TRAFFIC SIGN RECOGNITION USING CONVOLUTION NEURAL NETWORK WITH RTMAPS &BLUEBOX

Surya Kollazhi Manghat (2000257775)

Akash Sunil Gaikwad (2000116696)

Sarada Gajjala (0003453412)

ECE-59500 – Embedded Autonomous Systems
Electrical and Computer Engineering

Indiana University – Purdue University Indianapolis.
420 University Blvd, Indianapolis, IN 46202
Year: Spring 2018

PROJECT REPORT

Traffic Sign Recognition Using Convolution Neural Network with RTMaps & BlueBox

BY
Akash Sunil Gaikwad,
Surya Kollazhi Manghat,
Sarada Gajjala

Indiana University – Purdue University Indianapolis.
Year: Spring 2018

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signature:

Name: Akash Sunil Gaikwad (2000116696)

Signature:

Name: Surya Kollazhi Manghat (2000257775)

Signature:

Name: Sarada Gajjala (0003453412)

DATE: 05/01/2018

ACKNOWLEDGEMENT

We would like to express our gratitude and appreciation to all those who gave us the possibility to complete this report. A special thanks to our Project Guide, Professor Mohamed El-Sharkawy, whose help, stimulating suggestions and encouragement, helped us to coordinate our project especially in writing this report. given us his full effort in guiding the team in achieving the goal as well as his encouragement to maintain our progress in track.
We would also like to acknowledge with much appreciation of the crucial role of the University library and University Labs, which gave the permission to use all required resources to complete our project.

CONTENTS

ABSTRACT

The main goal of this project is to establish a neural network in RTMaps-embedded python block to identify the traffic sign board from the input images given. This project uses BlueBox 2.0 and RTMaps-Embedded as a software. The communication between RTMaps and Bluebox is set with TCP IP Connection. The main scope of this project is to identify the given input traffic sign images and print a label of the image under which category it comes under. We have categorized the images into 43 classes. Firstly, a convolution neural network is established and trained with the German traffic sign data set for about ~39000 images. Once the neural network is trained, it is then added to the RTMaps-Embedded python block. Along with that we also give the input images to be identified. So, finally the model created as RTMaps Component classifies the input traffic sign given to one of the 43 classes.

CHAPTER 1

INTRODUCTION

The main aim of this project is to establish a convolution neural network in RTMaps-Embedded Python block to identify the traffic sign board from the input images given and label the image with its corresponding name. The system is built using BlueBox 2.0 and the software used is RTMaps-Embedded version 4.5. Setup connection with Bluebox and RTMaps using TCP IP protocol. So, firstly the neural network has to be established and then implemented on the RTMaps python to run the system. The neural network is trained with some data and then fed to the RTMaps python block. We used German traffic sign dataset of ~39000 images for training. The images are categorized into 43 classes. Once the neural network is trained with data, it is then added to the python block. Then the RTMaps executes the python block on the set of 10 input images which will be reading from a folder classifies the input traffic sign to one of the 43 classes that are defined previously.

CHAPTER 2

REQUIREMENTS FOR THE PROJECT

2.1. Hardware

1. BlueBox 2.0

The NXP BlueBox is a development platform series that provides the required performance, functional safety and automotive reliability for engineers to develop self-driving cars. The latest addition to the series, the BLBX2-xx family, incorporates the S32V234 automotive vision and sensor fusion processor, the LS2084A embedded computer processor and the S32R27 radar microcontroller. The LS2 provides high single and multithreaded performance with eight ARM A72 cores. The S32V features a massively parallel Vision accelerator known as the APEX2, which is well suited for image processing algorithms and light machine learning inference work. The S32R is offered as an ASIL-D supervisor that complements the ASIL-B ready S32V

- LS2: Layerscape processor, Ethernet aggregation and high end ARM core complex (8 x A72 ARMv8 cores)
- S32V: Vision processor for Automotive (vision pipeline (APEX) + 4 x ARM A53 cores)
- S32R: Safety processor, ASIL-D capable

Operating System: Auto SDK Linux + Ubuntu RFS.

At a high-level, the BLBX2 consists of three major components integrated together to provide a powerful, yet flexible, and safe development platform for automotive and industrial applications. The LS2 provides high single and multithreaded performance with eight ARM A72 cores.The S32V features a massively parallel Vision accelerator known as the APEX2, which is well suited for image processing algorithms and light machine learning inference work. The S32R is offered as an ASIL-D supervisor that complements the ASIL-B ready S32V.

Features:

- Different enclosure styles possible for lab and in-vehicle use
- 12 V /24 V vehicle compatible power input
- High performance compute with 16 GB DDR4 and 256 GB SSD
- ASIL-B compute, automotive interfaces with vision acceleration
- ASIL-D subsystem, with dedicated interfaces
- Automotive I/O, numerous interfaces
- Ethernet 100M/1G/10Gbps, SFP+, 8 x 100BASE-T1, CAN-FD, FlexRay™, 8 x cameras

2.2. Software

1. RTMaps-Embedded

Asynchronous Real-time Multisensor Processing Framework is more intuitive to operate and easier to program in C++, Python, Simulink. Off-the-shelf component libraries for sensors and data processing, e.g., for OpenCV. A continuous transition from prototyping on PCs to production close implementations on embedded ARM platforms. A modular toolkit for your multimodal applications like ADAS, Autonomous vehicles, Robotics, UGVs, UAVs, HMI, data logging etc. The easiest way to develop, test, validate, benchmark and execute your applications.

- CAPTURE
- RECORD
- PLAYBACK, DEVELOP, TEST AND VALIDATE
- DEPLOY

RTMaps 4 is an asynchronous high performance platform designed to face and win multisensor challenges and to allow engineers and researchers to take an advantage of an efficient and easy-to-use framework for fast and robust developments.

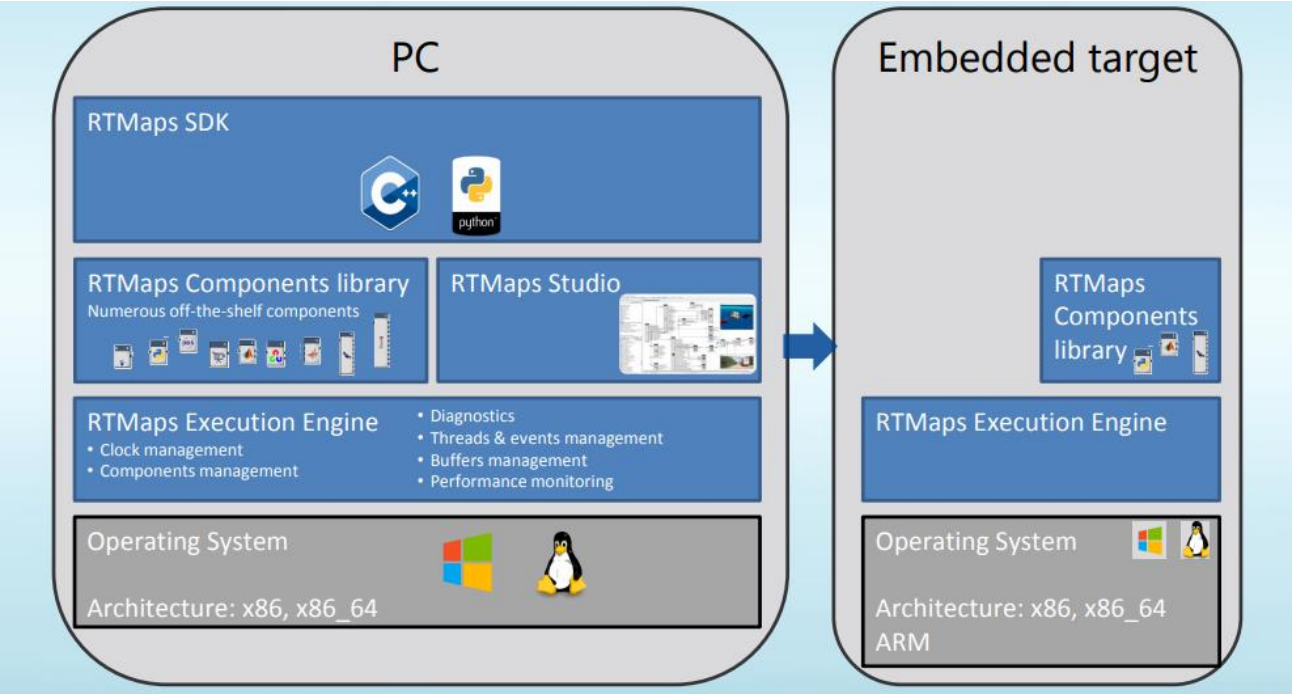
SOFTWARE ARCHITECTURE

RTMaps has been developed with few objectives in mind: performance, ease of use and modularity. Therefore it is made of several independent modules to be used in different context: RTMAPS RUNTIME THE ENGINE: this is the core of any rtmaps-based application.

THE RTMAPS STUDIO: the studio is the graphical design tool. Assemble components, configure them, fine-tune your application, assess its performance, with an easy-to-use graphical interface.

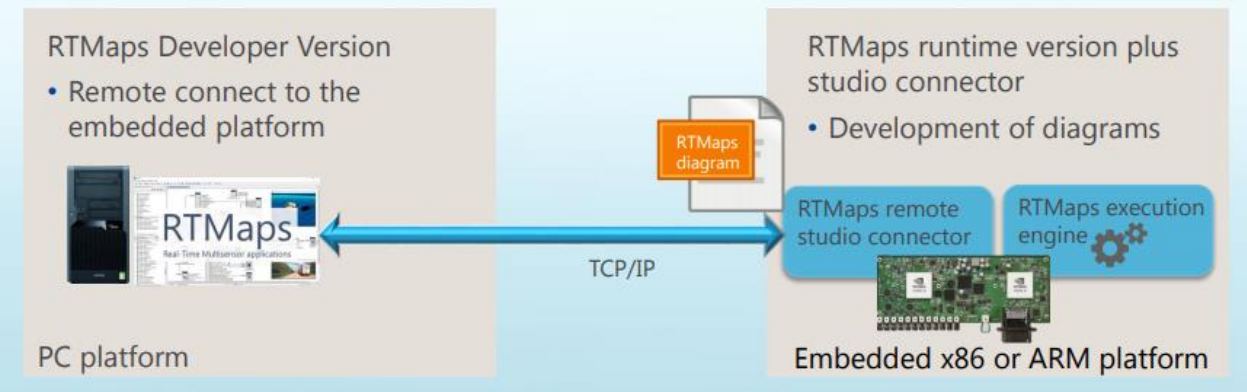
RTMAPS COMPONENTS LIBRARY: a large set of on-the-shelf components, to be used in a few clicks (sensors connectivity, pre-processing, communication, display, recording / playback, etc.

THE RTMAPS SDK: develop your own components and complement the components library at will integrate your code in C/C++ or python.



RTMaps Remote Studio Connector:

- Use RTMaps Developer Version on your PC to cross-develop algorithms and collect data directly on embedded platform
- Use RTMaps remote studio connector to remote control RTMaps on embedded platform
- Use RTMaps Run-Time Version on embedded target platform to execute RTMaps diagram disconnected from Developer PC

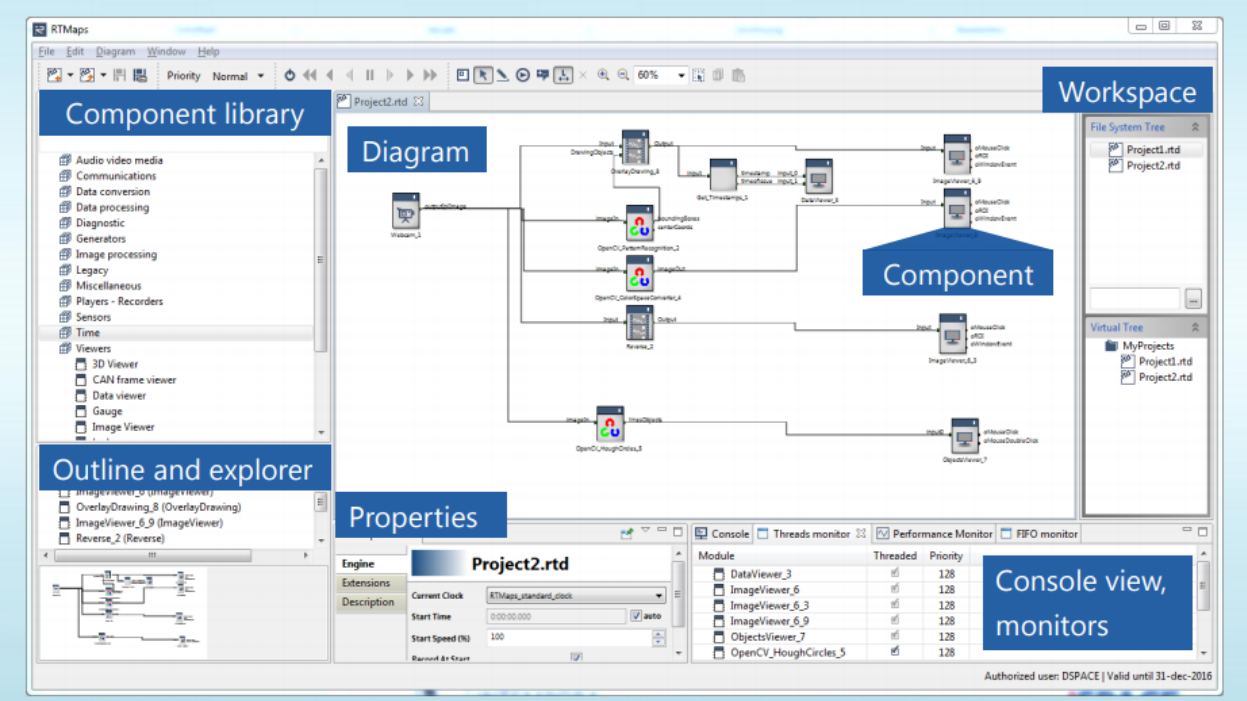


THE RTMAPS RUNTIME ENGINE:

The RTMaps Runtime Engine is the core of any RTMaps-based applications. It takes care of all the base services: components registration, connections, buffers management, time stamping, threading and priorities.

- Lightweight
- Highly optimized: no memory copies, just pointers exchange. No dynamic memory allocation during execution, event-based, polling or synchronous components communication.
- Multithreaded: your applications natively take advantage of multi-core architectures
- Easily deployable
- Embeddable in third-party applications

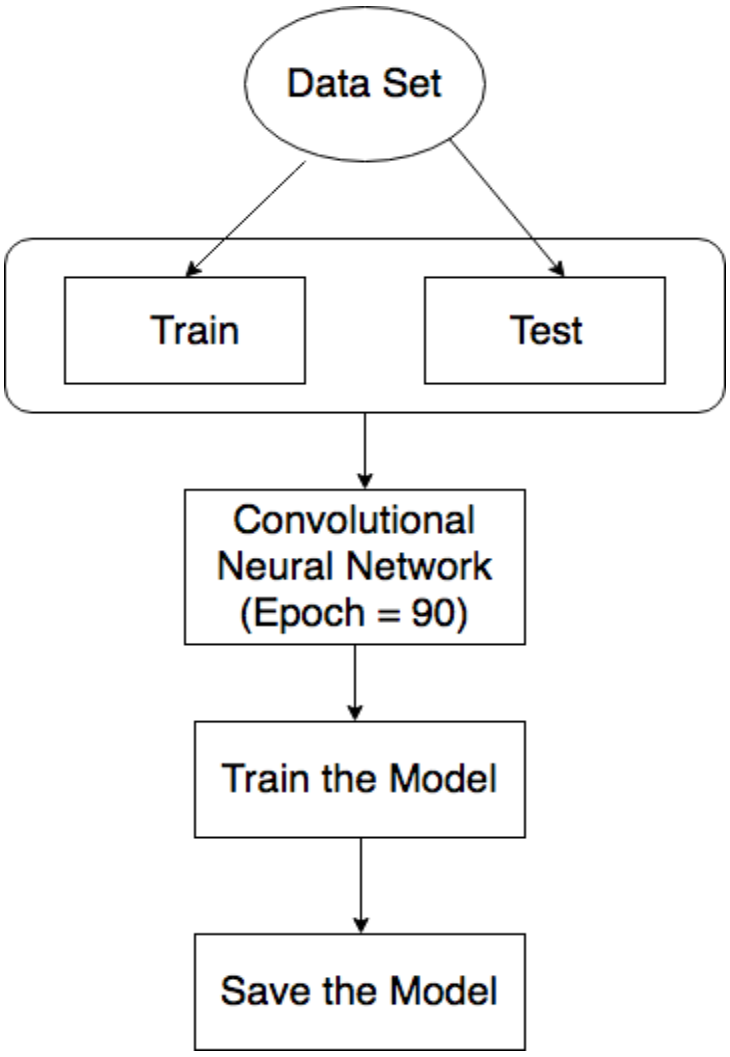
Views / Editors:

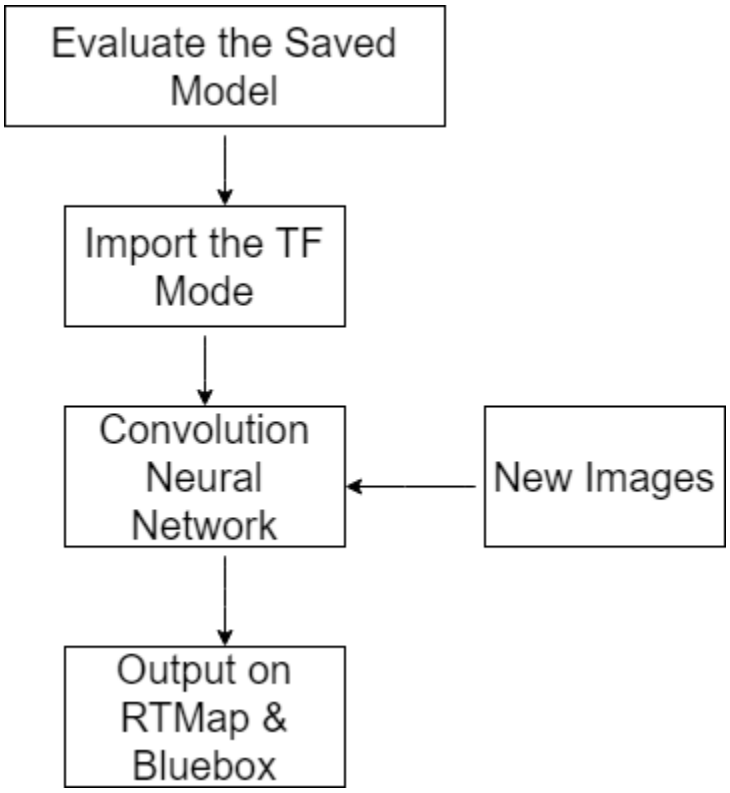


CHAPTER 3

IMPLEMENTATION

1. Flow Charts



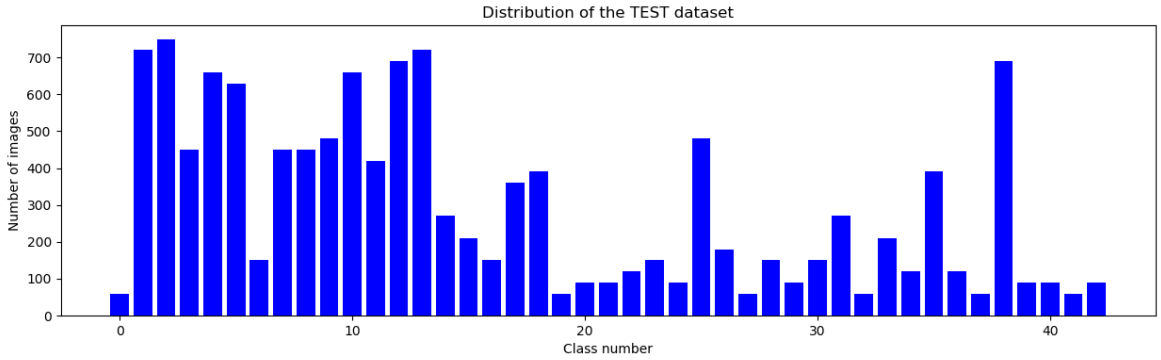


TRAINING THE CONVOLUTIONAL NEURAL NETWORK:

- Data preparation
- Data Augmentation
- Data Processing
- Convolution Neural Network Architecture
- Save TensorFlow Model
- Evaluation of the Model

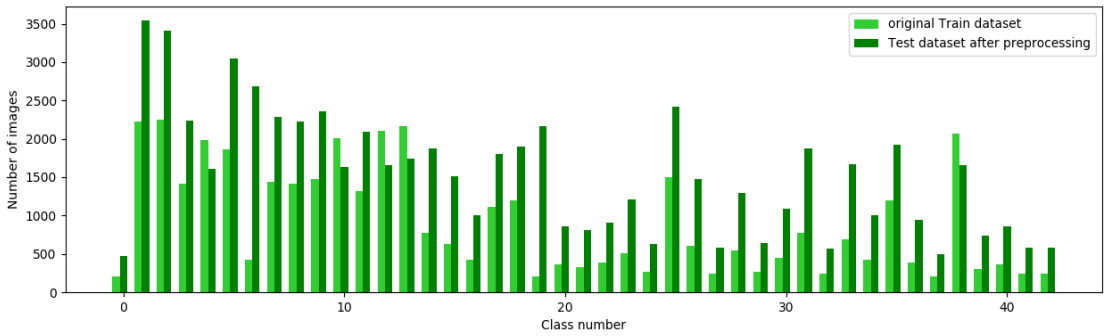
DATA PREPARATION:

- Divide the dataset in to into training and testing dataset. (German Traffic Signs)
- Create a Pickle Object using Training images and labels



DATA AUGMENTATION:

- Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips etc.
- Increase the number of samples of the training dataset

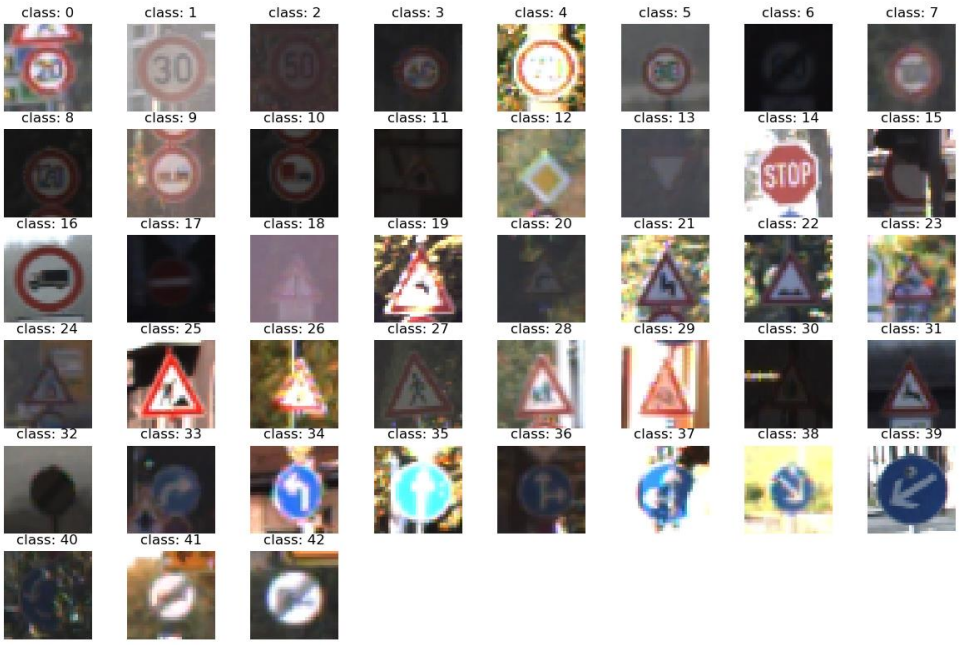


DATA PROCESSING:

- Transform the Images to Grayscale format.
- Transform the images to proper format (ex. 64x64)
- Normalize the Images

```
(keras_tf) akash@akash-GL553VE:~/Desktop/595_Final_Proj

Image Shape: (32, 32, 3)

Training Set:   39209 samples
Test Set:       12630 samples

Number of training examples = 39209
Number of testing examples = 12630

Image data shape = (32, 32, 3)
Number of classes = 43
Images for Train dataset (before enlargement process):
Minimum for class = 210
Mean for class = 912
Maximum for class = 2250
Images for Test dataset:
Minimum for class = 60
Mean for class = 294
```

IMAGES AFTER DATA PROCESSING:



IMAGES AFTER DATA PROCESSING:

#_____Layer 1_____

 # Convolutional. Input = 32x32x1. Filter = 5x5x1. Output = 28x28x6.

 # ReLu activation function

#_____Layer 2_____

 # Convolutional. Input = 28x28x6. Filter = 3x3x6. Output = 14x14x6.

 # Pooling. Input = 28x28x6. Output = 14x14x6.

#_____Layer 3_____#

 # Convolutional. Input = 14x14x6. Filter = 5x5x12. Output = 10x10x16.

 # ReLu activation function

 # Pooling. Input = 10x10x16. Output = 5x5x16

#_____Layer 4_____#

 # Flatten. Input = 5x5x16. Output = 400.

 # Fully Connected. Input = 400. Output = 120.

 # ReLu activation function

#_____Layer 5_____#

 # Fully Connected. Input = 120. Output = 84.

 # ReLu activation function

#_____Layer 6_____#

 # Fully Connected. Input = 84. Output = 43

TENSOR FLOW MODEL

Save the TensorFlow model as following files:

my_test_model-1000.index

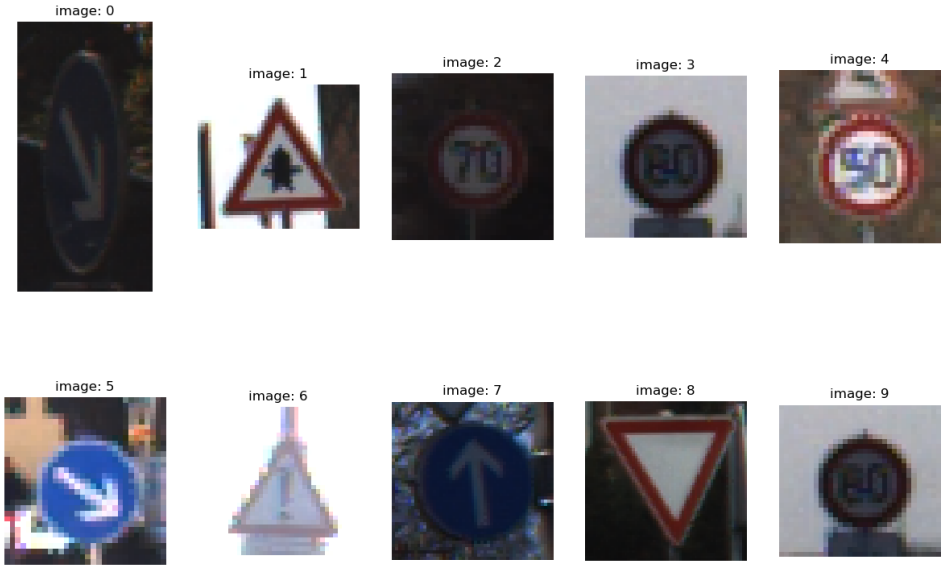
my_test_model-1000.meta

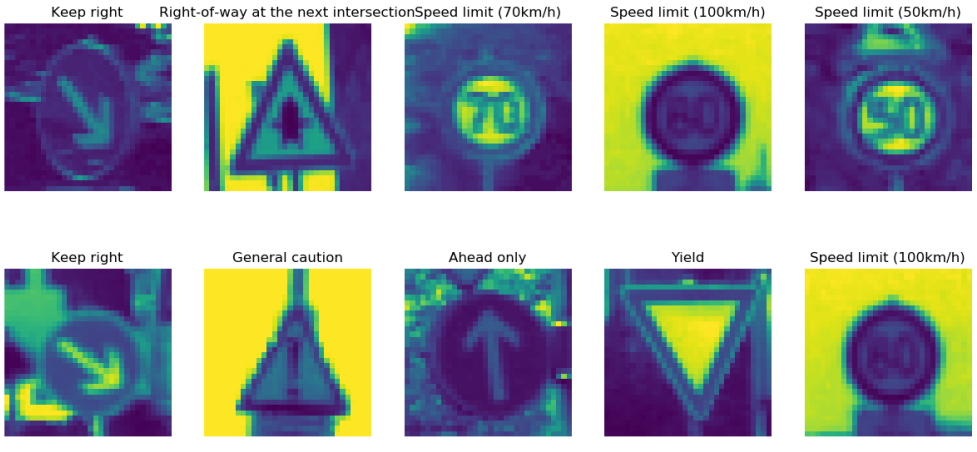
my_test_model-1000.data-00000-of-00001

checkpoint

EVOLUTION OF THE MODEL:

- Evaluate the CNN by passing New Images to Model.
- Pass New image to feed Forward Network and Get the probability of class



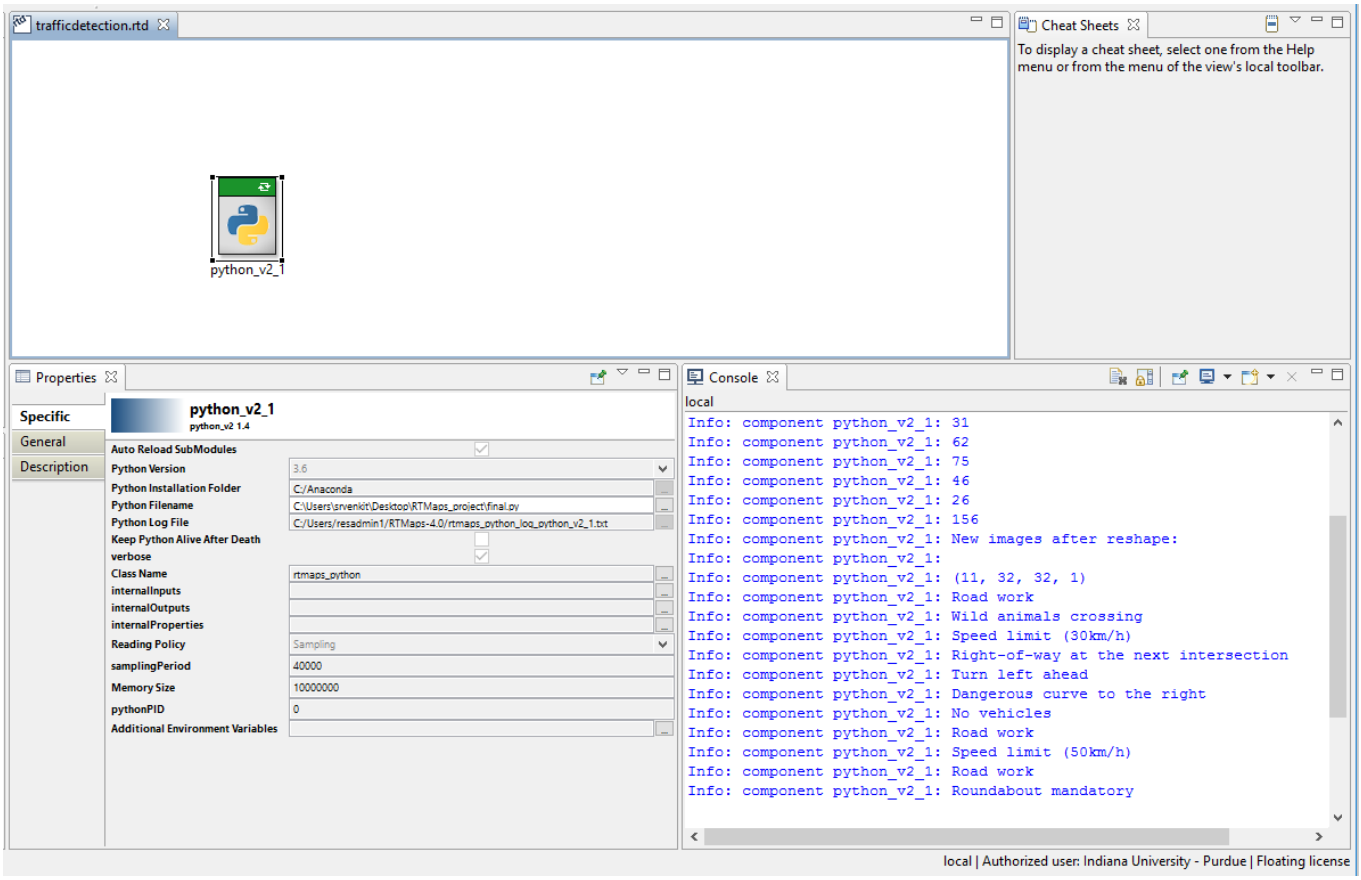


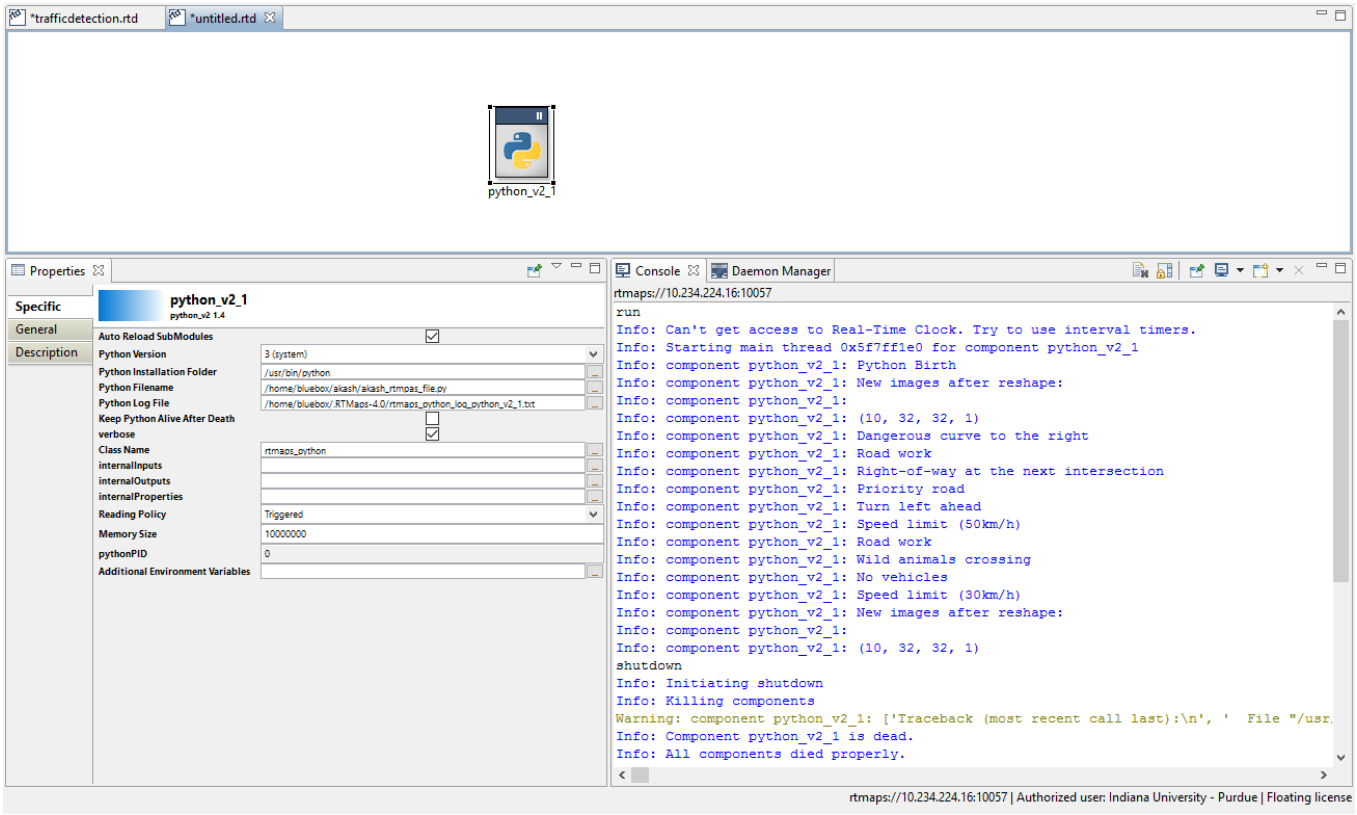
IMPLEMENTATION OF THE MODEL

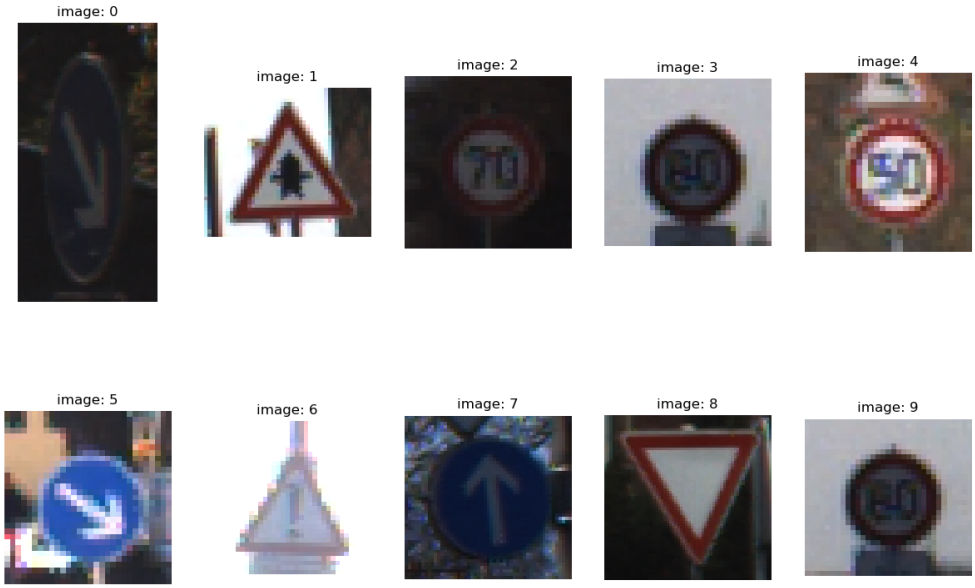
- Implementation of CNN on PC (Ubuntu 16.04 LTS)
- Implementation of CNN on RTMaps studio (Simulation)
- Implementation of CNN on RTMaps runtime engine and Bluebox. (Hardware)

CHAPTER 4

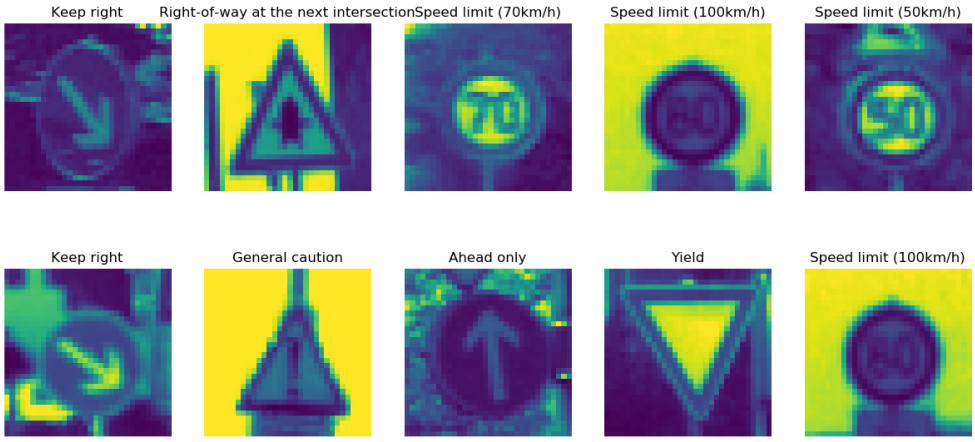
RESULT & CONCLUSION




*trafficdetection.rtd
*untitled.rtd
python_v2_1
Properties
Specific
General
Description
python_v2_1
python_v2 1.4
Auto Reload SubModules
Python Version
3 (system)
Python Installation Folder
/usr/bin/python
Python Filename
/home/bluebox/akash/akash_rtmpas_file.py
Python Log File
/home/bluebox/.RTMaps-4.0/rtmaps_python_log_python_v2_1.txt
Keep Python Alive After Death
verbose
Class Name
rtmaps_python
internalInputs
internalOutputs
internalProperties
Reading Policy
Triggered
Memory Size
10000000
pythonPID
0
Additional Environment Variables
Console
Daemon Manager
rtmaps://10.234.224.16:10057
run
Info: Can't get access to Real-Time Clock. Try to use interval timers.
Info: Starting main thread 0x5f7ff1e0 for component python_v2_1
Info: component python_v2_1: Python Birth
Info: component python_v2_1: New images after reshape:
Info: component python_v2_1:
Info: component python_v2_1: (10, 32, 32, 1)
Info: component python_v2_1: Dangerous curve to the right
Info: component python_v2_1: Road work
Info: component python_v2_1: Right-of-way at the next intersection
Info: component python_v2_1: Priority road
Info: component python_v2_1: Turn left ahead
Info: component python_v2_1: Speed limit (50km/h)
Info: component python_v2_1: Road work
Info: component python_v2_1: Wild animals crossing
Info: component python_v2_1: No vehicles
Info: component python_v2_1: Speed limit (30km/h)
Info: component python_v2_1: New images after reshape:
Info: component python_v2_1:
Info: component python_v2_1: (10, 32, 32, 1)
shutdown
Info: Initiating shutdown
Info: Killing components
Warning: component python_v2_1: ['Traceback (most recent call last):\n', ' File "/usr
Info: Component python_v2_1 is dead.
Info: All components died properly.
rtmaps://10.234.224.16:10057 | Authorized user: Indiana University - Purdue | Floating license



Input Images for Training



Output Images with Labels in Training

CONCLUSION:

The Convolution neural network is trained with the around 39,000 German traffic data set. After the training, it is then added to the python block of RTMaps-Embedded. The images are categorized to 43 classes. Once the model is created as RTMaps component, it classified the input traffic data sign to one of the 43classes.

CHAPTER 5

REFERENCES

1. https://intempora.com/products/rtmaps#the-rtmaps-runtime-engine
2. https://www.dspace.com/en/inc/home/products/sw/impsw/rtmaps.cfm
3. https://community.nxp.com/docs/DOC-335045