DUAL MODE CRASH AVOIDANCE

By

Ahmed Anwar Saeed Saad
Ahmed Saad Abdelfattah Ahmed
Hind Ashraf Taha Ali
Marehan Refaat Mohamed Mohamed
Nagham Wael Mohamed Mohamed
Omar Khaled Saleh Mohamed
Omar Mohamed Mostafa El-Nwishy
Omar Shaaban Roshdy Sharkawy

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the
BACHELOR DEGREE
in
Electronics and Communications Engineering

FACULTY OF ENGINEERING ,CAIRO UNIVERSITY
GIZA,EGYPT
2020

Approved by the Examining Committee:

Dr. Hassan Mustafa,	Thesis Main Advisor
Dr. Mohsen Rashwan,	Thesis Main Advisor
Dr. Ahmed Madian,	External Examiner
Dr. Mohamed Saeed,	External Examiner
Dr. Mohamed Abdelghany,	External Examiner

Acknowledgements

This dissertation would not have been possible without the great support of our supervisors, Dr. Hassan Mostafa and Dr. Mohsen Rashwan, as well as Eng. Abdelrahman Hussein, Eng.Mohamed Abdou and Eng.Eslam Bakr.

Dr. Hassan Mostafa and Dr. Mohsen Rashwan, we would like to thank you both for your guidance, patience, help and support. We appreciate the great supervision you offered us through the whole project, and the equipment you helped us access and use easily.

Eng. Abdelrahman Hussein, we are grateful for your technical guidance and concern about each individual of the team. Your knowledge and advice have always had a significant impact on our progression.

Thanks for Valeo, especially for Eng.Mohamed Abdou and Eng.Eslam Bakr, we would thank you for the support and guidance you provided us.

Table of Contents

List of Tables

List of Figures

Abstract

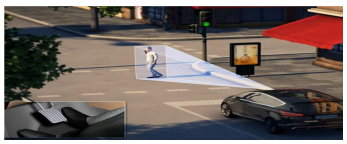
In this project, a solution for car accidents is proposed both in autonomous and manual driving vehicles. Crash avoidance in autonomous vehicles is achieved through some techniques like obstacle avoidance and lane overtaking, where it detects that there is an obstacle at certain distance and begin to see whether there is a possibility to overtake it or not. After applying many architectures, the best results on the simulator where achieved by CNN architecture with loss of 0.0738, model size of 127.92 MB and inference time of 1 ms. While in manual driving vehicles, a driver assistance and a real-time distraction detection system are built to decrease the probability of being distracted while driving, which in turn, decreases the probability of making an accident. The distraction detection system is based on ResNet50 architecture, which achieves classification accuracy and loss of 94.3% and 0.25, respectively. A quantized version of the trained ResNet50 architecture is introduced in order to facilitate the deployment and operation on the hardware. The quantized version achieves classification accuracy and loss of 92.2% and 0.26, respectively, with 3.75x reduction in memory usage and 2x reduction in inference time, resulting in a model size of 24MB and inference time of 75ms on the CPU in which the experiments are conducted. A speech recognition system, as a driver assistance system, is built to provide the driver some services that make him feel the luxury. Many classification networks (CNNs and RNN) are tried out by using some techniques to increase the data size for generalization. RNN is the best choice to represent speech recognition module when implementing it, as it has less inference time of 0.69 ms, but it has 4.4M number of parameters. By using ensemble learning, a speech recognition system within a specific domain of commands is made with classification accuracies of 97.55%, 97.73%, and 90.238% while the categorical cross-entropy losses are 0.1457, and 0.10314 on validation, test, and submission datasets respectively with a rate of 115 audios/sec on Tesla T4.

Chapter 1

Introduction

Advancements in science and technology have been reshaping our lives in recent decades, where Artificial Intelligence and Machine Learning are the core for building any smart system today. A lot of car accidents occur as the vehicles can't avoid crashing especially the self-driving ones, as they may be at high velocities. If an obstacle suddenly appeared, it may not have time to decrease its velocity which would lead to crashing. Also, many car accidents occur due to driver distraction, as shown in the figure ,Distraction has many shapes,whether this distraction is due to objects on the road, or due to the interaction of the driver with any other subject, like talking on the phone, texting, drinking, or talking with a passenger. Distraction may also occur due to the interaction with the vehicle itself, like operating the radio, turning the air conditioning on, or looking behind to grab something . These problems lead to the loss of many lives, and our project is introducing a solution to avoid it.Our problem it is divided into three modules,as each module can deal with some these shapes of distraction ,first module deals with the surrounding environment of the semi-self driving car,as shown in figure 1.1 and second and third modules deal with any distraction could happen inside the car from the driver ,as shown in figure 1.2.

(a) Crash due to a car



(b) Human in front of the car

Figure 1.1: Some shapes of distraction due to the surrounding environment

(a) Drinking

(b) Talking to a passenger

Figure 1.2: Some shapes of distraction due to driver

1.1 Lane Overtaking

The lane change maneuver is one of the most thoroughly investigated automatic driving operations for autonomous vehicles after trajectory tracking. This maneuver is used as a primitive for performing more complex operations like changing lanes on a highway, leaving the road, or overtaking another vehicle. [1]

A total of 13,939 fatal crashes occurred during overtaking maneuvers in the United States from 1994 to 2005, which lead to the death of 24,565 people. Most of these accidents were caused because of failing to leave enough distance, overtaking when there was poor visibility, or by not giving way to an overtaking vehicle. These accidents forced some governments to ban overtaking like that of Netherlands. So, the objective of the autonomous vehicles is to have this feature (overtaking), but by taking the precautions and be more safe. [1]

Deep Neural Network is used for detecting the existence of an obstacle and the precautions that it will take to avoid crashing. According to the distance available between the vehicle and the obstacle and the surrounding circumstances (the availability of the neighboring lanes), the decision will be taken either by decreasing the speed of the car or by stopping or by making lane overtaking. Below, figure 1.3, represents the technique of the lane overtaking.

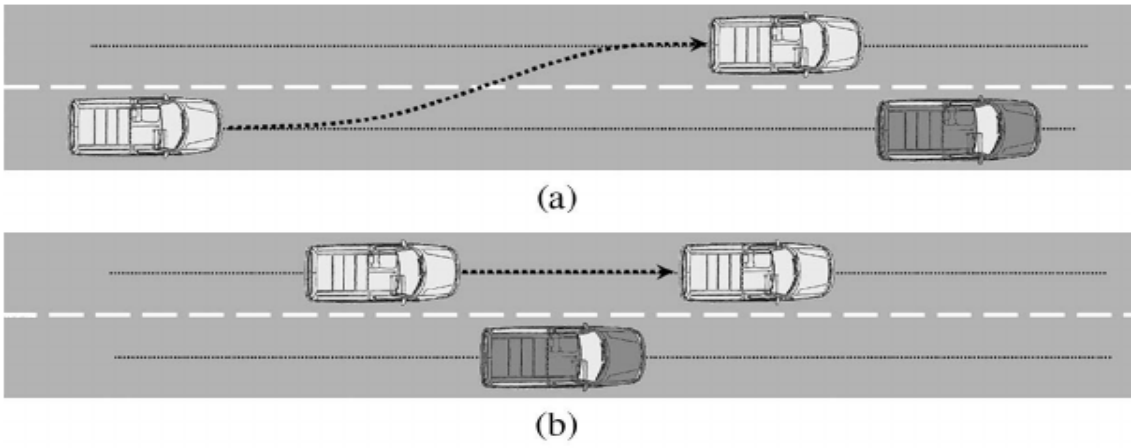


Figure 1.3: Lane overtaking technique

1.2 Distraction Detection

Road accidents occurring due to driver distraction increase as technology and means of distraction like In-Vehicle Information Systems (IVIS) develop. The World Health Organization (WHO) stated in the 2017 Global Status Report that 1.25 million yearly deaths worldwide take place due to road traffic accidents [2]. As the Center for Disease Control and Prevention (CDC) motor vehicle safety division reported, one in five car accidents are caused due to a distracted driver. According to the National Highway Traffic Safety Administration (NHTSA), 9 people are killed and 1000 are injured every day in crashes related to distracted driving in the United States [3]. Additionally, NHTSA reported that 9% of fatal crashes in 2016 were reported as distraction-affected crashes, and that 3450 people were killed in motor vehicle crashes involving distracted drivers [3]. All the mentioned reports about distracted driving threat raise the concern towards building distraction detection systems capable of identifying the posture and state of the driver, whether the driver is in a safe-driving state or distracted, and what kind of activity causes the distraction.

The NHTSA defines distracted driving as any activity that diverts attention from driving, including talking or texting on the phone, eating or drinking, talking to a passenger and operating the radio or navigation systems [4]. The CDC categorizes distracted driving into visual distraction (i.e taking one's eyes off the road), manual distraction (i.e taking one's hands off the driving wheel) and cognitive distraction (i.e taking one's mind off driving) [5].

A real-time distraction detection system is built to avoid the aforementioned problems of distraction. Figure 1.4 shows the pipeline of the proposed system. The system is based on Deep Learning, in which a CNN has been trained on different images of distracted drivers, in order to classify the state of the driver whether it is a safe driving or a distracted state. The different kinds of distraction and the datasets used in this work are discussed in section 4.2.1. The proposed system is able to operate in a real-time environment, with a satisfactory degree of generalization. Additionally, model optimization and hardware acceleration techniques such as pruning and quantization have been applied after training, giving the capability of deploying the model to an embedded system with less memory usage and faster inference time.

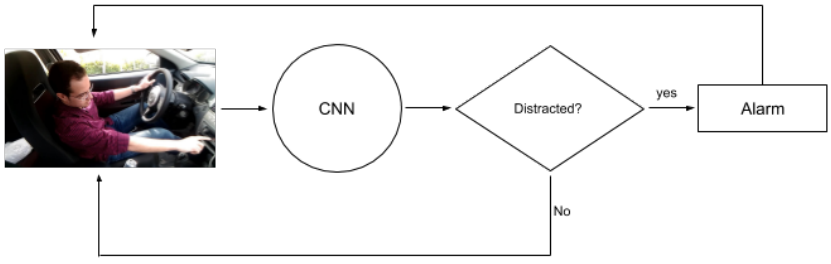


Figure 1.4: Distraction detection system pipeline

1.3 Speech Recognition

Speech recognition is a process of converting spoken words into text. It is also known as Automatic Speech Recognition (ASR). The technology gained acceptance and shape in early 1970s due the research funded by Advance Research Project Agency in U.S Department of Defense. It has been widely in use since 1970s in various domains such as automotive industry, health care, military, IT support centers, telephone directory assistance, embedded applications, automatic voice translation into foreign languages etc.

In-car speech recognition is our point of interest, which enables the driver to interact with the car. In-car speech recognition systems have become an almost standard feature in all many new vehicles on the market today. The days of getting in our cars and driving from point A to point B without any distractions is over. Even though safe driving behaviors (and in many places, the law) requires us to ignore the constant phone calls, emails, and text messages while behind the wheel, that kind of disconnectedness isn't the reality. A lot of voice technology was driven by the need to keep the public safe while still acknowledging the device-dependent epidemic. In-car speech recognition systems aim to remove the distraction of looking down at your mobile phone while you drive. Companies like Apple, Google, and Nuance are reshaping the way voice-activation is used in vehicles. They are the most popular companies providing this service that enables drivers to find directions, send emails, make phone calls, and play music, all by using the sound of their voice.

Command speech recognition system is built to assist the driver to interact with the surrounding environment and remove the distraction that may cause car crashes. By using the dataset provided from Tensorflow team on kaggle [6] as a challenge, ours models are trained to recognize them well. The following figure 1.5 illustrates the ASR process.

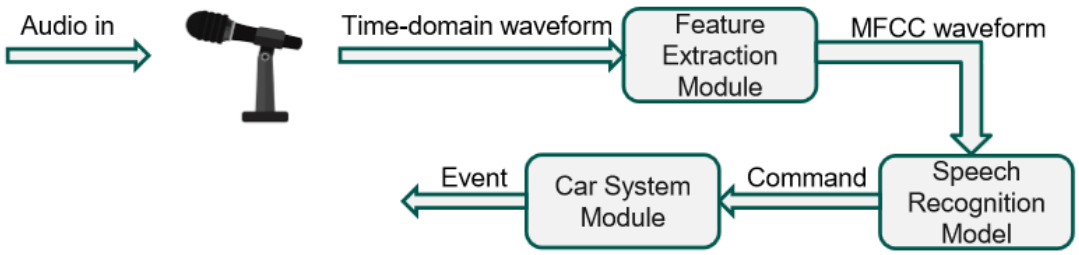


Figure 1.5: Speech recognition system

Chapter 2

Background

2.1 Automation phases

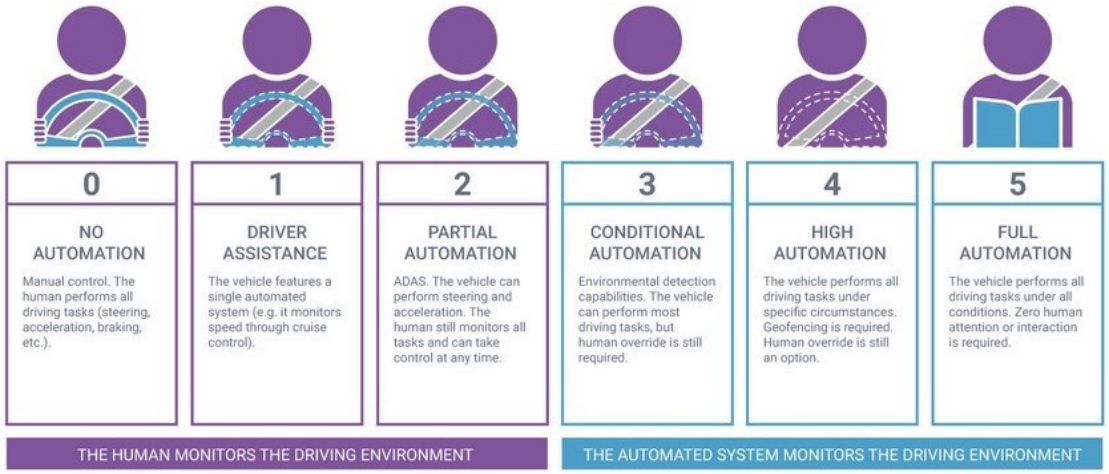


Figure 2.1: Automation phases

As shown in figure 2.1, there are five levels of automation. Here is much information about them: [7],[8],[9]

1. **Level 0**:
    - It is the level where there is no automation.
    - The human driver is in complete control of the vehicle. The driver performs all operating tasks like throttle, steering and breaking.
    - For example: **2005 Honda**

2. **Level 1**:

   - It is the level where there is a driver assistant.
   - The human driver sometimes receives help from an automated system inside the vehicle which controls some parts of the driving tasks.

     The vehicle can assist with some functions, but the driver still handles all braking, steering, throttle and monitoring the surrounding.

     For example: The car can brake a little extra when the diver is so close to another car.
   - This level could be found in almost cars today, such as: **2018 Toyota Corolla, 2018 Nissan Sentra** and others.

3. **Level 2**:

   - It is the level at which there is partial automation.
   - The vehicle can assist with steering or acceleration functions and allow the driver to disengage from some of their tasks. The driver must always be ready to take control of the vehicle and responsible for most safety-critical functions and all monitoring the environment.
   - For example: **Tesla Autopilot, Volvo Pilot Assist and Audi Traffic Jam Assist**

4. **Level 3**:

   - It is the level at which there is conditional automation.
   - The automated system can conduct some parts of the driving task and monitor the driving environment in some instances, but the driver should be ready to take back control on request.

     The vehicles are capable of driving themselves, but only under ideal conditions and with limitations, such as limited-access divided highways at a certain speed. Although hands are off the wheel, the driver is still required to take over when road conditions fall below ideal.

5. **Level 4**:

   - It is a level where high automation occurs.
   - The automated system can handle the driving task and monitor the driving environment. The vehicle is capable of steering, braking, accelerating, monitoring the environment, determining when to change lanes, turns and use signals.

     Although the driver will not need to take back control, the automated system will only be able to operate in certain environments and under certain conditions. For example: it cannot deal with traffic jams.

6. **Level 5**:

   - It is the level of full automation.
   - The vehicles are able to perform all driving tasks, monitor and go through all road conditions. There are no driver involvements at all, so, there is no need for pedals and steering wheel.
   - For example: **NVIDIA** announced an AI computer where drivers plug their destination and the vehicle controls everything.

2.2 Machine Learning

2.2.1 What is machine learning?

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves. [10] The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers to learn automatically without human intervention or assistance and adjust actions accordingly. [10]

2.2.2 History of machine learning

The first case of neural networks was in 1943, when neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper about neurons, and how they work. They decided to create a model of this using an electrical circuit, and therefore the neural network was born.
In 1950, Alan Turing created the world-famous Turing Test. This test is fairly simple for a computer to pass; it has to be able to convince a human that it is a human and not a computer.
1952 was the first computer program that could learn as it ran. It was a game which played checkers, created by Arthur Samuel.
1958 Frank Rosenblatt designed the first artificial neural network called Perceptron. The main goal of this was pattern and shape recognition.
1959 Bernard Widrow and Marcian Hoff created two models of them at Stanford University. The first was called "ADELINE", and it could detect binary patterns. For example in a stream of bits, it could predict what the next one would be. The next generation was called "MADELINE"; it could eliminate echo on phone lines so had a useful real world application. It is still in use today.

1982 John Hopfield suggested creating a network which had bidirectional lines similar to how neurons actually work.
1997 the IBM computer Deep Blue which was a chess-playing computer beat the world chess champion. Since then, there have been many more advances in the field.
1998 research at AT T Bell Laboratories on digit recognition resulted in good accuracy in detecting handwritten postcodes from the US Postal Service.
In the 2000s with the huge computational technological advancements, more machine learning researches were done and machine learning becomes a trending topic in the research area.[11]

2.2.3 Types of Machine Learning

There are three types of machine learning: Supervised learning, unsupervised learning and reinforcement learning.
Supervised Learning: ML algorithm is given a small training dataset to work with. This training dataset is a smaller part of the bigger dataset and serves to give the algorithm a basic idea of the problem, solution, and data points to be dealt with. The training dataset is also very similar to the final dataset in its characteristics and provides the algorithm with the labeled parameters required for the problem. The algorithm then, finds relationships between the parameters given essentially establishing a cause and effect relationship between the variables in the dataset. At the end of the training; the algorithm has an idea of how the data works and the relationship between the input and the output. [12]
Examples of Supervised learning algorithms:

- Regression:A set of statistical processes for estimating the relationships between a dependent variable(outcome value) and one or more independent variables (features or predictors).
  It includes different types like: Linear Regression as shown in Figure 1, Non-Linear Regression as shown in Figure 2.2 and Logistic Regression as shown in Figure 2.3.

- Classification :It refers to a predictive modeling problem where a class label is predicted for a given example of input data as shown in Figure 2.4.
  Examples of classification problems include: Given an example, classify if it is spam or not. Given a handwritten character, classify it as one of the known characters.

Unsupervised Learning: The labels allow the algorithm to find the exact nature of the relationship between any two data points. However, unsupervised learning does not have labels to work of resulting in the creation of hidden structures. Relationships between data points are perceived by the algorithm in an abstract manner with no input required from human beings.
Examples of unsupervised learning algorithms:

- Clustering:It mainly deals with finding a structure or pattern in a collection of uncategorized data. Clustering algorithms will process your data and find natural clusters (groups) if they exist in the data. You can also modify how many clusters your algorithms should identify as shown in Figure 2.5.

Reinforcement Learning:The algorithm interacts with a dynamic environment that provides feedback in terms of rewards and punishments. For example: self-driving cars being rewarded to stay on the road as shown in Figure 2.6.

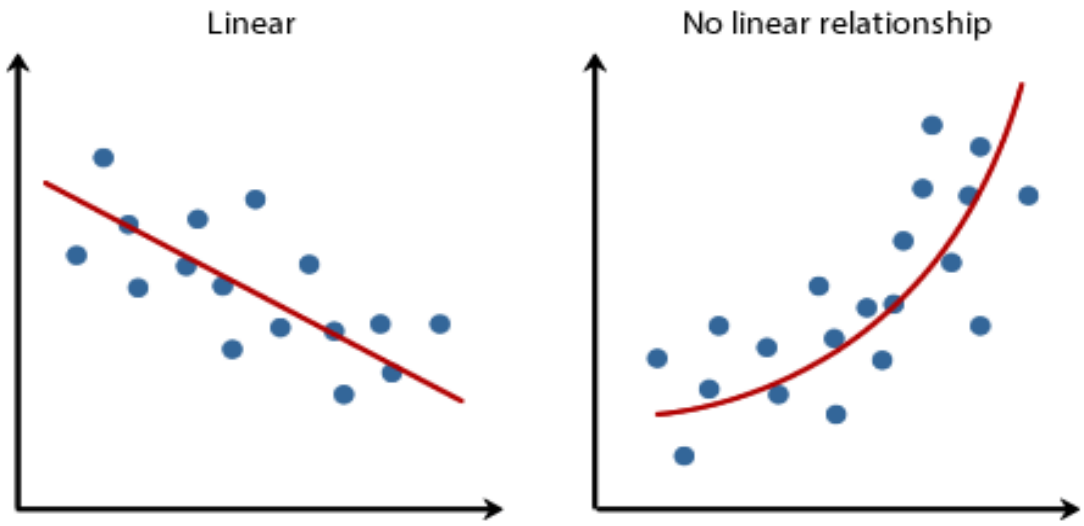


Figure 2.2: Linear and non- Linear Regression Model

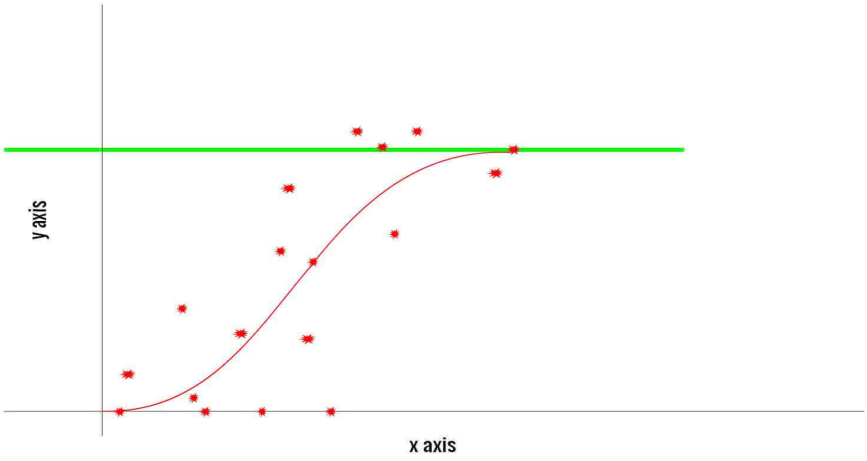


Figure 2.3: Logistic Regression Model

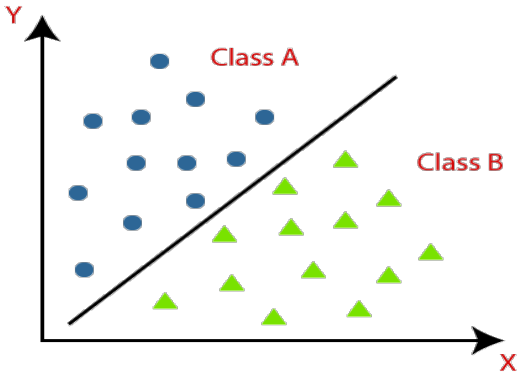


Figure 2.4: Classification Model

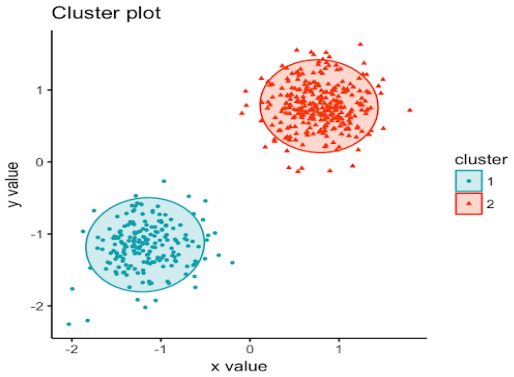


Figure 2.5: Clustering Model

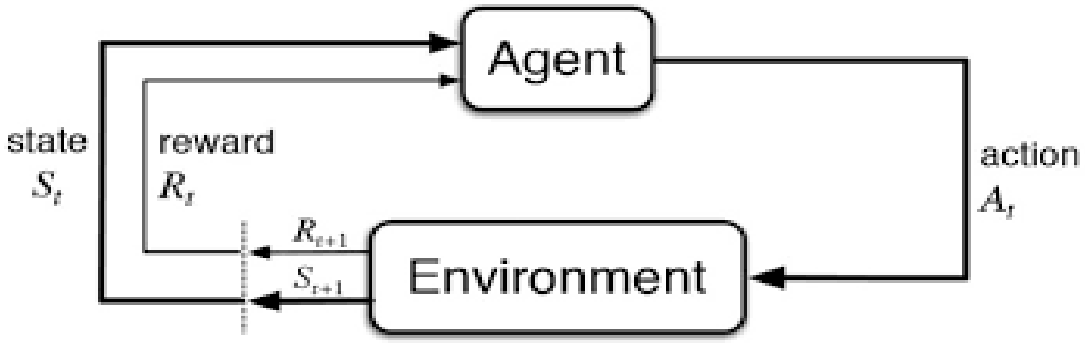


Figure 2.6: Block diagram of Reinforcement learning basic concept

2.3 Relation between deep learning and automation

Deep learning has been widely applied in image processing, natural language understanding, and so on. In recent years, more and more deep learning-based solutions have been presented in the field of self-driving cars and have achieved outstanding results. More and more solutions based on deep learning for self-driving cars have been presented, including obstacle detection, scene recognition, lane detection, and so on. [13]

During the last decade, deep learning has demonstrated to be an excellent technique in the field of AI. Deep learning methods have been used to solve various problems like image processing, speech recognition, and natural language processing. As deep learning can learn robust and effective feature representation through layer-by-layer feature transformation of the original signal automatically, it has a good capability to cope with some challenges in the field of self-driving cars. [13]

2.4 Deep Learning

2.4.1 What is deep learning?

Deep learning is an artificial intelligence (AI) function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence that has networks have the ability to do unsupervised, supervised and semi-supervised learning. Also known as deep neural learning or deep neural network. [14],[15]

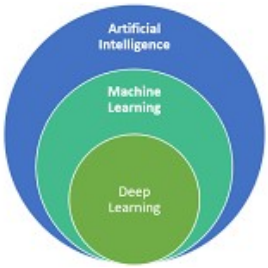


Figure 2.7: Relation between machine and deep learning

As shown in figure 2.7, deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces, as it is shown below in figure 2.8. [15]

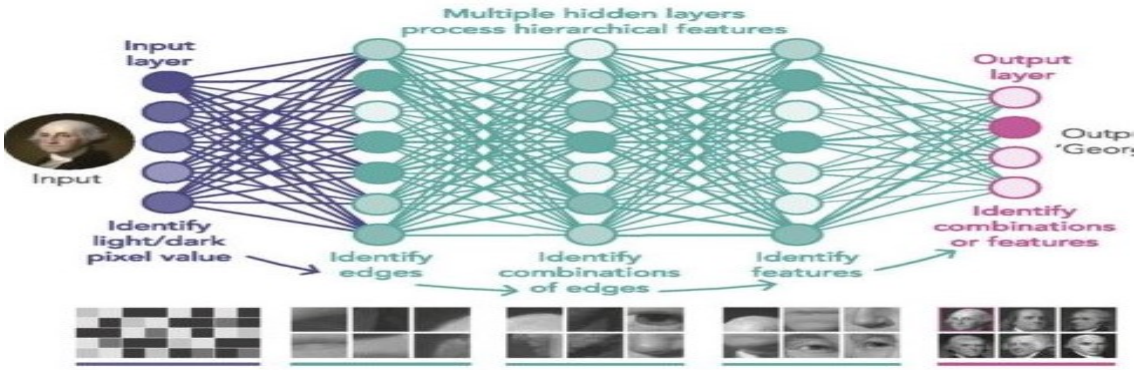


Figure 2.8: Deep learning neural network

Deep learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance. [15]

2.4.2 History of deep learning [16]

1. **Early start:** Early 40s to 60s
   (a) **McCullock and Pitts 1943:** Introduced the linear threshold "neuron".
   (b) **Rosenblatt 1962:** Applied a "Hebbian" learning rule.
   (c) **Novikoff 1962:** Proved the perceptron convergence theorem.
2. **Deep Winter I:** Late 60s through early 80s
   (a) **Robinson 1965:** Introduced resolution theorem proving.
   (b) **Minsky 1969:** Wined Turing Award for "promoting AI".
   (c) **McCarthy and Hayes 1968:** Introduced the situation calculus.
   (d) **Minsky and Papert 1969:** Published the book Perceptrons. They proved that many properties of images could not be determined by (single layer) perceptrons. Caused a decline of activity in neural network research.
   (e) **McCarthy, 1971:** Wined Turing Award.
   (f) **Minsky 1974:** Wrote "A Framework for Representing Knowledge".
   (g) **McCarthy 1980:** Introduced "non-monotonic logic".
3. **Deep Resurgence I:** Late 80s
   (a) **Fukushima 1980:** Introduced the Neocognitron (a form of CNN).
   (b) **Hinton and Sejnowski 1985:** Introduced the Boltzman machine.
   (c) **Rummelhart, Hinton and Williams 1986:** Demonstrated empirical success with backpropagation (itself dating back to 1961).
4. **Deep Winter II:** Late 90s' and 00's
   (a) **Valiant 1984:** Introduced the formal definition of PAC learnability. Credited with starting learning theory as a branch of computer science. Turing Award, 2010.
   (b) **Pearl 1995:** Published Probabilistic reasoning in intelligent systems: Networks of plausible inference. Credited with driving the "statistical revolution" in AI. Turing Award, 2011.
   (c) Convex optimization and convex relaxations (the marginal poly-type of a graphical model).
   (d) Nonparametric Bayesian inference (Dirichlet processes).
   (e) Submodular optimization.
   (f) **Schmidhuber et al. 1997:** Introduced LSTMs.
   (g) **LeCun 1998:** Introduced convolutional neural networks (CNNs) (LeNet).
   (h) **Bengio 2003:** Introduced neural language modeling.
5. **Deep Learning Exploding:** In 2012
   (a) **Alexnet** dominates the 2012 Imagenet challenge.
   (b) Google speech recognition converts to deep learning.

2.4.3 Models of deep learning

There are different types of models used in Deep Learning. [17]

- **Supervised Models:**
  - Classical Neural Networks (Multilayer Perceptrons)
  - Convolutional Neural Networks (CNNs)
  - Recurrent Neural Networks (RNNs)

- **Unsupervised Models:**
  - Self-Organizing Maps (SOMs)
  - Boltzman Machines
  - AutoEncoders

We will talk about the supervised models much clearly.

- **Classical Neural Networks:**

  Classic Neural Networks can also be referred to as Multilayer Perceptrons. The perceptron model was created in 1958 by American psychologist Frank Rosenblatt. Its singular nature allows it to adapt to basic binary patterns through a series of inputs, simulating the learning patterns of a human-brain.As shown in figure 2.9, a multilayer perceptron is the classic neural network model consisting of more than 2 layers (contains one hidden layer or more). [17]

  It is used in classification and regression problems where a set of real values is given as inputs. It is also used when the data set used us tabular data (formatted in rows and columns ex: CSV files) and when a higher level of flexibility is required in the model. [17]

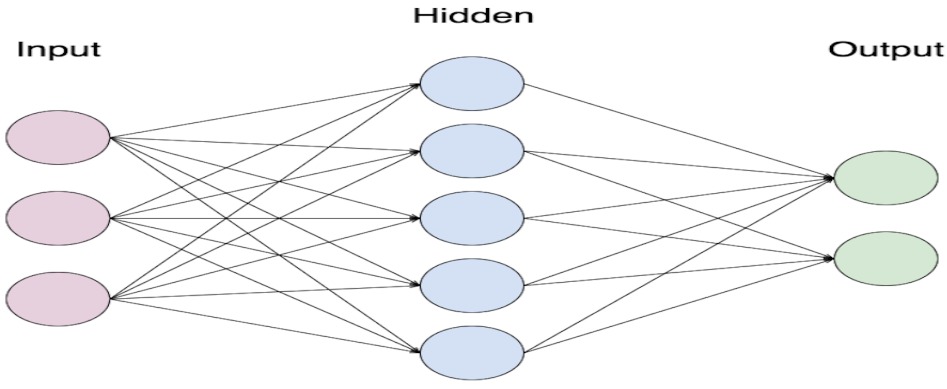


Figure 2.9: MLP architecture

- **Convolutional Neural Networks:**
  A more capable and advanced variation of classic artificial neural networks, a Convolutional Neural Network (CNN) is built to handle a greater amount of complexity around pre-processing, and computation of data. [17]

  CNNs were designed for image data and might be the most efficient and flexible model for image classification problems. Although they were not particularly built to work with non-image data, they can achieve stunning results with non-image data as well. [17]

  It is used for image datasets (including OCR document analysis) and when the model requires more complexity in calculating the output. The most common CNN architectures are **ZFNet, GoogleNet, VGGNet, AlexNet and ResNet**. [17],[18]

  In each CNN, there are two stages of training process, feed-forward stage and back-propagation stage. As shown in figure 2.10, CNN consists of 4 blocks: [17]

  1. **Convolutional layer:**
     A process in which feature maps are created (feature extraction occurs) out of the input data. Then, a function is applied to filter maps.
  2. **Subsampling:**
     It is done by taking average or the maximum. It enables the CNN to detect an image when presented with modification.
  3. **Flattening:**
     Flatten the data into an array, so CNN can read it.
  4. **Fully-connected layer:**
     The hidden layer where classification occurs. It also calculates the loss function of the model.

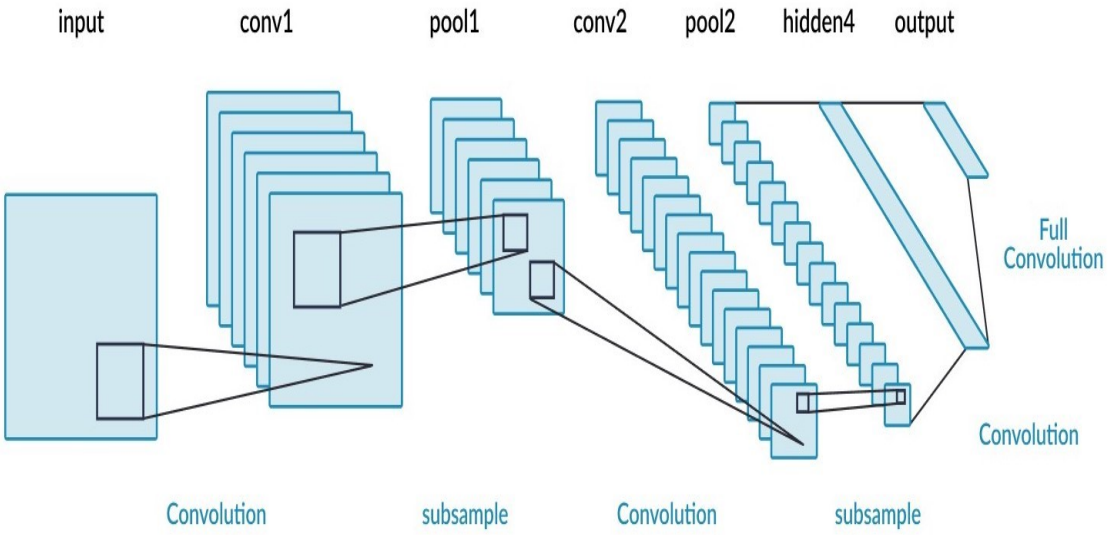


Figure 2.10: CNN architecture

- **Recurrent Neural Networks:**
  RNN is designed to recognize sequences and patterns such as speech, handwriting, text, and such applications. RNN benefits cyclic connections in the structure which employ recurrent computations to sequentially process the input data. [18]

  As shown in figure 2.11, RNN is basically a standard neural network that has been extended across time by having edges which feed into the next time step instead of into the next layer in the same time step. Each of the previous inputs data are kept in a state vector in hidden units, and these state vectors is utilized to compute the outputs. [18]

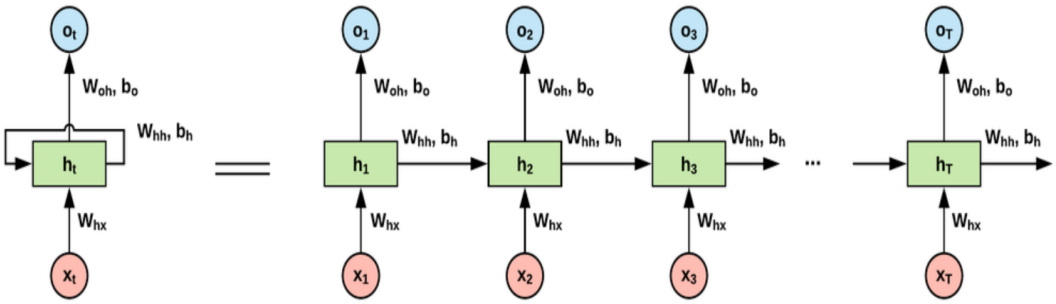


Figure 2.11: RNN architecture

  - **Long Short –Term Memory:**
    LSTM is an RNN method which benefits feedback connections to be used as a general-purpose computer. This method can be used for both sequences and patterns recognition and image processing applications. [18]

    In general, LSTM contains three central units, including input, output, and forget gates. LSTM can control on deciding when to let the input enter the neuron and to remember what was computed in the previous time step. One of the main strength of the LSTM method is that it decides all these based on the current input itself. Down below, figure 2.12 represents the architecture of the LSTM model. [18]

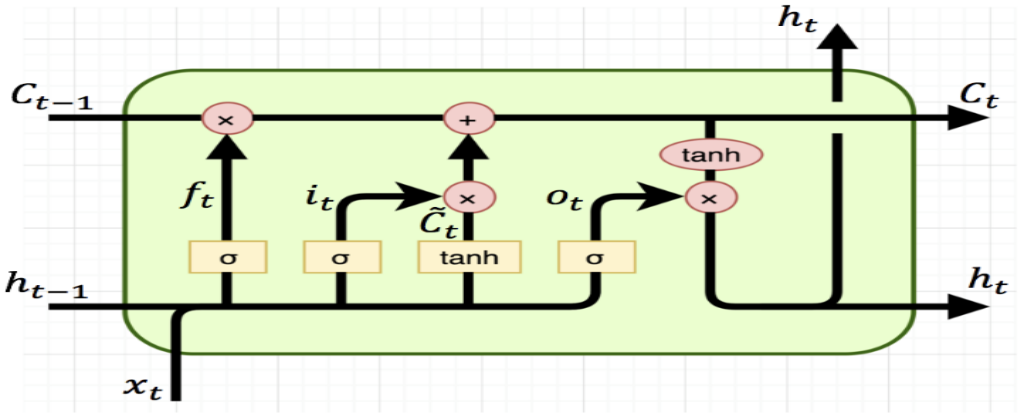


Figure 2.12: LSTM architecture

2.4.4 Impact of increasing data size on deep learning

Over the past several years, deep learning has become a very powerful technique for most of AI type problem, overshadowing classical machine learning. The first reason of that is that deep learning tries to present a new concept for solving real problems, which is artificial neural network that tries to simulate how the human brain works. The second reason is the increased amount of data collected for solving a problem, and that makes neural networks to achieve a superior performance on many AI fields including speech recognition.

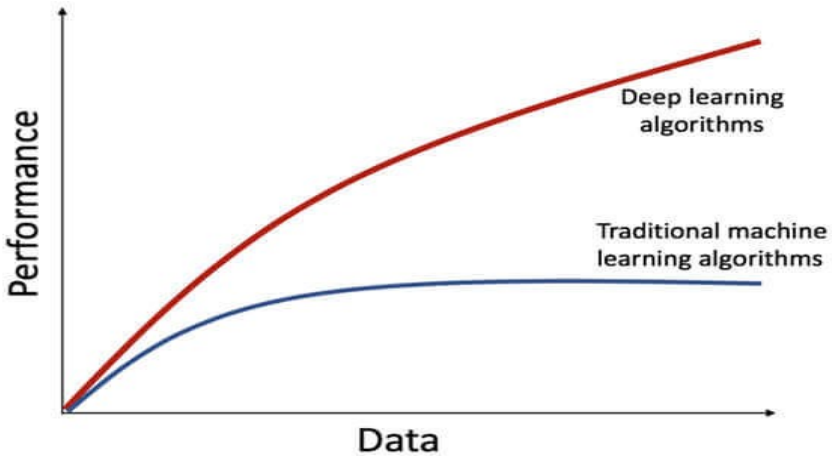


Figure 2.13: Comparison between ML and DL performance when increasing data

Increasing the amount of data has a good impact on the performance of deep learning models, especially on large neural networks, which are very deep and have many parameters, and makes the neural networks more generalized to many cases of the problem we want to solve (overfitting reduction). There are two ways to increase the data: collecting new data, which is very challenging process between big AI companies to improve their models more and more to serve the society. The second way is adding artificial data (augmented versions) to the dataset, which is cheaper and some techniques may consume much time for implementation.

There are many techniques to increase data size which will be covered in the next chapters. They made a great results in increasing the performance of the deep learning models, which are:

- Data augmentation
- Segmentation
- Semi-supervised learning
- Generative adversarial networks (GANs)

Chapter 3

Lane Overtaking

3.1 Carla simulator

Carla is an open-source simulator for autonomous driving research. It supports development, training and validation of autonomous urban driving systems. It also provides open digital assets (urban layouts, buildings, vehicles, pedestrians, street signals, ... etc) that are used freely.
It is used to study the performance of three approaches to autonomous driving: a classic modular pipeline, an end-to-end model trained via imitation learning and an end-to-end model trained via reinforcement learning. [19]

The simulation platform supports flexible setup of sensor suites and provides signals that can be used to train driving strategies, such as: GPS coordinates, speed, acceleration, detailed data on collisions and other infractions. A wide range of environmental conditions can be specified including the weather and the time. [19]
Down below, figure 3.1 illustrates various environmental conditions. It shows a street in Town 4 in 4 different weather conditions. Clockwisely from top left: clear day, daytime mid rain, daytime shortly after rain and clear sunset.

Figure 3.1: Sample of various environmental conditions for Town 4

Carla has been built for flexibility and realism in the rendering and physics simulation. It is implemented as an open-source layer over Unreal Engine(UE4). [19]

Carla simulates a dynamic world and provides a simple interface between the world and an agent that interacts with the world. To support this functionality, Carla is designed as a server-client system.
The server is responsible for everything related with the simulation itself: sensor rendering, computation of physics, updates on the world-state and its actors and much more. As it aims for realistic results, the best fit would be running the server with a dedicated GPU, especially when dealing with machine learning.
The client API is implemented in python and is responsible for the interaction between the autonomous agent and the server via sockets. The client sends commands and meta-commands to the server and receives sensor readings. The commands control the vehicle including steering, throttling and braking. While the meta-commands control the behavior of the server. They are used for resetting the simulation, changing the properties of the environment (weather conditions, density of the vehicles and pedestrians) and modifying the sensor suite. [19]

Carla allows for flexible configuration of the agent's sensor suite. Some of these sensors are **RGB camera** which acts as a regular camera capturing images from the scene, **Depth camera** which provides a view over the scene codifying the distance of each pixel to the camera (also known as **depth buffer** or **z-buffer**) and **Semantic Segmentation camera** which classifies every object in the view by displaying it in a different color according to the object class, for example: pedestrians appear in a different color than the vehicles. Down below, figure 3.2 illustrates two of sensing modalities provided by Carla. From left to right: RGB camera and Semantic Segmentation camera. [20]

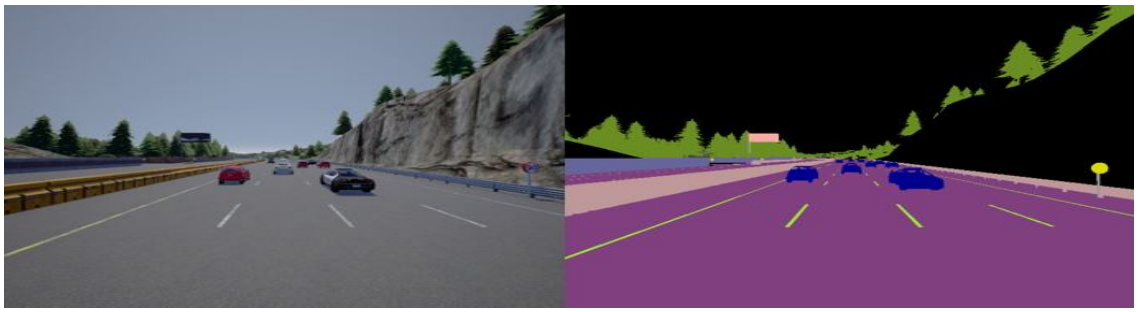


Figure 3.2: Two of sensing modalities provided by Carla

There are other sensors such as: **Lidar**, it simulates a rotating Lidar implemented using ray-casting, at which the points are computed by adding a laser for each channel distributed in the vertical field of view, then the rotation is simulated computing the horizontal angle that the Lidar rotated this frame and doing a ray-cast for each point that each laser was supposed to generate this frame. Another sensors, **Collision sensor** which registers an event each time the actor collisions against something in the world and **Lane Invasion sensor** which registers an events each time the actor crosses a lane marking. [20]

3.2 Literature Review

Accidents have spread significantly, either because of the dispersal of drivers doing something, or external factors resulting from the presence of cars or the appearance of people. As for the car while driving, Hence, some papers appeared attempting to solve these problems and reach solutions for how to deal when these problems are about to happen,as we found in Forward Collision Warning (FCW), the vehicle has the ability to warn the driver that there is an object in front of the vehicle. The driver has the responsibility of taking reasonable action. However, in Automatic Emergency Braking (AEB) the vehicle starts taking action through braking in case the vehicle comes close to an object in front of it. These actions are taken by the ego-vehicle based on integrating advanced sensors like Laser, Camera, and Radar. However, these vehicle's actions lead to the occurrence of many crashes, in addition to being a source of congestion because of its poorness[21].

According to this paper[22],we found that they tried to solve the problems that happened due to the distraction outside the moving vehicle ,as the 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.The path planning problem is one of the most important problems which autonomous driving cars face, as it is a safety-critical task, and needs full knowledge of the surrounding environment. During the last ten years, path planning problem was tackled using two approaches: multi-stage pipeline, and single-stage approach.The multi-stage approach decomposes the problem into the following blocks: a) perception which is responsible for perceiving the surrounding environment, b) trajectory prediction for the surrounding objects, c) trajectory planning which compute the trajectory depends on the perception and prediction for the environment, and d) control block which is responsible for taking the good actions depending on the whole information which are gathered by the previous block.

Their data collection pipeline consists of three main blocks, The first block is the waypoints generator which is considered as the most important block in the data collection phase.Waypoints are generated based on the well-defined map using CARLA simulator[19], then a postprocessing phase is applied on these waypoints to act as the traditional ground truth for path planning functionality. The second block is a PID controller which is composed of proportional, integral and derivative components which are tuned to achieve a smooth performance for the vehicle motion following the previously generated waypoints by controlling three vehicle actions: throttle, steering angle and brake. The third block is the noise generator which is inspired by CARLA simulator[19],they collected 1500 episodes which contain 440k data frames, each data frame consists of 4 images from the forward camera, the right camera, the left camera and the backward camera with its corresponding measurements which consist of throttle, brake, steering percentage and forward speed.The data is collected covering various and different scenarios, our vehicle is moving and the other vehicles are static.

They used Convolution neural network as a network architecture, and they made a concatenation between the images ,as a four separated images representation depends on the camera cocoon vehicle's installation, it simply takes the four separated images which are front, right, left, and rear views as input to a DNN.
They used Bird's eye view projection,as from the four cameras, a bird's eye view image is generated ,and their results were satisfied and reached to the least loss made their model work efficiently .

3.3 Simulation Setup

3.3.1 How data is generated?

- We used Carla simulator **0.9.6 version**.
- There are a lot of features for this version, here are some of them:
  - changing maps at runtime.
  - lane change extension.
  - Different towns with many lanes.
- Steps of generating data:
  - First, our car is moving using proportional-integral differential control(PID) and waypoints (we give it waypoint and a speed which is translated into steer, throttle and brake).

    The parameters of PID: [23]
    - First, a pure gain KP that scales the vehicle acceleration based on the speed error. This ensures that the vehicle is accelerating in the correct direction with the magnitude proportional to the error.
    - Second, in integral term KI sets up the output based on accumulated past errors. This ensures the steady steed errors are eliminated for ramp referencing.
    - Finally, the derivative term KD dampens the overshoot caused by the integration term.

    The values which are used: KP = 0.8, KD = 0.001, KI = 0.1
  - At the beginning, we calculate the distances between our car and the other cars that are in the same lane and in front of our one.
  - At a specific limit (6m) it begins to see whether there is a possibility for lane change or not (according to the street restrictions). If there is a possibility, it would be right or left or both. If it is less than 5m, it will stop.

- If it is:
  - Right:
    First, we will calculate the distance between the waypoint of the same location of our car in the right lane and the other cars that are in the same lane and in front of our one or behind. We get the minimum distance and its location.
    - If the right lane is empty, it will turn right.
    - As long as the difference between ours and the nearest car that is behind is $<$ 3m, it will stop otherwise it will calculate the minimum distance, if 0 $<=$ the minimum distance $<$ 9m, it will stop otherwise it will turn right.

  - Left:
    First, we will calculate the distance between the waypoint of the same location of our car in the left lane and the other cars that are in the same lane and in front of our one or behind. We get the minimum distance and its location.
    - If the left lane is empty, it will turn left.
    - As long as the difference between ours and the nearest car that is behind is $<$ 3m, it will stop otherwise it will calculate the minimum distance, if 0 $<=$ the minimum distance $<$ 9m, it will stop otherwise it will turn left.

  - Both:
    The default will be left. So, we will check the left first.
    First, we will calculate the distance between the waypoint of the same location of our car in the left lane and the other cars that are in the same lane and in front of our one or behind. We get the minimum distance and its location.
    - If the left lane is empty, it will turn left.
    - As long as the difference between ours and the nearest car that is behind is $>$ 3m and the minimum distance $>$ 9m, it will turn left.
    - If it is not in this range, we will check the right lane. We will first calculate the distance between the waypoint of the same location of our car in the right lane and the other cars that are in the same lane and in front of our one or behind. We get the minimum distance and its location.
      - If the right lane is empty, it will turn right.
      - As long as the difference between ours and the nearest car that is behind is $<$ 3m, it will stop otherwise it will check the minimum distance, if 0 $<=$ the minimum distance $<$ 9m, it will stop otherwise it will turn right.

3.3.2 Dataset

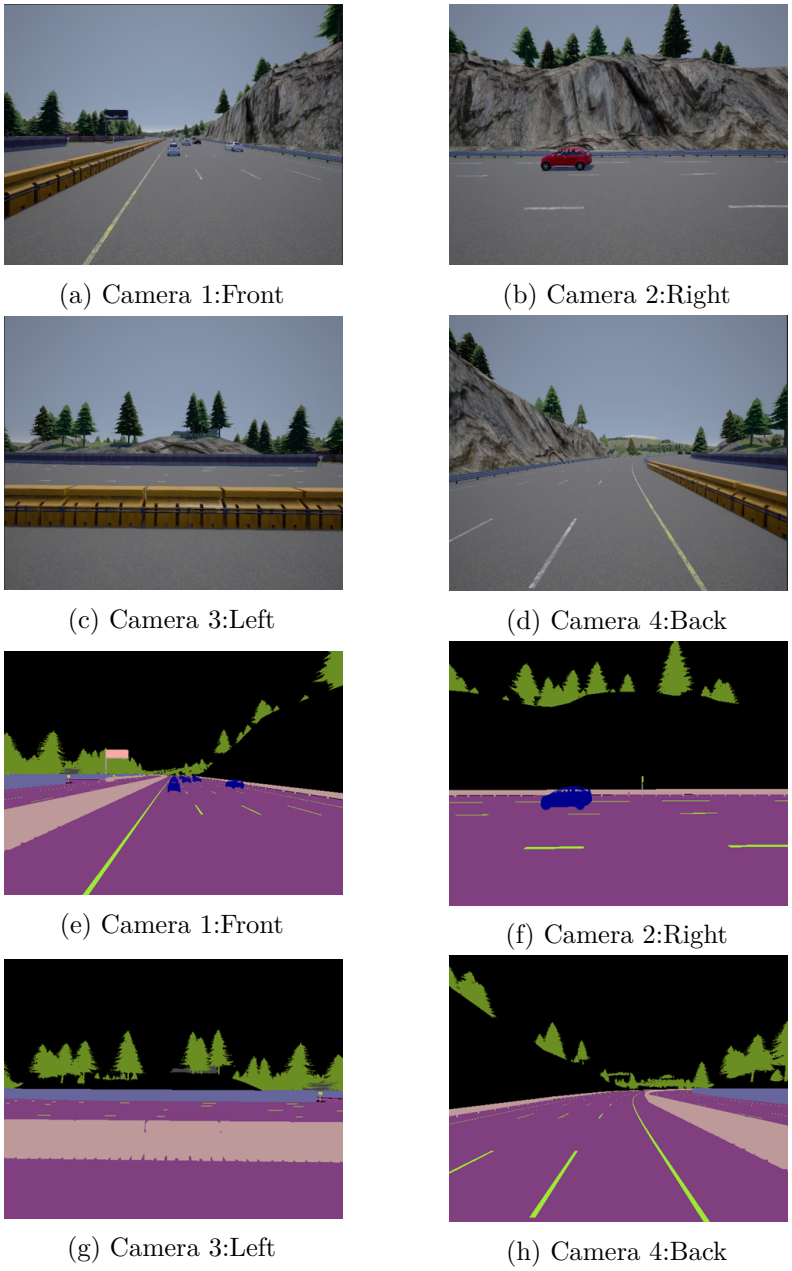
It is a group of images with 27 labels that had been collected under a some specific assumptions.

Assumptions:

1. We collect data from highway.
2. Town is divided into sections according to x and y values, as the town is divided into 8 sections ,every section has it's own x and y.
3. Other cars are obtained randomly from specific locations.First, we made the car walk in every section of the town manually .Then, along the car is walking we get the location (x, y and z points) and save it at an external text folder. Finally, the cars we installed in the simulator takes random locations from the text folder that we had.
4. Other cars are static,our car is the only one that walk with a determined steer , throttle for cases of walking forward and turning whatever right or left ,and a determined brake equals one for cases of stopping the car.
5. We are using 4 cameras.
    - First one is to collect the images from the front view,it's yaw angle is 0.0 deg and roll angle is 0.0 deg.
    - Second one is to collect the images from the right view,it's yaw angle is 90.0 deg and roll angle is 0.0 deg.
    - Third one is to collect the images from the left view,it's yaw angle is -90.0 deg and roll angle is 0.0 deg.
    - Fourth one is to collect the images from the backward view,it's yaw angle is 180.0 deg and roll angle is 0.0 deg .
    - All cameras are on the top of the car.

Dataset and labels :

- Dataset was collected using a 4 cameras to get the RGB images and 4 sensors to get the semantic segmentation images and the FPS is 20 as shown in the figure 3.3



(a) Camera 1:Front (b) Camera 2:Right

(c) Camera 3:Left (d) Camera 4:Back

(e) Camera 1:Front (f) Camera 2:Right

(g) Camera 3:Left (h) Camera 4:Back

Figure 3.3: The RGB and Semantic images getting from the four cameras

- Dataset is divided into training data ,validation data and test data
  (a) Trainin data : 14,313 for RGB and 14,313 for Semantic.
  (b) Validation data : 2,362 for RGB and 2,362 for Semantic.
  (c) Test data : 2,031 for RGB and 2,031 for Semantic.

- During taking the images that express the course of the car, its actions and the surrounding environment,we collect labels for the four images were taken by the four cameras in the same instant which include the location of the car in that instant, it's throttle, gear, steer, brake, velocity, rotation, lane invasion, sensor of collision, weather of the town and some other labels that describe the car and the surrounding environment .
- Total number of labels is 27,as shown in figure 3.4

Figure 3.4: the labels

3.3.3 Preprocessing

- **Why Preprocessing ?**
  - The curse of Dimensionality.
    - The quantity of training data grows exponentially with the dimension of the input space.
    - In practice, we only have limited quantity of input data .
      - Increasing the dimensionality of the problem leads to a poor representation of the mapping.
- **Steps:**
  1. We have a RGB semantic images from four cameras (forward, backward ,left and right) as shown in the figure1 ,there is a little overlapping between the four cameras ,as FOV for all cameras is " 110 degree " .
  2. We made a concatenation between every out coming four images from the four images for each instant to be one image ,as The front image has been attached to the image on the right, with the image on the left,with the image at the backward and we made this for RGB ones and the semantic ones as shown in the figure 3.3 .

3. Data balancing :

   (a) There are four classes (follow , right ,left and stop):

      – Follow : It includes every out coming image as long as the car is moving forward in the lane without being exposed to any obstacle.

      – Right :It includes every out coming image when the car take the decision to go right according to the conditions that were mentioned above.

      – Left :It includes every out coming image when the car take the decision to go left according to the conditions that were mentioned above.

      – Stop :It includes every out coming image when the car take the decision to stop according to the conditions that were mentioned above.

   (b) Balanced data to have an equal number of samples in each class.

   (c) Training data : 14,313 for RGB 14,313 for Semantic each class has 3,578 images.

   (d) Validation data : 2,362 for RGB 2,362 for Semantic each class has 591 images.

   (e) Test data : 2,031 for RGB 2,031 for Semantic each class has 508 images.

   (f) Different cars:

      – We collect the data of validation and test with a different cars from we used during collecting the training data .

      – The cars we used (chevorlet,citroen,mini.cooper,dodge,nissan,volks and mercedes)

4. concatenation :
   it aim to concatenate the four images which were being collected from the four cameras (front,right,left,back) each instant to act as one image ,as shown in figure 3.5 and figure 3.6 .

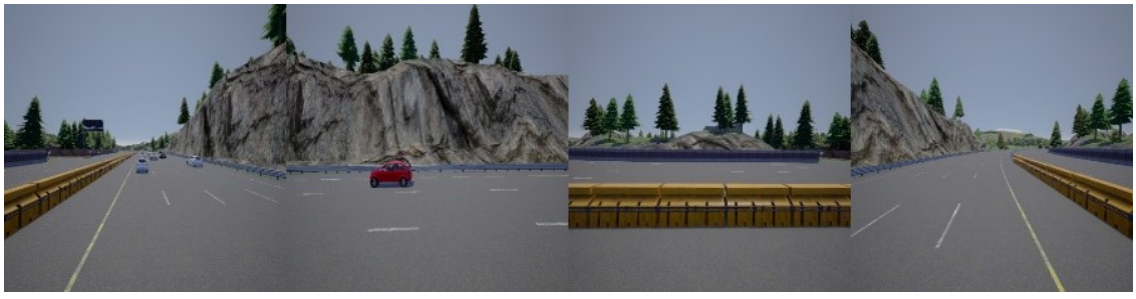


Figure 3.5: Concatenation of four RGB images in one image

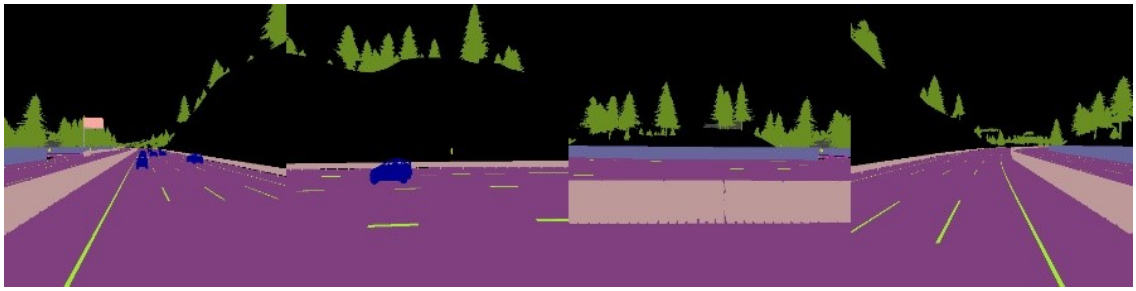


Figure 3.6: Concatenation of four Semantic images in one image

5. Augmentation :
   It aims to increase data set,seeing different shapes of images that can be happened at the real life during taking the pictures to collect the data set as a result human errors such as the images may be shifted or as a result of climate changes and improving the performance ,as the machine will see a variation of data shapes. We made an offline augmentation by using **imgaug** library instead of Keras online data augmentation

   - Applying different transformations,as shown in figure 3.7 :

     (a) Rotation (small angle) : the image can be rotated with small angle.

     (b) Shear (small angle) : the image can be sheared with small angle due to some human errors.

     (c) Addtion :Add a value to all pixels in an image

     (d) Multiplication : Multiply all pixels in an image with a specific value, thereby making the image darker or brighter

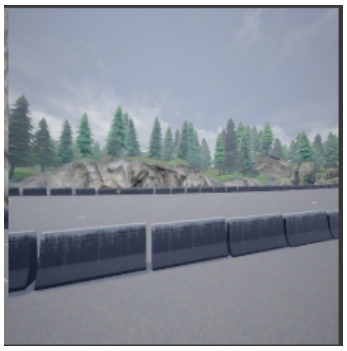
     (e) Addative Gaussian Noise :Add noise sampled from gaussian distributions elementwise to images

     (f) Rain

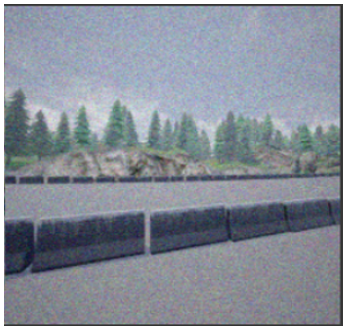
     (g) Clouds

   - We applied a group of the previous transformations on a group of data ,as we did not apply each transform on each image.

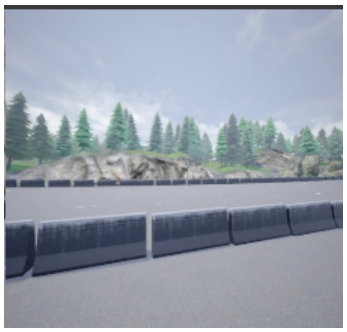
   - After applying augmentation , data set was increased five times the old data,as it was increased from 14313 to 71565.



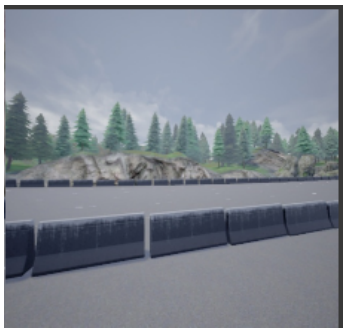
(a) The original image



(b) The image after applying addative gaussian noise



(c) The image after applying Multiplication



(d) The image after applying addtion

Figure 3.7: Applying some transformations on an image

3.4 Materials and Methods

Our problem is a regression problem in which that we have an input image and the output is that the prediction of car control signals: throttle which is a continuous value between 0 and 1, steer which is a continuous value between -1 and 1, and brake which is discrete value 0 or 1. We aim to reduce the loss function given by this equation 3.1

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2 \tag{3.1}$$

So we tried more than one approach to minimize loss function to be able to make the car take the right action. If there is an obstacle in front of it, it will check both; right lane and left lane in case that our car in the middle lane, if it is in the most right lane, it will check the left only and if it is in the most left lane; it will check the right only. According to prediction, it will decide to go left, right or it will stop. Moreover, if there is no obstacle in front of it, it will go straight. We have mainly two approaches: CNN and LSTM. With CNN, we have tried different models such as:

- CNN with RGB based on one image.
- CNN with RGB based on four images.
- CNN with RGB based on one image with augmentation.
- CNN with RGB based on four images with augmentation.
- CNN with Semantic Segmentation based on one image.
- CNN with Semantic Segmentation based on four images.
- CNN with Semantic Segmentation based on one image with augmentation.
- CNN with Semantic Segmentation based on four images with augmentation.
- CNN with Gray Scale based on one image.
- LSTM with Gray Scale based on one image.

3.4.1 First approach:CNN with branching

In this approach, we use CNN as it takes the input frame RGB image, Semantic Segmentation image, or Gray Scale image and it gives us the output of control signals: throttle, steer, and brake which makes the vehicle enables to move and takes different action. The main layers in the architecture are the convolution layer, dropout layer, batch normalization layer, pooling layer, flatten layer, and fully connected layer as shown in figure 3.8 and figure 3.10.The detailed CNN architecture shows the total number of parameters and the output shape of each layer as shown in table 3.1.

We will start to talk about the function of each layer and how it helps us to get our target output.

- Convolution layer : Convolution layer: This layer is used to extract features from a source image. Our source image here is (224,224,3) in the case of RGB image and Semantic Segmentation image and (224,224,4) in the case of Gray Scale image; we will discuss this in details later. Convolution helps with blurring, sharpen, edge detection, or other operations that can help the machine to learn.
- Drop-out layer:This layer is a technique used to prevent a model from over-fitting.We use dropout here of 0.1 with convolution layers and 0.5 with dense layer as shown in figure 3.8 and table 3.1.
- Batch Normalization layer: This layer to improve speed and performance.
- Pooling layer: This layer reduces the image dimensionality without losing important features.
- Flatten layer: This layer is that layer where all the inputs from one layer are connected to every activation unit of the next layer. In most popular machine learning models, the last few layers are full connected layers that compile the data extracted by previous layers to form the final output. Here we have more than one activation function because of the different value range of throttle, steer and brake so that the idea of the branching model comes from here and also the general reason to use activation function is that activation function is a function that is added into an artificial neural network to help the network learn complex patterns in the data.
  - Branch A : Sigmoid function in which the input to the function is transformed into a value between 0 and 1 which is suitable for throttle values' range from 0 to 1 as shown in figure 3.9.
  - Branch B : Tanh function in which the input to the function is transformed into a value between -1 and 1 which is suitable for steer values' range from -1 to 1 as shown in figure 3.9.
  - Branch C : Sigmoid function like Branch A and it is suitable for brake values' range from 0 to 1 as shown in figure 3.9.
  - Concatenate : This is used to concatenate the three branches A, B, and C to be the output of CNN.
  - Optimizer : We use Adam optimizer here which is used to change the attributes of neural network such as weights and learning rate to reduce losses and get results faster. The main idea of learning rate is that it controls how quality the model is adapted to the problem and in our case we use the learning rate of 0.001.
    For semantic segmentation the input is also (224,224,3), the model takes the input as RGB and classifies every object in the view by displaying it in a different color according to the object class, for example, trees appear in a different color than a vehicle. This is the idea of semantic segmentation as mention in the previous sections.

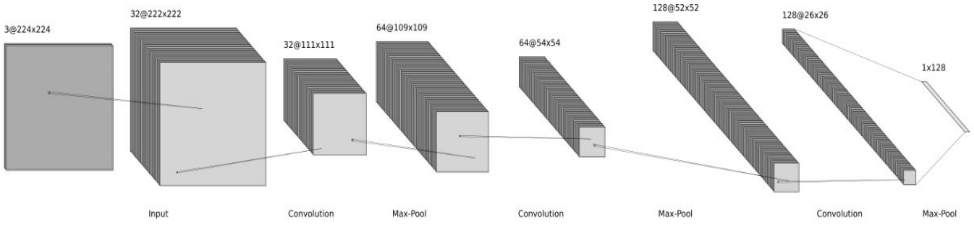


Figure 3.8: CNN architecture using branching

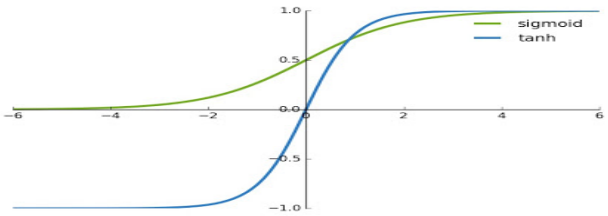


Figure 3.9: Graph to show the difference between Sigmoid and Tanh activation function

Table 3.1: Detailed CNN architecture using branching

Layer	Output Shape	Number Of Parameters
Input Layer	(224,224,3)	0
Convolution_1	(222,222,32)	896
Spatial_Dropout_1	(222,222,32)	0
Batch_Normalization_1	(222,222,32)	128
Max_Pooling_1	(111,111,32)	0
Convolution_2	(109,109,64)	18496
Spatial_Dropout_2	(109,109,64)	0
Batch_Normalization_2	(109,109,64)	256
Max_Pooling_2	(54,54,64)	0
Convolution_3	(52,52,128)	73856
Spatial_Dropout_3	(52,52,128)	0
Batch_Normalization_3	(52,52,128)	512
Max_Pooling_3	(26,26,128)	0
Flatten	86528	0
Dense_1	128	11075712
Batch_Normalization_4	128	512
Dropout_4	128	0
Dense_2	1	129
Dense_3	1	129
Dense_4	1	129
Concatenate	3	0

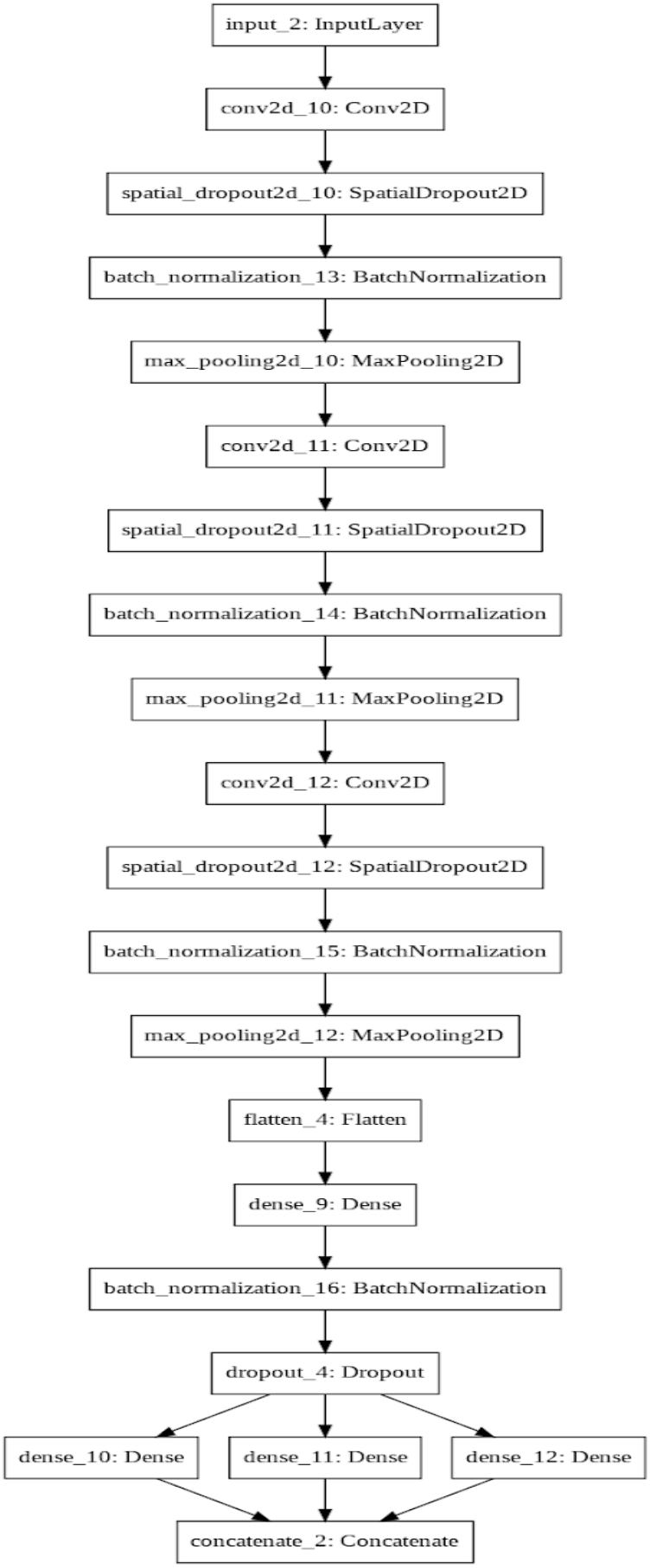


Figure 3.10: CNN architecture using branching flowchart

For GrayScale, the input to CNN is (224,224,4) as the idea is to convert the RGB image to grayscale and take the current film and last three frames in the past so the total number of channels becomes four which help the model to predict the output control signals depend on the current and past frames as shown in figure 3.11 and the detailed architecture of CNN with gray scale as shown in table 3.2.

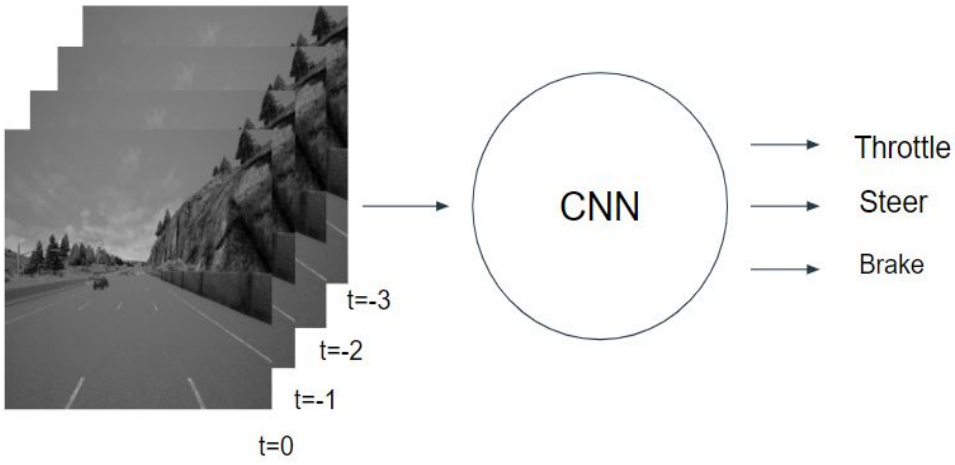


Figure 3.11: Temporal information in CNN input (Past)

Table 3.2: Detailed CNN architecture using branching Gray Scale

Layer	Output Shape	Number Of Parameters
Input Layer	(224,224,4)	0
Convolution_1	(222,222,32)	1184
Spatial_Dropout_1	(222,222,32)	0
Batch_Normalization_1	(222,222,32)	128
Max_Pooling_1	(111,111,32)	0
Convolution_2	(109,109,64)	18496
Spatial_Dropout_2	(109,109,64)	0
Batch_Normalization_2	(109,109,64)	256
Max_Pooling_2	(54,54,64)	0
Convolution_3	(52,52,128)	73856
Spatial_Dropout_3	(52,52,128)	0
Batch_Normalization_3	(52,52,128)	512
Max_Pooling_3	(26,26,128)	0
Flatten	86528	0
Dense_1	128	11075712
Batch_Normalization_4	128	512
Dropout_4	128	0
Dense_2	1	129
Dense_3	1	129
Dense_4	1	129
Concatenate	3	0

3.4.2 Second Approach : LSTM with branching

The main idea of LSTM that it has feedback connections. Here we have input shape of (4,224,224,1) the current frame and the last three past frames so the total number of frames becomes four all of them are grayscale and the one refers to the time dimension as shown in figure 3.12. The architecture is the same in the case of CNN, the difference is there is no Drop-out layer and Pooling layer. We use here the global average pooling layer instead of the flatten layer as shown in table 3.3 and figure 3.13. It still has the same function. The difference between flatten and global average pooling is that flatten will take a tensor of any shape and transform it into a one dimensional but keeping all values in the tensor and for global average pooling, it applies average pooling on the spatial dimensions until each spatial dimension is one, and leaves other dimensions unchanged.

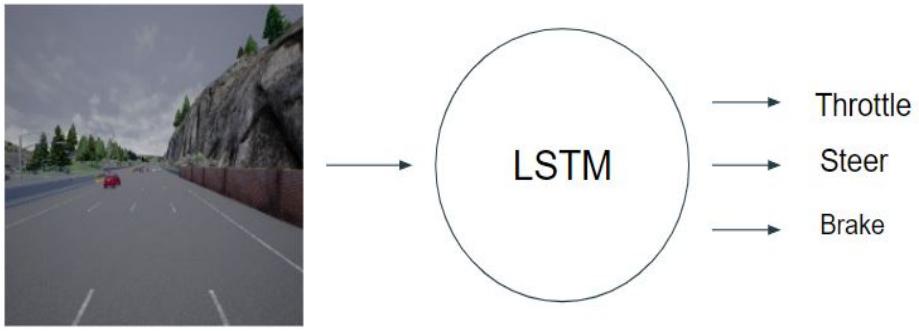


Figure 3.12: Sequence prediction using LSTM

Table 3.3: Detailed LSTM architecture using branching Gray Scale

Layer	Output Shape	Number Of Parameters
Input Layer	(4,224,224,1)	0
`Convolution_1`	(4,224,224,16)	9856
`Batch_Normalization_1`	(4,224,222,16)	64
`Convolution_2`	(4,224,244,32)	55424
`Batch_Normalization_2`	(4,224,224,32)	128
`Convolution_3`	(4,224,224,64)	221440
`Batch_Normalization_3`	(4,224,224,64)	256
`Global_Average_Pooling`	(1,64)	0
`Dense_1`	128	8320
`Batch_Normalization_4`	128	512
`Dropout_4`	128	0
`Dense_2`	1	129
`Dense_3`	1	129
`Dense_4`	1	129
Concatenate	3	0

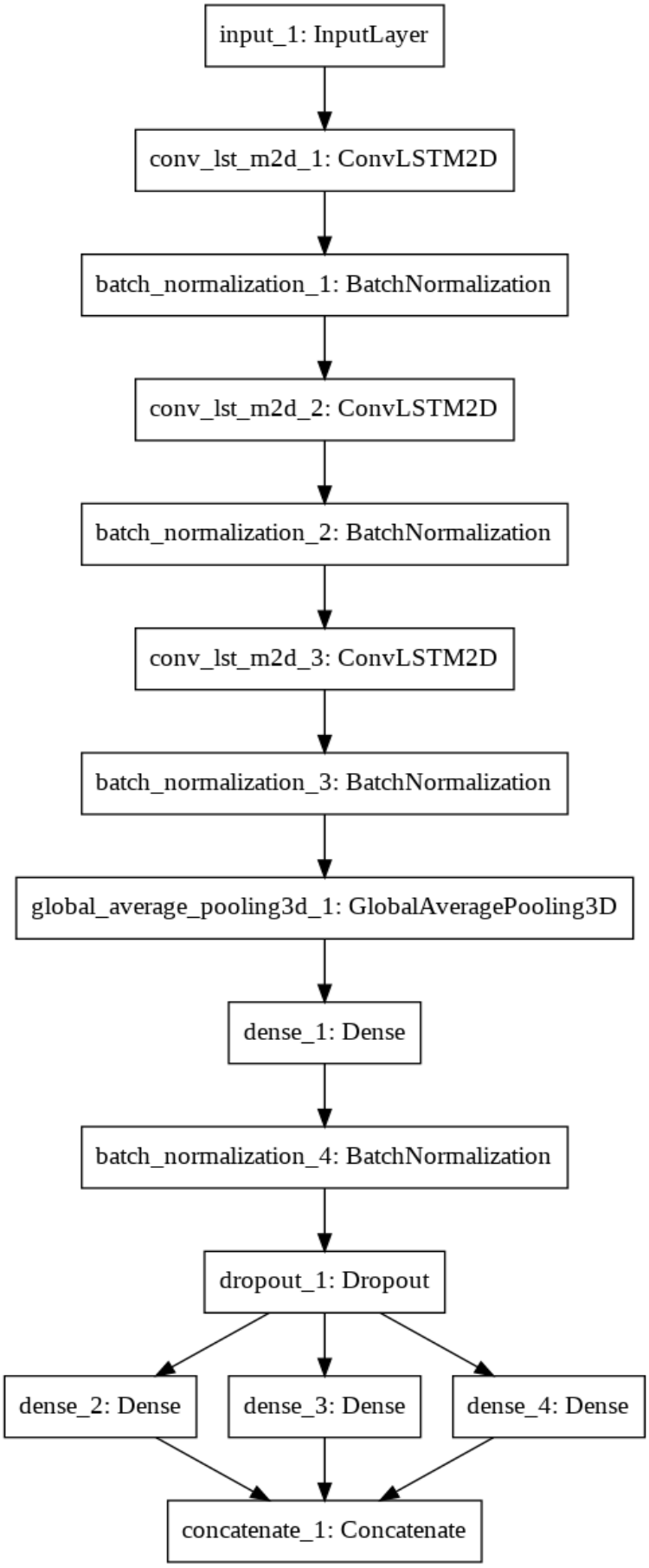


Figure 3.13: LSTM architecture using branching flowchart

3.5 Results and Discussion

We have a loss of throttle, steer, and break for training, validation, and test. We calculate them by equation 3.1. After that we get the training, validation, and test loss by taking the average of the throttle, steer, and break loss. According to this loss, we can compare among different architecture to get the least loss as we will show in this section.
We have also calculated the total number of parameters and inference time which is the process of using a trained machine-learning algorithm to make predictions.
We have tried the following architecture in case of one image from the front camera and case of four images from the front, back, right and left side cameras. Moreover, we have tried each of this architecture with and without augmentation.

3.5.1 First Approach:CNN with branching

3.5.1.1 Regression Model using branching RGB one image

We got the results of loss in throttle,steer and brake for one image RGB as shown in table 3.4 and after we applied the augmentation on Data-set for RGB one image, the results had been improved as shown in table 3.5.

Table 3.4: Results of Regression Model using branching RGB one image

Training loss	0.043
Training throttle loss	0.097
Training steer loss	0.022
Training brake loss	0.01
Validation loss	0.118
Validation throttle loss	0.177
Validation steer loss	0.055
Validation brake loss	0.122
Test Loss	0.072
Test throttle loss	0.151
Test steer loss	0.023
Test brake loss	0.042

- Number of parameters : 11.2 M
- Model size: 127.92 MB
- Inference Time
- GPU : Tesla T4

Table 3.5: Results of Regression Model using branching RGB one image with augmentation

Training loss	0.039
Training throttle loss	0.092
Training steer loss	0.019
Training brake loss	0.006
Validation loss	0.11
Validation throttle loss	0.162
Validation steer loss	0.051
Validation brake loss	0.117
Test Loss	0.064
Test throttle loss	0.138
Test steer loss	0.018
Test brake loss	0.036

- Number of parameters : 11.2 M -
- Model size: 127.92 MB
- Inference Time : 1 ms
- GPU : Tesla T4

3.5.1.2 Regression Model using branching RGB Four images

Here,We got the results of loss in throttle,steer and brake for four images RGB as shown in table 3.6 and after we applied the augmentation on Data-set for RGB four images, the results had been improved as shown in table 3.7.

Table 3.6: Results of Regression Model using branching RGB Four images

Training loss	0.051
Training throttle loss	0.105
Training steer loss	0.030
Training brake loss	0.017
Validation loss	0.099
Validation throttle loss	0.164
Validation steer loss	0.063
Validation brake loss	0.069
Test Loss	0.070
Test throttle loss	0.145
Test steer loss	0.031
Test brake loss	0.036

- Number of parameters : 11.2 M
- Model size: 127.92 MB
- Inference Time : 0.001456 sec
- GPU : Tesla T4

Table 3.7: Results of Regression Model using branching RGB Four images with augmentation

Training loss	0.034
Training throttle loss	0.079
Training steer loss	0.018
Training brake loss	0.004
Validation loss	0.078
Validation throttle loss	0.151
Validation steer loss	0.043
Validation brake loss	0.039
Test Loss	0.068
Test throttle loss	0.142
Test steer loss	0.015
Test brake loss	0.045

- Number of parameters : 11.2 M
- Model size: 127.92 MB
- Inference Time : 0.001417 sec
- GPU : Tesla T4

3.5.1.3 Regression Model using branching Semantic Segmentation one image

Here, We got the results of loss in throttle,steer and brake for one image Semantic Segmentation as shown in table 3.8 and after we applied the augmentation on Dataset for Semantic Segmentation one image, the results had been improved as shown in table 3.9.

Table 3.8: Results of Regression Model using branching Semantic Segmentation One image

Training loss	0.0399
Training throttle loss	0.0849
Training steer loss	0.02336
Training brake loss	0.01134
Validation loss	0.1143
Validation throttle loss	0.1709
Validation steer loss	0.06126
Validation brake loss	0.1109
Test Loss	0.071307
Test throttle loss	0.14177
Test steer loss	0.0414
Test brake loss	0.031

- Number of parameters : 11.2 M
- Model size: 127.92 MB
- Inference Time : 0.0014 sec
- GPU : Tesla T4

Table 3.9: Results of Regression Model using branching Semantic Segmentation One image with augmentation

Training loss	0.0156
Training throttle loss	0.0383
Training steer loss	0.00746
Training brake loss	0.00122
Validation loss	0.1016
Validation throttle loss	0.17706
Validation steer loss	0.04556
Validation brake loss	0.08219
Test Loss	0.0738
Test throttle loss	0.1689
Test steer loss	0.0285
Test brake loss	0.024

- Number of parameters : 11.2 M
- Model size: 127.92 MB
- Inference Time : 0.0013 sec
- GPU : Tesla T4

3.5.1.4 Regression Model using branching Semantic Segmentation Four images

Here,We got the results of loss in throttle,steer and brake for four images Semantic Segmentation as shown in table 3.10 and after we applied the augmentation on Data-set for Semantic Segmentation four images, the results had been improved as shown in table 3.11.

Table 3.10: Results of Regression Model using branching Semantic Segmentation Four images

Training loss	0.0350
Training throttle loss	0.0781
Training steer loss	0.0215
Training brake loss	0.0055
Validation loss	0.0979
Validation throttle loss	0.1573
Validation steer loss	0.0579
Validation brake loss	0.0783
Test Loss	0.0648
Test throttle loss	0.1431
Test steer loss	0.0301
Test brake loss	0.0212

- Number of parameters : 11.2 M
- Model size: 127.92 MB
- Inference Time : 0.00145045 sec
- GPU : Tesla T4

Table 3.11: Results of Regression Model using branching Semantic Segmentation Four images with augmentation

Training loss	0.0356
Training throttle loss	0.0810
Training steer loss	0.0189
Training brake loss	0.0068
Validation loss	0.0823
Validation throttle loss	0.1512
Validation steer loss	0.0508
Validation brake loss	0.0447
Test Loss	0.0598
Test throttle loss	0.1362
Test steer loss	0.0233
Test brake loss	0.0200

- Number of parameters : 11.2 M
- Model size: 127.92 MB
- Inference Time : 0.002448 sec
- GPU : Tesla K80

3.5.1.5 Regression Model using branching Gray Scale One image

Here,We got the results of loss in throttle,steer and brake for one image Gray Scale using time dimension as shown in table 3.12.

Table 3.12: Results of Regression Model using branching Gray Scale One image

Training loss	0.036
Training throttle loss	0.092
Training steer loss	0.011
Training brake loss	0.005
Validation loss	0.091
Validation throttle loss	0.256
Validation steer loss	0.014
Validation brake loss	0.003
Test Loss	0.038
Test throttle loss	0.097
Test steer loss	0.014
Test brake loss	0.004

- Number of parameters : 11.2 M
- Model size: 127.92 MB
- Inference Time : 0.000294 sec
- GPU : Tesla P100-PCIE-16GB

3.5.2 Second Approach:LSTM with branching

3.5.2.1 Regression Model using branching Gray Scale one image

Here, We got the results of loss in throttle,steer and brake for one image Gray Scale using time dimension as shown in table 3.13.

Table 3.13: Results of Regression Model using branching Gray Scale One image

Training loss	0.228
Training throttle loss	0.437
Training steer loss	0.217
Training brake loss	0.031
Validation loss	0.217
Validation throttle loss	0.405
Validation steer loss	0.232
Validation brake loss	0.014
Test Loss	0.216
Test throttle loss	0.414
Test steer loss	0.218
Test brake loss	0.017

- Number of parameters : 296,387
- Model size: 3.47 MB
- Inference Time : 0.028789 sec
- GPU : Tesla P100-PCIE-16GB

As we show in the previous tables the different results, after that we have tried each of these models on the Carla simulator. For RGB the results were not efficient enough to enable the vehicle to take the right action.
Semantic Segmentation with augmentation based on one image gives us the best results which make the vehicle to behave right while the four images did not give the best results as expected because the way of concatenation was not the best way to concatenate four images as the size of each image after concatenation becomes 224*896 and we resized it to become 224*224 to be ready as input for CNN so the image became unclear which makes the model was not enable to make a good prediction.
For CNN with Gray Scale,the results were better than the Semantic Segmentation but it did not work well in the Carla simulator.
LSTM was very slow in training and in the simulator.
At the end, we choose to work with CNN Semantic Segmentation based on one image with augmentation.

Chapter 4

Distraction Detection

In this chapter, a real-time driver distraction detection system is introduced. The system is built in an end-to-end approach, in which a single neural network infers whether the driver is in a safe-driving or a distracted state, and what kind of a distracted state is the driver in. The input to the neural network is an image captured by a camera mounted on the top right side of the driver. This end-to-end approach has the advantage of being more computationally efficient than a multi-stage system in which the classification process is based on multiple blocks combined together to form the final prediction. Different CNNs have been trained and tested for their classification performance on the used dataset. The trained CNNs include VGG16, ResNet50, InceptionV3, Xception, MobileNet and DenseNet. After the training process, model optimization and hardware acceleration algorithms as quantization and pruning are applied on the best performing model in order to compress and accelerate the neural network for hardware deployment. Model visualization is discussed, which is an essential part in the assessment of the model generalization on new unseen data.

Moreover, an end-to-end semantic segmentation system for hands and face is proposed. The system is built using only UNET for the segmentation process, which shows a satisfactory inference time, enabling the system to operate in a real-time environment.

4.1 Literature Review

Driver distraction is a serious problem that leads to the loss many lives. This section presents the related work in the field of distraction detection using Machine Learning and Deep Learning approaches, where various distraction detection systems have been proposed in order to limit the number of accidents that occur due to driver distraction.

[24] introduced a distracted driving dataset with with a side view of the driver. The distraction classes included in the dataset are: talking on a cellular phone, eating a cake, grasping the steering wheel and operating the shift lever. The proposed distraction detection system is based on a contourlet transform for feature extraction

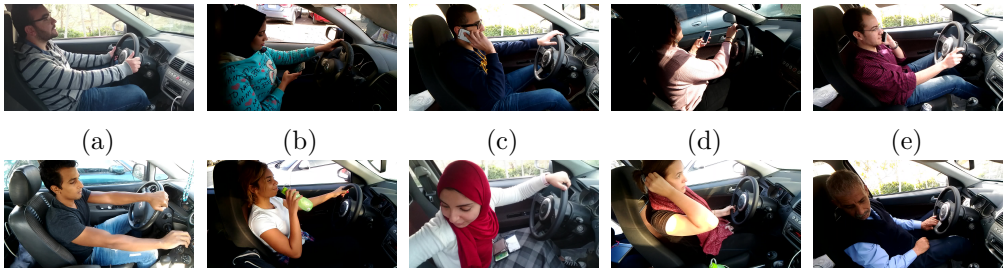
and random forest as a classifier with a classification accuracy of 90.5%. [25] demonstrates that a multiwavelet transform improves the accuracy of the multi-layer perceptron classifier to 90.61%, which was previously reported to be 37.06% in [24]. [26] shows that using a support vector machine (SVM) with an intersection kernel, followed by a radial basis function (RBF) kernel achieves classification accuracies of 92.81% and 94.25%, respectively, in comparison with [24] and [25]. [27] proposes a convolutional neural network (CNN) solution with a classification accuracy of 99.78% on the Southeast University Distracted Driver Dataset [24]. The introduced distraction detection system uses pre-trained sparse filters as the first convolution layer parameters, followed by fine tuning the network on the actual dataset. [28] proposes a deep learning-based solution that consists of a genetically weighted ensemble of convolutional neural networks. The proposed system achieves a classification accuracy of 90% using 5 AlexNet and 5 InceptionV3 networks. Additionally, a thinned version of the system that operates in a real-time environment on a CPU-based system is proposed using two NasNetMobile networks with a classification accuracy of 84.64%. [29] collected a driver distraction detection dataset using a developed assisted-driving test bed. The proposed distraction detection system is based on a GoogleNet model which achieves an accuracy of 89% and operates at a frequency of 11 Hz on a Jetson TX1 embedded computer board.

4.2 Simulation Setup

In this section, the datasets used to train the deep learning models are discussed. Additionally, preprocessing techniques applied on the datasets as a preparation to the training process are demonstrated.

4.2.1 Dataset

Two datasets are used in this work. The first dataset is the AUC Distracted Driver Dataset [30] [31], while the second dataset is the State Farm Distracted Driver Detection Dataset [32]. Both datasets contain images for distracted drivers with 10 different classes. Figure 4.2 and 4.1 show the 10 different classes from the AUC and State Farm datasets, respectively. Table 4.1 and 4.2 demonstrate the dataset distribution for both AUC and State Farm datasets, respectively.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 4.1: AUC Distracted Driver Dataset. (a) Safe driving (b) Texting - right (c) Talking on the phone - right (d) Texting - left (e) Talking on the phone - left (f) Operating the radio (g) Drinking (h) Reaching behind (i) Hair and makeup (j) Talking to a passenger.

(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 4.2: State Farm Distracted Driver Detection Dataset. (a) Safe driving (b) Texting - right (c) Talking on the phone - right (d) Texting - left (e) Talking on the phone - left (f) Operating the radio (g) Drinking (h) Reaching behind (i) Hair and makeup (j) Talking to a passenger.

Table 4.1: AUC dataset distribution

Dataset	Number of samples
Train	10555
Validation	1123
Total	11678

Table 4.2: State Farm dataset distribution

Dataset	Number of samples
Train	18732
Validation	3692
Test	79726
Total	102150

4.2.2 Preprocessing

4.2.2.1 Unique Drivers Problem

The problem of unique drivers is crucial to inspect in order to achieve a degree of satisfactory model generalization. When splitting the State Farm dataset into training and validation sets, the splitting is carried out so that no driver in the training set appears in the validation set and vice versa. This ensures that the validation results reflect the performance of the model on new unseen drivers. If some drivers in the validation set appear in the training set, the validation set results will not be a proper evaluation metric to assess model generalization, as the training and validation sets will be highly correlated. The AUC dataset is split into training and validation sets based on unique drivers, so this preprocessing step is performed only on State Farm dataset, by selecting some drivers for the validation set and excluding them from the training set.

4.2.2.2 Augmentation

The purpose of augmentation is to increase the number of samples in the dataset and introduce artificial alterations in the original images that the trained model may encounter in the future after training, which in turn increases the possibility of generalization. In order to augment the training set, the following transformations are applied on every image:

- Rotation and Shear
- Motion Blur
- Cropping
- Additive Gaussian Noise
- Addition
- Multiplication

Figure 4.3: Original sample from the AUC dataset. The class of image is talking on the phone - right

(a) (b)

(c)

(d) (e)

(f)

Figure 4.4: Augmented images. (a) Rotation and Shear (b) Cropping (c) Blur (d) Multiplication (e) Additive Gaussian Noise (F) Addition

4.3 Materials and Methods

4.3.1 End-to-End Classification System

Different experiments have been carried out in order to determine the best architecture and approach for classification. The first experiments are performed on the State Farm dataset only, while in the second experiment, the AUC and State Farm datasets have been merged into a larger dataset. Adding the two datasets together shows a better classification performance and generalization, due to the variance in the lighting conditions, recording cameras and drivers. The experiments are performed by utilizing Google Colaboratory resources, which are Nvidia Tesla P100-PCIE-16GB GPU, Intel ® Xeon ® CPU @ 2.20 GHz, and 25GB RAM.

The experiments including only the State Farm dataset are carried out using two approaches. In the first approach, a custom CNN has been built and trained from scratch. Different architectures have been tested for their generalization after training. Figure [ref] shows the best performing CNN architecture. The architecture consists of four convolutional blocks, each block consists of three layers: convolutional layer, spatial dropout layer and a max-pooling layer. The number of filters of the convolutional layers increases as the network depth increases, it starts by 64 filters in the first layer and ends by 256 filters in the the third and fourth layers. A flattening layer lies after the last convolutional block, followed by a fully connected layer of 1024 neurons. A last fully connected layer of 10 neurons is added for classification. Dropout layers are introduced between the flattening layer and the first fully connected layer, and between the two fully connected layers to avoid overfitting. The total number of model parameters is 2.1M.

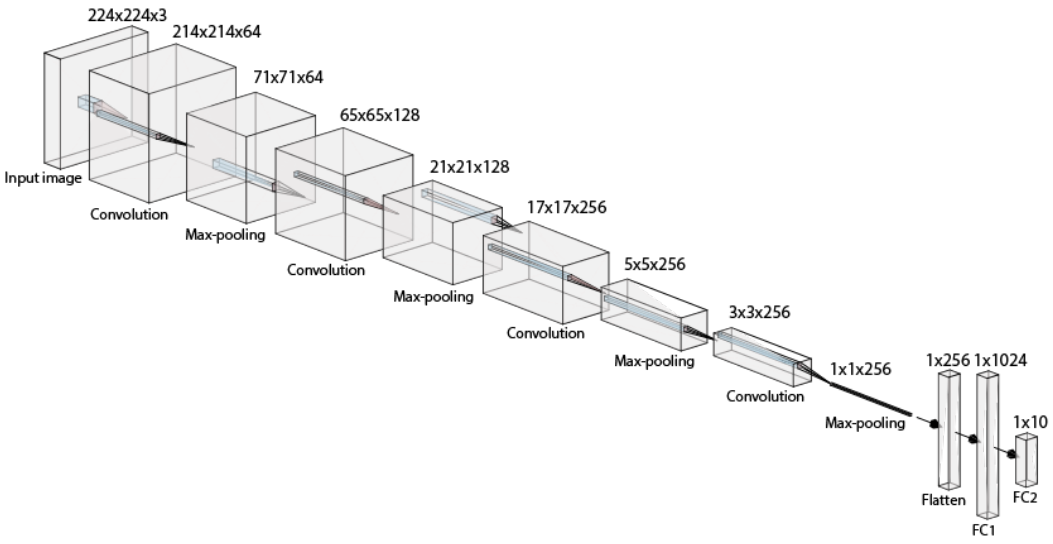


Figure 4.5: Custom CNN architecture

The second approach is based on Transfer Learning. Various pre-trained CNNs on the ImageNet dataset have been trained on the State Farm dataset. The CNNs used in the experiment are VGG16, ResNet50, InceptionV3, Xception, MobileNet and DenseNet. The last fully connected layers in the CNNs have been removed and replaced by new fully connected layers with randomly initialized weights. The training process is performed using ImageNet pre-trained weights as an initialization, then the whole network is trained without freezing the weights of any convolutional layer, which shows better results than keeping certain weights unchanged. The results of the two approaches are discussed in section 4.4. The Transfer Learning based experiment shows that ResNet50 outperforms other pre-trained CNNs used in the experiment. Consequently, section 4.3.1.1 discusses the architecture of ResNet50. An ensemble between all the trained CNNs is performed by calculating the average of the predicted probabilities by all the models for each class on every sample of the dataset. The advantage of the ensemble approach is that it improves the classification accuracy on the validation and test sets, on the other hand, it is computationally expensive due to the existence of six models contributing to the final prediction, leading to more memory usage and greater inference time.

4.3.1.1 ResNet50

Residual Networks is a type of neural networks proposed in [33] that is used for various computer vision tasks. This networks is the winner of ImageNet challenge in 2015. ResNet introduces the idea of skip connections which helps in training deep neural networks by preventing the problem of vanishing gradient descent. ResNet50 is a version of the ResNet architecture.

Figure 4.6 demonstrates the idea of the skip connection introduced in ResNet. The illustration on the left shows multiple stacked convolution layers, while the illustration on the right shows the same stacked convolution layers but with the original input added to the output of the convolutional block.

Figure 4.6: Skip connection image from DeepLearning.AI

The ResNet50 architecture consists of 5 stages each with a convolution and an identity block as shown in figure 4.7. Each convolutional block consists of 3 convolutional layers, and each identity block also consists of 3 convolutional layers. The ResNet50 architecture has around 23 million trainable parameters.

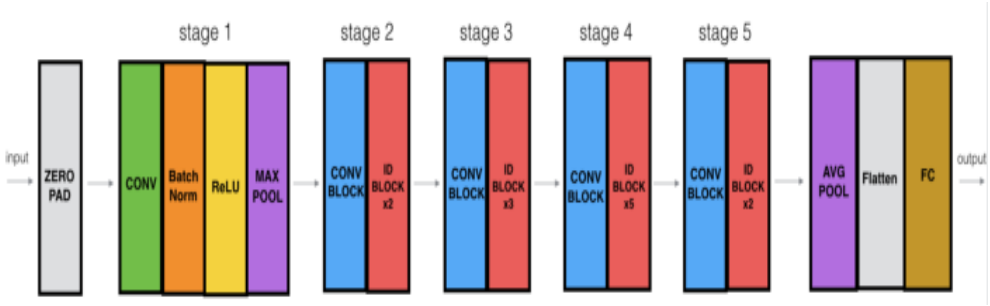


Figure 4.7: ResNet50 architecture

4.3.2 Face and Hands Semantic Segmentation

4.3.2.1 Image Segmentation

How does image segmentation work?

We can divide or partition the image into various parts called segments. It's not a great idea to process the entire image at the same time as there will be regions in the image which do not contain any information. By dividing the image into segments, we can make use of the important segments for processing the image. That, in a nutshell, is how image segmentation works.

An image is a collection or set of different pixels. We group together the pixels that have similar attributes using image segmentation.

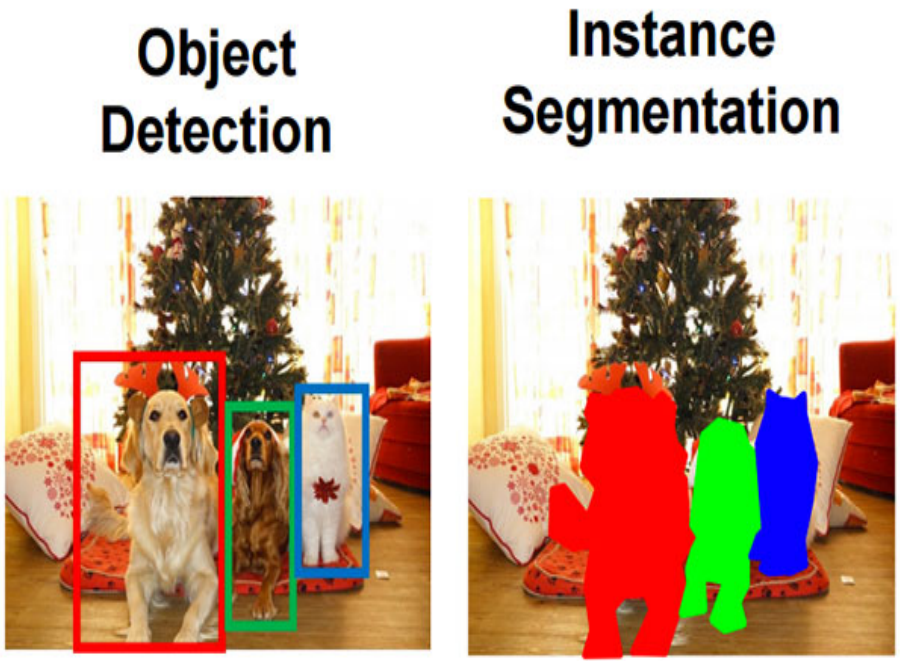


Figure 4.8: Image localization and segmentation

The different types of image segmentation:

We can broadly divide image segmentation techniques into two types. Consider the below images:

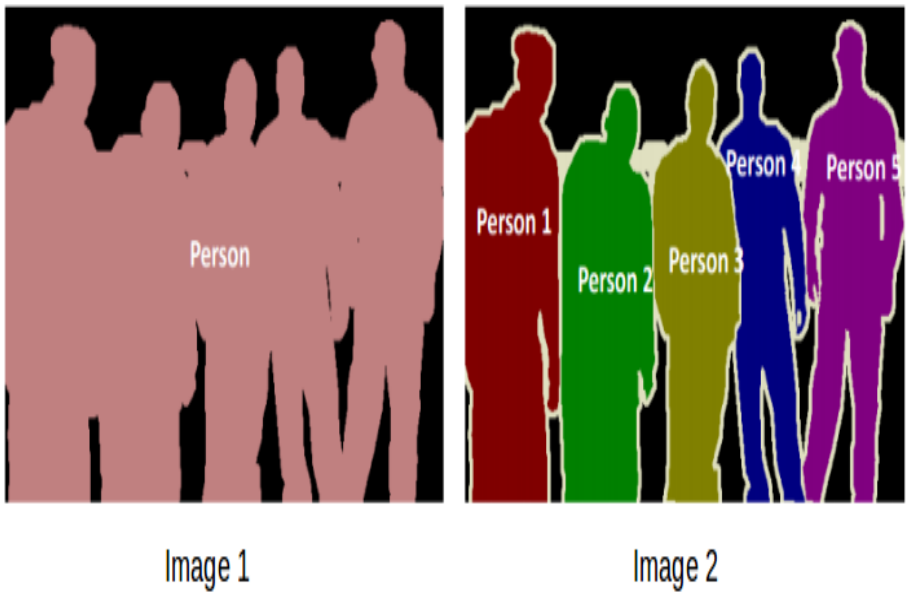


Figure 4.9: Semantic segmentation of the left, instance segmentation on the right

In image 1, every pixel belongs to a particular class (either background or person). Also, all the pixels belonging to a particular class are represented by the same color (background as black and person as pink). This is an example of semantic segmentation.

Image 2 has also assigned a particular class to each pixel of the image. However, different objects of the same class have different colors (Person 1 as red, Person 2 as green, background as black, etc.). This is an example of instance segmentation.

4.3.2.2 Region-based Segmentation

One simple way to segment different objects could be to use their pixel values. An important point to note is that the pixel values will be different for the objects and the image's background if there's a sharp contrast between them.

In this case, we can set a threshold value. The pixel values falling below or above that threshold can be classified accordingly (as an object or the background). This technique is known as **Threshold Segmentation**.

4.3.2.3 Edge Detection Segmentation

What divides two objects in an image? There is always an edge between two adjacent regions with different grayscale values (pixel values). The edges can be considered as the discontinuous local features of an image.

We can make use of this discontinuity to detect edges and hence define a boundary of the object. This helps us in detecting the shapes of multiple objects present in a given image. Now the question is how can we detect these edges? This is where we can make use of filters and convolutions.

Here's the step-by-step process of how this works:

- Take the weight matrix
- Put it on top of the image
- Perform element-wise multiplication and get the output
- Move the weight matrix as per the stride chosen
- Convolve until all the pixels of the input are used

4.3.2.4 Image Segmentation Based on Clustering

Clustering is the task of dividing the population (data points) into a number of groups, such that data points in the same groups are more similar to other data points in that same group than those in other groups. These groups are known as clusters.

One of the most commonly used clustering algorithms is k-means. Here, the k represents the number of clusters (not to be confused with k-nearest neighbor).

How k-means works?

1. First, randomly select k initial clusters.

2. Randomly assign each data point to any one of the k clusters.

3. Calculate the centers of these clusters.

4. Calculate the distance of all the points from the center of each cluster.

5. Depending on this distance, the points are reassigned to the nearest cluster.

6. Calculate the center of the newly formed clusters. Finally, repeat steps (4), (5) and (6) until either the center of the clusters does not change or we reach the set number of iterations.

4.3.2.5 Mask R-CNN

Mask R-CNN is an extension of the popular Faster R-CNN object detection architecture. Mask R-CNN adds a branch to the already existing Faster R-CNN outputs. The Faster R-CNN method generates two things for each object in the image:

1. Its class.

2. The bounding boxes coordinates.

Mask R-CNN adds a third branch to this which outputs the object mask as well. An intuition of how Mask R-CNN works on the inside:

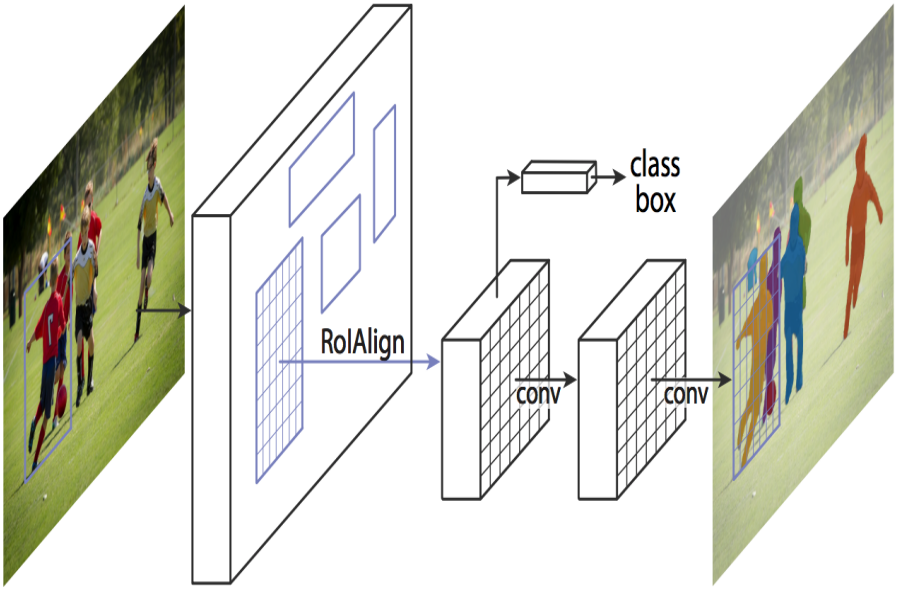


Figure 4.10: Mask R-CNN

1. We take an image as input and pass it to the ConvNet, which returns the feature map for that image.

2. Region proposal network (RPN) is applied on these feature maps,This returns the object proposals along with their objectness score.

3. RoI pooling layer is applied on these proposals to bring down all the proposals to the same size.

4. the proposals are passed to a fully connected layer to classify and output the bounding boxes for objects. It also returns the mask for each proposal.

Table 4.3: A comparison between algorithms

Algorithm	Description	Advantages	Limitations
Region Based Segmentation	Separates the objects into different regions based on some threshold value(s).	a. Simple calculations b. Fast operation speed c. When the object and background have high contrast, this method performs really well	When there is no significant grayscale difference or an overlap of the grayscale pixel values, it becomes very difficult to get accurate segments.
Edge Detection Segmentation	Makes use of discontinuous local features of an image to detect edges and hence define a boundary of the object.	It is good for images having better contrast between objects.	Not suitable when there are too many edges in the image and if there is less contrast between objects.
Segmentation based on Clustering	Divides the pixels of the image into homogeneous clusters.	Works really well on small datasets and generates excellent clusters.	a. Computation time is too large and expensive. b. k-means is a distance-based algorithm. It is not suitable for clustering non-convex clusters.
Mask R-CNN	Gives three outputs for each object in the image: its class, bounding box coordinates, and object mask	a. Simple, flexible and general approach. b. It is also the current state-of-the-art for image segmentation.	High training time.

4.3.2.6 Image Segmentation using Fully Convolutional Networks

1. **From Image Classification to Semantic Segmentation**

In classification, conventionally, an input image is downsized and goes through the convolution layers and fully connected (FC) layers, and output one predicted label for the input image, as follows:

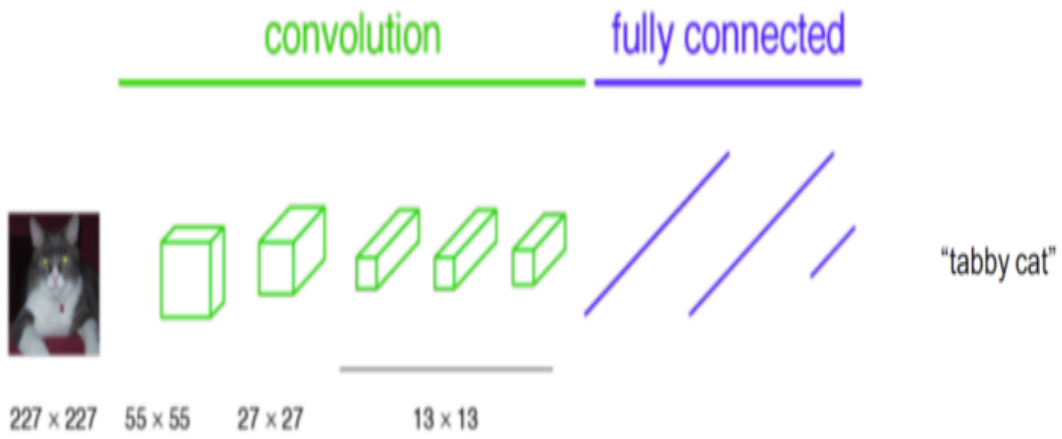


Figure 4.11: Image classification using CNN

Imagine we turn the FC layers into 1x1 convolutional layers:

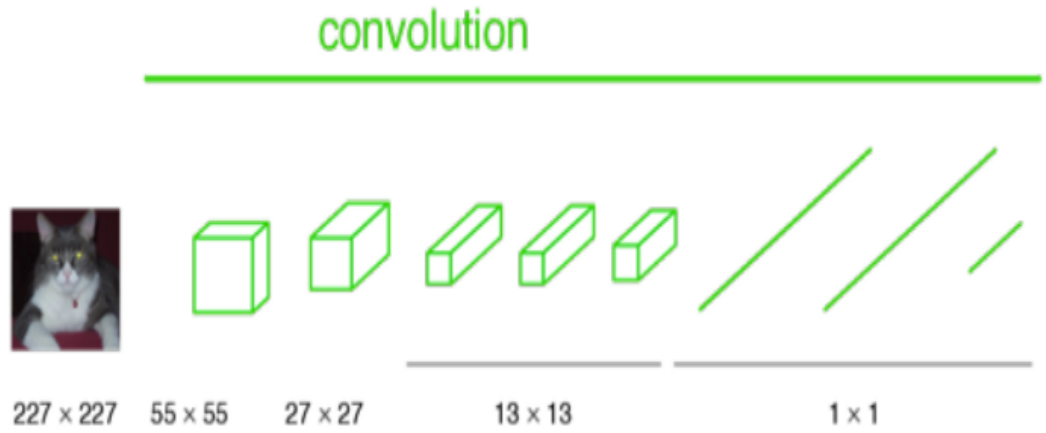


Figure 4.12: From image classification to semantic segmentation

And if the image is not downsized, the output will not be a single label. Instead, the output has a size smaller than the input image (due to the max pooling):

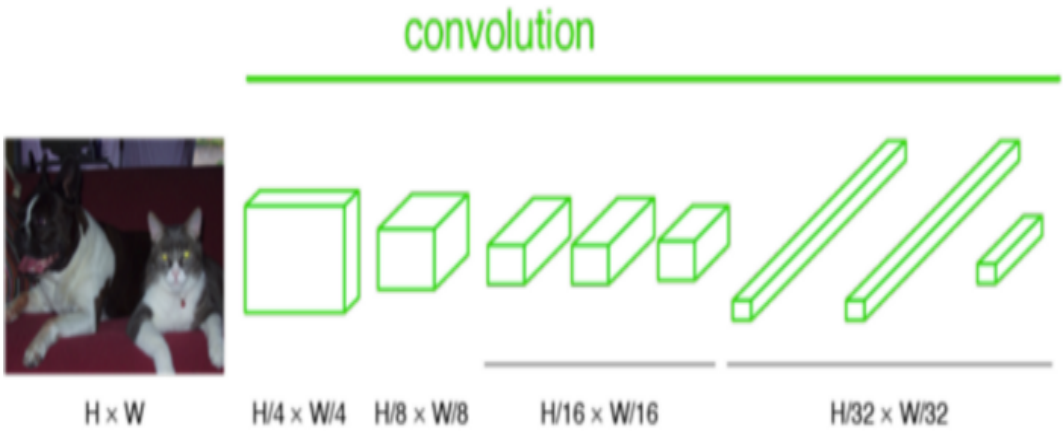


Figure 4.13: Feature map/Filter number along layers

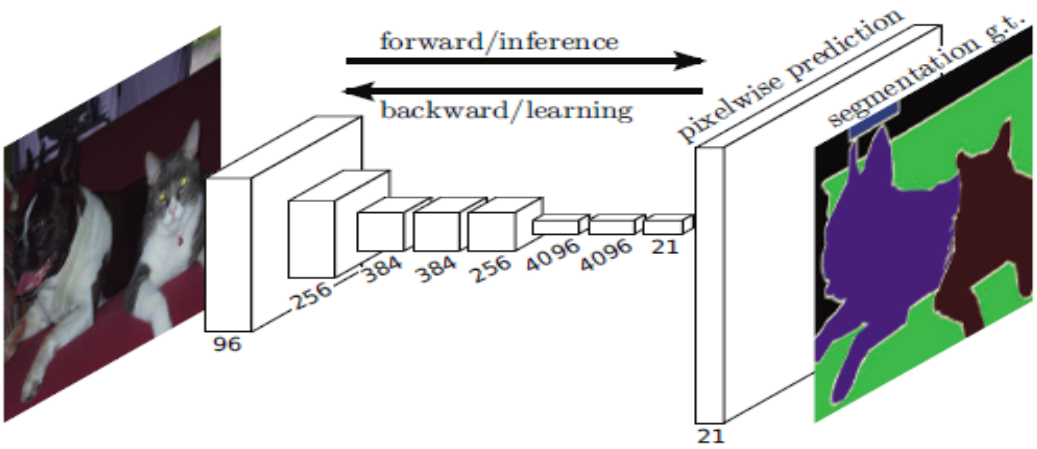


Figure 4.15: Feature map/Filter number along layers

If we upsample the output above, then we can calculate the pixel-wise output (label map) as below:

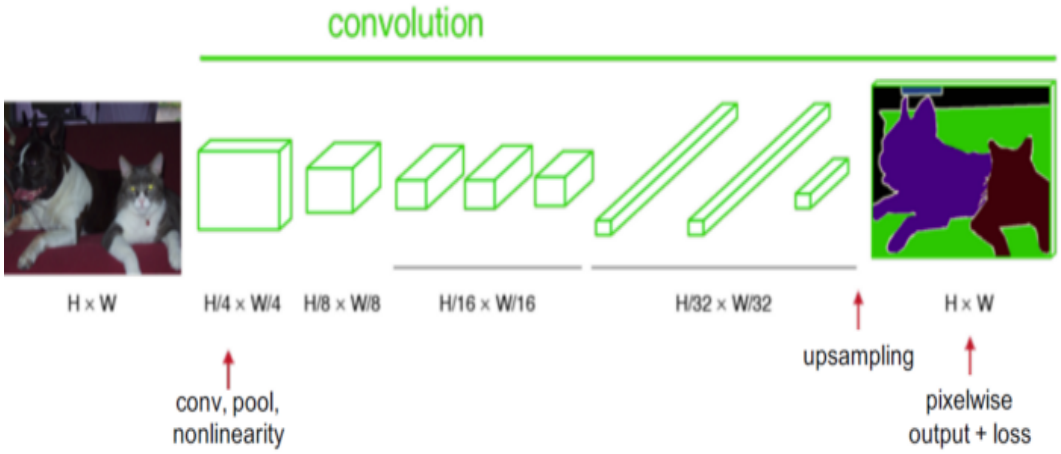


Figure 4.14: Feature maps

2. **Upsampling Via Deconvolution**

   Convolution is a process getting the output size smaller. Thus, the name, deconvolution, is coming from when we want to have upsampling to get the output size larger. (But the name, deconvolution, is misinterpreted as reverse process of convolution, but it is not.) And it is also called, up convolution, and transposed convolution. And it is also called fractional stride convolution when fractional stride is used.

3. **Fusing the Output**

   After going through conv7 as below, the output size is small, then 32 upsampling is done to make the output have the same size of input image. But it also makes the output label map rough. And it is called FCN-32s:

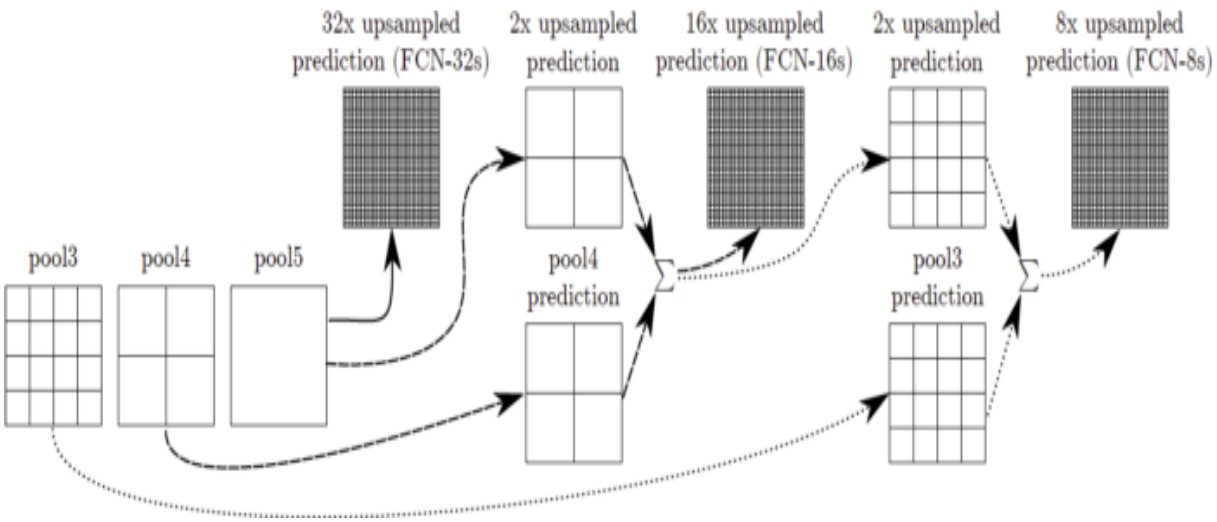


Figure 4.17: Fusing for FCN-16s and FCN-8s

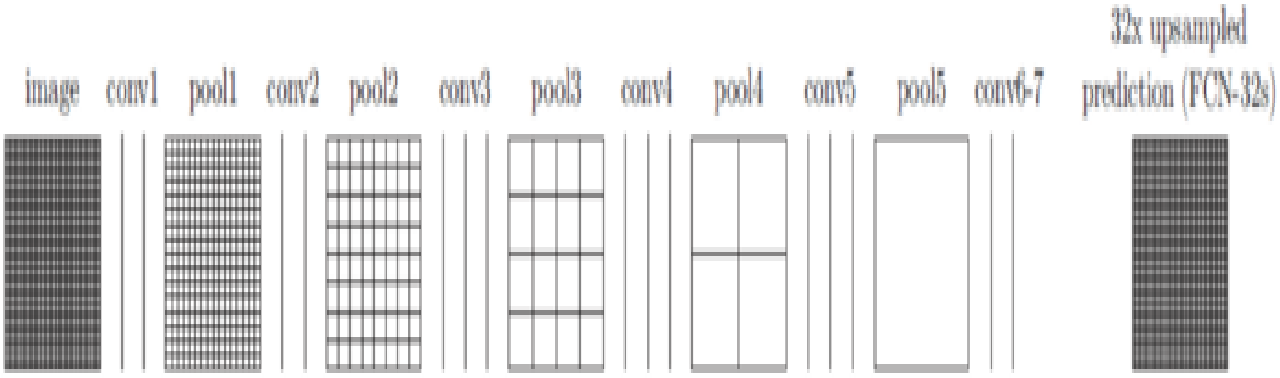


Figure 4.16: FCN-32s

This is because, deep features can be obtained when going deeper, spatial location information is also lost when going deeper. That means output from shallower layers have more location information. If we combine both, we can enhance the result.

To combine, we fuse the output (by element-wise addition): **FCN-16s:** The output from pool5 is 2 upsampled and fuse with pool4 and perform 16 upsampling. Similar operations for FCN-8s as in the figure above.

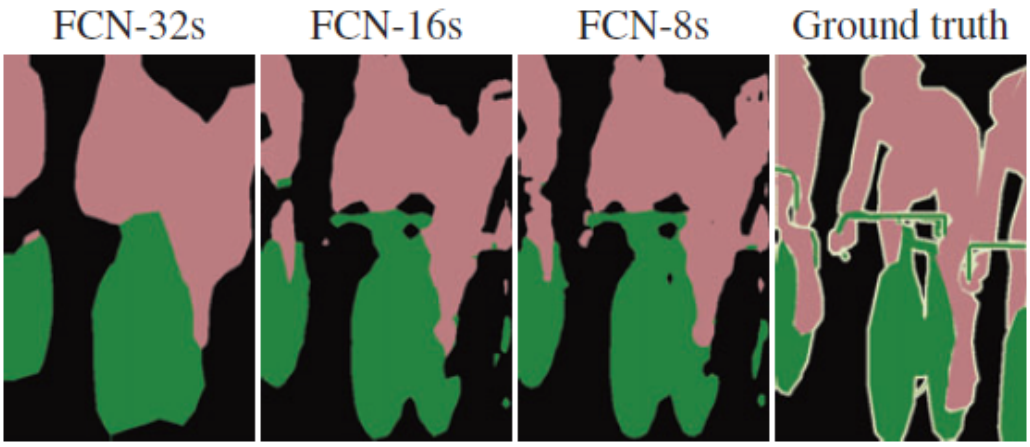


Figure 4.18: Comparison with different FCNs

FCN-32s result is very rough due to loss of location information while FCN-8s has the best result.

This fusing operation actually is just like the boosting / ensemble technique used in AlexNet, VGGNet, and GoogLeNet, where they add the results by multiple model to make the prediction more accurate. But in this case, it is done for each pixel, and they are added from the results of different layers within a model.

4. **Results**

	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	**90.3**	**75.9**	**62.7**	**83.2**

	pixel acc.	mean acc.	mean IU	f.w. IU
Gupta *et al.* [14]	60.3	-	28.6	47.0
FCN-32s RGB	60.0	42.2	29.2	43.9
FCN-32s RGBD	61.5	42.4	30.5	45.5
FCN-32s HHA	57.1	35.2	24.2	40.4
FCN-32s RGB-HHA	64.3	44.9	32.8	48.0
FCN-16s RGB-HHA	**65.4**	**46.1**	**34.0**	**49.5**

	pixel acc.	mean acc.	mean IU	f.w. IU	geom. acc.
Liu *et al.* [23]	76.7	-	-	-	-
Tighe *et al.* [33]	-	-	-	-	90.8
Tighe *et al.* [34] 1	75.6	41.1	-	-	-
Tighe *et al.* [34] 2	78.6	39.2	-	-	-
Farabet *et al.* [8] 1	72.3	50.8	-	-	-
Farabet *et al.* [8] 2	78.5	29.6	-	-	-
Pinheiro *et al.* [28]	77.7	29.8	-	-	-
FCN-16s	**85.2**	**51.7**	39.5	76.1	**94.3**

Figure 4.19: Pascal VOC 2011 dataset (Left), NYUDv2 Dataset (Middle), SIFT Flow Dataset (Right)

- FCN-8s is the best in Pascal VOC 2011.
- FCN-16s is the best in NYUDv2.
- FCN-16s is the best in SIFT Flow.

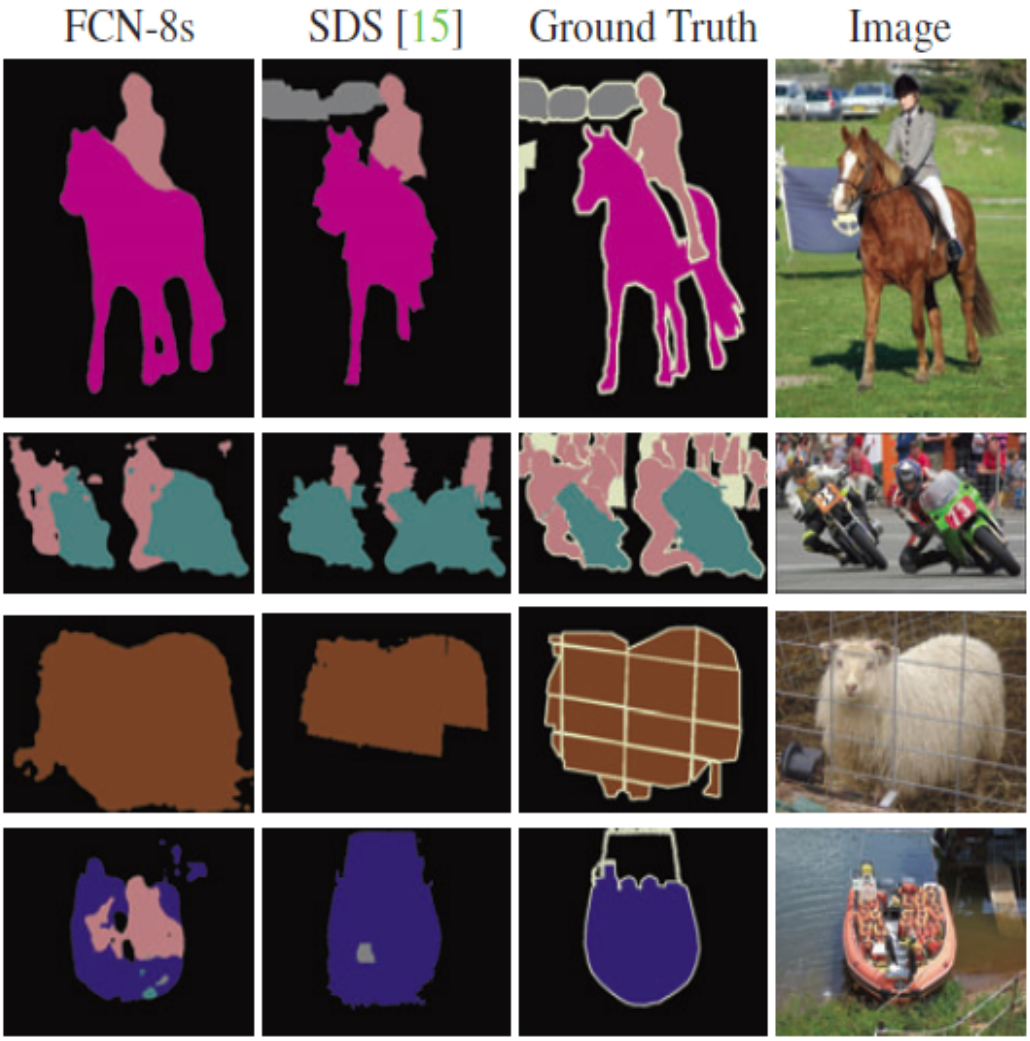


Figure 4.20: Visualized results.

4.3.2.7 U-Net

UNet, evolved from the traditional convolutional neural network, was first designed and applied in 2015 to process biomedical images. As a general convolutional neural network focuses its task on image classification, where input is an image and output is one label, but in biomedical cases, it requires us not only to distinguish whether there is a disease, but also to localise the area of abnormality.

UNet is dedicated to solving this problem. The reason it is able to localise and distinguish borders is by doing classification on every pixel, so the input and output share the same size.

The network has basic foundation looks like figure 4.21:

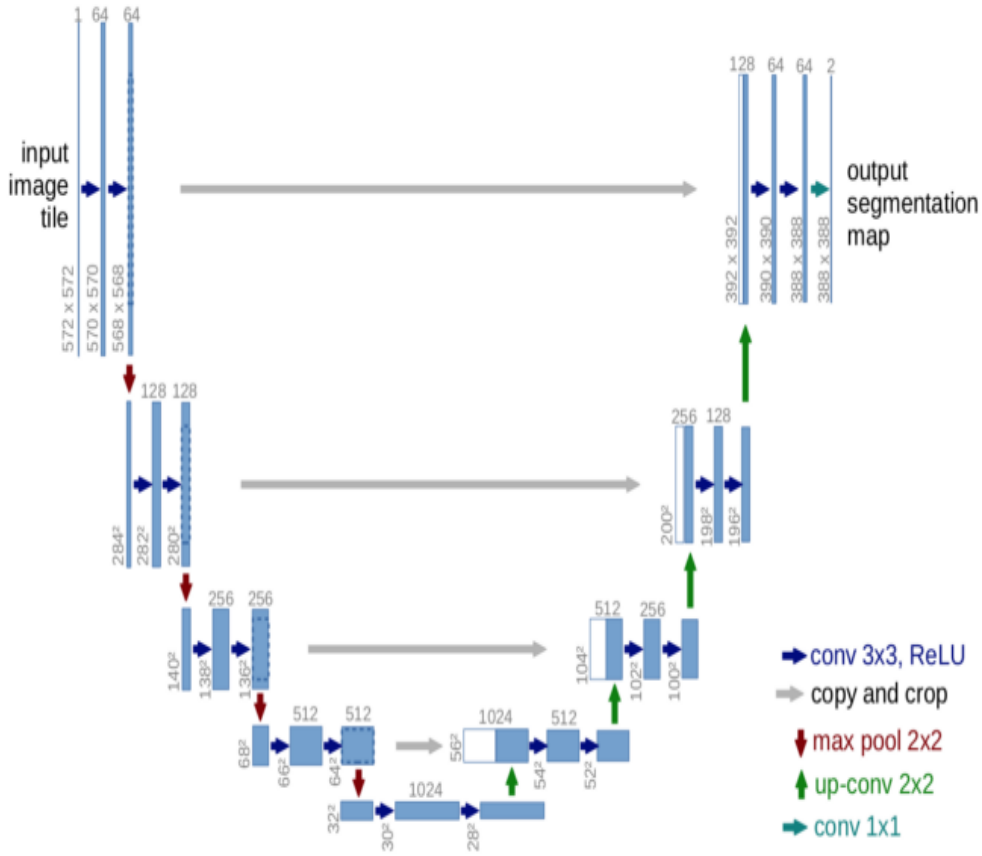


Figure 4.21: U-Net architecture

The architecture is symmetric and consists of two major parts — the left part is called contracting path, which is constituted by the general convolutional process; the right part is expansive path, which is constituted by transposed 2d convolutional layers.

We needed a data set for the U-NET approach, to train the model, so we had to manually annotate 2250 images for training and keeping 200 images for validation, here are some examples of the manually annotated data from both the state farm and AUC data set.

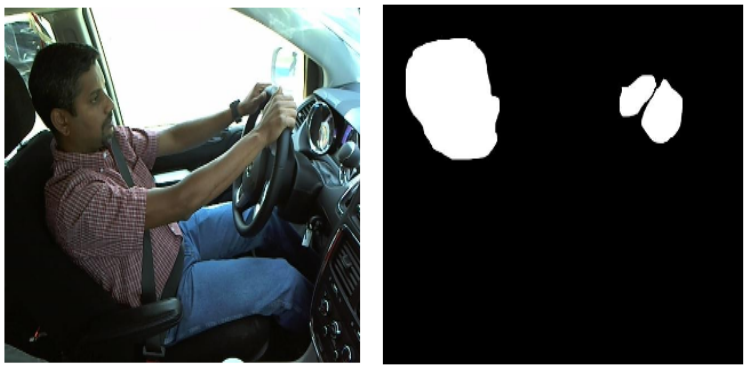


Figure 4.22: An example from the state farm data set

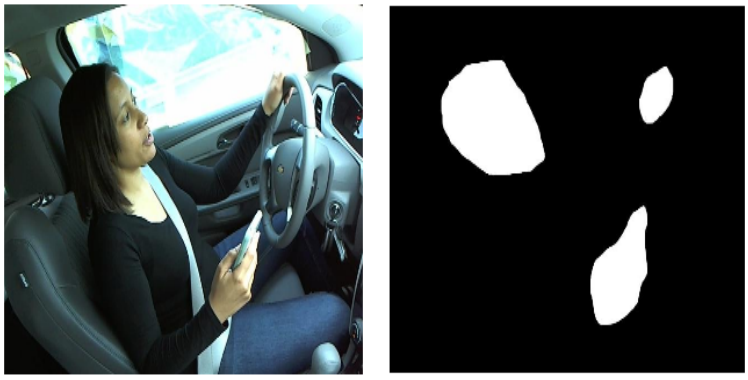


Figure 4.23: An example from the state farm data set

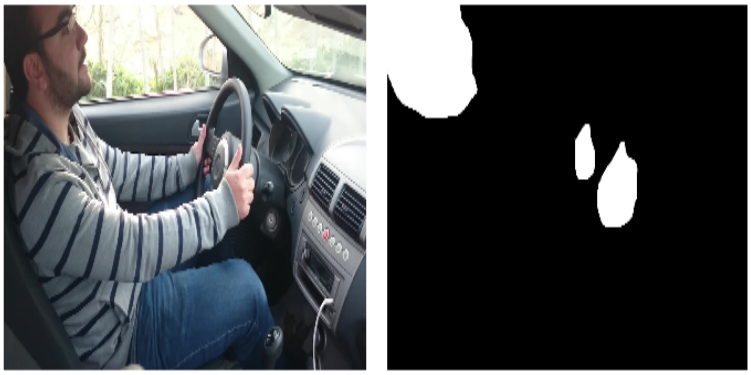


Figure 4.24: An example from the AUC data set

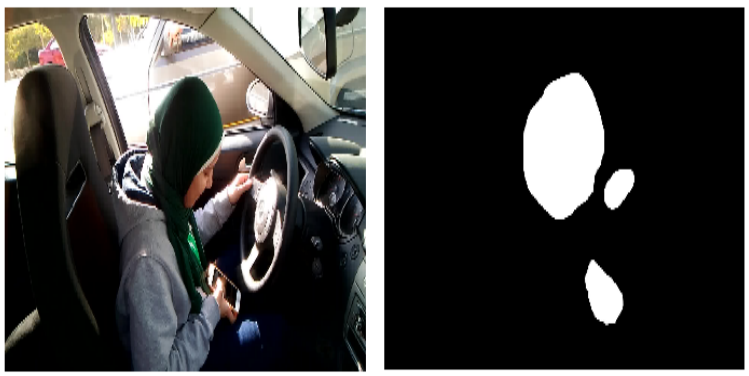


Figure 4.25: An example from the AUC data set

4.4 Results and Discussion

4.4.1 End-to-End Classification System

In this section, the results of each classification experiment are discussed. The first experiments are carried out using the State Farm dataset, while the second experiments are performed using both the AUC and State Farm datasets. Model optimization and hardware acceleration techniques are demonstrated. Moreover, saliency maps of the model predictions are illustrated in order to clarify the degree of generalization.

4.4.1.1 State Farm Dataset

The experiments based on the State Farm dataset are performed using both the custom CNN and pre-trained models discussed in section 4.3.1. Table 4.4 shows the training and validation accuracy and loss using the custom CNN, which are the best results achieved by training a CNN from scratch. It has been observed that transfer learning facilitates the training process and improves the ability of the CNNs to generalize on the problem of distraction detection. Table 4.5 demonstrates the validation accuracy and loss for different pre-trained CNNs after training on State Farm dataset, as well as the results of the ensemble between all the CNNs. ResNet50 outperforms other CNNs with a validation accuracy and loss of 88.4% and 0.38, respectively. The ensemble improves the validation accuracy and loss to 90.2% and 0.31, respectively, however, this approach leads to high memory usage, and it is more difficult to be deployed in a real-time environment.

Table 4.4: Training and validation accuracy and loss on State Farm dataset for the custom CNN

Metric	Train	Validation
Accuracy	86%	78%
Loss	0.4	0.8

Table 4.5: Validation accuracy and loss on State Farm dataset for different CNNs using Transfer Learning

Model	Validation Accuracy	Validation Loss
ResNet50	88.4%	0.38
VGG16	84.8%	0.48
InceptionV3	85%	0.5
Xception	85.8%	0.45
MobileNet	85.8%	0.54
DenseNet	85.8%	0.47
Ensemble of classifiers	90.2%	0.31

4.4.1.2 AUC and State Farm Datasets

After testing the ResNet50 model trained on the State Farm dataset on the AUC dataset, classification accuracy has been calculated to be 33%. This result illustrates the inability to generalize to new cases, as the AUC dataset includes different recording cameras, lighting conditions and drivers. Hence, the two datasets are combined to form a larger dataset, with more variance in lighting conditions and drivers than any of the two datasets separately. After training the ResNet50 model on the new formed dataset without augmentation, the classification accuracy and loss have been observed to be 90.4% and 0.34 respectively, which is close to the results obtained by the ensemble approach on the State Farm dataset only. After applying augmentation on the new dataset, the classification accuracy and loss have been improved to 92.5% and 0.26, respectively. By fine tuning the learning rate and dropout percentage, the classification accuracy and loss have been further improved to 94.3% and 0.25, respectively. Table 4.6 demonstrates the proposed real-time distraction detection system specifications. Figure 4.26 shows the normalized confusion matrix for the validation set. Most of the samples in the validation set are classified correctly, with some confusion between hair and makeup and drinking, and between talking to a passenger and safe driving classes. The model predicted the hair and makeup class samples accurately with an accuracy of 81%, with 9% misclassification from the drinking class. Regarding the talking to a passenger class, 78% of the samples are classified correctly, with 14% misclassification from the safe driving class.

Table 4.6: Proposed System Specifications

Model	ResNet50
Validation Accuracy	94.3%
Validation Loss	0.25%
Test Loss	0.247
Number of Parameters	23.5 M
Model Size	90 MB
Inference Time (GPU)	8 ms
Inference Time (CPU)	160 ms

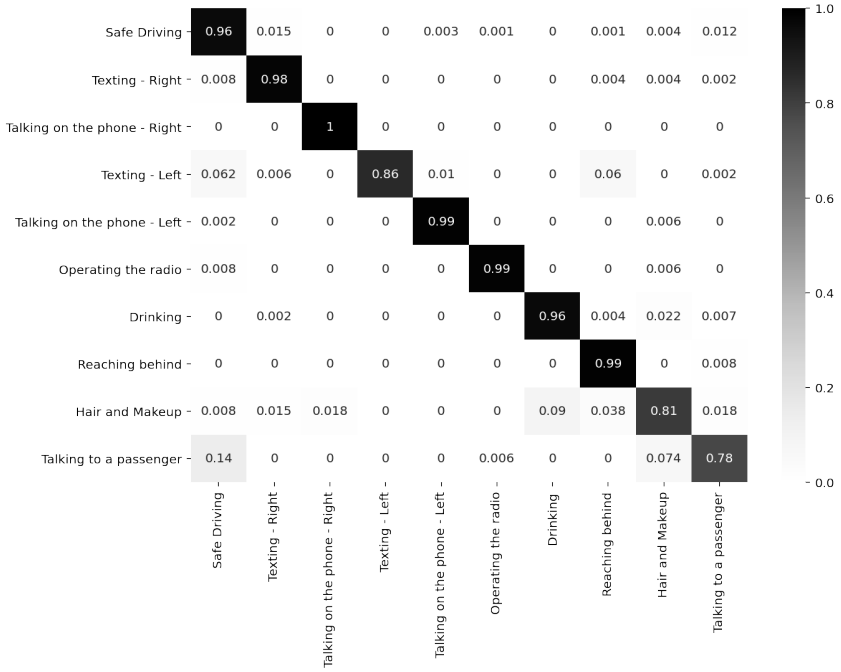


Figure 4.26: Normalized confusion matrix for the validation set

4.4.1.3 Model Optimization and Hardware Acceleration

Model optimization is an essential step in order to compress the trained model for an efficient deployment on hardware. The optimization process results in accelerating the forward propagation when inferring an input, which leads to less inference time, and more classified frames per second (FPS). Two optimization techniques are performed in this work, which are pruning and quantization.

The idea of pruning is to zero out the weights with the least contribution in the classification accuracy, keeping the reduction in the accuracy as low as possible. Weight pruning is performed using different criteria, such as L1 structured, L2 structured, L1 unstructured and L2 unstructured. L1 unstructured criterion is used in this work. The L1 unstructured criterion zeroes out the weights with the least L1 norm in different layers of the network. The L1 unstructured criterion is performed on every convolutional and linear layer in the CNN, with varying the amount of weights to be pruned in every layer. Figure 4.27 and 4.28 show the effect of changing the amount of pruning of the convolutional layers weights on the validation accuracy and loss, respectively. Figure 4.29 and 4.30 show the effect of changing the amount of pruning of the linear layers weights on the validation accuracy and loss, respectively. Figure 4.31 and 4.32 show the effect of changing the amount of pruning of the linear layers weights on the validation accuracy and loss, respectively, after pruning 30% of the convolutional layers weights. Table 4.7 demonstrates the effect of pruning on the validation accuracy, validation loss and number of model parameters.

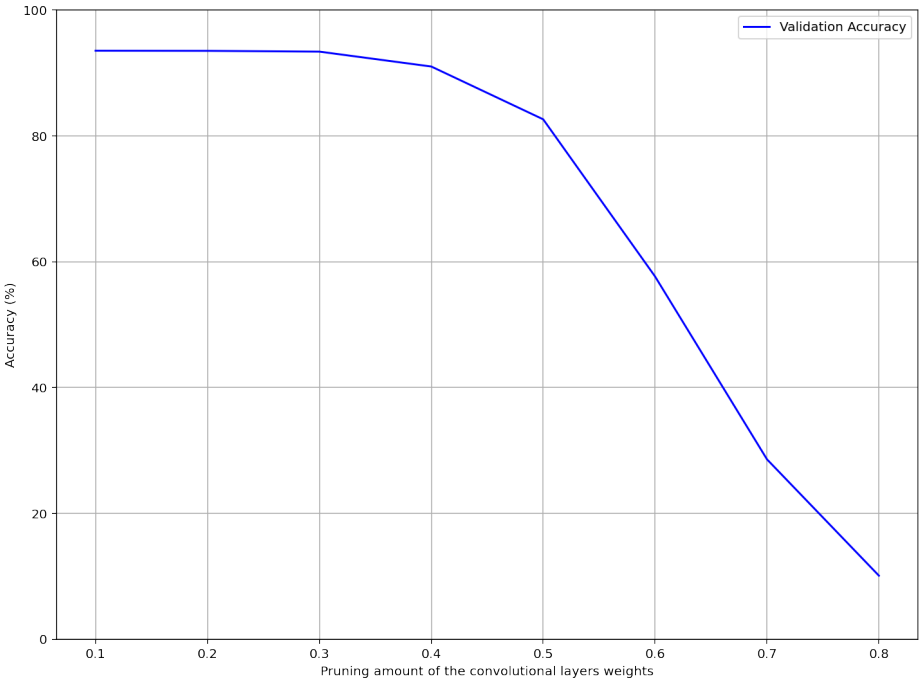


Figure 4.27: Effect of increasing pruning amount of the convolutional layers weights on the accuracy

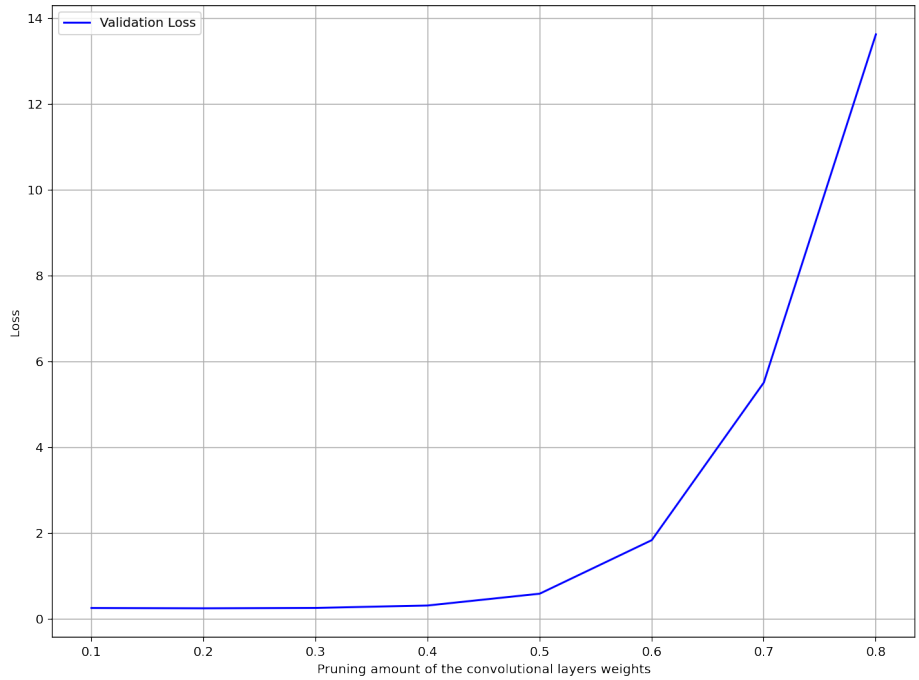


Figure 4.28: Effect of increasing pruning amount of the convolutional layers weights on the loss

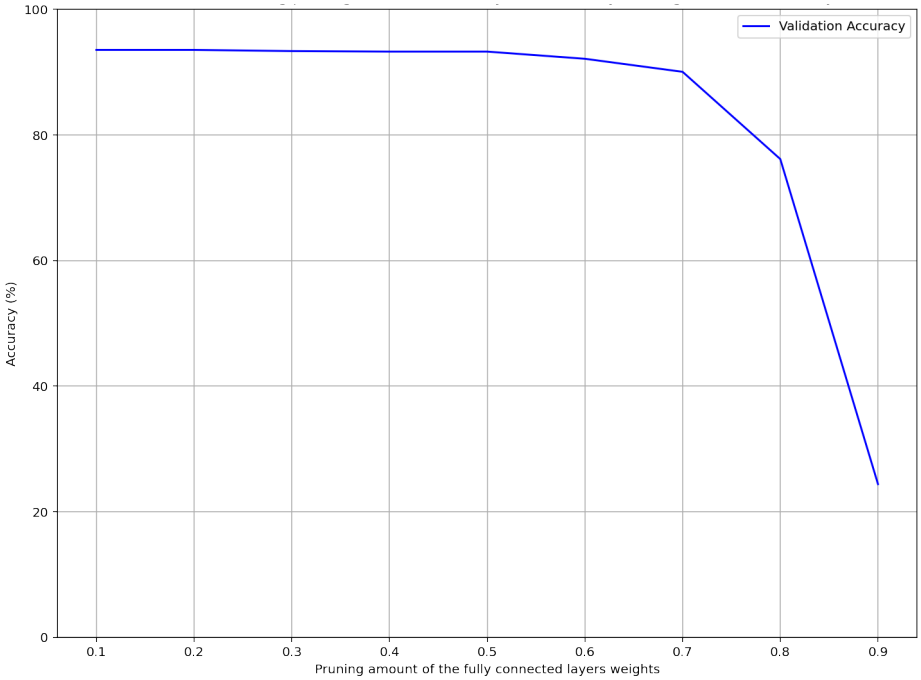


Figure 4.29: Effect of increasing pruning amount of the linear layers weights on the accuracy

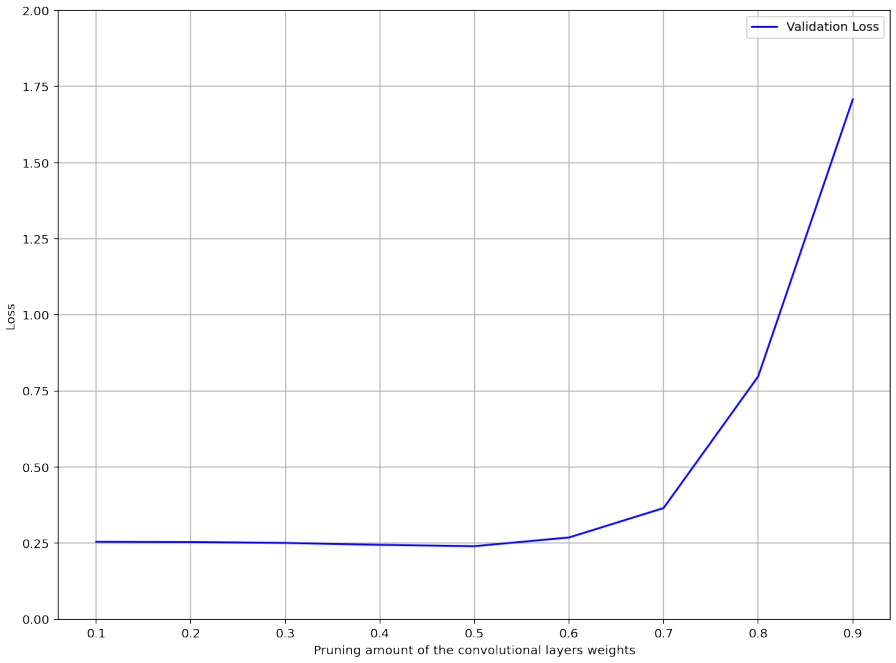


Figure 4.30: Effect of increasing pruning amount of the linear layers weights on the loss

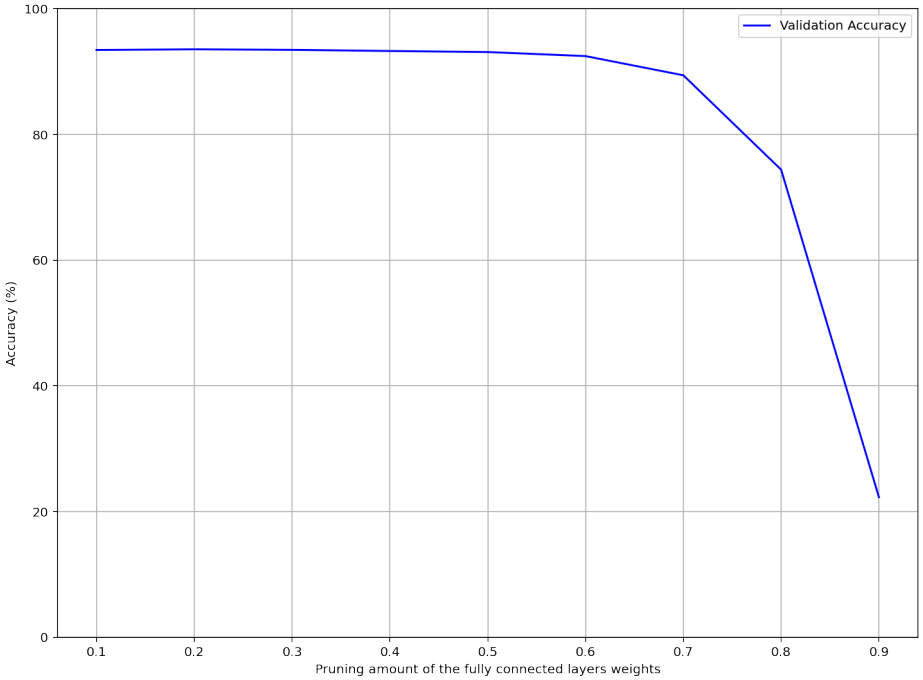


Figure 4.31: Effect of increasing pruning amount of the linear layers weights on the accuracy after pruning the weights of the convolutional layers

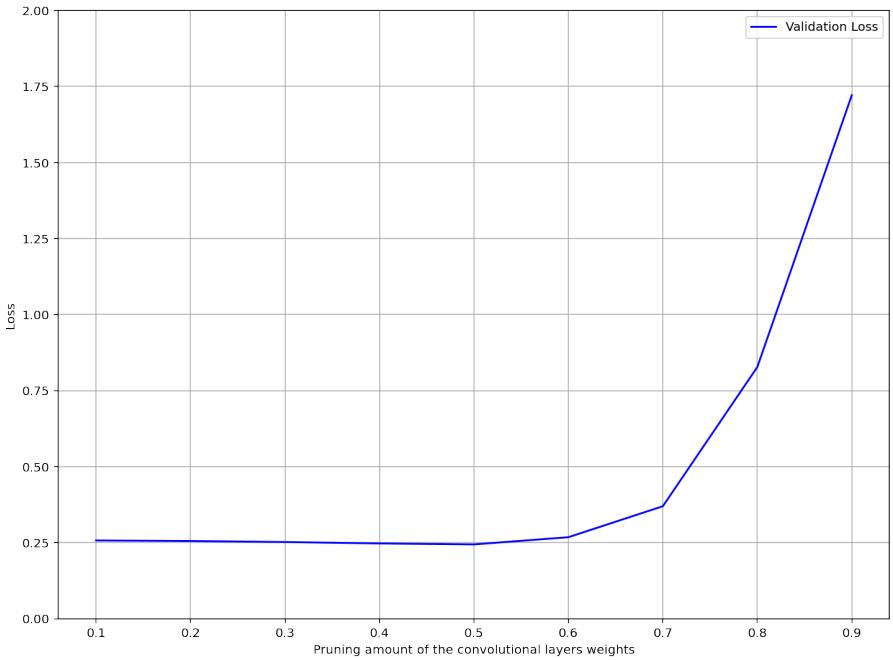


Figure 4.32: Effect of increasing pruning amount of the linear layers weights on the loss after pruning the weights of the convolutional layers

Table 4.7: Validation accuracy, validation loss and number of model parameters before and after pruning

Metric	Before Pruning	After Pruning
Validation Accuracy	93.5%	93.25%
Validation Loss	0.25%	0.247
Number of Parameters	26M	18M (Non zero)

Quantization is the process of converting the representation of the model parameters from float32 to int8, thus reducing the model size and the inference time, which in turn accelerates the model performance on hardware. Quantization aware training is used in this work, in which the model parameters are quantized based on 2 introduced new parameters which are the scale and zero point. Equation 4.1 demonstrates how each model parameter is quantized. Each parameter is divided by the scale value, then the result is added to the zero point value. After scaling the model parameter and adding the zero point value, the result is rounded to the nearest integer. The two introduced paramters (scale and zero point) are computed using an optimization technique which minimizes the error between the actual label and the prediction of the quantized model, thus the ResNet50 architecture has been trained on a subset of the dataset in order to find the optimum scale and zero point parameters. Table 4.8 shows the effect of quantization on the validation accuracy, validation loss, test loss, model size and inference time on CPU.

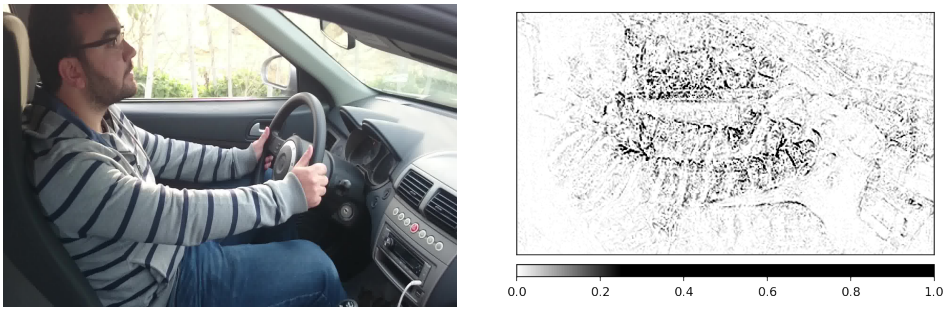
$$Q(x, scale, zero\_point) = round(\frac{x}{scale} + zero\_point) \tag{4.1}$$

Table 4.8: Validation accuracy, validation loss and number of model parameters before and after Quantization

Metric	Before Quantization	After Quantization
Validation Accuracy	94.3%	92.2%
Validation Loss	0.25%	0.262
Test Loss	0.247	0.28
Model Size	90 MB	24 MB
Inference Time (CPU)	160 ms	75 ms

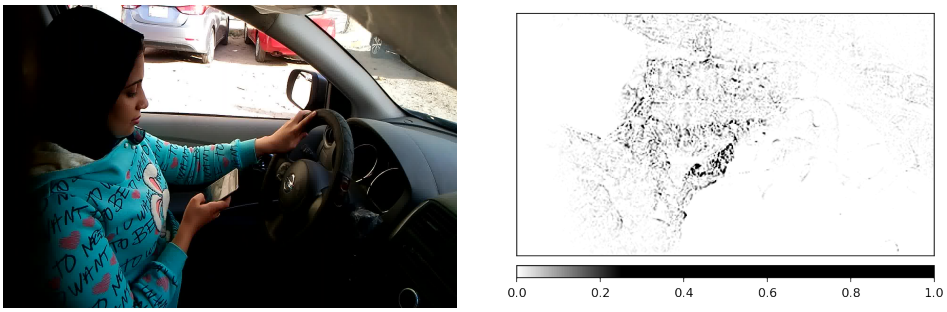
4.4.1.4 Model Interpretability

Interpretability and visualization play a key role in evaluating the Neural Networks performance. In order to assess model generalization, it is crucial to identify the most relevant parts of the image that the trained model considers when predicting a certain class for an input image. The following figures illustrate the saliency map based on the ResNet50 model prediction for each class in both the AUC and State Farm datsets.



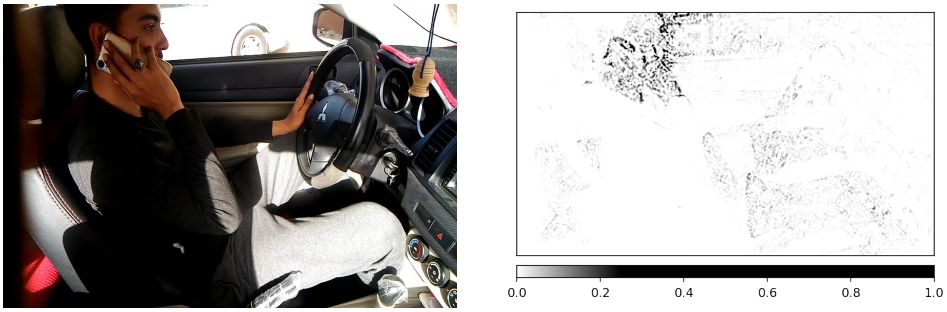
(a) (b)

Figure 4.33: Visualization of the important parts of the image that contribute in the prediction. (a) Original safe driving sample from the AUC dataset (b) Saliency map



(a) (b)

Figure 4.34: Visualization of the important parts of the image that contribute in the prediction. (a) Original texting - right sample from the AUC dataset (b) Saliency map

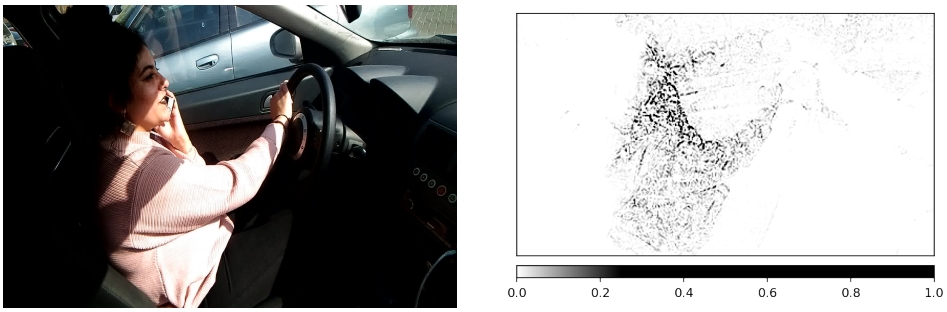


(a) (b)

Figure 4.35: Visualization of the important parts of the image that contribute in the prediction. (a) Original talking on the phone - right sample from the AUC dataset (b) Saliency map

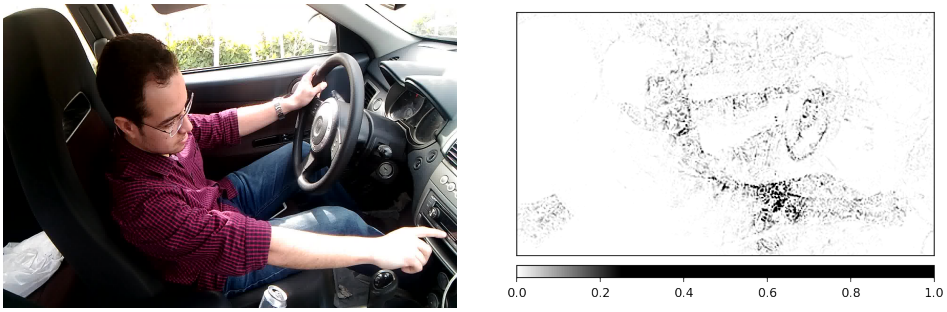
(a) (b)

Figure 4.36: Visualization of the important parts of the image that contribute in the prediction. (a) Original texting - left sample from the AUC dataset (b) Saliency map



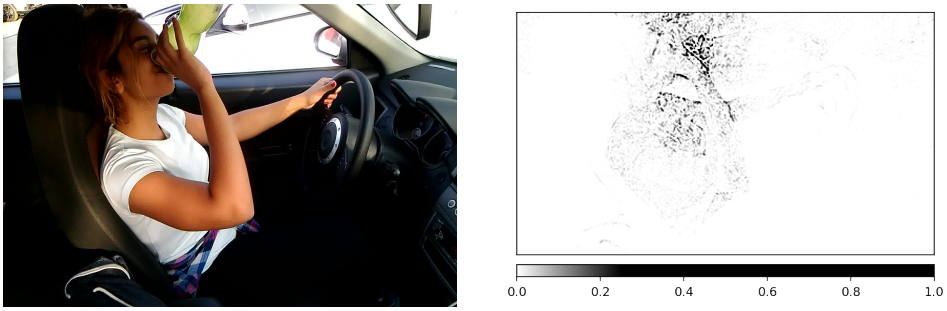
(a) (b)

Figure 4.37: Visualization of the important parts of the image that contribute in the prediction. (a) Original talking on the phone - left sample from the AUC dataset (b) Saliency map



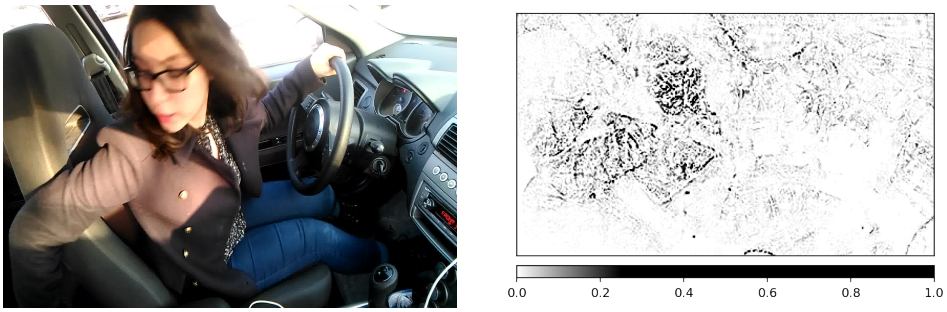
(a) (b)

Figure 4.38: Visualization of the important parts of the image that contribute in the prediction. (a) Original operating the radio sample from the AUC dataset (b) Saliency map



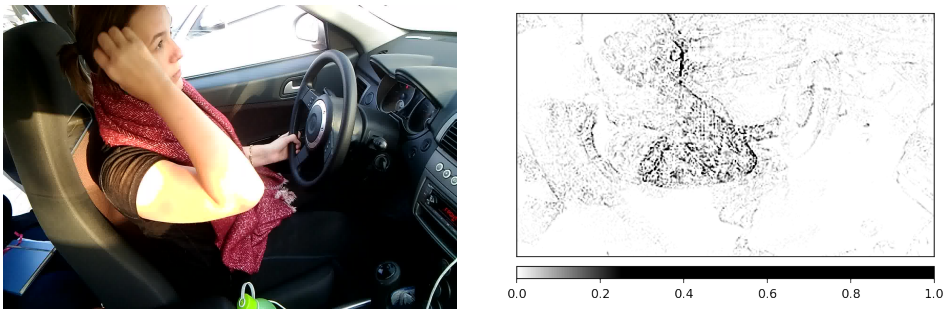
(a) (b)

Figure 4.39: Visualization of the important parts of the image that contribute in the prediction. (a) Original drinking sample from the AUC dataset (b) Saliency map



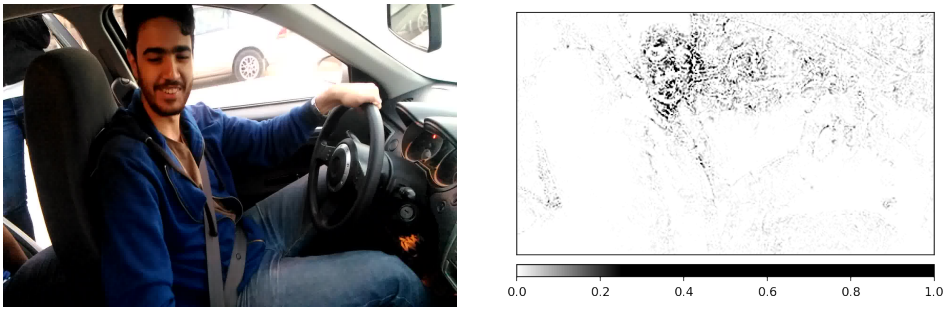
(a) (b)

Figure 4.40: Visualization of the important parts of the image that contribute in the prediction. (a) Original reaching behind sample from the AUC dataset (b) Saliency map



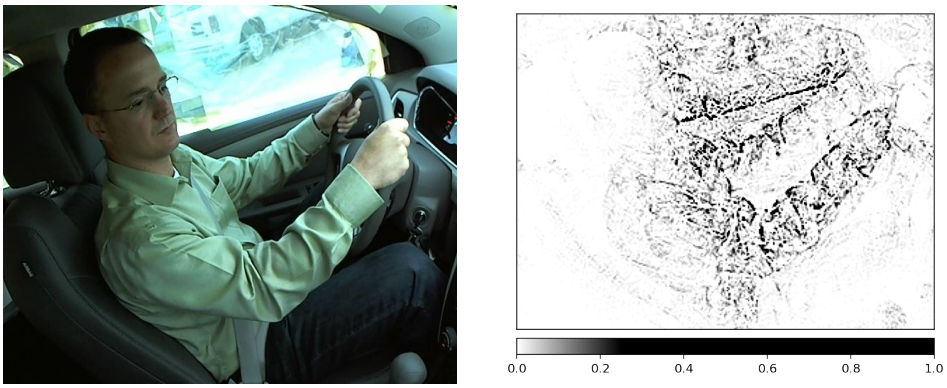
(a) (b)

Figure 4.41: Visualization of the important parts of the image that contribute in the prediction. (a) Original hair and makeup sample from the AUC dataset (b) Saliency map



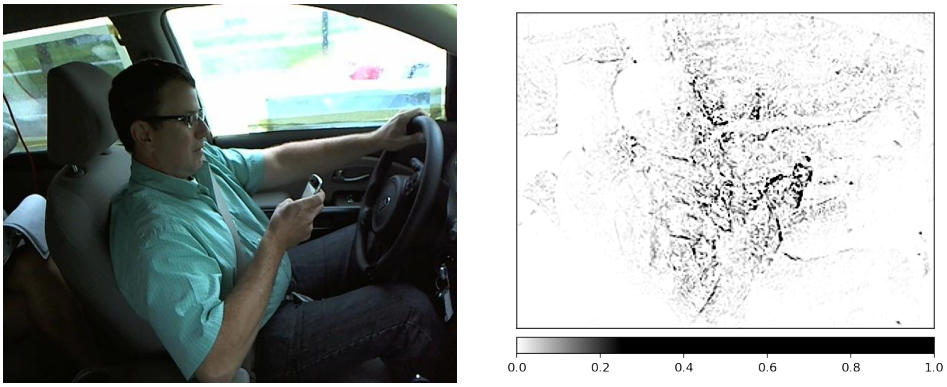
(a) (b)

Figure 4.42: Visualization of the important parts of the image that contribute in the prediction. (a) Original talking to a passenger sample from the AUC dataset (b) Saliency map



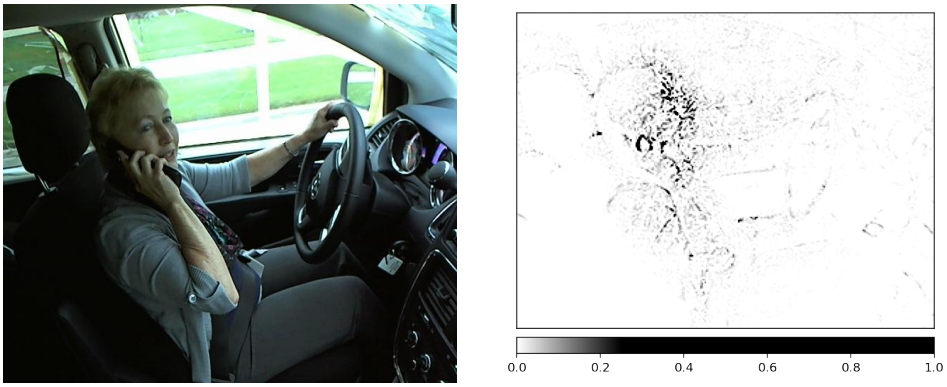
(a) (b)

Figure 4.43: Visualization of the important parts of the image that contribute in the prediction. (a) Original safe driving sample from State Farm dataset (b) Saliency map



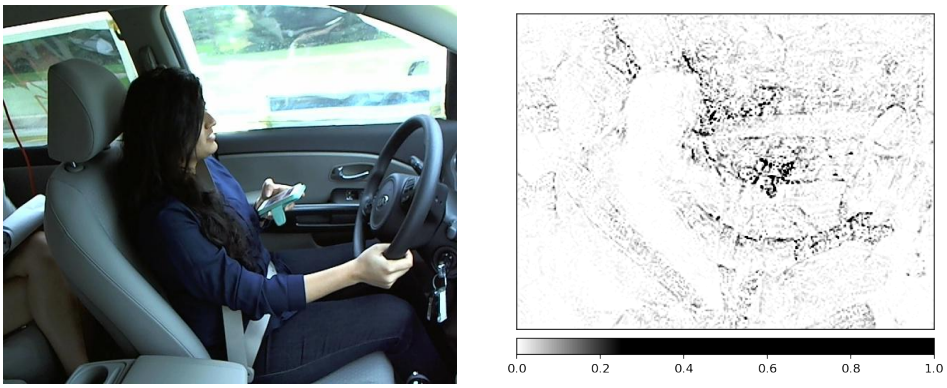
(a) (b)

Figure 4.44: Visualization of the important parts of the image that contribute in the prediction. (a) Original texting - right sample from State Farm dataset (b) Saliency map



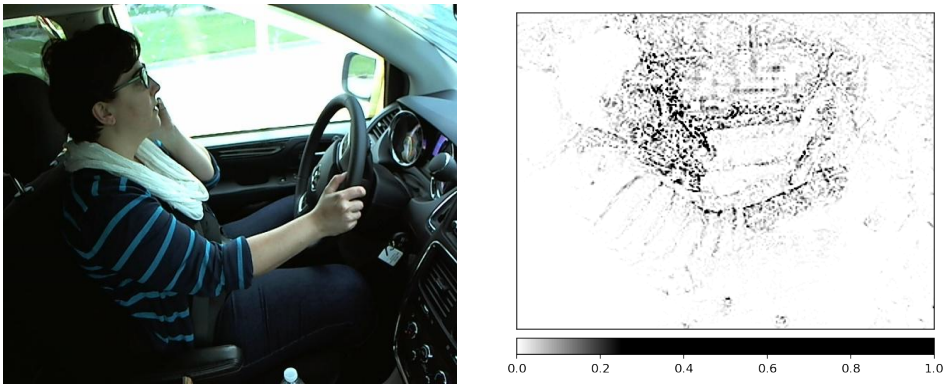
(a) (b)

Figure 4.45: Visualization of the important parts of the image that contribute in the prediction. (a) Original talking on the phone - right sample from State Farm dataset (b) Saliency map



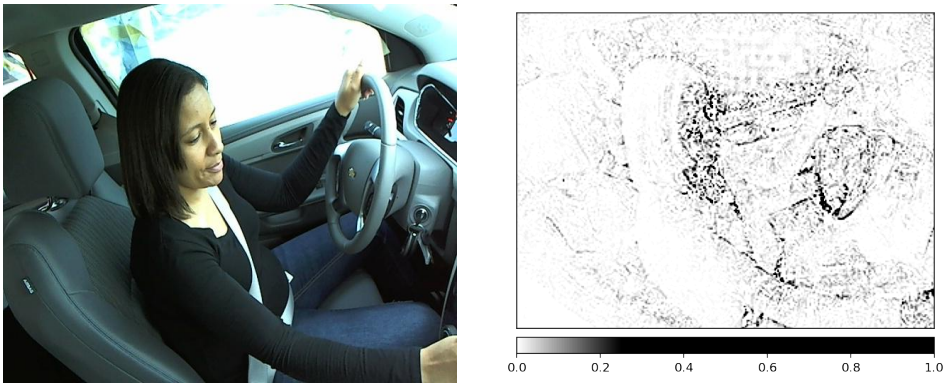
(a) (b)

Figure 4.46: Visualization of the important parts of the image that contribute in the prediction. (a) Original texting - left sample from State Farm dataset (b) Saliency map



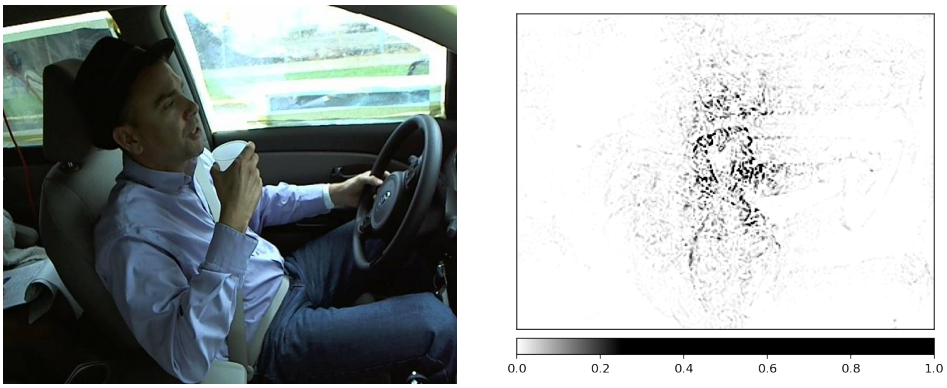
(a) (b)

Figure 4.47: Visualization of the important parts of the image that contribute in the prediction. (a) Original talking on the phone - left sample from State Farm dataset (b) Saliency map



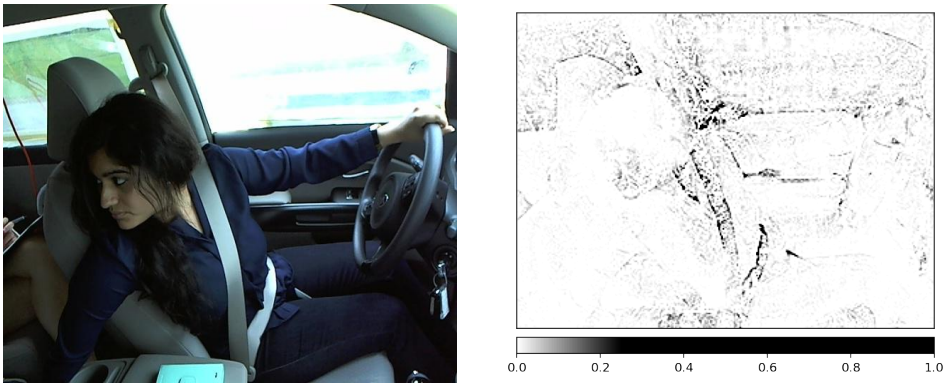
(a) (b)

Figure 4.48: Visualization of the important parts of the image that contribute in the prediction. (a) Original operating the radio sample from State Farm dataset (b) Saliency map



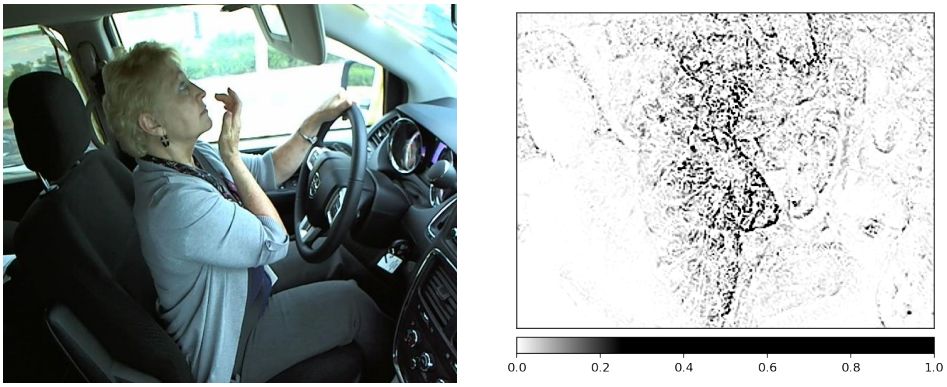
(a) (b)

Figure 4.49: Visualization of the important parts of the image that contribute in the prediction. (a) Original drinking sample from State Farm dataset (b) Saliency map



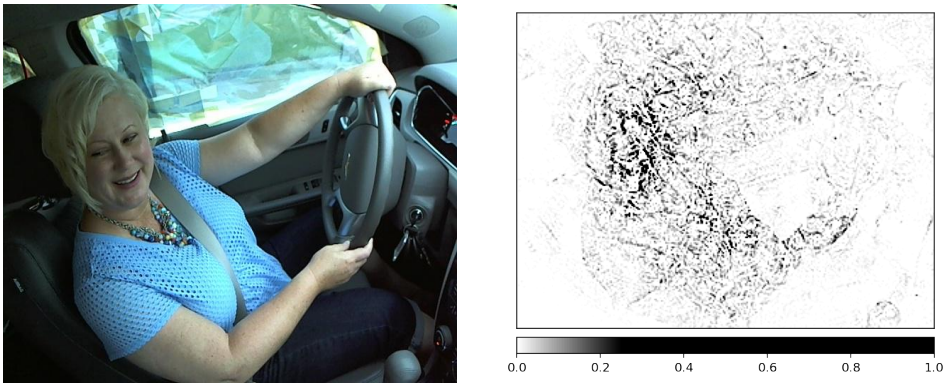
(a) (b)

Figure 4.50: Visualization of the important parts of the image that contribute in the prediction. (a) Original reaching behind sample from State Farm dataset (b) Saliency map



(a) (b)

Figure 4.51: Visualization of the important parts of the image that contribute in the prediction. (a) Original hair and makeup sample from State Farm dataset (b) Saliency map



(a) (b)

Figure 4.52: Visualization of the important parts of the image that contribute in the prediction. (a) Original talking to a passenger sample from State Farm dataset (b) Saliency map

4.4.1.5 Semantic Segmentation

- Mask R-CNN results:

  First model we tried was Mask R-CNN, it was fed the input image, and outputted the segmentation mask as following:

Figure 4.53: Driver and background segmentation masks

Figure 4.54: Original image after applying the segmentation mask

  There was a critical problem with the mask rcnn as it's inference time was about **15-20 seconds**, and that was not suitable for real-time systems.

- U-NET:

  For the U-NET model, we used it to segment the face and hands of the driver, it was implemented using Keras, and we got the following results shown in table 4.9:

Table 4.9: U-NET results

Data/Metric	Loss	Mean IOU
Training	0.0409	0.9045
Validation	0.0667	0.9044

After testing the U-NET and evaluating it, we had the segmented output images as below:

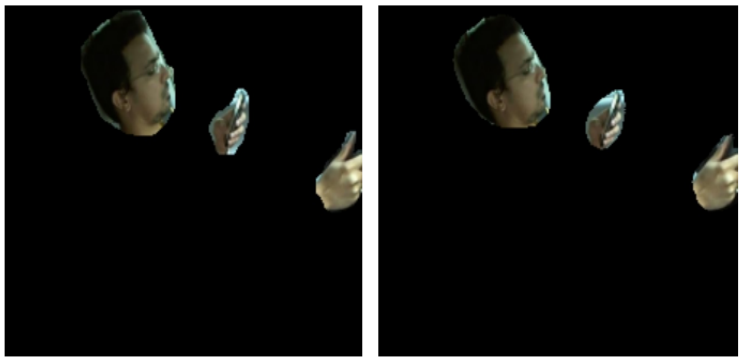


Figure 4.55: On the left is the ground truth segmented image, on the right is the prediction of the U-NET model

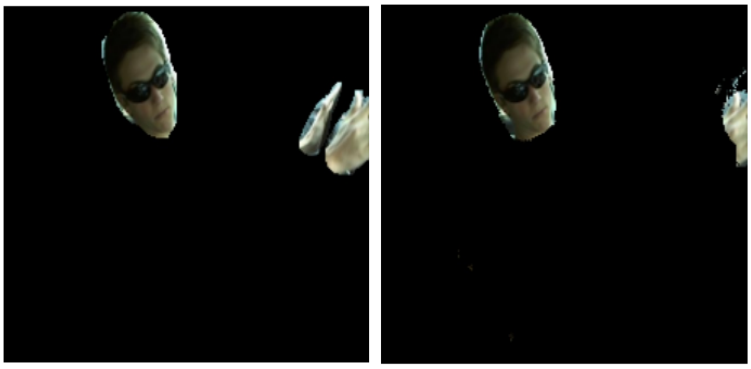


Figure 4.56: On the left is the ground truth segmented image, on the right is the prediction of the U-NET model

Figure 4.57: U-NET predictions on the AUC dataset

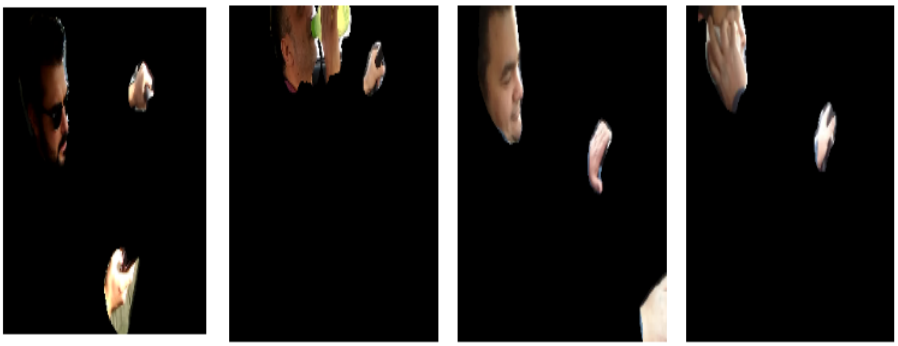


Figure 4.58: U-NET predictions on the AUC dataset

After training the U-NET model, we used it to segment the whole data set, but unfortunately after implementing the segmentation model along with the classification model, the classification results were not satisfying in comparison with the E2E ResNet50 model.

The reason for that would be that the manually annotated segmentation data set was not enough. The only solution for that problem is increasing the data set and the annotation of more data, so we can get better results from the 2-block model.

The good thing about that model is its inference time, which has decreased from 15-20 seconds as in the mask R-CNN to **2-5 ms**.

Chapter 5

Speech Recognition

There is a rising demand for better speech recognition systems proposed on the AI market that authorize contact-less control to several devices and equipment. The speech recognition system enables the user to do the task with minimum effort or to focus on other advanced tasks. The speech recognition system that is proposed in this chapter can be integrated in systems that require limited number of tasks like tuning devices and equipment while driving the car. This system enables the driver to be more focused on the driving task and not get distracted by these other tasks.

5.1 Literature Review

This section presents the related work in the field of Speech recognition. This paper [34] introduced RNN model with a classification accuracy on test set (about 7K samples) 96.5%. The first place at this competition [6] on Kaggle achieved 91% on the submission set (about 150K samples).

5.2 Simulation Setup

5.2.1 Dataset

The dataset is a mandatory block of a successful deep learning system. This section will introduce speech datasets in two different languages. The speech recognition system is trained on the English dataset although the Arabic dataset is mentioned because it has been collected by our team. The Arabic dataset is small that prevents our team to build a reliable Arabic speech recognition system however it is an open-source dataset to increase it and build a reliable system soon.

5.2.1.1 English dataset

This dataset is publicly available from Tensorflow team on kaggle [6] as a challenge to build an algorithm that understands simple speech commands in English. The predicted 12 labels are "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go" and everything else should be considered either "unknown" or "silence". These are the core classes that the challenge compute the accuracy score on them however there are additional 20 classes that are available to use which are "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "bed", "bird", "cat", "dog", "happy", "house", "Marvin", "sheila", "tree", and "wow". This dataset does not cover the modeling of the "unknown" or the "noise" classes. In section Dataset distribution and Modeling of the missing classes the creation of these two classes are covered.

5.2.1.2 Arabic dataset

Arabic speech dataset is rare to be found as a reliable open source, from this point a website [35] for collecting an open source Arabic speech dataset is deployed to have a reliable dataset structure as an Arabic version from the dataset created by Tensorflow team on kaggle [6]. Building the website will be discussed briefly below. The targeted classes are shown in figure 5.1. Although the number of users who recorded these words is still small, it is a serious initiative to construct a reliable speech Arabic dataset for the Arabic AI community.

'صفر', 'واحد', 'اثنان', 'ثلاثة', 'اربعة', 'خمسة', 'ستة',
'سبعة', 'ثمانية', 'تسعة', 'تشغيل', 'ايقاف', 'قف', 'انطلق',
'فوق', 'تحت', 'يسار', 'يمين', 'نعم', 'لا', 'تكييف', 'راديو',
'نوافذ'

Figure 5.1: Targeted Arabic labels

Due to the importance and scarce of the Arabic data, a web application was built to simply collect Arabic data with 27 classes as shown in figure 5.1. The website is composed of two parts: front-end and back-end. Front-end is the part that the client sees and interacts with, it concerns on designing, templating and formatting the text using HTML, CSS and JavaScript. Back-end is the part that concerns on the connection to database and servers using Flask and pythonanywhere. The following figure 5.2 illustrates this process.

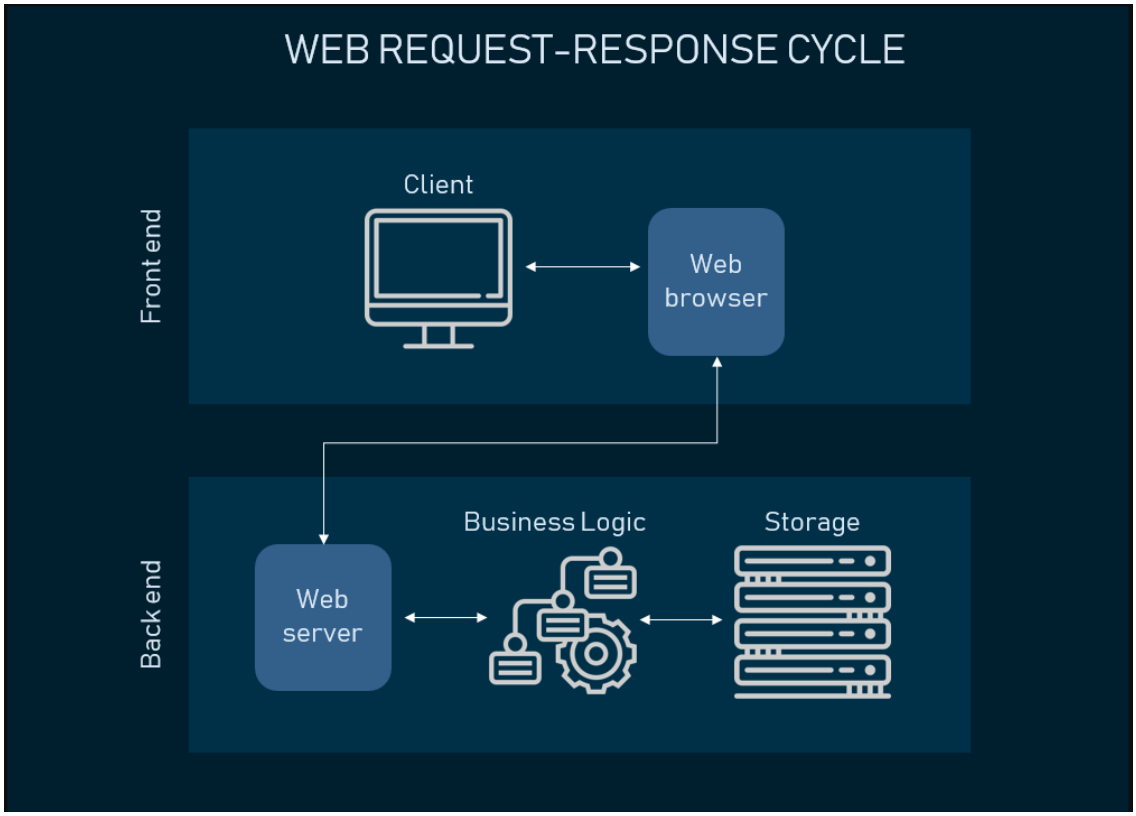


Figure 5.2: Web request-response cycle

Flask is a micro web framework that facilitates building the web application, the routing happened between pages and creation of session between the client and server. Pythonanywhere' is a web hosting service based on Python language, on which the website deployed and took a domain name. It is our database to store the recorded audios of 512 MB free storage, knowing that the size of one audio is 31 KB, and by using the console of the web hosting service 'Pythonanywhere', which supports python language, the recorded audios are sent to google drive.

To record audio stream, an API called 'Mediastream recording' is used, because it provides an easy way to record from the user's input devices and instantly save the captured audio in a blob that contains the recorded audio chunks and format type ogg format. Then, the audio is converted to a 16-bit little-endian PCM-encoded wav file at a 16000 sample rate to be in familiar format and ready for use. After that, AJAX technique is used to send this information in JSON format to the directory created in pythonanywhere, and the name of the saved file is the id of the session that Flask created for the client.

The figures below shows the website when a client visits it:

The figure 5.3 is welcome page that contains a brief about the website and some instructions related to recording, and the second one 5.4 is a tutorial page that illustrates what the user should do to facilities the process.

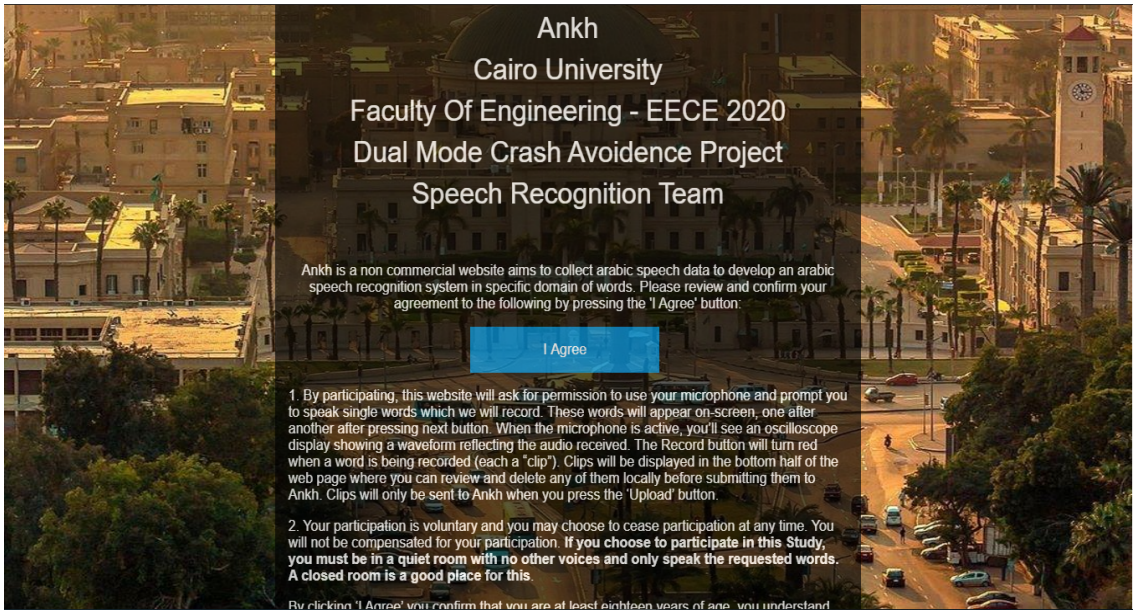


Figure 5.3: Welcome page of the website

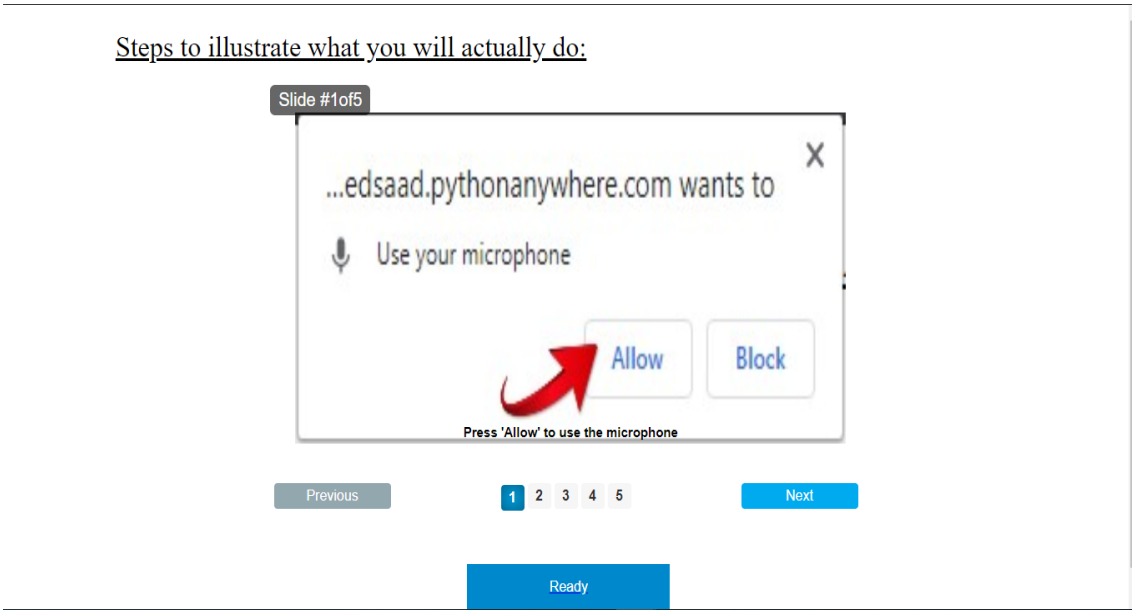


Figure 5.4: Tutorial page of the website

After that, recording page will appear but press allow to use the microphone. The labels are written in the black box, and the recording starts when the blue button is pressed, the recording time is about 1 sec and stops automatically, as shown in the figures 5.5, 5.6. The user can listen to the recorded audio and download it if needed. After finishing all the words, click upload button to send all the recorded audios to the database.

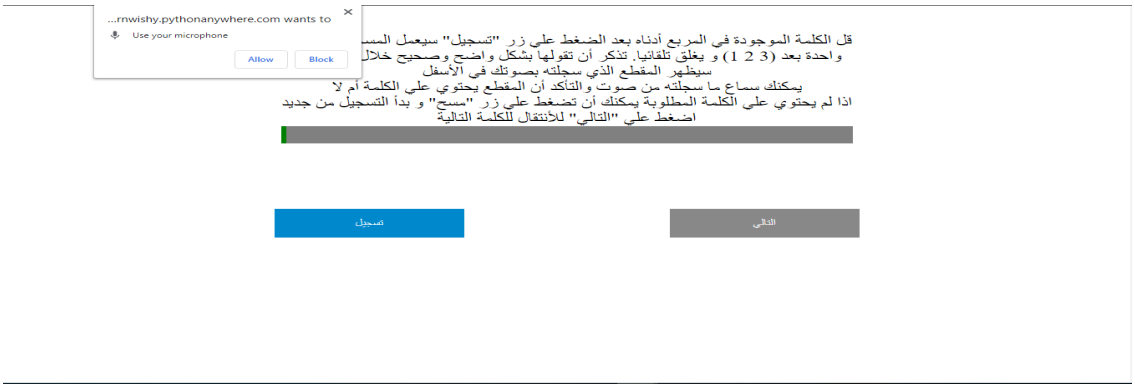


Figure 5.5: A request to allow the microphone

Figure 5.6: Recording page of the website

5.2.2 Preprocessing

5.2.2.1 Dataset distribution and Modeling of the missing classes

Modeling of classes "unknown" and "silence" was a mandatory task for this competition to achieve a high score. The dataset contains a folder for different kinds of background noises. Background clips had been cut into 1-sec slices and mixed for the "silence" class modeling. The "unknown" class is conducted from two resources. The first one is from the other unused classes that exist in the dataset like "marvin", "bird", etc. The second resource is from random people talking video on youtube[36]. The whole dataset is split into training, validation, testing, and unlabeled submission set. The submission set is the dataset that is submitted to kaggle to calculate the model score on it and it is about 152000 samples.

The training, validation, and testing datasets contain unique speakers that the speaker who contributes to the training set doesn't contribute to the testing set nor the validation set and vice versa. This guarantees well-generalized models that don't bias to the speaker's identity. The dataset distribution is shown in table 5.1.

Table 5.1: Dataset distribution

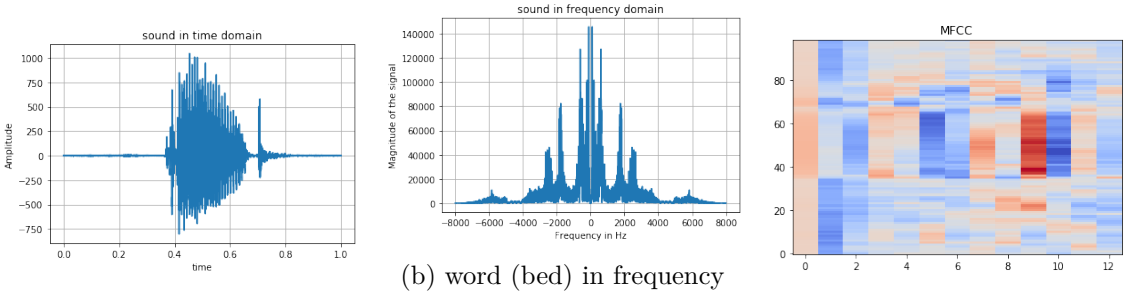
Class	Train	Validation	Test
Down	1842	264	253
Go	1861	260	251
Left	1839	247	267
No	1853	270	252
Off	1839	256	262
On	1864	257	246
Right	1852	256	259
Silence	1600	199	201
Stop	1885	246	249
Unknown	34154	4421	4470
UP	1843	260	272
Yes	1860	261	256
Total	54292	7197	7238

5.2.2.2 Features level

In order to teach our model and make it with high efficiency there are several ways for that the most important is to have sufficient data as they say in the deep learning world more data is better but the presence of a large amount of data does not guarantee getting the best results for teaching the model from here the explanation of Some of Features levels that have been done for our data to get the most benefit.

some of these features is to ensure that all of our data are well organized and unique and the duration of each sound segment is equal and has same rate to ensure access to the same features when processing on sound clips, Example of these features is MFCC as it is considered one of the most important features, and to make the data be equal in duration the sound clips was padded to ensure equal duration

And the most important of these features as mentioned in the previous paragraph is the use of MFCC Let's first see what is MFCC means MFCC stands for Mel frequency cepstral coefficients. As you can see there are 4 words in the abbreviation. Mel, frequency, cepstral and coefficients. The idea of MFCC is to convert audio in time domain 5.7a into frequency domain 5.7b so that it can understand all the information present in speech signals. But just converting time domain signals into frequency domain may not be very optimal. so do more than just converting time-domain signals into frequency domain signals. Our ear has cochlea which basically has more filters at low frequency and very few filters at higher frequency. This can be mimicked using Mel filters. So, the idea of MFCC is to convert time domain signals into frequency domain signal by mimicking cochlea function using Mel filters 5.7c, this is to extract the best features found in the sound by using suitable number of filters.



(a) word (bed) in time domain

(b) word (bed) in frequency domain

(c) word (bed) in MFCC

Figure 5.7: word (bed) in time domain, frequency domain and MFCC

5.2.2.3 Classical augmentation

In order to increase the accuracy and reduce the loss and the presence of a limited number of data,the augmentation is used to make the model learn more from the data by seeing many changes that occur in the sound in the daily life such as low volume and loudness and speed of speech and slow And for the possibility of losing part of the word, using the augmentation, the model will be able to learn more and see many possible changes that can happen.

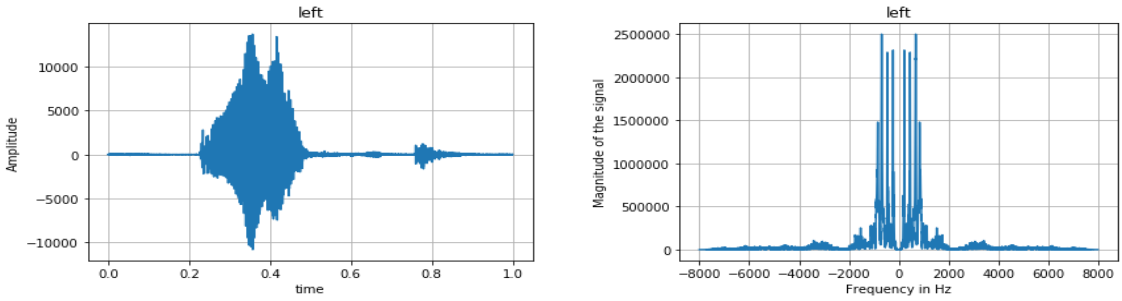
Data augmentation is a technique to artificially get more training data from the existing training data. It is a type of data pre-processing that makes different transformations to the training data, and gives the training data the property of diversity. This technique makes the classifier learns features that are irrelevant to the time, frequency or power (signal strength). The basic methods of data augmentation for audio is implemented from scratch using scipy and numpy packages instead of librosa library that supports many methods of audio augmentation.

many functions are used to increase data and let's talk about each of them in this section, which are:

1. Changing speed
2. Volume control
3. Mask
4. Time shifting
5. Cropping audio
6. Adding random noise
7. Fourier transform
8. Pitch change

The disadvantage of Librosa augmentation is that it takes long time to do one epoch in the training phase of about 6 Hrs on google colab virtual machine. This is because audio files are stored as float numbers, which makes librosa augmentation deals with floating point operations. But using scipy and numpy packages, audio files are stored as integer numbers. Our implementation for audio augmentation class, as a result, has accelerated the process and epoch's time in the training phase takes about 15 minutes on google colab.

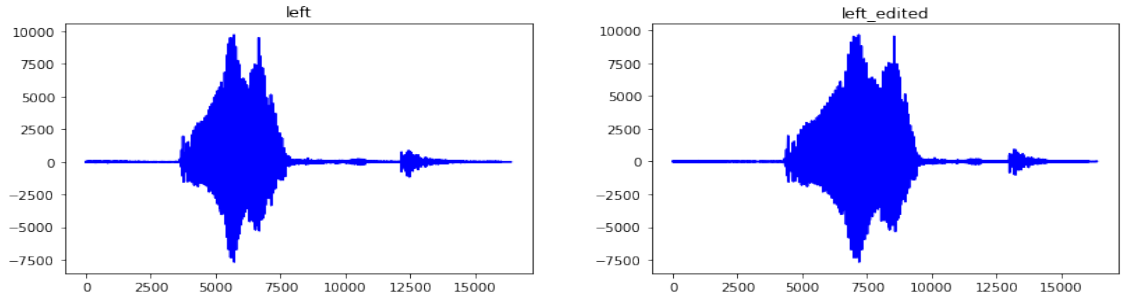
A sample from class 'Left' is taken to show the effect of each augmentation method on it, the following figures 5.8a, 5.8b show the command 'Left' in time and frequency domains.



(a) Time domain of command 'Left' (b) Frequency domain of command 'Left'

Figure 5.8: Time domain of command 'Left', Frequency domain of command 'Left'

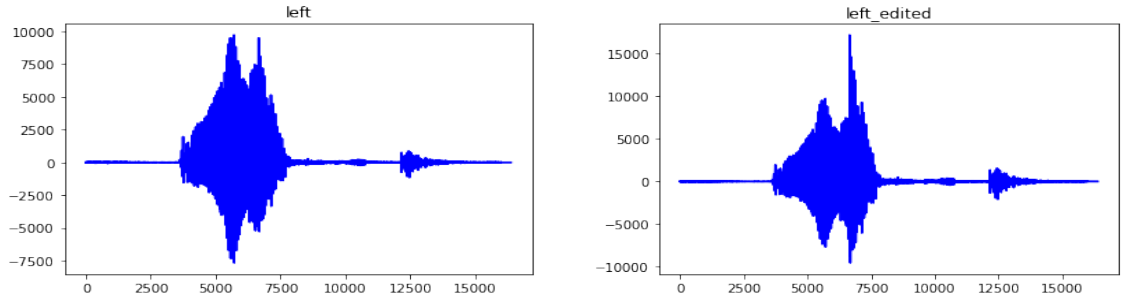
1. Changing speed:
   By using this function speed up or slow down the voice or part of it can be done at random proportions. The following figures 5.9b,shows its effect on 'Left' command:



(a) Time domain of command 'Left' before Changing speed (b) Time domain of command 'Left' after Changing speed

Figure 5.9: Time domain of command 'Left' before and after Changing speed

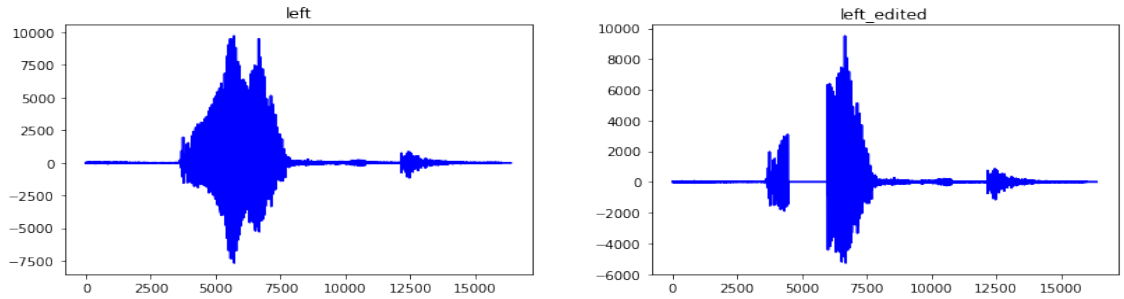
2. Volume control:
   This function enables us to control the volume level by increasing or decreasing or increasing part and decrease part or vice versa in random ways, which ensures the occurrence of most possible possibilities. The following figures 5.10b,shows its effect on 'Left' command:



(a) Time domain of command 'Left' before Volume control (b) Time domain of command 'Left' after Volume control

Figure 5.10: Time domain of command 'Left' before and after Volume control

3. Mask:
   This function disturbs or loses part of the sound, either by adding noise or losing part of the sound completely, which is what sometimes happens when using bad microphones so a certain zone can be selected then do a mask on it and the mask can be random noise or zeros(silence). The following figures 5.11b,shows its effect on 'Left' command:

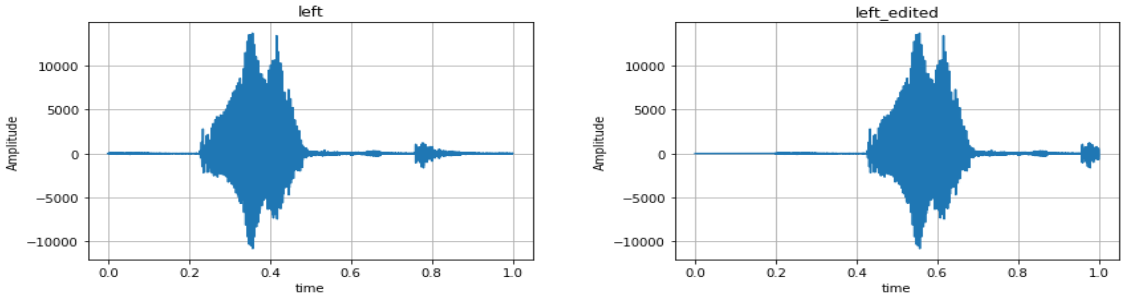


(a) Time domain of command 'Left' before applying Mask

(b) Time domain of command 'Left' after applying Mask

Figure 5.11: Time domain of command 'Left' before and after applying Mask

4. Time shifting:
   It shifts the audio given as an argument in time domain. Its parameters: the audio wanted to shift, sampling rate of the signal, direction to shift 'right, left or both which randomly choose the direction', and maximum limit on shifting in milliseconds for just keep the quality of the audio as good as possible. The shifting factor is randomly chosen between 0 to maximum limit of shift. The following figures 5.12a, 5.12b shows its effect on 'Left' command:

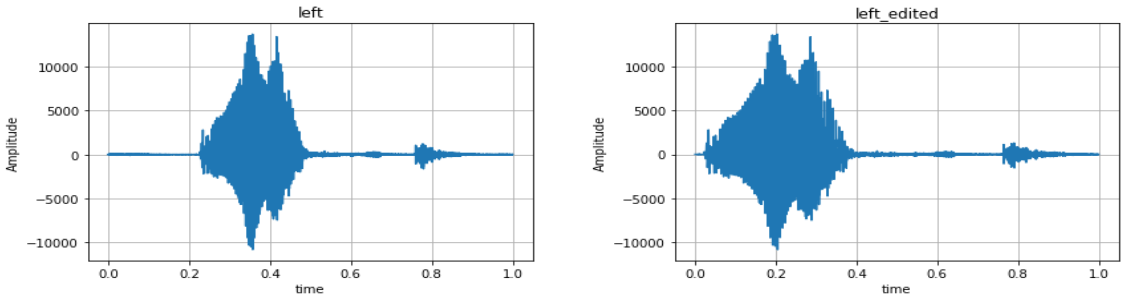


(a) Time domain of command 'Left'

(b) Time shifted version of command 'Left'

Figure 5.12: Time domain of command 'Left', Time shifted version of command 'Left'

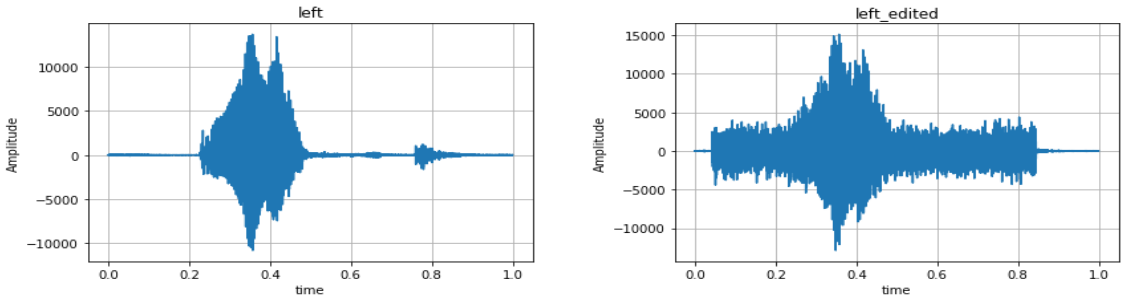
5. Cropping audio:
   It crops or cuts a small frame from the audio. Its parameters: the audio to make the process on, sampling rate, duration of the audio in seconds, and the start and end of cropping which are randomly chosen. The following figures 5.13a, 5.13b shows its effect:



(a) Time domain of command 'Left' (b) Cropped version of command 'Left'

Figure 5.13: Time domain of command 'Left', Cropped version of command 'Left'

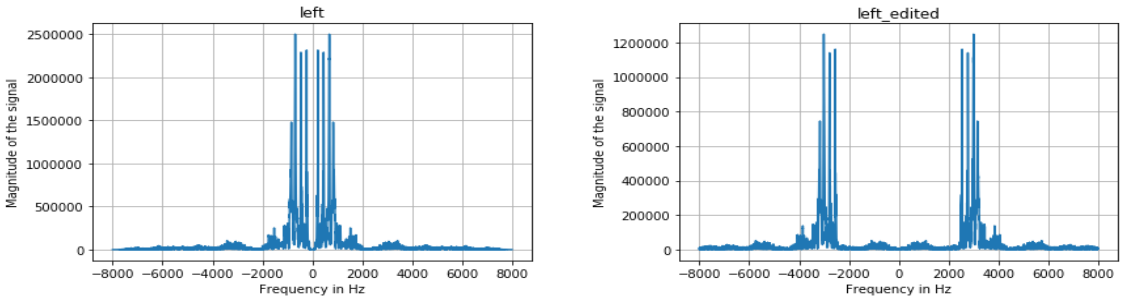
6. Adding random noise:
   It adds random noise to the signal, either white Gaussian noise or background noise that provided from the competition. Its parameters: the audio that noise is added to, the intensity of the noise and chosen randomly, augmented zone which is a range of the audio to apply augmentation on, and the background noise. The following figures 5.14a, 5.14b shows its effect:



(a) Time domain of command 'Left' (b) Noisy version of command 'Left'

Figure 5.14: Time domain of command 'Left', Noisy version of command 'Left'

7. Fourier transform:
   It shifts the audio in the frequency domain, by applying on it fast Fourier transform, then shifting the result and perform inverse Fourier transform. Its parameters: the audio, its sampling rate and shifting factor which is chosen randomly, it can be negative and figures out how many samples the audio is shifted. Shifting factor is a number between upper and lower bounds, which are chosen accurately to keep the quality of the audio as good as possible. The following figures 5.15a, 5.15b shows its effect:



(a) Frequency domain of command 'Left'

(b) Freq. shifted version of command 'Left'

Figure 5.15: Frequency domain of command 'Left', Freq. shifted version of command 'Left'

8. Pitch change:
   The scientific aspect of pitch change is shifting the fundamental frequency of the signal. Pitch-shifting is easy once you have sound stretching. If you want a higher pitch, you first stretch the sound while conserving the pitch, then you speed up the result, such that the final sound has the same duration as the initial one, but a higher pitch due to the speed change. Its implementation depends on two functions which are create-Frames' that stretches the original audio and fusion-Frames' that compresses the processing audio. The following figure 5.16 illustrates the process of changing the pitch:

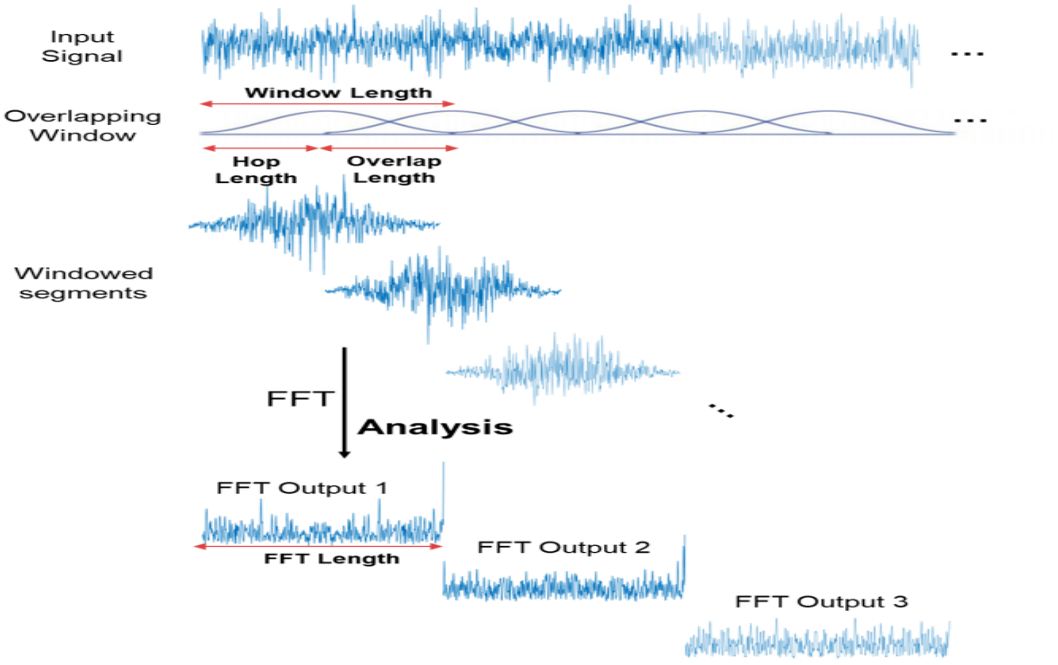


Figure 5.16: Pitch change mechanism

Pitch change function parameters:

(a) audio that will be augmented.

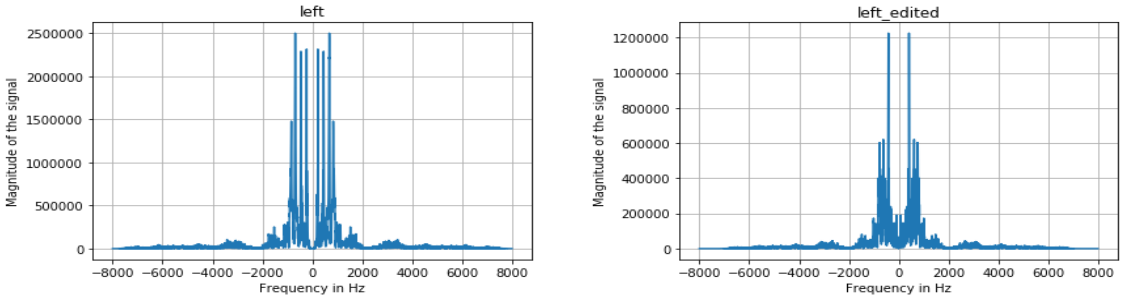
(b) Sampling rate of the audio.

(c) Window size: which depends on sampling frequency, it must be a number divisible by 2 and 2^8 is chosen after some tries to get a good quality.

(d) Overlap factor: which is taken to be 75% overlapping between the current frame and the next one and this also determines the size of the hop.

(e) Number of semitones: which is the number of shifting steps and of range [-10:10] for the quality of the audio.

The following figures 5.17a, 5.17b shows its effect:



(a) Frequency domain of command 'Left'

(b) Changing pitch of command 'Left'

Figure 5.17: Frequency domain of command 'Left', Changing pitch of command 'Left'

5.2.2.4 GAN as an augmentation network

Generative Adversarial Networks (GANs) have had a huge success since they were introduced in 2014 by Ian J. Goodfellow and co-authors in this paper [37], which are a type of neural network architecture that allow neural networks to generate data. In the past few years, they've become one of the hottest sub fields in deep learning, going from generating fuzzy images of digits to photo realistic images of faces. Generative Adversarial Networks are composed of two models: the first model is the generator which aims to generate new samples similar to the existed one, the second model is the discriminator and its goal is to recognize if an input data is real (belongs to original data distributions) or fake (generated from the generator).
GANs learn a probability distribution of a dataset by pitting two neural networks against each other. The generator is able to take random noise (latent space vector of a certain dimension) and map it to image which will be the input to the discriminator. The competition between these two models is what improves their knowledge, until the generator succeeds in creating realistic data. The following figure 5.18 shows how GANs work:

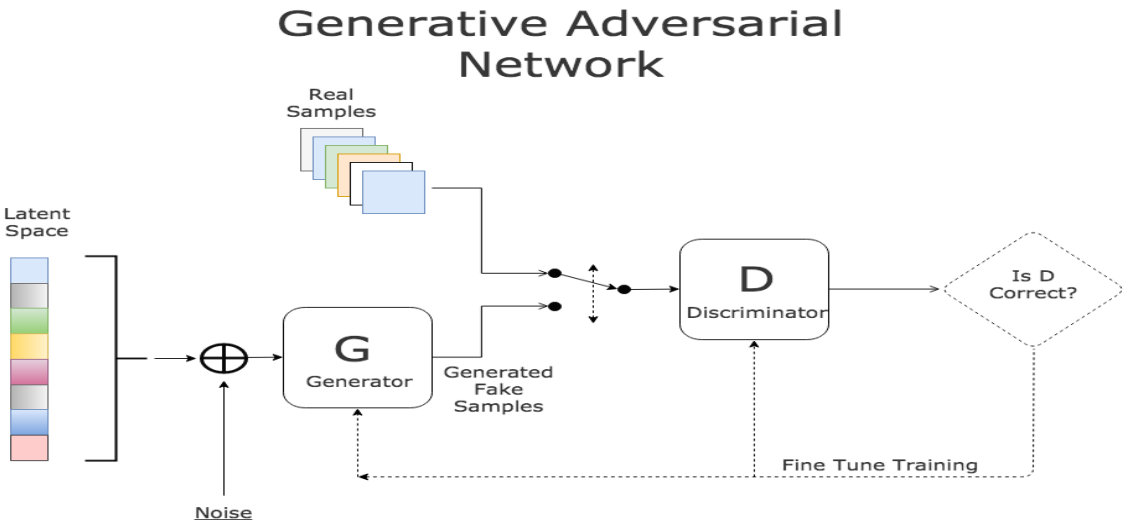


Figure 5.18: Understanding GAN mechanism

GANs are a form of unsupervised generative modeling, where you can just provide data and have the model create synthetic data from it. However, the state-of-the-art GANs use a technique called Conditional-GANs which turn the generative modeling task into a supervised learning one, requiring labeled data. In Conditional-GANs, class labels are embedded into the generator and discriminator to facilitate the generative modeling process. Unconditional GANs refer to Goodfellow's original idea, in which no class labels are needed for generative modeling.

Deep learning researchers work in two ways: the first one is making different GAN architectures by using new concepts, and the second is performing different mechanism in calculating loss function. Deep convolutional generative adversarial network (DCGAN) is one of the popular and successful network design for GAN [38]. It mainly composes of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling as shown in figure 5.19. Many GAN models take the architecture of DCGAN and change only the loss function that measures the similarity of the actual and the generated data.

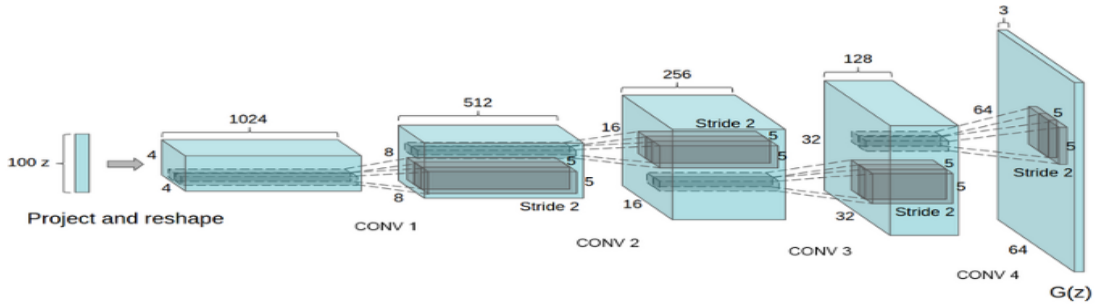


Figure 5.19: DCGAN architecture

5.2.2.4.1 Limitations on standard GANs

Standard GANs have some drawbacks that researchers must understand to improve GAN's performance properly. Researchers try to make GAN training more stable. These problems are areas of active research,and none of them have been completely solved. The first problem that standard GANs faced is vanishing gradients, it affects the training of the generator specifically and makes no updates to the parameters (weights and biases) and that due to superiority the discriminator in determining real-fake samples. The second problem is mode collapse, which is a mode that the discriminator gets stuck in a local minimum and the generator generates similar outputs to make the discriminator in this situation. If the generator starts producing the same output (or a small set of outputs) over and over again, the discriminator will learn to always reject that output. The third problem is failure to converge, As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. This progression poses a problem for convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse.

5.2.2.4.2 Data pre-processing

To make GAN ready for the training phase, the dataset must be in an appropriate format. So, each audio sample is converted to one channel mel-spectrogram (stft then scale the linear amplitude to dB) image of size $128 * 128$. Then the image is normalized by means and standard deviations of all dataset samples. After generating samples from GAN, it will be converted to audio again of sampling rate 16KHz and duration of one second by denormalize the image first, then perform inverse mel-spectrogram on it (istft between mel-spectrogram and phase which is random vector).

5.2.2.4.3 GAN architecture and training phase

WGAN is a neural network that tries to solve the problems encountered by previous GAN architectures [39]. This is done by using new concept in calculating loss function, that tells how similarity the generated samples are to the actual data by calculating the distance between the actual data distribution and the generated data distribution. This is called Wasserstein distance. This approach provides a solution to the first two problems discussed above.

5.2.2.4.4 Generation and labelling phase

After training the generator and discriminator, GAN has been tuned with weights that enables it to generate augmented versions of the actual data. In our case, more than 100K samples in total of classes 'yes, no, up, down, right, left, go, stop, on, off' are generated to begin our third training phase for some models. But before that, GAN labelling of the generated samples is checked by passing them as inputs to the best ensemble version of models to predict generated samples labels. The results will be discussed in section 5.4.

5.2.2.5 YouTube dataset generator

The amount of the labeled data is considered to be one of the most important characteristics in the deep learning world. Therefore, in order to increase accuracy and reduce losses, it was desired to generate data from YouTube because of the huge amount of audio data, because of that the videos are automatically translated by YouTube for the English language, that helps us to locate the words and build a unet model that can segment those words and collect a huge number of labeled data.

let's discuss the steps taken to collect the labeled data that needed to increase the data in classes that are desired to be classified.

1-first of all its needs to bring the channel's ID through where the audio files and the words are to be collected, and it must choose a channel that contains clear sound and is recorded in high quality to ensure the quality of the data that will be collected.

2- After selecting the appropriate channel, it's needed to pass it's ID to the library named Python-YouTube [40], which will bring the identities of the videos inside.

3- After bringing the identities of the videos, they are passed first to the library named YouTube-transcript-API [41], which will give the translation of the video, and when each sentence begins and its duration and this data will be needed later to collect the data. Secondly, the videos' IDs can be used to download those videos by passing the video ID to the library named pytube [42] and then convert it to mp3.

4- After obtaining the video as a sound and knowing each sentence through translation, when it appeared and the duration of its stay, Words in our classes can now be searched for, so now it can only take the sound clips of the sentence that contain our classes and the rest of the other sound clips are deleted, each audio clip called a chunk.

5- In the penultimate step, the method that will work on dividing the sentence and extracting our word are chosen. There are two methods for that, the first is by using the hard code, which depends on knowing the location of the word in the sentence by the subtitle and making a segmentation for the sentence by choosing a limit for the amplitude of the voice if the amplitude of the voice is less than it Then it will be considered silence so the sentence can be segmented and the word can be combined, but this method is very weak in segmentation because it does not depend on knowing the word and if the speaker does not leave a space between each word and the other which is what usually happens so this method will fail, The second method is by using deep learning by building a unet model that can learn and know the word to be collected, and this method will depend on the strength of the model that was built in order to be able to know the required word and bring it out.

6- After building a unet model and teaching it to learn the word and extract it from the sentence, the sentences (chunks) that were collected from the videos that contain the words to be collected are entered to the unet model and those words can be collected, and by this, a huge amount of data can be collected to improve the accuracy of our classifier.

Unfortunately, after building the unet model and trying to use it, it did not give a good results, because the data used to teach are from one source,so the model did not learn from different sources, so it could not work well when using it in another source, but it was giving very good results in the same source which is learned from.

5.3 Materials and Methods

5.3.1 Speech recognition classification networks

Speech or sound signals have many waveforms to visualize, if a change has affected a signal, the effect of this change may not be observed well in the time domain as an example, but in the frequency domain can represent that effect well. Speech signal also has many waveforms, which are time-domain waveform, which is very common, spectrogram, mel-spectrogram and MFCC. As discussed in the section of feature level, speech signal is treated and represented as an image that carries information about the signal (MFCC features).
The success of convolutional neural networks made data pre-processing an important step and widened the view to think of speech signals as images. This is a good point that CNN made a great success in many fields, so its implementation on another fields such as Speech recognition may reflect its success and make a breakthrough in that field. Our first attempt to solve the problem is to try CNN models and convert speech dataset into images (MFCC features).
Many different models are trained for solving the same problem. In this section, the used CNN models are covered from their architectures point of view. Our second attempt is to train a sequence model that its input is raw data (time domain waveform) with some extensions added to it. The variation in the architectures of the models used to solve the problem and the features is a good point that will help when ensemble learning is performed.

5.3.1.1 Sequence Models

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. RNN is very useful in speech recognition systems. In some cases it outperforms CNNs with the same number of parameters however, it takes a long time in training. In this section we have implemented the RNN proposed in the paper with some modifications on the input size to be less than the proposed [34]. Two approaches are taken to outperform the results of the paper [34]. The first one is using the semi supervised technique explained in section 5.3.1.3 while the second approach is the two stages classifiers which will be explained in this section.

"silence" and "unknown" classes are too large due to their good modeling explained in section 5.2.2.1. This forces us to do down sampling to these classes to get a balanced distribution. Two stages model solves this problem. The first stage is 2 or 3 classes classifier model that is able to take all the data size while the second stage is 11 or 10 classes classifier. This technique allows to use all the available data of the "unknown" and "silence" samples.

Figure 5.20 shows an example of 2 stages. The first model is a "silence"/"voiced" classifier while the second model classifies what the user have said if the first model passes the voiced word to it.

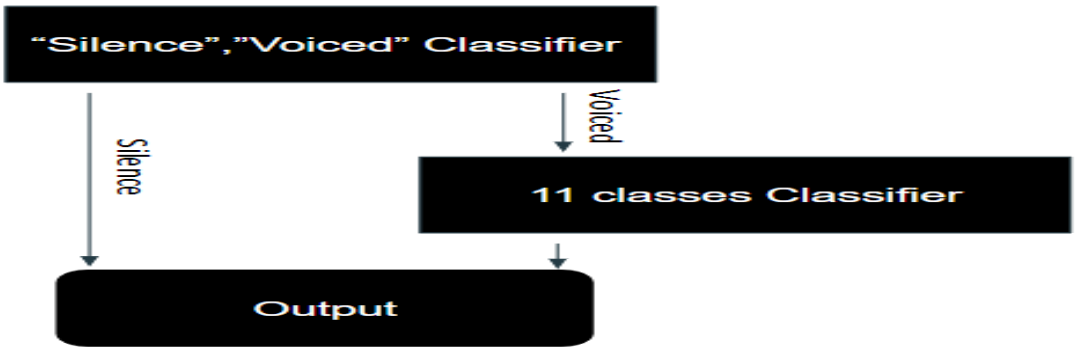


Figure 5.20: 2 stages block diagram for silence/voiced classifier

5.3.1.2 CNN Models

CNN is a type of neural network model which allows us to extract higher representations for the image content. Unlike the classical image recognition where you define the image features yourself, CNN takes the image's raw pixel data, trains the model, then extracts the features automatically for better classification.

There are many CNN models that have proven their ability to learn well and let's talk about these models.

1. VGG16 model and VGG19 model

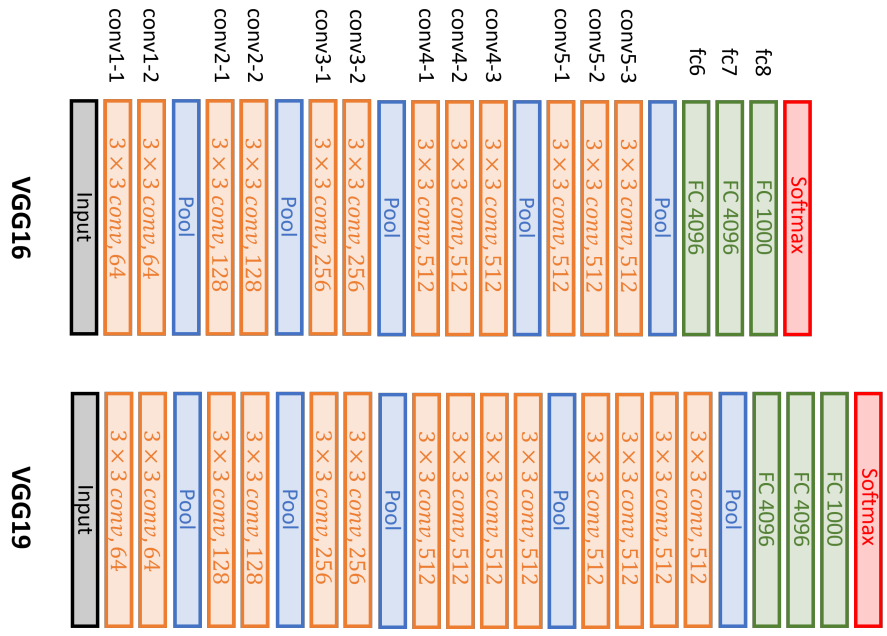


Figure 5.21: VGG16 and 19 model structure

VGG16 and VGG19 5.21 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" [43]. The model achieves 92.7% top-5 test accuracy in ImageNet 5.22, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014.

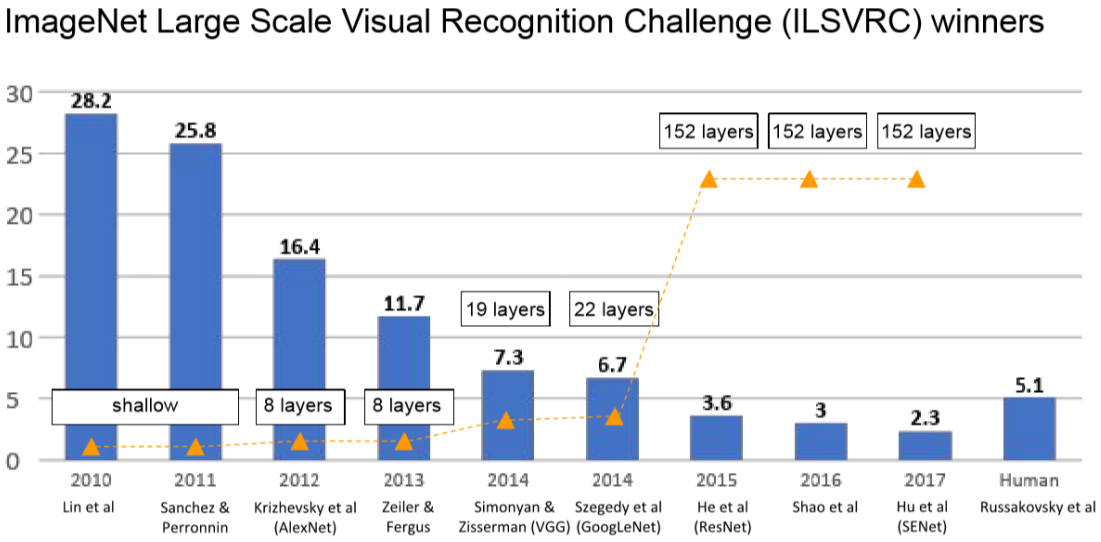


Figure 5.22: ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

It makes the improvement over AlexNet 5.23 by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple $3 * 3$ kernel-sized filters one after another.

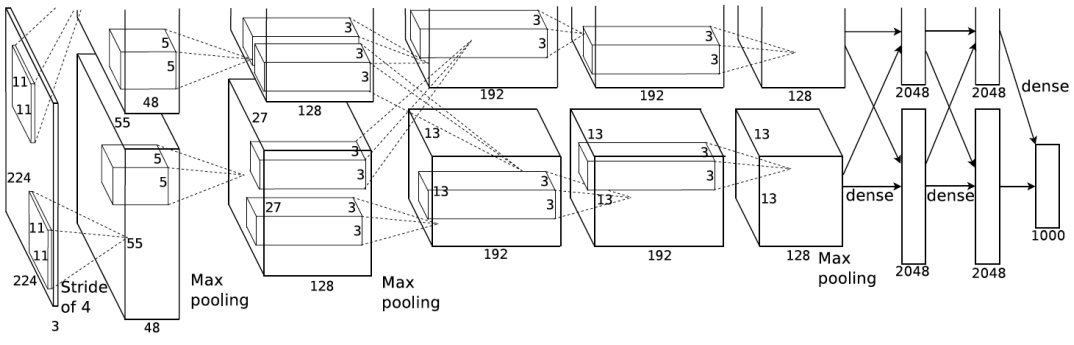


Figure 5.23: alexnet model structure

by using Keras to load pre-trained model of VGG16 and VGG19 then by removing the classifier of VGG16 and VGG19 and added our classifier Which consists of 2 FC layer followed by the output layer (softmax) as shown in 5.24

Layer (type)	Output Shape	Param #
Image_input (InputLayer)	(None, 99, 40, 3)	0
vgg16 (Model)	(None, 3, 1, 512)	14714688
global_average_pooling2d_1 (	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
batch_normalization_1 (Batch	(None, 512)	2048
dropout_2 (Dropout)	(None, 512)	0
predictions (Dense)	(None, 32)	16416

Total params: 15,258,464
Trainable params: 15,257,440
Non-trainable params: 1,024

(a) VGG16 layers

Layer (type)	Output Shape	Param #
Image_input (InputLayer)	(None, 99, 40, 3)	0
vgg19 (Model)	(None, 3, 1, 512)	20024384
global_average_pooling2d_1 (	(None, 512)	0
dense_1 (Dense)	(None, 1024)	525312
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
batch_normalization_1 (Batch	(None, 512)	2048
dropout_2 (Dropout)	(None, 512)	0
predictions (Dense)	(None, 32)	16416

Total params: 21,092,960
Trainable params: 21,091,936
Non-trainable params: 1,024

(b) VGG19 layers

Figure 5.24: VGG19 and VGG16 layers

2. resnet50 model

Figure 5.25: resnet50 model structure

ResNet 5.25 is a powerful backbone model that is used very frequently in many computer vision tasks, as it is the first net to outperform human imagenet 5.22, ResNet uses skip connection to add the output from an earlier layer to a later layer. This helps it mitigate the vanishing gradient problem

by using Keras to load pre-trained model of resnet50 then by removing the classifier of resnet50 and added our classifier Which consists of 2 FC layer followed by the output layer (softmax) as shown in 5.26

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Image_input (InputLayer)     (None, 99, 40, 3)         0
_________________________________________________________________
resnet50 (Model)             (None, 4, 2, 2048)        23587712
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
predictions (Dense)          (None, 32)                16416
=================================================================
Total params: 26,229,152
Trainable params: 26,175,008
Non-trainable params: 54,144
_________________________________________________________________
```

Figure 5.26: resnet50 layers

3. densenet model

   DenseNet 5.27 is a new CNN architecture that reached State-Of-The-Art (SOTA) results on classification datasets (CIFAR, SVHN, ImageNet) using less parameters, thanks to its new use of residual it can be deeper than the usual networks and still be easy to optimize.

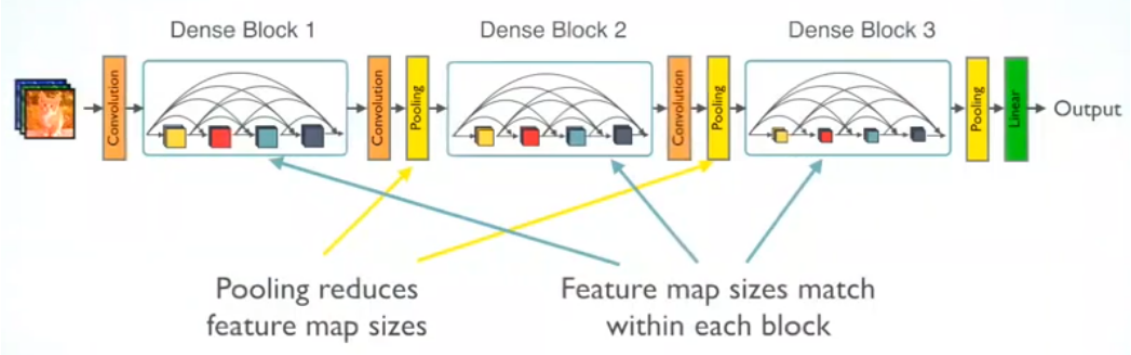


Figure 5.27: densenet structure

DenseNet is composed of Dense blocks 5.28. In those blocks, the layers are densely connected together: Each layer receive in input all previous layers output feature maps, this extreme use of residual creates a deep supervision because each layer receive more supervision from the loss function thanks to the shorter connections.

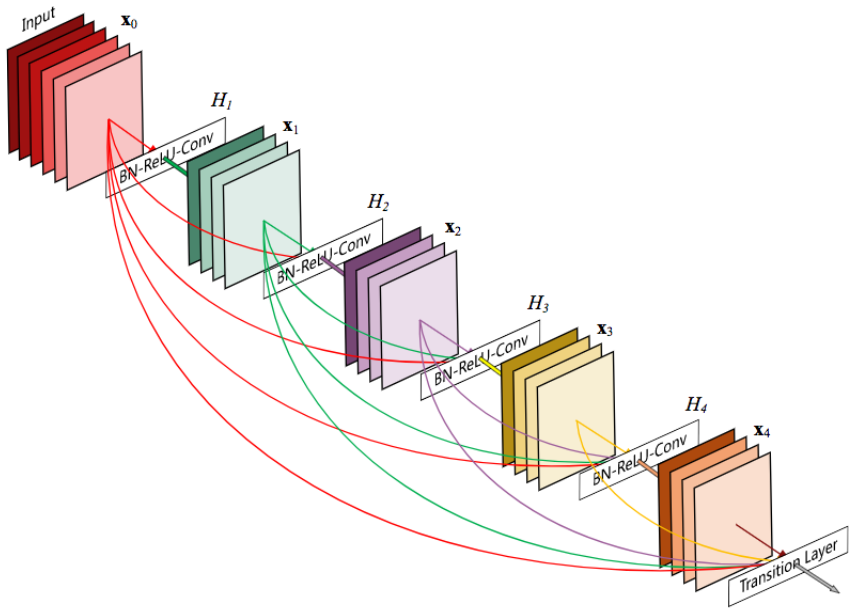


Figure 5.28: Dense blocks

Instead of summing the residual like in ResNet, DenseNet concatenates all the feature maps, DenseNet has lower need of wide layers because as layers are densely connected there is little redundancy in the learned features. All layers of a same dense block share a collective knowledge.

The final architecture of DenseNet is as shown in 5.29

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

Figure 5.29: densenet Architecture

by using Keras to load pre-trained model of DenseNet then by removing the classifier of DenseNet and added our classifier Which consists of 2 FC layer followed by the output layer (softmax) as shown in 5.30

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Image_input (InputLayer)     (None, 99, 40, 3)         0
_________________________________________________________________
densenet169 (Model)          (None, 3, 1, 1664)        12642880
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1664)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               852480
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
predictions (Dense)          (None, 32)                4128
=================================================================
Total params: 13,565,664
Trainable params: 13,407,008
Non-trainable params: 158,656
_________________________________________________________________
```

Figure 5.30: densenet layers

To summarize, the DenseNet architecture uses the residual mechanism to its maximum by making every layer (of a same dense block) connect to their subsequent layers, This model's compactness makes the learned features non-redundant as they are all shared through a common knowledge, It is also far more easy to train deep network with the dense connections because of an implicit deep supervision where the gradient is flowing back more easily thanks to the short connections [44].

4. mobilenet model

   The best feature in mobilenet 5.31 is it runs very efficiently on mobile devices and is nearly as accurate as much larger convolutional networks like our good friend VGGNet-16 [45].

Figure 5.31: mobilenet Architecture

The big idea behind MobileNets: Use depthwise separable convolutions 5.32 to build light-weight deep neural networks, When these two things are put together — a depthwise convolution followed by a pointwise convolution — the result is called a depthwise separable convolution. A regular convolution does both filtering and combining in a single go, but with a depthwise separable convolution these two operations are done as separate steps.

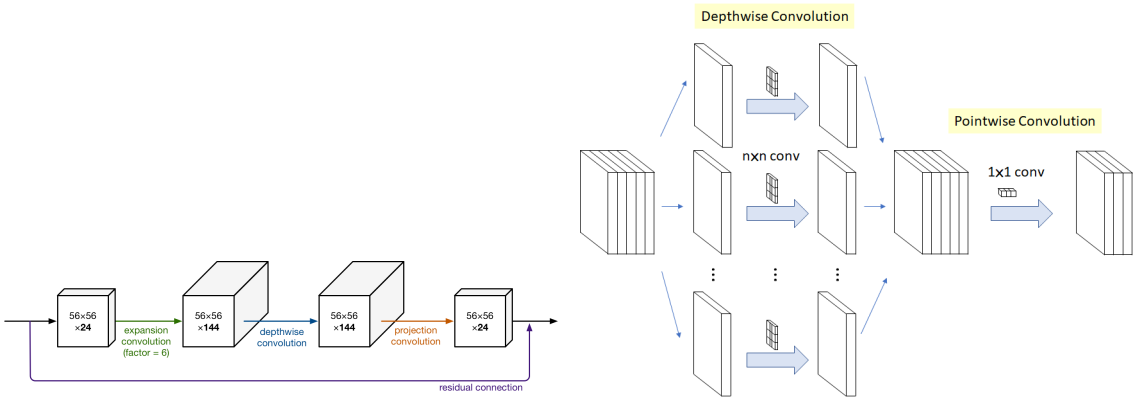


Figure 5.32: Depthwise Separable Convolution

The purpose of the depthwise convolution is to filter the input channels. Think edge detection, color filtering, and so on, The purpose of this pointwise convolution is to combine the output channels of the depthwise convolution to create new features, But Why do this? The end results of both approaches are pretty similar — they both filter the data and make new features — but a regular convolution has to do much more computational work to get there and needs to learn more weights.

So even though it does (more or less) the same thing, the depthwise separable convolution is going to be much faster! It's about 3x as fast as Inception and 10 as fast as VGGNet-16, and it uses way less battery power than both. That's largely due to the much smaller number of learned parameters (4 million versus 24 million for Inception-v3 and 138 million for VGGNet-16).
Accessing memory is the biggest drain on battery power, so having many fewer parameters is a big plus.

Speed isn't everything, of course. MobileNets is only useful if it's also accurate. So how accurate is it? By experience [46] MobileNets is quite close to VGGNet indeed,That's great because VGGNet-16 is often used as a feature extractor for other neural networks, so you can now simply replace that part of the network with this new MobileNets model and get an immediate 10x speed-up.

by using Keras to load pre-trained model of mobilenet then by removing the classifier of mobilenet and added our classifier Which consists of 2 FC layer followed by the output layer (softmax) as shown in 5.33.

Layer (type)	Output Shape	Param #
Image_input (InputLayer)	(None, 99, 40, 3)	0
mobilenet_1.00_224 (Model)	(None, 3, 1, 1024)	3228864
global_average_pooling2d_3 (	(None, 1024)	0
dense_5 (Dense)	(None, 512)	524800
dropout_5 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 512)	262656
batch_normalization_3 (Batch	(None, 512)	2048
dropout_6 (Dropout)	(None, 512)	0
predictions (Dense)	(None, 32)	16416

Total params: 4,034,784
Trainable params: 4,011,872
Non-trainable params: 22,912

Figure 5.33: mobilenet layers

5. xception model

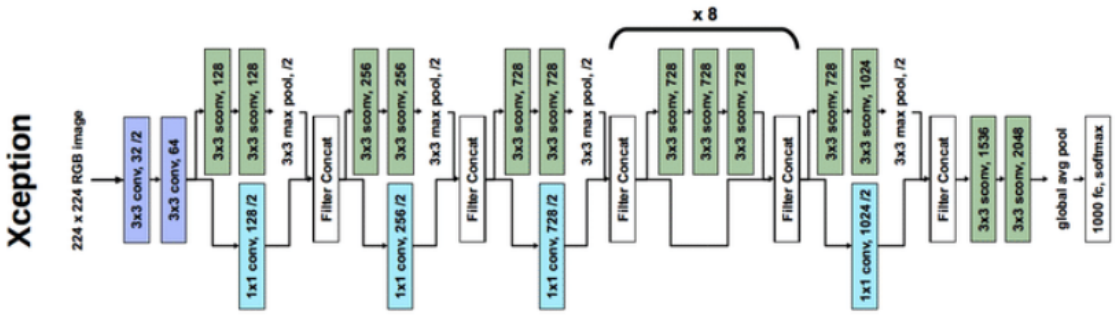
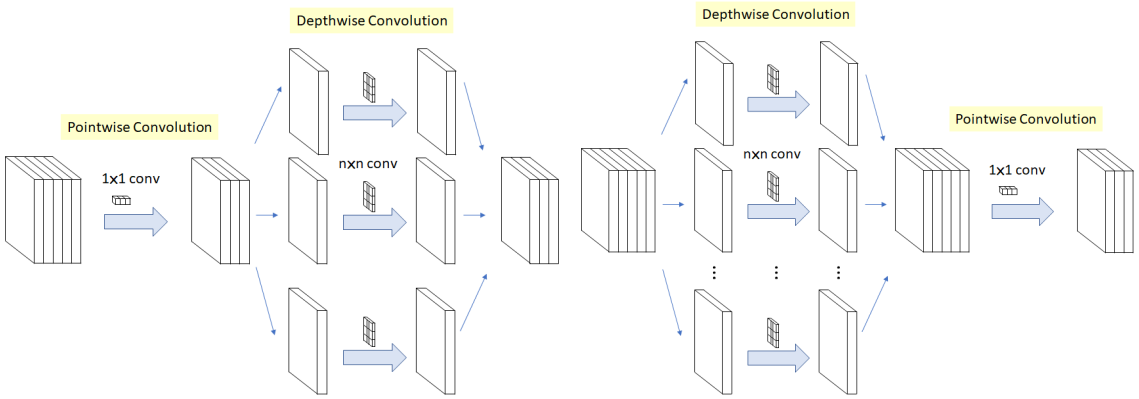


Figure 5.34: xception Architecture

The big idea behind xception 5.34: is Used Modified depthwise separable convolutions 5.35a which is the pointwise convolution followed by a depthwise convolution, and the Two minor differences between Modified depthwise separable convolutions and depthwise separable convolutions 5.35b are The order of operations and The Presence/Absence of Non-Linearity as In Xception, the modified depthwise separable convolution, there is NO intermediate ReLU non-linearity.



(a) Modified Depthwise Separable Convolution (b) Depthwise Separable Convolution

The modified depthwise separable convolution with different activation units are tested. the Xception without any intermediate activation has the highest accuracy compared with the ones using either ELU or ReLU.

Xception outperforms VGGNet, ResNet, and Inception-v3 in ImageNet — ILSVRC [47].

by using Keras to load pre-trained model of Xception then by removing the classifier of Xception and added our classifier Which consists of 2 FC layer followed by the output layer (softmax) as shown in 5.36.

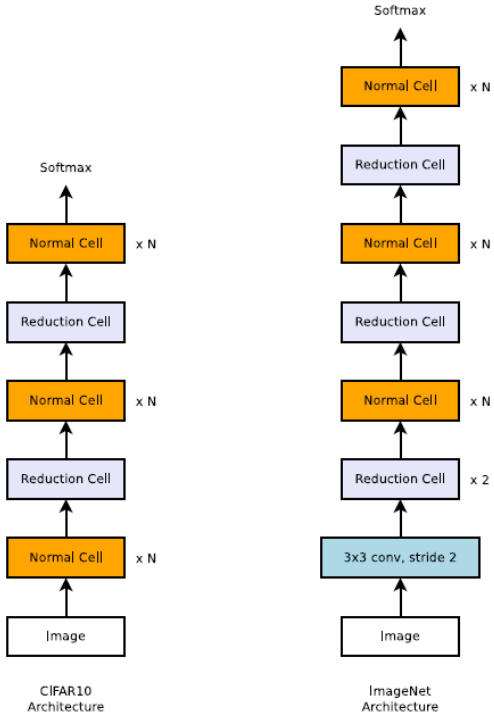
Layer (type)	Output Shape	Param #
Image_input (InputLayer)	(None, 99, 40, 3)	0
xception (Model)	(None, 3, 2, 2048)	20861480
global_average_pooling2d_3 (	(None, 2048)	0
dense_5 (Dense)	(None, 1024)	2098176
dropout_5 (Dropout)	(None, 1024)	0
dense_6 (Dense)	(None, 1024)	1049600
batch_normalization_290 (Bat	(None, 1024)	4096
dropout_6 (Dropout)	(None, 1024)	0
predictions (Dense)	(None, 32)	32800

Total params: 24,046,152
Trainable params: 23,989,576
Non-trainable params: 56,576

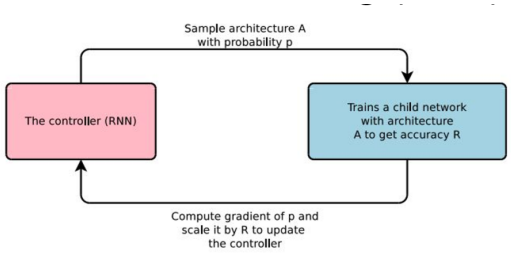
Figure 5.36: xception layers

6. Nasnet model

   In NASNet, though the overall architecture is predefined as shown in 5.37a, the blocks or cells are not predefined by authors. Instead, they are searched by reinforcement learning search method, the number of motif repetitions N and the number of initial convolutional filters are as free parameters, and used for scaling, Specifically, these cells are called Normal Cell and Reduction Cell, Normal Cell: Convolutional cells that return a feature map of the same dimension, Reduction Cell: Convolutional cells that return a feature map where the feature map height and width is reduced by a factor of two, Only the structures of (or within) the Normal and Reduction Cells are searched by the controller RNN (Recurrent Neural Network) 5.37b.



(a) Scalable Architectures for CIFAR-10 and ImageNet



(b) controller RNN (Recurrent Neural Network)

Nasnet 5.38 is "Controller" network that learns to design a good network architecture (output a string corresponding to network design).

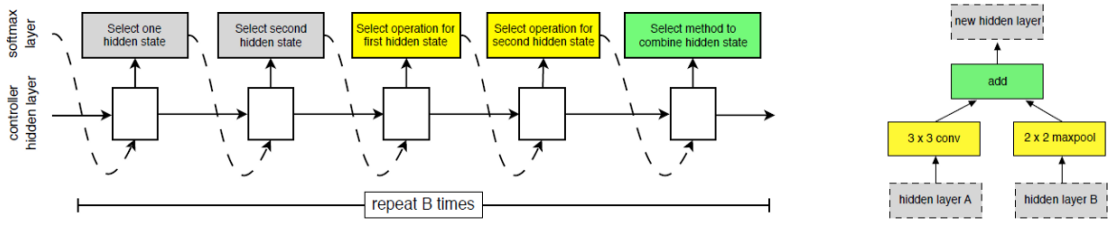


Figure 5.38: Controller model architecture for recursively constructing one block of a convolutional cell

how it Iterate?

(a) Sample an architecture from search space

(b) Train the architecture to get a "reward" R corresponding to accuracy

(c) Compute gradient of sample probability, and scale by R to perform controller parameter update 5.37b. (increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)

Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive , Design a search space of building blocks ("cells") that can be flexibly stacked, NASNet is Used NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet.

NASNets outperform Inception-v1, MobileNet-V1 and ShuffleNet-V1 with higher accuracy but with similar or smaller models, and by Using Faster R-CNN, NASNet-A outperforms MobileNetV1, ShuffleNet V1, ResNet, and Inception-ResNet-v2 [48].

by using Keras to load pre-trained model of NASNets then by removing the classifier of NASNets and added our classifier Which consists of 2 FC layer followed by the output layer (softmax) as shown in 5.39.

Layer (type)	Output Shape	Param #
Image_input (InputLayer)	(None, 99, 40, 3)	0
NASNet (Model)	(None, 4, 2, 4032)	84916818
global_average_pooling2d_5 (	(None, 4032)	0
dense_9 (Dense)	(None, 512)	2064896
dropout_9 (Dropout)	(None, 512)	0
dense_10 (Dense)	(None, 512)	262656
batch_normalization_5 (Batch	(None, 512)	2048
dropout_10 (Dropout)	(None, 512)	0
predictions (Dense)	(None, 32)	16416

Total params: 87,262,834
Trainable params: 87,065,142
Non-trainable params: 197,692

Figure 5.39: NASNet layers

There are another neural networks that are trained to solve the problem, which are DenseNet-BC (densenet with bottleneck extension and compression ratio) [44], WideResNet which is a neural network that solves the problems faced ResNet [49], ResNext which provides a new factor to update ResNet [50], and DPN-92 [51]. All these neural networks are randomly initialized, this helps in breaking symmetry and every neuron is no longer performing the same computation. These models are implemented on PyTorch framework, and have improved the performance and increased the submission accuracy, and that will be discussed in section 5.4.

5.3.1.3 Semi supervised learning

Semi-supervised learning is an approach that makes a big breakthrough in machine learning and deep learning fields, it is an intermediate between supervised and unsupervised learning. Supervised learning needs training data with its ground truth labels to feed the neural network with, while unsupervised learning is a technique used with unlabelled data. Semi-supervised learning is a recommended technique when the dataset is mixture between labelled and unlabelled data, and the labelled data is small compared to the unlabelled one. The figure below 5.40 shows the difference between three learning techniques.

Amazon, which is one of the biggest company in the field of AI, used this approach to train their voice-driven intelligent assistant 'Alexa', and improve its performance. In 2017, they said that semi-supervised approach enhances Alexa's spoken language understanding by more than 25 percent over the last 12 months.

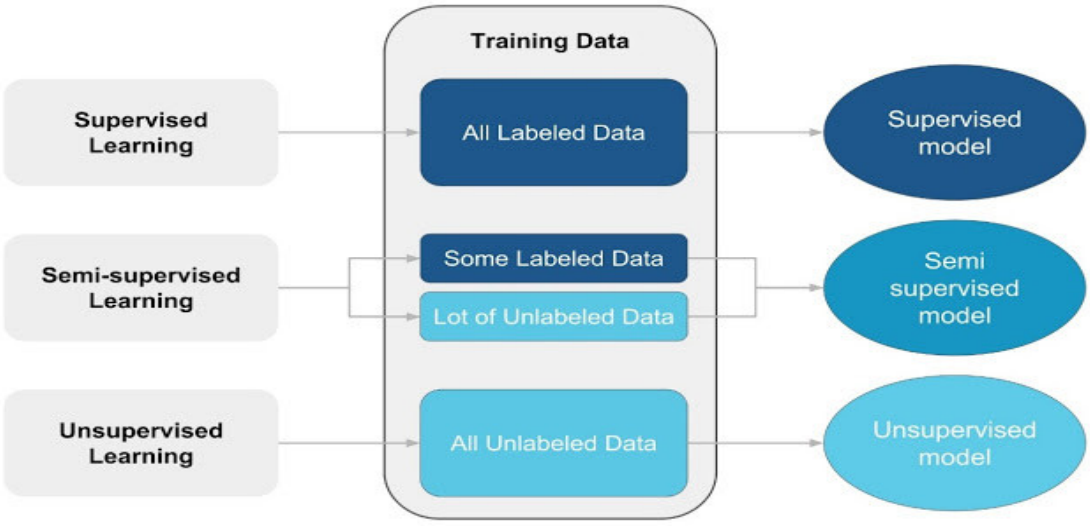


Figure 5.40: Illustration of semi-supervised learning

Nowadays, data is a very important factor in deep learning field, which is the fuel of the neural network to learn and improve itself. Due to the expensiveness and the difficultly of collecting new data and annotating it, semi-supervised technique is applied to increase the amount of data and train our models well. As discussed in the section of dataset that the competition provides a submission dataset that kaggle evaluates the score on it, but it is unlabelled. After training our models with annotated dataset using supervised learning, the models have been tuned with a set of weights that can predict answers for similar untagged data, and the submission data has been used for this purpose.

The implementation of ensemble learning is a great idea in this situation, so the outputs of four CNN models are combined together to get the best prediction. After that, this labelled data has been an input to the models to retrain them well and tune their weights.

Semi-supervised technique enhances the results of submission accuracy by 0.1 to 0.3 %, which is a satisfactory effect. The results of this powerful technique are covered in results and discussion section. The following figure 5.41 simply illustrates the idea of semi-supervised learning:

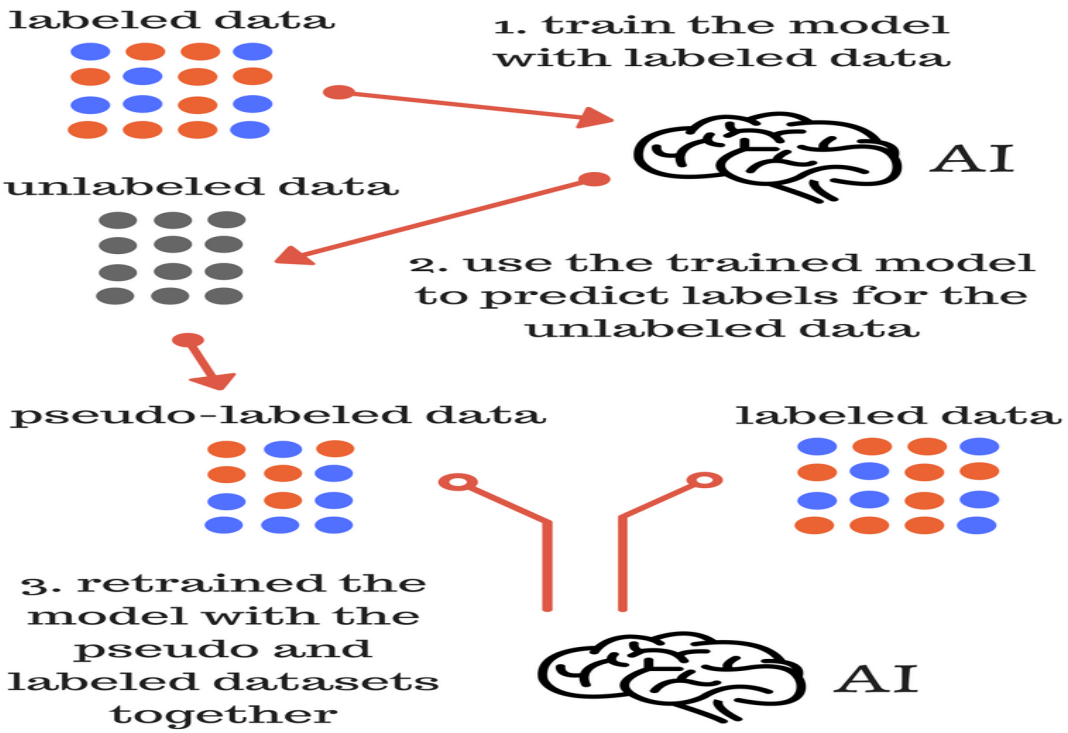


Figure 5.41: Pseudo labelling technique

5.4 Results and Discussion

The results of all the proposed CNN models in section Speech recognition classification networks are shown in table 5.2 while RNN models are shown in table 5.3.

Table 5.2: Results of all the proposed CNN models

Model	Val. acc.	Test. acc.	Val. loss	Test loss	Parameters	msec/sample
Xception	0.906	0.908	0.629	0.702	24.046 M	0.41
Mobilenet	0.949	0.954	0.2	0.173	4.034 M	0.138
Resnet50	0.956	0.959	0.167	0.152	26.229 M	0.414
Densenet	0.959	0.964	0.145	0.137	13.565 M	0.69
Nasnet	0.939	0.946	0.258	0.219	87.262 M	1.5
VGG16	0.965	0.963	0.137	0.138	15.258 M	0.27
VGG19	0.957	0.962	0.157	0.139	21.092 M	0.27
Densenet (semi)	0.959	0.962	0.153	0.139	13.776 M	0.69
Mobilenet (semi)	0.872	0.874	0.435	0.430	4.034 M	0.138
Nasnet (semi)	0.943	0.945	0.22	0.201	87.262 M	1.5
Resnet50 (semi)	0.954	0.954	0.173	0.171	26.229 M	0.414
VGG16 (semi)	0.957	0.958	0.17	0.163	15.258 M	0.27
VGG19 (semi)	0.957	0.96	0.166	0.155	20.193 M	0.27
Densenet (random initialized)	0.952	0.958	0.202	0.189	0.769 M	3.3
Densenet (semi) (random initialized)	0.948	0.95	0.193	0.172	0.769 M	3.3
DPN92	0.951	0.954	0.201	0.19	34.240 M	4.6
DPN92 (semi)	0.94	0.94	0.19	0.172	34.240 M	4.6
Resnext	0.935	0.937	0.244	0.229	34.427 M	4.8
Resnext (semi)	0.942	0.947	0.231	0.219	34.427 M	4.8
WideResnet	0.963	0.97	0.18	0.165	75.205 M	5.48
WideResnet (semi)	0.959	0.961	0.191	0.166	75.205 M	5.48

Table 5.3: Results of all the proposed RNN models

Model	Val. acc.	Test. acc.	Val. loss	Test loss	Parameters	Input Size	msec/sample
RNN	0.927	0.933	0.246	0.249	1.292 M	(80*125)	2.48
RNN (2 stages silence)	0.987	0.989	0.218	0.211	4.96 M	(80*125)	6
RNN (2 stages unknown)	0.928	0.928	0.423	0.5	4.96 M	(80*125)	6
RNN (2 stages 3 classes)	0.926	0.93	0.472	0.461	4.96 M	(80*125)	6.2
RNN	0.932	0.936	0.237	0.222	4.407 M	(32*32)	0.69
RNN (2 stages silence)	0.987	0.989	0.211	0.217	8.814 M	(32*32)	3
RNN (2 stages unknown)	0.956	0.959	0.167	0.152	8.814 M	(32*32)	3
RNN (2 stages 3 classes)	0.93	0.932	0.44	0.487	8.814 M	(32*32)	3.3
RNN (semi)	0.961	0.963	0.141	0.131	4.407 M	(32*32)	0.69
RNN (semi) (2 stages silence)	0.995	0.995	0.106	0.109	8.814 M	(32*32)	3
RNN (semi) (2 stages unknown)	0.964	0.959	0.265	0.307	8.814 M	(32*32)	3
RNN (semi) (2 stages 3 classes)	0.963	0.962	0.284	0.276	8.814 M	(32*32)	3.3

Different combinations of ensemble learning between models have been carried out targeting to get the best accuracy on submission dataset (150k samples). The best combination of models is Densenet (randomly initialized) (semi supervised), Wideresnet, Resnext (semi supervised) ,DPN92 and RNN (32*32) (semi supervised). This combination has achieved accuracy of 0.9755, 0.973 and 90.238 on the validation, test and submission set respectively. This achievement placed our team on the 25^{th} out of 1314 competitors on the competition. The Confusion matrix of the validation set is shown in figure 5.42 while that of the test set is shown in figure 5.43.

The Classification report of the validation set is shown in figure 5.44 while that of the test set is shown in figure 5.45.

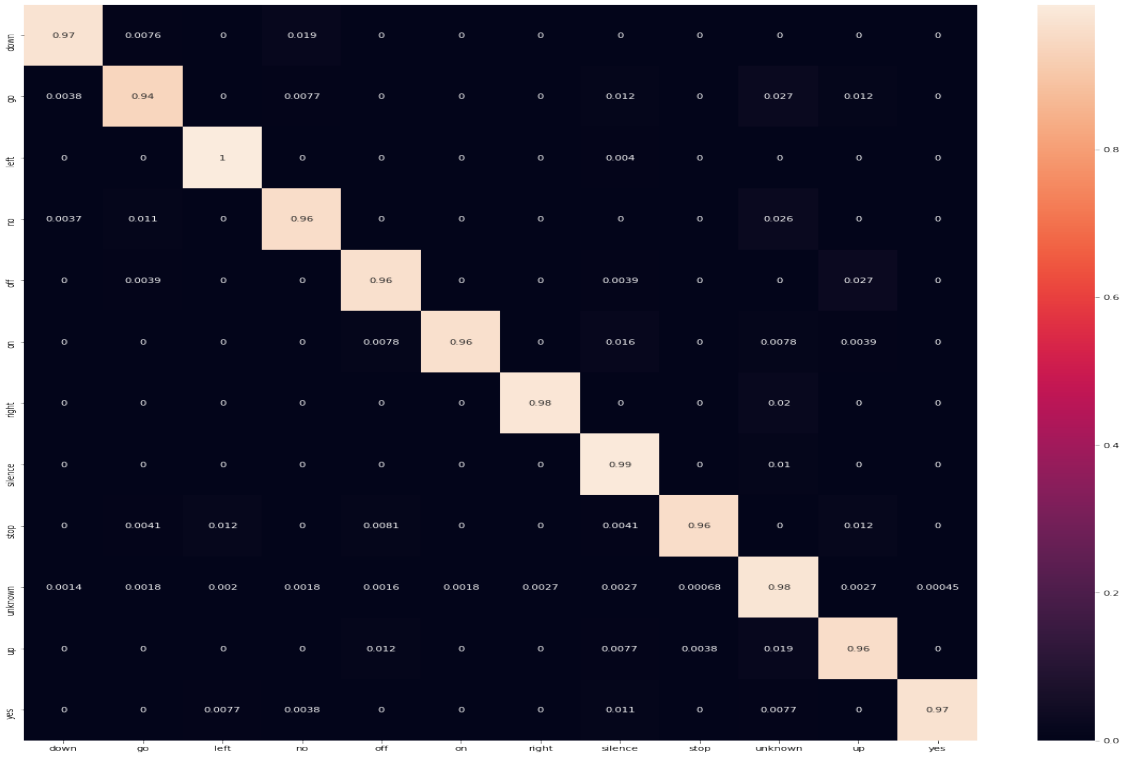


Figure 5.42: Confusion matrix of the validation set

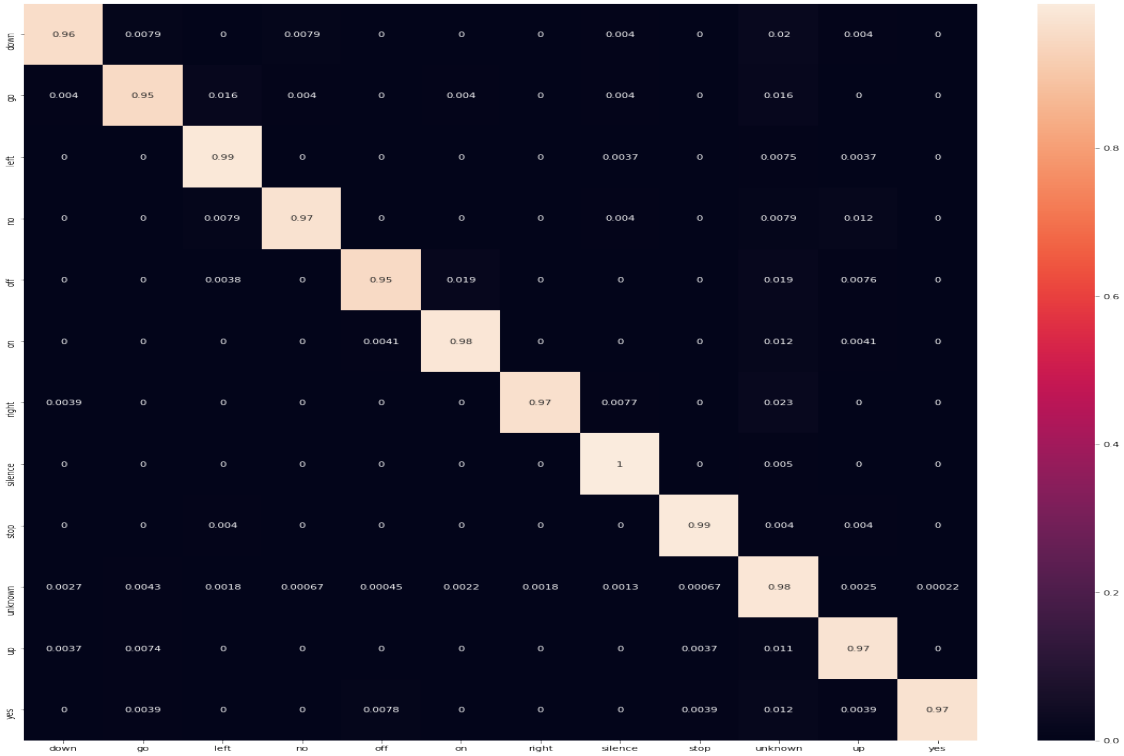


Figure 5.43: Confusion matrix of the test set

	precision	recall	f1-score	support
down	0.97	0.97	0.97	264
go	0.94	0.94	0.94	260
left	0.95	1.00	0.97	247
no	0.94	0.96	0.95	270
off	0.95	0.96	0.96	256
on	0.97	0.96	0.97	257
right	0.95	0.98	0.97	256
silence	0.88	0.99	0.93	199
stop	0.98	0.96	0.97	246
unknown	0.99	0.98	0.99	4421
up	0.91	0.96	0.93	260
yes	0.99	0.97	0.98	261
accuracy			0.98	7197
macro avg	0.95	0.97	0.96	7197
weighted avg	0.98	0.98	0.98	7197

Figure 5.44: Classification report of the validation set

	precision	recall	f1-score	support
down	0.94	0.96	0.95	253
go	0.91	0.95	0.93	251
left	0.94	0.99	0.96	267
no	0.98	0.97	0.97	252
off	0.98	0.95	0.97	262
on	0.94	0.98	0.96	246
right	0.97	0.97	0.97	259
silence	0.94	1.00	0.97	201
stop	0.98	0.99	0.98	249
unknown	0.99	0.98	0.99	4470
up	0.93	0.97	0.95	272
yes	1.00	0.97	0.98	256
accuracy			0.98	7238
macro avg	0.96	0.97	0.96	7238
weighted avg	0.98	0.98	0.98	7238

Figure 5.45: Classification report of the test set

Chapter 6

Discussion and Conclusion

6.1 Conclusion

All in all, we have presented three modules in order to minimize the car crashes. First module, lane overtaking, which helps in detecting the obstacles and take suitable decisions according to the surrounding conditions whether by stopping or by making lane change. Second module, distraction detection, at which it detects whether the driver is internally distracted or not and how it is distracted (drinking, talking on the phone, operating the radio, makeup,.... etc). Third module, speech recognition, at which it assists the driver to interact with the surrounding environment and remove the distraction that may cause car crashes.

As shown in figure 6.1, there may be an integration of the three modules where first, it detects whether the driver is distracted or not. If yes there will be an alarm then it will check the input command, if not it will check the input command. If it is lane overtaking, it will see whether it can be done or not according to the surrounding situations and take the suitable actions. If it is not lane overtaking, it will do the event according to the input command then return to see whether the driver is distracted or not and so on.

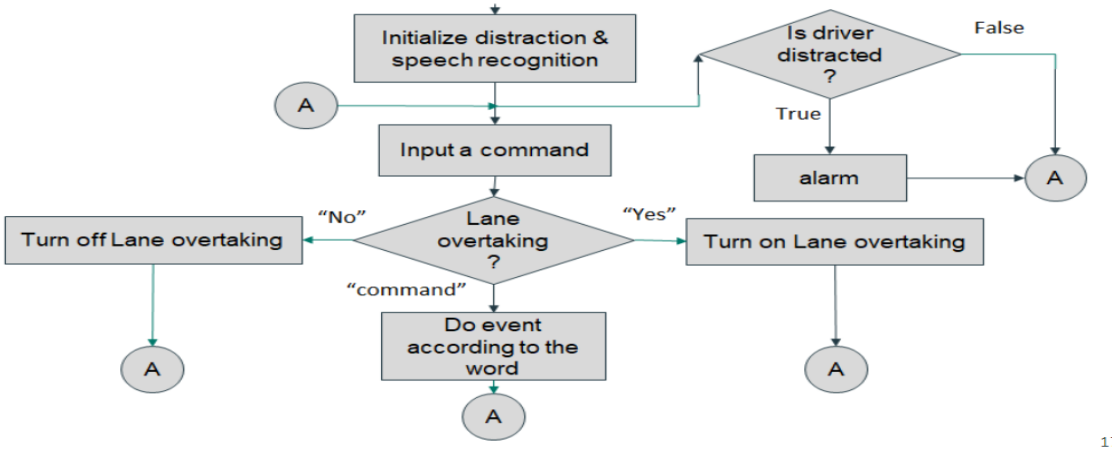


Figure 6.1: Block diagram of the proposed system

References

[1] J. E. Naranjo, C. Gonzalez, R. Garcia, and T. De Pedro, "Lane-change fuzzy control in autonomous vehicles for the overtaking maneuver", *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 3, pp. 438–450, 2008.

[2] W. H. Organization, "World health statistics 2017: Monitoring health for the sdgs, sustainable development goals", 2017.

[3] National Highway Traffic Safety Administration, *Traffic safety facts research notes 2016: Distracted driving.s. department of transportation, washington, dc: Nhtsa; 2015*. [Online]. Available: `https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812517`.

[4] T. M. Pickrell, H. R. Li, and S. KC, *Traffic safety facts, 2016*. [Online]. Available: `https://www.nhtsa.gov/risky-driving/distracted-driving`.

[5] U. D. o. H. &. H. Services, *Distracted driving, 2016*. [Online]. Available: `https://www.cdc.gov/%20motorvehiclesafety/distracted_driving/`.

[6] P. Warden, "Speech commands: A public dataset for single-word speech recognition.", *Dataset available from https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/data*, 2017.

[7] *The 5 levels of autonomous vehicles*, 2018. [Online]. Available: `https://www.truecar.com/blog/5-levels-autonomous-vehicles/`.

[8] *The 5 autonomous driving levels explained*, 2020. [Online]. Available: `https://www.iotforall.com/5-autonomous-driving-levels-explained/`.

[9] *Self-driving car fundamentals: The five stages of automation*, 2019. [Online]. Available: `https://www.wiredbrief.com/artificial-intelligence/self-driving-car-fundamentals-the-five-stages-of-automation/`.

[10] *Machine learning definition*, 2020. [Online]. Available: `https://expertsystem.com/machine-learning-definition/`.

[11] *History of machine learning*, 2018. [Online]. Available: `https://www.doc.ic.ac.uk/~jce317/history-machine-learning.html`.

[12] P. Analytics, *What is machine learning: Definition, types, applications and examples | potentia analytics inc.* 2019. [Online]. Available: `https://www.potentiaco.com/what-is-machine-learning-definition-types-applications-and-examples/`.

[13] J. Ni, Y. Chen, Y. Chen, J. Zhu, D. Ali, and W. Cao, "A survey on theories and applications for self-driving cars based on deep learning methods", *Applied Sciences*, vol. 10, no. 8, p. 2749, 2020.

[14] *Deep learning*, 2019. [Online]. Available: `https://www.investopedia.com/terms/d/deep-learning.asp`.

[15] *Deep learning*. [Online]. Available: https://en.wikipedia.org/wiki/Deep_learning.

[16] *The history of deep learning*, 2018. [Online]. Available: https://www.quora.com/What-is-the-history-of-deep-learning.

[17] *Deep learning models*, 2019. [Online]. Available: https://towardsdatascience.com/6-deep-learning-models-10d20afec175.

[18] A. Mosavi, S. Ardabili, and A. R. Varkonyi-Koczy, "List of deep learning models", in *International Conference on Global Research and Education*, Springer, 2019, pp. 202–214.

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator", *arXiv preprint arXiv:1711.03938*, 2017.

[20] *Carla's sensors and cameras*. [Online]. Available: https://carla.readthedocs.io/en/0.9.6/cameras_and_sensors/#sensorlidarray_cast.

[21] M. K. Pal, N. Debabhuti, P. Sadhukhan, and P. Sharma, "A novel real-time collision avoidance system for on-road vehicles", in *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, IEEE, 2018, pp. 141–146.

[22] E. Mohammed, M. Abdou, S. A. Engineer, and O. A. Nasr, "End-to-end deep path planning and automatic emergency braking camera cocoon-based solution", in *Machine Learning for Autonomous Driving, NeurIPS 2019 Workshop*, 2019.

[23] 2019. [Online]. Available: https://devmesh.intel.com/projects/self-driving-cars-longitudinal-and-lateral-control-design.

[24] C. H. Zhao, B. L. Zhang, J. He, and J. Lian, "Recognition of driving postures by contourlet transform and random forests", *IET Intelligent Transport Systems, 6(2):161-168*, 2011.

[25] C. Zhao, Y. Gao, J. He, and J. Lian, "Recognition of driving postures by multiwavelet transform and multilayer perceptron classifier", *Engineering Applications of Artificial Intelligence, 25(8):1677â"1686*, 2012.

[26] C. Zhao, B. Zhang, J. Lian, J. He, T. Lin, and X. Zhang, "Classification of driving postures by support vector machines", *In Image and Graphics (ICIG), 2011 Sixth International Conference on, pages 926â"930. IEEE*, 2011.

[27] C. Yan, F. Coenen, and B. Zhang, "Driving posture recognition by convolutional neural networks", *IET Computer Vision, 10(2):103â"114*, 2016.

[28] H. Eraqi, Y. Abouelnaga, M. Saad, M. Moustafa, "Driver distraction identification with an ensemble of convolutional neural networks", *Journal of Advanced Transportation, Machine Learning in Transportation*, 2019.

[29] Duy Tran, Ha Manh Do, Weihua Sheng, He Bai, Girish Chowdhary, "Real-time detection of distracted driving based on deep learning", *IET Intelligent Transport Systems*, July 2018.

[30] Y. Abouelnaga, H. Eraqi, and M. Moustafa, *"Real-time Distracted Driver Posture Classification"*. Neural Information Processing Systems (NIPS 2018), Workshop on Machine Learning for Intelligent Transportation Systems, Dec. 2018.

[31] H. Eraqi, Y. Abouelnaga, M. Saad, M. Moustafa, *"Driver Distraction Identification with an Ensemble of Convolutional Neural Networks"*, Journal of Advanced Transportation, Machine Learning in Transportation (MLT) Issue, 2019.

[32] State Farm, *State Farm Distracted Driver Detection. Can computer vision spot distracted drivers? 2016.* [Online]. Available: `https://www.kaggle.com/c/state-farm-distracted-driver-detection/data/`.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep residual learning for image recognition", *arXiv:1512.03385v1 [cs.CV]*, 2015.

[34] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, *A neural attention model for speech command recognition*, 2018. eprint: `arXiv:1808.08929`.

[35] Saad, Nwishy, and Shabaan, *Ankh: A website for collecting an open source arabic speech dataset*, `https://ahmedsaad.pythonanywhere.com/`, 2020.

[36] Your Questions Answered channel. (Oct. 2016). 10 hours of people talking, [Online]. Available: `https://www.youtube.com/watch?v=PHBJNN-M_Mo`.

[37] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, *Generative adversarial networks*. [Online]. Available: `https://arxiv.org/pdf/1406.2661`.

[38] Alec Radford, Luke Metz, Soumith Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*. [Online]. Available: `https://arxiv.org/abs/1511.06434`.

[39] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville, *Improved training of wasserstein gans*. [Online]. Available: `https://arxiv.org/abs/1704.00028`.

[40] Ikaros Kun, *Python-youtube*. [Online]. Available: `https://pypi.org/project/python-youtube/`.

[41] Jonas Depoix, *Youtube-transcript-api*. [Online]. Available: `https://pypi.org/project/youtube-transcript-api/`.

[42] Nick Ficano, *Pytube*. [Online]. Available: `https://pypi.org/project/pytube/`.

[43] Karen Simonyan, Andrew Zisserman, *Very deep convolutional networks for large-scale image recognition*. [Online]. Available: `https://arxiv.org/abs/1409.1556`.

[44] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, *Densely connected convolutional networks*. [Online]. Available: `https://arxiv.org/abs/1608.06993`.

[45] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*. [Online]. Available: `https://arxiv.org/abs/1704.04861v1`.

[46] Matthijs Hollemans, *Mobilenets*. [Online]. Available: `https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/`.

[47] Sik-Ho Tsang, *Xception â" with depthwise separable convolution*. [Online]. Available: https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568.

[48] Barret Zoph , Vijay Vasudevan, Jonathon Shlens, Quoc V. Le, *Learning transferable architectures for scalable image recognition*. [Online]. Available: https://arxiv.org/pdf/1707.07012.

[49] Sergey Zagoruyko, Nikos Komodakis, *Wide residual networks*. [Online]. Available: https://arxiv.org/abs/1605.07146.

[50] Saining Xie, Ross Girshick, Piotr DollÃ¡r, Zhuowen Tu, Kaiming He, *Aggregated residual transformations for deep neural networks*. [Online]. Available: https://arxiv.org/abs/1611.05431.

[51] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, Jiashi Feng, *Dual path networks*. [Online]. Available: https://arxiv.org/abs/1707.01629.