AUTOMATIC TRAFFIC SIGN DETECTION SYSTEM

B.Tech. Major Project-1 Report

Submitted in partial fulfilment of the requirement for the award of degree
of
Bachelor of Technology
in

COMPUTER SCIENCE AND ENGINEERING

Submitted By-	**Submitted To-**
Goutham Reddy (IIITU14126)	**Dr. T. P. Sharma**
Ankita Mehra (IIITU14104)	**Associate Professor**
Arvind (IIITU14109)	**NIT Hamirpur**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
HAMIRPUR-177005, HP (INDIA)

DECEMBER, 2017

CERTIFICATE

I hereby certify that the work which is being presented in the B.Tech. Major Project Report entitled **“AUTOMATIC TRAFFIC SIGN DETECTION SYSTEM”,** in fulfilment of the requirements for the award of the **Bachelor of Technology in Computer Science and Engineering** and submitted to the Department of Computer Science and Engineering of National Institute of Technology Hamirpur HP is an authentic record of our own work carried out during a period from July 2017 to December 2017 under the supervision of **Dr. T.P. Sharma, Computer Science and Engineering Department**.

The matter presented in this thesis has not been submitted by me for the award of any other degree elsewhere.

Signature of Candidates

Goutham Reddy (IIITU14126)

Ankita Mehra (IIITU14104)

Arvind (IIITU14109)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Signature of Supervisor

Date:

Dr. T.P. Sharma,
Associate Professor,
NITH

ACKNOWLEDGEMENT

Completion of this Project-I dissertation was possible with the support of several people. We would like to express our sincere gratitude to all of them. First of all, we are extremely grateful to our project mentor Dr. T.P. Sharma, Associate Professor, CSED, National Institute of Technology Hamirpur, H.P. for his valuable guidance, scholarly inputs and consistent encouragement that we received throughout the project work.

This feat was possible only because of the unconditional support provided by him with an amicable and positive disposition. He has always made himself available to clarify our doubts despite his busy schedule and we consider it as a great opportunity to do our Project-I course under his guidance and to learn from his research expertise. We want to thank him for all the help and support he has provided us throughout the course of the project

We would also like to thank him for the academic support and the facilities provided to carry out the research work at the Institute. As a teacher and mentor, he has taught us more than we could ever give him credit for here. Nobody has been more important to us in the pursuit of this project than the members of our families. We would like to thank our parents, whose love and guidance is with us in whatever we pursue. Last but not the least we want to thank our mates whose words of encouragement and criticism have moulded this project in its present form.

Goutham Reddy (IIITU14126)

Ankita Mehra (IIITU14104)

Arvind (IIITU14109)

CONTENTS

Figure Index

Table Index

ABSTRACT

Traffic signs are integral part of our road infrastructure. They provide information to road users so that they can adjust their driving behaviour. This is to make sure that the drivers cling to the rules and regulations of driving on road. Without useful signs, we are likely be faced with more accidents as drivers will not get proper assistance about the road works, sharp turn etc. Now a days around 1.3M people die on roads each year. The autonomous vehicles will decrease road accidents considerably by recognizing and understanding the road signs and taking actions accordingly. The autonomous bot that we have made uses Convolutional Neural Networks (ConvNets) for traffic sign classification, computer vision method (using android) for cropping the image. The cropped image is then sent to the server (laptop) and the sever replies to the bot after doing certain computations. The bot works using Raspberry Pi which is connected to laptop that is used as a local server. This servers takes the images from bot (which is taken by the mobile phone that is connected to laptop through wifi-hotspot) and processes them using deep learning and gives the command to the bot as per the sign detected. This system yields an accuracy of about 90+%. Traditional standards to detect and classify traffic signs require considerable time-consuming manual work to handcraft important features in images. Instead, by using deep learning for this problem we have created a model that reliably classifies traffic signs with less requirement for manual work.

1. Introduction

The last few decades have seen a tremendous acceleration in the adoption of deep learning algorithms across an increasingly broad range of applications, many of which assist and simplify our everyday tasks. TRAFFIC sign recognition has direct real-world applications such as driver assistance and safety, urban scene understanding, automated driving, or even sign monitoring for maintenance. It is a relatively constrained problem in the sense that signs are unique, rigid and intended to be clearly visible for drivers, and have little variability in appearance. In particular, with recent developments of general-purpose computing on graphics processing units (GPGPU) and the availability of large open data sets, training artificial neural networks (ANNs or NNs) with many hidden layers has become feasible. This approach to machine learning, now known as deep learning, has revolutionized research in computer vision, speech recognition and natural language processing, and is now rapidly being adopted in a large number of applications across a wide range of industrial sectors. Major technology companies such as Google, Microsoft, Facebook, Yahoo!, and IBM are currently re-thinking some of their core products and services to include deep learning based Solutions. At the same time, hardware producers such as NVidia, Mobileye, Altera, Qualcomm, and Samsung have started research and development projects with the aim of efficiently implementing deep neural networks on chips or field-programmable gate arrays (FPGAs), in order to deploy state-of-the-art vision systems in smartphones, autonomous vehicles, and robots. In the automotive industry, machine learning algorithms are playing an important role in the development of advanced driver assistance systems (ADAS) for improving the driver's safety and comfort. Deep learning is now considered by many automotive companies as one of the most promising emerging technologies, not only in the improvement of ADAS, but also in the broader context of autonomous vehicles, bound to revolutionize the entire automotive sector in the near future. In particular, systems for automatic traffic sign recognition, such as Volvo's Road Sign Information (RSI), or Toyota's Road Sign Assist (RSA), have already been on the market for a few years and are key components of modern ADASs. Improving the accuracy of both the detection and classification of traffic sign would increase the effectiveness of current systems, and potentially play an important role in the development of future auto-pilot computer systems. There is therefore a strong interest among vehicle manufacturers to leverage recent developments in deep learning, namely deep Convolutional Neural Networks (CNNs, or ConvNets), which have achieved state-of-the-art performance in a wide range of computer vision tasks such as image classification, localization, and detection

The dataset provided by the GTSRB(German Traffic Sign Recognition Benchmark) presents a number of difficult challenges due to real-world variability's such as viewpoint variations, lighting conditions (saturations, low-contrast), motion-blur, occlusions, sun glare, physical damage, colours fading, graffiti, stickers. Although signs are available as video sequences in the training set, temporal information is not in the test set. The present project aims to build a robust recognizer with good accuracy.

2. Components

It has basically three main parts
a) Hardware
b) Networking
c) Software
The below is the components list that is needed for the project.

Table.1.Components List

Hardware	**Networking**	**Software**
Raspberry Pi3	Routers for Access Point	Caffe
DC Motors		Shell scripting, PHP
IP camera		OpenCV, Apache

2.1. Raspberry Pi

The **Raspberry Pi** is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The Raspberry Pi 3 is the third-generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016.

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A

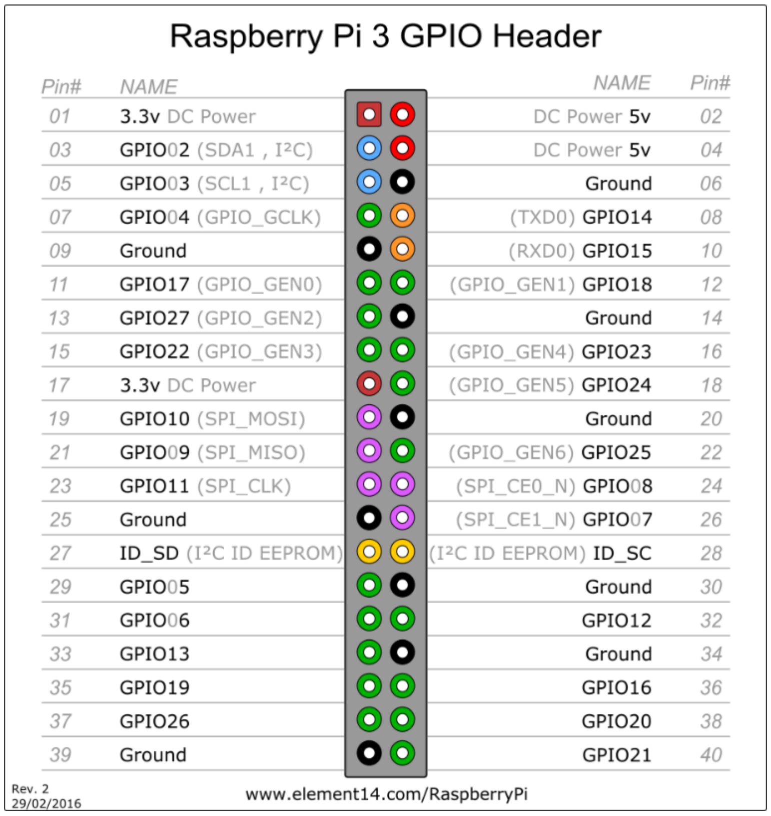
2.1.1 Processor

The Broadcom BCM2835 SoC used in the first generation Raspberry Pi is somewhat equivalent to the chip used in first modern generation smartphones (its CPU is an older ARMv6 architecture), which includes a 700 MHz ARM1176JZF-S processor, VideoCore IV graphics processing unit (GPU),and RAM. It has a level 1 (L1) cache of 16 KB and a level 2 (L2) cache of 128 KB. The level 2 cache is used primarily by the GPU. The SoC is stacked underneath the RAM chip, so only its edge is visible. The earlier models of Raspberry Pi 2 use a Broadcom BCM2836 SoC with a 900 MHz 32-bit quad-core ARM Cortex-A7 processor, with 256 KB shared L2 cache. The Raspberry Pi 2 V1.2 was upgraded to a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, the same SoC which is used on the Raspberry Pi 3. The Raspberry Pi 3 uses a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, with 512 KB shared L2 cache.

2.1.2. Performance

The Raspberry Pi 3, with a quad-core Cortex-A53 processor, is described as 10 times the performance of a Raspberry Pi 1. This was suggested to be highly dependent upon task threading and instruction set use. Benchmarks showed the Raspberry Pi 3 to be approximately 80% faster than the Raspberry Pi 2 in parallelized tasks.

Raspberry Pi 2 includes a quad-core Cortex-A7 CPU running at 900 MHz and 1 GB RAM. It is described as 4–6 times more powerful than its predecessor. The GPU is identical to the original. In parallelized benchmarks, the Raspberry Pi 2 could be up to 14 times faster than a Raspberry Pi 1 Model B+. While operating at 700 MHz by default, the first generation Raspberry Pi provided a real-world performance roughly equivalent to 0.041 GFLOPS. On the CPU level the performance is similar to a 300 MHz Pentium II of 1997–99. The GPU provides 1 Gpixel/s or 1.5 Gtexel/s of graphics processing or 24 GFLOPS of general purpose computing performance. The graphical capabilities of the Raspberry Pi are roughly equivalent to the performance of the Xbox of 2001.



Raspberry Pi 3 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

Fig.1.Raspberry pi3 pin configuration

2.2. DC Motor

A **DC motor** is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor.

A geared DC Motor has a gear assembly attached to the motor. The speed of motor is counted in terms of rotations of the shaft per minute and is termed as RPM .The gear assembly helps in increasing the torque and reducing the speed. Using the correct combination of gears in a gear motor, its speed can be reduced to any desirable figure. This concept where gears reduce the speed of the vehicle but increase its torque is known as gear reduction. This Insight will explore all the minor and major details that make the gear head and hence the working of geared DC motor.

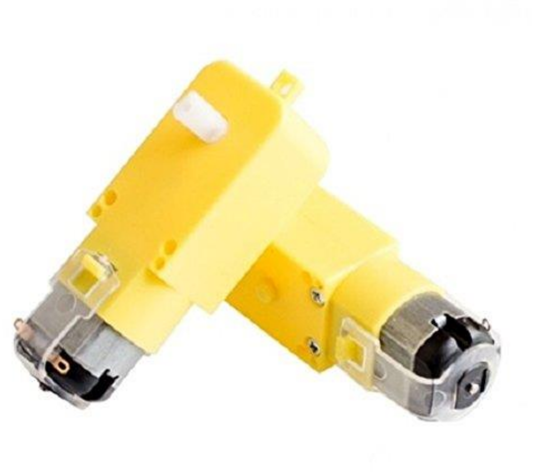


Fig.2. DC motors

Table.2.Specification of DC Motor

Voltage	3-12V Dc
No Load Speed	200 rpm
Shaft length	9mm
Torque	500 gf-cm
Shaft Diameter	5.3x3.7mm

2.3. IP camera

IP camera, is a type of digital video camera commonly employed for surveillance. An IP camera is typically either centralized (requiring a central network video recorder (NVR) to handle the recording, video and alarm management) or decentralized. IP cameras differ from previous generation analog cameras which transmitted video signals as a voltage, whereas IP camera images are sent using the transmission and security features of the TCP/IP protocol.

Some advantages to this approach include:

- Two-way audio via a single network cable allows users to listen to and speak to the subject of the video (e.g. gas station clerk assisting a customer on how to use the pay pumps)
- The use of a Wi-Fi or wireless network.
- Secure data transmission through encryption and authentication methods such as WPA, WPA2, TKIP, and AES.
- Remote accessibility allowing live video to be viewed from any device with sufficient access privileges

2.4. Caffe

Caffe is a deep learning framework made with expression, speed and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors.

Expressive architecture encourages application and innovation. Models and optimization are defined by configuration without hard-coding. Switch between CPU and GPU by setting a single flag to train on a GPU machine then deploy to commodity clusters or mobile devices. **Extensible code** fosters active development. In Caffe's first year, it has been forked by over 1,000 developers and had many significant changes contributed back. Thanks to these contributors the framework tracks the state-of-the-art in both code and models. **Speed** makes Caffe perfect for research experiments and industry deployment. Caffe can process **over 60M images per day** with a single NVIDIA K40 GPU. That's 1 ms/image for inference and 4 ms/image for learning and more recent library versions and hardware are faster still. We believe that Caffe is among the fastest ConvNet implementations available.

2.5. OPENCV

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous computer platform.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equip-ment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

2.6. Fast Android Networking library

For interconnectivity between server and bot we use fast android networking library. Android lets your application connect to the internet or any other local network and allows you to perform network operations. A device can have various types of network connections. Android contains the standard Java network java.net package which can be used to access network resources. The base class for HTTP network access in the java.net package is the HttpURLConnection class. Performing network operations with standard Java API can be cumbersome. To simplify these operations several popular Open Source libraries are available. The most popular ones are the following:

- OkHttp for efficient HTTP access
- Retrofit for REST based clients
- Glide for image processing

Web Sockets are a standard based on HTTP for asynchronous message-based communication between a client and a server. To start a web socket communication, you create a HTTP GET request with a special HTTP headers. If the server accepts this request, the client and the server can send each other messages. Messages can be text or binary data and should be relatively small, as the web socket protocol is intended to be used with small payloads in the data. It is good practice to use JSON as data format for the messages.

2.7. Android Layout

The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The **View Group** is a subclass of **View** and provides invisible container that hold other Views or other View Groups and define their layout properties. At third level we have different layouts which are subclasses of View Group class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/View Group** objects or you can declare your layout using simple XML file **main_layout.xml** which is located in the res/layout folder of your project.

2.7.1. Android Intent

An intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and startService(Intent) or bindService(Intent, ServiceConnection, int) to communicate with a background Service. An Intent provides a facility for performing late runtime binding between the codes in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

There are two primary forms of intents:-

- **Explicit Intents** have specified a component (via setComponent or setClass), which provides the exact class to be run. Often these will not include any other information, simply being a way for an application to launch various internal activities it has as the user interacts with the application.
- **Implicit Intents** have not specified a component; instead, they must include enough information for the system to determine which of the available components is best to run for that intent.

When using implicit intents, given such an arbitrary intent we need to know what to do with it. This is handled by the process of Intent resolution, which maps an Intent to an Activity, BroadcastReceiver, or Service (or sometimes two or more activities/receivers) that can handle it. The intent resolution mechanism basically revolves around matching an Intent against all of the <intent-filter> descriptions in the installed application packages. (Plus, in the case of broadcasts, any BroadcastReceiver objects explicitly registered with registerReceiver(BroadcastReceiver, IntentFilter).) More details on this can be found in the documentation on the IntentFilter class.

There are three pieces of information in the Intent that are used for resolution: the action, type, and category. Using this information, a query is done on the PackageManager for a component that can handle the intent. The appropriate component is determined based on the intent information supplied in the AndroidManifest.xml file as follows:

- The **action**, if given, must be listed by the component as one it handles.
- The **type** is retrieved from the Intent's data, if not already supplied in the Intent. Like the action, if a type is included in the intent (either explicitly or implicitly in its data), then this must be listed by the component as one it handles.
- For data that is not a content: URI and where no explicit type is included in the Intent, instead the **scheme** of the intent data (such as http: or mailto:) is considered. Again like the action, if we are matching a scheme it must be listed by the component as one it can handle.
- The **categories**, if supplied, must *all* be listed by the activity as categories it handles. That is, if you include the categories CATEGORY_LAUNCHER and CATEGORY_ALTERNATIVE, then you will only resolve to components with an intent that lists both of those categories. Activities will very often need to support the CATEGORY_DEFAULT so that they can be found by Context.startActivity().

3. Methodology

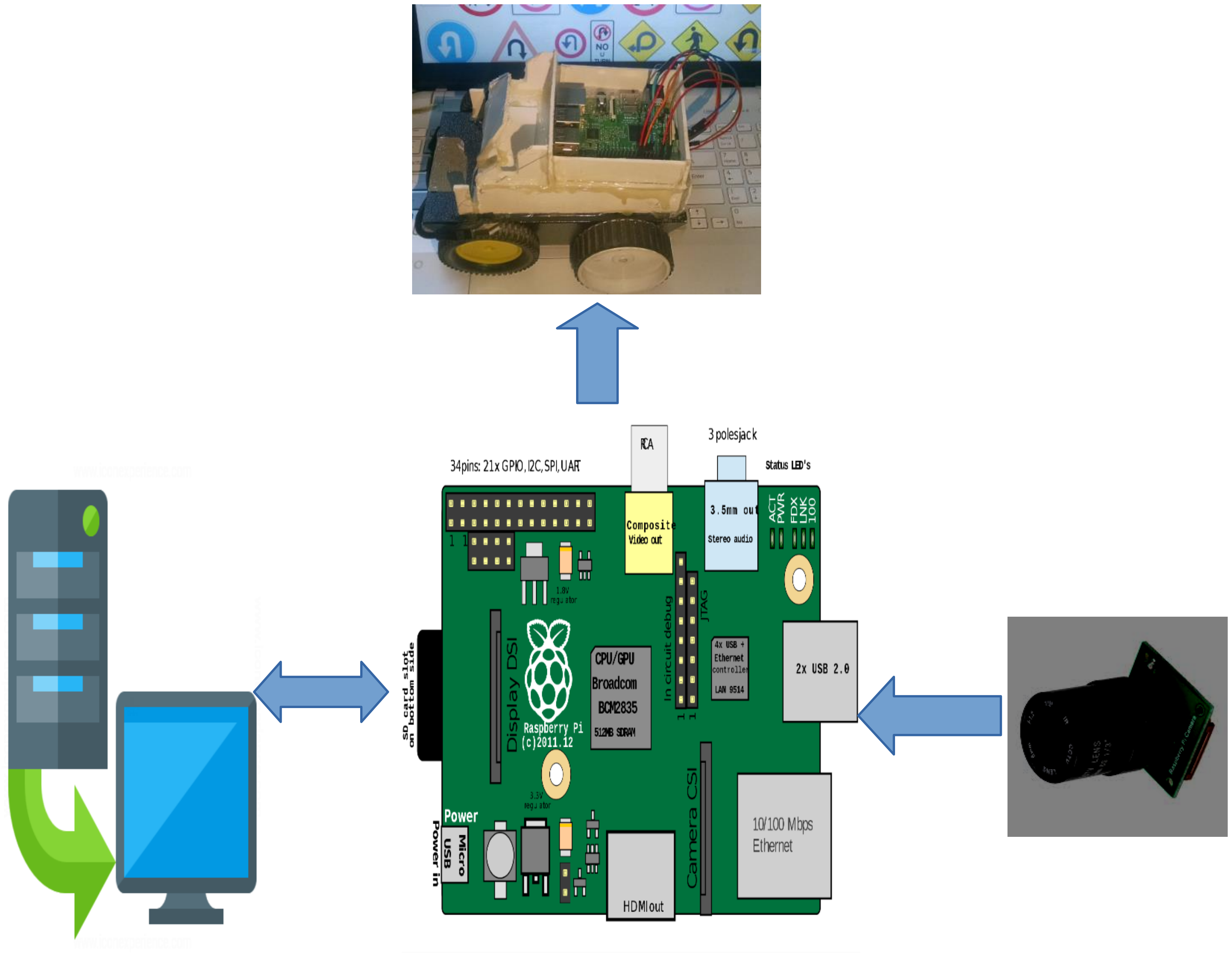


Fig 3. Block diagram of communication between server and client

The components of the project are Raspberry Pi mounted on a bot which moves by servo motors. The Raspberry Pi controls the movement of bot. Raspberry Pi is connected to local server (laptop) using routers. The Pi is connected to a camera mounted on bot. Pi continuously takes photos using camera and sends them to local server over Wi-Fi. The server will send to a folder of laptop. By using PHP we will create text file for image location by using shell scripting and caffe commands we will run the image and sends output to the bot for its movement over Wi-Fi. The Pi reads the output from the server and moves the bot accordingly.

To detect the sign we use neural network created on the server. This network is created using the software called Caffe. After the neural network is created it should be trained using data set. A data set consists of input images and the output text corresponding to image. The network processes the input in several stages and arrives at an output. Each stage is associated with some weights. Based on the output of the network and the data set output changes are made to these weights. And finally the network becomes trained able to get the correct output. After the training session the weights obtained are stored in a file. This file is the crucial component of the network. When a new image is given to the network it processes the image using these weights to get the correct output.

The training takes a lot of time but after getting weights the processing doesn't take much time. Here, to train the network we used the dataset provided by GTSRB (German Traffic Sign Recognition Benchmark). The design and development of algorithms allowing a machine to learn from large amounts of data and make predictions about the future is critically dependent on the process of extracting the most informative and non-redundant information from such data in their raw form. This process of feature extraction is traditionally carried out by humans and requires in general a great deal of expertise and labor, often including trial and error approaches. Increasingly, this process is being replaced by representation learning, a set of methods to automatically learn and discover new representations of the raw data, often outperforming traditional hand-crafted feature extraction. In this chapter, a family of representation learning methods known as deep learning will be introduced.

3.1. Convolutional neural networks

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other.
Convolutional neural networks (CNNs, or ConvNets) are a type of feedforward neural networks designed to handle data organized in the form of arrays with some degree of spatial structure (i.e. locally correlated). A typical example is a color (RGB) image, which is essentially a stack of three 2D arrays, and can be seen as a 3D tensor, with the third dimension being the color channel. The architecture of CNNs was born in the 1980s, with Fukushima’s neocognitron, inspired by a neurophysiological model of mammals’ visual primary cortex introduced by Hubel and Wiesel in the 1960s. Many fundamental aspects of this early model are still used by modern CNNs. At its core, a convolutional neural network is a stack of layers transforming a 3D input tensor to a 3D output tensor in a feedforward fashion. These layers perform linear or nonlinear transformation of their input, and these operations may or may not involve additional parameters and hyper parameters. Although this description may arguably apply to any form of FFNN, convolutional neural networks are different in that the extraction of features in the first layers, as well as the composition of such features into higher-level features, is performed through mathematical operations called discrete convolutions Convolutional neural networks were first trained with backpropagation and gradient descent by LeCun et al. in 1989, for the purpose of classifying handwritten digits. In the 1990s, CNNs-based algorithms were already used in commercial applications, such as reading cheques (the CNN architecture known as “LeNet-5” used in this work). However, at that time the computational power needed to train larger and deeper models to solve more complex tasks was simply too

large. This aspect, together with the rapid progress made in other areas of machine learning, hindered the diffusion of neural networks in the computer vision community for almost two decades.

3.2. The Conv.NET library

Several deep learning toolboxes have been released in recent years, thanks to a joint effort of academic departments (e.g. University of Montreal, UC Berkeley) and IT companies (e.g. Google, Microsoft, Facebook).

Each building block is implemented in its own class, derived from the base Layer class. In accordance to the simple layer notation, the base Layer class has two fields of class Neurons, called input Neurons and output Neurons, where the input and output activations of the layer are stored. Each derived class then implements different methods to transform input activations into output activations (forward pass), and to back propagate error signals (also stored in dedicated fields of the Neurons class) backwards (backward pass), giving rise to different types of layers.

3.3. Traffic sign recognition

Until the beginning of the decade, research in machine learning applied to the problem of traffic sign recognition has focused on the extraction of hand-crafted features for both the detection of a traffic sign instance and the subsequent classification into one of multiple classes. Convolutional neural networks were first applied to this problem in the beginning of this decade, along with the appearance of large publicly available data sets of labelled traffic sign images. The first traffic sign classification challenge was held at the 2011 International Joint Conference on Neural Networks (IJCNN), using the GTSRB data set. Training several neural networks and using the resulting ensemble for inference is a common technique to obtain a higher classification accuracy than that of individual networks.

3.4. The GTSRB data set

The German Traffic Sign Recognition Benchmark (GTSRB) is a publicly available data set containing 51839 images of German road signs, divided into 43 classes. A representative image for each class is shown in Figure 4. The data set was published during a competition held at the 2011 International Joint Conference on Neural Networks

Images in the data set exhibit wide variations in terms of shape and color among some classes, as well as strong similarities among others (e.g. different speed limit signs). The data pose several challenges to classification, including varying lighting and weather conditions, motion-blur, viewpoint variations, partial occlusions, physical damage and other real-world variabilities (some samples considered difficult to classify are shown in Figure. Furthermore, resolution is not homogeneous and as low as 15_15 pixels for some images. For these reasons, human performance on this data set is not perfect, and estimated at around 98.84% on average. An additional challenge in training classification algorithms on the GTSRB data is that the 43 classes are not equally represented. The relative frequency of classes 0, 19, and 37 for example, is only 0.56%, significantly lower than the mean 1=43. Moreover, since the data set was created by extracting images from video footage, it contains a track of 30 images for each unique physical real-world traffic sign instance. As a result, images from the same track are heavily correlated.

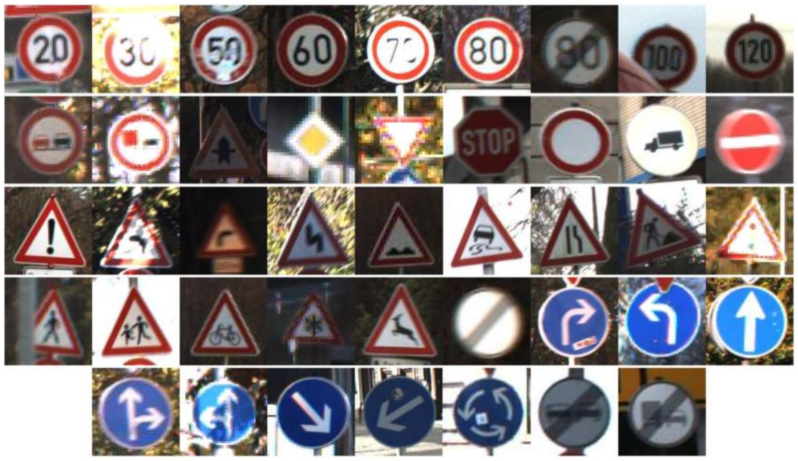


Fig.4.GTSRB Traffic signs

3.5. Data preprocessing

In accordance with the idea behind representation learning, GTSRB images were minimally preprocessed, and used to train CNNs almost in their raw form. The preprocessing steps described in this section follow the approach used by Sermanet and LeCun as closely as possible, for a better comparison.
First of all, a validation set was extracted from the training set by randomly picking one track (30 images) for each class, for a total of 1290 images. The remaining training set thus contains 37919 images1. This step was repeated twice, in order to perform holdout (2-fold) cross-validation (although with data sets of different size). In each image, the provided region of interest (ROI) containing the traffic sign was cropped and then rescaled to 32_32 pixels. The resulting images were then either converted to greyscale (GS), following Sermanet and LeCun, or kept as raw RGB images. Finally, each feature (i.e. pixel) of each image in the training set was normalized to zero mean and unit variance across the data set. Images were not normalized individually (across the image), but only across the data set. The same normalization used on the training data was then applied to the validation and test data.

3.6. HTTP Multipart

A HTTP multipart request is a HTTP request that HTTP clients construct to send files and data over to a HTTP Server. It is commonly used by browsers and HTTP clients to upload files to the server.
In a multipart/form-data body, the HTTP Content-Disposition general header is a header that can be used on the subpart of a multipart body to give information about the field it applies to. The subpart is delimited by the boundary defined in the Content-Type header. Used on the body itself, Content-Disposition has no effect. The Content-Disposition header is defined in the larger context of MIME messages for e-mail, but only a subset of the possible parameters apply to HTTP forms and POST requests. Only the value form-data, as well as the optional directive name and filename, can be used in the HTTP header.

3.7. Protocol used to connect microcontroller

SSH PROTOCOL

The SSH protocol (also referred to as Secure Shell) is a method for secure remote login from one computer to another. It provides several alternative options for strong authentication, and it protects the communications security and integrity with strong encryption. It is a secure alternative to the non-protected login protocols (such as telnet, rlogin) and insecure file transfer methods (such as FTP).

TYPICAL USES OF THE SSH PROTOCOL

The protocol is used in corporate networks for:

- providing secure access for users and automated processes
- interactive and automated file transfers
- issuing remote commands
- Managing network infrastructure and other mission-critical system components.

4. Literature Survey

4.1 Architecture Overview

Traditionally, convolution neural networks are implemented as shown below.

- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R, G, B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the max (0, x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a down sampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume

4.2 Input layer

Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: **width, height, depth**. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.)

Input layer defines the size of the input images of a convolutional neural network and contains the raw pixel values of the images. In the image shown below flickr32 is input layer n.
Let the image size given to the input layer be 9X9. The image consists of colours. Generally RGB (Red, Green, and Blue) model is used to specify the colours of image. Each pixel will have three components of red, green and blue. Hence an image is represented as three matrices of 9x9 size which corresponds to red, green and blue components of image for each pixel. Hence the image is now 9x9x3.
One key idea behind the strength of neural networks is that these hierarchically arranged features can to some extent be learned independently of one another, in a distributed fashion.
A distributed representation of features is a highly desirable property in machine learning when dealing with high-dimensional data. In the case of a d-dimensional input space (e.g. Rd), a distributed representation of n features can divide the space into O (nd) regions, corresponding to intersection of half-spaces.

If such regions represent concepts, then exponentially many concepts can be distinguished using such a representation, since the different attributes of these concepts are shared. The first layer of a neural network object must always be of type Input Layer. This is an auxiliary class performing the identity function in the forward pass (the backward pass has 1no meaning, and is thus not implemented), used to pass a mini-batch of examples into the network, through the method FeedData (). In adding an Input Layer instance to an empty Neural Network, the input tensor depth and spatial dimensions must be passed as arguments in the constructor. The mini-batch size (i.e. the first dimension of the tensor of activations) is instead set using a property of the static class NetworkTrainer, reflecting the fact that it is not a property of the neural network, but rather of the training procedure.

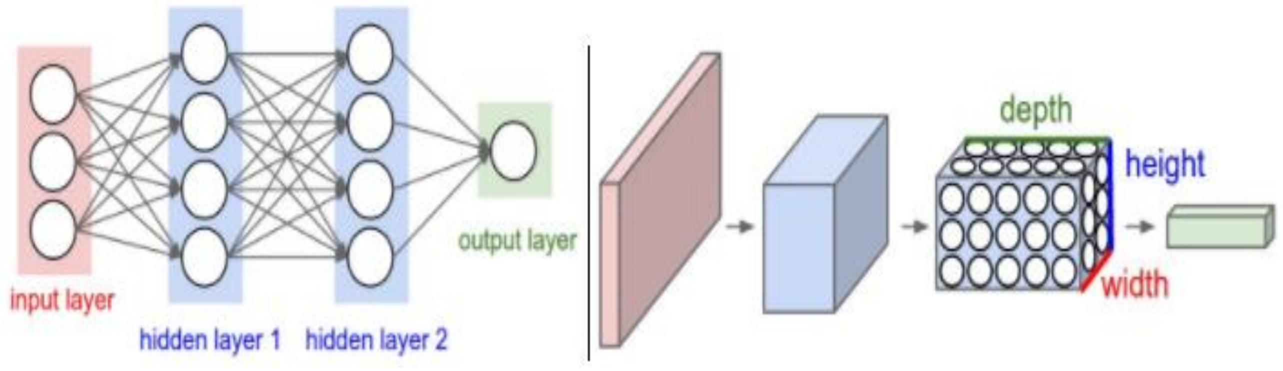


Fig.5. Convolution Neural Network

4.3 Convolution layer

In the network shown below conv1, conv2, conv3 are convolution layers. These convolution layers have filters that process the image. In the figure shown below, input volume corresponds to an image. Here, a 5x5 image is made 7x7 by padding with zeros as shown below. Now, the image is 7x7x3 and is convoluted with 2 filters w0 and w1 of size 3x3x3 with stride 2. The output is 3x3x2. The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.

The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.

When dealing with high-dimensional inputs such as images, as we saw above it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyper parameter called the **receptive field** of the neuron (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space, but always full along the entire depth of the input volume.

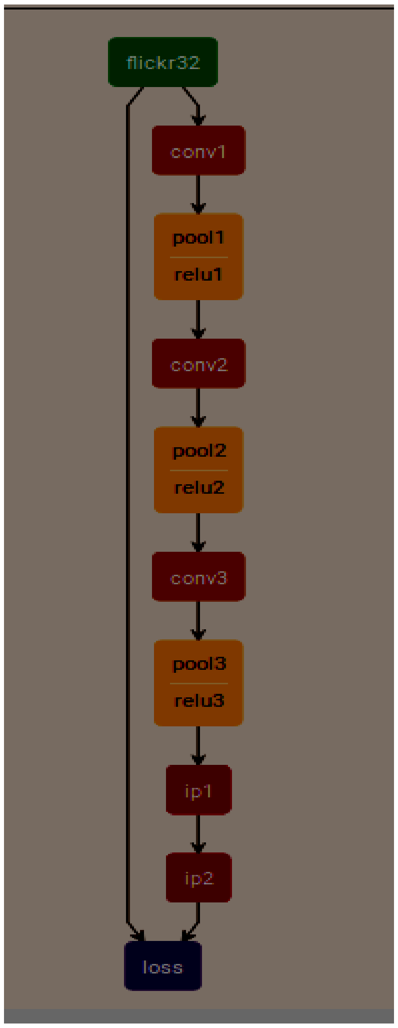


Fig.6.Network Architecture of traditional convolution neural network

This is the block diagram of a simple neural network system containing three pool layers
And three relu layers.

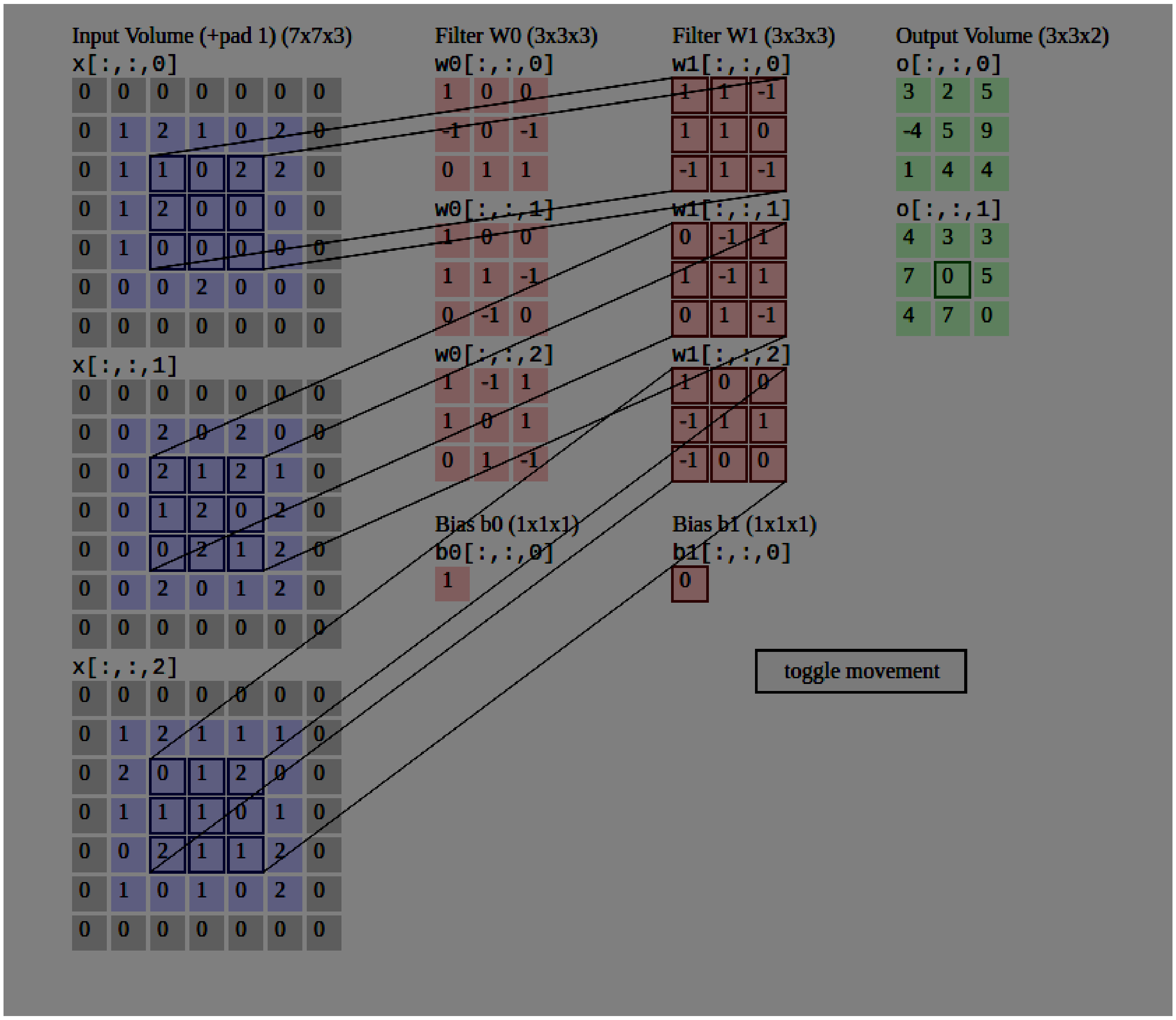


Fig.7. Process of convolution in the convolution layer

4.4 Pooling layer

Pooling layers are used to down sample input spatial information, and they have been traditionally added between stages of convolutions. Each pooling unit in channel k of the output tensor computes a summary statistic of a patch of units in channel k of the input tensor. As a result, only spatial dimensions of the input tensor X are down sampled, whereas the depth is always preserved. The most important effect of pooling is that it makes the feature representation invariant to small translations and distortions, allowing to detect the presence of a feature regardless of its exact location in the input image.

In the network shown above pool1, pool2, pool3 etc. are pooling layers. They reduce the size of the matrix obtained after convolution to reduce large amount of computation. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged.

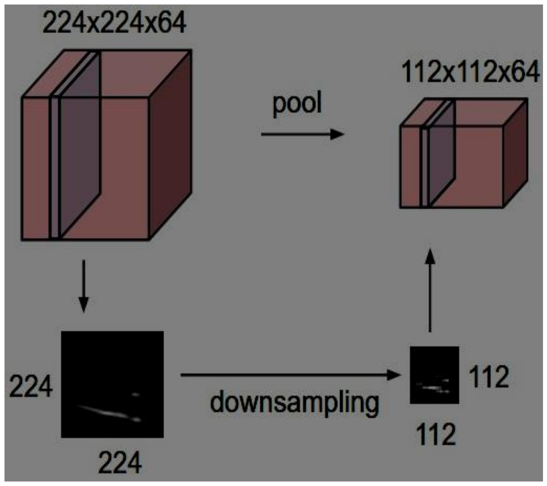


Fig.8.Downsampling

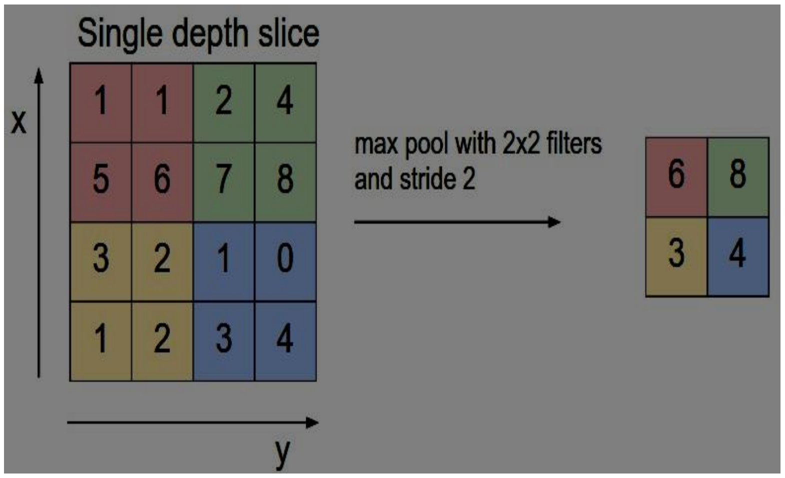


Fig.9.pooling

Left:
In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved.
Right: The most common down sampling operation is max, giving rise to **max pooling**, and here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

4.5 Activation layer

Activation layer will apply an element wise activation function, such as the max (0,x) thresholding at zero. This leaves the size of the volume unchanged. This makes the calculation simpler by making all the negative values to zero. There are several activation functions like relu, tanh etc.

4.6 Fully Connected layer

Fully-connected layers are usually employed at the end of CNNs as classifiers. Their task is to process higher-level features learned by the underlying convolutional modules and learn to classify the input example based on such features.
In the above network ip1 and ip2 are fully connected layers. After passing through all the convolution layers the input to fully connected layer will be generally like 1x1xn. Now, these values are mapped to certain number of classes. The value of each class will be affected by all the elements, n. Let the number of classes be p. Then the output of fully connected layer will be p.

4.7 Classifier

The classifier computes the percentage of every element in output of fully connected layer, p. The element with the highest percentage is the answer for the given input. In the above network loss is classifier.
The above described ConvNets are organized in strict feed-forward layered architectures in which the output of one layer is fed only to the layer above. Instead, the output of the first stage is branched out and fed to the classifier, in addition to the output of the last stage. The motivation for combining representation from multiple stages in the classifier is to provide different scales of receptive fields to the classifier. In the case of 2 stages of features, the second stage extracts global and invariant shapes and structures, while the first stage ex- tracts local motifs with more precise details. It is found out that skip layered networks converge faster than the traditional feed forward networks and also it does not over-fit readily compared to traditional feed forward network. Sermanet was one of the first to propose this method of multiscale approach for classification.

4.8 Flattening layer

Flattening layer converts the pixel matrices into single row matrices. If the input is 9x9x3 the output is 243x1x1.

4.9 Concatenation Layer

Concatenation layer joins the two inputs. A row is attached to the end of the other.

5. Code Implementation

5.1 Code executed on BOT side

```
#!/bin/bash
python2 WT15-DL_SignKafe_main.py
rm -r /var/www/html/signs/*
haricl=$(<output.txt)
#echo $haricl
case "$haricl" in
	 "20 speed") i=0
			;;
	"30 speed") i=1
		;;
	"50 speed") i=2
		;;
	"60 speed") i=3
		;;
	"70 speed") i=4
		;;
	"80 speed") i=5
		;;
	"100 speed") i=7
		;;
	"120 speed") i=8
		;;
	"stop") i=14
		;;
	"attention left turn") i=19
		;;
	"attention right turn") i=20
		;;
	"attention bumpers") i=22
		;;
	"attention slippery") i=23
		;;
	"attention traffic light") i=26
		;;
	"attention children") i=28
		;;
	"attention bikes") i=29
		;;
	"attention deer") i=31
		;;
```

```
        "turn right") i=33
                ;;
        "turn left") i=34
                ;;
                *)i=8
                ;;
esac
echo $i > outputpi.txts
cp outputpi.txt /var/www/html/
```

5.2 Code executed on Server Side

```
#!/usr/bin/env python2

Classify an image using a model archive file

import os
import time

from example import classify

#def classify_image(image_files, use_gpu=True):
def classify_image(image_files, use_gpu=False):
    caffemodel = '../caffe/20151207-223900-80d9_epoch_30.0/snapshot_iter_19140.caffemodel'
    deploy_file = '../caffe/20151207-223900-80d9_epoch_30.0/deploy.prototxt'
    mean_file = '../caffe/20151207-223900-80d9_epoch_30.0/mean.binaryproto'
    labels_file = '../caffe/20151207-223900-80d9_epoch_30.0/labels.txt'

    classify(caffemodel, deploy_file, image_files,
            mean_file=mean_file, labels_file=labels_file, use_gpu=use_gpu)


if __name__ == '__main__':
    script_start_time = time.time()

    image_files = [os.path.join('../images/', f) for f in os.listdir('../images/')]

    classify_image(image_files)

    print 'Script took %s seconds.' % (time.time() - script_start_time,)
```

5.3 Camera Detection on Android

```
public class CameraActivity extends Activity implements CvCameraViewListener2{
  int i=0;
      private static final Scalar    FACE_RECT_COLOR     = new Scalar(0, 255, 0, 255);
      private CameraBridgeViewBase mCameraView;
      private ListView listDetectedSigns;
      private RelativeLayout listRelativeLayout;
      private CascadeClassifier cascadeClassifier;
      private ArrayList<Sign> listSign;
      private Detector detector;
      private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
          @Override
          public void onManagerConnected(int status) {
            switch (status) {
              case LoaderCallbackInterface.SUCCESS:
                  mCameraView.enableView();
                  detector = new Detector(CameraActivity.this);
                break;
              default:
                super.onManagerConnected(status);
                break;
            }
          }
        };
      private Mat mRgba;
      private Mat mGray;

            //detector = new Detector(CameraActivity.this);
      private void Initialze(){
            mCameraView = (CameraBridgeViewBase)findViewById(R.id.mCameraView);
            listDetectedSigns = (ListView)findViewById(R.id.listView1);
            listRelativeLayout = (RelativeLayout)findViewById(R.id.listViewLayout);
            mCameraView.setCvCameraViewListener(this);
            listRelativeLayout.setVisibility(View.GONE);
      }

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
            setContentView(R.layout.camera_preview);
```

```
			Initialze();
		}
		@Override
	public void onResume() {
	    super.onResume();
			if (!OpenCVLoader.initDebug()) {
				Log.d("S", "Internal OpenCV library not found. Using OpenCV Manager for
initialization");
				OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_11, this,
mLoaderCallback);
			} else {
				Log.d("s", "OpenCV library found inside package. Using it!");
				mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCESS);
			}

	}

		@Override
		public boolean onCreateOptionsMenu(Menu menu) {
			// Inflate the menu; this adds items to the action bar if it is present.
			getMenuInflater().inflate(R.menu.main, menu);
			return true;
		}

		@Override
		public void onCameraViewStarted(int width, int height) {
			// TODO Auto-generated method stub

		}

		@Override
		public void onCameraViewStopped() {
			// TODO Auto-generated method stub

		}

		@Override
		public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
			//TODO Auto-generated method stub

			mRgba = inputFrame.rgba();
			if(i==100) {
```

```
                i=0;
                mGray = inputFrame.gray();

                Imgproc.equalizeHist(mGray, mGray);
                MatOfRect signs = new MatOfRect();
                listSign = new ArrayList<Sign>();

                detector.Detect(mGray, signs, 1);
                Rect[] prohibitionArray = signs.toArray();
                Draw(prohibitionArray);

                detector.Detect(mGray, signs, 2);
                Rect[] dangerArray = signs.toArray();
                Draw(dangerArray);
                //Core.rectangle(inputFrame, facesArray[i].tl(), facesArray[i].br(), new Scalar(0,
255, 0, 255), 3);*/

            }
            i++;
            return mRgba;
        }

        public void Draw(Rect[] facesArray){
            if(facesArray.length<=0){
        runOnUiThread(new Runnable() {

                    @Override
                    public void run() {
                        // TODO Auto-generated method stub
                        listRelativeLayout.setVisibility(View.GONE);
                    }
                });

       }
       for (int i = 0; i <facesArray.length; i++){
        final int ii = i;
        Mat subMat = new Mat();
        subMat = mRgba.submat(facesArray[i]);
        Sign.myMap.put("image"+i, Utilities.convertMatToBitmap(subMat));

        Core.rectangle(mRgba,facesArray[i].tl(), facesArray[i].br(), FACE_RECT_COLOR, 2);
        runOnUiThread(new Runnable() {

                                @Override
                                public void run() {
                                        // TODO Auto-generated method stub
                                        Sign sign = new Sign("unknown", "image"+ii);
                        listSign.add(sign);
                                        listRelativeLayout.setVisibility(View.VISIBLE);
                                        itemAdapter        adapter=        new        itemAdapter(listSign,
CameraActivity.this);
                                        adapter.notifyDataSetChanged();
                                        listDetectedSigns.setAdapter(adapter);
                                }
                        });

      }
        }
}
```

5.4 Sign Detection on Android

```
public class Detector {
        private Activity activity;
        private CascadeClassifier cascadeClassifier;
        public Detector(Activity activity){
                this.activity = activity;
        }
        public void Detect(Mat mGray,MatOfRect signs,int type){
                //loadCascadeFile(type, cascadeClassifier);
                loadCascadeFile(type);
                if (cascadeClassifier != null) {
        cascadeClassifier.detectMultiScale(mGray, signs, 1.1, 3, 0, new Size(30,30),new Size());
      }
        }
        private void loadCascadeFile(int type){
                try {
                        InputStream is = null;
                        File cascadeDir = activity.getDir("cascade", Context.MODE_PRIVATE);
                        File cascadeFile=null;
                        switch (type) {
                        case 1:
                                is = activity.getResources().openRawResource(R.raw.bienbaocam);
                        cascadeFile = new File(cascadeDir, "bienbaocam.xml");
                        break;
                    case 2:
                    default:
                        is = activity.getResources().openRawResource(R.raw.biennguyhiem);

                        cascadeFile = new File(cascadeDir, "biennguyhiem.xml");
                        break;
                    }

                    FileOutputStream os = new FileOutputStream(cascadeFile);
                    byte[] buffer = new byte[4096];
        int bytesRead;
        while ((bytesRead = is.read(buffer)) != -1) {
           os.write(buffer, 0, bytesRead);
        }
        is.close();
        os.close();


        // Load the cascade classifier
        cascadeClassifier = new CascadeClassifier(cascadeFile.getAbsolutePath());

            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
}
```

6. Real Time Implementation (BOT Practical Prototype)

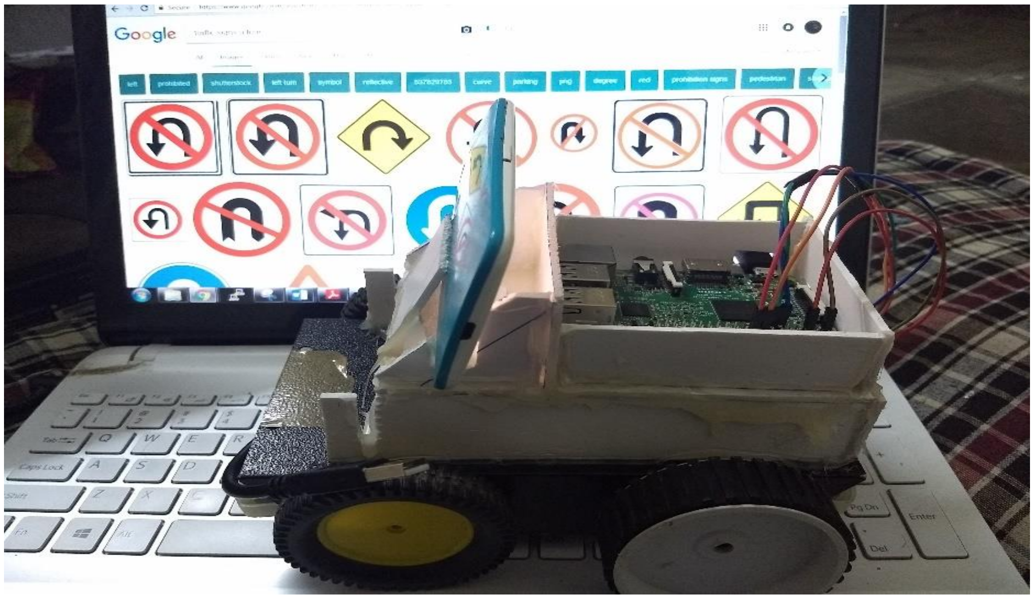


Fig.10 BOT car

Above shown is BOT (car) which is automatically driven and the camera captures the images of traffic sign which are then processed by server and hence the car moves according to the output given by server.

7. Image detection using Android Sign detection App

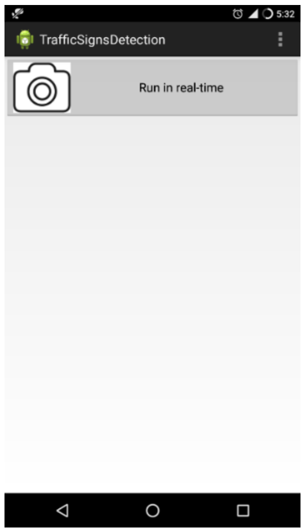


Fig 11. Home View of the sign detection app which uses OPENCV library for segmentation

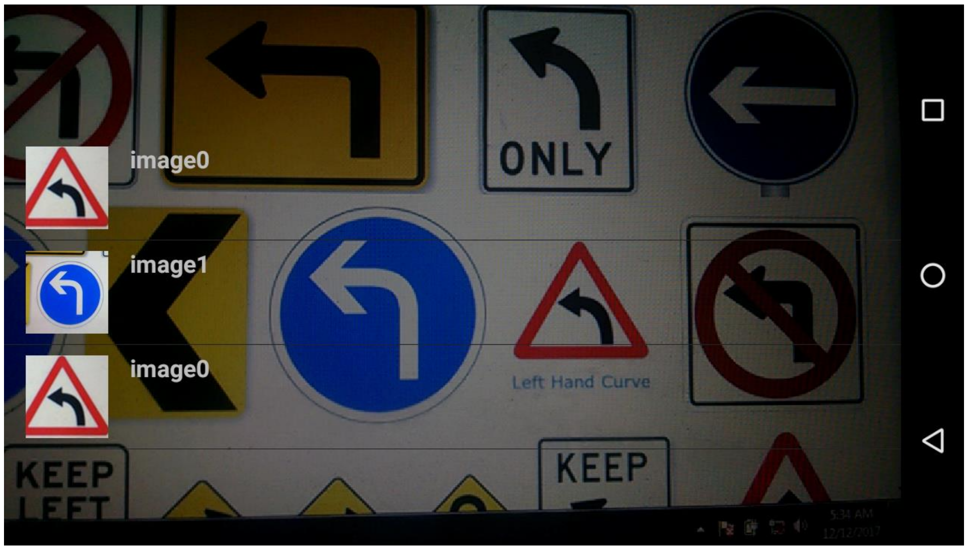


Fig.12 Here multiple left sign images are detected irrespective of color and size.

At the bottom left corner Multiple Images are being detected as shown and detected images Are then sent for further processing to server where deep learning part is applied.

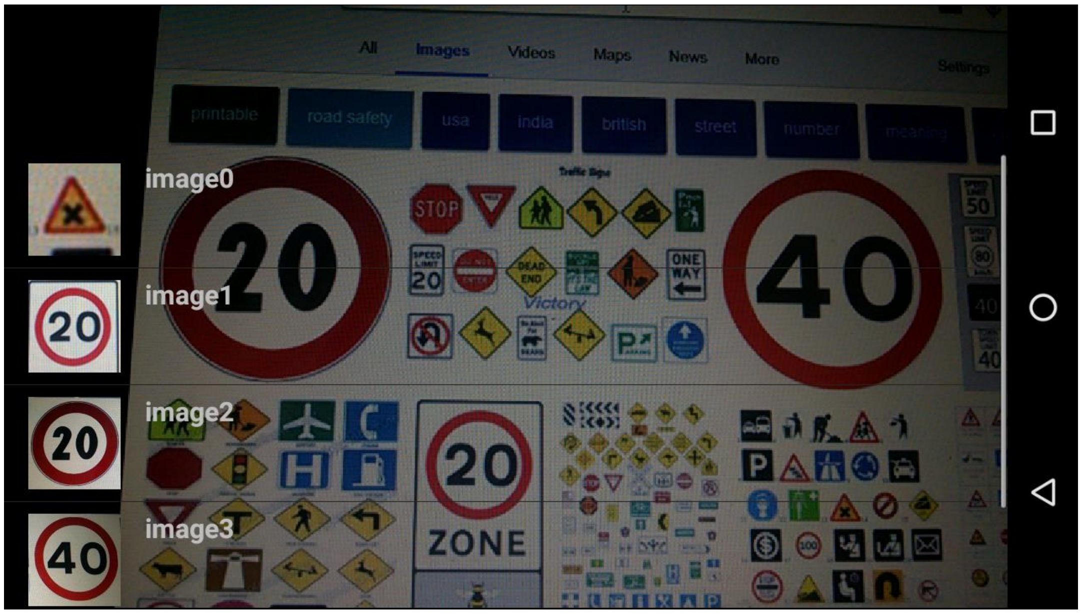


Fig.13 Segmentation of speed limit images automatically after 100 frames using sign detection app.

8. Conclusion

The project aims to provide an optimal solution to detect traffic signs in automated car. The Convolution Neural network algorithm provides 90% accuracy in detecting the traffic signs. OpenCV library used in android app applies segmentation and cropping to only the useful part of traffic sign and hence increases the output efficiency of deep learning algorithm. Using deep learning the machine itself understands and analyses the image and hence produces suitable Instruction for the bot to move. This reduces human intervention and can be monitored using IP camera.

9. References

[1] Stallkamp, J, Schlipsing, M, Salmen, J, and Igel, C. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In International Joint Conference on Neural Networks, 2011.
[2] CS231n: Convolution Neural Networks for Visual Recognition.
[3] Pierre Sermanet, and Yann LeCun.Traffic Sign Recognition with Multi-Scale Convolutional Networks.
[4] A. de la Escalera, Road Traffic Sign Detection and Classification, Journal, IEEE
[5] J. Crissman, C. E. Thorpe, "UNSCARF a color vision system for the detection of unstructured roads", Proc. IEEE Int. Conf. Robotics and Automation, pp. 2496-2501, Apr. 1991.