PROJECT REPORT

Vehicle Number Plate Detection

Created by Vikram Singh

GitHub Profile : VkasRajpurohit (github.com)

TABLE OF CONTENTS

ABSTRACT

Automatic vehicle license plate detection and recognition is a key technique in most traffic related applications and is an active research topic in the image processing domain. As a result, different methods, techniques and algorithms have been developed for license plate detection and recognition.

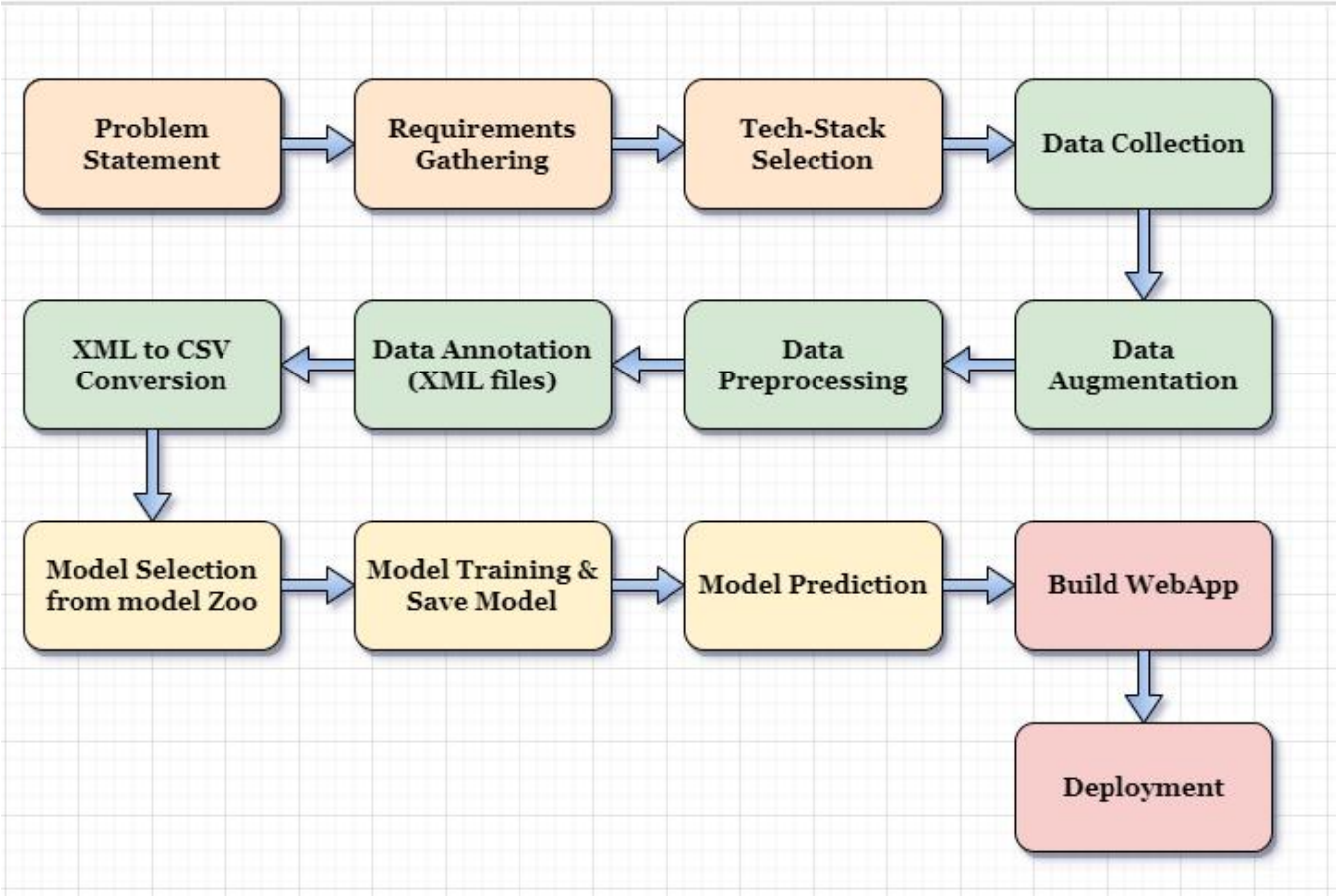
In the case study, It is shown that how to process image data, detect objects inside the image and create bounding box for the detected object. Inception-ResNet-v2 TensorFlow model used for this case study, OpenCV for this project in order to identify the license number plates and the python pytesseract for the characters and digits extraction from the plate.

GitHub link -- [VkasRajpurohit/Vehicle-Number-Plate-Detection (github.com)]

Overview:

Title	:	Vehicle Number Plate Detection
Domain	:	Traffic Surveillance and Security
Tools & Technology	:	Python | OpenCV | PyTesseract | Tensorflow | Keras | Image Data-Processing | Deep Learning | Flask-API | HTML | Bootstrap | GitHub | AWS
Tensorflow Model	:	Inception-ResNet-v2
IDE	:	PyCharm | Google Colab

Process Flow:



CHAPTER 1
INTRODUCTION

1. Introduction:

Now a Day's security has become one of the biggest concerns for any organization, and automation of such security is essential. However, many of the current solutions are still not robust in real-world situations, commonly depending on many constraints.

In this project, we will understand how to detect & recognize License number plates using the Python programming language. We will utilize **OpenCV** for this project in order to identify the license number plates and the python **pytesseract** for the characters and digits extraction from the plate. We will build Web App using Flask-API that automatically recognizes the License Number Plate.

- License plate detection and recognition is the technology that uses **computer vision** to detect and recognize a license plate from an input image of a vehicles.

1.1 Business Problem:

Automatic vehicle license plate detection and recognition is a key technique in most traffic related applications and is an active research topic in the image processing domain. As a result, different methods, techniques and algorithms have been developed for license plate detection and recognition.

Approach: Due to the varying characteristics of the license plate from country to countries, like numbering system, colors, the language of characters, style (font) and sizes of a license plate, further research is still needed in this area.

Results: In most countries, they use Arabic and English letters, plus their national logo. Thus, it makes the localization of plate number, the differentiation between Arabic and English letters and logo's object and finally, the recognition of those characters become more challenging research task. The use of the artificial neural network has proved itself beneficial for plate recognition, but it has not been applied for plate detection.

- Trying to build a solution that should recognize many places with uttermost clarity in any circumstances. With a varying distance and colour combination, it should work for any Indian continent.

1.2 Expected Solution:

- Build a solution that should be able to get an input as vehicle image and provide the result as extracted number from vehicle number plate.

1.3 Use cases of Vehicle Number Plate Detection:

Vehicle Number Plate Detection & Recognition is essential for a wide range of applications, where the detection, identification, or localization of vehicles is important.

1) **Law enforcement -** Police forces use this for law enforcement purposes, including to check if a vehicle is registered or to identify vehicles related to traffic violations. The ability to detect and recognize number plates in real-time allows authorities to identify vehicles and track their location.
2) **Car parking management** - Car parking management requires an integrated solution to detect individual vehicles. Hence, automated number plate recognition is the key to efficient car parking management. This allows parking garages to have automated parking management because every car is accounted for by its license plate number. Therefore, parking garage users avoid the stress of managing their own tickets and tracking time spent, risking penalties for inaccurate ticket payments or losing their tickets.
3) **Journey time analysis** - Journey time analysis (JTA) is a crucial application for authorities to identify passing through vehicles and their time from one node to another. In addition, such analytic allows better route planning for traffic administrators.
4) **Traffic management** - This can be used throughout cities to detect over-speeding vehicles, vehicles that drive rashly, or any accidental occurrence. This provides solutions for measuring and analyzing area-related traffic data of a certain area or an entire city. On a larger scale, traffic management allows insights into traffic congestion for better traffic planning.
5) **Retail park security -** Retail parks often deal with unauthorized parking, leading to hassles for rightful parking spaces or sometimes suspicious activities. Such security risks can be addressed with this technology by ensuring only the authorized vehicles are using parking spaces.
6) **Tollbooth Records** - Manual tollbooth management on highways is still a significant practice in some parts of the world. Often, tollbooths leverage different technologies for autonomous tollbooth management. Vehicle Number Plate Detection & Recognition enables efficient toll booth management, decreasing the operational time needed and thus increasing productivity.

CHAPTER 2
Business Problem & Business Constraints

2. 1 Formulation of the business problem:

First Cut Approach

- First of all, we need to **collect data** i.e. image data (vehicle images with number plate), also increase dataset size by **data augmentation** techniques to avoid overfitting problem.
- After data collection, preprocess the dataset, need to **resize** all the images in the same size.
- Now need to do **data annotation**, for labeling we will be using tool i.e. **LabelImg**. Need to labeling manually for each image data.
- After data annotation, we will save this **xml files** & then need to **convert xml to csv**.
- Now for **model training**, we will be using Tensorflow-Keras **Inception-Resnet-V2 Model**.
- Inception-ResNet-v2 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 164 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The Inception-ResNet-v2 was used for the classification task. The architecture of the network is shown in Figure 9 . Inception-Resnet-v2 is formulated based on a combination of the Inception structure and the Residual connection. In the Inception-Resnet block multiple sized convolutional filters are combined by residual connections. The usage of refusal connections not only avoids the degradation problem caused by deep structures but also reduces the training time.
- After model training, **save the model** for later use.
- Now **build flask web application** & **create pipeline**.
- For **number extraction** from number plate, we will be using **TESSERACT OCR**. Tesseract OCR have a python API and it is open source. For this task, we will load our image and convert to array. Crop our bounding box with coordinates of it. We will identify region of interest (ROI).
- Finally, we will deploy the model in productionisation.

2.2 Business Constraints:

1) **Time:** Latency is really not a major issue, even a few seconds of the latency is considerable.
2) **Accuracy:** Accuracy is a vital constraint too get the accurate number plate.
3) **Interpretability:** Model should be easy to interpret and user friendly.

CHAPTER 3
Data Preparation & Model Building

3. Data Preparation & Model Building:

For Data preparation & Model building, we will divide big problem into small-small steps or modules and then we will combine all these steps/modules and create pipeline.

3.1 Data Collection:

We have collected data from different-different sources i.e. collected by web scraping (didn't got data as expected), then collected data manually by recording & clicking pictures on traffic signal, also a part of dataset collected via open-source data.

3.2 Data Augmentation:

Collected data size wasn't enough for model preparation, hence applied some of the data augmentation techniques to increase the size of the dataset to avoid over-fitting.

- On dataset, applied augmentation techniques like Rotating, Random Brightness, Random color, Random Brightness, Random color, Random Contrast, Random Distortion, Random Erasing, Zooming.

3.3 Data Resize:

We have to give same size of the image data to the model.

- Resize dataset into 640x640.
- To resize all the dataset, written a python script/function shown below -

```python
# Import required libraries
import os, cv2

def resizer(path, width, height):
  '''It will return resized images
     Written by : Vikram Singh
     Date : Jun 5th, 2022'''

  dir_path = os.path.join(path, 'dataset/Augmented_Data')
  dir_path = os.chdir(dir_path)
  dim = (width, height)  # Image result size

  try:
    for filename in os.listdir(dir_path):
      if filename.endswith(".jpg") or filename.endswith(".jpeg") or filename.endswith(".png"):
        image = cv2.imread(filename)
        image_resized = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)
        cv2.imwrite(filename, image_resized)
    print('Resize operation completed successfully.')

  except Exception as e:
    print('Unable to complete process, getting error: ', e)
```

3.4 Data Annotation:

For Data Annotation, we used tool LabelImg.

- We have done simple annotation i.e. used only one class (numberplate). For the same, we will have to extract number using OCR.
- Annotated the number plate of each & every image manually.
- Annotated data output is the xml files, example shown below -

```
<annotation>
        <folder>train</folder>
        <filename>dataset_original_img026.jpg_fb3f0d41-4e85-43f3-a66c-b6bd515c4d98.jpg</filename>
        <path>C:\Users\vkasr\Desktop\INeuron\Case_Study\Vehicle Number Plate Detection\imgs\train\dataset_
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>640</width>
                <height>640</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>numberplate</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>229</xmin>
                        <ymin>215</ymin>
                        <xmax>304</xmax>
                        <ymax>632</ymax>
                </bndbox>
        </object>
```

- After completing labeling process, need to convert xml to csv since for the training process we need data in array format. For that, we will take the useful information from the label which are the diagonal points of the rectangle box or bounding box which are xmin, ymin, xmax, ymax respectively.

3.5 XML to CSV:

We will convert bounding information into CSV and later on, we will convert that into an array using Pandas.

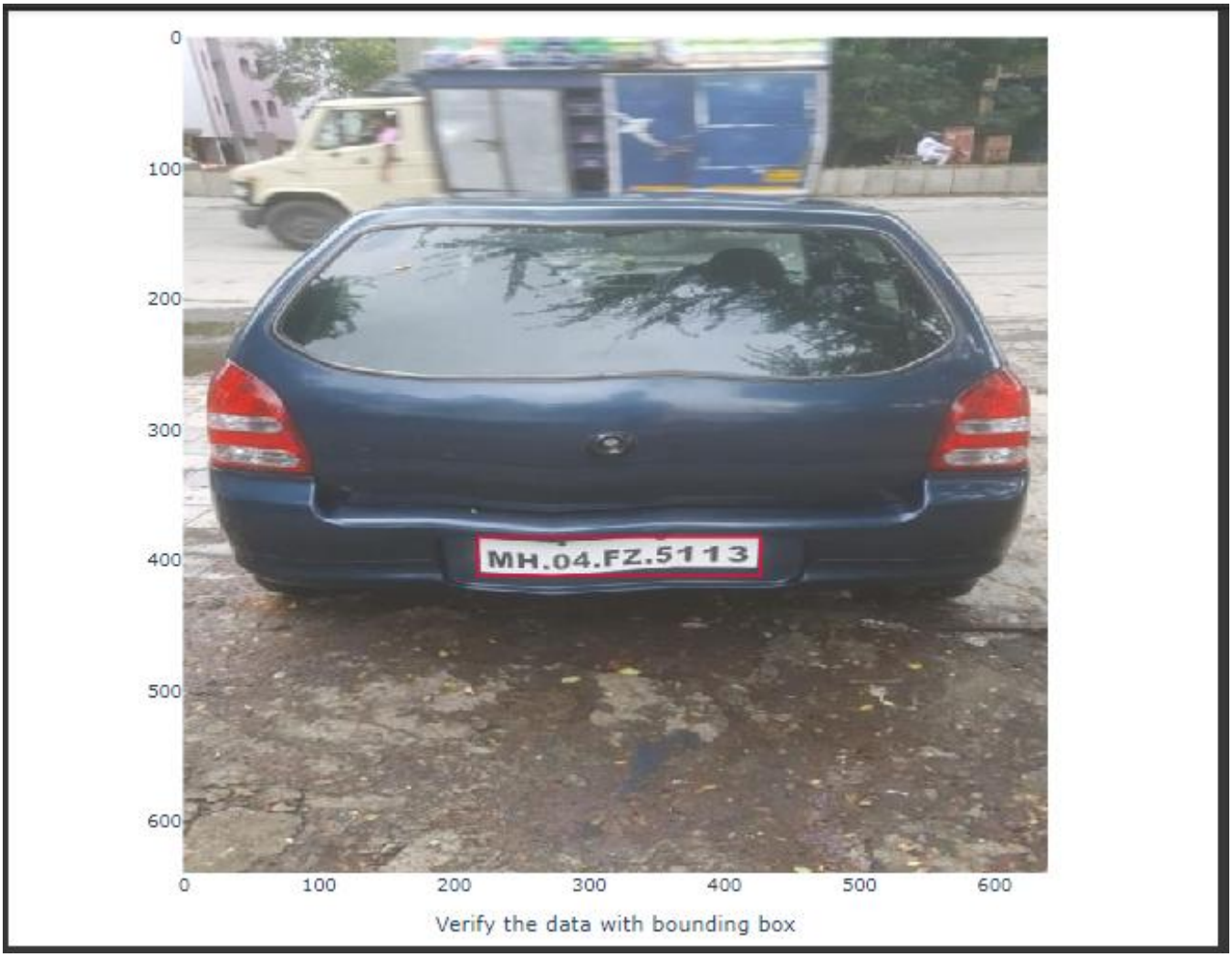
- After converting xml to csv file -

	filepath	xmin	xmax	ymin	ymax
0	imgs/train/dataset_original_img001.xml	181	517	394	447
1	imgs/train/dataset_original_img002.xml	401	456	140	469
2	imgs/train/dataset_original_img003.xml	218	427	382	412
3	imgs/train/dataset_original_img004.xml	202	440	388	423
4	imgs/train/dataset_original_img005.xml	149	565	404	472

- Verify if bounding box appearing properly or not -

```
file_path = image_path[2] # path of our image
img = cv2.imread(file_path) # read the image

img = io.imread(file_path) # Read the image
fig = px.imshow(img)
fig.update_layout(width=800, height=640, margin=dict(l=10, r=10, b=10, t=10),
                  xaxis_title='Verify the data with bounding box')
fig.add_shape(type='rect', x0=218, x1=427, y0=382, y1=412, xref='x', yref='y', line_color='crimson')
```

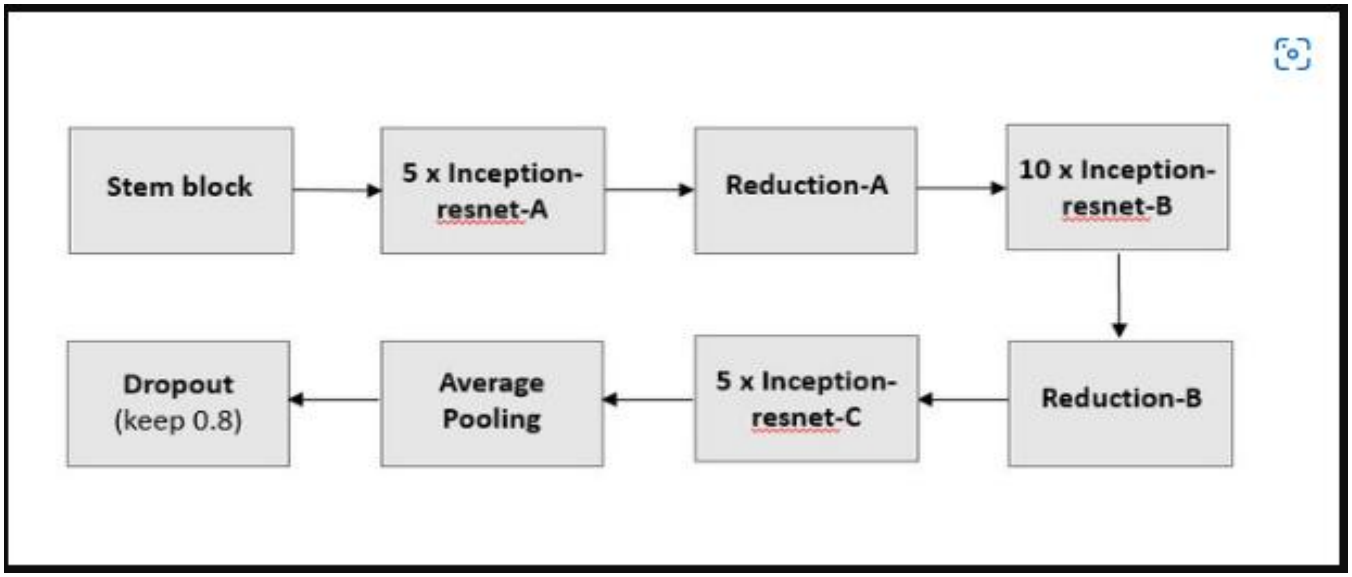


- Seems like good result as expected.

3.6 Model Selection:

- We will use the **Inception-ResNet-v2 model** with pre-trained weights and train this to our data.

Inception-ResNet-v2 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 164 layers deep and can classify images into 1000 object categories, such as the keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 299-by-299, and the output is a list of estimated class probabilities. It is formulated based on a combination of the Inception structure and the Residual connection. In the Inception-Resnet block, multiple sized convolutional filters are combined with residual connections. The usage of residual connections not only avoids the degradation problem caused by deep structures but also reduces the training time. The figure shows the basic network architecture of Inception-Resnet-v2.



Basic Architecture of InceptionResNetV2

3.7 Model Training & Save:

After model compile:

==========

Total params: 65,918,040
Trainable params: 65,857,496
Non-trainable params: 60,544

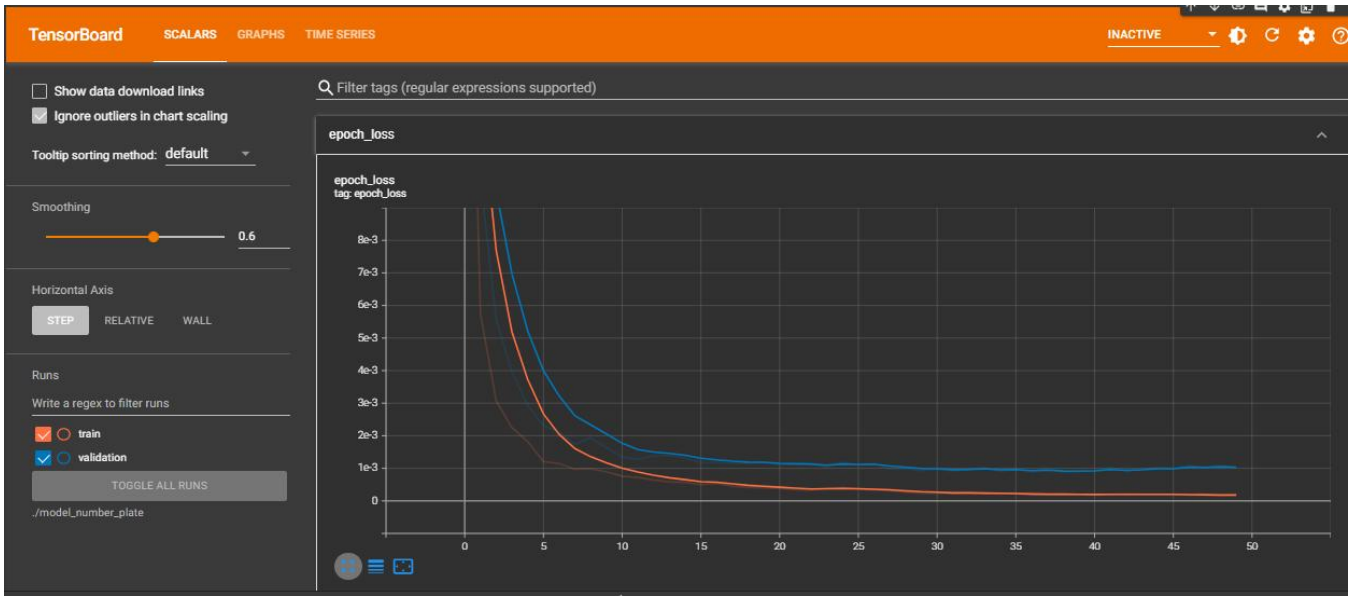
Training Model:

```
tfb = TensorBoard('model_number_plate')
history = model.fit(x=X_train, y=y_train, batch_size=32, epochs=50, validation_data=(X_test, y_test), callbacks=[tfb])
```

- After training model saved as **model_number_plate.h5**.

3.8 TensorBoard:

- Loss decreasing as the number of epochs increases.



- Model training has been completed.

3.9 Model Prediction:

For prediction, we will take an input image and perform all the steps done in training, need to create pipeline for the same.

- First, we will load our saved model i.e. model_number_plate.h5.
- Now, we will define a function i.e. **number_plate_detection(path)**, which will perform entire pipeline.
  - It will read the image from given path, convert into 8bit array, resize to 224x224, then data preprocessing.
  - **Data Preprocessing:**
    - **De-normalize the output image -**
      The output of the model is the normalized output. So, need to convert back into our original form values, in the training process, we have the original form values and convert that normalized one. So basically, we will de-normalize the values back.
  - **Bounding Box:** Now, we will draw bounding box on top of the image, just providing the two diagonal points & with these points, draw the rectangle box.

- **Visualization:** In the end, visualizing image with coordinates of bounding box, result shown below-

- It is capturing bounding box with high accuracy. Now, need to **extract numbers of number plate from bounding box**. For the same, we will use **pytesseract OCR**.

- **OCR:** Python-tesseract is an optical character recognition (OCR) tool for python i.e. it will recognize and "read" the text embedded in images.We will load the image and convert to array, crop our bounding box with coordinates of it, identify region of interest (ROI), cropped image shown below -

- **Extract Text From Image:**

```
# extract text from image
text = pt.image_to_string(roi)
print(text)
```

```
HR.26.BR 90445
```

- Seems like, model is working fine.
- We can improve the accuracy by training model with more clear images/HD images and by increasing dataset size.

Till now, model preparation has been completed, Our model is working fine. Now we need to build a web application for productionisation. Our model is ready, now front-end team will take care further for productionisation process.

CHAPTER 4
Conclusion

4. Conclusion:

With the increase in the number of vehicles, vehicle tracking has become an important research area for efficient traffic control, surveillance, and finding stolen cars. For this purpose, efficient license plate detection and recognition are of great importance. Due to the variation in the background and font color, font style, size of the license plate, and non-standard characters, license plate recognition is a great challenge in developing countries.

To overcome such issues, this case study applies a deep-learning strategy to improve license plate detection & recognition efficiency. The collected images have been captured under various lighting/contrast conditions, distance from the camera, varying angle of rotation, and validated to produce a high recognition rate. The approach can be effectively used by law enforcement agencies and private organizations to improve homeland security. Future work may include training and validation of the existing algorithm using the hybrid classifier method and improvement of the robustness of the license plate recognition system in varying weather conditions.

References:

Definitions: Wikipedia
Data Annotation tool : LabelImg (tzutalin.github.io)
Model Selection : Inception ResNet v2 | Papers With Code
OCR : PyTesseract · PyPI

Created by Vikram Singh
GitHub Profile : VkasRajpurohit (github.com)
Code : VkasRajpurohit/Vehicle-Number-Plate-Detection (github.com)

---------- **End of Project Report** ----------

Thank You !

...