

**International Institute of
Information
Technology, Bangalore**
Software Production Engineering project
Car Number plate Detection

Under the Guidance of
Prof. B. Thangaraju
&
TA : Sanandan



Sai Krishna Kopparapu

IMT2019045

Naga Srihith Penjarla

IMT2019056

Table of Contents

1. Abstract.....	3
2. Introduction.....	4
2.1 Overview.....	4
2.2 Features.....	4
2.3 why DevOps.....	4
2.3.1 Devops features.....	4
3. System configuration.....	5
3.1 Operating System.....	5
3.2 CPU and Ram.....	5
3.3 Kernel Version.....	5
3.4 Language.....	5
3.5 Devops Tools.....	5
4. Software Development life cycle.....	5
4.1 Installation.....	5
4.1.1 python setup.....	5
4.2 Source control management.....	6
4.3 Building.....	7
4.4 Testing.....	8
4.4.1 CI workflow.....	10
4.5 Docker artifact.....	13
4.5.1 CI workflow.....	15
4.6 Deployment using ansible.....	17
4.7 Monitoring using ELK.....	19
4.8 Building workflow.....	23
5 Model.....	25
6. Results and Discussions.....	26
7. Future Scope.....	31
8. Conclusion & Summary.....	31
9. Links.....	32

1. Abstract:

Number plate detection allows computers to read and interpret vehicle number plates from digital images. It involves the use of image processing techniques to detect, segment, and recognize characters on license plates, which can be used for various applications such as traffic monitoring, law enforcement, and toll collection.

The number plate detection problem is complex and challenging, as it can appear in different sizes, shapes, orientations, and lighting conditions, and can be partially obscured or blurred. In addition, number plates can contain different fonts, colors, and background designs, which further complicates the task of detecting and recognizing them accurately and reliably.

In this project we first used CNN for identifying number plates, but it wasn't performing well. So, we then shifted to the YOLOv5 model to identify the number plates. The dataset that we used contains the pictures of car number plates taken from different angles and different lighting systems.

We design an application where one can upload the image containing a car's number plate to see the output our trained model gives.

[Link to the dataset](#)

The architecture of our project demands two layers.

- Front end.
- Back end.

The front end of the project is handled by "html", javascript and the backend is swiftly handled by "Python", using "Flask" for interaction between ends.

2 Introduction

2.1. OVERVIEW

Our project is about the number plate detection on cars. We can feed the image of a car along with the number plate and our website can identify the Number plate as well as parse the text on the number plate. We designed this website keeping in mind the traffic monitoring, law enforcement, and toll collection. They can just upload the pictures of the cars they have taken and our website gives them the number plate and text on it.

2.2 FEATURES

- **Number Plate Detection:** Given a picture, the website identifies the bounding box/region where the number plate is located on the car.
- **Text Detection on the number plate:** The website also parses/gives us the text present on the number plate it identified.

2.3. WHY DEVOPS?

Our whole approach to the project was modular, we wanted to make different sets of development modules and wanted to deploy them with every new release without any hindrance. Wanted to test the changed code with continuous integration and then continuously deploying it. Devops provides all the tools to increase the capability to complete above set goals within minimum time and less trouble for developers. DevOps tools consist of configuration management, test and build systems, application deployment, version control and monitoring tools. Continuous integration, continuous delivery and continuous deployment require different tools.

2.3.1. Devops Features

- Improve deployment frequency
- Achieve faster time to market with lower failure rate
- More stable operating environments
- Improve communication and collaboration among teams

3. System Configuration

3.1. Operating system

Ubuntu 18.04 or above

3.2. RAM

RAM 4 GB or above

3.3 KERNEL VERSION

Linux machine 5.8.0-44-generic

3.4. Language

python3, Flask framework.

3.5. DevOps Tools

- Source Control Management - GitHub
- Continuous Integration – Jenkins
- Containerization - Docker
- Continuous deployment – Ansible
- Monitoring - ELK Stack (Elastic Search, Logstash, Kibana)

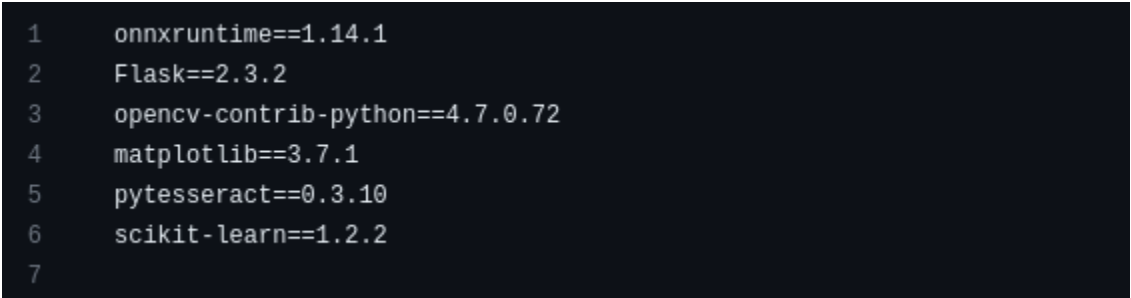
4. Software Development life cycle

4.1. INSTALLATION

4.1.1 Python Setup

The entire ML code is written in python and the web server is written using flask, a library in python. To install all those libraries required we use pip.

python3 -m pip install -r requirements.txt- This command installs the version of libraries into the environment using requirements.txt



```
1 onnxruntime==1.14.1
2 Flask==2.3.2
3 opencv-contrib-python==4.7.0.72
4 matplotlib==3.7.1
5 pytesseract==0.3.10
6 scikit-learn==1.2.2
7
```

Fig 1 : requirements.txt looks like this.

For parsing the Number Plate from the image, we need *tesseract-ocr* along with *pytesseract* (which will be installed using pip). To install *tesseract-ocr*, we have to use the command “*sudo apt install tesseract-ocr -y*”.

4.2. SOURCE CONTROL MANAGEMENT

A Source Code Management (SCM) is a software tool used by programmers to manage the source codes. For our project every team member would clone the repository from github, create a different branch locally on their system and then merge it with master. The cycle of pulling the latest code from git, resolving any conflicts and then pushing the changes to git keeps happening for every update.

- **git clone** - This command copies the entire data on the git url
- **git add** -This command adds changes in the working directory to the staging area
- **git commit -m “commit message”**-This command is used to save your changes to the local repository with -m used to provide a concise description that helps your teammates (and yourself) understand what happened.
- **git pull**-This command is used to update the local version of a repository from a remote.
- **git push**-This command will push all the latest code to the repository.

For this project GitHub link used was

<https://github.com/kopparapusaikrishna/Number-Plate-detection.git>

4.3 BUILDING

PYTHON:

To build the python project we use requirements.txt file, and using pip3 we install all the dependent libraries for the project. Here's the link to complete requirements.txt

<https://github.com/kopparapusaikrishna/Number-Plate-detection/blob/main/requirements.txt> requirements.tx

4.4 TEST

For testing the project, we used unit tests. Test cases are written in keeping a view to cover all possible cases.

Steps to use unittest.

- 1 Import unittest from the standard library
- 2 Create a class called TestModel that inherits from the TestCase class
- 3 Convert the test functions into methods by adding self as the first argument
- 4 Change the assertions to use the self.assertEqual() method on the TestCase class
- 5 Change the command-line entry point to call unittest.main()

```
def test1(self):
    text_list, bb_confidences = test_function_for_bb_detections(get_image(2))
    self.assertEqual(text_list[0], "MH20BY3665")
    self.assertNotEqual(text_list[0], -1)
    if text_list[0]==-1:
        print("No Number Plate found in Image-1")
    else:
        print("Bounding box of Image-1 has been predicted with an accuracy of", bb_confidences[0], "and the detected Number plate text is", text_list[0])

def test2(self):
    text_list, bb_confidences = test_function_for_bb_detections(get_image(4))
    self.assertEqual(text_list[0], "MH14EU3498")
    self.assertNotEqual(text_list[0], -1)
    if text_list[0]==-1:
        print("No Number Plate found in Image-2")
    else:
        print("Bounding box of Image-2 has been predicted with an accuracy of", bb_confidences[0], "and the detected Number plate text is", text_list[0])

def test3(self):
    text_list, bb_confidences = test_function_for_bb_detections(get_image(9))
    self.assertEqual(text_list[0], "HR26U7501")
    self.assertNotEqual(text_list[0], -1)
    if text_list[0]==-1:
        print("No Number Plate found in Image-3")
    else:
        print("Bounding box of Image-3 has been predicted with an accuracy of", bb_confidences[0], "and the detected Number plate text is", text_list[0])

def test4(self):
    text_list, bb_confidences = test_function_for_bb_detections(get_image(6))
    self.assertEqual(text_list, -1)
    self.assertNotEqual(text_list, "SaiKsk")
    if text_list==-1:
        print("No Number Plate found in Image-4")
    else:
        print("Bounding box of Image-4 has been predicted with an accuracy of", bb_confidences[0], "and the detected Number plate text is", text_list[0])

def test5(self):
    text_list, bb_confidences = test_function_for_bb_detections(get_image(8))
    self.assertEqual(text_list[0], "DL3CAY9324")
    self.assertNotEqual(text_list, -1)
    if text_list==-1:
        print("No Number Plate found in Image-5")
    else:
        print("Bounding box of Image-5 has been predicted with an accuracy of", bb_confidences[0], "and the detected Number plate text is", text_list[0])

def test6(self):
    text_list, bb_confidences = test_function_for_bb_detections(get_image(7))
    self.assertEqual(text_list[0], "GJW115A1138")
    self.assertNotEqual(text_list, -1)
    if text_list==-1:
        print("No Number Plate found in Image-6")
    else:
        print("Bounding box of Image-6 has been predicted with an accuracy of", bb_confidences[0], "and the detected Number plate text is", text_list[0])
```

Fig 2: Unit test cases in our project


```

nagasrihith608@srihith-Laptop:~/Desktop/Number-Plate-detection$ python3 testing.py
Beginning Testing ...

Bounding box of Image-1 has been predicted with an accuracy of 0.9110005497932434 and the detected Number plate text is MH20BY3665
.Bounding box of Image-2 has been predicted with an accuracy of 0.9093791246414185 and the detected Number plate text is MH14EU3498
.FNo Number Plate found in Image-4
.Bounding box of Image-5 has been predicted with an accuracy of 0.8908911347389221 and the detected Number plate text is DL3CAY9324
.F
=====
FAIL: test3 (__main__.TestModel)
=====
Traceback (most recent call last):
  File "/home/nagasrihith608/Desktop/Number-Plate-detection/testing.py", line 46, in test3
    self.assertEqual(text_list[0], "HR26U7501")
AssertionError: 'ROVE' != 'HR26U7501'
- ROVE
+ HR26U7501

=====
FAIL: test6 (__main__.TestModel)
=====
Traceback (most recent call last):
  File "/home/nagasrihith608/Desktop/Number-Plate-detection/testing.py", line 79, in test6
    self.assertEqual(text_list[0], "GJW115A1138")
AssertionError: 'GUW115A1138' != 'GJW115A1138'
- GUW115A1138
? ^
+ GJW115A1138
? ^

-----
Ran 6 tests in 1.946s

FAILED (failures=2)

```

Fig 3: Running all our test cases and output shows the prediction accuracy along with the parsed text or identifies incorrect detections.

4.4.1 CI workflow(Jenkins):

Jenkins is an open-source automation server that facilitates the continuous integration and delivery of software. It provides a framework for automating various aspects of the software development lifecycle, such as building, testing, and deploying applications. By integrating with version control systems, Jenkins detects changes in code repositories and triggers automated builds and tests, helping to identify issues early on. Its extensible architecture allows for integration with a wide range of tools and technologies commonly used in software development. With Jenkins, developers can streamline their workflows, increase productivity, and ensure that software remains in a releasable state.

One of the key features of Jenkins is its support for building pipelines, where multiple stages of the development process can be defined and executed sequentially. This enables the creation of complex workflows that include tasks like compilation, testing, code analysis, packaging,

and deployment. Moreover, Jenkins provides monitoring capabilities, visualizing build trends and test results, and supporting notifications through email, chat, and other systems to keep teams informed about build statuses and failures.

The first step of the Jenkins pipeline is to pull the code from GitHub. The pipeline script for this looks as shown below.

```
stages {
    stage('Git Pull') {
        steps {
            // Get some code from a GitHub repository
            git url: 'https://github.com/kopparapusaikrishna/Number-Plate-detection.git',
              branch: 'main'
        }
    }
}
```

Fig 4a: Pulling the code from github

```
15
16
17
18
19
20
21
22
23
24
25
26

stage('Testing') {
    steps {
        sh 'pip3 install -r requirements.txt'
        sh 'sudo apt-get update'
        sh 'sudo apt-get install -y tesseract-ocr'
        sh 'sudo apt-get install -y libgl1-mesa-glx'
        sh 'python3 testing.py'
    }
}
```

Fig 4b: Running test cases by first installing the requirements.

4.5. DOCKER ARTIFACT

Docker is a software platform for building applications based on containers which are small and lightweight execution environments that make shared use of the operating system kernel but otherwise run in isolation from one another. Docker images of the module are created and then pushed to Docker Hub, from where we can pull those images and run it at our end. Dockerfile is created for the project that will run when we build these Docker images.

<https://github.com/kopparapusaikrishna/Number-Plate-detection/blob/main/Dockerfile>

```

Dockerfile
1 FROM python:3.10.6
2
3 COPY . /myapp
4 WORKDIR /myapp
5
6 RUN apt-get update && apt-get install -y \
7     tesseract-ocr \
8     libgl1-mesa-glx
9
10 RUN pip3 install -r requirements.txt
11
12 EXPOSE 5000
13 CMD ["python3", "app.py"]

```

Fig 5: Docker file for Number plate detection project, used to build docker image.

Command to build docker image in terminal:

```

nagasrith608@srihith-Laptop: ~/Desktop/Number-Plate-detection$ docker build -t new_image .
Sending build context to Docker daemon 245.3MB
Step 1/7 : FROM python:3.10.6
--> d25a66380b10
Step 2/7 : COPY . /myapp
--> 041bb45797e7
Step 3/7 : WORKDIR /myapp
--> Running in 32a97c55517b
Removing intermediate container 32a97c55517b
--> b0d49d8c98f8
Step 4/7 : RUN pip3 install -r requirements.txt
--> Running in 7792df858241
Collecting Flask==2.3.2
  Downloading Flask-2.3.2-py3-none-any.whl (96 kB)
    96.9/96.9 kB 107.1 kB/s eta 0:00:00
Collecting opencv-contrib-python==4.7.0.72
  Downloading opencv_contrib_python-4.7.0.72-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (67.9 MB)
    67.9/67.9 MB 3.6 MB/s eta 0:00:00
Collecting matplotlib==3.7.1
  Downloading matplotlib-3.7.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
    11.6/11.6 MB 6.4 MB/s eta 0:00:00
Collecting pytesseract==0.3.10
  Downloading pytesseract-0.3.10-py3-none-any.whl (14 kB)
Collecting scikit-learn==1.2.2
  Downloading scikit_learn-1.2.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (9.6 MB)
    9.6/9.6 MB 6.4 MB/s eta 0:00:00
Collecting Werkzeug==2.3.3
  Downloading Werkzeug-2.3.4-py3-none-any.whl (242 kB)
    242.5/242.5 kB 979.8 kB/s eta 0:00:00
Collecting click>=8.1.3
  Downloading click-8.1.3-py3-none-any.whl (96 kB)
    96.6/96.6 kB 349.6 kB/s eta 0:00:00
Collecting Jinja2>=3.1.2
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
    133.1/133.1 kB 162.4 kB/s eta 0:00:00
Collecting blinker>=1.6.2
  Downloading blinker-1.6.2-py3-none-any.whl (13 kB)
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting numpy>=1.17.0
  Downloading numpy-1.24.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
    17.3/17.3 MB 7.3 MB/s eta 0:00:00
Collecting cycler>=0.10
  Downloading cycler-0.11.0-py3-none-any.whl (6.4 kB)
Collecting packaging>=20.0
  Downloading packaging-23.1-py3-none-any.whl (48 kB)
    48.9/48.9 kB 155.1 kB/s eta 0:00:00
Collecting pyparsing>=2.3.1
  Downloading pyparsing-3.0.9-py3-none-any.whl (98 kB)
    98.3/98.3 kB 348.6 kB/s eta 0:00:00
Collecting contourpy>=1.0.1
  Downloading contourpy-1.0.7-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (300 kB)
    300.3/300.3 kB 1.2 MB/s eta 0:00:00

```

Fig 6: Building of docker image in local system.

```
nagasrihith608@srihith-Laptop:~/Desktop/Number-Plate-detection$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
new_image	latest	d761977401b5	About a minute ago	2.02GB
<none>	<none>	255c41176c0c	16 minutes ago	2.15GB
<none>	<none>	8507c5ecdadea	43 minutes ago	2.11GB
<none>	<none>	d5815625d819	51 minutes ago	1.13GB
<none>	<none>	7281cef5f1fc	55 minutes ago	2.04GB
<none>	<none>	783cfecea46f	2 hours ago	2.1GB
<none>	<none>	cfaebe7ce11a	2 hours ago	1.13GB
<none>	<none>	efa3d6aa7a42	2 hours ago	2.06GB
<none>	<none>	1f13c07ba6f1	3 hours ago	1.13GB
<none>	<none>	52a88a172749	3 hours ago	1.13GB
<none>	<none>	ced3b5fc28bc	3 hours ago	1.34GB
<none>	<none>	83b962ba3b83	3 hours ago	1.34GB
<none>	<none>	2a491ce4051b	3 hours ago	1.34GB
<none>	<none>	66f437663206	3 hours ago	1.34GB
npdi	latest	770f7b238b2d	12 hours ago	2.03GB
<none>	<none>	82ad7a3f927e	13 hours ago	1.34GB
<none>	<none>	8f5b2a077687	13 hours ago	1.99GB
major	latest	61f7503b0583	13 hours ago	1.77GB
majorproject	latest	61f7503b0583	13 hours ago	1.77GB
<none>	<none>	e472962ee7c7	14 hours ago	1.99GB
<none>	<none>	ad6add36ed1d	2 days ago	1.81GB

Fig 7: command to check docker images after building.

```
nagasrihith608@srihith-Laptop:~/Desktop/Number-Plate-detection$ sudo docker run -it new_image /bin/bash
root@e9bc6898272c:/myapp# ls
Dockerfile  Verification  __pycache__  app.py  app_model.py  detection_logs.log  inventory  playbook.yml  reqs.txt  requirements.txt  static  templates  testing.py
root@e9bc6898272c:/myapp# cd static/
root@e9bc6898272c:/myapp/static# ls
Verification  background.jpeg  models  predict  roi  upload
root@e9bc6898272c:/myapp/static# cd models/
root@e9bc6898272c:/myapp/static/models# ls
best.onnx
```

Fig 8: Command to go into the docker container

```
nagasrihith608@srihith-Laptop:~/Desktop/Number-Plate-detection$ sudo docker run -it -p 5000:5000 new_image
```

```
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 878-213-027
```

```
172.17.0.1 - - [14/May/2023 10:50:15] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [14/May/2023 10:50:16] "GET /static/Verification/1.jpg HTTP/1.1" 200 -
172.17.0.1 - - [14/May/2023 10:50:16] "GET /static/Verification/3.jpg HTTP/1.1" 200 -
172.17.0.1 - - [14/May/2023 10:50:16] "GET /static/Verification/5.jpg HTTP/1.1" 200 -
172.17.0.1 - - [14/May/2023 10:50:16] "GET /static/Verification/11.jpg HTTP/1.1" 200 -
172.17.0.1 - - [14/May/2023 10:50:17] "GET /favicon.ico HTTP/1.1" 404 -
MH46X9996
172.17.0.1 - - [14/May/2023 10:50:21] "POST /process_image HTTP/1.1" 200 -
172.17.0.1 - - [14/May/2023 10:50:21] "GET /static/predict/1.jpg HTTP/1.1" 200 -
TN21AT0480
172.17.0.1 - - [14/May/2023 10:50:30] "POST /process_image HTTP/1.1" 200 -
172.17.0.1 - - [14/May/2023 10:50:30] "GET /static/predict/11.jpg HTTP/1.1" 200 -
TN21AT0480
172.17.0.1 - - [14/May/2023 10:50:33] "POST /process_image HTTP/1.1" 200 -
172.17.0.1 - - [14/May/2023 10:50:34] "GET /static/predict/3.jpg HTTP/1.1" 200 -
```

Fig 9: Command to run docker image.

4.5.1 CI Workflow

First we need to add credentials for our docker hub and some others if necessary.

To do that

1. We have to go to Manage Jenkins from the Jenkins dashboard.
2. Then select credentials.
3. Now when we click on global in “*Stores scoped to Jenkins*” section, you can find the Add credentials button from where you can add the credentials to the docker hub.

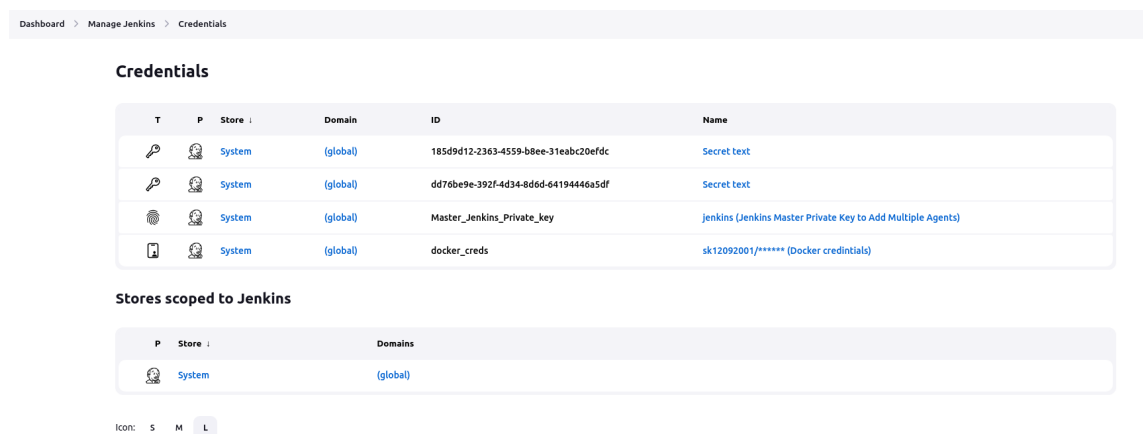


Fig 10a: Way to add credentials

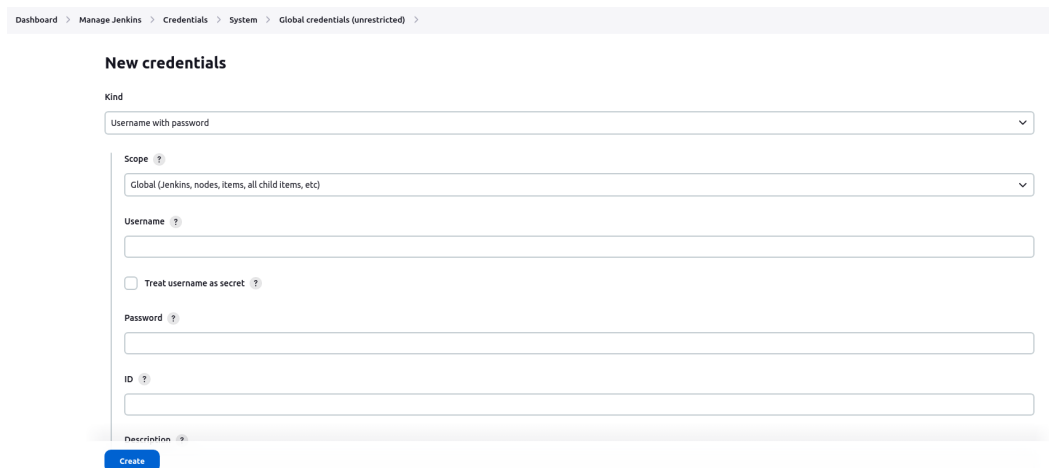


Fig 10b: way to add credentials.

- Now we will add the jenkins pipeline for building the docker image and pushing it to docker hub.
- We have to make sure that the docker credential here in script matches our docker credential id set earlier.

```

15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
stage('Docker Build Image')
{
    steps{
        script{
            imageName = docker.build "sk12092001/number_plate_detection:latest"
        }
    }
}
stage('Push Docker Image')
{
    steps{
        script{
            docker.withRegistry("", 'docker_creds' ){
                imageName.push()
            }
        }
    }
}

```

Fig 11: Building and deploying docker image on docker hub in jenkins pipeline.

- After successful deployment of docker images we can see the result on our docker-hub account.
- https://hub.docker.com/repository/docker/sk12092001/number_plate_detection/general

The screenshot shows the Docker Hub interface for the repository `sk12092001/number_plate_detection`. The page includes a search bar, navigation links (Explore, Repositories, Organizations, Help), and a user profile for `sk12092001`. The repository page has tabs for General, Tags, Builds, Collaborators, Webhooks, and Settings. The General tab is active, showing a description field, a 'Last pushed' timestamp of 2 hours ago, and a table of tags. The 'latest' tag is listed with OS 'linux', Type 'Image', and pushed 2 hours ago. There are also sections for Docker commands and Automated Builds.

Tag	OS	Type	Pulled	Pushed
latest	linux	Image	2 hours ago	2 hours ago

Fig 12: Docker Hub repository

4.6 Deployment using ansible:

```
1  ---
2  - name: Pull docker image
3    hosts: all
4    tasks:
5      - name: Pull the image from dockerhub
6        docker_image:
7          name: nagasrihith608/numberplatedetection
8          source: pull
9
10     - name: Write command after pulling image
11       shell: 'echo "Finished pulling image"'
12
13     - name: running container
14       shell: docker run -it -d nagasrihith608/numberplatedetection /bin/bash
```

Fig 13: Playbook file

```
inventory
1  localhost ansible_user=nagasrihith608 ansible_connection=local ansible_python_interpreter=/usr/bin/python3
2
```

Fig 14: inventory file

```
34     stage('Ansible deploy')
35     {
36     steps{
37         sh "/usr/bin/pip3 install docker"
38         sh "ansible-playbook playbook.yml -i inventory"
39     }
40     }
```

Fig 15: Pipeline script for ansible deploy

```
nagasrihith608@srihith-Laptop:~/Desktop/Number-Plate-detection/static/models$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nagasrihith608/numberplatedetection	latest	9d227aabe8c2	2 minutes ago	2.12GB
new_image	latest	de865e09d963	About an hour ago	2.1GB

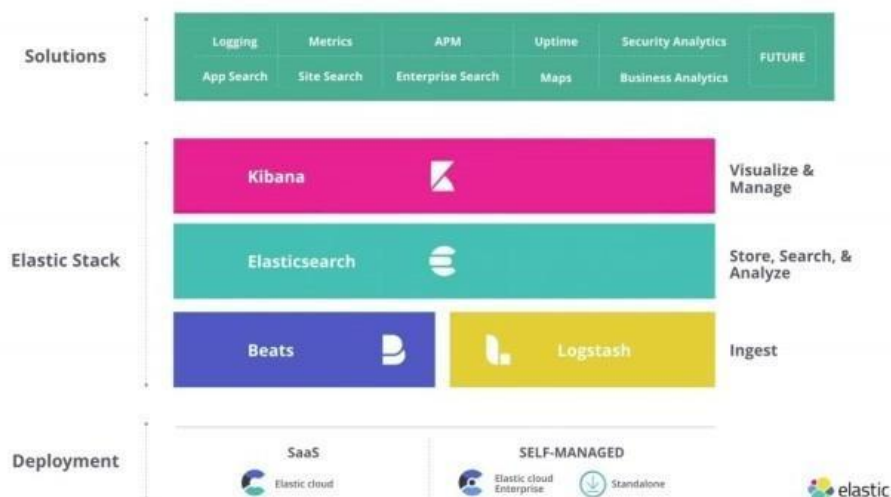
Fig 16: You can find an image with name ***“nagasrihith608/numberplatedetection”*** has been deployed from Jenkins pipeline using ansible.

4.7 Monitoring using ELK

Continuous monitoring provides near immediate feedback and insight into performance and interactions across the network.

"ELK" is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana .

- Elasticsearch is a search and analytics engine.
- Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch.
- Kibana lets users visualize data with charts and graphs in Elasticsearch.



- We generated logs and stored them in a file “app.txt”. Then we uploaded this file in Elastic and set the grok pattern to
`%{YEAR:year}-%{MONTHNUM:month}-%{MONTHDAY:day} %{TIME:time}
%{WORD:logger} %{LOGLEVEL:loglevel} %{GREEDYDATA:message}`
- Then we performed visualizations which are shown below.

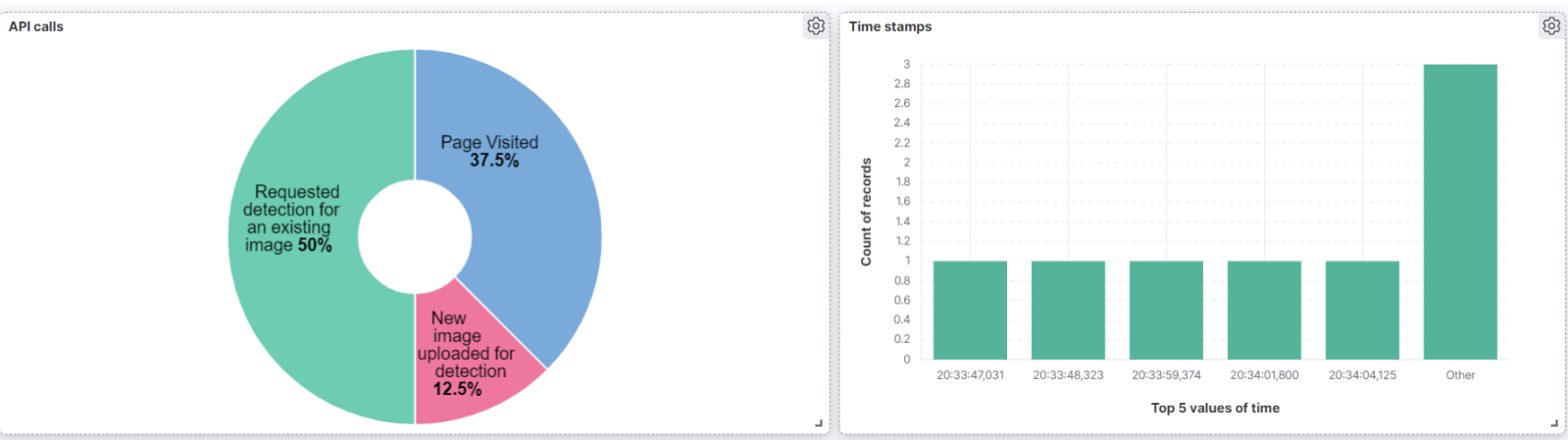


Fig 17: The left figure shows the No.of times each API call has been done.

4.8. BUILDING WORKFLOW

From the above sections we have seen how we have integrated docker and other necessary tools with the jenkins pipeline. We were able to build docker images with jenkins. Here’s the reference figure to the entire pipeline and its final full stage view after a successful run on jenkins.



Fig 18: Jenkins Pipeline

5. Model:

1. Dataset:

- The dataset contains 433 car images and their corresponding xml files.
- XML files contain annotations which represent the location of the number plate in the image.
- The four parameters present in the annotations are
 - i. x_{min}
 - ii. x_{max}
 - iii. y_{min}
 - iv. y_{max}
- These four parameters form a rectangle on the input image and it is called the bounding box of the image.

2. PreProcessing:

- Preprocessing for CNN model:
 - i. While using CNN model we gave images as input and we asked the model to predict the list $[x_{max}, y_{max}, x_{min}, y_{min}]$
 - ii. The dataset contains images of different sizes. But the problem here is that the size of the input image to the model should be the same.
 - iii. To handle this problem we should resize the every input image to a fixed shape which we used is 200×200 .
 - iv. But when we resize the image, the x_{min} , y_{min} , x_{max} and y_{max} will also change.
 - v. To overcome this problem we re-compute the values accordingly.
 - vi. One more thing to note here is that the values x_{center} , y_{center} , width, height should be in between 0 and 1. So to achieve this we performed normalization.
- Preprocessing for YOLOv5 model:
 - i. Unlike the CNN model, the YOLOv5 model predicts the list $[x_{center}, y_{center}, width, height]$ of the bounding box.
 - ii. The problem here is that the dataset contains $[x_{max}, y_{max}, x_{min}, y_{min}]$ of the bounding box, but we need $[x_{center}, y_{center}, width, height]$ of the bounding box.
 - iii. To handle this problem we use some basic mathematics and get the required values.
 - iv. One more thing to note here is that the values x_{center} , y_{center} , width, height should be in between 0 and 1. So to achieve this we performed normalization.
 - v. 321 images are kept in the training dataset and remaining in the testing dataset.

3. Models used:

- CNN Architecture:

- i. Some hyper-parameters used are;
 - 1. No.of epochs: 50
 - 2. Batch-size: 32
 - 3. Optimizer: Adam
- ii. Used Model Check points for saving the best model.

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 6, 6, 512)	14714688
flatten (Flatten)	(None, 18432)	0
dense (Dense)	(None, 128)	2359424
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 4)	260

```
=====  
Total params: 17,099,140  
Trainable params: 2,384,452  
Non-trainable params: 14,714,688  
=====
```

- YOLOv5 model:

- i. We create a data.yaml file which contains path to the train and test datasets and other information link no.of classes present as shown in the image below.

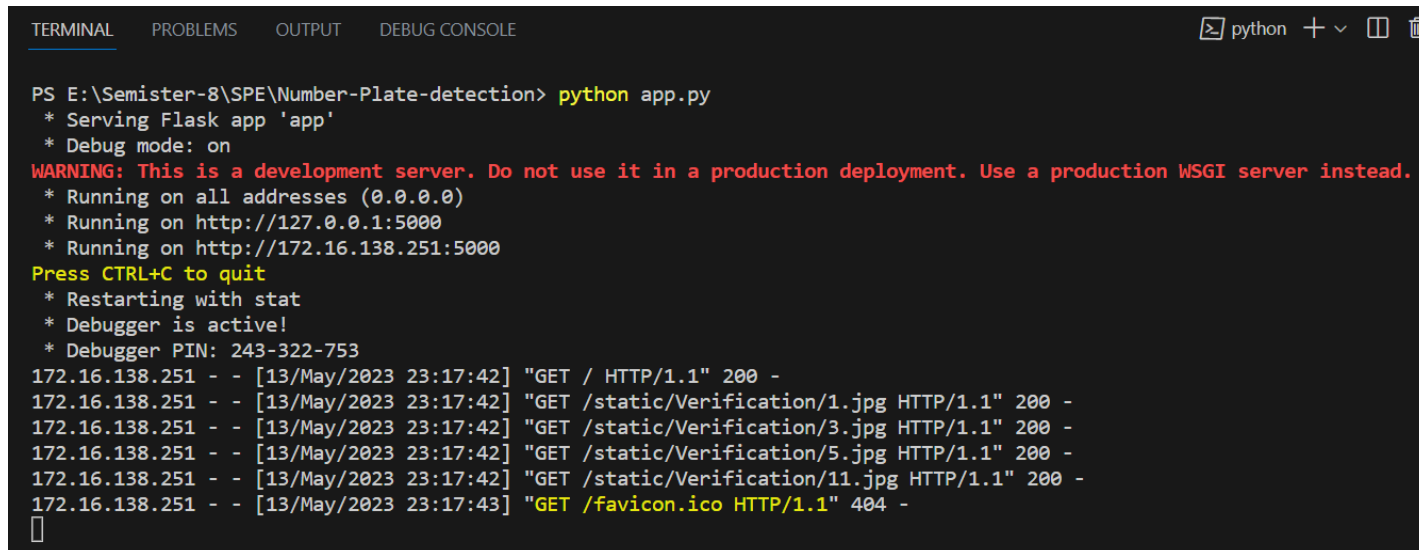
```
train: data_images/train  
val: data_images/test  
nc: 1  
names: [  
  "LP"  
]
```

- ii. Both the train and test folders contain 2 files for each image where one is a .png file which contains the image and the other is a .txt file which contains x_center, y_center, width and height of the bounding box.

- iii. Some hyper parameters we used are;
 - 1. Batch-size: 8
 - 2. No.of epochs: 100
 - iv. The time taken to train the model using GPU is 50 minutes.
4. Text parsing from the detected number plate:
- After the Number plate of the car is detected, we performed text parsing using the pytesseract library.
 - First we took the sub image which contained only the number plate.
 - Then we passed it to a function called “*image_to_string*” which extracts the text from the image.
 - The parameters of “*image_to_string*” function are;
 - i. lang: english
 - ii. tesseract_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ012345

6: Results and discussion:

To run it in the local server we should go to the project directory and we should use the command “**python app.py**”.



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  python + v [ ] [X]

PS E:\Semister-8\SPE\Number-Plate-detection> python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.16.138.251:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 243-322-753
172.16.138.251 - - [13/May/2023 23:17:42] "GET / HTTP/1.1" 200 -
172.16.138.251 - - [13/May/2023 23:17:42] "GET /static/Verification/1.jpg HTTP/1.1" 200 -
172.16.138.251 - - [13/May/2023 23:17:42] "GET /static/Verification/3.jpg HTTP/1.1" 200 -
172.16.138.251 - - [13/May/2023 23:17:42] "GET /static/Verification/5.jpg HTTP/1.1" 200 -
172.16.138.251 - - [13/May/2023 23:17:42] "GET /static/Verification/11.jpg HTTP/1.1" 200 -
172.16.138.251 - - [13/May/2023 23:17:43] "GET /favicon.ico HTTP/1.1" 404 -
[ ]
```

Fig 19: Running the code in our local machine.

We can see that in the above figure there is a line saying “**Running on <http://172.16.138.251:5000>**”. If we open that link our Car Plate Detection local page will be visible in our machine.

Using the app:

1. Upload the image of a car containing the number plate and click on the “Upload” button.
2. The page will be scrolled down and you can see the output
3. Alternatively one can also click on any of the four default images present to just try out the website. When you click on one of them, the page will be scrolled down and you can see the output.

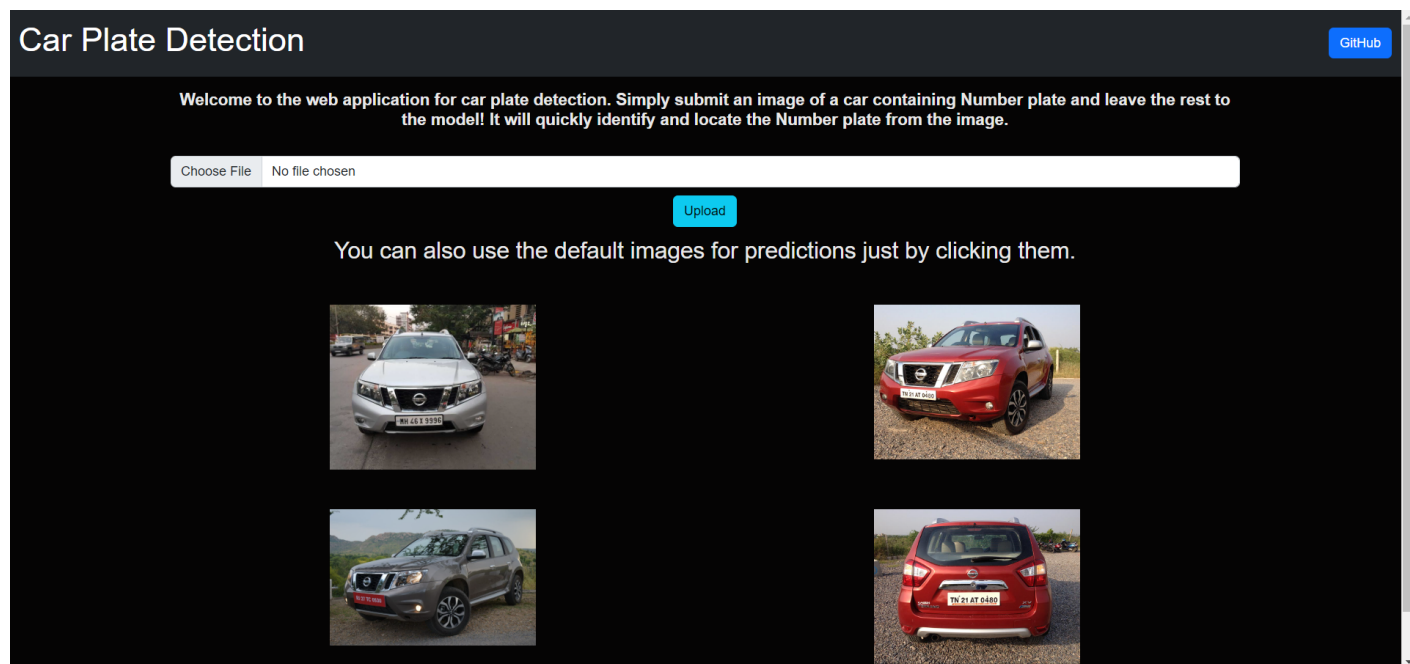


Fig 20: This is how it looks when we open the website.

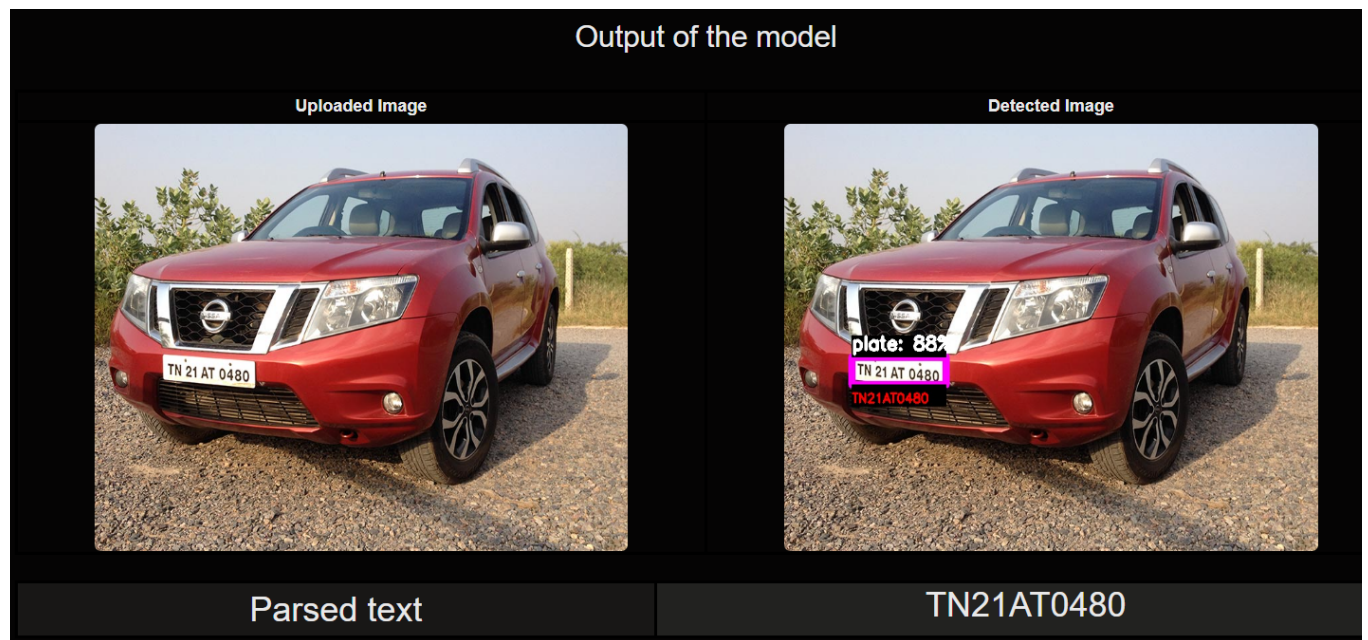


Fig 21: The image in the left is the input image and the image in the right is the output image which contains a bounding box around the number plate of the car. From the image one can also see that the number plate of the car is predicted with 88% confidence.

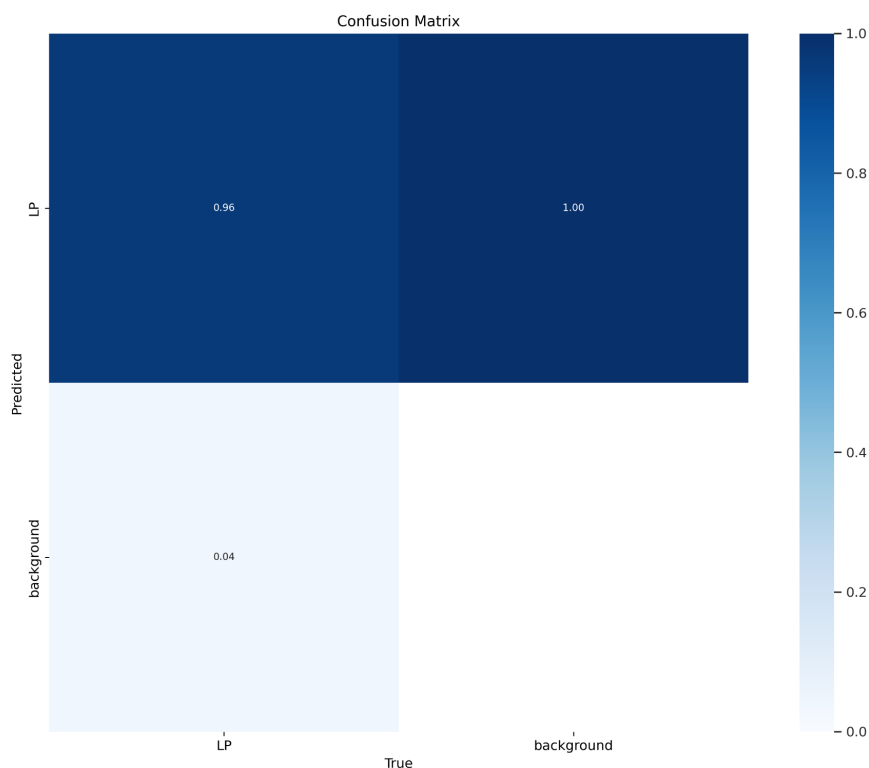


Fig 22: Confusion matrix. Here LP class means “License Plate” class

7 FUTURE SCOPE

Currently the scope of this project is to identify the number plates given car images and to parse the text written on the number plates. As future work we would like to extend this website to identify and parse the text on number plates for other vehicles as well like a bike, scooter or auto etc.

We would require data on each of the vehicles on which we would like the website to work on and train our model accordingly.

8 CONCLUSION & SUMMARY

The Devops methodology and life cycle tools prove to be better than the Agile methodology in terms of technical, cultural and business benefits. By minimizing friction between independent teams, DevOps enables a collaborative approach for enterprise software development and delivery that reflects the needs of the entire application life cycle for today's modern enterprises. Thus, we can develop, test, deploy and monitor the application easily.

Our website does 2 tasks which are number plate detection and text detection on the number plate. For the number plate detection we initially used a CNN model using the image-net weight but the bounding box over the number plate was not accurate. So, then we moved on to the YOLOv5 model and got better and accurate results to detect the number plate. For the text detection on the number plate we used Pytesseract, a Python library that provides a simple interface to the OCR (Optical Character Recognition).

9 Links:

- [GitHub Repository Link](#)
- [Docker Hub link](#)
- [Link to the dataset](#)