ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH

(Formerly College of Engineering and Technology) Odisha

Techno Campus, Ghatikia, Mahalaxmi Vihar, Bhubaneswar-751029

MINOR PROJECT REPORT ON

"ANPR System"

Submitted by:

Nishikanta Parida

Regd. No.: **2124100015**

Under the Guidance of

Dr. Debasis Gountia

(Associate Professor in CSA Department)

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATION

Year 2021 - 2023

. . . .

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH BHUBANESWAR

(Techno Campus, PO-Ghatikia, Mahalaxmi Vihar, Bhubaneswar, 751003)

CERTIFICATE

This is to certify that the work embodied in the project work entitled "**ANPR System**" being submitted by **Mr. NISHIKANTA PARIDA** in partial fulfilment for the award of the Degree of Master of Computer Science and Application to the Odisha University of Technology & Research is a record of bonafide work carried out by him under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Signature of Internal Guide

Name: **Dr. Debasis Gountia**
Designation: Associate Professor

Signature of Head of the department

Name: **Dr. Jibitesh Mishra**
Designation: Associate Professor

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH BHUBANESWAR

(Techno Campus, PO-Ghatikia, Mahalaxmi Vihar, Bhubaneswar, 751029)

DECLARATION

I **Nishikanta Parida** bearing Regd. no.: **2124100015,** a bonafide student of Odisha University of Technology & Research, would like to declare that the Project work entitled "**ANPR System**", is a record of an original work done by me under the esteemed guidance of **Dr. Debasis Gountia**, Associate Professor in Computer Science & Application Department. This project work is submitted in the partial fulfilment of the requirements for Master's degree & not submitted for the award of my degree.

Nishikanta Parida

Regd. No.: 2124100015

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my advisor **Dr. Debasis Gountia**, Associate Professor, Department of Computer Science and Application, whose knowledge and guidance has motivated me to achieve goals I never thought possible. She has consistently been a source of motivation, encouragement, and inspiration.

I would also like to convey my deep regards to all other faculty members of Department of Computer Science and Application, who have bestowed their great effort and guidance at appropriate times without which it would have been very difficult on my part to finish this work. Finally, I would also like to thank my friends for their suggestions and cooperation.

What I write or mention in this sheet will hardly be adequate in return for the amount of help and cooperation I have received from all the people who have contributed to make this project a reality. I am grateful to all for their constant support.

Nishikanta Parida

Date: ***Regd. No.: 2124100015***

ABSTRACT

Automatic number-plate recognition (ANPR) is a technology that uses object detection and optical character recognition on images to read vehicle registration plates. The objective of this project is to build an efficient automatic number plate recognition system which can extract the number plates of vehicles in real time from a live feed. The possible use cases of this ANPR system can be in Law enforcement, Car parking management, Journey time analysis, Traffic management, etc.

In this ANPR system, we used an object detection model to identify the license plate from the image/frame and then applied Optical character recognition(OCR) to get the license plate number from it. We store the recognized license plate numbers as well as the detected license plate section by the Object detection model for further training and vehicle identification purposes.

Transfer learning, a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. We used the Tensorflow object detection model for transfer learning for license plate detection. And another model(RPnet) using deep convolutional neural network to perform detection and recognition in one model.

INDEX TERMS:

ANPR, Object detection, OCR, transfer learning, Faster R-CNN,SSD networks. scan

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

INTRODUCTION:

The latest advancements in highway research domain and increase in the number of vehicles everyday led to wider exposure and attention towards the development of efficient Intelligent Transportation System (ITS). One of the popular research areas i.e., Automatic Number Plate Recognition (ANPR) aims at determining the characters that exist in the license plate of the vehicles. The ANPR process is a difficult one due to the differences in viewpoint, shapes, colors, patterns, and non-uniform illumination at the time of capturing images.

The objective of this project is to build an efficient automatic number plate recognition system which can extract the number plates of vehicles in real time.

The current project develops a robust Deep Learning (DL)-based ANPR model using Convolutional Neural Network (CNN), called the RPnet model and another model using Transfer- learning to make a light weight ANPR model. The presented technique has a total of four major processes namely pre-processing, License Plate (LP) localization and detection, character segmentation, and recognition.

In this ANPR system, we used an object detection model to identify the license plate from the image/frame and then applied Optical character recognition (OCR) to get the license plate number from it. We store the recognized license plate numbers as well as the detected license plate section by the Object detection model for further training and vehicle identification purposes.

The possible use cases of this ANPR system can be in Law enforcement, Car parking management, Journey time analysis, Traffic management, etc.

CHAPTER 2

SOFTWARE & HARDWARE REQUIREMENTS

SOFTWARE & HARDWARE REQUIREMENTS:

2.1. SOFTWARE REQUIREMENTS

2.1.1. Anaconda interpreter (to run deployed project i.e., Python file)

2.1.2. Jupyter Notebook (For Model building, training and testing)

2.1.3. Visual Studio Code

2.1.4. Python Libraries

- **os:** provides functions for creating and removing a directory (folder), fetching its contents etc.
- cv2
- Numpy
- Pandas
- Matplotlib
- TensorFlow
- TensorFlow Object Detection API
- OpenCV
- PyTorch

2.2. HARDWARE REQUIREMENTS

2.2.1. Windows 10- 64 Bit

2.2.2. AMD Ryzen 5 Gen 3

2.2.3. Graphics card GeForce GTX 1650 4GB

2.2.4. RAM 8 GB

2.2.5. SSD: 512 GB (50 GB minimum)

CHAPTER 3

Literature Survey

3. Literature Survey

The latest advancements in highway research domain and increase in the number of vehicles everyday led to wider exposure and attention towards the development of efficient Intelligent Transportation System (ITS). One of the popular research areas i.e., Automatic Number Plate Recognition (ANPR) aims at determining the characters that exist in the license plate of the vehicles. The ANPR process is a difficult one due to the differences in viewpoint, shapes, colours, patterns, and non-uniform illumination at the time of capturing images.

LP detection algorithms can be roughly divided into traditional methods and neural network models.

Traditional LP detection methods always exploit the abundant edge information [1,2] or the background colour features [3]. Hsieh et al. [2] utilized morphology method to reduce the number of candidates significantly and thus speeded up the plate detection process. Yu et al. [4] proposed a robust method based on wavelet transform and empirical mode decomposition analysis to locate a LP. In [5] the authors analysed vertical edge gradients to select true plate regions. Wang et al. [6] exploited cascade AdaBoost classifier and a voting Towards End-to-End License Plate Detection and Recognition 5 mechanism to elect plate candidates. In [7] a new pattern named Local Structure Patterns was introduced to detect plate regions. Moreover, based on the observation that the LP background always exhibits a regular colour appearance, many works utilize HSI (Hue, Saturation, Intensity) colour space to filter out the LP area. Deb et al. [3] applied HSI colour model to detect candidate regions and achieve 89% accuracy on 90 images. In [8] the authors also exploited a colour checking module to help find LP regions.

Recent progress on region-based **Convolutional Neural Network [**9] stimulates wide applications [10,11] of popular object detection models on LP detection problem. Faster-RCNN [12] utilizes a region proposal network which can generate high-quality region proposals for detection and thus detects objects more accurately and quickly. SSD [13] completely eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network. YOLO [14] and its improved version [15] frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities.

Using the Convolutional Neural Network approach, the VLPR model [16] utilises the Squirrel Search Algorithm (SSA)-based Convolutional Neural Network (CNN) called the SSA-CNN model to effectively recognizes the characters that exist in the segmented image by optimal tuning of CNN parameters.

An event-driven plan recognition model using intuitionistic fuzzy theory was devised in the literature [17]. The authors [18] developed a CN-ELM model to recognize the Electrocardiograms. Besides, a deep local search method using internal spanning tree was devised for parameterized and approximation algorithms. Another lightweight DL model to classify the traffic signs was developed in the literature [19]. A new grammatical model was also presented in the study conducted earlier [20]. An improved model for inspecting deep packets with the help of regular expression was proposed in research conducted earlier [21]. Another improved model to inspect deep packets in data stream detection was introduced in the study [22].

ulan et al. [23] presented a model to exploit weak and sparse classification methods and a strong CNN to isolate the readable LP. In the character analysis, the model eliminated the segmentation phase with the application of a sweeping SVM classifier and a hidden Markov approach to infer the positions. The character classifier was trained using a real sample that has been labelled by existing classifier. However, during the performance validation, a performance loss was observed when the network underwent training on synthetic data.

We have to mention also According to the TensorFlow object detection documentation this pretrained state of the art model can be leveraged to fine tune this model to detect any specific object.

CHAPTER 4

SOFTWARE REQUIREMENT ANALYSIS

SOFTWARE REQUIREMENT ANALYSIS:

4.1 PROBLEM STATEMENT

Automatic number-plate recognition (ANPR) a key technique in most of traffic related applications and is an active research topic in the image processing domain.

Different methods, techniques and algorithms have been developed for license plate detection and recognitions.

But due to the varying characteristics of the license plate from country to country like numbering system, colours, language of characters, style (font) and sizes of license plate, further research is still needed in this area.

4.2 PURPOSED SOLUTION

Our proposed solution is to implement the ANPR system by Transfer Learning method using **TensorFlow Object Detection** model and “easy ocr.”To make a lite model with acceptable accuracy so that it can be used in situation where the exact License Plate(LP) of the vehicle is not required for example congestion analysis.

And another model**(RPnet)** using **deep convolutional neural network** to perform detection and recognition in one model with high accuracy which can used to identify each vehicles and can be used by law enforcement agency, to analyse journey time, traffic management etc.

4.3 DATASET DESCRIPTION:

The data has been obtained from the source:

4.3.1 CCPD (Chinese City Parking Dataset, ECCV):

https://drive.google.com/open?id=1rdEsCUcIUaYOVRkx5IMTRNA7PcGMmSgc

CCPD, a large and comprehensive LP dataset. All images are taken manually by workers of a roadside parking management company and are annotated carefully. To our best knowledge, CCPD is the largest publicly available LP dataset to date with over 250k unique car images, and the only one provides vertices location annotations. With CCPD, we present a novel network model which can predict the bounding box and recognize the corresponding LP number simultaneously with

high speed and accuracy

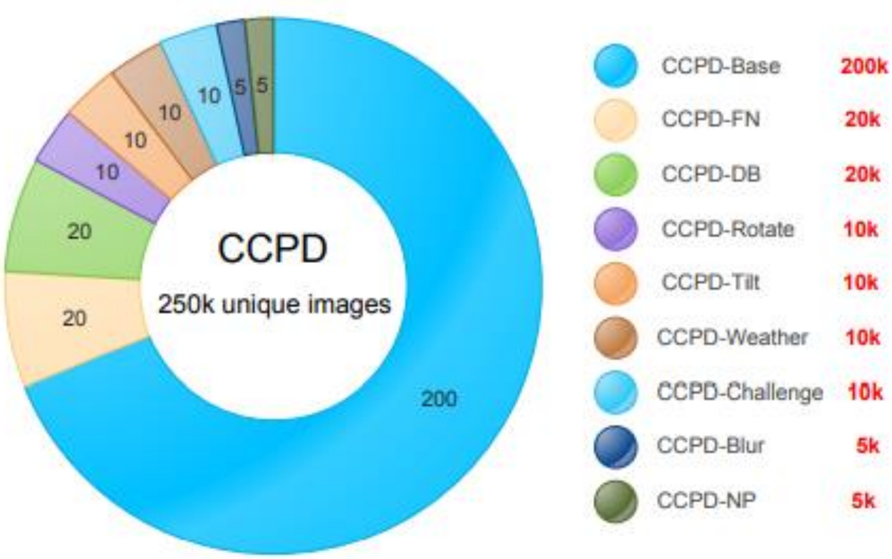


Fig. 2. CCPD layout.

4.4 SOFTWARE DESCRIPTION:

4.4.1 Python:

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming. Many other paradigms are supported via extensions, including design by contract and logic programming. Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

4.4.2 Anaconda:

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and MacOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, both of which are not free. Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source package can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator as a graphical alternative to the command line interface (CLI).

13

The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

4.4.3 Jupyter Notebook:

The Jupyter Notebook is an opensource web application that you can use to create and share documents that contain live code, equations, visualizations, and text. In other words, Jupyter Notebook is an open-source, web-based IDE with deep cross language integration that allows you to create and share documents containing live code, equations, visualizations, and narrative text. Data scientists and engineers use Jupyter for data cleaning and transformation, statistical modeling, visualization, machine learning, deep learning, and much more. Jupyter Notebook's format (ipynb) has become an industry standard and can be rendered in multiple IDEs, GitHub, and other places. Jupyter has support for over 40 programming languages, including Python, R, Julia, and Scala. Notebooks can be shared easily with others, and your code can produce rich, interactive output, including HTML, images, videos, and custom MIME types. It allows you to leverage big data tools such as Spark and explore that same data with pandas, scikit-learn, TensorFlow, and ggplot2.

4.4.4 Numpy:

Numpy is a general-purpose array-processing package. It provides a highperformance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficientmulti dimensional container of generic data. At the core of the NumPy package, is the ndarray
object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

14

4.4.5 Pandas:

Pandas is a fast, powerful, flexible and easy to use opensource data analysis and manipulation tool, built on top of the Python programming language. Pandas is a Python package providing fast, flexible and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis is python.

4.4.6 Matplotlib:

Matplotlib is quite possibly the simplest way to plot data in Python. It is similar to plotting in MATLAB, allowing users full control over fonts, line styles, colors, and axes properties. This allows for complete customization and fine control over the aesthetics of each plot, albeit with a lot of additional lines of code. Plotly is another great Python visualization tool that's capable of handling geographical, scientific, statistical, and financial data. Plotly has several advantages over matplotlib. One of the main advantages is that only a few lines of codes are necessary to create aesthetically pleasing, interactive plots. The interactivity also offers a number of advantages over static matplotlib plots.

4.4.7 TensorFlow:

Tensorflow is an open-source library for numerical computation and large-scale machine learning that ease Google Brain TensorFlow, the process of acquiring data, training models, serving predictions, and refining future results.

Tensorflow bundles together Machine Learning and Deep Learning models and algorithms. It uses Python as a convenient front-end and runs it efficiently in optimized C++.Tensorflow allows developers to create a graph of computations to perform. Each node in the graph represents a mathematical operation and each connection represents data. Hence, instead of dealing with low-details like figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application.

4.4.8 TensorFlow Object Detection API:

The TensorFlow object detection API is the framework for creating a deep learning network that solves object detection problems. There are already pretrained models in their framework which they refer to as Model Zoo. This includes a collection of pretrained models trained on the COCO dataset, the KITTI dataset, and the Open Images Dataset. These models can be used for inference if we are interested in categories only in this dataset. They are also useful for initializing the models when training on the novel dataset.

4.4.9 OpenCV:

OpenCV (Open-Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel. OpenCV features GPU acceleration for real-time operations. is an open source computer vision and machine learning software library.OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception.

4.4.10 PyTorch:

PyTorch is an open-source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Meta AI. A number of pieces of deep learning software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers,

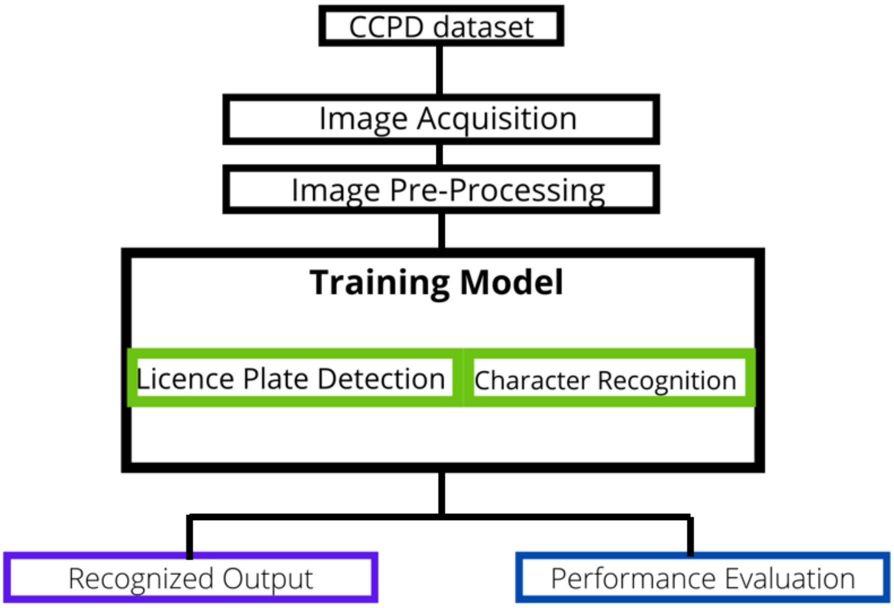
PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU).
- Deep neural networks built on a tape-based automatic differentiation system.

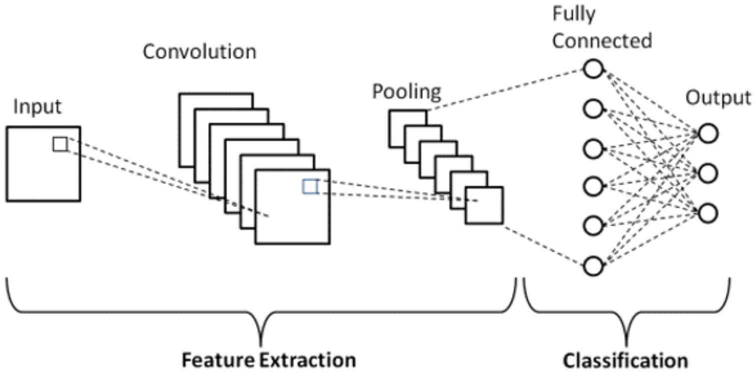
CHAPTER 5

SOFTWARE DESIGN

5.1 METHODOLOGY:



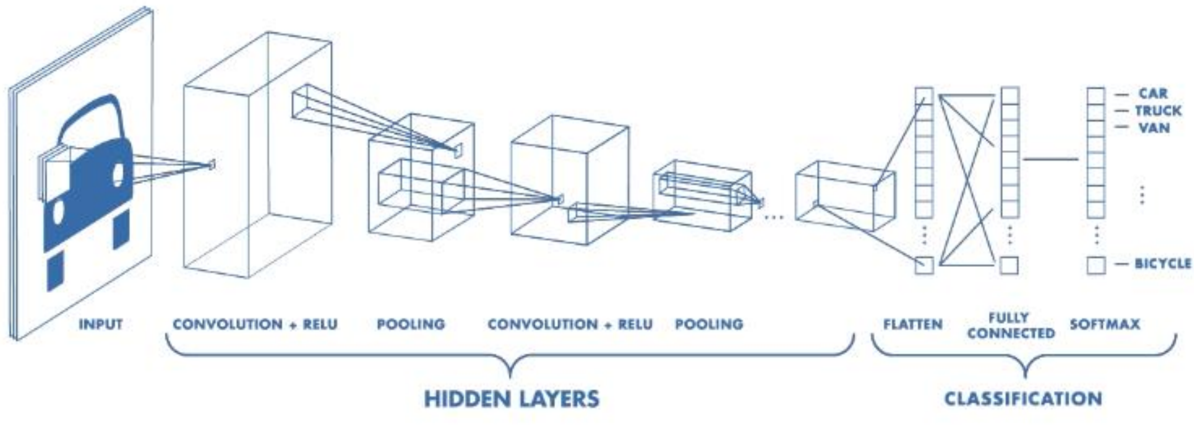
5.2 Convolutional Neural Network Model:



Convolutional Neural Network (CNN) Algorithm:

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what colour each pixel should be.

The human brain processes a huge amount of information the second we see an image. Each neuron works in its own receptive field and is connected to other neurons in a way that they cover the entire visual field. Just as each neuron responds to stimuli only in the restricted region of the visual field called the receptive field in the biological vision system, each neuron in a CNN processes data only in its receptive field as well. The layers are arranged in such a way so that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along. By using a CNN, one can enable sight to computers.



5.2.1 Convolution Layer

The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load. This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three

(RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

5.2.2 Pooling Layer

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

5.2.3 Fully Connected Layer

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect. The Fully Connected layer helps to map the representation between the input and the output.

5.2.4 Non-Linearity Layers

Since convolution is a linear operation and images are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map.

There are several types of non-linear operations, the popular ones being:

a. <u>Sigmoid</u>

The sigmoid non-linearity has the mathematical form $\sigma(\kappa) = 1/(1+e^{-}\kappa)$. It takes a real-valued number and “squashes” it into a range between 0 and 1.

b. <u>Tanh</u>

Tanh squashes a real-valued number to the range [-1, 1]. Like sigmoid, the activation saturates, but — unlike the sigmoid neurons — its output is zero centered.

c. <u>ReLU</u>

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function $f(\kappa)=\max(0, \kappa)$. In other words, the activation is simply threshold at zero.

CHAPTER 6

CODE TEMPLATES AND RESULTS

6.1 -The program codes to create, train and test the TensorFlow Model:

0. Setup Paths

```
import os


CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
PRETRAINED_MODEL_URL = 
'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'


paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
```

```
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }

files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models', CUSTOM_MODEL_NAME, 'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'], TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}

for path in paths.values():
    if not os.path.exists(path):
        if os.name == 'posix':
            !mkdir -p {path}
        if os.name == 'nt':
            !mkdir {path}
```

1. Download TF Models Pretrained Models from Tensorflow Model Zoo and Install TFOD

```
if os.name=='nt':
    !pip install wget
    import wget


if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
```

```
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}



# Install Tensorflow Object Detection
if os.name=='posix':
    !apt-get install protobuf-compiler
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && cp object_detection/packages/tf2/setup.py . && python -m pip install .


if os.name=='nt':
    url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-win64.zip"
    wget.download(url)
    !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
    !cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
    os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && copy object_detection\\packages\\tf2\\setup.py setup.py && python setup.py build && python setup.py install
    !cd Tensorflow/models/research/slim && pip install -e .


VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_builder_tf2_test.py')
# Verify Installation
!python {VERIFICATION_SCRIPT}


import object_detection




if os.name =='posix':
```

```
    !wget {PRETRAINED_MODEL_URL}

    !mv {PRETRAINED_MODEL_NAME+'.tar.gz'}
{paths['PRETRAINED_MODEL_PATH']}

    !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf
{PRETRAINED_MODEL_NAME+'.tar.gz'}

if os.name == 'nt':

    wget.download(PRETRAINED_MODEL_URL)

    !move {PRETRAINED_MODEL_NAME+'.tar.gz'}
{paths['PRETRAINED_MODEL_PATH']}

    !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf
{PRETRAINED_MODEL_NAME+'.tar.gz'}
```

2. Create Label Map

```
labels = [{'name':'ThumbsUp', 'id':1}, {'name':'ThumbsDown', 'id':2},
{'name':'ThankYou', 'id':3}, {'name':'LiveLong', 'id':4}]


with open(files['LABELMAP'], 'w') as f:

    for label in labels:

        f.write('item { \n')

        f.write('\tname:\'{}\'\n'.format(label['name']))

        f.write('\tid:{}\n'.format(label['id']))

        f.write('}\n')
```

3. Create TF records

```
# OPTIONAL IF RUNNING ON COLAB

ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')

if os.path.exists(ARCHIVE_FILES):

  !tar -zxvf {ARCHIVE_FILES}
```

```
if not os.path.exists(files['TF_RECORD_SCRIPT']):
    !git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}
```

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

4. Copy Model Config to Training Folder

```
if os.name =='posix':
    !cp {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKPOINT_PATH'])}
if os.name == 'nt':
    !copy {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKPOINT_PATH'])}
```

5. Update Config For Transfer Learning

```
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
```

```
config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])



pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)



pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'checkpoint', 'ckpt-0')
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'train.record')]
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'test.record')]




config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    f.write(config_text)
```

6. Train the model

```
TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection', 'model_main_tf2.py')


command = "python {} --model_dir={} --pipeline_config_path={} --
num_train_steps=2000".format(TRAINING_SCRIPT,
paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])


!{command}
```

7. Evaluate the Model

```
command = "python {} --model_dir={} --pipeline_config_path={} --
checkpoint_dir={}".format(TRAINING_SCRIPT,
paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'],
paths['CHECKPOINT_PATH'])


!{command}
```

8. Load Train Model From Checkpoint

```
import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
```

```
detection_model = model_builder.build(model_config=configs['model'], is_training=False)


# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-5')).expect_partial()


@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

9. Detect from an Image

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline


category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])


IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'abc.jpg')


img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)


input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
```

```
detections = detect_fn(input_tensor)

num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections

# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

label_id_offset = 1
image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
          image_np_with_detections,
          detections['detection_boxes'],
          detections['detection_classes']+label_id_offset,
          detections['detection_scores'],
          category_index,
          use_normalized_coordinates=True,
          max_boxes_to_draw=5,
          min_score_thresh=.8,
          agnostic_mode=False)

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

"

10. Apply ROI filtering and OCR

```
for result in ocr_result:
    print(np.sum(np.subtract(result[0][2],result[0][1])))
    print(result[1])
#OCR Filtering
region_threshold = 0.05
filter_text(region, ocr_result, region_threshold)
#Bring it Together
region_threshold = 0.6
def ocr_it(image, detections, detection_threshold, region_threshold):

    # Scores, boxes and classes above threhold
    scores = list(filter(lambda x: x> detection_threshold, detections['detection_scores']))
    boxes = detections['detection_boxes'][:len(scores)]
    classes = detections['detection_classes'][:len(scores)]

    # Full image dimensions
    width = image.shape[1]
    height = image.shape[0]

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

```
    text = filter_text(region, ocr_result, region_threshold)


    plt.imshow(cv2.cvtColor(region, cv2.COLOR_BGR2RGB))
    plt.show()
    print(text)
    return text, region
text, region = ocr_it(image_np_with_detections, detections, detection_threshold, region_threshold)
```


```
['("HR26DO5554']
```

6.2 The program codes to create, train and test the RPnet Model:

6.2.1 Load_data.py

```
from torch.utils.data import *
from imutils import paths
import cv2
import numpy as np


class labelFpsDataLoader(Dataset):
    def __init__(self, img_dir, imgSize, is_transform=None):
        self.img_dir = img_dir
        self.img_paths = []
        for i in range(len(img_dir)):
            self.img_paths += [el for el in paths.list_images(img_dir[i])]
        # self.img_paths = os.listdir(img_dir)
        # print self.img_paths
        self.img_size = imgSize
        self.is_transform = is_transform

    def __len__(self):
        return len(self.img_paths)

    def __getitem__(self, index):
        img_name = self.img_paths[index]
```

```
        img = cv2.imread(img_name)
        # img = img.astype('float32')
        resizedImage = cv2.resize(img, self.img_size)
        resizedImage = np.transpose(resizedImage, (2,0,1))
        resizedImage = resizedImage.astype('float32')
        resizedImage /= 255.0
        lbl = img_name.split('/')[-1].rsplit('.', 1)[0].split('-')[-3]

        iname = img_name.rsplit('/', 1)[-1].rsplit('.', 1)[0].split('-')
        # fps = [[int(eel) for eel in el.split('&')] for el in iname[3].split('_')]
        # leftUp, rightDown = [min([fps[el][0] for el in range(4)]), min([fps[el][1] for el in
range(4)])], [
        #     max([fps[el][0] for el in range(4)]), max([fps[el][1] for el in range(4)])]
        [leftUp, rightDown] = [[int(eel) for eel in el.split('&')] for el in iname[2].split('_')]
        ori_w, ori_h = [float(int(el)) for el in [img.shape[1], img.shape[0]]]
        new_labels = [(leftUp[0] + rightDown[0]) / (2 * ori_w), (leftUp[1] + rightDown[1]) /
(2 * ori_h),
                      (rightDown[0] - leftUp[0]) / ori_w, (rightDown[1] - leftUp[1]) / ori_h]

        return resizedImage, new_labels, lbl, img_name



class labelTestDataLoader(Dataset):
    def __init__(self, img_dir, imgSize, is_transform=None):
        self.img_dir = img_dir
        self.img_paths = []
        for i in range(len(img_dir)):
            self.img_paths += [el for el in paths.list_images(img_dir[i])]
        # self.img_paths = os.listdir(img_dir)
        # print self.img_paths
```

```
        self.img_size = imgSize
        self.is_transform = is_transform

    def __len__(self):
        return len(self.img_paths)

    def __getitem__(self, index):
        img_name = self.img_paths[index]
        img = cv2.imread(img_name)
        # img = img.astype('float32')
        resizedImage = cv2.resize(img, self.img_size)
        resizedImage = np.transpose(resizedImage, (2,0,1))
        resizedImage = resizedImage.astype('float32')
        resizedImage /= 255.0
        lbl = img_name.split('/')[-1].split('.')[0].split('-')[-3]
        return resizedImage, lbl, img_name




class ChaLocDataLoader(Dataset):
    def __init__(self, img_dir,imgSize, is_transform=None):
        self.img_dir = img_dir
        self.img_paths = []
        print("image dir no :{}".format(len(img_dir)))
        for i in range(len(img_dir)):
            self.img_paths += [el for el in paths.list_images(img_dir[i])]
        # self.img_paths = os.listdir(img_dir)
        # print self.img_paths
        self.img_size = imgSize
```

```
        self.is_transform = is_transform

    def __len__(self):
        return len(self.img_paths)

    def __getitem__(self, index):
        img_name = self.img_paths[index]
        img = cv2.imread(img_name)
        resizedImage = cv2.resize(img, self.img_size)
        resizedImage = np.reshape(resizedImage, (resizedImage.shape[2],
resizedImage.shape[0], resizedImage.shape[1]))

        iname = img_name.rsplit('/', 1)[-1].rsplit('.', 1)[0].split('-')
        [leftUp, rightDown] = [[int(eel) for eel in el.split('&')] for el in iname[2].split('_')]

        # tps = [[int(eel) for eel in el.split('&')] for el in iname[2].split('_')]
        # for dot in tps:
        #     cv2.circle(img, (int(dot[0]), int(dot[1])), 2, (0, 0, 255), 2)
        # cv2.imwrite("/home/xubb/1_new.jpg", img)

        ori_w, ori_h = float(img.shape[1]), float(img.shape[0])
        assert img.shape[0] == 1160
        new_labels = [(leftUp[0] + rightDown[0])/(2*ori_w), (leftUp[1] +
rightDown[1])/(2*ori_h), (rightDown[0]-leftUp[0])/ori_w, (rightDown[1]-
leftUp[1])/ori_h]

        resizedImage = resizedImage.astype('float32')
        # Y = Y.astype('int8')
        resizedImage /= 255.0
        # lbl = img_name.split('.')[0].rsplit('-',1)[-1].split('_')[:-1]
```

```
        # lbl = img_name.split('/')[-1].split('.')[0].rsplit('-',1)[-1]
        # lbl = map(int, lbl)
        # lbl2 = [[el] for el in lbl]

        # resizedImage = torch.from_numpy(resizedImage).float()
        return resizedImage, new_labels



class demoTestDataLoader(Dataset):
    def __init__(self, img_dir, imgSize, is_transform=None):
        # img_dir="D:\CCPD2019\demot"
        self.img_dir = img_dir
        self.img_paths = []

        print("image dir no:{}".format(len(img_dir)))
        # print(img_dir[0])
        # print(img_dir[1])
        # print(img_dir[2])
        for i in range(len(img_dir)):
            self.img_paths += [el for el in paths.list_images(img_dir[i])]
        # self.img_paths = os.listdir(img_dir)
        print("image no:{}".format(len(self.img_paths)))

        self.img_size = imgSize
        self.is_transform = is_transform

    def __len__(self):
        return len(self.img_paths)
```

```
def __getitem__(self, index):
    img_name = self.img_paths[index]
    img = cv2.imread(img_name)
    # img = img.astype('float32')
    resizedImage = cv2.resize(img, self.img_size)
    resizedImage = np.transpose(resizedImage, (2,0,1))
    resizedImage = resizedImage.astype('float32')
    resizedImage /= 255.0
    return resizedImage, img_name
```

6.2.2 roi_pooling.py

```
import torch
import torch.autograd as ag
from torch.autograd.function import Function
from torch._thnn import type2backend


class AdaptiveMaxPool2d(Function):
    def __init__(self, out_w, out_h):
        super(AdaptiveMaxPool2d, self).__init__()
        self.out_w = out_w
        self.out_h = out_h

    def forward(self, input):
        output = input.new()
        indices = input.new().long()
        self.save_for_backward(input)
        self.indices = indices
        self._backend = type2backend[input.type()]
        self._backend.SpatialAdaptiveMaxPooling_updateOutput(
            self._backend.library_state, input, output, indices,
            self.out_w, self.out_h)
        return output

    def backward(self, grad_output):
        input, = self.saved_tensors
        indices = self.indices
```

```
        grad_input = grad_output.new()
        self._backend.SpatialAdaptiveMaxPooling_updateGradInput(
            self._backend.library_state, input, grad_output, grad_input,
            indices)
        return grad_input, None


def adaptive_max_pool(input, size):
    return AdaptiveMaxPool2d(size[0], size[1])(input)


def roi_pooling(input, rois, size=(7, 7), spatial_scale=1.0):
    assert (rois.dim() == 2)
    assert (rois.size(1) == 5)
    output = []
    rois = rois.data.float()
    num_rois = rois.size(0)

    rois[:, 1:].mul_(spatial_scale)
    rois = rois.long()
    for i in range(num_rois):
        roi = rois[i]
        im_idx = roi[0]
        # im = input.narrow(0, im_idx, 1)
        im = input.narrow(0, im_idx, 1)[..., roi[2]:(roi[4] + 1), roi[1]:(roi[3] + 1)]
        output.append(adaptive_max_pool(im, size))

    return torch.cat(output, 0)
```

```
def roi_pooling_ims(input, rois, size=(7, 7), spatial_scale=1.0):
    # written for one roi one image
    # size: (w, h)
    assert (rois.dim() == 2)
    assert len(input) == len(rois)
    assert (rois.size(1) == 4)
    output = []
    rois = rois.data.float()
    num_rois = rois.size(0)

    rois[:, 1:].mul_(spatial_scale)
    rois = rois.long()
    for i in range(num_rois):
        roi = rois[i]
        # im = input.narrow(0, im_idx, 1)
        im = input.narrow(0, i, 1)[..., roi[1]:(roi[3] + 1), roi[0]:(roi[2] + 1)]
        output.append(adaptive_max_pool(im, size))

    return torch.cat(output, 0)

if __name__ == '__main__':
    input = ag.Variable(torch.rand(2, 1, 10, 10), requires_grad=True)
    rois = ag.Variable(torch.LongTensor([[1, 2, 7, 8], [3, 3, 8, 8]]), requires_grad=False)

    out = roi_pooling_ims(input, rois, size=(8, 8))
    out.backward(out.data.clone().uniform_())

    # input = ag.Variable(torch.rand(2, 1, 10, 10), requires_grad=True)
```

```
    # rois = ag.Variable(torch.LongTensor([[0, 1, 2, 7, 8], [0, 3, 3, 8, 8], [1, 3, 3, 8, 8]]),
requires_grad=False)
    # rois = ag.Variable(torch.LongTensor([[0,3,3,8,8]]),requires_grad=False)


    # out = adaptive_max_pool(input, (3, 3))
    # out.backward(out.data.clone().uniform_())


    # out = roi_pooling(input, rois, size=(3, 3))
    # out.backward(out.data.clone().uniform_())
```

6.2.3 wr2.py

```
# Code in cnn_fn_pytorch.py
from __future__ import print_function, division
import cv2
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
```

```
import numpy as np
import os
import argparse
from time import time
from load_data import *
from torch.optim import lr_scheduler
if __name__ == '__main__':
 startInitial = time()
 ap = argparse.ArgumentParser()
 ap.add_argument("-i", "--images", required=True,
        help="path to the input file")
 ap.add_argument("-n", "--epochs", default=25,
        help="epochs for train")
 ap.add_argument("-b", "--batchsize", default=4,
        help="batch size for train")
 ap.add_argument("-r", "--resume", default='111',
        help="file for re-train")
 ap.add_argument("-w", "--writeFile", default='wR2.out',
        help="file for output")
 args = vars(ap.parse_args())

 use_gpu = torch.cuda.is_available()
 print (use_gpu)
#  torch.backends.cudnn.benchmark = True
 if torch.cuda.is_available():
    device = torch.device("cuda")
    print("working on gpu")
 else:
    device = torch.device("cpu")
```

```
  print("working on cpu")

numClasses = 4
imgSize = (480, 480)
batchSize = int(args["batchsize"]) if use_gpu else 8
modelFolder = 'wR2/'
storeName = modelFolder + 'wR2.pth'
if not os.path.isdir(modelFolder):
    os.mkdir(modelFolder)

epochs = int(args["epochs"])
#   initialize the output file
with open(args['writeFile'], 'wb') as outF:
  pass



def get_n_params(model):
    pp=0
    for p in list(model.parameters()):
        nn=1
        for s in list(p.size()):
            nn = nn*s
        pp += nn
    return pp



class wR2(nn.Module):
    def __init__(self, num_classes=1000):
        super(wR2, self).__init__()
```

```
hidden1 = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=48, kernel_size=5, padding=2, stride=2),
    nn.BatchNorm2d(num_features=48),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
hidden2 = nn.Sequential(
    nn.Conv2d(in_channels=48, out_channels=64, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=64),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden3 = nn.Sequential(
    nn.Conv2d(in_channels=64, out_channels=128, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=128),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
hidden4 = nn.Sequential(
    nn.Conv2d(in_channels=128, out_channels=160, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=160),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden5 = nn.Sequential(
```

```
    nn.Conv2d(in_channels=160, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
hidden6 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden7 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
hidden8 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden9 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=3, padding=1),
```

```
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
hidden10 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=3, padding=1),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
self.features = nn.Sequential(
    hidden1,
    hidden2,
    hidden3,
    hidden4,
    hidden5,
    hidden6,
    hidden7,
    hidden8,
    hidden9,
    hidden10
)
self.classifier = nn.Sequential(
    nn.Linear(23232, 100),
   # nn.ReLU(inplace=True),
    nn.Linear(100, 100),
   # nn.ReLU(inplace=True),
```

```
            nn.Linear(100, num_classes),
        )

    def forward(self, x):
        x1 = self.features(x)
        x11 = x1.view(x1.size(0), -1)
        x = self.classifier(x11)
        return x



epoch_start = 0
resume_file = str(args["resume"])
if not resume_file == '111':
    # epoch_start = int(resume_file[resume_file.find('pth') + 3:]) + 1
    if not os.path.isfile(resume_file):
        print ("fail to load existed model! Existing ...")
        exit(0)
    print ("Load existed model! %s" % resume_file)
    model_conv = wR2(numClasses)
    model_conv = torch.nn.DataParallel(model_conv,
device_ids=range(torch.cuda.device_count()))
    model_conv.load_state_dict(torch.load(resume_file))
    model_conv = model_conv.cuda()
else:
    model_conv = wR2(numClasses)
    if use_gpu:
        model_conv = torch.nn.DataParallel(model_conv,
device_ids=range(torch.cuda.device_count()))
        model_conv = model_conv.cuda()
```

```
print(model_conv)
print(get_n_params(model_conv))

criterion = nn.MSELoss()
optimizer_conv = optim.SGD(model_conv.parameters(), lr=0.001, momentum=0.9)
lrScheduler = lr_scheduler.StepLR(optimizer_conv, step_size=5, gamma=0.1)

# optimizer_conv = optim.Adam(model_conv.parameters(), lr=0.01)

# dst = LocDataLoader([args["images"]], imgSize)
# print(args["images"])
dst = ChaLocDataLoader(args["images"].split(','), imgSize)
# print(dst.shape)

trainloader = DataLoader(dst, batch_size=batchSize, shuffle=True, num_workers=4)


def train_model(model, criterion, optimizer, num_epochs=25):
   # since = time.time()
    for epoch in range(epoch_start, num_epochs):
        lossAver = []
        model.train(True)
        lrScheduler.step()
        start = time()

        for i, (XI, YI) in enumerate(trainloader):
            # print('%s/%s %s' % (i, times, time()-start))
            YI = np.array([el.numpy() for el in YI]).T
            if use_gpu:
```

```
            x = Variable(XI.cuda(0))
            y = Variable(torch.FloatTensor(YI).cuda(0), requires_grad=False)
        else:
            x = Variable(XI)
            y = Variable(torch.FloatTensor(YI), requires_grad=False)
        # Forward pass: Compute predicted y by passing x to the model
        y_pred = model(x)

        # Compute and print loss
        loss = 0.0
        if len(y_pred) == batchSize:
            loss += 0.8 * nn.L1Loss().cuda()(y_pred[:][:2], y[:][:2])
            loss += 0.2 * nn.L1Loss().cuda()(y_pred[:][2:], y[:][2:])
            lossAver.append(loss.data)

            # Zero gradients, perform a backward pass, and update the weights.
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            torch.save(model.state_dict(), storeName)
        if i % 50 == 1:
            with open(args['writeFile'], 'a') as outF:
                outF.write('train %s images, use %s seconds, loss %s\n' % (i*batchSize,
time() - start, sum(lossAver[-50:]) / len(lossAver[-50:])))
    print ('epoc:%s %s epoc time:%s time elapsed:%s\n' % (epoch, sum(lossAver) /
len(lossAver), time()-start,time()-startInitial))
    with open(args['writeFile'], 'a') as outF:
        outF.write('Epoch: %s %s %s\n' % (epoch, sum(lossAver) / len(lossAver), time()-
start))
    torch.save(model.state_dict(), storeName + str(epoch))
```

```
    return model

#  torch.backends.cudnn.benchmark = True
 model_conv = train_model(model_conv, criterion, optimizer_conv, num_epochs=epochs)
```

6.2.4 rpnet.py

```
# Compared to fh0.py
# fh02.py remove the redundant ims in model input
from __future__ import print_function, division
import cv2
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
import numpy as np
import os
import argparse
from time import time
from load_data import *
from roi_pooling import roi_pooling_ims
from torch.optim import lr_scheduler


if __name__ == '__main__':
 startInitial = time()
 ap = argparse.ArgumentParser()
 ap.add_argument("-i", "--images", required=True,
                help="path to the input file")
                #default is 100000
 ap.add_argument("-n", "--epochs", default=25,
                help="epochs for train")
 ap.add_argument("-b", "--batchsize", default=5,
                help="batch size for train")
 ap.add_argument("-se", "--start_epoch", required=True,
                help="start epoch for train")
```

```
ap.add_argument("-t", "--test", required=True,
        help="dirs for test")
ap.add_argument("-r", "--resume", default='111',
        help="file for re-train")
ap.add_argument("-f", "--folder", required=True,
        help="folder to store model")
ap.add_argument("-w", "--writeFile", default='fh02.out',
        help="file for output")
args = vars(ap.parse_args())

# wR2Path = './wR2/wR2.pth2'
wR2Path = 'wR2.pth'
use_gpu = torch.cuda.is_available()
print (use_gpu)

numClasses = 7
numPoints = 4
classifyNum = 35
imgSize = (480, 480)
# lpSize = (128, 64)
provNum, alphaNum, adNum = 38, 25, 35
batchSize = int(args["batchsize"]) if use_gpu else 2
trainDirs = args["images"].split(',')
testDirs = args["test"].split(',')
print("image folder is:{}".format(str(args["images"]) ))
print("test image folder is:{}".format(str(args["test"]) ))
modelFolder = str(args["folder"]) if str(args["folder"])[-1] == '/' else str(args["folder"]) +
'/'
print("model folder is:{}".format(modelFolder))
```

```
storeName = modelFolder + 'fh02.pth'
if not os.path.isdir(modelFolder):
    os.mkdir(modelFolder)

epochs = int(args["epochs"])
#   initialize the output file
if not os.path.isfile(args['writeFile']):
    with open(args['writeFile'], 'wb') as outF:
        pass



def get_n_params(model):
    pp=0
    for p in list(model.parameters()):
        nn=1
        for s in list(p.size()):
            nn = nn*s
        pp += nn
    return pp



class wR2(nn.Module):
    def __init__(self, num_classes=1000):
        super(wR2, self).__init__()
        hidden1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=48, kernel_size=5, padding=2, stride=2),
            nn.BatchNorm2d(num_features=48),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
```

```
    nn.Dropout(0.2)
)
hidden2 = nn.Sequential(
    nn.Conv2d(in_channels=48, out_channels=64, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=64),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden3 = nn.Sequential(
    nn.Conv2d(in_channels=64, out_channels=128, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=128),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
hidden4 = nn.Sequential(
    nn.Conv2d(in_channels=128, out_channels=160, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=160),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden5 = nn.Sequential(
    nn.Conv2d(in_channels=160, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
```

```
)
hidden6 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden7 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
hidden8 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden9 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=3, padding=1),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
```

```
        hidden10 = nn.Sequential(
            nn.Conv2d(in_channels=192, out_channels=192, kernel_size=3, padding=1),
            nn.BatchNorm2d(num_features=192),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
            nn.Dropout(0.2)
        )
        self.features = nn.Sequential(
            hidden1,
            hidden2,
            hidden3,
            hidden4,
            hidden5,
            hidden6,
            hidden7,
            hidden8,
            hidden9,
            hidden10
        )
        self.classifier = nn.Sequential(
            nn.Linear(23232, 100),
            # nn.ReLU(inplace=True),
            nn.Linear(100, 100),
            # nn.ReLU(inplace=True),
            nn.Linear(100, num_classes),
        )

    def forward(self, x):
        x1 = self.features(x)
```

```
        x11 = x1.view(x1.size(0), -1)
        x = self.classifier(x11)
        return x


class fh02(nn.Module):
    def __init__(self, num_points, num_classes, wrPath=None):
        super(fh02, self).__init__()
        self.load_wR2(wrPath)
        print("model loaded")
        self.classifier1 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
            # nn.ReLU(inplace=True),
            # nn.Dropout(),
            nn.Linear(128, provNum),
        )
        self.classifier2 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
            # nn.ReLU(inplace=True),
            # nn.Dropout(),
            nn.Linear(128, alphaNum),
        )
        self.classifier3 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
            # nn.ReLU(inplace=True),
            # nn.Dropout(),
```

```
        nn.Linear(128, adNum),
    )
    self.classifier4 = nn.Sequential(
        # nn.Dropout(),
        nn.Linear(53248, 128),
       # nn.ReLU(inplace=True),
       # nn.Dropout(),
        nn.Linear(128, adNum),
    )
    self.classifier5 = nn.Sequential(
       # nn.Dropout(),
        nn.Linear(53248, 128),
        # nn.ReLU(inplace=True),
        # nn.Dropout(),
        nn.Linear(128, adNum),
    )
    self.classifier6 = nn.Sequential(
        # nn.Dropout(),
        nn.Linear(53248, 128),
        # nn.ReLU(inplace=True),
        # nn.Dropout(),
        nn.Linear(128, adNum),
    )
    self.classifier7 = nn.Sequential(
       # nn.Dropout(),
        nn.Linear(53248, 128),
        # nn.ReLU(inplace=True),
       # nn.Dropout(),
        nn.Linear(128, adNum),
```

```
        )

    def load_wR2(self, path):
        self.wR2 = wR2(numPoints)
        self.wR2 = torch.nn.DataParallel(self.wR2,
device_ids=range(torch.cuda.device_count()))
        if not path is None:
            self.wR2.load_state_dict(torch.load(path))
            # self.wR2 = self.wR2.cuda()
        # for param in self.wR2.parameters():
        #     param.requires_grad = False

    def forward(self, x):
        x0 = self.wR2.module.features[0](x)
        _x1 = self.wR2.module.features[1](x0)
        x2 = self.wR2.module.features[2](_x1)
        _x3 = self.wR2.module.features[3](x2)
        x4 = self.wR2.module.features[4](_x3)
        _x5 = self.wR2.module.features[5](x4)

        x6 = self.wR2.module.features[6](_x5)
        x7 = self.wR2.module.features[7](x6)
        x8 = self.wR2.module.features[8](x7)
        x9 = self.wR2.module.features[9](x8)
        x9 = x9.view(x9.size(0), -1)
        boxLoc = self.wR2.module.classifier(x9)

        h1, w1 = _x1.data.size()[2], _x1.data.size()[3]
        p1 = Variable(torch.FloatTensor([[w1,0,0,0],[0,h1,0,0],[0,0,w1,0],[0,0,0,h1]]).cuda(),
requires_grad=False)
```

```
        h2, w2 = _x3.data.size()[2], _x3.data.size()[3]
        p2 = Variable(torch.FloatTensor([[w2,0,0,0],[0,h2,0,0],[0,0,w2,0],[0,0,0,h2]]).cuda(),
requires_grad=False)
        h3, w3 = _x5.data.size()[2], _x5.data.size()[3]
        p3 = Variable(torch.FloatTensor([[w3,0,0,0],[0,h3,0,0],[0,0,w3,0],[0,0,0,h3]]).cuda(),
requires_grad=False)

        # x, y, w, h --> x1, y1, x2, y2
        assert boxLoc.data.size()[1] == 4
        postfix = Variable(torch.FloatTensor([[1,0,1,0],[0,1,0,1],[-0.5,0,0.5,0],[0,-
0.5,0,0.5]]).cuda(), requires_grad=False)
        boxNew = boxLoc.mm(postfix).clamp(min=0, max=1)

        # input = Variable(torch.rand(2, 1, 10, 10), requires_grad=True)
        # rois = Variable(torch.LongTensor([[0, 1, 2, 7, 8], [0, 3, 3, 8, 8], [1, 3, 3, 8, 8]]),
requires_grad=False)
        roi1 = roi_pooling_ims(_x1, boxNew.mm(p1), size=(16, 8))
        roi2 = roi_pooling_ims(_x3, boxNew.mm(p2), size=(16, 8))
        roi3 = roi_pooling_ims(_x5, boxNew.mm(p3), size=(16, 8))
        rois = torch.cat((roi1, roi2, roi3), 1)

        _rois = rois.view(rois.size(0), -1)

        y0 = self.classifier1(_rois)
        y1 = self.classifier2(_rois)
        y2 = self.classifier3(_rois)
        y3 = self.classifier4(_rois)
        y4 = self.classifier5(_rois)
        y5 = self.classifier6(_rois)
        y6 = self.classifier7(_rois)
        return boxLoc, [y0, y1, y2, y3, y4, y5, y6]
```

```
epoch_start = int(args["start_epoch"])
resume_file = str(args["resume"])
if not resume_file == '111':
    # epoch_start = int(resume_file[resume_file.find('pth') + 3:]) + 1
    if not os.path.isfile(resume_file):
        print ("fail to load existed model! Existing ...")
        exit(0)
    print ("Load existed model! %s" % resume_file)
    model_conv = fh02(numPoints, numClasses)
    model_conv = torch.nn.DataParallel(model_conv,
device_ids=range(torch.cuda.device_count()))
    model_conv.load_state_dict(torch.load(resume_file))
    model_conv = model_conv.cuda()
else:
    model_conv = fh02(numPoints, numClasses, wR2Path)
    if use_gpu:
        model_conv = torch.nn.DataParallel(model_conv,
device_ids=range(torch.cuda.device_count()))
        model_conv = model_conv.cuda()

print(model_conv)
print(get_n_params(model_conv))

criterion = nn.CrossEntropyLoss()
# optimizer_conv = optim.RMSprop(model_conv.parameters(), lr=0.01, momentum=0.9)
optimizer_conv = optim.SGD(model_conv.parameters(), lr=0.001, momentum=0.9)

dst = labelFpsDataLoader(trainDirs, imgSize)
```

```
trainloader = DataLoader(dst, batch_size=batchSize, shuffle=True, num_workers=8)
lrScheduler = lr_scheduler.StepLR(optimizer_conv, step_size=5, gamma=0.1)


def isEqual(labelGT, labelP):
    compare = [1 if int(labelGT[i]) == int(labelP[i]) else 0 for i in range(7)]
    # print(sum(compare))
    return sum(compare)


def eval(model, test_dirs):
    count, error, correct = 0, 0, 0
    dst = labelTestDataLoader(test_dirs, imgSize)
    testloader = DataLoader(dst, batch_size=1, shuffle=True, num_workers=8)
    start = time()
    for i, (XI, labels, ims) in enumerate(testloader):
        count += 1
        YI = [[int(ee) for ee in el.split('_')[:7]] for el in labels]
        if use_gpu:
            x = Variable(XI.cuda(0))
        else:
            x = Variable(XI)
        # Forward pass: Compute predicted y by passing x to the model

        fps_pred, y_pred = model(x)

        outputY = [el.data.cpu().numpy().tolist() for el in y_pred]
        labelPred = [t[0].index(max(t[0])) for t in outputY]
```

```
        #   compare YI, outputY
        try:
            if isEqual(labelPred, YI[0]) == 7:
                correct += 1
            else:
                pass
        except:
            error += 1
    return count, correct, error, float(correct) / count, (time() - start) / count


#epoc is 25
 def train_model(model, criterion, optimizer, num_epochs=25):
    # since = time.time()
    for epoch in range(epoch_start,num_epochs):
        lossAver = []
        model.train(True)
        lrScheduler.step()
        start = time()

        for i, (XI, Y, labels, ims) in enumerate(trainloader):
            if not len(XI) == batchSize:
                continue

            YI = [[int(ee) for ee in el.split('_')[:7]] for el in labels]
            Y = np.array([el.numpy() for el in Y]).T
            if use_gpu:
               #  print('using gpu')
                x = Variable(XI.cuda(0))
                y = Variable(torch.FloatTensor(Y).cuda(0), requires_grad=False)
```

```
 else:
    #  print(' not using gpu')
     x = Variable(XI)
     y = Variable(torch.FloatTensor(Y), requires_grad=False)
# Forward pass: Compute predicted y by passing x to the model

 try:
     fps_pred, y_pred = model(x)
 except:
     continue

# Compute and print loss
 loss = 0.0
 loss += 0.8 * nn.L1Loss().cuda()(fps_pred[:][:2], y[:][:2])
 loss += 0.2 * nn.L1Loss().cuda()(fps_pred[:][2:], y[:][2:])
 for j in range(7):
     l = Variable(torch.LongTensor([el[j] for el in YI]).cuda(0))
     loss += criterion(y_pred[j], l)

# Zero gradients, perform a backward pass, and update the weights.
 optimizer.zero_grad()
 loss.backward()
 optimizer.step()

 try:
     lossAver.append(loss.data)
 except:
     pass
```

```
            if i % 50 == 1:
                with open(args['writeFile'], 'a') as outF:
                    outF.write('train %s images, use %s seconds, loss %s\n' % (i*batchSize, time() - start, sum(lossAver) / len(lossAver) if len(lossAver)>0 else 'NoLoss'))
                torch.save(model.state_dict(), storeName)
        print ('%s %s %s\n' % (epoch, sum(lossAver) / len(lossAver), time()-start))
        model.eval()
        count, correct, error, precision, avgTime = eval(model, testDirs)
        with open(args['writeFile'], 'a') as outF:
            outF.write('%s %s %s\n' % (epoch, sum(lossAver) / len(lossAver), time() - start))
            outF.write('*** total %s error %s precision %s avgTime %s\n' % (count, error, precision, avgTime))
            print ('epoc:%s %s epoc time:%s time elapsed:%s\n' % (epoch, sum(lossAver) / len(lossAver), time()-start,time()-startInitial))
        torch.save(model.state_dict(), storeName + str(epoch))
    return model



model_conv = train_model(model_conv, criterion, optimizer_conv, num_epochs=epochs)
```

6.2.4 rpnetEval.py

```
#encoding:utf-8
import cv2
import torch
from torch.autograd import Variable
import torch.nn as nn
import argparse
import numpy as np
from os import path, mkdir
from load_data import *
from time import time
from roi_pooling import roi_pooling_ims
from shutil import copyfile
if __name__ == '__main__':
 ap = argparse.ArgumentParser()
 ap.add_argument("-i", "--input", required=True,
                help="path to the input folder")
 ap.add_argument("-m", "--model", required=True,
                help="path to the model file")
 ap.add_argument("-s", "--store", required=True,
                help="path to the store folder")
 args = vars(ap.parse_args())

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
 use_gpu = torch.cuda.is_available()
 print (use_gpu)

 numClasses = 4
```

```
numPoints = 4
imgSize = (480, 480)
batchSize = 8 if use_gpu else 8
resume_file = str(args["model"])

provNum, alphaNum, adNum = 38, 25, 35
provinces = ["皖", "沪", "津", "渝", "冀", "晋", "蒙", "辽", "吉", "黑", "苏", "浙", "京", "闽", "赣", "鲁", "豫", "鄂", "湘", "粤", "桂",
             "琼", "川", "贵", "云", "藏", "陕", "甘", "青", "宁", "新", "警", "学", "O"]
alphabets = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
             'X', 'Y', 'Z', 'O']
ads = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
       'Y', 'Z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'O']


class wR2(nn.Module):
    def __init__(self, num_classes=1000):
        super(wR2, self).__init__()
        hidden1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=48, kernel_size=5, padding=2, stride=2),
            nn.BatchNorm2d(num_features=48),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
            nn.Dropout(0.2)
        )
        hidden2 = nn.Sequential(
            nn.Conv2d(in_channels=48, out_channels=64, kernel_size=5, padding=2),
            nn.BatchNorm2d(num_features=64),
```

```
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden3 = nn.Sequential(
    nn.Conv2d(in_channels=64, out_channels=128, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=128),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
hidden4 = nn.Sequential(
    nn.Conv2d(in_channels=128, out_channels=160, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=160),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
hidden5 = nn.Sequential(
    nn.Conv2d(in_channels=160, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
hidden6 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192, kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
```

```
        nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
        nn.Dropout(0.2)
    )
    hidden7 = nn.Sequential(
        nn.Conv2d(in_channels=192, out_channels=192, kernel_size=5, padding=2),
        nn.BatchNorm2d(num_features=192),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
        nn.Dropout(0.2)
    )
    hidden8 = nn.Sequential(
        nn.Conv2d(in_channels=192, out_channels=192, kernel_size=5, padding=2),
        nn.BatchNorm2d(num_features=192),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
        nn.Dropout(0.2)
    )
    hidden9 = nn.Sequential(
        nn.Conv2d(in_channels=192, out_channels=192, kernel_size=3, padding=1),
        nn.BatchNorm2d(num_features=192),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
        nn.Dropout(0.2)
    )
    hidden10 = nn.Sequential(
        nn.Conv2d(in_channels=192, out_channels=192, kernel_size=3, padding=1),
        nn.BatchNorm2d(num_features=192),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
```

```
            nn.Dropout(0.2)
        )
        self.features = nn.Sequential(
            hidden1,
            hidden2,
            hidden3,
            hidden4,
            hidden5,
            hidden6,
            hidden7,
            hidden8,
            hidden9,
            hidden10
        )
        self.classifier = nn.Sequential(
            nn.Linear(23232, 100),
            # nn.ReLU(inplace=True),
            nn.Linear(100, 100),
            # nn.ReLU(inplace=True),
            nn.Linear(100, num_classes),
        )

    def forward(self, x):
        x1 = self.features(x)
        x11 = x1.view(x1.size(0), -1)
        x = self.classifier(x11)
        return x
```

```
class fh02(nn.Module):
    def __init__(self, num_points, num_classes, wrPath=None):
        super(fh02, self).__init__()
        self.load_wR2(wrPath)
        self.classifier1 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
            # nn.ReLU(inplace=True),
            # nn.Dropout(),
            nn.Linear(128, provNum),
        )
        self.classifier2 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
            # nn.ReLU(inplace=True),
           # nn.Dropout(),
             nn.Linear(128, alphaNum),
        )
        self.classifier3 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
            # nn.ReLU(inplace=True),
            # nn.Dropout(),
            nn.Linear(128, adNum),
        )
        self.classifier4 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
           # nn.ReLU(inplace=True),
```

```
            # nn.Dropout(),
            nn.Linear(128, adNum),
        )
        self.classifier5 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
            # nn.ReLU(inplace=True),
            # nn.Dropout(),
            nn.Linear(128, adNum),
        )
        self.classifier6 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
           # nn.ReLU(inplace=True),
             # nn.Dropout(),
            nn.Linear(128, adNum),
        )
        self.classifier7 = nn.Sequential(
            # nn.Dropout(),
            nn.Linear(53248, 128),
            # nn.ReLU(inplace=True),
            # nn.Dropout(),
            nn.Linear(128, adNum),
        )

    def load_wR2(self, path):
        self.wR2 = wR2(numPoints)
        self.wR2 = torch.nn.DataParallel(self.wR2,
device_ids=range(torch.cuda.device_count()))
```

```
        if not path is None:
            self.wR2.load_state_dict(torch.load(path))
            # self.wR2 = self.wR2.cuda()
        # for param in self.wR2.parameters():
        #     param.requires_grad = False

    def forward(self, x):
        x0 = self.wR2.module.features[0](x)
        _x1 = self.wR2.module.features[1](x0)
        x2 = self.wR2.module.features[2](_x1)
        _x3 = self.wR2.module.features[3](x2)
        x4 = self.wR2.module.features[4](_x3)
        _x5 = self.wR2.module.features[5](x4)

        x6 = self.wR2.module.features[6](_x5)
        x7 = self.wR2.module.features[7](x6)
        x8 = self.wR2.module.features[8](x7)
        x9 = self.wR2.module.features[9](x8)
        x9 = x9.view(x9.size(0), -1)
        boxLoc = self.wR2.module.classifier(x9)

        h1, w1 = _x1.data.size()[2], _x1.data.size()[3]
        p1 = Variable(torch.FloatTensor([[w1,0,0,0],[0,h1,0,0],[0,0,w1,0],[0,0,0,h1]]).cuda(),
requires_grad=False)
        h2, w2 = _x3.data.size()[2], _x3.data.size()[3]
        p2 = Variable(torch.FloatTensor([[w2,0,0,0],[0,h2,0,0],[0,0,w2,0],[0,0,0,h2]]).cuda(),
requires_grad=False)
        h3, w3 = _x5.data.size()[2], _x5.data.size()[3]
        p3 = Variable(torch.FloatTensor([[w3,0,0,0],[0,h3,0,0],[0,0,w3,0],[0,0,0,h3]]).cuda(),
requires_grad=False)
```

```
        # x, y, w, h --> x1, y1, x2, y2
        assert boxLoc.data.size()[1] == 4
        postfix = Variable(torch.FloatTensor([[1,0,1,0],[0,1,0,1],[-0.5,0,0.5,0],[0,-
0.5,0,0.5]]).cuda(), requires_grad=False)
        boxNew = boxLoc.mm(postfix).clamp(min=0, max=1)

        # input = Variable(torch.rand(2, 1, 10, 10), requires_grad=True)
        # rois = Variable(torch.LongTensor([[0, 1, 2, 7, 8], [0, 3, 3, 8, 8], [1, 3, 3, 8, 8]]),
requires_grad=False)
        roi1 = roi_pooling_ims(_x1, boxNew.mm(p1), size=(16, 8))
        roi2 = roi_pooling_ims(_x3, boxNew.mm(p2), size=(16, 8))
        roi3 = roi_pooling_ims(_x5, boxNew.mm(p3), size=(16, 8))
        rois = torch.cat((roi1, roi2, roi3), 1)

        _rois = rois.view(rois.size(0), -1)

        y0 = self.classifier1(_rois)
        y1 = self.classifier2(_rois)
        y2 = self.classifier3(_rois)
        y3 = self.classifier4(_rois)
        y4 = self.classifier5(_rois)
        y5 = self.classifier6(_rois)
        y6 = self.classifier7(_rois)
        return boxLoc, [y0, y1, y2, y3, y4, y5, y6]


def isEqual(labelGT, labelP):
    # print (labelGT)
    # print (labelP)
```

```
    compare = [1 if int(labelGT[i]) == int(labelP[i]) else 0 for i in range(7)]
    # print(sum(compare))
    return sum(compare)


model_conv = fh02(numPoints, numClasses)
model_conv = torch.nn.DataParallel(model_conv,
device_ids=range(torch.cuda.device_count()))
model_conv.load_state_dict(torch.load(resume_file))
print("model loaded")
model_conv = model_conv.cuda()
model_conv.eval()

# efficiency evaluation
# dst = imgDataLoader([args["input"]], imgSize)
# trainloader = DataLoader(dst, batch_size=batchSize, shuffle=True, num_workers=4)
#
# start = time()
# for i, (XI) in enumerate(trainloader):
#     x = Variable(XI.cuda(0))
#     y_pred = model_conv(x)
#     outputY = y_pred.data.cpu().numpy()
#     #   assert len(outputY) == batchSize
# print("detect efficiency %s seconds" %(time() - start))


count = 0
correct = 0
error = 0
```

```
sixCorrect = 0
sFolder = str(args["store"])
sFolder = sFolder if sFolder[-1] == '/' else sFolder + '/'
if not path.isdir(sFolder):
    mkdir(sFolder)

dst = labelTestDataLoader(args["input"].split(','), imgSize)
trainloader = DataLoader(dst, batch_size=1, shuffle=True, num_workers=1)
with open('fh0Eval', 'wb') as outF:
    pass

start = time()
for i, (XI, labels, ims) in enumerate(trainloader):
    count += 1
    YI = [[int(ee) for ee in el.split('_')[:7]] for el in labels]
    if use_gpu:
        x = Variable(XI.cuda(0))
    else:
        x = Variable(XI)
    # Forward pass: Compute predicted y by passing x to the model

    fps_pred, y_pred = model_conv(x)

    outputY = [el.data.cpu().numpy().tolist() for el in y_pred]
    labelPred = [t[0].index(max(t[0])) for t in outputY]

    #   compare YI, outputY
    # try:
    if isEqual(labelPred, YI[0]) == 7:
```

```
            correct += 1
            sixCorrect += 1
        else:
            sixCorrect += 1 if isEqual(labelPred, YI[0]) == 6 else 0

    #  print(i)
        if count % 50 == 0:
            print ('total %s correct %s error %s precision %s six %s avg_time %s' % (count, correct, error, float(correct)/count, float(sixCorrect)/count, (time() - start)/count))
 with open('fh0Eval', 'a') as outF:
        outF.write('total %s correct %s error %s precision %s avg_time %s' % (count, correct, error, float(correct) / count, (time() - start)/count))
```

CHAPTER 7

MODEL COMPARISON

7.1 METRIC

As each image in CCPD contains only a single license plate (LP). Therefore, we do not consider recall and concentrate on precision. Detectors are allowed to predict only one bounding box for each image.

Detection. For each image, the detector outputs only one bounding box. The bounding box is considered to be correct if and only if its IoU with the ground truth bounding box is more than 70% (IoU > 0.7). Also, we compute AP on the test set.

Recognition. A LP recognition is correct if and only if all characters in the LP number are correctly recognized.

7.2 TensorFlow Object Detection

```
DONE (t=0.01s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.639
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.983
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.881
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.639
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.688
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.688
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.688
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.688
INFO:tensorflow:Eval metrics at step 6000
I0802 09:15:15.827595 140012604307328 model_lib_v2.py:1015] Eval metrics at step 6000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP: 0.639151
```

ACCURACY METRICS :

mAP (mean Average Precision) : 0.88

MODEL SIZE : 11MB

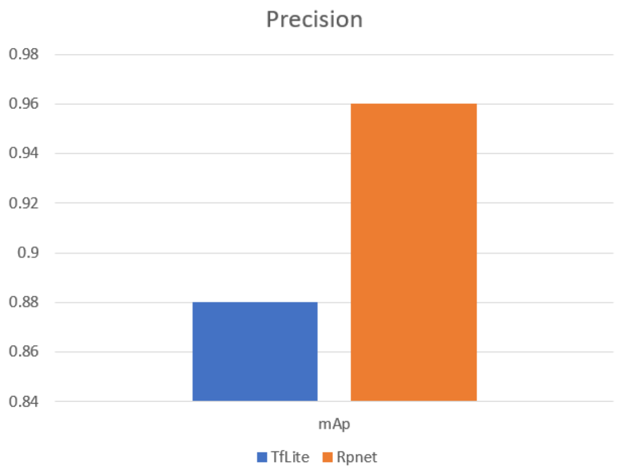
7.3 <u>RPnet</u>

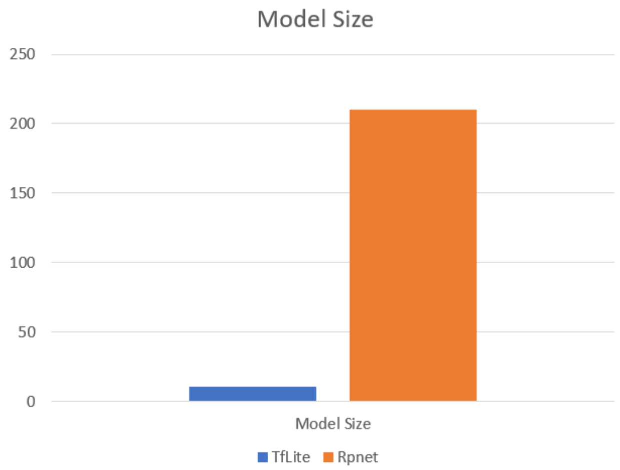
```
(rpnet) D:\CCPDCode\CCPD-master\rpnet>python rpnetEval.py -m "D:\CCPDCode\CCPD-master\rpnet\fh02.pth" -i "D:\CCPD2019\ccpd_base" -s "D:\CCPD2019\failuerDemo"
True
model loaded
total 50 correct 48 error 0 precision 0.96 six 0.96 avg_time 0.11983622074127197
total 100 correct 97 error 0 precision 0.97 six 0.98 avg_time 0.07959944725036622
total 150 correct 145 error 0 precision 0.9666666666666667 six 0.9733333333333334 avg_time 0.06607595602671305
total 200 correct 194 error 0 precision 0.97 six 0.98 avg_time 0.059457687139511106
total 250 correct 242 error 0 precision 0.968 six 0.98 avg_time 0.05546674633026123
total 300 correct 291 error 0 precision 0.97 six 0.9833333333333333 avg_time 0.05290946006774902
total 350 correct 340 error 0 precision 0.9714285714285714 six 0.9857142857142858 avg_time 0.050931384222848076
total 400 correct 389 error 0 precision 0.9725 six 0.9875 avg_time 0.049402871131896973
total 450 correct 438 error 0 precision 0.9733333333333334 six 0.9888888888888889 avg_time 0.04818950547112359
total 500 correct 484 error 0 precision 0.968 six 0.984 avg_time 0.04744087600708008
```

ACCURACY METRICS :

mAP (mean Average Precision) : 0.97

MODEL SIZE : 210MB



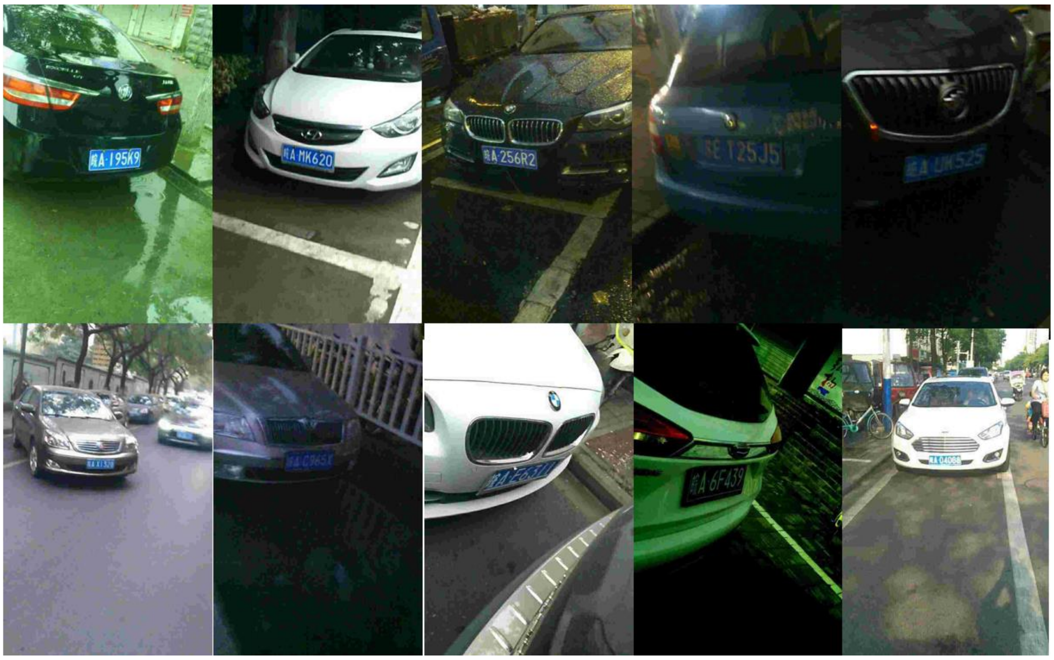


After analysing the two model and their Accuracy we selected the RPnet model due to its hight Precision and robust architecture which enables it to perform under very challenging condition.

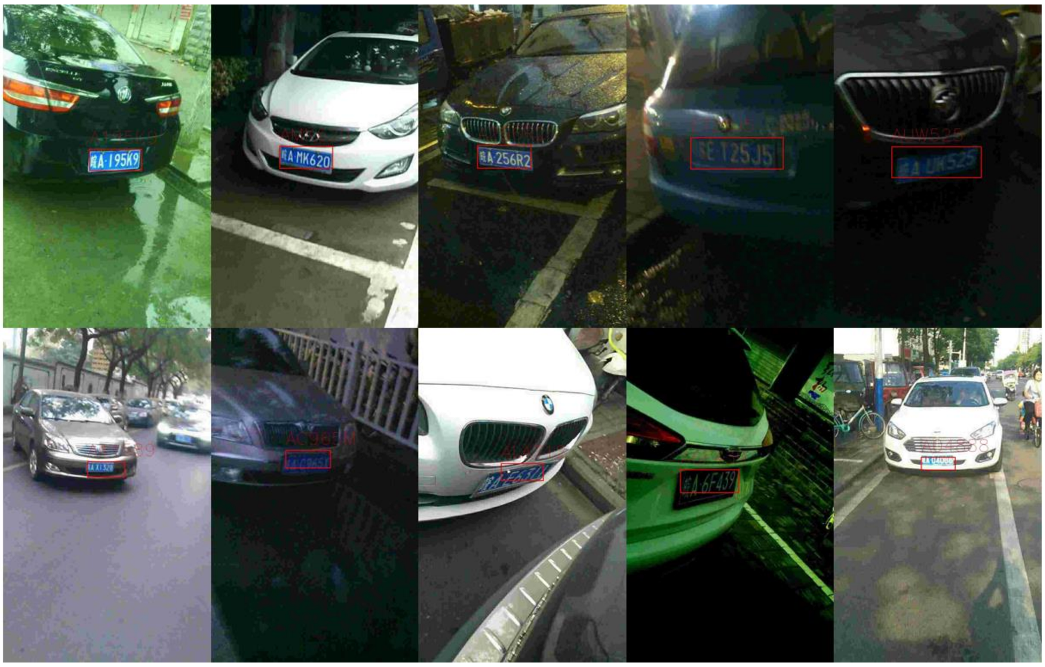
CHAPTER 8

TESTING

INPUT:



OUTPUT:



As shown in the above figure the RPnet model is performing well in very challenging condition like with low light ,tilt, and blurred situation.

CHAPTER 9

CONCLUSION

CONCLUSION

In this we present a large-scale and diverse license plate dataset named Chinese City Parking Dataset(CCPD), a network architecture named RPnet and TensorFlow Object Detection(TFOD) with EasyOcr for unified license plate detection and recognition. Images in CCPD are annotated carefully and are classified into different categories according to different features of LPs. The great data volume (250k unique images), data diversity (eight different categories), and detailed annotations make CCPD a valuable dataset for object detection, object recognition, and object segmentation. Extensive evaluations on CCPD

demonstrate our proposed RPnet outperforms state-of-the-art TFOD both in speed and accuracy.

But in the case where the license plate precision does not matter like in detecting congestion

The TfLite model can be used as it is very light weight and can detect number plates which can be taken account to count the number of vehicles on the road to identify if there is any congestion or not.

CHAPTER 10

REFERENCES/ BIBILIOGRAPHY

REFERENCES:

1. Haralick, R.M., Sternberg, S.R., Zhuang, X.: Image analysis using mathematical morphology. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-9(4) (July 1987) 532–550
2. sieh, J.W., Yu, S.H., Chen, Y.S.: Morphology-based license plate detection from complex scenes. In: Pattern Recognition, 2002. Proceedings. 16th International Conference on. Volume 3., IEEE (2002) 176–179
3. Deb, K., Jo, K.H.: Hsi color based vehicle license plate detection. In: Control, Automation and Systems, 2008. ICCAS 2008. International Conference on, IEEE (2008) 687–691
4. Yu, S., Li, B., Zhang, Q., Liu, C., Meng, M.Q.H.: A novel license plate location method based on wavelet transform and emd analysis. Pattern Recognition 48(1) (2015) 114–125
5. Saha, S., Basu, S., Nasipuri, M., Basu, D.K.: License plate localization from vehicle images: An edge based multi-stage approach. International Journal of Recent Trends in Engineering 1(1) (2009) 284–288
6. Wang, R., Sang, N., Huang, R., Wang, Y.: License plate detection using gradient information and cascade detectors. Optik-International Journal for Light and Electron Optics 125(1) (2014) 186–190
7. Lee, Y., Song, T., Ku, B., Jeon, S., Han, D.K., Ko, H.: License plate detection using local structure patterns. In: Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on, IEEE (2010) 574–579
8. Yao, Z., Yi, W.: License plate detection based on multistage information fusion. Information Fusion 18 (2014) 78–85
9. Girshick, R.: Fast r-cnn. arXiv preprint arXiv:1504.08083 (2015)
10. Laroca, R., Severo, E., Zanlorensi, L.A., Oliveira, L.S., Gon¸calves, G.R., Schwartz, W.R., Menotti, D.: A robust real-time automatic license plate recognition based on the yolo detector. arXiv preprint arXiv:1802.09567 (2018)
11. Montazzolli, S., Jung, C.: Real-time brazilian license plate detection and recognition using deep convolutional neural networks. In: 2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI). (2017) 55–62

12. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2016) 779–788
13. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European conference on computer vision, Springer (2016) 21–37
14. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European conference on computer vision, Springer (2016) 21–37
15. Redmon, J., Farhadi, A.: Yolo9000: better, faster, stronger. arXiv preprint 1612 (2016)
16. T. Vaiyapuri, S. Nandan Mohanty, M. Sivaram, I. V. Pustokhina, D. A. Pustokhin *et al.*, "Automatic vehicle license plate recognition using optimal deep learning model," *Computers, Materials & Continua*, vol. 67, no.2, pp. 1881–1897, 2021.

17. X. Wang, L. Wang, S. Li and J. Wang, “An event-driven plan TEnsi algorithm based on intuitionistic fuzzy theory,” Journal of Supercomputing, vol. 74, no. 12, pp. 6923–6938, 2018
18. S. R. Zhou and B. Tan, “Electrocardiogram soft computing using hybrid deep learning CNN-ELM,” Applied Soft Computing, vol. 86, pp. 105778–105789, 2020.
19. J. Zhang, W. Wang, C. Lu, J. Wang and A. K. Sangaiah, “Lightweight deep network for traffic sign classification,” Annals of Telecommunications, vol. 75, no. 7–8, pp. 369–379, 2020.
20. P. He, Z. Deng, C. Gao, X. Wang and J. Li, “Model approach to grammatical evolution: Deepstructured analyzing of model and representation,” Soft Computing, vol. 21, no. 18, pp. 5413– 5423, 2017.
21. R. Sun, L. Shi, C. Yin and J. Wang, “An improved method in deep packet inspection based on regular expression,” Journal of Supercomputing, vol. 75, no. 6, pp. 3317–3333, 2019.
22. C. Yin, H. Wang, X. Yin, R. Sun and J. Wang, “Improved deep packet inspection in data stream detection,” Journal of Supercomputing, vol. 75, no. 8, pp. 4295–4308, 2019.
23. O. Bulan, V. Kozitsky, P. Ramesh and M. Shreve, “Segmentation-and annotation-free license plate recognition with deep localization and failure identification,” IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 9, pp. 2351–2363, 2017.
24. Tensorflow,TensorFlow Object Dectection , https://www.tensorflow.org/lite/examples/object_detection/overview,2022.