AUTOMOTIVE ACCIDENT DETECTION

By

Rawan Mohamed Idris

Zeinab Hosny Mohamed

Maha Hassan Mohamed

Mohamed Eid Ebrahem

Mayada Zaki Mohamed

Mayada Essam Ali

Under the Supervision of Associate Prof. Hassan Mostafa

A Graduation Project Report Submitted to

the Faculty of Engineering at Cairo University in

Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science in

Electronics and Communications Engineering Faculty of Engineering,

Cairo University, Giza, Egypt.

July 2019

Abstract

Road accidents can't be predicted, they happen suddenly causing huge damage and even taking lives. Most drivers have made an accident or more, they even witnessed a large number of accidents on the road daily.

Cars are provided with sensors to reduce the effect of the accident on the driver like airbags. But however, they have done, it's not enough as people are still dying and the number of accidents increases year by year.

The causes of death and bad injuries are emergency arrives late due to traffic jam and lack of protections in the car itself.

So, the aim of this project is to build a system in cars to help emergency to reach faster to the accident site and to help to save critical conditions.

Keywords: Gazebo, Alexnet, V2V, GNU radio, USRP.

Acknowledgment

We are using this opportunity to express our gratitude to everyone who supported us throughout the graduation project. We are thankful for their aspiring guidance and friendly advice.

Firstly, a special thanks to our major advisor Dr. Hassan Mustafa for his encouragement throughout the whole year, his caring about following up each stage in the project and his suggestions to solve some problems we faced during the project work.

Secondly, we want to thank Eng. Mohamed Abduo, Senior Algorithms Engineer and Deep Learning Researcher at Valeo Egypt, for providing his time, experience to help us overcome some obstacles we faced during some stages especially when dealing with new concepts and tools.

Thirdly, Listed below are the names of the people who provided us with significant help in developing our graduation project in addition to our sponsor Valeo Egypt. To all, we extend our sincere thanks.

Eng. Ahmed Radwan

Eng. Habiba Eltoudy

Eng. Zeinab Ahmed

Finally, we want to thank our families for their support, tolerance and love during this year especially during the hard times they were always there having faith in what we do. We are grateful to our families, colleagues and friends for always motivating us, without them we wouldn’t have come so far.

Contents

Table of Figures

List of Tables

List of Abbreviations

ROS	Robot Operating System
SDF	Simulation descriptive format
RViZ	Ros visualization
CNN	Convolution neural network
ReLU	Rectified linear unit
LRN	Local response normalization
DNN	Deep neural network
FC	Fully connected
MLP	Multi-layer perceptron
V2V	Vehicle to vehicle
V2I	Vehicle to infrastructure
V2X	V2I and V2V
IEEE	Institute of electrical and electronic engineers
IEEE 802.11P	IEEE standard for wireless communication
MAC	Medium access control
LAN	Local area network
PHY	Physical layer
STD	Standard
OFDM	Orthogonal frequency division multiplexing
PDU	Protocol data unit
USRP	Universal software radio peripheral

Chapter 1 Introduction

In this thesis, we are going to present our work in the project to achieve End-to-End crash avoidance including going through the phases and taking the suitable techniques for our required functionality.

1.1 Motivation

In recent years, artificial intelligence and deep learning have shown their utility and effectiveness in solving many real-world computation-intensive problems. The motivation of these is to eliminate the need of direct programming and create an intelligent system that can automatically extract features and recognize a particular pattern and after having learned to recognize a particular pattern, extend that capability to objects that it hasn't actually seen before. In other words, it doesn't have to be trained on every single situation that could possibly exist, that makes deep-learning algorithms better suited in variable, situation-dependent decisions as in self-driving cars than traditional, rules-based approach. It learns the entire processing pipeline needed to steer an automobile by creating models that meet or exceed the ability of a human driver which could save thousands of lives a year.

Among various deep learning algorithms, CNN (convolutional neural networks) is one of the key algorithms for visual content understanding and classification, with significantly higher accuracy than traditional algorithms in many applications, such as image/video processing, face recognition, machine language translation, advances in medicine, autonomous driving and more.

Classification is a central topic in machine learning that has to do with teaching machines how to group together data by particular criteria. Classification is the process where computers group data together based on predetermined characteristics- this is called supervised learning.

Car accidents are a huge problem in our world. Nearly 1.3 *million* people die internationally every year from car accidents and in addition up to 50 million people are injured.

Recently, Crash Avoidance functionality is partially integrated in different ways and levels. Forward Collision Warning (FCW) and Automatic Emergency Braking (AEB) are considered as the initial trials to integrate Crash Avoidance functionality. In FCW functionality, the vehicle has the ability to only warn the driver that there is an object in front of you, so the driver either takes the responsibility of taking reasonable actions. However in AEB, the vehicle starts taking action through braking in case the vehicle comes near to the front object. These actions are taken by the ego-vehicle based on integrating advanced sensors like: Laser, Camera and Radar, but these vehicle actions lead to occurrence of many crashes, in addition to being source of congestion because of its poorness.

And no one can deny that Deep Learning and Computer Vision, recently, invade the automotive field powerfully. Deep Learning is contributing greatly in many automotive applications like: Autonomous Driving (AD), and Augmented Reality (AR). Autonomous Driving needs sophisticated and advanced functionalities to be ensured like: Automatic Emergency Braking (AEB), Lane Keeping Assist (LKA), Active Cruise Control (ACC), Traffic Jam Assist (TJA), and Crash Avoidance (CA).

Fully Autonomous Driving is one of the difficult problems faced the automotive applications. It is forbidden due to the presence of some restricted Laws that prevent cars from being autonomous for the fear of accidents occurrence. However, researchers try to reach autonomous driving as a new area for research for the aim of having a strong push against these restricted Laws. Crash Avoidance functionality is considered as one of the most important features in Self-Driving Cars. Recently, it is partially integrated in the Self-Driving Car system.

In addition, instead of vehicles working independently and spending a lot of time in the road traffic, V2V communication systems can make it possible to share critical information across

vehicle surroundings, so that vehicles can have all the possible information about what happens in their routes and take the suitable decision.

In a NHTSA announcement, administrator Mark Rosekind said **"V2V and automated vehicle technologies each hold great potential to make our roads safer, and when combined, their** potential **is untold. Advanced vehicle technologies may well prove to be the silver bullet in saving lives on our roadways."**

1.2 Problem statement

Thousands of people across the world are losing their lives due to car accidents and road disasters every year and the related lost costs due to medical expenses and general maintenance and repairs costs of the road and highway systems are very large. Another big problem we face because of accidents is the traffic crowd, many people lose their time in roads due to car accidents, and this crowd also makes the ambulance and the rescue reach the accident location very late which may cause losses in lives.

So, we aim in this project to help in solving these problems, by providing a car accident avoidance system.

1.3 Solution approach

The aim of this project is to build a system in cars to help emergency to reach faster to the accident site and to help saving critical conditions. This would include the following:

1. Analyzing the video which is taken by the front camera of each car.

2. Frames are considered the input of a well-trained neural network.

3. Classifying the frames whether they have an accident or not.

4. Sending an alarm, if an accident occurs, to all nearby cars using V2V Communication system.

So, the expected outcome is to send an alarm message through V2V from the car that captured the accident to all nearby cars so the drivers can change their lane or even route to avoid traffic, allowing emergency to arrive faster and help in saving lives.

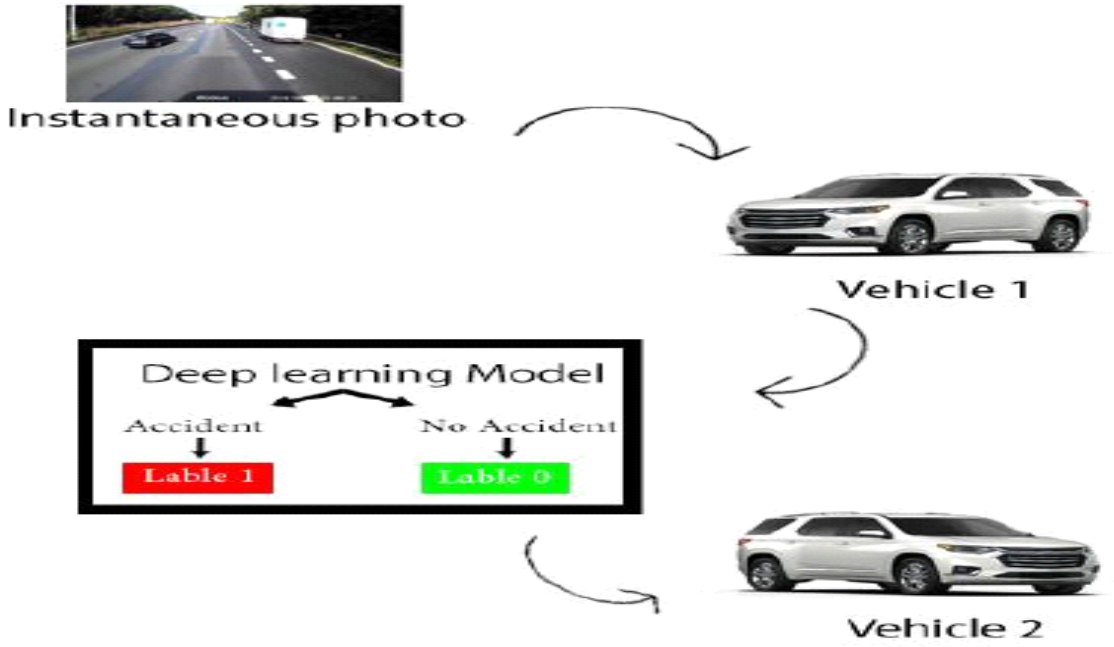


Figure 1: System flow.

Figure 1 shows the functional requirements and operations of the system. Where the input of the system is the instantaneous image of the front camera of vehicle one then the operation of the deep learning model is to classify this image to an accident with label 1 or non-accident with label 0, after classification the output is sent by V2V protocol to vehicle two.

1.4 Project milestones

Milestone 1: Getting started with the project

At first, we needed to divide our project core into three main parts which are simulation part, deep learning model algorithm and V2V technology.

• Concerning simulation part; GAZEBO simulation tool mimics the real accidents environment.

• Concerning deep learning model; Alexnet classifier trains the training images simulated in GAZEBO.

• Concerning V2V technology; the expected scenario will be as following: The ego vehicle will communicate with the upcoming vehicles by V2V announcing accident information to them.

Milestone 2: Learning phase

In this phase, we started to learn more about the simulation part and the deep learning part as following:

1- For the simulation part, we managed to learn more about how to interface with Gazebo tool using in this some online tutorials.

2- For deep learning part, we managed to complete the online deep learning courses especially CNN ones.

Milestone 3: Excel sheet for accidents and non-accidents scenarios

In this phase, an excel sheet is developed to collect all the possible real scenarios for accidents and non- accidents to use it for simulating different scenarios in GAZEBO.

Milestone 4: Going deeper in a deep learning algorithm and GAZEBO simulator

In this phase, we went deeper in deep learning part and the GAZEBO simulator part as the following:

1- For deep learning part, we managed to investigate different classification CNN's such as LENET 5, Alexnet, VGG 16 … so that we can choose our suitable classification CNN.

2- For GAZEBO simulator part, we managed with the help of online tutorials to insert different models in GAZEBO such as cars, houses, routes …. And by using those models, we managed to build a small world in GAZEBO and also, we managed to move cars manually.

Milestone 5: Work progress in training and extracting data

The work progress in this phase can be represented in the following:

1- We managed to build a big real world in GAZEBO containing more models such as humans, schools, factors, multi-types of cars…...and started to simulate different scenarios based on two sources:

• Firstly, the different scenarios in the excel sheet.

• Secondly, real online accident videos.

Then from the front camera in the car in GAZEBO, training data are extracted.

2- After extracting training data, we are ready to start training the classifier model by those data (Alexnet is our chosen classifier model).

Milestone 6: Continuing work progress

In this phase, we continued in extracting more training data and training the classifier model. Also, we started to extract some different scenarios for test data and started to get some knowledge about V2V technology to be ready to use it in further phases.

Milestone 7: Work progress in V2V and model testing

The work progress in this phase can be represented in the following:

1- For test data, we extracted more data by simulating new scenarios in GAZEBO and by using online videos to be source of real accident and no accident frames.

2- For test frames taken from online videos, we made the needed calibration for them.

3- Using the test data, we tested the classifier model.

4- We started to search for a software tool for V2V communication and hardware for it if it is possible.

Milestone 8: Continuing working in V2V communication part, real simulation and final thesis

The work progress in this phase can be represented in the following:

1- Implementing the V2V part by using both tools; GNU radio and USRP, where GNU radio represents the software environment to represent the tx and rx sides and USRP represents the wireless channel used to transfer the data from one end to another.

2- Real simulation represented in capturing frames from live videos in college, taken from camera, then sending the classifier result to the receiver through USRP channel.

3- Writing this final thesis.

1.5 Organization

The following chapters discuss the CNN algorithm structure how it is implemented by software, the way that we follow to obtain the dataset and the protocol we used to communicate between cars. The remainder of this thesis is organized as follows:

Chapter 2 provides the system design and implementation; it discusses the design of each part in the project, including software and hardware tools used in this system.

Chapter 3 provides the simulation tool which is used to generate the dataset, the accident and no-accident scenarios that are realistic enough to simulate the real scenarios that models and scripts that we needed to build the environment, the amount dataset that we reached finally.

Chapter 4 provides the deep learning model architecture used for classifying accident and non-accident data. It discusses the preprocessing steps those applied on the data before entering the model.

Chapter 5 provides the V2V communication process and the design of this system.

Chapter 6 presents our results and our attempts to achieve these results.

Chapter 7 provides a brief overview of the findings, draws conclusions, and recommends directions for future work.

Chapter 2 System design and implementation

2.1 System design

The system design involves three main parts:

2.1.1 Simulating the scenarios

The scenarios are made according with the following procedures:

1- Models are imported to gazebo to build our simulation world such as vehicles, walking and standing humans, trees, houses…etc.

2- Python scripts are written to:

a) Move the dynamic models (vehicles) in the world.

b) Capture images from the front camera and save them.

c) Give labels to the captured images (‘0’ refers to no accident, ‘1’ refers to accident).

3- Run the scripts and start the simulation.

2.1.2 Building the classifier

Alexnet is a convolution neural network proposed by Alex Krizhevsky which has had a large impact on the field of machine learning especially in the application of deep learning to machine vision.

It is trained on more than a million images from the Image Net database and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. Alexnet is composed of 5 convolution layers followed by 3 fully connected layers as shown in the following structure.

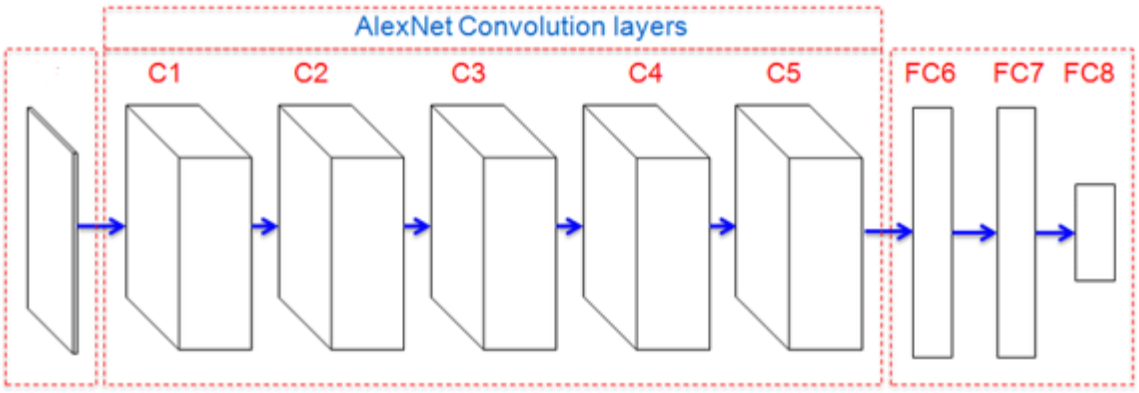


Figure 2: AlexNet architecture.

Alexnet implementation and features fit our objectives, so we used the Alexnet paper in designing our convolution neural network and applying some modifications. Figure 3 shows Alexnet flow.

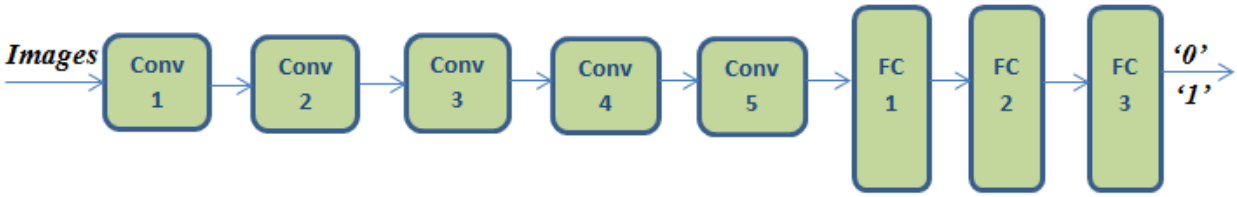


Figure 3: AlexNet layers.

2.1.3 Transmitting the classifier output among vehicles

The classifier output will be sent by V2V technology to the nearby vehicles as illustrated in figure 4.

Figure 4: communication flow.

2.2 System implementation

2.2.1 Software platform

Software component	Description
Gazebo-ROS	This software tool is launched on Linux operating system to simulate accidents and non-accidents scenarios, to be captured by the front camera of the car and extracted as a jpeg format. The simulation is carried out by scripts written by python programming language.
Anaconda	We used anaconda navigator to create the environment that includes the required libraries such as Tensorflow, numpy…etc. The classifier is built in the environment using mainly python programming language and Tensorflow library.
GNU radio	The GNU Radio software provides the framework and tools to build and run software radio or just general signal-processing applications. GNU radio represents in our project the software platform that simulates both wi-fi tx and wi-fi rx sides.
Google Colaboratory	It is a free cloud service that allows us to develop deep learning applications using popular libraries such as Keras, Tensorflow and OpenCV. It also provides GPU and is totally for free, so for these reasons sometimes we used Colab for training and testing our model.

Table 1: Software platform

2.2.2 Modules/component acquired from external sources

Hardware component	Description
GPU(GTX1080ti)	We needed a fast GPU to reduce the processing time of the training and test the classifier to obtain a real time performance. The GPU is acquired from Cairo University to help us to train the classifier.
USRP	It stands for Universal Software Radio Peripheral. The device is used as transceiver for radio frequency signals in wireless communication systems through the development of Software-defined radios. We need this tool in our project to perform as a channel between the transmitting vehicle and the received one.

Table 2: System hardware

Chapter 3 Dataset

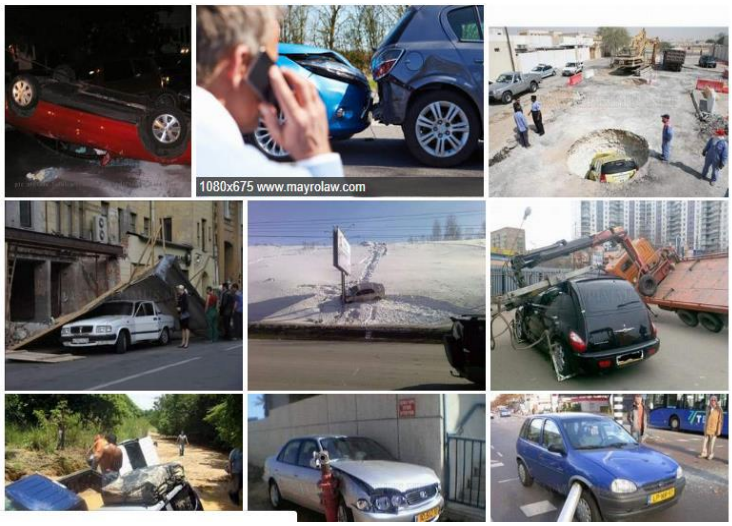
This chapter includes the simulation tool which is used to generate the dataset that is needed to train the deep learning model, the accident and no-accident scenarios that are realistic enough to imitate the real scenarios, the scripts that we needed to build our environment, and it discusses the augmentation applied on the generated data. It also involves the augmentation that is applied to the images in order to increase the number of data.

3.1 Image-Net

It is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. But this method is not appropriate enough to cover our requirements and considerations.

3.2 Google dataset

Indeed, we found a lot of data; however, the training data must be from the same camera with the same resolution and depth, also we cannot find all the required scenarios. So, we found this way cannot guarantee such requirements.



3.3 Papers

There is a paper found in this field that helped us to find around 526 samples of logic "0" but there are no open-source samples of logic 1, which is "A Novel Approach to Automatic Road-Accident Detection using Machine Vision Techniques".

3.4 ROS

3.4.1 Overview

ROS is open-source. A meta-operating system for robots runs on Unix-based platforms.it provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

The ROS runtime "graph" is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure.

Basic concepts:

- **Package:** is a place where we can put all ROS files related to specific ROS program. It contains two main folders which are launch folder and source folder, and two main files which are CMakeLists.txt and package.xml.
- **Catkin workspace:** is a directory in the hard-disk where the ROS packages must reside in order to be usable by ROS, and it's usually named as "catkin-ws".
- **Node:** is a process that executes a ROS program. ROS nodes can be found out using this command: `rosnode list`
- **Roscore:** is a collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate. It is launched using the `roscore` command.
- **Topics:** are like kind of pipe of channel that transports information and then nodes can publish information to this topic or subscribe information from this topic, so we can communicate with the robots.

- **Service:** it allows coding any specific functionality for the robot and then providing it for anyone to call this service.
- **Parameter server:** is a dictionary that ROS uses to store the parameters.

ROS processes are represented as **nodes** in a graph structure, connected by edges called **topics**. ROS nodes can pass messages to one another through topics, make **service** calls to other nodes, provide a service for other nodes, or set or retrieve shared data from a communal database called the **parameter server**.

3.4.2 Gazebo

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At your fingertips are a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant community.

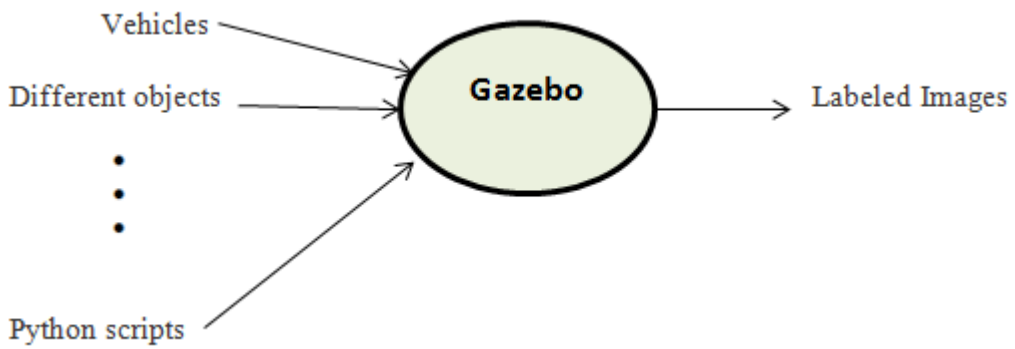


Figure 5: simulation illustration

3.5 Scenarios

Accident and no-accident scenarios were created to imitate the real situations that happen in the real life as much as possible. In order to organize our work, excel sheet was made with the most popular cases of accident and no-accident scenarios considering**:**

- Different environments.
- Types of vehicles.
- Crashes with different objects and with different sides.
- Crashes with simulated persons.

And we followed this excel sheet in the simulation part.

Figure 6: example of simulated no-accident image

Figure 7: example of simulated accident image.

3.6 Environment

This section describes the creation of the simulation environment starting from the models (static and dynamic objects) till ending with the overall city.

3.6.1 Terminology

World: The term used to describe the layout of robots, sensors, light sources, user interface components, physical properties, and objects (such as buildings, tables, and lights).

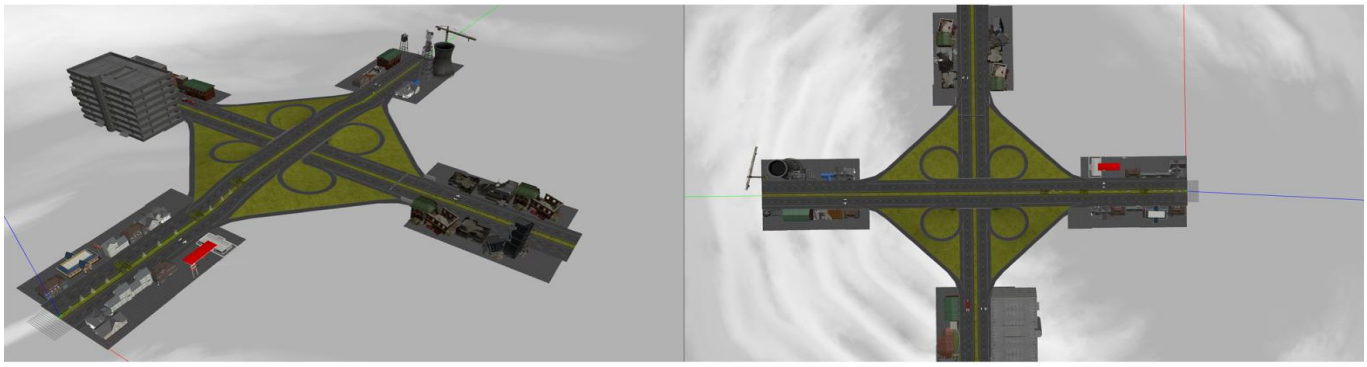
Static objects: Entities marked as static, those having the "<static>true</static>" element in the SDF file, are objects which only have collision geometry. All objects which are not meant to move should be marked as static.

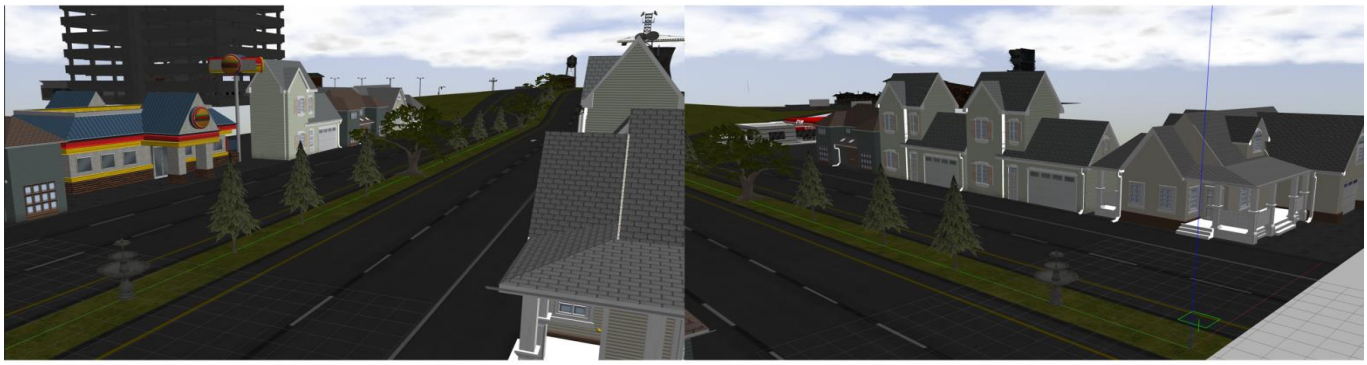
Dynamic objects: Entities marked as dynamic, either missing the "<static>" element or setting false in the SDF file, are objects which have both inertia and collision geometry.

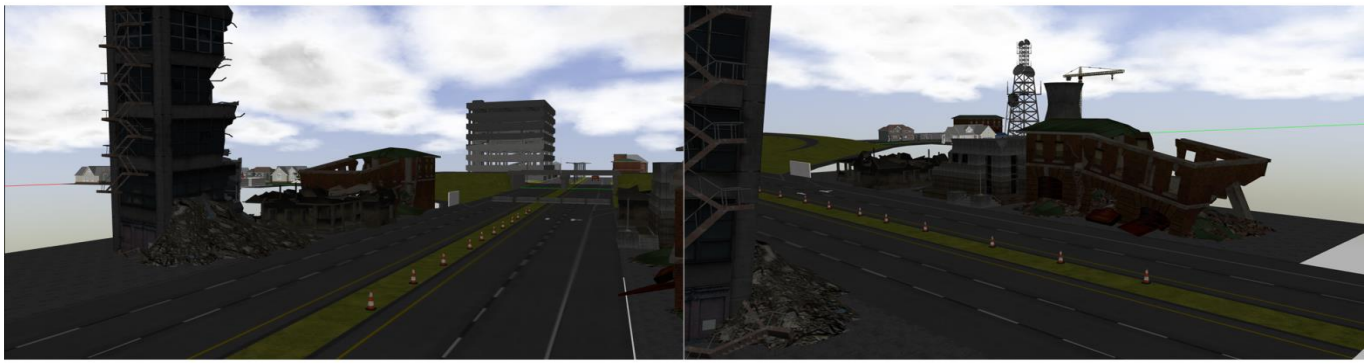
3.6.2 Project world

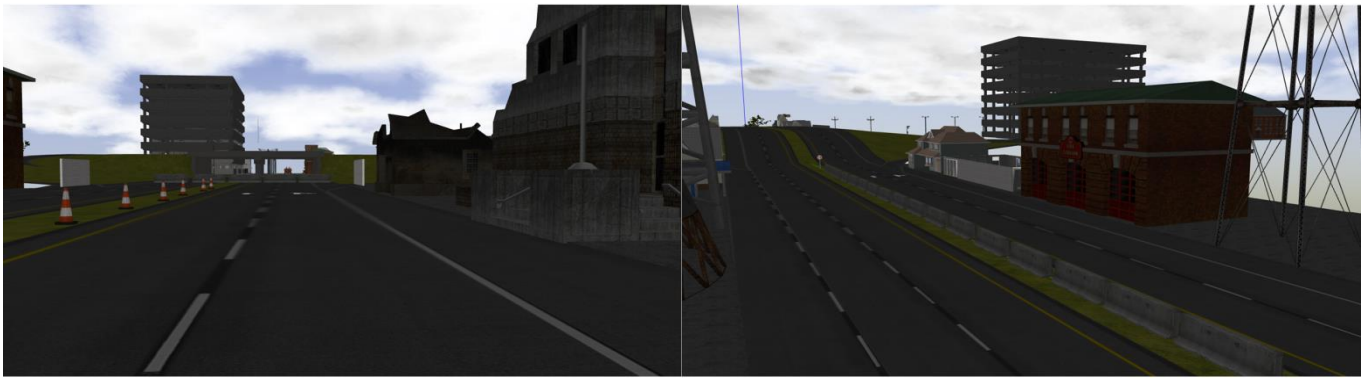
To build a city that looks like a real city in the real world:

- A new world was created including the sky and the sun to obtain more realistic environment.
- Building our city from:
  - "mcity" (a built-in model in gazebo).
  - Add more objects such as crossroads, buildings, restaurants, trees, traffic jam, road signs, persons and vehicles.

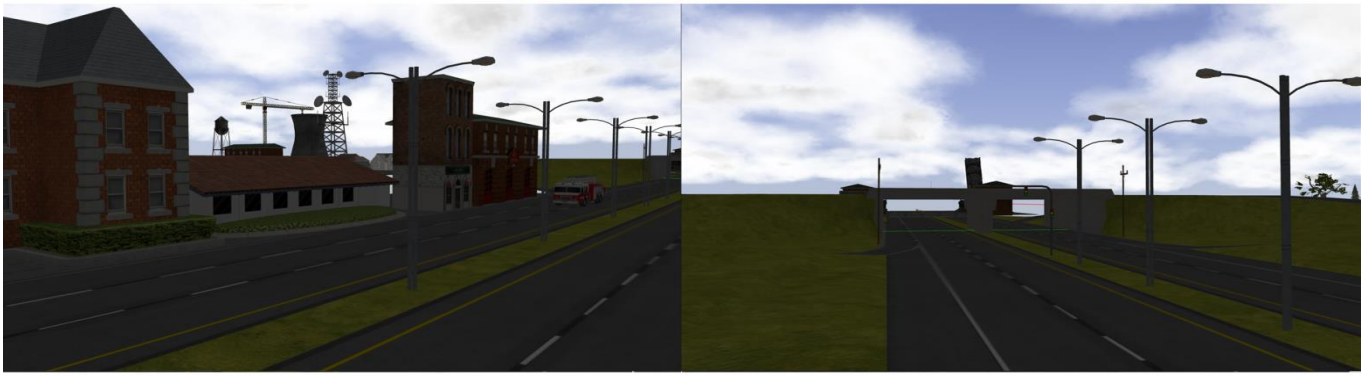









Grocery Store



3.6.3 Static objects

Attached below a table of all the models that were used in the final world.

Sun	fountain
Construction Cone	grey_wall
Fast Food	grocery_store
Gas Station	jersey_barrier
House 1	law_office
House 2	pine_tree
House 3	parking_garage
Lamp Post_clone	osrf_first_office
asphalt_plane	water_tower
cloverleaf_interchange	Construction Cone
cafe	oak_tree
collapsed_fire_station	jersey_barrier
collapsed_police_station	school
fire_hose_long	Lamp Post
fire_station	post_office
fire_truck	Radio tower

Table 3: Static objects in our environment

3.6.4 Dynamic objects

Mainly, there are two dynamic models (Ford, Prius) in our topic, with the inertia, wheels and steering parts. Also they have all the possible sensors we can use in a car as laser, camera, far sonar.

3.6.4.1 Prius

Package: “car_demo”.

Description: simulation of a “Prius” car in “mcity” using ROS Kinetic and Gazebo 8. ROS enabled the simulation to be developed faster by using existing software and libraries. The vehicle's throttle, brake, steering, and transmission are controlled by publishing to a ROS topic. All sensor data is published using ROS, and can be visualized with RViz debugging tool.

Taking the advantages of Gazebo's capabilities to incorporate the existing models and sensors. The world contains a new model of “mcity” and a freeway interchange. There are also models from the gazebo models repository including dumpsters, traffic cones, and a gas station.

Sensors: there is a 16 beam lidar on the roof, 8 ultrasonic sensors, 4 cameras, and 2 planar lidar.

Topics: we used two main topics.

- Front camera topic: “prius/front_camera/image_raw”.
- Velocity topic: “prius/cmd_vel”.

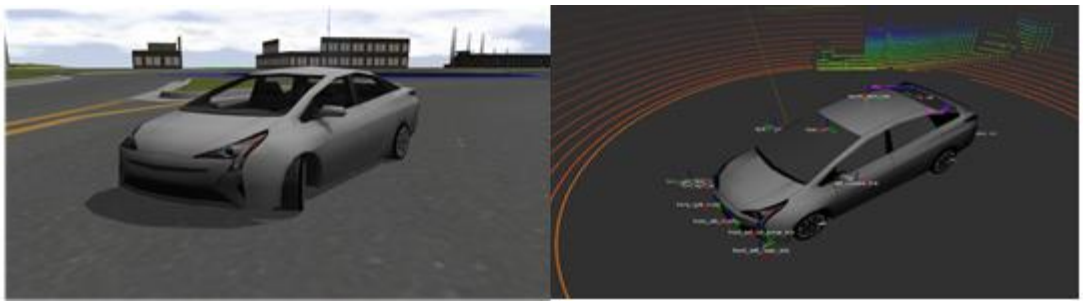


Figure 8: prius car.

3.7 Python scripts

There are two main python scripts were created to simulate our environment.

3.7.1 "camera.py" script

This script was used to capture images from the front camera of the simulation car in Gazebo.

Code description

1) Use "cv_bridge" message to convert the ROS images to Opencv2 images.
2) Use "sensor_msgs.msg" to import ROS image message.
3) Import cv2 to save the Opencv2 images as a jpeg with the required label either "0" for no-accident images or "1" for accident images.
4) Inside the main function, image topic "/prius/front_camera/image_raw" was defined to be subscribed by "image_listener" node.

3.7.2 "car_movement.py" script

Another way to control the car movement, its speed and directions is through this python script.

Code description

1) Use “geometry_msgs.msg” to import “Twist” message for linear and angular movement.
2) Inside the main function, “movement” node was created to publish on “/cmd_vel” topic to control the movement of the cars. So, we can move the cars in x,y “linear” directions and rotate the cars in z “angular” direction.

3.8 Data augmentation

We generated from gazebo environment about 5000 images contains different accident and non-accident scenarios; then using data augmentation we managed to double the total number of images. The type of data augmentation that used was flipping which was done using a python code to save a flipped version of each image. Figure 9 shows the original image and Figure 10 shows its flipped version.

Figure 9: Original image before data augmentation.

Figure 10: flipped image.

After that we divided them into two sets where each scenario was divided randomly by 80% for train and 20% validation so:

1. The training dataset contains 7560 images.
2. The validation dataset contains 2047 images.

For testing the model, we also generated 1032 image from the simulation environment which balanced between accident and non-accident scenarios.

After achieving a proper accuracy on the test dataset of gazebo, another dataset was collected for testing the model using real images from different streets in different location that contains accident and non-accident scenarios; this dataset is collected from online videos that was downloaded and sampled and calibrated using MATLAB code after that we managed to test the model on 1346 image.

Chapter 4 Deep Learning Model

This chapter includes the detailed architecture of the deep learning model which is used to achieve a proper accuracy on our dataset which is generated from gazebo and also taking into consideration achieving a proper accuracy on real dataset from real world.

4.1. Overview

4.1.1 Deep learning

In recent years, deep learning has garnered a huge success in many application domains. This new field of machine learning has been growing rapidly, and has been applied to most traditional application domains as well as some new areas that present more opportunities. Different methods have been proposed based on different categories of learning, including supervised, semi-supervised, and un-supervised learning. The experimental results show state-of-the-art performance using deep learning when compared to traditional machine learning approaches in many fields such as image processing, computer vision, speech recognition, machine translation, art, medical imaging, medical information processing, robotics and control, bio-informatics, natural language processing (NLP), cybersecurity, and many others.

4.1.1.1. Deep learning algorithms

- **Supervised learning**

The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X)$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data. It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers from the dataset; the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

- **Unsupervised learning**

Unsupervised learning is where you only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning above there are no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

- **Reinforcement learning**

Reinforcement learning refers to goal-oriented algorithms, which learn how to attain a complex objective (goal) or maximize along a particular dimension over many steps; for example, maximize the points won in a game over many moves. They can start from a blank slate, and under the right conditions they achieve superhuman performance. Like a child incentivized by spankings and candy; these algorithms are penalized when they make the wrong decisions and rewarded when they make the right ones.

4.1.2 CNN

Deep Convolutional Neural Networks (CNNs) are the state-of-the-art systems for image classification and scene understating. Deep learning algorithms provide a high performance in CNNs. CNNs are designed to recognize visual patterns directly from pixel images with minimal preprocessing. They can recognize patterns with extreme variability such as handwritten characters, and with robustness to distortions and simple geometric transformations.

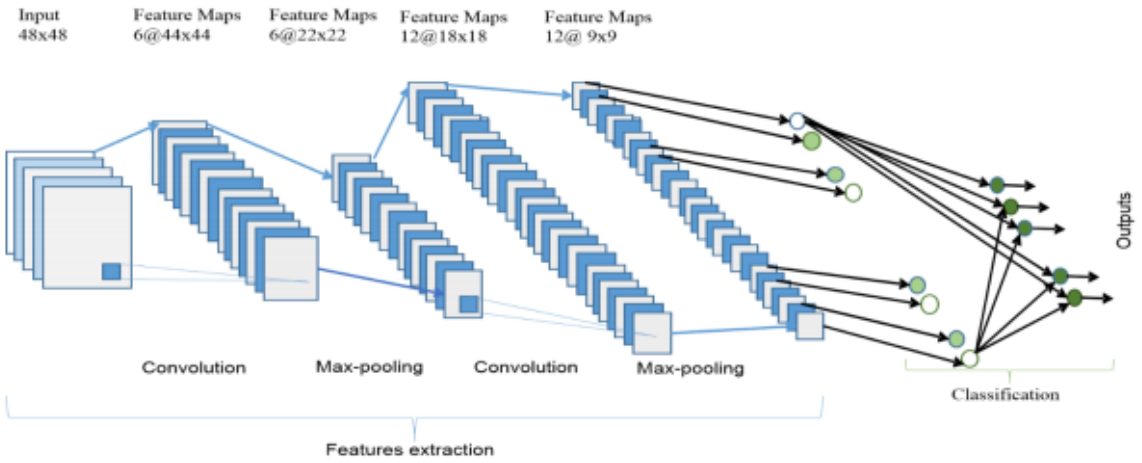


Figure 11: CNN architecture

A typical CNN structure consists of a feature extractor and a classifier. The feature extractor extracts an input image's features and sends them to the classifier. According to these features, the classifier decides the category that the input image belongs to as shown in Figure 11

4.2. AlexNet

AlexNet is a convolutional neural network that has had a large impact on the field of machine learning. It was one of the first deep networks to push ImageNet Classification accuracy by a significant stride in comparison to traditional methodologies. It is trained on more than million images from the ImageNet database and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals then competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. The network achieved in the competition a winning top-5 test error rate of 15.3% compared to 26.2% achieved by the second-best entry with 41% margin which is a very large margin improvement than before as shown in Figure 12 below.

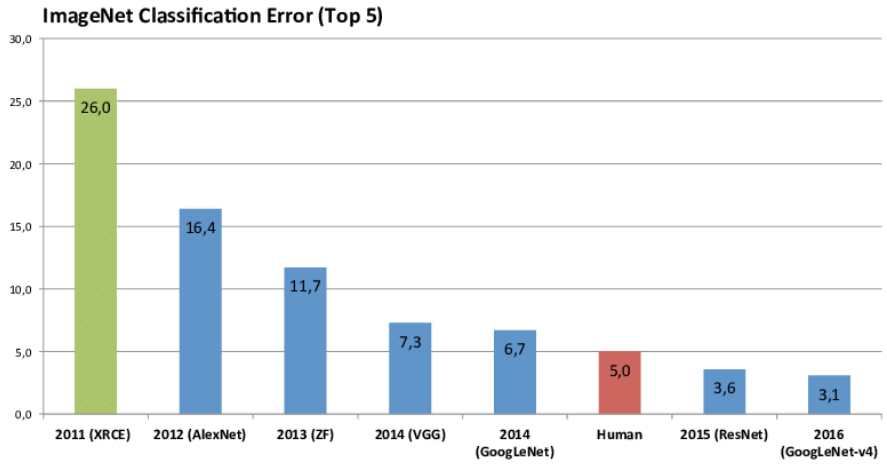


Figure 12: ImageNet Classification Error.

So, it was the start of a revolution not just for computer vision, but for artificial intelligence in general. Researchers working on speech recognition, natural language processing, and every other kind of intelligent data analysis have adopted approaches similar to AlexNet's.

4.3. Network architecture

This network which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of them are followed by max-pooling layers, and three fully-connected layers with a final 1000-way soft-max as shown in Figure 13.

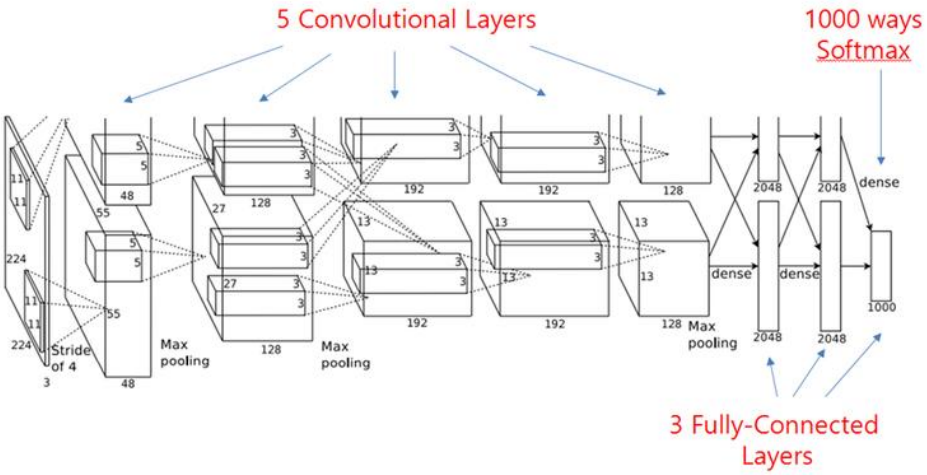


Figure 13: AlexNet architecture.

The architecture used in the project is similar to the design of AlexNet paper but with some modifications to fit our problem as shown in Figure 14.

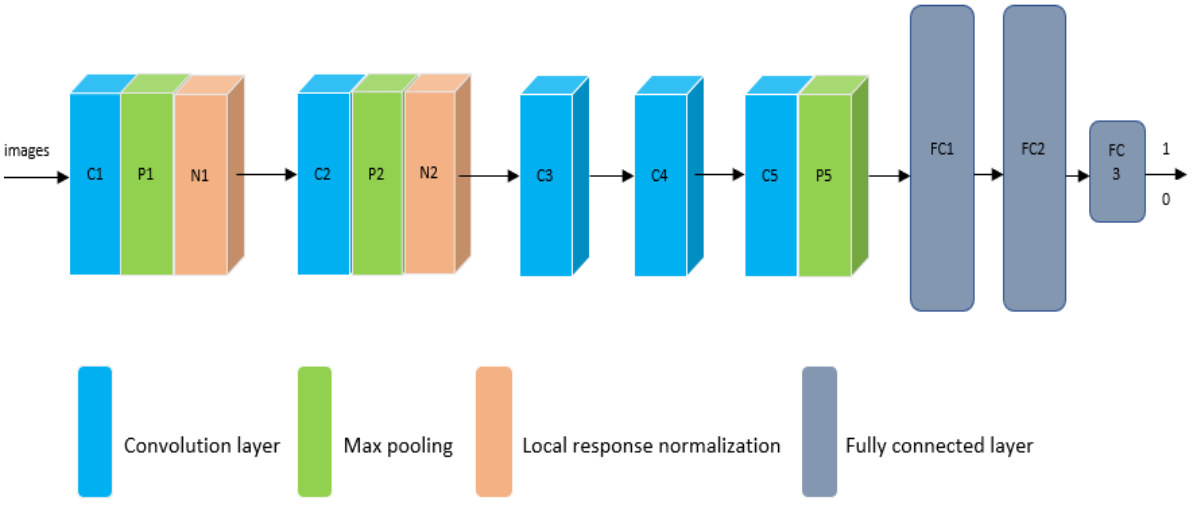


Figure 14: modified AlexNet layers.

Our network specifications:

- It is composed of 5 convolution layers followed by 3 fully connected layers as shown in Figure 14.
- The input of the network is the images provided from the simulation of size 224*224 JPEG.
- The output is mapped to be either '0' or '1', which indicates if there is an accident in the image or not.
- The total number of parameters in the network is 143.47405 million parameters.
- Local response normalization is applied to the first two convolution layers.

Table 4 shows the details about the sizes of the convolution layers:

Layer number	Filter size	Number of filters	Stride	Input size	Output size (before max-pooling)	Padding	Max-pooling	Number of parameters	Activation function
1	11x11	96	4	224x224x3	56x56x96	Same	exists	34944	ReLU
2	5x5	256	1	27x27x96	27x27x256	Same	exists	614656	ReLU
3	3x3	384	1	13x13x256	13x13x384	Same	_	885120	ReLU
4	3x3	384	1	13x13x384	13x13x384	Same	_	1327488	ReLU
5	3x3	256	1	13x13x384	13x13x256	Same	exists	884992	ReLU

Table 4: Convolution layers specifications.

Table 5 shows the max pooling layers:

Convolution layer number	Filter size	Stride	Padding	Input size	Output size
1	3x3	2	Valid	56x56x96	27x27x96
2	3x3	2	Valid	27x27x256	13x13x256
5	3x3	2	Valid	13x13x256	6x6x256

Table 5: Max pooling layers specifications

Table 6 shows the details about the sizes of the fully connected layers:

Number of neurons	Dropout layer	Number of parameters	Activation function
1024	exists	138675200	ReLU
1024	exists	1049600	ReLU
2	_	2050	Softmax

Table 6: Fully connected layers specifications.

4.3.1. Convolution layer

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. The number of convolutional layers has no limits depending on the desired application of the network. Conventionally, the first Convolution Layer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network, which has the wholesome understanding of images in the dataset, similar to how we would.

- **Convolution process**

The input image is represented by three channels, each channel is representing one of the basic colors (red, green, blue) as a 2D matrix containing pixels values as shown in Figure 15.

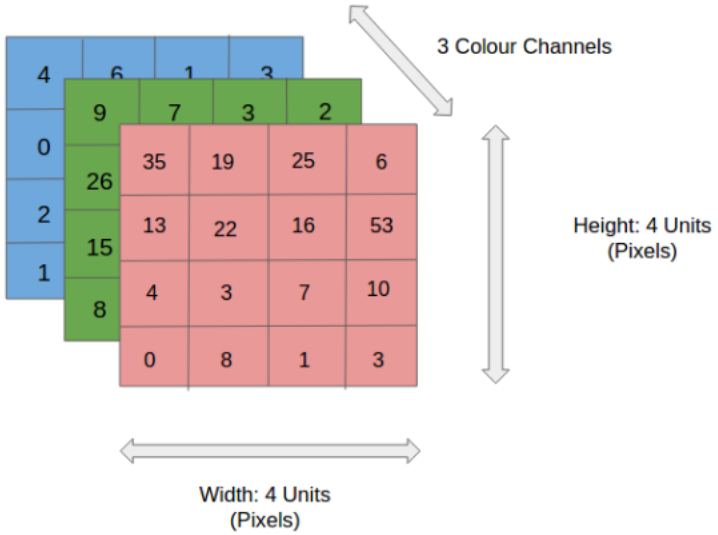


Figure 15: RGB image representation.

Convolution is just a matrix multiplication operation between the input image and the filter as shown in Figure 16. As the filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image. These multiplications are all summed up to produce a single output. The result is a matrix called the Convolved Feature Map.

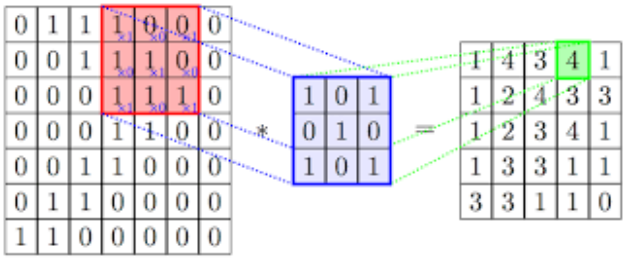


Figure 16: Convolution process.

The formula to calculate the output size from the convolution layer is given below:

$$\text{Output size} = \frac{image\ size - filter\ size + 2 * padding}{stride} + 1$$

where the size can be calculated separately for both height and width if the input image is not square.

- **Filters**

Each convolution layer has its own filters. Number of filters varies from one layer to another as going deeper in network needs more features. Filters have learnable parameters which are responsible of abstracting important features from the input as edges, colors and curves as shown in Figure 17.

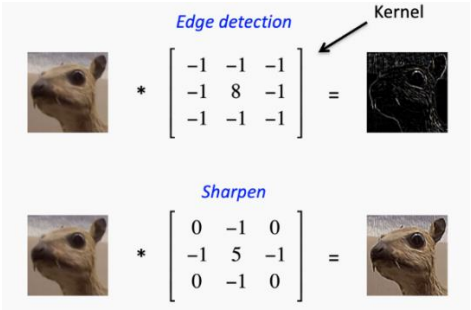


Figure 17: Some common filters.

A CNN learns the values of the parameters of these filters on its own during the training process. However, other parameters are still need to be specified such as number of filters, filter size, architecture of the network before the training process.

- **Stride**

Stride is the number of pixels shifts over the input image. So, when the stride is 1 then the filter moves 1 pixel at a time and it moves 2 pixels when the stride is 2 and so on. Figure 18 shows the convolution operation with a stride of 2.

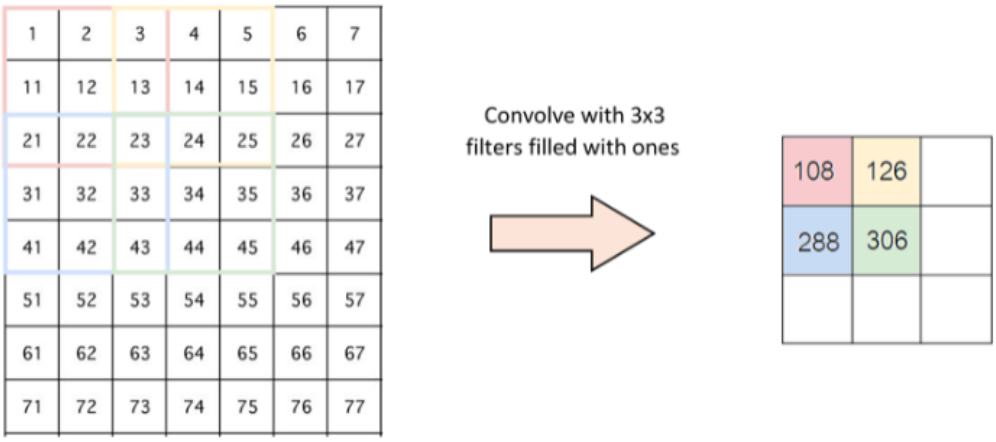


Figure 18: Convolution with stride of 2.

- **Zero padding**

Sometimes filter does not perfectly fit the input image. It will be convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes as we have three options:

- Pad the picture with zeros so that its size can fit the filter.
- Pad the picture with zeros to exactly preserve the spatial size of the input so the output has the same size as the input this called the same padding.
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

4.3.2. Pooling layer

Pooling is the process of extracting the features from the image output of a convolution layer. This will also follow the same process of sliding over the image with a specified kernel size. There are two types of pooling:

- **Max Pooling**

Max Pooling is being used widely and it will just keep the highest number in the pool and discard the rest. By getting the highest value in each pool we will be getting the significant features of the image. Max pooling extracts the most important features like edges as shown in Figure 19.

- **Average Pooling**

Average pooling will just keep the average number of the total numbers in the pool. Average pooling extracts features so smoothly as shown in Figure 19. Average pooling sometimes can't extract good features because it takes all into count and results an average value which may not be important for object detection type tasks.

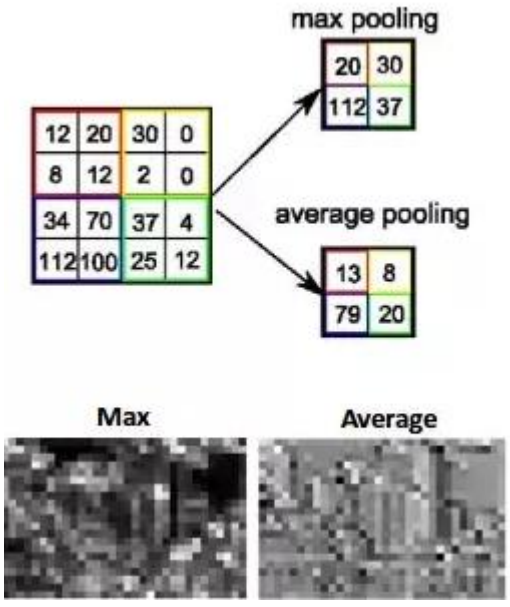


Figure 19: Average and Max pooling.

So, after convolution and pooling we will be having an image which is small and having all the features of the original image. Also, the number of parameters in the network is very less because each pixel in the resultant image is connected to the pixels in the part of the original image (size of the pool) and not all the pixels of the original image. This will improve the model performance and accuracy.

4.3.3. ReLU

Neural networks are used to implement complex functions, and non-linear activation functions enable them to approximate arbitrarily complex functions. Without the non-linearity introduced by the activation function, multiple layers of a neural network are equivalent to a single layer neural network.

ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function. It is used at the end of convolution layers and fully connected layers to add non-linearity. Figure 20 shows ReLU activation function which can be represented by the following formula:

$$f(x) = \max(0, x)$$

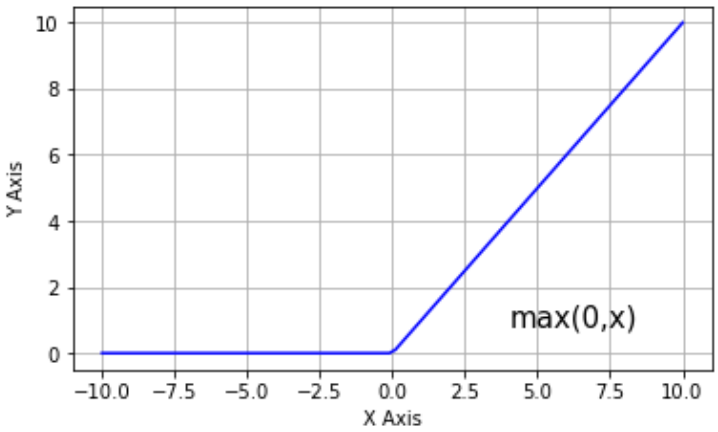


Figure 20: ReLU activation function.

There are other types of activation functions like hyperbolic tangent function and sigmoid function, but ReLU is the most common used in convolutional neural networks and also in deep neural networks in general because it is faster to train and saves computational resources.

The hyperbolic tangent and sigmoid functions are much slower than ReLU because:

- They are saturating activation function as shown in Figure 21 so their derivative becomes very small in the saturating region therefore the updates to the weights almost vanish, which is called vanishing gradient problem.
- They are complex and computationally expensive due to the computation of the exponential function.

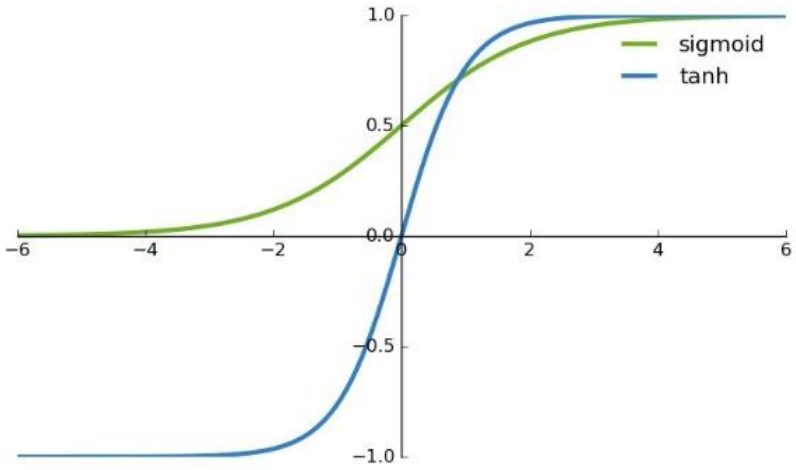


Figure 21: Sigmoid and Tanh activation functions.

Figure 22 shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network. The figure shows that the network with ReLU which is solid line reaches a 25% training error rate six times faster than its equivalent network with tanh in dashed line where the learning rates for each network were chosen independently to make training as fast as possible.

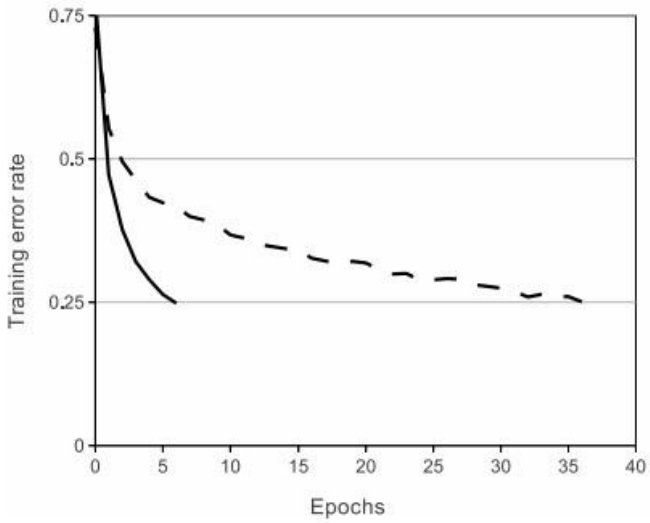


Figure 22: Training error rate of ReLU and Tanh.

4.3.4. Local response normalization

Local Response Normalization (LRN) was first introduced in AlexNet architecture where the activation function used was ReLU as opposed to the more common tanh and sigmoid at that time. This layer is useful when we are dealing with ReLU neurons because ReLU neurons have unbounded activations and we need LRN to normalize that. The high frequency features are needed to be detected with a large response. If we normalize around the local neighborhood of the excited neuron, it becomes even more sensitive as compared to its neighbors.

The other reason for using LRN was to encourage lateral inhibition. It is a concept in Neurobiology which refers to the capacity of a neuron to reduce the activity of its neighbors. In DNNs, the purpose of this lateral inhibition is to carry out local contrast enhancement so that locally maximum pixels values are used as excitation for next layers.

LRN is a non-trainable layer that square-normalizes the pixel values in a feature map in a within a local neighborhood. There are two types of LRN based on the neighborhood defined. The first type is Inter-Channel LRN which is originally what the AlexNet paper used. In which the neighborhood is defined across the channel. However, the other type is Intra-Channel LRN where the neighborhood is extended within the same channel only.

Figure 23 illustrates the difference between the two types which shows that the Intra-channel LRN uses 2D neighborhood around the pixel under-consideration as opposed to 1D neighborhood in Inter-Channel LRN.

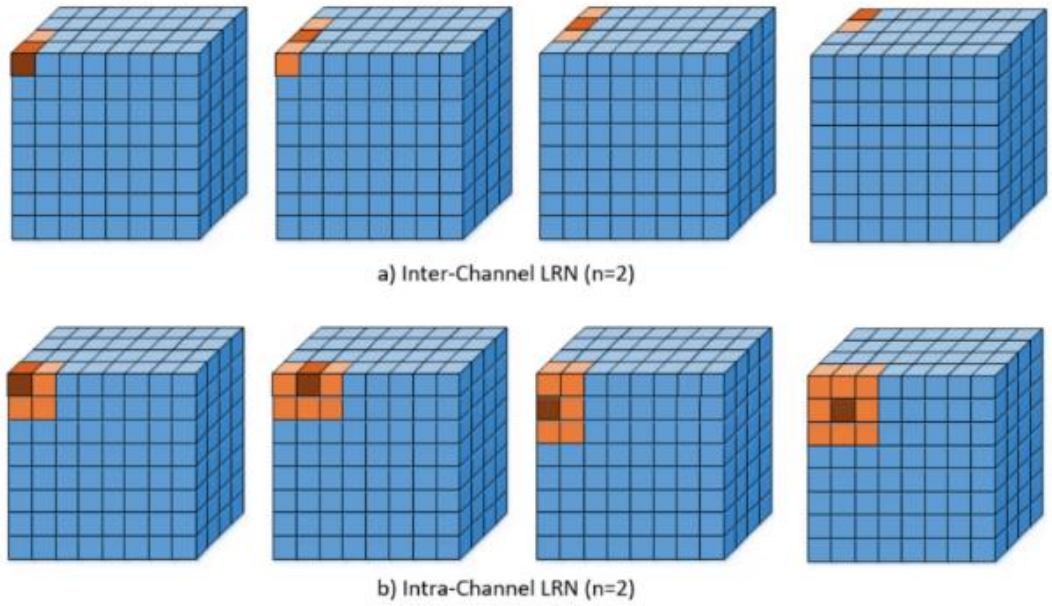


Figure 23: Types of LRN.

- **Inter-Channel LRN:**

For each feature map pixel in (x, y) position, the normalization is carried out in a depth dimension and can be given by the following formula:

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

Figure 24 illustrates the above formula where:

- *i* indicates the output of filter after ReLU.
- *a(x,y)* , *b(x,y)* are the pixel values at *(x,y)* position before and after normalization respectively.
- N is the total number of kernels in the layer.
- The constants *(k, α, β, n)* are hyper-parameters whose values are determined using a validation set.
- *k* is used to avoid any singularities like division by zero, *α* is used as a normalization constant, while *β* is contrast constant.
- The constant *n* is used to define the neighborhood length i.e. how many consecutive pixel values need to be considered while carrying out the normalization.
- The case of *(k, α, β, n) = (0,1,1, N)* is the standard normalization.

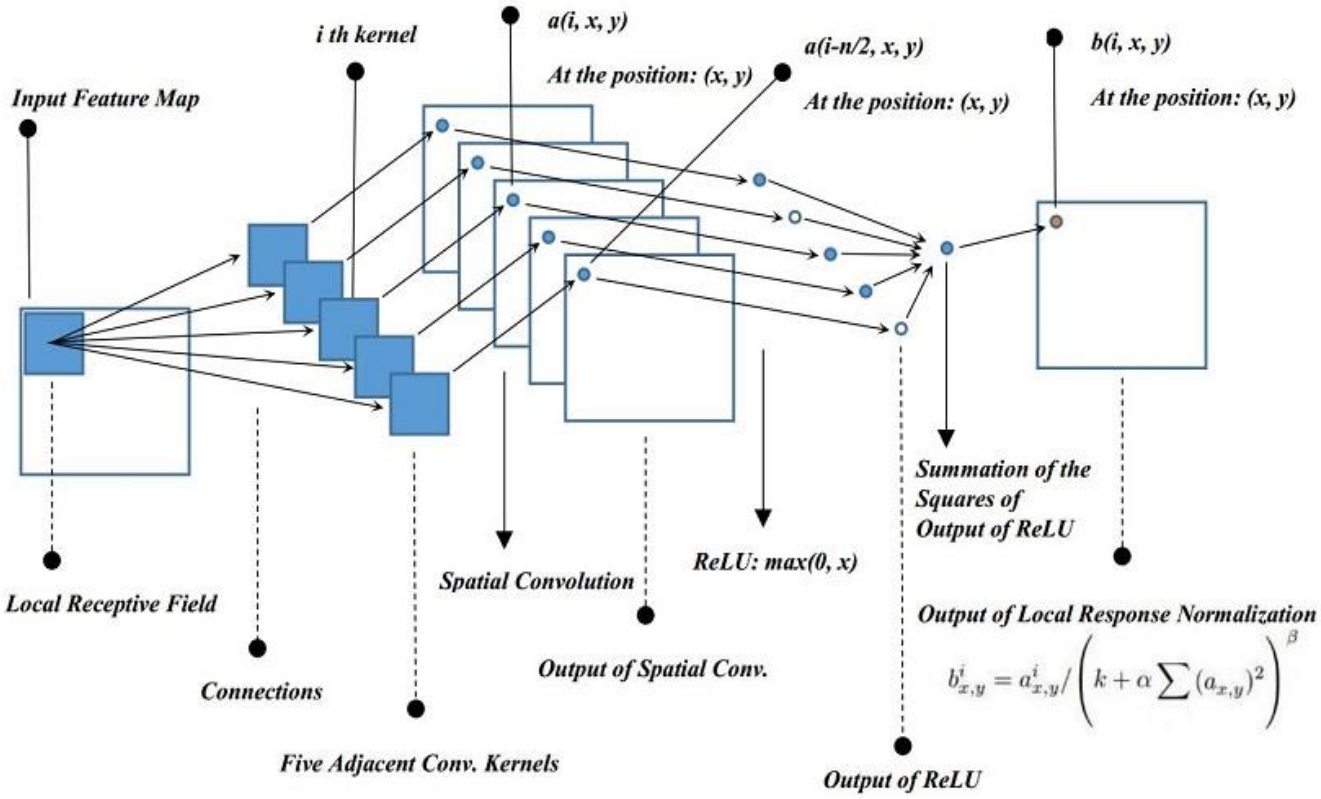


Figure 24: Inter-Channel LRN.

- **Intra-Channel LRN:**

For each feature map pixel in (x, y) position, the normalization is carried out using the following formula where (W, H) are the width and height of the feature map:

$$b^k_{x,y} = a^k_{x,y} / \left(k + \alpha \sum_{i=max(0,x-n/2)}^{min(W,x+n/2)} \sum_{j=max(0,y-n/2)}^{min(H,y+n/2)} (a^k_{i,j})^2\right)^{\beta}$$

The used type in our network architecture in the first two convolution layers is Inter-Channel LRN with the following hyper-parameters:

- Depth radius (*n*) = 5
- k = 2
- α = 1e-4
- β = 0.75

4.3.5. Fully connected layer

The fully connected layer is a traditional MLP with Soft-max activation function. The network can be divided for classification problem into two parts:

- **Feature extraction**

  The convolutional layers are serving the same purpose of feature extraction. CNNs capture better representation of data and hence we don't need to do feature engineering.

- **Classification**

  After feature extraction we need to classify the data into various classes, this can be done using a fully connected neural network. We generally end up adding FC layers to make the model end-to-end trainable.

In our deep learning classifier fully connected layers allow the network to flatten the feature map to end up with two neurons that help defining the class that the input image belongs to as shown in Figure 25.

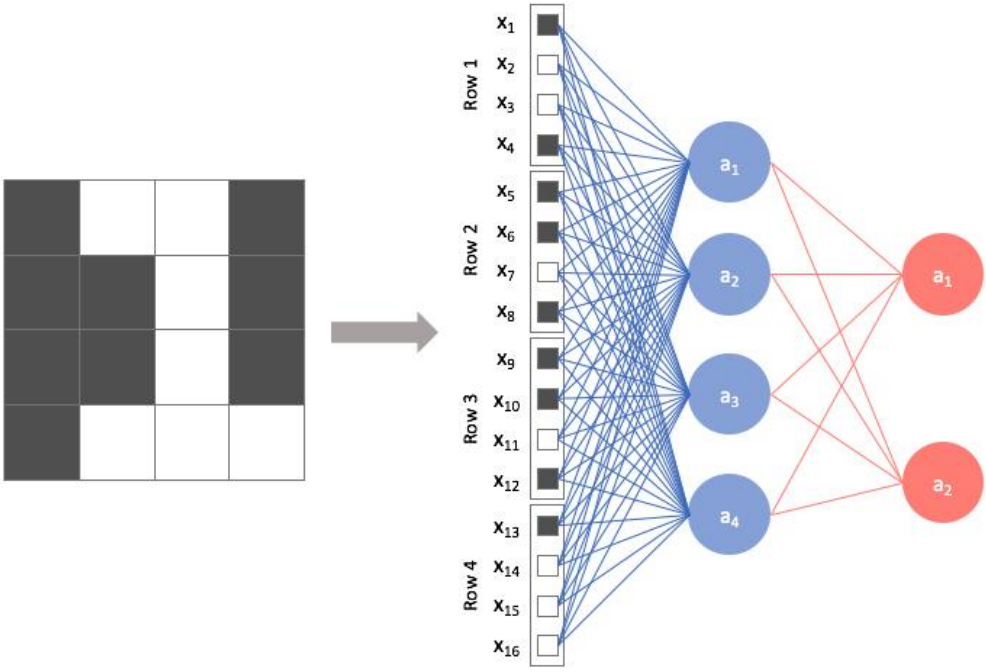


Figure 25: Fully connected layers.

4.3.6. Softmax

Softmax is used as the output function of the last layer in neural networks (if the network has n layers, the n-th layer is the Softmax function). This fact is important because the purpose of the last layer is to turn the score produced by the neural network into values that can be interpreted by humans.

Softmax is a function that takes the input vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities as shown in the equation below

$$P(y = j|\, x) = \frac{e^{z_{y_j}}}{\sum_{k=1}^{k=K} e^{z_k}} \,, j = 1,2, \dots \dots \dots K$$

$$where\ z_j = x^T w\ and\ K\ is\ the\ total\ number\ of\ classes\ in\ the\ dataset.$$

So, after applying Softmax, the output is probabilities as shown in Figure 26.

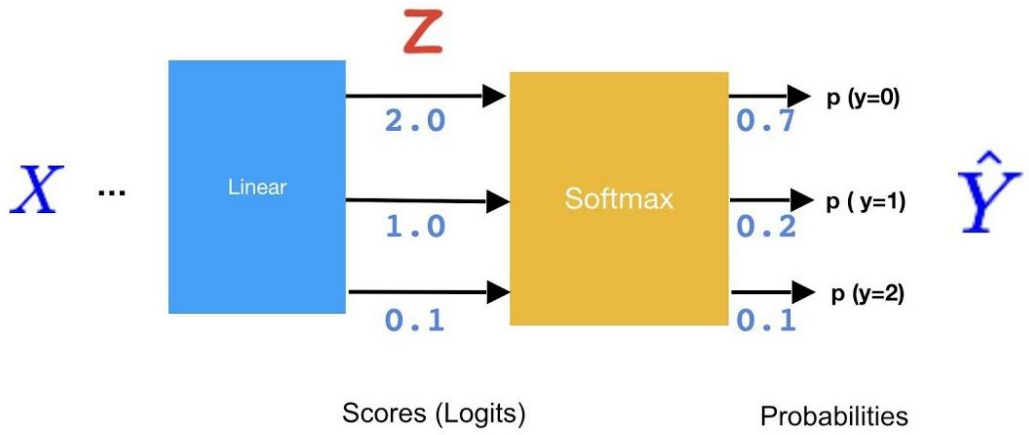


Figure 26: Softmax output.

The highest probability represents which class the input image belongs to. To return 0 or 1, the output of the Softmax is applied to argmax function which returns the index of the highest probabilities and makes it easy to deal with as the predicted label.

4.3.7. Dropout

Deep neural network with multiple layers and large number of neurons suffer from over fitting during the training phase. Overfitting happens when the model fits too well to the training set. It then becomes difficult for the model to generalize to new examples that were not in the training set. One of the main Steps for reducing overfitting is the dropout regularization method which was introduced for first time at AlexNet.

Dropout consists of setting to zero the output of each hidden neuron with probability 0.5 as shown in Figure 27. The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in backpropagation. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. So, it is forced to learn more robust features. At test time, all neurons are used but multiply their outputs by 0.5. Dropout layer is applied in the first two fully connected layers.

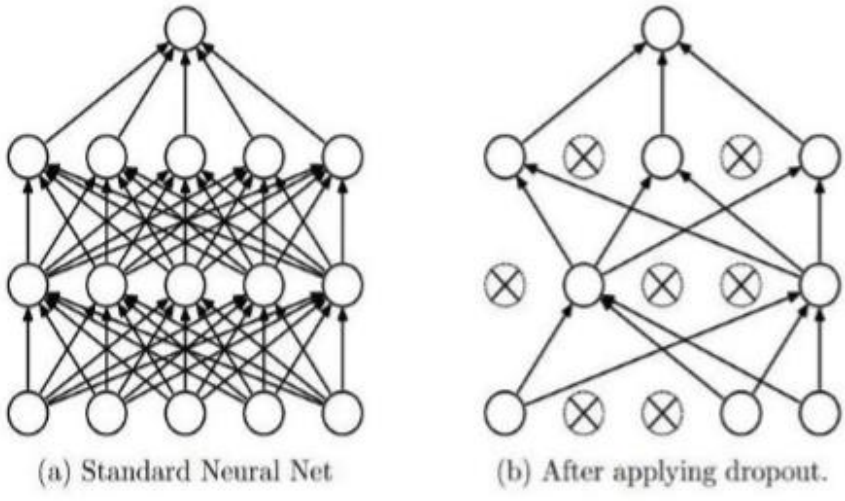


Figure 27: Applying dropout.

4.4. Preprocessing

Image processing is divided into analog image processing and digital image processing. Digital image processing is the use of computer algorithms to perform image processing on digital images. Digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data. The aim of digital image processing is to improve the image data (features) by suppressing unwanted distortions and/or enhancement of some important image features so that our AI-Computer Vision models can benefit from this improved data to work on.

Steps of image preprocessing in our network

- Normalizing pixels values of the input image

An image is nothing more than a two-dimensional array of numbers (or pixels) ranging between 0 and 255. Normalization is done by dividing each value by 255 so the values range becomes between 0 and 1. This allow easier and faster computation process in the conv layers.

- Resizing the input image

Some images captured by gazebo camera and fed to our network vary in size; therefore, we should establish a base size for all images fed into our network. This is done in our python code by using the function cv2.resize and passes the size 720*720 which is applied for all images in our dataset. Figure 28 shows the original image while the resized image is shown in Figure 29.

Figure 28: Original image from gazebo.

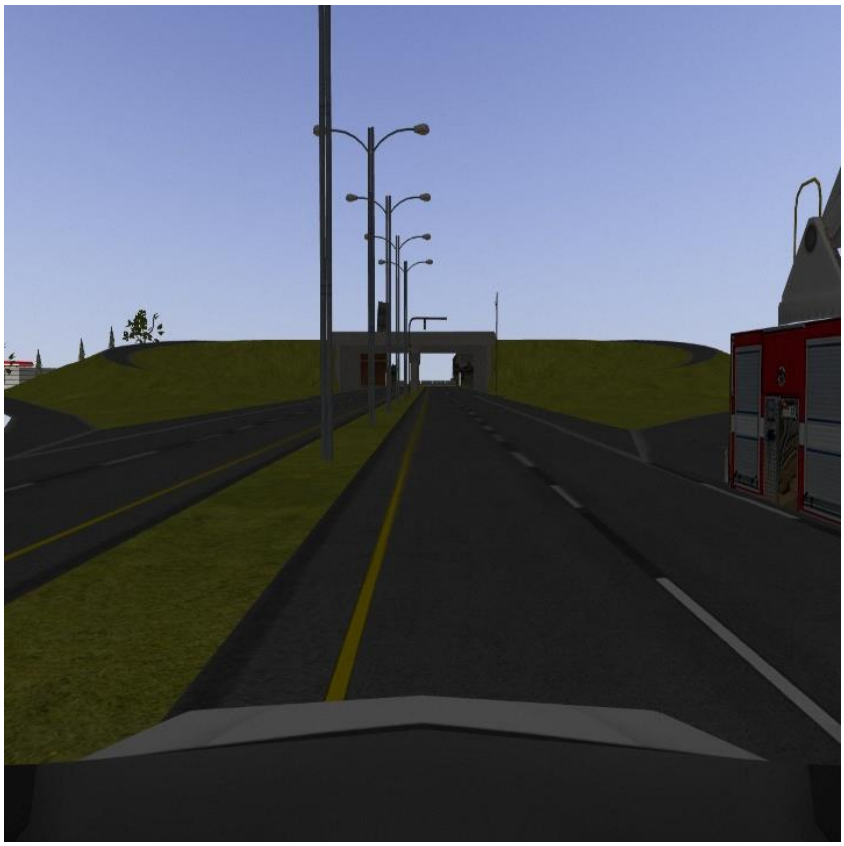


Figure 29: Resized image.

- Cropping unwanted parts

For better features extraction, the unwanted parts from can be removed from our dataset like the hoods as shown in Figure 30. the final step is resizing all the images in the dataset to the default size of AlexNet input which is 224*224 as shown in Figure 31.

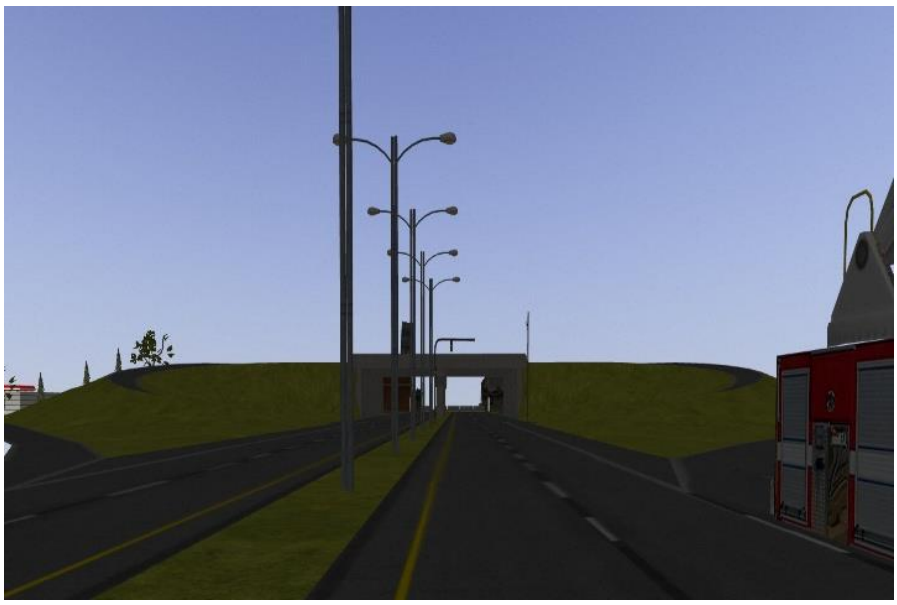


Figure 30: Clear image without hoods.

Figure 31: Final image.

4.5. Accuracy

Our problem is a supervised learning problem so the accuracy was calculated by comparing the predicted label with the original label of the test dataset. Training process was done on 7560 images and 2047 images for validation.

- **Data preparation**

  - First trial, the whole data of training and validation (about 9607 images) was separated randomly to train by 80% and validation by 20%.
  - Second trial, the accident data and the no accident data were separated then divided randomly by 80% for train and 20% validation.
  - Third trial, to ensure that every scenario is being selected in training set and validation set, data was separated in scenarios and each scenario was divided randomly by 80% for train and 20% validation.

- **Model hyper parameters:**

After modifying the network hyper parameters several times and apply the training and validation set from third trial of data preparation, the following hyper parameters is chosen to achieve the best accuracy on the test dataset:

  - Number of epochs is equals to 25.
  - Fully connected layers consist of 1024 neurons.
  - Batch size equals to 120.
  - Keep probability equals to 0.5.
  - Learning rate equals to 1e-4

- **Results**

The model of the best testing accuracy achieved training and validation accuracies above 90%. It is tested using 1032 images balanced between accident scenarios and non-accident scenarios that generated from gazebo environment and achieved the following results:

  - 76.7% test accuracy for the whole test dataset.
  - 68.9% test accuracy for the images containing accident scenarios.
  - 84.8% test accuracy for the images containing non-accident scenarios.

Chapter 5 V2V communication

In this chapter, we will describe the third and the last phase in our project which is V2V communication and how we employ this technology to achieve the required functionality.

5.1 Introduction

5.1.1 Why do we need this technology?

Most accidents occur because the driver can only see, with the sensors and the current electronic driver aids, as far as the vehicles directly in front of him/her, behind him/her, or on either side. A competent driver might notice more than one car ahead or behind, notice the signal lights and act preemptively to prevent any sudden actions or accidents. However, sometimes this isn't enough. If any sudden action was taken faster than the driver's reaction such as a vehicle coming in a very high speed next to him/her or realizing there's a huge obstacle when the car is too near to take the needed precautions, this will lead to dangerous consequences. As a result, there has to be another solution that will car itself notice the sudden changes to take precautions if the driver couldn't. Also there has to be a solution to make the ability to see more than 2 vehicles ahead or behind to alert the driver of the changes that happen a little further than his/her sight so that the driver can act smoothly and preemptively. In addition, for the upcoming cars it is a waste of time to spend long time in the place of accident due to the traffic so sure there is a solution of that.

One of the technological advances that could solve this problem is vehicle to vehicle communication.

5.1.2 about V2X

Vehicle to vehicle communication is, as its name describes, a way for vehicles to send and receive signals to each other to explain their location, speed and direction. If there was a car that decided to change lanes and the driver didn't pay attention to the other cars who want to do the same, the car that falls behind in line with 3-4 cars between will send signals to this vehicle to inform it to wait until it passes to prevent future possible accidents. That way, the car can know what other out-of-sight cars, are doing or about to do. In addition to that, the communication will not be between vehicles only, but between vehicles and infrastructure as well (V2I), reducing any human errors that lead to accidents.

V2X communications are being to be important technology for achieving autonomous driving and standardized in various countries.

Figure 32: V2V communication signals.

5.1.3 Standardized V2X protocols

Since, V2X requires devices and vehicles of different manufacturers communicate with each other, there has to be a standard that all companies and manufacturers will follow. That's why IEEE developed the 802.11p standard which explains the physical and mac layers of vehicular transceivers. That way, any other European or American standards developed, will have to be based on the lower-level IEEE 802.11p standard, to ensure the compatibility of different devices communicating with each other.

5.2 Standard IEEE-802.11P overview

This standard is developed by IEEE (Institute of electrical and electronics engineers) organization to describe telecommunications and information exchange between systems Local and metropolitan area networks and it mainly determines Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

The standard of IEEE 802.11 has more than one amendment, but our project is following mainly amendment **IEEE STD 802.11P^{Tm}-2010**: Wireless Access in Vehicular Environments (Amendment 6). Specifically it is an implementation for **orthogonal frequency division multiplexing (OFDM) PHY specification** part in the standard.

5.3 Tools Used

5.3.1 Software Tool (GNU radio)

In this section, we'll describe GNU radio which is very important software that was used in the V2V PHY layer implementation, we'll first make an overview on it and describe its general usage, and then we'll go through how it was used in this phase of project.

5.3.1.1 Overview

GNU Radio is a free software development toolkit that provides signal processing blocks to implement software-defined radios and signal-processing systems. It can be used with external RF hardware to create software-defined radios, or without hardware in a simulation-like environment.

The GNU Radio software provides the framework and tools to build and run software radio or just general signal-processing applications. The GNU Radio applications themselves are generally known as **'flow graphs'**, which are a series of signal processing blocks connected together, thus describing a data flow.

These flow graphs can be written in either C++or the Python programming language. The GNU Radio infrastructure is written entirely in C++, and many of the user tools are written in Python.

5.3.1.2 Block diagrams of GNU radio

Wi-Fi Physical hierarchy

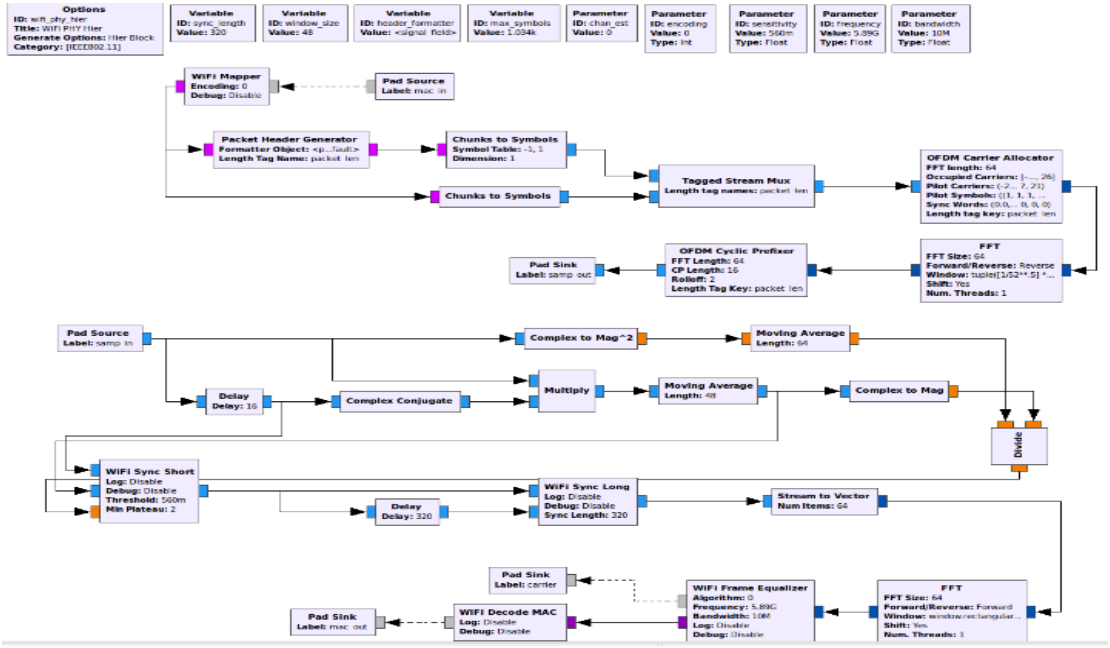


Figure 33: Wi-Fi Physical hierarchy.

The previous Figure 33 shows the blocks of the physical hierarchy in details and as explained in the standard.

Wi-Fi transmitter

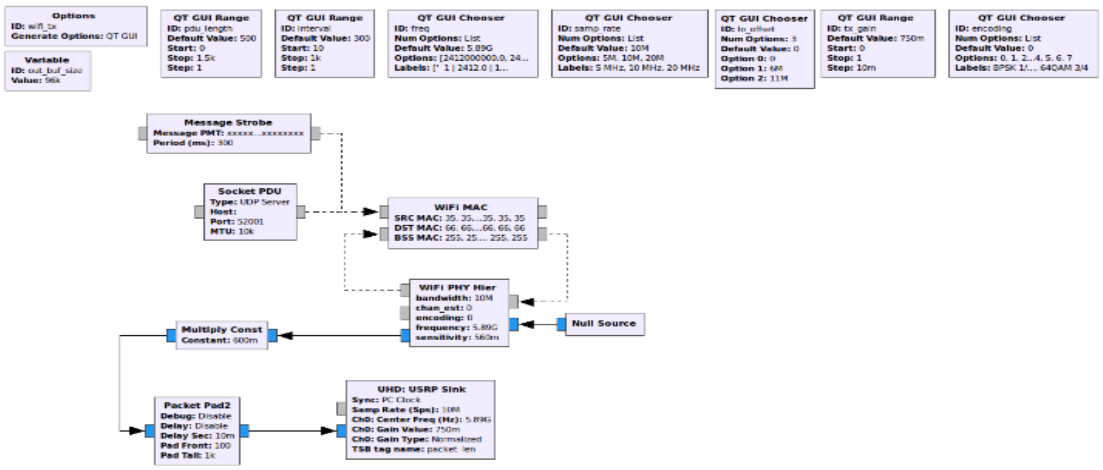


Figure 34: Wi-Fi transmitter.

The physical hierarchy is all inserted into a single block called Wi-Fi PHY hierarchy. This block diagram shows the insertion of data into the MAC layer, then into the physical layer up to the USRP block which puts the data on the channel to be sent. Figure 34 shows Wi-Fi transmitter block diagram.

Wi-Fi Receiver

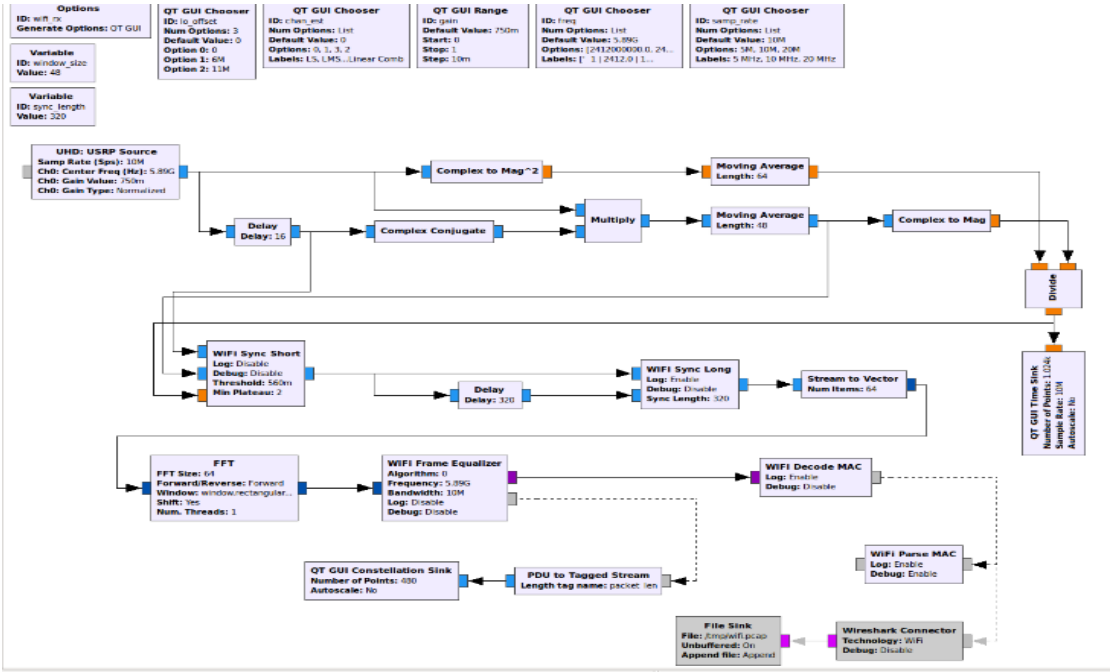


Figure 35: Wi-Fi Receiver.

This block diagram takes the receiver part of the physical hierarchy to receive the data from the channel first using the USRP block, then recover the data from the channel and start de-modulating, de-interleaving and de-scrambling the data. Figure 35 shows Wi-Fi receiver block diagram.

Wi-Fi transceiver

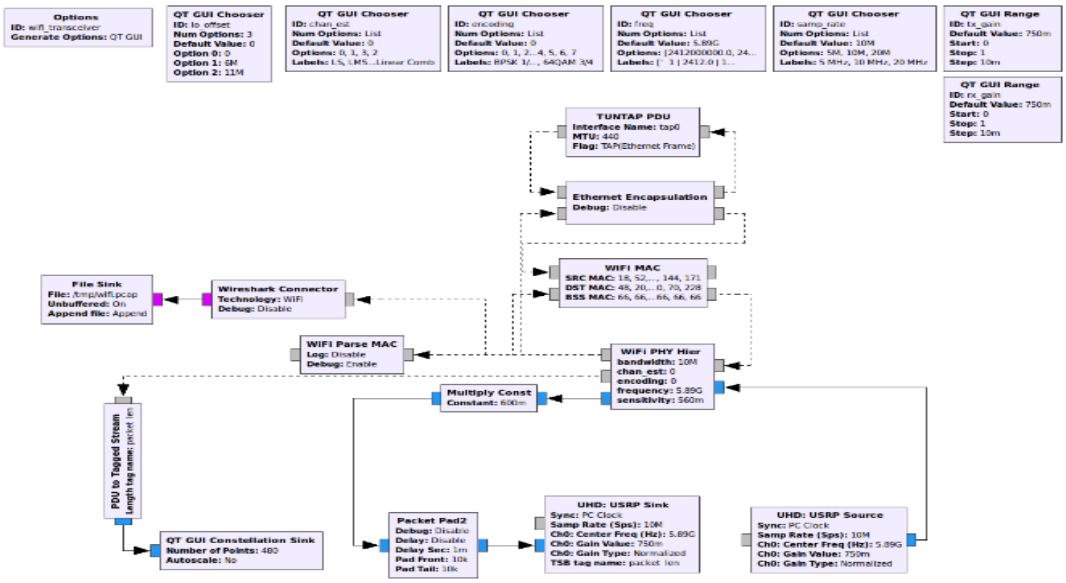


Figure 36: Wi-Fi transceiver

To show both sides in one block diagram, this block diagram shows the transmitter, the channel and the receiver. The channel here is created using USRP. Figure 36 shows Wi-Fi transceiver block diagram.

Wi-Fi loopback

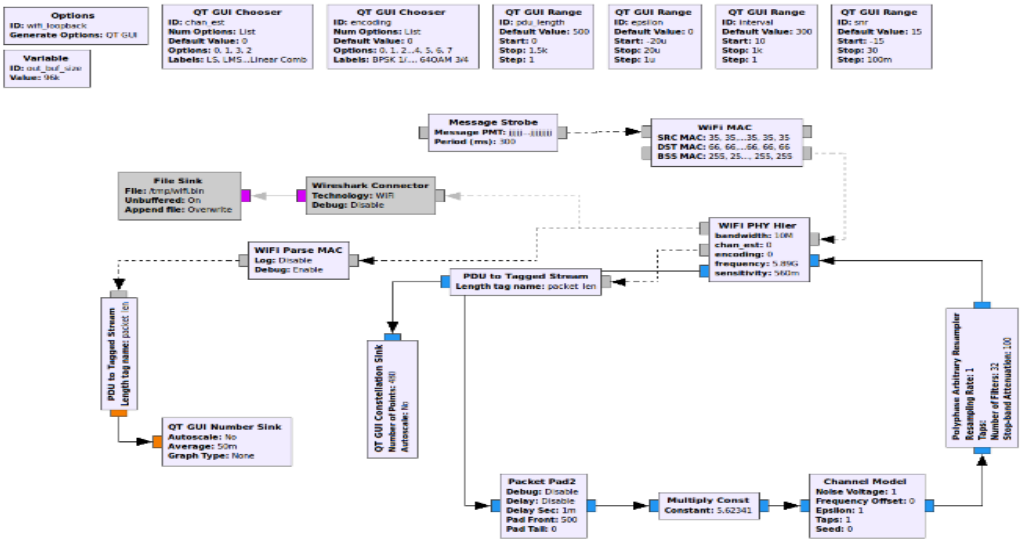


Figure 37: Wi-Fi loopback.

Loopback also shows both sides in one block diagram, which are the transmitter, the channel and the receiver. The difference from transceiver is that the channel here is virtual channel model. Figure 37 shows Wi-Fi loopback block diagram.

5.3.1.3 Usage in the project

In order to employ GNU radio in this phase of project, we need the Wi-Fi Transmitter and the Wi-Fi Receiver block diagrams to transmit data through the USRP channel, but we added some modifications in the block diagrams to achieve required functionality.

Modifications added to the Wi-Fi Transmitter block diagram:

1- Owing to V2V communication phase isn't a standalone phase as it receives data from the previous phase from it which is classifier model results, a file source is needed, instead of message strobe, to put the text file generated from classifier model in it.

2- Tagged stream to PDU block: A tagged stream to PDU block is a block that works on streamed, but packetized input data. However, in traditional GNU radio blocks, if we stream N bytes into a block, there's no way of knowing the packet boundary.

3- Stream to Tagged stream block: Connecting regular stream blocks to tagged stream blocks will usually not work (as connecting the file source to the tagged stream to PDU).

   One solution is to use the stream to Tagged stream adapter block. It will periodically add tags at regular intervals, making the input valid for the tagged stream block. Figure 38 shows modified Wi-Fi transmitter block diagram.

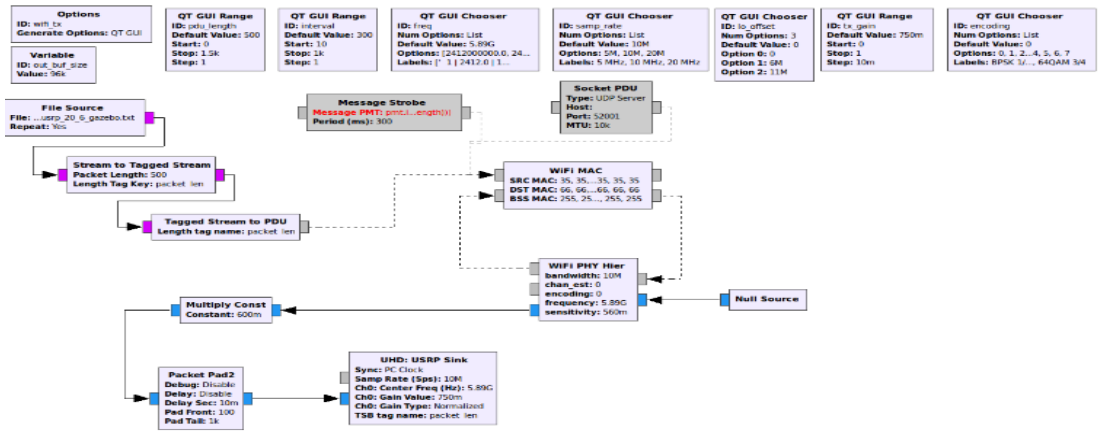


Figure 38: Modified Wi-Fi Transmitter block diagram.

5.3.2 Hardware Tool (USRP)

5.3.2.1 Overview

It stands for Universal Software Radio peripheral, the device is used as transceiver for radio frequency signals in wireless communication systems through the development of Software-defined radios. Figure 39 shows USRP B200 kit.

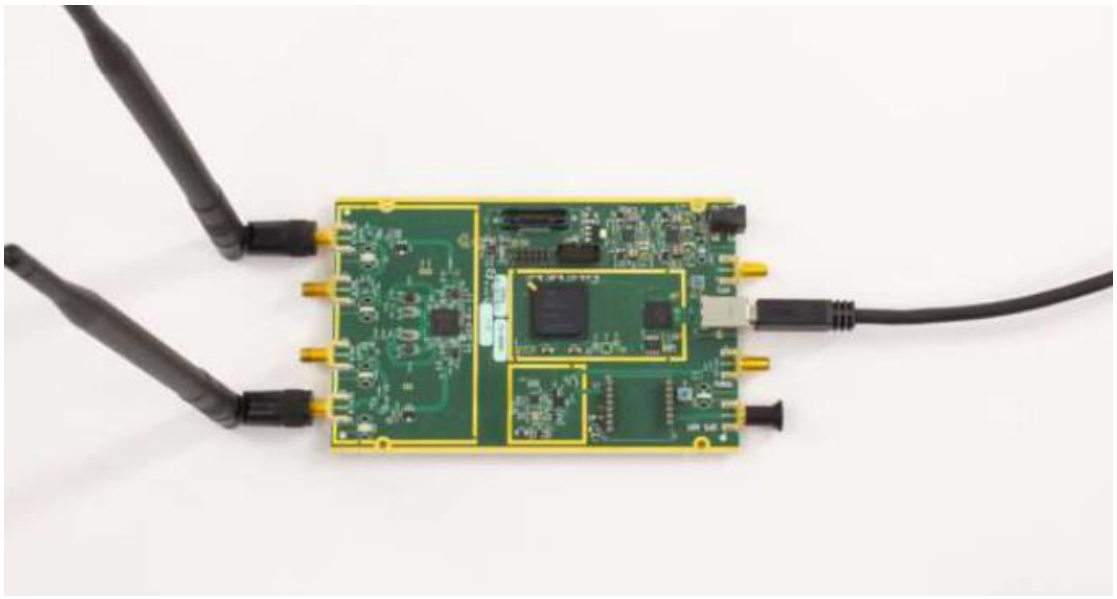


Figure 39: USRP B200.

5.3.2.2 Usage in the project

In our project the USRP B200 used to transfer the data from one end to the another end after setting the required antenna parameters and bandwidth occupied by the transferred data and this happened through two principle blocks in GNU radio; **USRP sink** which is responsible for adjusting the parameters at the transmitter side and **USRP source** which is responsible for adjusting the parameters at the receiver side.

5.4 Integrating both tools for functionality

First: Connecting the two RF antennas to the two USRP (one for each USRP).

Second: Connecting the two USRP to two laptops representing Tx side and Rx side.

Third: Writing in the terminal of both Tx and Rx these commands:

1- `Sudo uhd_usrp_probe`

This command is for detecting successful connection of USRP device.

2- `Sudo gnuradio-companion`

This command to open GNU radio.

Fourth: Open the Wi-Fi Tx file in GNU radio transmitter side and make the modifications (as mentioned before) and open the Wi-Fi Rx file in GNU radio receiver side.

Finally: Start transmission of data and observe the received data in the receiver side.

Chapter 6 Results

In this chapter, we will present our results and our attempts to achieve these results.

6.1 Deep learning Classifier

In this section we will present our achieved results which came after several trials

6.1.1 Testing on data extracted from Gazebo

We tested on about 1032 balanced images from accidents and no-accidents scenarios extracted from Gazebo different from the scenarios that are used in training the model.

In training the model, we have tried several changes in parameters representing in number of epochs, dropout probability, number of neurons in the three fully connected layers and number of kernels in the convolution layers.

The training accuracies of all these models are close to each other. So, only the test accuracy will enable us to choose the best model.

Table 7 shows the models that we have tried and the corresponding test accuracies.

Model number	Parameters								Test accuracy
	No of neurons in FC	No of epochs	Dropout probability	No of kernals in convolution layers					
				C1	C2	C3	C4	C5	
1	4069	25	0.8	96	256	384	384	256	67%
2	1024	25	0.5	96	256	384	384	256	77.10%
3	1024	25	0.8	96	256	384	384	256	68.30%
4	1024	18	0.5	96	256	384	384	256	70.20%
5	1024	18	0.8	96	256	384	384	256	66.90%
6	1024	15	0.5	96	256	384	384	256	69%
7	1024	25	0.5	96	48	256	192	256	69%
8	1024	25	0.8	96	48	256	192	256	68.99%
9	1024	18	0.8	96	48	256	192	256	63.60%

Table 7: Gazebo testing results.

From table 7 we found that model 2 is the best model as it gives the highest test accuracy among other models which is 77.1%. Also we tested this model for only the 512 non-accident frames and the accuracy is found 68.9% while testing this model for only the 512 accident frames gave an accuracy of 84.8%.

6.1.2 Testing on real data (generalization)

After testing on Gazebo scenarios and getting the best model that gives the best test accuracy, we tested on real data to ensure that the model can generalize.

The used real data is represented in 526 non-accident images found in a paper of name "Road-Accident Detection" and 820 balanced accident and non-accident frames captured from online videos.

We used a Matlab code that takes the video and captures frames from it and we made a suitable delay between capturing frames so that we can get clearly the non-accident and the accident frames from the video.

Firstly, we tested the 526 non-accident images from the paper and the resulting accuracy found to be 48%. And we thought that it is a predictable result as the distribution of these images is different from Gazebo images, so we need to calibrate the real images.

Calibration means to have same distribution in both images, so we calibrated the real data by the parameters found in the generated code from a tool in Matlab.

Secondly, we tested the calibrated real data and we got the results in table 8.

Type of test data		No of images	Test accuracy
Paper dataset	Non_accidents	526	61%
Online videos	Accidents	410	55%
	Non_accidents	410	44%
	All	820	48%

Table 8: Real data testing results

From table 8 we can notice that:

1- Testing our neural network model with real non-accident scenarios from online videos that contain bikes and motorbikes results in an accuracy which is not satisfied enough compared to the accuracy resulted from the paper dataset which is close to the environment exists in Gazebo.
2- The sensitivity of detecting accident is more than detecting non-accident which isn't bad.

6.2 V2V communication process

Testing this process is in the following procedure:

1- Inputting a text file to GNU radio of the predictions of the classifier deep learning model (we tested for 30 frames)
2- Connecting USRP channel to GNU radio in both Tx and Rx sides.
3- Starting to transmit the data exists in the text file.
4- The communication process can be observed in the Rx side represented in the constellation window, which represents that there is a communication between both sides, and in the terminal as shown in figures 40 and 41.

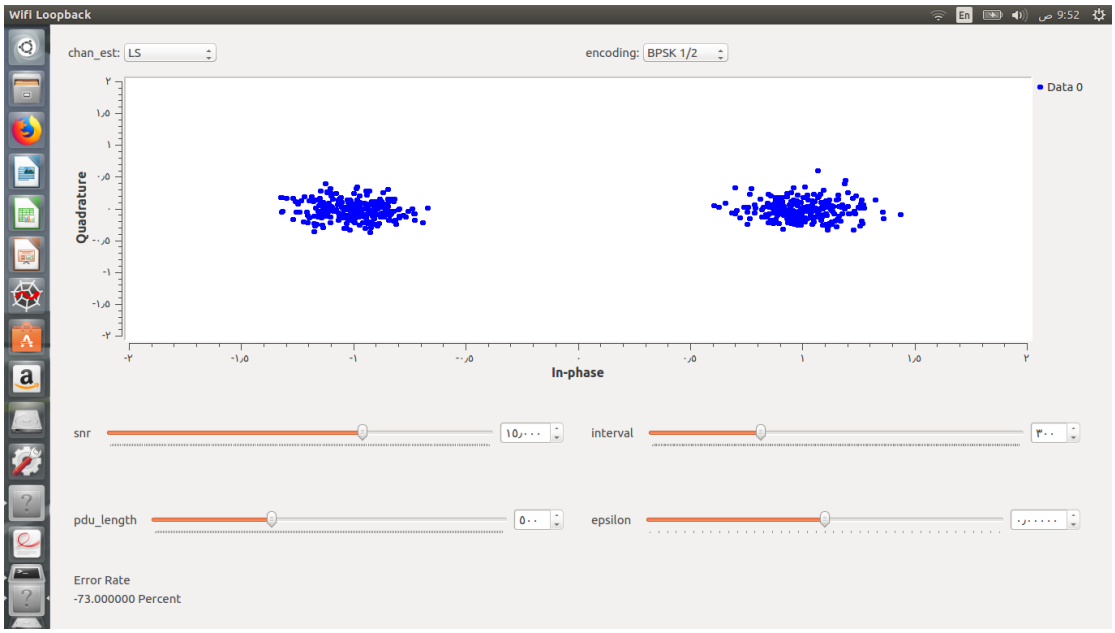


Figure 40: constellation points of the transmitted data.

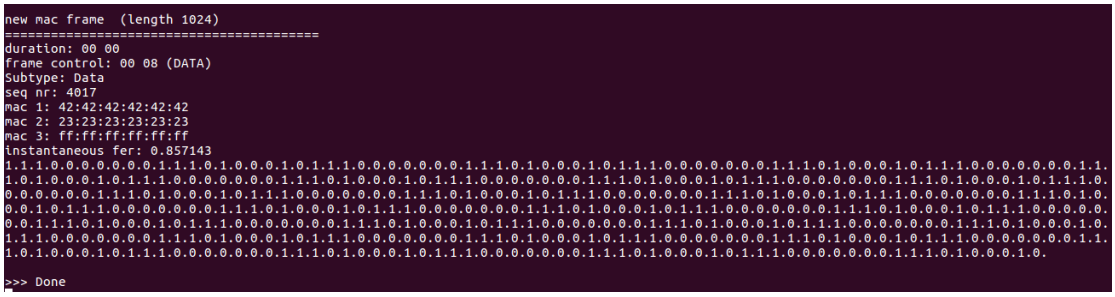


Figure 41: Transmitted data at the receiver terminal.

6.3 Real simulation

This is an extra phase in which we tried to perform a real complete simulation cycle for our project.

The following procedure represents the complete simulation cycle:

1- Taking live video in streets of the college using USB camera connected to Laptop.
2- Using Open-CV, we managed to capture frames from this video with a sufficient delay between frames to be the input to the classifier deep learning model.
3- Testing these frames using our classifier model and writing the predictions in a text file using python code.
4- Transmitting this text file which is in the file source in Tx side to the Rx side by using the USRP channel.

From this real simulation, we recorded these results:

1- The test accuracy for non-accident video is 90%
2- The test accuracy for accident video is 73.3%
   (We can't take real accident videos in the college, so we made the accident ourselves using a car toy)
3- V2V communication range is theoretically up to 300 meters but when trying this real simulation we found that this range is affected by weather conditions like any other wireless channels.

4- The time taken for the complete cycle starting from capturing frames till receiving the data in the Rx side depends on number of frames captured in the one cycle, the conditions of the channel between the transmitter and the receiver and running the model either on CPU or GPU (in this real simulation we run the classifier model on CPU). For 20 frames it takes about 22 seconds.

The figures below show samples from the non-accident frames that were captured.

And the following figure shows a sample from the accident frames that were captured.

Chapter 7 Conclusion

End-to-End Crash Avoidance Multi-task Deep Learning Solution System is proposed solving the poor actions taken nowadays by the already existing solutions like Forwarding Collision Braking (FCW), and Automatic Emergency Braking (AEB). It is considered as the first time merging the Deep Learning field with the V2V communication technology especially in the automotive applications. Crash Avoidance Functionality is one of the most important functionalities in Self-Driving Car Systems.

Currently, it is partially integrated into the system, so our proposed architecture aims to extend this functionality wider. Our proposed architecture depends on the detection neural network, and Accident Information Spreading IOT (especially V2V protocol). Detection Neural Network aims to detect that there is a collision in front of the ego-vehicle. This Network is trained on simulated data generated from ROS with various scenarios, and it could be augmented with some Real data in order to ensure having a generalized Deep Learning model. Accident Information Spreading IoT aims to inform the upcoming vehicles with advanced information about the accident.

7.1 Challenges

We encountered a lot of problems during the project, and we'll show some of them (the main challenges) in this part:

First: Deep learning and machine learning more generally, needs a good dataset to work properly and fit our project. We tried to found this dataset in any published paper. Unfortunately, we didn't found enough dataset to train our neural network model. As a result, using Gazebo ROS simulation to generate the data ourselves was a suitable satisfactory solution.

Second: Our goal was to test our neural network with real images from the real-life and to obtain highest accuracy. However, the real images must be calibrated with the same parameters as those of the simulation dataset. So, when we applied the calibration parameters on the real images, the calibrated "real" images had a distortion. That's because (as we suppose) the front camera of the gazebo car is not realistic enough. Then we applied the parameters found in the generated code from a tool in Matlab and they were better than Gazebo parameters with higher test accuracy.

Third: Concerning V2V communication, it will be better for GNU radio if it can receive continuously classification results from the model to transmit them to the other side, which represents the received vehicle, and so achieving real time application, however this couldn't be achieved till now in our project as we need at each transmission to restart GNU radio to take new classification results. So till now we suppose that we take a number of frames from the car front camera, classifying them and transmitting the results by the V2V USRP channel and then restarting GNU radio to take new number of frames from the camera and starting a new cycle, so as a future work we hope this problem could be solved.

7.2 Lessons learned throughout the year

There were a lot of lessons learned from this project and absolutely will be kept with us in the upcoming future.

a) We learned that there are many simulation tools available to generate the appropriate dataset ourselves with our requirements to best fit our model instead of readymade dataset.
b) Proficient use of Linux.
c) Python programming language.
d) Be familiar with the V2V communication protocol.

7.3 Future Work

The future work of the project will be divided into many paths which we'll discuss in this section;

First: Due to the non-capability of generating some real scenarios in Gazebo, for example, that includes bikes and motorbikes, testing our neural network model with real scenarios that contain bikes and motorbikes results in an accuracy which is not satisfied enough. It's recommended future work to solve this problem which -we suppose- is making fine-tuning for some scenarios that contain motorbikes and bikes so that the model can be generalized better.

Second: Crash Avoidance functionality is considered as one of the most important features in Self-Driving Cars. Recently, it is partially integrated into the Self-Driving Car system. This network will have the ability to control the Steering angle, and Throttle of the ego-vehicle based on the same Front Camera images used in the previous Detection Neural Network.

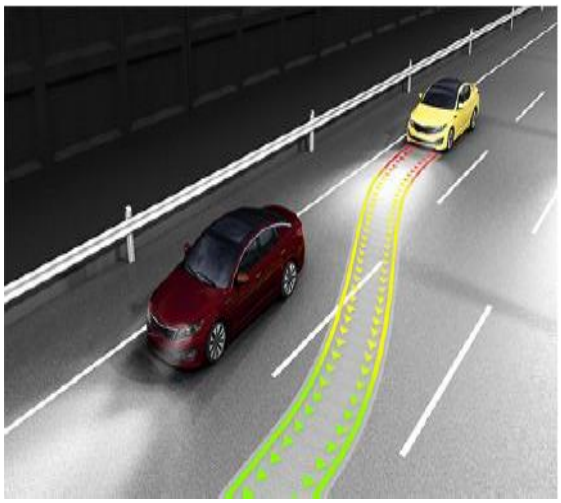


Figure 42: Path Planning.

Third: As a plus, it's supposed to make a mobile application that helps the driver to take the decision himself if the car isn't Self-Driving. Also, it may be integrated into more useful projects as connecting the vehicles with emergency stations.

References

[1] Ros Wiki-Documentation, ROS.org. [online]. Available:

http://wiki.ros.org/Documentation

[Accessed: 1-July-2019]

[2] Gazebo tutorial. [online]. Available:

http://gazebosim.org/?fbclid=IwAR2bLTUT8Aysu_DM6DcVunehOO1D4OZLXS6sQYU5YQ1KE6cdxYdUle5VO2U

[Accessed: 1-July-2019]

[3] ROS tutorial. [online]. Available:

http://wiki.ros.org/?fbclid=IwAR1B6rNHkcOzBr0QGWtR7rdnmh4MqUGetZOWcsBuY4BLHW6Vbwfqf4ZbNHU

[Accessed: 2-July-2019]

[4] AlexNet Wikipedia. [online]. Available:

https://en.wikipedia.org/wiki/AlexNet

[Accessed: 1-July-2019]

[5] Deep learning in five and a half minutes, videantis. [online]. Available:

http://www.videantis.com/deep-learning-in-five-and-a-half-minutes.html

[Accessed: 1-July-2019]

[6] ImageNet Classifier with TensorFlow, GitHub. [online]. Available:

https://github.com/narenkmanoharan/ImageNet-Classifier-Tensorflow

[Accessed: 1-July-2019]

[7] ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks, CV-Tricks.com. [online]. Available:

https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

[Accessed: 1-July-2019]

[8] A Comprehensive Guide to Convolutional Neural Networks Towards data Science. [online]. Available:

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[Accessed: 1-July-2019]

[9] Machine Learning - Convolution for image processing, Francium Tech. [online]. Available:

https://blog.francium.tech/machine-learning-convolution-for-image-processing-42623c8dbec0

[Accessed: 1-July-2019]

[10] Understanding of Convolutional Neural Network (CNN)—Deep Learning. [online]. Available:

https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[Accessed: 1-July-2019]

[11] Convolutional Neural Networks (CNNs / ConvNets). [online]. Available:

http://cs231n.github.io/convolutional-networks/#pool

[Accessed: 1-July-2019]

[12] Understanding of Convolutional Neural Network (CNN),Deep Learning [online]. Available:

https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep- learning-99760835f148

[Accessed: 1-July-2019]

[13] The benefit of using average pooling rather than max pooling,Qoura. [online]. Available:

https://www.quora.com/What-is-the-benefit-of-using-average-pooling-rather-than-max-pooling

[Accessed: 1-July-2019]

[14] Convolutional neural network, Wikipedia [online]. Available:

https://en.wikipedia.org/wiki/Convolutional_neural_network#ReLU_layer

[Accessed: 1-July-2019]

[15] Understanding Activation Functions in Deep Learning, Learn OpenCV. [online]. Available:

https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

[Accessed: 2-July-2019]

[16] Calculating the derivative of Tanh - step by step, Ronny Restrepo. [online]. Available:

http://ronny.rest/blog/post_2017_08_16_tanh/

[Accessed: 2-July-2019]

[17] Difference between Local Response Normalization and Batch Normalization, Towards data Science. [online]. Available:

https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-_normalization-272308c034ac

[Accessed: 2-July-2019]

[18] Local Response Normalization In Convolutional Neural Networks, PERPETUAL ENIGMA

. [online]. Available:

https://prateekvjoshi.com/2016/04/05/what-is-local-response-normalization-in-convolutional-neural-networks/

[Accessed: 2-July-2019]

[19] Normalizations in Neural Networks. [online]. Available:

http://yeephycho.github.io/2016/08/03/Normalizations-in-neural-networks/

[Accessed: 2-July-2019]

[20] cross validated. [online]. Available:

https://stats.stackexchange.com/

[Accessed: 2-July-2019]

[21] Understanding the Softmax activation function, Bartosz Mikulski . [online]. Available:

https://www.mikulskibartosz.name/understanding-the-softmax-activation-function/

[Accessed: 2-July-2019]

[22] Deep Learning #3: More on CNNs & Handling Overfitting, Towards data Science. [online]. Available:

https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d

[Accessed: 2-July-2019]

[23] Image Pre-processing, Towards data Science. [online]. Available:

https://towardsdatascience.com/image-pre-processing-c1aec0be3edf

[Accessed: 2-July-2019]

[24] Md Zahangir Alom , Tarek M. Taha ," The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches"

[25] Supervised and Unsupervised Machine Learning Algorithms. [online]. Available:

https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[Accessed: 2-July-2019]

[26] A Beginner's Guide to Deep Reinforcement Learning. [online]. Available:

https://skymind.ai/wiki/deep-reinforcement-learning

[Accessed: 2-July-2019]

[27] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton," ImageNet Classification with Deep
Convolutional Neural Networks", 2012.

[28] vehicle-to-vehicle communications, extreme tech. [online]. Available:

https://www.extremetech.com/extreme/176093-v2v-what-are-vehicle-to-vehicle-communications-and-how-does-it-work

[Accessed: 2-July-2019]

[29] Wireless LAN 802.11 Wi-Fi, Engineering and technology history Wiki , extreme tech. [online]. Available:

https://ethw.org/Wireless_LAN_802.11_Wi-Fi

[Accessed: 2-July-2019]

[30] GNU Radio Manual and C++ API Reference. [online]. Available:

https://www.gnuradio.org/doc/doxygen/page_tagged_stream_blocks.html

[Accessed: 2-July-2019]

Appendix

A.1 Installation Guide for Gazebo-ROS

```
1) sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-
latest.list'

2) sudo apt-key adv --keyserver hkp://ha.pool.sks-
keyservers.net:80 --recv-key
421C365BD9FF1F717815A3895523BAEEB01FA116

3) sudo apt-get update

4) sudo apt-get install ros-kinetic-desktop-full

5) apt-cache search ros-kinetic

6) sudo rosdep init

7) rosdep update

8) echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc

9) source ~/.bashrc

10) sudo apt install python-rosinstall python-rosinstall-
generator python-wstool build-         essential
11)sudo apt remove gazebo7

12)sudo apt install gazebo8

13)sudo apt-get install python-catkin-pkg python-catkin-tools

14)sudo apt-get install ros-kinetic-gazebo8-msgs

15)sudo apt-get install ros-kinetic-gazebo8-ros-control

16)sudo apt-get install ros-kinetic-gazebo8-plugins

17)sudo apt-get install ros-kinetic-gazebo8-ros-pkgs

18)sudo apt-get install ros-kinetic-gazebo8-ros

19)sudo apt-get install ros-kinetic-gazebo8-ros-pkgs

20)sudo apt-get install ros-kinetic-ros-control

21)sudo apt-get install ros-kinetic-ros-controllers
```

22) sudo apt-get install ros-kinetic-controller-manager

23) sudo apt-get install ros-kinetic-trac-ik

24) sudo apt-get install ros-kinetic-trac-ik-kinematics-plugin

25) sudo apt-get install ros-kinetic-gazebo8-plugins

A.2 Installation guide for GNU Radio

1) sudo apt-get update
2) sudo apt-get -y install git swig cmake doxygen build-essential libboost-all-dev libtool libusb-1.0-0 libusb-1.0-0-dev libudev-dev libncurses5-dev libfftw3-bin libfftw3-dev libfftw3-doc libcppunit-1.13-0v5 libcppunit-dev libcppunit-doc ncurses-bin cpufrequtils python-numpy python-numpy-doc python-numpy-dbg python-scipy python-docutils qt4-bin-dbg qt4-default qt4-doc libqt4-dev libqt4-dev-bin python-qt4 python-qt4-dbg python-qt4-dev python-qt4-doc python-qt4-doc libqwt6abi1 libfftw3-bin libfftw3-dev libfftw3-doc ncurses-bin libncurses5 libncurses5-dev libncurses5-dbg libfontconfig1-dev libxrender-dev libpulse-dev swig g++ automake autoconf libtool python-dev libfftw3-dev libcppunit-dev libboost-all-dev libusb-dev libusb-1.0-0-dev fort77 libsdl1.2-dev python-wxgtk3.0 git-core libqt4-dev python-numpy ccache python-opengl libgsl-dev python-cheetah python-mako python-lxml doxygen qt4-default qt4-dev-tools libusb-1.0-0-dev libqwt5-qt4-dev libqwtplot3d-qt4-dev pyqt4-dev-tools python-qwt5-qt4 cmake git-core wget libxi-dev gtk2-engines-pixbuf r-base-dev python-tk liborc-0.4-0 liborc-0.4-dev libasound2-dev python-gtk2 libzmq-dev libzmq1 python-requests python-sphinx libcomedi-dev python-zmq python-setuptools
3) cd $HOME
4) mkdir workarea-uhd
5) cd workarea-uhd
6) wget http://www.sbrac.org/files/build-gnuradio && chmod a+x build-gnuradio && ./build-gnuradio
7) cd build
8) cmake ../

9) make
10) solve the too many arguments to function volk_32f_8u_polarbutterfly_32f error by deleting the block_size() argument
11) make
12) make test
13) sudo make install
14) sudo idconfig
15) To open the gnu radio for the first time we need to open it through the terminal ,so we write (gnuradio-companion)
16) To get the blocks Write these commands:

- sudo apt-get install liblog4cpp5-dev
- sudo port install log4cpp
- git clone https://github.com/bastibl/gr-foo.git
- cd gr-foo
- mkdir build
- cd build
- cmake ..
- Make
- sudo make install
- sudo ldconfig
- git clone git://github.com/bastibl/gr-ieee802-11.git
- cd gr-ieee802-11
- mkdir build
- cd build
- cmake ..
- make
- sudo make install
- sudo ldconfig
- sudo sysctl -w kernel.shmmax=2147483648

A.3 Installation guide for USRP hardware driver

1) Open terminal window
2) Write these commands:
    - sudo apt -get install libuhd –dev libuhd uhd -host
    - sudo add-apt-repository ppa:ettusresearch/uhd
    - sudo apt-get update
    - sudo apt-get install libuhd-dev libuhd003 uhd-host