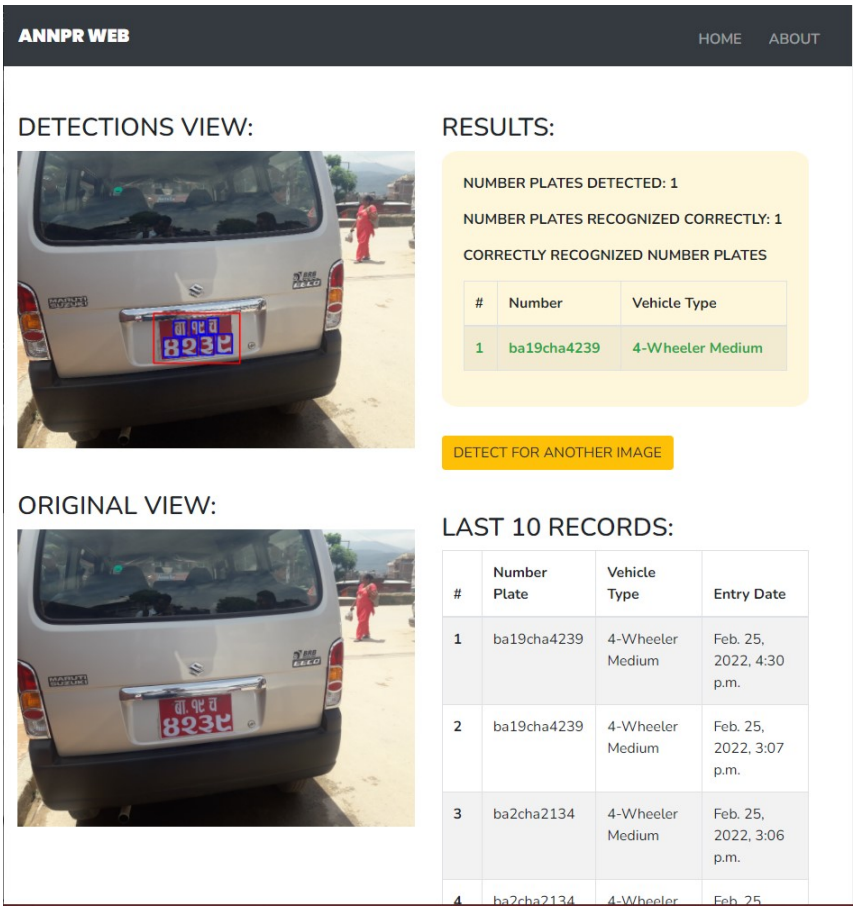
B.5.2 About Page For Web App



B.5.3 Upload Form For Web App



B.5.4 Detection Page For Web App



B.6 Unit Testing

B.6.1 License Plate Detection

(a) For busy road of Kathmandu

(b) For clearly seen number-plate

(c) For detection form video

Figure 8.2: Test case for Number-plate

B.6.2 Character Segmentation

(a) For 4:1 number-plate

(b) For 4:3 number-plate

(c) For slanted number-plate

(d) For very skewed number-plate

Figure 8.3: Test case for character separation

B.6.3 Character Recognition

(a) For character present in dataset (b) For character not present in dataset

Figure 8.4: Test case for character recognition

B.7 Integration Testing

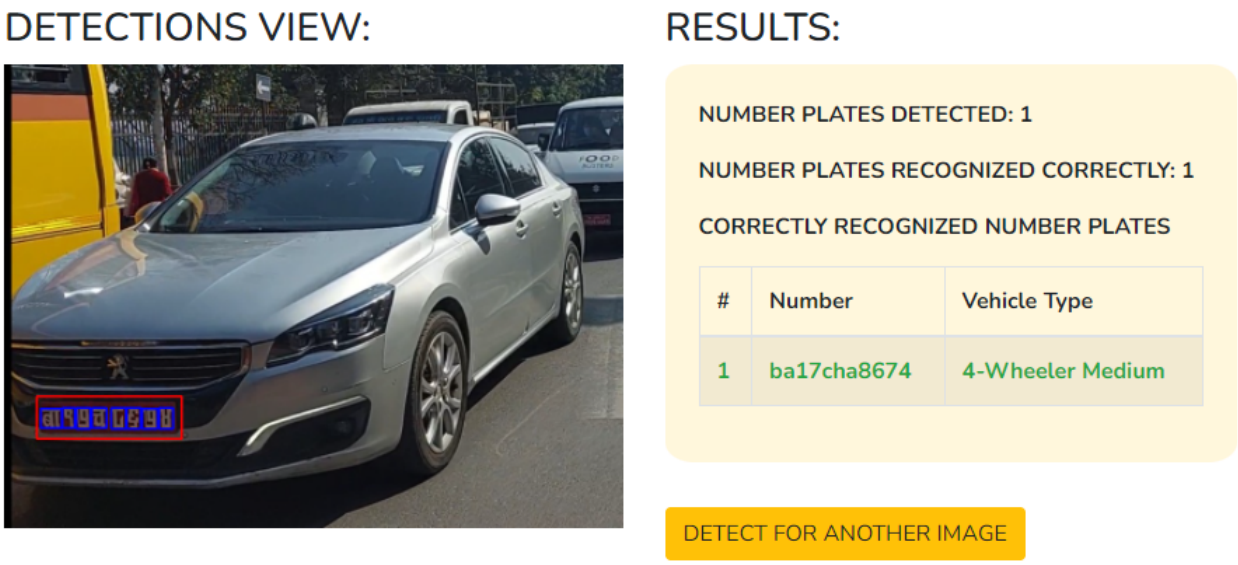


Figure 8.5: Test for image with clearly visible number plate

RESULTS:

NUMBER PLATES DETECTED: 1

NUMBER PLATES RECOGNIZED CORRECTLY: 1

CORRECTLY RECOGNIZED NUMBER PLATES

#	Number	Vehicle Type
1	ba11cha4898	4-Wheeler Medium

Figure 8.6: Test for image with clearly visible number plate of ration 4:3

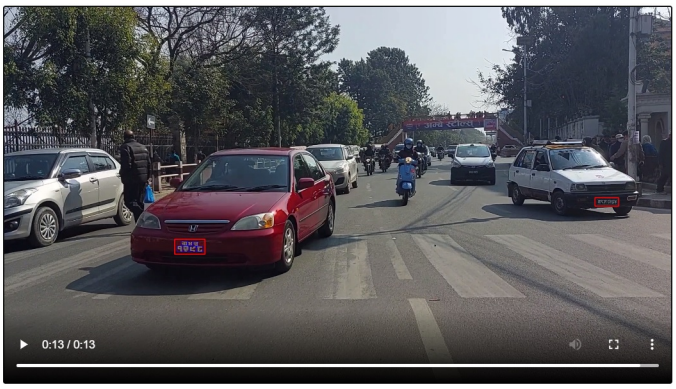
NUMBER PLATES DETECTED: 2

NUMBER PLATES RECOGNIZED CORRECTLY: 1

CORRECTLY RECOGNIZED NUMBER PLATES

#	Number	Vehicle Type
1	ba96pa2711	2-Wheeler

Figure 8.7: Test for image with multiple number plates



NUMBER PLATES RECOGNIZED CORRECTLY: 2

CORRECTLY RECOGNIZED NUMBER PLATES

#	Number	Vehicle Type
1	ba5cha1298	4-Wheeler-Medium
2	ba10cha7122	4-Wheeler-Medium

DETECT FOR ANOTHER VIDEO

Figure 8.8: Test for video with multiple cars passing