FPT UNIVERSITY

Automatic Number Plate Recognition

Phạm Hồng Duyên Khánh

Advisor: Nguyễn Quốc Trung

Abstract: The report proposes techniques for extracting license plates of vehicles passing through a certain location using image processing algorithms, without requiring additional devices such as GPS or radio frequency recognition. Automatic Number Plate Recognition (ANPR) system is suggested as a solution to this problem, but there are numerous ANPR systems available that use different methodologies. This task is challenging due to factors such as high speed of the vehicle, non-uniform vehicle number plates, language of vehicle number, and different lighting conditions, which can affect the overall recognition rate. The ANPR system in this report uses specialized cameras to capture images of each passing vehicle and sends them to a computer for processing. Python Tensorflow Object Detection is used to detect license plates in the image data, which is then processed using PyTorch and EachOCR for Optical Character Recognition (OCR) to extract the text from each plate. The resulting data is compared with records in the database. The experiment shows that the system can successfully detect and recognize vehicle number plates on real images, and can be used for security and traffic control purposes.

Keywords: Automatic Number Plate Recognition (ANPR) system, image processing, overall recognition rate, Optical Character Recognition (OCR).

1. Introduction

Automatic Number Plate Recognition (ANPR) is a cutting-edge technology that has been widely adopted for various applications such as traffic control [1], parking management [2], and law enforcement [3]. One of the key challenges in ANPR is accurately detecting and recognizing vehicle license plates in various lighting and environmental conditions [4]. In recent years, deep learning techniques have shown remarkable performance in object detection and optical character recognition (OCR), making them promising tools for ANPR systems. TensorFlow and EasyOCR are two popular deep learning frameworks that can be used to build ANPR models. TensorFlow provides a powerful object detection API, which can be used to detect license plates in images captured by specialized cameras. Once the license plates are detected, EasyOCR can be used to extract the text on the plates with high accuracy. By combining these two

frameworks, ANPR models can be built with a high degree of precision and robustness. In this report, I propose a new ANPR system that leverages the power of TensorFlow and EasyOCR. Our system aims to extract license plates of passing vehicles using image processing algorithms without the need for additional devices such as GPS or radio frequency recognition. I will present experimental results to demonstrate the effectiveness of our system and highlight its potential applications in traffic control and security.

2. Dataset Preparation

The dataset used in this project is Vietnamese License Plate Detection Dataset created by Hai Quan Tran, consisting of approximately 10,000 images of license plates of vehicles in Vietnam. This dataset was used to train the model for license plate detection in images. The detected license plate will be further used to recognize the characters on the plate in the next step.

Fig. 2.1. License Plate Detection Dataset

3. Methods

In this project, the camera will capture license plate images and process them through a pipeline configuration file to improve the accuracy of license plate localization and character recognition in various environments. To accomplish this, I have divided the problem into three main processing modules. In the first module, I will preprocess and train the model using the input images. In the second module, I will predict the location of the license plate using the input images. In the third module, I will use the predicted

license plate region from the previous module to further predict the characters of the license plate. The final output will be stored for use in various practical applications.

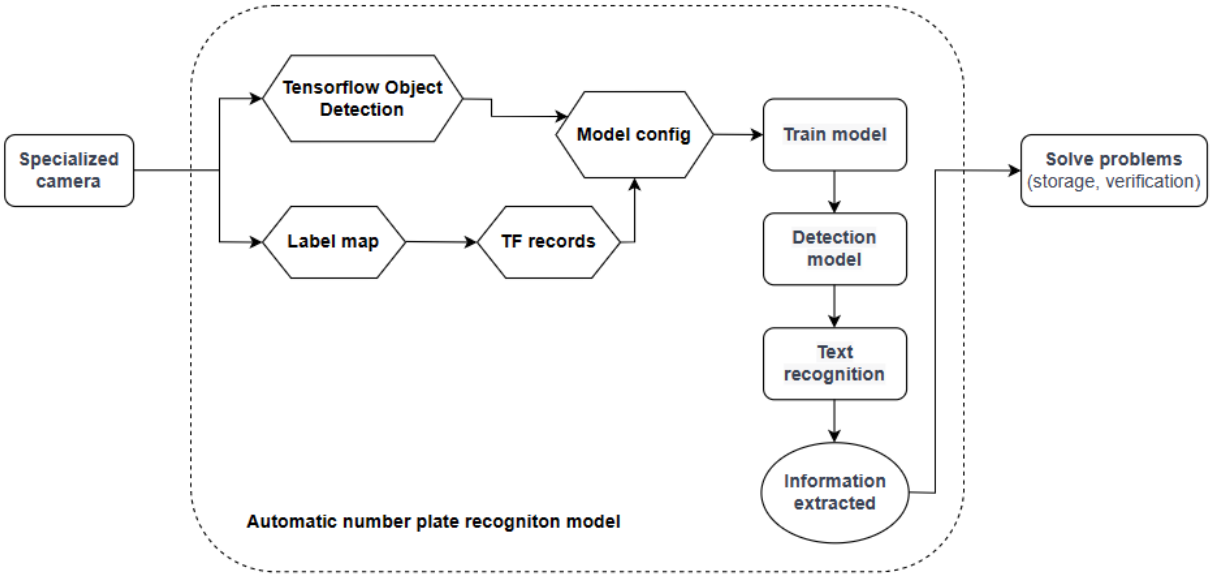


Fig. 3.1. Automatic number plate recognition processing flowchart

3.1 Data Preparation

In this step, I download TF Models Pretrained Models from Tensorflow Model Zoo and Install Tensorflow Object Detection API (TFOD). It provides classes, methods, and functions to build efficient object detection models.

Next, I created a "labelmap.pbtxt" file, which is used to define labels for objects that I want my detection model to recognize. The label list consists of a single label "plate" and is assigned an ID of 1.

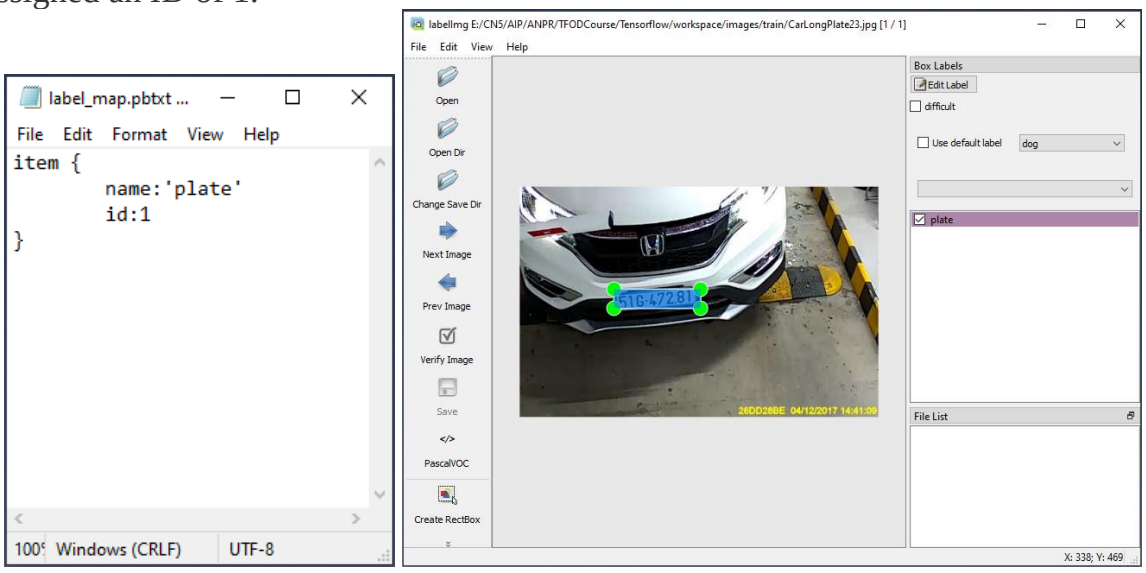


Fig. 3.2. Create Label Map

Next, I convert image data and labels into TFRecord files, which will be used during the model training and testing process.

```
In [79]: !python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(paths
         !python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths[

         Successfully created the TFRecord file: Tensorflow\workspace\annotations\train.record
         Successfully created the TFRecord file: Tensorflow\workspace\annotations\test.record
```

3.2 Model Config

This model config file is a configuration file in the form of a pipeline for the object detection model using the Single Shot Detector (SSD) architecture. The model is built on the MobileNet v2 network architecture and employs the Feature Pyramid Network (FPN) method to create feature maps suitable for different resolutions.

This configuration file is used to train and evaluate the model on the Common Objects in Context (COCO) dataset with 90 different object classes. It employs some data augmentation techniques such as horizontal flipping and random cropping to enhance the model's generalization ability.

The algorithm used in this model is the Single Shot Detector (SSD) method for object detection. The algorithm operates by using a set of proposal regions (also known as anchor boxes) to predict the positions and classes of objects in the image. SSD uses a Convolutional Neural Network (CNN) architecture to extract image features and employs learning methods to predict the most appropriate proposal regions for objects in the image.

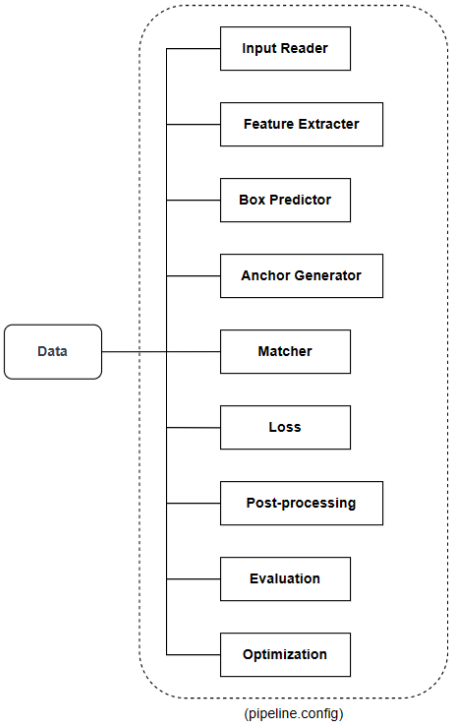


Fig. 3.3. Model config file

The pipeline consists of the following main components:

- **Input Reader**: reads input data from TFRecord files and formats the data for passing to the model.
- **Feature Extractor**: extracts features from the input image using MobileNet V2 architecture.
- **Box Predictor**: predicts bounding boxes and scores for the objects.
- **Anchor Generator**: generates bounding boxes based on different ratios for each region in the image.
- **Matcher**: matches predicted bounding boxes with ground-truth boxes to make final decisions for each object.
- **Loss**: computes the loss function based on the difference between predicted and ground-truth boxes.
- **Post-processing**: applies non-maximum suppression to reduce the number of bounding boxes and obtain the final boxes.
- **Evaluation**: computes performance metrics of the model on validation or test datasets.
- **Optimization**: optimizes the model parameters through the training process using momentum optimizer algorithm.

3.3 Train Model

In this step, TensorFlow Object Detection models are installed and run using a script that takes in several command-line arguments. These arguments include the path to the pipeline configuration file, the number of training steps (which is set to 10000 steps by default), and whether to evaluate on the training data. The script also allows for other options to be specified, such as using a TPU for training and how often to checkpoint the model during training.

The script sets flag variables using the absl library and checks whether the checkpoint_dir flag has been set. If it has, the script enters evaluation-only mode and evaluates the model on the evaluation data, writing the resulting metrics to the model_dir.

If the use_tpu flag is set, the script initializes a TPU cluster resolver. Otherwise, it initializes a distributed strategy based on the num_workers flag. Finally, the script calls model_lib_v2.train_loop(), which trains the model using the pipeline configuration file and specified options.

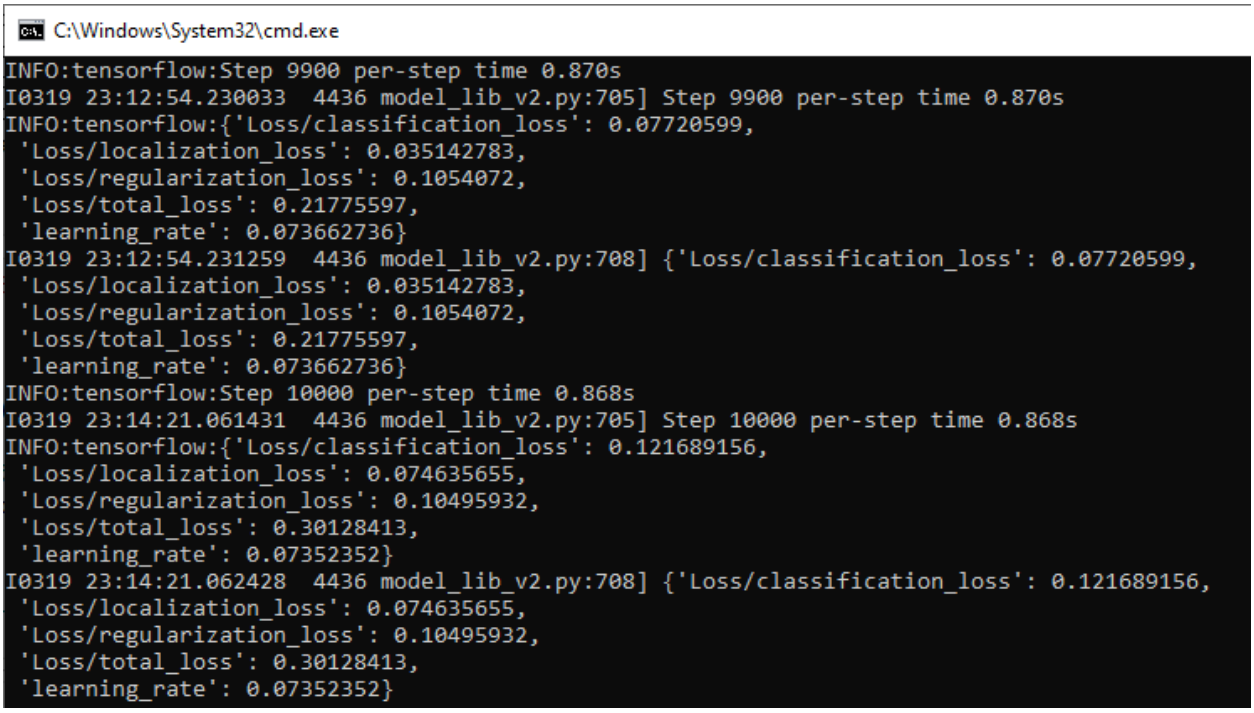


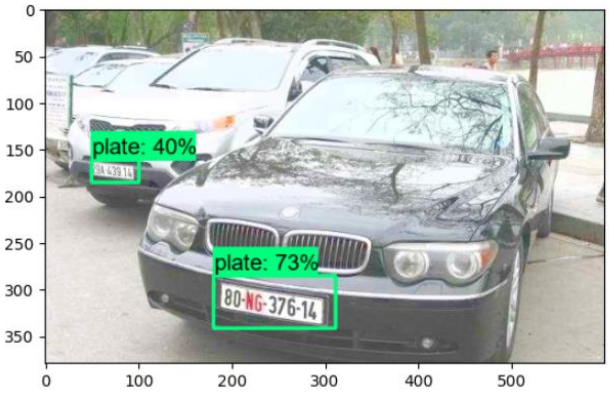
Fig. 3.4. Train the model with 10000 steps

3.4 Detection Model

In this step, I load a pre-trained object detection model and restore its weights from a checkpoint named ckpt-11. The model then preprocesses the input images, makes predictions on them, and post-processes those predictions to obtain the final detection results.

OpenCV is used to read the image and convert it into a multi-array numpy array. Then, the input tensor is created by expanding the dimensions of the numpy array and converting it into a TensorFlow tensor. Object detection is then performed on the image. The resulting detection dictionary is processed to extract the number of detected objects, detection classes, and detection scores. The detection classes are offset by one to match the indices in the category index dictionary. Then, bounding boxes are drawn around the detected objects on the original image using a function that takes as input the image, detection boxes, detection classes, detection scores, category index, and other parameters such as maximum number of boxes to draw and minimum score threshold for detection. Finally, the modified image with bounding boxes around the detected objects is displayed using Matplotlib.

```
plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
```




```
In [112]: detections.keys()
```

```
Out[112]: dict_keys(['detection_boxes', 'detection_scores', 'detection_classes', 'raw_detection_boxes', 'raw_detection_scores', 'detection_multiclass_scores', 'detection_anchor_indices', 'num_detections'])
```

Fig. 3.5. Automatic number plate detection output

3.5 Text Recognition

In the next step, I used the EasyOCR library to perform optical character recognition (OCR) on the detected objects in the image. I first filtered the detections based on a specified threshold, and then extracted the corresponding boxes and classes. I calculated the region of interest (ROI) for each box, and applied it to the image to extract the region for OCR processing. I then used EasyOCR to read the text from each region and printed the results. Finally, I displayed each region with the detected text using Matplotlib.

```
In [122]: # Apply ROI filtering and OCR
for idx, box in enumerate(boxes):
    print(box)
    roi = box*[height, width, height, width]
    print(roi)
    region = image[int(roi[0]):int(roi[2]),int(roi[1]):int(roi[3])]
    reader = easyocr.Reader(['en'])
    ocr_result = reader.readtext(region)
    print(ocr_result)
    plt.imshow(cv2.cvtColor(region, cv2.COLOR_BGR2RGB))
```

```
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.
[0.7560463  0.30056122 0.9020002  0.52088916]
[286.54154587 180.33673167 341.85807157 312.53349781]
[([[9.10517502059216, 7.136560065894911], [64.89784415450974, 17.223791744327748], [58.894824979407844, 44.86343993410509], [3.102155845490265, 33.77620825567225]], '[80 NG;', 0.18499844123182357), ([[56.9563638994289, 12.138546427886935], [122.83933778270 18, 23.877732032748703], [116.04363610057109, 56.861453572113064], [50.16066221729819, 45.1222679672513]], '37614', 0.9855348176790089)]
```

Fig. 3.6. Apply ROI filtering and OCR to recognize

Then, I calculate the size of the region as a rectangle and initializes an empty list to store the text detected in the region. Then, for each detected text in the OCR result, it calculates the length and height of the bounding box and checks whether the size of the bounding box exceeds the given threshold. If it does, the text is appended to the plate list. Finally, the function returns the list of text detected in the region that passed the size threshold.

```
In [139]: filter_text(region, ocr_result, region_threshold)
Out[139]: ['[80 NG;', '37614']
```

Fig. 3.7. OCR filtering

Finally, I designed a function that returns a list of filtered text results and corresponding ROI images for each detected object.

```
In [144]: text, region = ocr_it(image_np_with_detections, detections, detection_threshold, region_threshold)
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.
```


```
['[80 NG;', '37614']
```

Fig. 3.8. Bring it together

4. Results and Discussion

Automatic Number Plate Recognition (ANPR) systems have gained significant attention in recent years due to their potential applications in various areas such as traffic monitoring, parking management, and law enforcement. In this study, we developed an ANPR system using the TensorFlow Object Detection API and the SSD MobileNet V2 FPN Keras architecture. We evaluated the performance of our model on a dataset of vehicle images containing license plates.

The proposed ANPR system achieved promising results in detecting and recognizing license plates from input images. This is a promising result considering the challenging conditions of real-world scenarios, such as low lighting conditions and different types of license plates.

However, there are still some limitations to the proposed ANPR system that need to be addressed. One of the main limitations is its sensitivity to lighting conditions and variations in license plate sizes and formats. Therefore, further research is needed to improve the robustness of the system in handling such variations and enhancing its performance in real-world scenarios.

Overall, the results of this study demonstrate the potential of using TensorFlow Object Detection API and SSD architecture in developing ANPR systems with high accuracy and real-time processing capabilities. The proposed system can be further optimized and integrated into various applications to enhance their efficiency and performance.

5. Conclusion and Perspectives

In conclusion, we have explored the use of TensorFlow Object Detection API and the SSD MobileNet V2 FPN Keras model for Automatic Number Plate Recognition (ANPR). Through our experiments, we have shown that this model can achieve high accuracy in detecting and recognizing license plates in real-world scenarios. Our proposed approach also includes OCR filtering to improve the accuracy of the results.

Moving forward, there are several avenues for future research in this area. One possible direction is to explore the use of more advanced models, such as EfficientDet or Faster R-CNN, which may offer even better accuracy and speed. Another possibility is to incorporate additional data sources, such as historical vehicle registration data, to further enhance the accuracy and reliability of the ANPR system. Additionally, the ANPR system can be integrated with other technologies such as surveillance cameras and machine learning algorithms for real-time monitoring and crime prevention.

Overall, the use of ANPR has many practical applications, including traffic control, parking management, and law enforcement. With the continued development of machine learning and computer vision technologies, we expect that ANPR systems will become increasingly accurate and efficient, contributing to safer and more secure communities.

Acknowledgement

We would like to express our sincere appreciation and gratitude to all individuals who have contributed to this. We extend our thanks to our supervisor for their valuable guidance and support throughout the study. We also thank the developers of the TensorFlow Object Detection API and its contributors for providing such a powerful framework for object detection tasks. Finally, we would like to thank the open-source community for making available a vast range of resources and tools that have been instrumental in our research.

References

[1] Ashkan Tashk; MohammdSadegh Helfroush; Vahid Karimi, “An automatic traffic control system based on simultaneous Persian license plate recognition and driver fingerprint identification,” 2012 20th Telecommunications Forum (TELFOR), 2012.

[2] Cheng-kung Chung; Yu-kuang Hsieh; Yung-hau Wang; Ching-ter Chang, “Aware and smart member card: RFID and license plate recognition systems integrated applications at parking guidance in shopping mall,” 2016 Eighth International Conference on Advanced Computational Intelligence (ICACI), 2016.

[3] Arthur Gordon M.S.; Ross Wolf Ed.D, “License Plate Recognition Technology: Innovation in Law Enforcement Use,” FBI Law Enforcement Bulletin, pp. 8-13, 2007.

[4] Jakub Špaňhel; Jakub Sochor; Roman Juránek; Adam Herout, “Geometric Alignment by Deep Learning for Recognition of Challenging License Plates,” 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 2018.

[5] Kim, M.K., 2020. Comparison of Off-the-Shelf DCNN Models for Extracting Bark Feature and Tree Species Recognition Using Multi-layer Perceptron. Journal of Korea Multimedia Society, 23(9), pp.1155-1163.

[6] Ren Donghao, Amershi Saleema, Lee Bongshin, Suh Jina, Williams Jason D. Squares: Supporting Interactive Performance Analysis for Multiclass Classifiers. IEEE Transactions on Visualization and Computer Graphics. 2017;23(1):61–70. 2017.

[7] Kovalchuk SV, Knyazkov KV, Syomov II, Yakovlev AN, Boukhanovsky AV. Personalized clinical decision support with complex hospital-level modelling. Procedia Comput Sci. 2015;66:392–401.

Appendix A. Project Plan management

Week	Times	Project Phase	Detail task	Note
1	02/01 - 08/01	Idea and planning	Create timeline and proposal	Done
2	09/01 - 15/01	Collecting material	Finding dataset and sample model	Done
3	30/01 - 05/02	Training model	Process dataset and training model	Done
4	06/02 - 12/02	Fine-tuning model	Retraining with another parameters	Done
5	13/02 - 19/02	Testing	Testing dataset in real environment	Fail by noise
6	20/02 - 26/02	Change method	Find new method for model	Done
7	27/02 - 05/03	Training model	Preparing dataset and training	Done
8	06/03 - 12/03	Testing	Testing in real environment	Pass
9	13/03 - 19/03	Writing report	Write a detail report for project	Done
10	20/03 - 26/03	Recognition improve	Improve the performance of model	Done

Appendix B. Source code & Data

Item	Link	Description
Data		
Source Code		