Universitat Politecnica De Catalunya

Bachelors in Computer Science and Engineering

Estimating Urban Traffic Density Using Street Camera Images

Bachelor’s Thesis of Shreyas Chandrakaldharan

Director: Prof. Javier Bejar

Date: 21/06/2017
Abstract

Traffic is a very real problem in today’s world. Elon Musk, CEO of SpaceX and Tesla, has created a start-up which aims to bore tunnels to better manage traffic. While we do not dare to challenge him, we propose to better manage traffic on roads by leveraging currently available technologies in a novel way. Our project aims to give accurate real-time predictions of traffic, so that prospective commuters can choose routes that are free of traffic, thereby automatically balancing traffic. There are systems in place which give estimates of traffic but ours is cheaper, easier, and much more comprehensive.

Many existing systems solve the problem of estimation of traffic mathematically, by using positions of different cars, average speed of cars passing, etc. as statistics. We try to solve the problem as humans do. When we, humans, look at a lot of cars on a road we know it’s heavy traffic. Similarly, with recent advances in computer vision and machine learning, we can train computers to do the same.

We strive to develop a program that can look at an image of a street and automatically decide whether it is congested or not. We propose to implement a model that uses a convolutional neural network (CNN) to decide whether the given image is of a congested street or a free street. We further hope to deploy this model on a Raspberry Pi Mini Computer with an attached camera, which will be installed in the street corners. Thus, the Raspberry Pi will capture images of the street at brief intervals, use the CNN to estimate traffic density and then relay this information to a central server, where further processing can be done.
Acknowledgement

First and foremost, I would like to thank the Almighty for giving me this opportunity and guiding me always.

I am eternally grateful to my mentor Professor Javier Bejar for letting me work on this project and his kind thoughtful guidance throughout.

I would also like to thank the Barcelona Supercomputing Centre and Universitat Politecnica De Catalunya for allowing us to use the NVIDIA GPU Center of Excellence and giving us access to its resources and machines for developing this thesis

Lastly, I would like to thank my parents for their eternal love, support and confidence.
# Contents

Index of Figures ................................................................. vii
Index of Tables ................................................................. ix

1 Introduction .............................................................................. 1
   1.1 Why the Traffic Management Domain? ................................. 1
   1.2 Existing Solutions ................................................................ 2
       1.2.1 State of the art .......................................................... 2
       1.2.2 Related Work ............................................................ 3
   1.3 Problem Formulation ......................................................... 3
   1.4 Why Computer Vision and Machine Learning? ....................... 4
   1.5 Methodology ................................................................. 4

2 Scope, Stakeholders, and Challenges .......................................... 6
   2.1 Scope .............................................................................. 6
   2.2 Stakeholders .................................................................... 6
   2.3 Challenges ...................................................................... 7
       2.3.1 Dataset ................................................................. 7
       2.3.2 Computational Resources .......................................... 8
       2.3.3 Raspberry Pi .......................................................... 8

3 Project Management ................................................................. 9
   3.1 Initial Milestone ............................................................. 9
       3.1.1 Temporal Planning .................................................... 9
       3.1.2 Financial Planning ................................................... 15
       3.1.3 Sustainability ........................................................ 20
   3.2 Final Milestone ............................................................. 22
       3.2.1 Temporal Planning .................................................... 22
5.2.6 Optimizers............................................................................................................48

6 Experimental Results..................................................................................................49
  6.1 Procedure ..................................................................................................................49
  6.2 Alternative Approaches ............................................................................................59
    6.2.1 Individual Classifiers .......................................................................................59
    6.2.2 Regression Modelling ......................................................................................60

7 Deployment..................................................................................................................62

8 Conclusion & Future Work ............................................................................................64
  8.1 Conclusion ..................................................................................................................64
  8.2 Future Work ...............................................................................................................64
  8.3 Personal Conclusion ..................................................................................................65

9 References .....................................................................................................................67

Appendix A: Requirements ...............................................................................................69

Appendix B: Background Knowledge .............................................................................71
  Convolutional Neural Networks ....................................................................................71
  Layers of a Convolutional Neural Network .................................................................73
  Gradient Descent Algorithm .........................................................................................75
  Raspberry Pi ....................................................................................................................76
# Index of Figures

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Name</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Losses due to Traffic</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Gantt Chart of Schedule</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>Label distribution with 5-minute max. delay</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>Label distribution with 15-minute max. delay</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>Gradient of Red. Continuous vs Discrete</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>Histogram plot of entropies of images</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>Image with low entropy that is filtered</td>
<td>31</td>
</tr>
<tr>
<td>8</td>
<td>Image with low entropy that is filtered</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>Image with high entropy that is filtered</td>
<td>31</td>
</tr>
<tr>
<td>10</td>
<td>Blurry image that is not filtered</td>
<td>32</td>
</tr>
<tr>
<td>11</td>
<td>A Laplacian Kernel</td>
<td>32</td>
</tr>
<tr>
<td>12</td>
<td>Blurry image filtered by Laplacian Variance</td>
<td>33</td>
</tr>
<tr>
<td>13</td>
<td>Original Image</td>
<td>34</td>
</tr>
<tr>
<td>14</td>
<td>Grayscale Image</td>
<td>34</td>
</tr>
<tr>
<td>15</td>
<td>Label distribution after Horizontal Flipping</td>
<td>35</td>
</tr>
<tr>
<td>16</td>
<td>Original Image</td>
<td>35</td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>17</td>
<td>The mask used for the image in Figure 16</td>
<td>35</td>
</tr>
<tr>
<td>18</td>
<td>Image from Figure 16 after applying mask from Figure 17</td>
<td>36</td>
</tr>
<tr>
<td>19</td>
<td>Label distribution without Hour Ban</td>
<td>37</td>
</tr>
<tr>
<td>20</td>
<td>Label distribution with Hour Ban</td>
<td>37</td>
</tr>
<tr>
<td>21</td>
<td>Image at 0.25 Zoom</td>
<td>38</td>
</tr>
<tr>
<td>22</td>
<td>Image at 0.5 Zoom</td>
<td>38</td>
</tr>
<tr>
<td>23</td>
<td>Image at 1.0 Zoom</td>
<td>38</td>
</tr>
<tr>
<td>24</td>
<td>Flowchart of the Basic Model</td>
<td>40</td>
</tr>
<tr>
<td>25</td>
<td>The Inception V3 Architecture</td>
<td>42</td>
</tr>
<tr>
<td>26</td>
<td>A Shortcut connection</td>
<td>42</td>
</tr>
<tr>
<td>27</td>
<td>Residual Network Architecture</td>
<td>45</td>
</tr>
<tr>
<td>28</td>
<td>SqueezeNet Architecture</td>
<td>46</td>
</tr>
<tr>
<td>29</td>
<td>A screenshot of the webpage with the prediction</td>
<td>63</td>
</tr>
<tr>
<td>30</td>
<td>Raspberry Pi Internet Server and Client Model</td>
<td>64</td>
</tr>
<tr>
<td>31</td>
<td>Layers in a Neural Network</td>
<td>73</td>
</tr>
<tr>
<td>32</td>
<td>A convolution neural network</td>
<td>73</td>
</tr>
</tbody>
</table>
## Index of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Name</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Summary of Schedule</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Hardware Resources</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Human Resources Budget</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>A Task based distribution of the Resources</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>Total Budget</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>Revised Total Budget</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>Label distribution for different time delays</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>Label distribution after hour ban</td>
<td>36</td>
</tr>
<tr>
<td>9</td>
<td>Summary of the important results of our experiments with the basic model</td>
<td>55-56</td>
</tr>
<tr>
<td>10</td>
<td>Summary of the Best Models</td>
<td>58</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Why the Traffic Management Domain?

"Traffic is only one of the side effects of growth." - Roy Barnes

"In 2013 traffic congestion cost Americans $124 billion in direct and indirect losses, this number will rise to $186 billion in 2030." - Center for Economics and Business Research, 2014

Urbanization has had a lot of benefits. However, on the other side of the coin, it has also exponentially increased vehicular traffic in urban areas. These cause traffic jams, leading to air pollution, problems in personal health, loss of productive labor time, and economic losses. According to the World Resources Institute, Average travel speed in the center of Bangkok, during peak hours, is about 12 km/hr. To exacerbate this situation, 400 to 600 vehicles are added to its streets every day.

The following chart (Figure 1), taken from a World Resources Institute Report, illustrates the link between economic losses and traffic congestions in key South-East Asian cities.

![Figure 1: Losses due to Traffic (Source: World Resources Institute, World Resources 1996-97)]
Hence, it is vitally important and necessary to deal with traffic with the best of innovation and technology, keeping in mind the huge loss caused by traffic and the need to reduce the same for a better tomorrow. Quick and reliable real-time estimation of traffic density is crucial to reduce traffic as traffic density of a road is an excellent indicator of how suitable that road is for a prospective commuter.

1.2 Existing Solutions

1.2.1 State of the art

Today, most navigation software use traffic density as one of the parameters to assign routes for its users. To any new user who wishes to travel to a destination, it assigns the fastest route with the least amount of traffic.

In most cities, the current system in place, to estimate traffic density, uses sensors placed under roads or on the side of pavements which detect how many cars pass by it. Then this data is used to decide the amount of traffic in that street. While this mechanism may give good results, it is expensive to install and furthermore to repair in case of errors. [6][7]

Some systems, like Google Maps, also use GPS based location broadcasts. [11] They works as follows, every car broadcasts its location to a server using GPS. The server collects all the information and calculates how many cars are in the same area using distance measures. It then retransmits the calculations to every car, which will alert the driver about the traffic situation in that area. While this method can accurately describe the traffic situation in an area, it is computationally intensive and lacks the ability to predict future traffic situations.

Also, research is being done on using intelligent systems to predict traffic density. These systems, as opposed to statistical analysis, employ the use of artificial intelligence to ‘learn’ to detect traffic and predict traffic. They are trained with a lot of historical traffic data and are used to predict the nature of traffic for real time data. The standard practices in this method are using trained classifiers to decide which class a data point (In this case, a state of traffic)
belongs to. Some of the common classifiers used are Feed Forward Neural Networks, Hidden Markov Models, and Support Vector Machines.

1.2.2 Related Work

We reviewed highly seminal literature in the area of traffic density estimation and traffic management which gave us a good idea of the related work undertaken in this domain. They are listed below.

- Traditionally, road traffic density has been estimated using several techniques including roadside magnetic loop detectors, surveillance cameras, wireless vehicle sensors, and speed guns. [6] [7]
- In [2] Mao et al. dealt with the problem by making each vehicle estimate its local road traffic density in a road segment using some simple measurements, i.e. the number of neighbors.
- In [3]–[5], the authors presented methods to estimate vehicle density on a road segment based on the traffic data collected at the two wireless vehicle sensors placed at both ends of the road segment.
- In a novel way, Orzkurt et al. in [1] provided a method to use feed forward neural networks to analyze videos of traffic movement and predict the number of cars present.
- [8] highlighted an interesting method wherein a neural network classifier was trained with the audio samples of cars passing by and this was used to predict traffic density on three levels namely: low traffic, medium traffic, and heavy traffic.

We propose to give a similar result as [8] but by using visual input (images) instead of vocal input (audio) and with five classification levels instead of three namely: no traffic, low traffic, medium traffic, high traffic, and traffic jam.

1.3 Problem Formulation

The existing solutions fail to realize the suitability of convolutional neural networks to this problem. The most closely related work in this domain has considered only feed forward neural networks to solve this problem [8].
We are striving to better the current standard of traffic density estimation by using convolutional neural networks (CNNs) to facilitate the process in a more efficient manner. The use of convolutional neural networks in our problem domain is a pioneering effort, as discussed in our literature review.

1.4 Why Computer Vision and Machine Learning?

In our domain, trying to solve the problem mathematically, with sensors and statistics, will only take us so far. Making that solution of use in real life with respect to response time and scaling it to real world size would require huge computational and storage capabilities. Even if we were to use resources of such magnitude, the solution still will won’t be as elegant as one involving Computer Vision because Computer Vision is an emulation of the most advanced, the most powerful computer there exists: The Human Brain.

We try to solve the problem of determining whether there is traffic, just as simple as a human would do. By looking at an image and seeing whether there are lots of cars. If the answer is yes, then there is traffic, if not, then no. A computer can be trained to do the same with the help of Machine Learning techniques. We choose Convolutional Neural Networks (CNNs) as the Machine Learning technique to be used because CNNs, inherently by their design are similar to visual cortex of the brain and likewise more suited to deal with processing visual data such as images.

Also, our proposal would involve placing a mini computer (Raspberry Pi) with a camera at every street corner. This is a cheaper, less power consuming and a more efficient alternative to the current system of placing sensors under roads. Convolutional Neural Networks and Raspberry Pi are explained in more detail in the Appendix (10).

1.5 Methodology

This section gives a brief overview of the methodology to be followed in the project. It involves five major steps:

- Get past traffic density data and street camera images
• Refine the data into a dataset suitable for training a classifier
• Build a convolution neural network model classifier
• Tune the hyper-parameters of the model
• Deploy the final tuned model on a Raspberry Pi
Chapter 2

Scope, Stakeholders, and Challenges

2.1 Scope

The initial phase of the project will involve working with the dataset to test different mapping schemes of images and labels to decide on the most optimal scheme. This will be followed by pruning the dataset to remove unsuitable images to enhance the quality training data.

Second, the most optimum values for the parameters of the convolutional network model will be decided by running experiments on models with different parameters to study their impact on the result.

Finally, the fully trained model will be deployed in a Raspberry Pi as a web server that captures an image in real time, runs the network on the model, predicts the classification (Level of traffic) and sends it back to a central server, where further processing and logging can be done.

2.2 Stakeholders

This project has a very diverse and large target audience as elucidated below.

- It can be used by navigation software companies like Google Maps, Uber, etc. to give the fastest routes to its users.
- In turn, it benefits all commuters in the urban area by giving them the traffic status of a particular route.
- It can be used by city planners for data about the densest roads with respect to traffic for better city planning, traffic management and road construction.
• It can also be used by the police, hospitals, and other such agencies for tackling emergency situations by finding free roads which can be blocked. For example, in case an emergency green corridor route needs to set up, it can be done with the information generated by our project.

2.3 Challenges

2.3.1 Dataset

The data for this project is generated from street cameras in Barcelona via the official government server ‘http://bcn.cat/’ in two folds. First, there are 27 street cameras in Barcelona and they generate images at an interval of approximately 15 minutes. These images are the training images for our network. Second, for the training labels, we get the history of traffic density predictions from the same server. This data is also generated at an interval of 15 minutes each.

Mapping the data

Since the training images and the training labels are obtained separately and at different time periods, the first challenge is to map them correctly. We also realize that there is always a possibility of an error in the mapping as we can never be sure if we are mapping the image to its corresponding label. Even during manual verification of image-label pairs, in the cases where a camera points at two way roads, we cannot be sure of the direction which the labels correspond to.

Cleaning the data

Not all images obtained are good enough to be used for training

• Some images are blurry because of the low-quality cameras
• Some images are hazy because of fog
• Some images are fuzzy because of unsuitable lighting conditions, especially at night
Some images are covered by trees because of the improper alignment of the cameras.

So, a great challenge is to prune the dataset and remove the images that are not favorable for training.

**Getting more data**

Currently, we have the data for the month of November 2016, January 2017, and February 2017 as training data and two days of December 2016 as test data. However, this does not represent the entire sample set because of seasonal variations in different months. Getting the data for an entire year or a couple of years would be a very big challenge in terms of scale as training so many images even on a simple neural network would require very high computational power.

**2.3.2 Computational Resources**

The more complicated our convolutional network model is, the more layers it has and consequently the longer it takes to train. Training a simple 2-layer convolutional network for 30 days and testing for 2 days for 1 epoch takes approximately 30 minutes on a machine with Intel Core i7-3537 2 Ghz processor and 8 GB of RAM. To achieve good results, we would be required to train a more complex network for more than 100 epochs. Thus, getting enough computational resources to run the experiments fast is a big challenge.

**2.3.3 Raspberry Pi**

Our final model is planned to be deployed on Raspberry Pi. Since the Raspberry Pi is a minimized computer with less computational resources and functionalities, running the fully developed model with the capabilities of Raspberry Pi can prove to be a challenge.
Chapter 3

Project Management

3.1 Initial Milestone

Before beginning the project, we estimated the temporal and financial aspects of the project which include the schedule to be followed and the expected budget of the project. We present those estimates in this section. After completion of the project, we re-evaluated the temporal and financial aspects of the project to a finer precision and compared it to our initial estimates. The comparisons are presented in the next section (4.2 Final Milestone)

3.1.1 Temporal Planning

The project begins on the 9th of February. The hard deadline is on 19th of June, a week before the final defense (Expected to be on the 26th of June). Thus, a total of 18 weeks is at our disposal for the completion of the project. We will be undertaking the project by splitting it into different smaller tasks and executing the tasks in an ordered manner. The tasks have been ordered in a sequenced manner for execution. They have been described in detail in the following section.

3.1.1.1 Tasks

- Initial Preparatory Research

  This, being the first task, involves obtaining the background knowledge required to embark on the project. The main workers on this task will be the software designer and the project manager. It can be subdivided into three tasks as follows:
• Learning about Convolutional Neural Networks
• Learning about Keras
• Reading related Research Papers

**Time:** A duration of 2 weeks (50 hrs.) is both necessary and sufficient for this task.

**Resources:** Working Laptop with Internet connection, Web Browser, PDF Viewer

**Experiments**

**Refining the training data**

One of the primary tasks of the project. This involves working with the dataset to find out the most optimal organization of the training data. The main refinements to consider are:

- Pruning the unusable images
- Mapping images to their corresponding predictions
- Estimating the best zoom factor

**Deciding the model architecture**

The most fundamental task of the project. The finer details of our convolutional neural network will be decided in this task. Some of the important parameters which will be fixed in this step are:

- Number of Classes
- Weights given to each Class
- Learning Rate
- Batch Size
- Number of Layers
- Size of each Layer
- Classification Approach/ Regression Approach

- **Simulation**

  This task can be divided into three parts:

  - Running trial simulation based on current model with the current dataset
  - Analyzing the results
  - Based on the results, we go back to Task 2.A to further refine the dataset for better accuracy. Then execute Task 2.B again with to update the parameters based on the obtained results. Then run Task 2.C, a new simulation.

  Thus, Tasks 2.A, 2.B and 2.C form a continuous cyclic process of experimenting with the goal of bettering the accuracy of the model as much as possible. All three subtasks require an equal distribution of time and resources allocated for the collective task. Also, all these sub-tasks would require the major involvement of the project manager, software designer and software developer.

  **Time:** Thus, this task of Experiments is the major and most important part of this project and will require the most amount of time. Hence, a total time of 10 weeks (250hrs) has been allocated for this task, which is approximately 50% of the total duration of the project.

  **Resources:** Python IDE, Working Laptop with Internet connection

- **Implementing on a Raspberry Pi**

  This task deals with the actual implementation of the project on a Raspberry Pi computer. It is the deployment of the code on the Raspberry Pi. The software developer is the main worker on this task. It is supervised by the project manager. It can be subdivided into:
• Deploying the classifier model built with Keras
• Developing a web server on the Pi to retrieve the result and transmit it to a central server

**Time:** Deployment, which involves a small chunk of the coding part, is allocated a period of 2 weeks (50 hrs.). Roughly, 10% of the project’s total duration.

**Resources:** Raspberry Pi 2, Camera for Raspberry Pi, Working Laptop with Internet connection, Ethernet cable for Raspberry Pi/ Wi-Fi Adapter

• **Testing**

One of the final stages of the project. This task will involve conducting a thorough test of the entire project. The main objectives are testing the classifier and testing the Raspberry Pi server. It will encompass testing the sub-components too. The software tester is responsible for this task throughout under the supervision of the project manager.

**Time:** A 2-week (50 hrs.) period is both necessary and sufficient for running various tests that completely verify all the components of the project. Thus, this task has been scheduled to run for 2 weeks (50 hrs.).

**Resources:** Python IDE, Working Laptop with Internet connection

• **Documentation**

Last but not the least part of the project. The final phase of the project where the thesis is written. As, the thesis requires a lot of time to write correctly and thoroughly, a dedicated time of two weeks has been allocated for the same. Within this period, the presentation for the final defense is also scheduled to be prepared. The project manager takes care of this task, with inputs from the software designer, software developer and software tester.
Time: Again, this task has been allocated 2 weeks (50 hrs.) for its completion as it involves writing the thesis and an additional task of preparing the presentation for the defense.

Resources: Microsoft Word, Working Laptop with Internet connection

- Project Management Course

The Project Management Course (GEP) is an important task that will run in parallel with the project till it’s completion (April 3rd). There is a total of 6 deliverables as part of this course, which aim to teach the basics of project management. It helps us plan our project and has the added benefit of ensuring a good plan for our project. Thus, it is vital to the timely and successful completion of the project.

Time: 75h (37.5 hrs. of Guided Learning + 37.5 hrs. of Self Study)

Resources: Web Browser, PDF Viewer, Microsoft Word, Microsoft Excel

Summary

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Start Date</th>
<th>End Date</th>
<th>Duration (Weeks)</th>
<th>Duration (Hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Preparatory Research</td>
<td>Feb 9, 17</td>
<td>Feb 23, 17</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>Experiments</td>
<td>Feb 24, 17</td>
<td>May 5, 17</td>
<td>10</td>
<td>250</td>
</tr>
<tr>
<td>Implementing on a Raspberry Pi</td>
<td>May 6, 17</td>
<td>May 20, 17</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>Testing</td>
<td>May 21, 17</td>
<td>Jun 4, 17</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>Documentation</td>
<td>Jun 5, 17</td>
<td>Jun 19, 17</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>Project Management Course</td>
<td>Feb 21, 17</td>
<td>Apr 3, 17</td>
<td>6</td>
<td>75</td>
</tr>
</tbody>
</table>

Table 1: Summary of Schedule

3.1.1.2 Action Plan

We propose to execute the project as per our plan. Table 1 and Figure 2 give the tabular and Gantt Chart representation of the schedule respectively. We will finish the tasks one by one in
the listed order. The given order is logical and sequenced in such a way that there will no scenario wherein a task has to be started without its preceding requirements’ completion.

![Gantt Chart](image)

**Figure 2: Gantt Chart of Schedule**

Also, we have distributed our available period of 18 weeks among the different tasks with a reasonable estimate, accounting for 30h of work per week for a total of 540h for the project. Optimistically, this is a definite and attainable schedule, as delays due to holidays and other unforeseen circumstances leading to losses in the number of hours per week can be compensated by putting in more hours in the following weeks. Thus, in extreme cases, most likely a workload of 40h/week can be present. However, since this amount is also an easily surmountable figure, we hope to complete the project in time.

We also realize that as in the case of any other projects, there might be unexpected obstacles during each task, especially beginning from the Experiments phase. Thus, if we run out of time during any task, we will go forward with the best version we have then. Of course, there will be priorities that will be met such as more than 50% accuracy of the classifier and working implementation in the Raspberry Pi. In case, pessimistically, we are faced with a severe need for more time to deploy the project, we can cut down on the extent of the Experiments phase. Planned weekly meetings with the director of this project will further help us to track, expedite and follow our proposed plan to ensure the deadlines are met.
3.1.2 Financial Planning

The budget of the project has been estimated in the following section. We have divided the total budget into three main categories of Hardware, Software, and Human Resources.

3.1.2.1 Hardware Costs

Amortization has been using the formula:

\[ \text{Amortized Cost} = \text{Actual Cost} \times \left( \frac{\text{No of Years in Use}}{\text{Useful Years}} \right) \]

No of Years in Use is the length of the project in years, which is, 0.42 years (5 months)

Table 2 (Fig. Below) shows the required calculation for the Hardware Costs, which amounts to a total of **99.06 Euros**

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>Useful Years</th>
<th>Amortization (in Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>715</td>
<td>4</td>
<td>74.47</td>
</tr>
<tr>
<td>Mouse</td>
<td>4</td>
<td>1</td>
<td>1.67</td>
</tr>
<tr>
<td>Raspberry Pi 2</td>
<td>70</td>
<td>2</td>
<td>14.58</td>
</tr>
<tr>
<td>Raspberry Pi Camera</td>
<td>40</td>
<td>2</td>
<td>8.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Grand Total</strong> 99.06</td>
</tr>
</tbody>
</table>

*Table 2: Hardware Resources*

3.1.2.2 Software Costs

All the software we need and will use for the project are available free of cost. They are listed below.

- Ubuntu 16.10
- PyCharm Community Edition
- Google Chrome
- Sublime Text Editor
3.1.2.3 Human Resource Costs

<table>
<thead>
<tr>
<th>Role</th>
<th>Euros/Hour</th>
<th>Hours</th>
<th>Total (in Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>50.00</td>
<td>135</td>
<td>6750.00</td>
</tr>
<tr>
<td>Software Designer</td>
<td>35.00</td>
<td>195</td>
<td>6825.00</td>
</tr>
<tr>
<td>Software Programmer</td>
<td>25.00</td>
<td>100</td>
<td>2500.00</td>
</tr>
<tr>
<td>Software Tester</td>
<td>20.00</td>
<td>110</td>
<td>2200.00</td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td></td>
<td></td>
<td><strong>18275.00</strong></td>
</tr>
</tbody>
</table>

Table 3: Human Resources Budget

Table 3 gives a detailed distribution of the Human Resources budget, which is a total of 18,275 Euros.

The human resources budget has been estimated by considering the different roles required to complete the project. We wish to bring to notice that there is only one person working on the project and he will adopt different roles as and when required. They are:

- **Project Manager**

  He oversees the entire project planning. He must work during all tasks and phases of the project ensuring the project is proceeding as per plan.

- **Software Designer**

  He is responsible for the design of the program we are going to develop. He will have a major portion of his work during the Experiments phase, when he should redesign the project model according to the results of the experiments.
- **Software Programmer**

  He is the actual coder. He writes code for the software designed by the Software Designer. Mainly, he must work during the Experiments phase alongside the Software Designer.

- **Software Tester**

  As the name suggests, he takes care of testing the working of the entire project, detecting bugs, and reporting it to the Project Manager. He will be continuously involved in the initial stages. However, he will have a dedicated period as the final phase for thorough testing of the developed project.

### 3.1.2.4 Indirect & Unforeseen Costs

In this section, we have considered costs incurred from other categories than Hardware, Software, or Human Resources. It considers indirect costs and unforeseen costs. This accounts for possible deviations in the course of the project.

Under Indirect Costs, we have Internet. We use eduroam Wi-Fi for the project which is free. Thus, we have not considered expenses due to Internet. However, in case we are unable to use eduroam, then we must spend money for Internet. This is considered under unforeseen costs.

Also, we need not consider office space as an expense, as the physical material constituting office space will be intact even after our project completion for further use. Electricity consumption while working in the university is zero as the expense is borne by the university. Since we do not plan to work outside the university, this will not affect our budget as well.

The reason for allocating a sum of money to unforeseen costs is to prepare for contingencies. We hope the project will proceed per plan, but there might be sudden and unforeseen changes in the plan. We should be prepared to handle these as well.
Some situations that might crop up are:

- Human resource estimation might exceed our budget. For example, if during the testing phase, an error is detected and some part of the code must be modified, we should use the Software Designer and Software Programmer during the testing phase also which we have not accounted for in the calculation. Such instances will come under unforeseen costs.
- We don’t expect to use any more hardware than we have listed but there might be two cases in which we might have to.
  - Any of our hardware gets corrupt or needs replacement
  - Our laptop isn’t computationally sufficient to train our network. In that case, we hope to use the supercomputing cluster present at Barcelona Supercomputing Center (BSC), which is free of cost to academicians. However, if we use the supercomputer at BSC, then we should include it in the budget for a more precise estimate, as it is not free for all. To get an estimate of its cost, we can compare it with the commercially available computing power from Amazon EC2 which will cost 300 euros for our requirement, which is 300hrs of computation.
- Finally, during the deployment phase of the project, to install the camera and Raspberry Pi on the street, we might need additional equipment or license. This is also considered under unforeseen costs.

Taking into account, all of the above considerations, we propose to allocate 1000 euros (5% of all the other costs put together) for unforeseen expenses under Other Expenses.

### 3.1.2.5 Task-based distribution

A task-based view of the different facets of our budget has been presented in Table 4
### Table 4: A Task based distribution of the Resources

<table>
<thead>
<tr>
<th></th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task 4</th>
<th>Task 5</th>
<th>Grand Total (in Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware Costs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mouse</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>Raspberry Pi 2</td>
<td>0.00</td>
<td>0.00</td>
<td>7.29</td>
<td>7.29</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>Raspberry Pi Camera</td>
<td>0.00</td>
<td>0.00</td>
<td>4.17</td>
<td>4.17</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>Supercomputer</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td><strong>Total Hardware Costs</strong></td>
<td>15.23</td>
<td>15.23</td>
<td>26.69</td>
<td>26.69</td>
<td>15.23</td>
<td>99.06</td>
</tr>
<tr>
<td><strong>Human Resource Costs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Manager</td>
<td>30.00</td>
<td>60.00</td>
<td>15.00</td>
<td>10.00</td>
<td>20.00</td>
<td></td>
</tr>
<tr>
<td>Software Designer</td>
<td>30.00</td>
<td>120.00</td>
<td>25.00</td>
<td>0.00</td>
<td>20.00</td>
<td></td>
</tr>
<tr>
<td>Software Programmer</td>
<td>0.00</td>
<td>80.00</td>
<td>10.00</td>
<td>0.00</td>
<td>10.00</td>
<td></td>
</tr>
<tr>
<td>Software Tester</td>
<td>0.00</td>
<td>40.00</td>
<td>10.00</td>
<td>50.00</td>
<td>10.00</td>
<td></td>
</tr>
<tr>
<td><strong>Total Human Resource Costs</strong></td>
<td>2550.00</td>
<td>10000.00</td>
<td>2075.00</td>
<td>1500.00</td>
<td>2150.00</td>
<td>18275.00</td>
</tr>
<tr>
<td><strong>Total Costs</strong></td>
<td>2565.23</td>
<td>10015.23</td>
<td>2101.69</td>
<td>1526.69</td>
<td>2165.23</td>
<td>18374.06</td>
</tr>
</tbody>
</table>

### 3.1.2.6 Budget Control

Seeing, that we have accounted for almost all possible deviations for the budget in the project, we have a strong belief the project will not exceed the proposed budget. Our consideration of unforeseen costs and possible deviations is an extensive one and is a robust budget control measure. Thus, it is an indicator that the proposed budget is definitely an upper limit on the actual budget. All considerations in case the project requires more resources, including time, have been dealt with under unforeseen costs. In case, the project finishes earlier than expected or requires only fewer resources, then we will have a lower actual budget.
### 3.1.2.7 Total Budget

<table>
<thead>
<tr>
<th></th>
<th>Cost (in Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>99.06</td>
</tr>
<tr>
<td>Software</td>
<td>0.00</td>
</tr>
<tr>
<td>Human Resources</td>
<td>18275.00</td>
</tr>
<tr>
<td>Contingencies (Unforeseen Costs)</td>
<td>1000.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>19374.06</strong></td>
</tr>
</tbody>
</table>

*Table 5: Total Budget*

Thus, as apparent from the Table 5 (Fig. Above), the total estimated budget for this project is **19,374.06 Euros**.

### 3.1.3 Sustainability

We analyze the sustainability of the project under three main categories: Economic, Social and Environmental Sustainability. They have been described in detail, respectively in the following sections.

#### 3.1.3.1 Economic Sustainability

Our project is economically a very sustainable one, as apparent from the budget we have proposed. We have assessed all costs (material and human) that will be incurred during the project. We also have accounted for indirect and unforeseen costs, that may have escaped our initial estimate.

All the software used in this project is free and there is a very minimal requirement of hardware as well. With respect to human resources, we have a reasonable and viable estimate, which can exceed only in highly unrealistic circumstances. It is also impossible to reuse code of any kind, as our solution is a novel one and there exists no code that does what we hope to do. It cannot be done using fewer resources of any kind. Thus, we are confident that this is economically the most efficient budget possible for this project.

Next, the project is a cheaper solution that the existing methods for tackling the same problem, which involves placing sensors under the road to estimate traffic density.
Maintenance and repair of our implementation, a camera placed in the corner of a street, will much cheaper than repairing sensors under the road. Also, updating the software periodically will also prove to be a necessity in the future, however it will not have much effect economically as it just involves just software changes using existing hardware and manpower.

Finally, the potential return of investment of this project is huge as it is a novel solution in the traffic management domain, which is a crucial area today. Thus, there is a huge scope for a large-scale implementation of this project with possible government or corporate investments. We think this project qualifies as an 8, on a scale of 1 to 10, in the Economic Sustainability analysis.

3.1.3.2 Social Sustainability

Social sustainability is where our project scores the highest because it has a huge positive impact on society. This project does not have any regional, political or social barriers to its successful implementation in various places because it deals with a problem that is common to all, which is Traffic. It improves the solution of such a crucial problem making everybody’s lives better, hence it is extremely necessary.

Also, it offers a solution that is much less of a hassle than the existing solutions. Installation, maintenance and repair of our model, essentially a street camera, will be easier than the current model of placing sensors under the road, which can lead to disruption of daily traffic while installation or maintenance, thus angering citizens.

Considering the usefulness and efficiency of our project as a societal contributor, we award it a 9, on a scale of 1 to 10, in the Social Sustainability analysis.

3.1.3.3 Environmental Sustainability

Under environmental sustainability, we analyze the project in two different phases: During the development and After the deployment.

During the development, the only environmental concern for the project is the consumption of electricity. The main contributor for electricity consumption is our Laptop. However, since laptops have become a necessity in today’s world and doing a project without a laptop is next
to impossible. We chose to ignore the electricity factor. Even we had considered it, we require 540 hrs. for the project. Assume our laptop consumes 200 W while working, which gives us a total of 108 KWH, which is equivalent to 41.58 kg of CO$_2$, which is well within the permissible amount for any project. A positive feature of the project is that the project is almost entirely paper-free. We only use computers for analysis, design, development and implementation.

After the deployment, we have to consider the fact that the street cameras we placed are in the open environment. While there will be no toxic emissions, they will consume a constant amount of electricity to work. The cameras will have to work at regular intervals of time. The consumption will be analogous to the electricity consumption of traffic lights and is deemed as necessary for the project. The equipment might also malfunction due to environmental factors such as smoke from cars, rain, snow, extremely low temperatures, etc. Then they would need to be repaired or replaced depending on the extent of the damage.

Considering all the points mentioned in the above discussion, we grade the Environmental Sustainability of this project as 8, on a scale of 1 to 10.

### 3.2 Final Milestone

After the completion of the project, we compared the initial milestone estimates with the actual temporal and financial outcomes. The main goal of the project and the methodology to be followed did not change from the inception of the project throughout its course. Thus, there are no major changes to temporal and financial aspects of the project as well. The comparisons are presented in the following sections.

#### 3.2.1 Temporal Planning

The initial schedule was an accurate estimate of the timeline of the project. All tasks roughly took the same amount of time as expected.

Some minor changes were observed in the first task of Initial Preparatory Research, which took longer than the allocated 2 weeks. It took a week extra, owing to the fact that the project
developer needed more time to understand the technologies used in the project and familiarize himself with the technical environment, apart from gaining background knowledge on the related subjects.

However, in a compensatory manner, the task of Implementing the code in a Raspberry Pi took a week lesser than the initial estimate of 2 weeks. With the thorough understanding of the project, obtained from the extensive phase of running experiments, the task of implementation was easier than expected.

Thus, the overall schedule and deadlines remain unaffected at the final milestone.

### 3.2.2 Financial Planning

With respect to the budget of the project, all expected expenses were accurate. No extra resources in terms of hardware, software or time were necessary for the completion of the project. Thus, there were no deviations from the initial estimate of the budget.

Particularly, we needed more hardware resources because our laptop did not have enough computational capacity to run the experiments in a timely and efficient manner. So, we used the servers in Barcelona Supercomputing Center to remotely run our experiments and get the results in less time. Since, we had already considered this expense under Contingencies (Table 5), for which we had allocated 1000 euros, renting the extra computing resource did not deviate the budget.

To be precise, we had allocated an amount that was more than required for Contingencies. The only contingency that had to be handled was the rent of the extra hardware resource, which costs only 300 euros (as we had calculated in our initial estimate in Section 4.1.2.4). Thus, actually the budget was overestimated by 700 euros. So, the actual budget of the project is only **18,674.06 Euros**. The updated budget is reflected in Table 6.
<table>
<thead>
<tr>
<th>Cost (in Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
</tr>
<tr>
<td>Software</td>
</tr>
<tr>
<td>Human Resources</td>
</tr>
<tr>
<td>Contingencies (Unforeseen Costs)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
</tr>
</tbody>
</table>

*Table 6: Revised Total Budget*

### 3.2.3 Sustainability

The sustainability factor of the project remains the same across economic, social, and environmental dimensions as the project proceeded according to the plan without any major deviations in terms of time, money, effort, or the result. The deliverables that were promised were given within the expected temporal, financial, social, and environmental boundaries. Thus, there are no changes in the sustainability of the project.

### 3.2.4 Legal Implications

The images we are using are from city street cameras and they are publicly available for free. Also, since the images are not clear enough to recognize any faces or car plate numbers, there are no privacy concerns as well. Thus, the project does not infringe any law or regulation.
Chapter 4

About the Data

In this section, we shall discuss the data and various preprocessing techniques used to refine the data.

4.1 Obtaining the Data

The data we work with is a raw set of images and traffic density values obtained from http://www.bcn.cat/transit/.

Images are captured by 27 cameras which are present in different locations throughout the city of Barcelona. Each camera captures an image every 15 mins. The images are grouped into folders with each folder having the images for one day. Every image label has the time of Image Capture and the Name of the Camera. (Note: The name of the camera corresponds to the area it is placed in)

We also have the set of estimated traffic density values of all streets in Barcelona for every 15 minutes.
There is one traffic density value data file for each day. Each line in that file is the traffic density value of a street at a time. Every traffic density value has the Street Number, the time of estimation, the estimated current traffic density, and the predicted traffic density in 15 mins.

The traffic density is classified in 5 levels from 1 to 5, where 1 stands for no traffic, 2 stands for low traffic, 3 stands for medium traffic, 5 stands for high traffic and 5 stands for very high traffic.

We have the images and traffic data from the month of November 2016 till April 2017
4.2 Creating the Dataset

To create the training dataset, we must map every image to its corresponding traffic density value.

To help us with this, we have, for each camera, the information of the two closest streets to it. Using this, for every image, we find the traffic densities of the two closest streets to the camera which captured the image, at the time which is closest to the time of image capture. From these two values, we assume the higher value will be the correct traffic density and map the image to that value.

Thus, we have a training dataset of image-value pairs where each value, an integer from 1 to 5, tells the amount of traffic in the corresponding image.

4.3 Refining the Dataset

4.3.1 Imposing a Maximum Delay

For ensuring a good quality in the training dataset, we impose a delay criterion by which we only consider image-value pairs which have a time difference (The difference between the time of image capture and the time of traffic density estimation) less than the specified delay.

The number of image-value pairs we generate for the training data depends on the maximum delay that we specify. Intuitively, with a larger delay we generate more number of image-value pairs.

The following table summarizes the differences in the training data on having a maximum delay of 15 mins and a maximum delay of 5 mins.
Since the amount of traffic on roads is high only during few times throughout the day, the
distribution of training data available for different classes (0-4) is skewed. We have a lot of
examples for low classification levels (0 & 1) whereas very few examples for higher
classification levels (2,3 & 4).

The following figures (Figure 3 & Figure 4) highlight the distribution of training data across
different classes.

<table>
<thead>
<tr>
<th>Max Delay\Labels</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 minutes</td>
<td>28,546</td>
<td>47,309</td>
<td>7,825</td>
<td>3,310</td>
<td>1,119</td>
</tr>
<tr>
<td>15 minutes</td>
<td>89,768</td>
<td>144,635</td>
<td>23,970</td>
<td>10,481</td>
<td>3,517</td>
</tr>
</tbody>
</table>

Table 7: Label distribution for different time delays. The figures are for the entire data
available from November 2016 to April 2017

Since the amount of traffic on roads is high only during few times throughout the day, the
distribution of training data available for different classes (0-4) is skewed. We have a lot of
examples for low classification levels (0 & 1) whereas very few examples for higher
classification levels (2,3 & 4).

The following figures (Figure 3 & Figure 4) highlight the distribution of training data across
different classes.

**Figure 3: With 5-minute max. delay**

**Figure 4: With 15-minute max. delay**

Though 15-minute delay generates more image-value pairs, we prefer to use 5-minute delay
because with 15-minute delay, an image captured at 23:00 and traffic status for that street at
23:15 will also be mapped, however such a mapping is not favorable as the label will most
likely be very wrong. 15 minutes is enough for a congested street to become completely free.
Thus, ‘1’ labelled images might actually belong to label ‘4’. Due to the huge volume of images,
it is impossible to manually check the validity of each label for each image. With 5-minute
delay, there is a less chance of wrong labelling and also training is much faster as there are
fewer examples. We tested both 5-minute delay and 15-minute delay and 15-minute delay did
not give any improvement over 5-minute delay mapping. Thus, we prefer to use a smaller training set with a high probability correct labels rather than a bigger training set with lesser probability of correct labels. From here on, all the figures and statistics assume 5-minute maximum delay mapping.

From a statistical standpoint, to give a measure of the incorrectness of mapping even in the 5-minute delay method, we manually counted the number of images that were labelled as ‘0’, which meant the image must not have any amount of traffic but had some amount of traffic. Out of a random set of 50 images, 10 images were incorrectly classified as ‘0’. On extrapolating we can infer that there is approximately a 20% error in the mapping of the images and labels. However, as we had previously stated in Challenges (Section 2.3.1), even during manual verification, in cases of two way roads, we cannot be sure of the direction which the label corresponds to. During our manual verification, we assumed that the label ‘0’ would correspond to the direction which had the least amount of traffic.

4.3.2 Data Pruning

Among the Images we have, not all images obtained are good enough to be used for training

- Some images are blurry because of the low-quality cameras
- Some images are hazy because of fog
- Some images are fuzzy because of unsuitable lighting conditions, especially at night
- Some images are covered by trees because of the improper alignment of the cameras

So, we must prune such low-quality images to get a high-quality dataset.

We tried two different methods to prune such images. Both are discussed below.

4.3.2.1 RGB Entropy

The main motivation behind this method was to detect and remove images that have a lot of similar colored pixels as this is a good indication that the image is covered by trees, too dark,
sun flares or some other problem with coloring in the camera. Thus, these images would be unfitting for training.

Images covered with trees have a lot of green colored pixels. Similarly, dark images have a lot of black colored pixels. This method generalizes that property and detects any image which has a lot of same colored pixels. Images with a lot of similar pixels have a low entropy. In Information Theory, Entropy is the amount of information contained in a message. Entropy of an image is the amount of information needed to encode that image. In order to calculate the entropy of an image only with the pixel values, we use RGB Entropy. RGB Entropy is the process of calculating the Entropy of an image with the RGB values of its pixels.

First, we need to discretize the continuous values that pixels can take. For example, a fully red pixel will have the value (255,0,0) as its RGB component. Assuming we want to discretize the pixels into finite points. We will first normalize the RGB components to a scale of 0 to 1 and then multiply it by the number of points we require and round it off to nearest integer. Thus (255,0,0) will be normalized to (1,0,0) and then assuming we want 10*10*10 points i.e. 10 values for each color, we multiply it with (10,10,10) for it to become (10,0,0). The point (254,0,0), which is also a red pixel, will be normalized to (0.99,0,0), and then multiplied with (10,10,10) to become (9.9,0,0) which when rounded off will again become (10,0,0). So, the total number of pixels mapped to the point (10,0,0) is 2. As you can observe, similar colored points will be mapped to the same 3D point via this method.

Figure 5 gives a good idea about how a continuous gradient of 0 - 255 is discretized into 10 intervals. Each interval represents one point i.e. one value.

We must keep count of how many pixels map to each point. Then we calculate the probability of each point, which is the number of pixels in that point divided by the total number of pixels in the image.

\[
p(i,j,k) = \frac{\text{No. of pixels in } (i,j,k)}{\text{Total Number of pixels in image}}
\]
With the probabilities calculated, we can calculate the entropy of the image with the below formula

\[ E = - \sum p(i) \log p(i) \]

The resulting entropy of each point represents the amount of information needed to encode that point. We plot a histogram between each point and the entropy of that point, to find a separation of low entropy points and high entropy points. We note down the value of entropy at the separation. Later, we use this entropy value as a cutoff criterion to filter only images of good quality. In Figure 6 we have given the histogram plot of the images of 1st December 2016. The red line indicates the cutoff entropy value on the y-axis (2.8, in this case).

![Figure 6: Histogram plot of entropies of images from 01/12/2016. Entropy cut-off value: 3.0 (Red Line)](image)

Figures 7,8 and 9 are some of the images filtered by this cut-off entropy value.
Figure 7: Image with low entropy that is filtered

Figure 8: Image with low entropy that is filtered

Figure 9: Image with low entropy that is filtered wrongly

Figure 7 & Figure 8 have been correctly filtered by this method. Figure 7 has a low entropy because of a lot of similar red colored pixels. Figure 8 has a low entropy because of a lot of green colored pixels. However, Figure 9 has been erroneously filtered even though is clear and high-quality image fit for training, which leads us to the main disadvantage of this method. This filter has a high chance of wrongly filtering out images with empty roads as those images are bound to have a lot of pixels with the same color intensity. Simply put, the pixels representing the empty road will all be similar and thus might be pruned by mistake.

Figure 10: Blurry image that is not filtered

Another disadvantage of this method is that it fails to filter images that are blurry in nature because it does not consider the sharpness of an image in its calculation of entropy. Blurry images have low sharpness. For example, the Figure 10 is an example of a blurry image which has
an entropy value of more than 2.8 and thus will not be filtered by RGB Entropy. To overcome this, we used a new method called Laplacian Variance which is discussed in the next section.

4.3.2.2 Laplacian Variance

The Laplacian Variance method is used to detect blurry images. It is based on the principle of the Laplacian Operator, which is used to measure the second derivative of an image. The Laplacian Operator highlights rapid intensity changes in an image. Thus, it is used mainly for Edge Detection. To detect blurry images, we can assume that sharp images have high variance i.e. a lot of edge-like and non-edge like responses. But, blurry images have very less edges and thus have a low variance in response to the Laplacian operator. Thus, if the variance falls below a pre-defined threshold, then the image is considered blurry; otherwise, the image is not blurry.

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}
\]

*Figure 11: A Laplacian Kernel*

The Laplacian operator is described in Figure 11. To get the response of the image to this operator, the kernel is convolved over the entire image. After that, the variance of the resulting response is calculated. Mathematically, variance is the average squared deviation from the mean of the data. Then the image is filtered based on the pre-defined threshold.

The Laplacian variance method can successfully filter out blurry images that RGB Entropy fails to filter, such as Figure 12. Also, this method will wrongly allow images that are covered by trees but are sharp in nature. However, that is not a major concern, as we can deal with images that are covered by trees by simply neglecting those particular cameras. Since, trees are a constant obstruction all images taken by those cameras are bound to be covered by trees and can be safely neglected.
Thus, we use Laplacian Variance to prune our image set, rather than RGB Entropy, as the latter wrongly filters images of empty roads which are important to train the classifier. Also, we handle images that are covered by trees by neglecting the cameras altogether. The ideal Laplacian Variance threshold was manually determined to be 0.025. At 1.0 zoom, this method eliminated 77,596 images from 400,000, which is roughly 1% of all the images. However, at 0.25 zoom, since all the images are already blurry, this method does not eliminate a lot of images. At 0.25 zoom, from the raw set of 400,000 images, this method eliminated 747 images, which is 0.0018%

4.4 Experiments with the Dataset

4.4.1 Grayscale

We experimented with the data by converting all images to grayscale. The motivation behind this was to check whether abstraction of color leads to better accuracy of the model as sometimes many colors in an image can confuse the classifier, especially colors due to buildings, traffic lamps and street lights.

The problem with conversion to grayscale is that there are numerous methods of conversions and it is impractical to try everything. However, we tried one such conversion, the default one available. It converts an RGB image into a grayscale image using the following formula:

\[ L = R \times \frac{299}{1000} + G \times \frac{587}{1000} + B \times \frac{114}{1000} \]

Where L is the luminosity i.e. intensity of the resulting pixel.
The conversion is illustrated in Figure 13 and Figure 14:

![Figure 13: Original Image](image1) ![Figure 14: Grayscale Image](image2)

### 4.4.2 Horizontal Flipping

Horizontal Flipping is a Data Augmentation method that we experimented with. It was motivated by the need to balance the label distribution in the dataset. We had a highly disproportionate number of examples for all the labels, for example, 54% of our training data was labelled ‘2’. Labels ‘1’ & ‘2’ collectively account for 86% of the examples.

To reduce this disparity, we doubled the training data for the Labels ‘3’, ‘4’ and ‘5’ by creating a copy of each of the examples for these labels in the training data. After doubling the examples for the three labels, labels ‘1’ & ‘2’ accounted for 75% of the data, which was still high in disparity, but better than the initial distribution. The distribution after Horizontal Flipping is better explained by the Figure 15.

### 4.4.3 Masks

In the captured images, the only useful information for the classifier is the road and the cars on it. The surrounding imagery in the form of buildings, street lights, sky, pedestrians, and mainly parked vehicles are potential distractions for the classifier. A better accuracy can be achieved if we keep the classifier focused on only the important information namely the road and the cars. Hence, we decided to mask all the images such that only the road and the cars were visible. All the other portions of the image were blacked out.
The masking has been illustrated in Figures 16, 17 and 18.

**Figure 15: Distribution after Horizontal Flipping**

**Figure 16: Original Image**

**Figure 17: The mask used for Figure 16**

**Figure 18: Image from Figure 16 after applying mask from Figure 17**
As you can observe, the mask successfully eliminates all the redundant information from the image. Also, because each camera is aligned differently and thus captures the road at different angles and different zooms, separate masks had to be developed for each camera. So, we created a total of 27 masks, one for each camera and used it to mask all the images to be given as input.

### 4.4.4 Hour Ban

Hour Ban was another method we used for balancing the label distribution in the training dataset. Images are captured throughout the day, and during the night, traffic is mostly not present. Thus, nighttime generates the most number of ‘1’ and ‘2’ labels. We used this inference to prune all images that were obtained from 11 pm to 7 am. The resulting dataset had a different label distribution. From Table 8, we can see that the number of ‘1’ labels decreased significantly, by about 75%. The other labels were relatively less affected. However, the domination of ‘2’ labels increased further. The comparison illustrated in Figures 19 and 20 showcase this clearly.

<table>
<thead>
<tr>
<th>Max Delay\Labels</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Hour Ban</td>
<td>28,546</td>
<td>47,309</td>
<td>7,825</td>
<td>3,310</td>
<td>1,119</td>
</tr>
<tr>
<td>With Hour Ban</td>
<td>7,964</td>
<td>43,135</td>
<td>7,136</td>
<td>2,973</td>
<td>987</td>
</tr>
</tbody>
</table>

*Table 8: Label distribution after hour ban. The figures are for the entire data available from November 2016 to April 2017*

![Figure 19: Without Hour Ban](image1)

![Figure 20: With Hour Ban](image2)
4.4.5 Zoom

Another variable in the training dataset was the zoom factor of the images. The original image was of size 320*240. While at this size, the image will be very clear but processing the image will take a lot of time. Considering we have tens of thousands of images to process, it is not practical to use the image in its original size. So, we can scale the image to a lower resolution using in-built functions in Python Imaging Library to reduce the image size. While reducing the size of the image will also reduce the amount of information in the image, the prospect of a boost in terms of the time taken to train the classifier made us test this method.

We tried three zoom factors: 0.25, 0.5 and 1.0. 1.0 is effectively the image of the same size. The Figures 21, 22 and 23 give you a better idea of how the images will look when zoomed in differently.

![Figure 21: 0.25 Zoom](image1)

![Figure 22: 0.5 Zoom](image2)

![Figure 23: 1.0 Zoom](image3)
Chapter 5

About the Model

This section describes the different convolutional neural network models that we have used as the classifier. We first describe the architecture of each model in detail and then discuss the different experiments carried out with each model in hopes of improving the accuracy.

5.1 Architectures

All models use a different combination of layers from a set of core layers namely Convolutional Layer, Pooling Layer, Fully Connected Layer or Dense Layer & Dropout Layer. We use a softmax activated dense layer as the final layer of all models. All the different layers have been explained in the Appendix section.

5.1.1 Basic Model

Our basic model has a small and simple architecture. It was designed as a fast and efficient network that requires less computational resource and hence can be tested and debugged quickly while also giving a good accuracy as a classifier.

It begins with 2 Convolutional Layers followed by a max pooling layer. Then, two fully connected layers followed by a final dense layer with softmax activation, which is the output layer. To reduce overfitting, we include a dropout layer after each of the convolutional layer and the fully connected layers.

Both the convolutional layers have kernels of size 3*3 with 32 filters each. The fully connected layers are of sizes 128 & 64 respectively. The final layer is a softmax activated dense layer of
size 5, which is the number of labels. This layer will fire the neuron corresponding to the predicted label for the input image.

The flowchart of the model has been illustrated in Figure 24 for better clarity.

Figure 24: Flowchart of the Basic Model
5.1.2 Inception Model

Inception Architecture is a famous architecture for Convolutional Neural Networks developed by Google. It relies on stacking multiple layers of small convolutional modules to get a very deep neural network. Each convolutional module has many small convolutional layers of sizes 1*1, 3*3 and 5*5. Every module also a pooling layer and a fully connected layer at the end, thereby making every module a small network on its own. Thus, the entire architecture is called ‘Inception’ owing to its Networks within Network arrangement.

The main design principle behind Inception was to use dimensional reduction to keep the number of parameters small. So, instead of having large kernels of size 7*7 or 9*9, it employs sets of smaller kernels of size 3*3. It stacks more convolutional layers to reduce the number of fully connected layers and the number of parameters in turn. It performs well under strict constraints of memory and computation.

We used the Inception-v3 incarnation of the Inception Architecture. Inception-v3 is trained for the ImageNet Large Visual Recognition Challenge using the data from 2012. This is a standard task in computer vision, where models try to classify entire images into 1000 classes, like "Zebra", "Dalmatian", and "Dishwasher". In the ImageNet Database, Inception-v3 reaches 3.46% "top-5 error rate", which is the number of times the model fails to predict the correct answer as one of their top 5 guesses. [12]

The entire Inception-v3 model has been represented in Figure 25.

In order to use the Inception-v3 model to solve our problem, we had to attach an average pooling layer and a 5-way softmax activated dense layer classifier as the last layer. This is because the Inception-v3 was originally designed for the ImageNet Classification which had 1000 labels. However, since we have only 5 labels, we had to extend the model as mentioned above to get our desired output.
5.1.3 Residual Model

Deeper networks famously suffer from the degradation problem i.e. the reduction in accuracy with increasing depth of the network after reaching a maxima. And this reduction in accuracy is not due to overfitting on the training set; the training error actually starts to increase. If a shallower network performs with a certain accuracy then any deeper network can do at least as well as the shallower network. The extra layers can just be identity mappings and give the same result as the shallow network. But this does not happen in practice. This suggests that solvers have difficulty in learning identity mappings by multiple non-linear layers. [13]

The degradation problem and the inability of multiple non-linear layers to learn identity mappings motivate the reformulation into the deep residual learning framework.

The basic concept of deep residual learning is that instead of expecting each few stacked layers to directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping. Let \( H(x) \) be the desired underlying mapping. We try to make the stacked non-linear layers fit another mapping \( F(x) = H(x) - x \). Then the original
mapping can be recast as $F(x) + x$. As proved empirically in [13], it is easier to optimise the residual mapping than to optimise the original mapping.

Here is where short-cut connections come into play. Short-cut connections are the connections which skip one or more layers (Figure 26). In the figure an identity short-cut is added to the learned residual map $F(x)$ to obtain the desired mapping $H(x)$. The authors of [13] have demonstrated experimentally that extremely deep residual networks are easier to optimise than their counterpart plain networks. Also, residual networks do take care of the degradation problem and give better performance than the plain networks. Shortcut onnections have an added benefit of not adding any extra parameters or computational complexity.

We use ResNet50, an incarnation of the Residual Network model. Like the Inception-v3, ResNet50 was also designed for the ImageNet database, which meant it had a 1000-way softmax activated dense layer classifier as the last layer. To use it to solve our problem, we had to extend the model with an average pooling layer followed by a 5-way softmax activated dense layer classifier.

The architecture of ResNet50 has been illustrated in Figure 27.

### 5.1.4 SqueezeNet

Though Inception and Residual Networks are state of the art architectures for Convolutional Neural Networks, they are memory and computation intensive. Since we plan to deploy our classifier on a Raspberry Pi, a mini computer with modest computational resources, such huge architectures will not be usable on it. So, in order to have an efficient alternative, we used the SqueezeNet. [14]

SqueezeNet achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. Additionally, with model compression techniques, we can compress SqueezeNet to less than 0.5MB (510smaller than AlexNet). [15]

To achieve this, three main strategies were used:

- Replace 3x3 filters with 1x1 filters
• Decrease the number of input channels to 3x3 filters
• Down sample late in the network so that convolution layers have large activation maps

SqueezeNet (refer Figure 28) begins with a standalone convolution layer (conv1), followed by 8 Fire modules (fire2-9). A Fire module is comprised of: a squeeze convolution layer (which has only 1x1 filters), feeding into an expand layer that has a mix of 1x1 and 3x3 convolution filters. SqueezeNet ends with a final conv layer (conv10). The number of filters per fire module is gradually increased from the beginning to the end of the network. SqueezeNet performs max-pooling with a stride of 2 after layers conv1, fire4, fire8, and conv10.

Since SqueezeNet was also developed for the ImageNet database classification challenge, like our previous models, we had to attach a 5-way softmax activated dense classifier as the final layer to get our desired output.

5.2 Experiments with Models

Among the four architectures that we have tried, only the first one, our basic model, required most of manual tuning of the hyperparameters as we developed it from scratch. The other three architectures were models that had already been perfected with best hyperparameters.

The loss function we employed for all the models was categorical cross entropy. Categorical cross entropy is measured as

\[ E = \frac{1}{N} \sum_{n=1}^{N} - \log p_n \]

Where ‘N’ is the total number of classes and \( p_n \) is the probability of the predicted class for the nth class.

Cross entropy is a better loss function than simple classification error, which is the ratio of number of correct predictions to number of wrong predictions, because Cross entropy takes into account the closeness of a prediction and is a more granular way to compute error.
Figure 27: Residual Network Architecture
Also, in all our experiments we gave different weights to different classes to reduce the imbalance of label distribution. We gave exponentially increasing weights to the classes based on the number of labels of the class. For example, for class ‘1’ which had the highest number of labels we gave a weight of 1.0, to class ‘0’ which had the second highest number of labels.
we gave a weight of 2.0, then a weight of 4.0 to class ‘2’, 8.0 to class ‘3’ and finally 16 to class ‘4’, which had the least number of examples. We tested other schemes of weights but they did not prove to be as effective.

In our basic model, most of the experiments we ran were to tune the hyperparameters of the model such as learning rate, dropout rate, number of layers, etc. We also tried using different optimizers for the model to decide on the best one to use. Following this, we also tweaked the model by including additional layers that were touted as the current best practices in building convolutional neural networks.

All the experiments we ran are described in detail in the following sections. Since all the experiments were carried out simultaneously in different combinations. The experimental results are presented in a chronological manner separately in the next chapter.

5.2.1 Tuning the Learning rate

The learning rate is probably the most important parameter in any neural network model. It determines the rate at which the weights of the network are modified and is the key variable for the Gradient Descent Algorithm, which is the fundamental optimization algorithm behind neural networks. The Gradient Descent algorithm has been explained in detail in the Appendix Section. Tuning the learning rate involves a lot of manual experimentation to figure out the most optimal learning rate for the model. There is no universal ‘Best Learning Rate’ as it is subject to each model and each training set.

Rather than a vanilla strategy for learning, we used a momentum based approach. Momentum is a technique for accelerating the Gradient Descent Algorithm. Using momentum, we can achieve faster learning rates and better our chances of converging to a global minimum. We also used a learning rate decay. It is the measure by which the learning rate will be reduced in successive epochs. This is done in order to ensure continuous learning even at later epochs. Otherwise, the learning will satiate after some epochs because the learning rate will become too high. Thus, it needs to be gradually reduced for optimal learning.
5.2.2 Tuning the Dropout Rate

To reduce overfitting, we include a dropout layer after each of the convolutional layer and the fully connected layers. The earmark of dropout layers is that there is only one parameter, the dropout rate, that needs to be tuned. The dropout rate is the probability with which neurons are dropped in the next layer. [18]

5.2.3 Tuning the Number of Layers

The number of convolutional layers and fully connected layers in our basic model was decided to be 2 each. This was fixed at 2 because having more layers would create more parameters and thus exponentially increase the size of our model for every layer that was added. The main objective of the basic model was to be less intensive in terms of memory and computation.

As stated earlier in this section, there was no scope to tune the number of layers in our three out-of-the-box models.

5.2.3 Batch Normalization

We used Batch Normalization as suggested in [19] to improve our basic model. Batch Normalization draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Normalization (shifting inputs to zero-mean and unit variance) is often used as a pre-processing step to make the data comparable across features. As the data flows through a deep network, the weights and parameters adjust those values, sometimes making the data too big or too small again - a problem the authors of [19] refer to as "internal covariate shift". By normalizing the data in each mini-batch, this problem is largely avoided. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer.

Since we used out-of-the-box models for the other three models, we did not implement Batch Normalization in those models.
5.2.4 Exponential Linear Units

Rectified Linear Units (ReLUs) have been the standard activations used in Neural Networks. Their performance has been proved empirically. However recently, many improvements to ReLUs have been suggested and we used the most recent and advanced one, which were Exponential Linear Units (ELUs)

In contrast to ReLUs, ELUs have negative values which allows them to push mean unit activations closer to zero. Zero means speed up learning because they bring the gradient closer to the unit natural gradient. Like batch normalization, ELUs push the mean towards zero, but with a significantly smaller computational footprint. ELUs saturate to a negative value with smaller inputs and thereby decrease the propagated variation and information. Therefore, ELUs code the degree of presence of particular phenomena in the input, while they do not quantitatively model the degree of their absence. Consequently, dependencies between ELUs are much easier to model and distinct concepts are less likely to interfere. [20]

As in the case of Batch Normalization, since we used out-of-the-box models for the other three models, we did not implement ELUs in those models.

5.2.5 Noise

In our basic model, we also tried adding a Gaussian Noise layer as the first layer in order to add noise to the input data. Adding noise to the training input will help in making the model generalize to other inputs better.

5.2.6 Optimizers

The most common choice for optimizers for neural networks is the Stochastic Gradient Descent (SGD) Optimizer. We used the same optimizer for most of our experiments. SGD requires us to manually tune the learning rate, decay, and momentum. The methods in which they were tuned were discussed in Section 5.2.1 ‘Tuning the learning rate’. While SGD gave good results, we also tested other optimizers to see if they would fit better.
Chapter 6

Experimental Results

6.1 Procedure

We started experimenting with our basic model (Section 5.1.1). We used the data for the month of November as training data and the first two days of December as testing data. Also, we recoded the labels into 3 classes instead of 5 by combining labels '3','4' and '5' into '3'. This was done in order to simplify the classification and achieve high accuracy. Accuracy refers to the accuracy on the testing set.

Initially our goal was to narrow down the optimal hyperparameters of the network. We started by using an SGD optimizer with a learning rate of 0.01 with momentum as 0.9 and decay as ratio of learning rate to number of epochs. The network's accuracy was stuck at ~16%. This was because the network classified all the labels as '1'. This did not improve even after a lot of epochs. So, clearly the learning rate was very high. We successively reduced the learning rate until the network started to learn at 0.005 learning rate. It achieved an accuracy of 43% and made guesses for all labels instead of classifying everything into a single label. Tuning the learning rate more, we found that the most optimal value was 0.003 where the model achieved 70% accuracy.

Then, we tuned the dropout rates of the model. We had a dropout layer after each convolutional layer and each dense layer, thus, four in total. We considered the two dropout layers after the convolutional layers as set 1 and the two layers after the dense layers as set 2. We started with values of (0.8,0.6) to Set 1 and Set 2 respectively (The values refer to the probability of a neuron being dropped in the next layer). It achieved 43% accuracy. We iteratively reduced the dropout and (0.6,0.6) gave the best accuracy of 70%. The accuracy started to decrease on further reducing the dropout.
Moving on, we tried to determine the best optimizer. Other than SGD, we tested Adam, RMSProp, and Adadelta. RMSProp and Adam performed poorly when compared to SGD. However, Adadelta achieved higher accuracy than SGD with a lower dropout value of (0.5,0.3). It touched 73.6% accuracy. Higher or Lower dropout values did not improve the performance of Adadelta.

The best models were SGD with learning rate of 0.003 and dropout of (0.6,0.6) with an accuracy of 70% and Adadelta with dropouts of (0.5,0.3) and accuracy of 73%.

After narrowing down on the hyperparameters with 3 classes, we recoded the model into the original 5 class model. With 5 classes, our best models did not achieve the expected levels of accuracy. They achieved 52% (SGD) and 56% (Adadelta) respectively. In order to make them more suited for 5 classes, we further tuned the dropouts. SGD with learning rate of 0.003 and dropout of (0.5,0.4) was able to achieve 63% and Adadelta with dropouts of (0.3,0.15) achieved 64.8%. However, the problem with these models was they started to overfit the training data a lot. The SGD model achieved 86% training accuracy and only 63% testing accuracy after 200 epochs. Similarly, Adadelta model achieved 91.7% training accuracy but only 64.8% testing accuracy.

We also tried increasing the number of dense layers from 2 to 3. Instead of having two layers with sizes [64,32] we tested 3 layers with sizes [128,64,32]. However, it did not show any significant improvement in the results. Also, when having more convolutional layers we could not see significant improvements in accuracy. We tested this empirically by having 3 and 4 convolutional layers in our model. We could not find any improvements in accuracy from the same.

For a month of training data at a zoom of 0.25, each epoch took 0.5 minutes on the GPU.

At this point, we got more training data to work with. We had the months of January and February available for training as well. Thus, we had three months of data for training and 2 days of December for testing.

With 3 months of data, we observed that increasing the sizes of the fully connected layers was fruitful. SGD Model with a learning rate of 0.003 and (0.5,0.4) dropout gave 63% accuracy
with [64,32] sized dense layers. However, it gave 67% accuracy with [128,64] sized layers. The increase in time required for an epoch was also negligible. With 3 months of data, every epoch took 0.9 mins approximately.

The best model we developed at this stage was SGD with a learning rate of 0.003 and dropout of [0.5,0.4]. It achieved 67% accuracy in testing and 63% in training thus was also free from overfitting.

Next, we tried increasing the maximum time delay for mapping images to labels from 5 minutes to 15 minutes. The statistical differences were presented earlier in Section 4.3.1. Practically, we observed that with 15-minute delay and 3 months of data, a single epoch took 2.8 mins. Roughly 3 times the time taken with 5-minute delay. Also, no gains in accuracy were observed with 15-minute delay. Thus, we stuck to using 5-minute delay for our experiments.

Following that, we experimented with Batch Normalization. We added a Batch Normalization layer after every convolutional and dense layer. The reasons were explained in Section 5.2.3. After batch normalization, the average time for each epoch became 1.4 mins. Thus, there was a 0.5x decrease in the time for one epoch. With Batch Normalization, we could reach 68% accuracy with Adadelta (0.2,0.2) dropout and SGD with learning rate of 0.003 and dropout (0.5,0.4) However, the problem of overfitting became a serious issue as with Batch Normalization, the training accuracy started touching 100% within 25 epochs. To combat this, while reading about Batch Normalization, we found that using Batch Normalization before activations, if any, resulted in better accuracy by practice. In our case, we did not find any significant difference between using Batch Normalization before or after activations. However, we chose to stick with using it before activation to follow convention.

The next experiment we focused on mainly to reduce overfitting was adding noise (Section 5.2.5). This experiment however proved futile as either it made the performance worse, in case of large noise, or the performance was unaffected, in case of less noise.

Afterwards, we tested changing the activation from ReLU to ELU as explained in Section (5.2.4) Using ELU instead of ReLU significantly reduced the overfitting in our models with Batch Normalization. Also, with ELU we were able to use a higher learning rate of 0.04 to
achieve 67% accuracy with all other parameters remaining the same. Our model with ELU took 1.6 mins per epoch, roughly a 15% increase in the time taken.

As another attempt to increase accuracy we tried decreasing the batch size from 256 to 32. However, no improvements were observed.

Table 9 summarizes all the important experiments on the basic model and their results.

Before moving on to using out-of-the-box models, we experimented on the dataset. The details of which were elaborated in the Section 4.4. The results of experiments are presented below.

- **Grayscale (4.4.1)**

  The accuracy of the model did not improve on using grayscale images. Intuitively, we can argue that the loss of color information is deteriorating for the classifier, contradicting to our initial expectation.

- **Horizontal Flipping (4.4.2)**

  This method did not provide a significant increase in accuracy of the classifier. The reason for the same can be speculated as not balancing the label distribution enough. Though, further duplication of labels, say tripling the labels ‘2’, ‘3’ and ‘4’ was not done for fear of over-fitting those training examples by providing them as input to the classifier again and again.

- **Masks (4.4.3)**

  This method was also futile as the accuracy of the classifier did not improve on using masked images. In fact, the masking of images did not contribute any change in the training of the classifier. Thus, we can only infer that black pixels are equally undesirable as colored pixels, in the surrounding imagery.
• **Hour Ban (4.4.4)**

The training dataset with hour ban did not improve the accuracy of the classifier as the label distribution was still quite uneven. The domination of ‘2’ labels (low traffic) is an innate attribute of the dataset, owing to the nature of traffic.

• **Zoom (4.4.5)**

Higher zoom factor did not improve the accuracy of the basic model. However, higher zoom factor meant more training time. 1.0 zoom took 9x the time required for 0.25 zoom. Also, some models that we used, for example, Inception and Residual Models, required a zoom factor of 0.75 and 1.0 respectively for the input size to be sufficient for the model and thus they took much longer to train. Our basic model and Squeeze Net model worked with 0.25 zoom as well thus eliminating the need to work with 1.0 zoom in their cases.

**Inception-v3**

Once we were convinced we had extensively tweaked and tuned our basic model, we switched over to using out of the box models. We first started with the Inception Architecture model: Inception-v3 [ Refer Section 5.1.2]

We initialized the model with ImageNet weights. These weights gave the best results on the ImageNet classification challenge. Though, after loading the weights, we had also trained the model with our own dataset to make it more specific and better in our domain. Also, Inception Model required a minimum input size of 139*139. Thus, a zoom factor of 0.25 will not be sufficient input. We used a zoom factor of 0.75 which gave an input image size of (172,232).

To train the Inception-v3 model with our data we started with SGD Optimizer and an initial learning rate of 0.04. However, the model gave an accuracy of only 34% and did not improve even after many epochs. So, we iteratively reduced the learning rate until 0.0005 which worked best. This can be intuitively reasoned as being so because of the fact that since this model already have well trained weights, it needs only very minor changes and thus a very low
learning rate. The Inception model performed poorly with the Adadelta optimizer. We also tried Random Initialization for the weights instead of loading ImageNet weights. The Random Initialized model could not achieve the same accuracy as the model with ImageNet weights.

With a learning rate of 0.0005, Inception-v3 achieved 71.4% accuracy. However, since the model and the data size is so huge, with 3 months of data each epoch took 20 minutes on the GPU.

**ResNet50**

After trying Inception Architecture, we also tested Residual Network Architecture [ Refer Section 5.1.3] using ResNet50 model. The method of experimentation on the ResNet50 was similar to Inception-v3. ResNet50 required a minimum input image size of (197,197). Thus, we had to use a zoom factor of 1.0, essentially the entire image. The input image size was [230,310]. Though the original image is 240*320, since we make a crop of 5 pixels on all sides to eliminate the border, we get a size of 230*310.

We loaded ResNet with ImageNet weights and we started with a SGD Optimizer and a learning rate of 0.0001. The model touched an accuracy of 73%. However, the model also tended to overfit a lot as it reached 97% accuracy on the training dataset. To overcome this, we added a final dropout layer with 0.5 dropout rate. With the dropout layer, the model touched 74% accuracy. As with Inception model, even in ResNet model, SGD was better than Adadelta in terms of the accuracy achieved. With 3 months of data, each epoch took 34 minutes on the GPU.
<table>
<thead>
<tr>
<th>Experiment</th>
<th>Changes Made</th>
<th>Number of Epochs</th>
<th>Time take for 1 Epoch</th>
<th>Training Accuracy</th>
<th>Testing Accuracy</th>
<th>Number of Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>LR = 0.01, Dropout = (0.8,0.6)</td>
<td>50</td>
<td>~0.5 mins</td>
<td>31.7</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>LR</td>
<td>LR = 0.005, Dropout = (0.8,0.6)</td>
<td>50</td>
<td>~0.5 mins</td>
<td>55.6</td>
<td>43</td>
<td>3</td>
</tr>
<tr>
<td>Dropout</td>
<td>LR = 0.005, Dropout = (0.6,0.5)</td>
<td>50</td>
<td>~0.5 mins</td>
<td>68.9</td>
<td>66</td>
<td>3</td>
</tr>
<tr>
<td>LR &amp; Dropout</td>
<td>LR = 0.003, Dropout = (0.6,0.6)</td>
<td><strong>50</strong></td>
<td>~0.5 mins</td>
<td><strong>69.6</strong></td>
<td><strong>70</strong></td>
<td>3</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adadelta, Dropout = (0.8,0.6)</td>
<td>25</td>
<td>~0.5 mins</td>
<td>31.7</td>
<td>16.1</td>
<td>3</td>
</tr>
<tr>
<td>Dropout</td>
<td>Adadelta, Dropout = (0.5,0.3)</td>
<td>25</td>
<td>~0.5 mins</td>
<td>78.7</td>
<td>73.6</td>
<td>3</td>
</tr>
<tr>
<td>Optimizer</td>
<td>RMSProp, Dropout = (0.6,0.5)</td>
<td>50</td>
<td>~0.5 mins</td>
<td>66.2</td>
<td>57</td>
<td>3</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam, Dropout = (0.6,0.5)</td>
<td>25</td>
<td>~0.5 mins</td>
<td>31.7</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>Number of Classes</td>
<td>LR = 0.003, Dropout = (0.6,0.6)</td>
<td>30</td>
<td>~0.5 mins</td>
<td>58.2</td>
<td>52</td>
<td>5</td>
</tr>
<tr>
<td>Dropout</td>
<td>LR = 0.003, Dropout = (0.5,0.4)</td>
<td><strong>200</strong></td>
<td>~0.5 mins</td>
<td><strong>77</strong></td>
<td><strong>62</strong></td>
<td>5</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adadelta, Dropout = (0.5,0.3)</td>
<td>50</td>
<td>~0.5 mins</td>
<td>56</td>
<td>56</td>
<td>5</td>
</tr>
<tr>
<td>Dropout</td>
<td>Adadelta, Dropout = (0.3,0.15)</td>
<td><strong>200</strong></td>
<td>~0.5 mins</td>
<td><strong>91.7</strong></td>
<td><strong>64.8</strong></td>
<td>5</td>
</tr>
<tr>
<td>Number of Layers</td>
<td>SGD, LR = 0.003, Dropout = (0.5,0.4), Dense Layers = [128,64,32]</td>
<td>200</td>
<td>~0.5 mins</td>
<td>91</td>
<td>62</td>
<td>5</td>
</tr>
<tr>
<td>More data</td>
<td>SGD, LR = 0.003, Dropout = (0.5,0.4), Dense Layers = [64,32]</td>
<td>30</td>
<td>~0.9 mins</td>
<td>61.7</td>
<td>63.1</td>
<td>5</td>
</tr>
<tr>
<td>Experiment</td>
<td>Changes Made</td>
<td>Number of Epochs</td>
<td>Time take for 1 Epoch</td>
<td>Training Accuracy</td>
<td>Testing Accuracy</td>
<td>Number of Classes</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------</td>
<td>------------------</td>
<td>------------------------</td>
<td>-------------------</td>
<td>------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Number of Layer</td>
<td>SGD, LR = 0.003, Dropout = (0.5,0.4), Dense Layers = [128,64]</td>
<td>30</td>
<td>~0.9 mins</td>
<td>63.8</td>
<td>67.5</td>
<td>5</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adadelta, Dropout = (0.2,0.2)</td>
<td>30</td>
<td>~0.9 mins</td>
<td>86.5</td>
<td>65.9</td>
<td>5</td>
</tr>
<tr>
<td>Max Time Delay</td>
<td>Adadelta, Dropout = (0.2,0.2)</td>
<td>44</td>
<td>~2.7 mins</td>
<td>78.9</td>
<td>63.7</td>
<td>5</td>
</tr>
<tr>
<td>Batch Normalization</td>
<td>Adadelta, Dropout = (0.2,0.2)</td>
<td>25</td>
<td>~1.4 mins</td>
<td>99.9</td>
<td>67</td>
<td>5</td>
</tr>
<tr>
<td>Batch Normalization</td>
<td>SGD, LR = 0.003, Dropout = (0.5,0.4)</td>
<td>30</td>
<td>~1.4 mins</td>
<td>82.3</td>
<td>68.6</td>
<td>5</td>
</tr>
<tr>
<td>ELU</td>
<td>SGD, LR = 0.04, Dropout = (0.5,0.5)</td>
<td>50</td>
<td>~1.6 mins</td>
<td>77</td>
<td>67</td>
<td>5</td>
</tr>
</tbody>
</table>

*Table 9: Summary of the important results of our experiments with the basic model*

**SqueezeNet**

SqueezeNet was the last model we experimented with. We loaded SqueezeNet model with ImageNet weights and started with a SGD optimizer with learning rate of 0.0001. The model performed poorly, classifying all images as label '2'. It was unable to learn even after 25 epochs. We tried adding a dense layer of 8 neurons and a ELU activation layer followed by a dropout layer with dropout rate of 0.3. However, the result did not improve. Surprisingly, Adadelta optimizer worked well with SqueezeNet when we added the extra dense layer with ELU activation and the dropout layer with dropout rate of 0.3. It gave an accuracy of 68% with 0.25 zoom. Each epoch took 1.4 minutes. With a zoom factor of 1.0, the accuracy was again 68% though the time taken for one epoch was 9 times more. Each epoch took 9.6 minutes. Thus, there was no need to use 1.0 zoom for SqueezeNet.
More data

In the final phase of the project, we got more data for the entire month of December and the months of March and April. Thus, we could use the images from November 2016 to April 2017. We used the last 3 days of April for testing and the rest of the data for training. With an increase in the amount of training data, by approximately 2 times, all the models took similarly twice the amount of time for one epoch. The accuracies obtained with the new test data cannot be compared with our previous test data as they are entirely different sets. However, the accuracy values with the new test data were in the similar range as with the old test data.

A common feature observed with the new data was that the overfitting was considerably less even with complex models such as Inception-v3 and ResNet50. The training accuracy was always close to the test accuracy thus justifying the use of more data. We have compared the results of the models on the new data against the results on the old data below. We have also given the confusion matrices for each of the model for better clarity.

ResNet50 achieved 72% accuracy, which is close to the 74% accuracy it obtained on the old test data. ResNet50 also took 67 minutes for one epoch as opposed to the 34 minutes it took with the previous training data.

\[
\begin{array}{cccc}
377 & 104 & 13 & 0 \\
127 & 512 & 39 & 5 \\
3 & 35 & 34 & 3 \\
0 & 8 & 7 & 16 \\
0 & 2 & 5 & 5 \\
\end{array}
\]

Confusion Matrix of ResNet50

Inception-v3 was able to achieve only 65% with the new data while it was able to achieve 72% with the old data. The time taken was 39 minutes which was in accordance with the 19.5 minutes it took when using the old data.

\[
\begin{array}{cccc}
317 & 149 & 20 & 7 \\
138 & 483 & 40 & 17 \\
8 & 45 & 16 & 5 \\
0 & 22 & 5 & 4 \\
0 & 8 & 0 & 0 \\
\end{array}
\]

Confusion Matrix of Inception-v3
Likewise, SqueezeNet achieved 68% accuracy with the new data and took 2.7 minutes per epoch at 0.25 zoom.

![Confusion Matrix of SqueezeNet]

Our Basic Model, achieved 57% accuracy with the new data and each epoch took 3 minutes at 0.25 zoom.

![Confusion Matrix of our Basic Model]

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of Epochs</th>
<th>Time taken for 1 Epoch</th>
<th>Training Accuracy</th>
<th>Testing Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Model (0.25 Zoom)</td>
<td>25</td>
<td>~3 minutes</td>
<td>70</td>
<td>57</td>
</tr>
<tr>
<td>Inception-v3 (0.75 Zoom)</td>
<td>10</td>
<td>~39 minutes</td>
<td>63</td>
<td>65</td>
</tr>
<tr>
<td>ResNet50 (1.0 Zoom)</td>
<td>10</td>
<td>~67 minutes</td>
<td>83</td>
<td>72</td>
</tr>
<tr>
<td>SqueezeNet (0.25 Zoom)</td>
<td>30</td>
<td>~2.4 mins</td>
<td>68</td>
<td>68</td>
</tr>
</tbody>
</table>

*Table 10: Summary of the Best Models*

Table 10 gives a summary of the best models we have trained on the latest dataset.
From Table 10, we can clearly observe that out of the four models, if we were concerned only about the accuracy of our model, ResNet50 is the model of choice. However, for deployment, there are factors other than accuracy such as resources used and performance time which come into play. With ResNet50, we get a 180 MB model and such a big model is bound to take a long time to classify an image. As our deployment is on a Raspberry Pi, which is incapable of running the huge ResNet50 model, and we want faster performance than ResNet50, we use the SqueezeNet model on the Raspberry Pi, which requires only 33 MB and can classify 2 images per minute. The Basic Model is not used as SqueezeNet is smaller, faster and more accurate than our Basic Model. With respect to the Inception-v3, it is again trumped by SqueezeNet in terms of size, efficiency and accuracy. Thus, SqueezeNet is the model we chose to deploy on the Pi.

6.2 Alternative Approaches

Some of the alternative approaches that were considered to solve the problem are listed below.

6.2.1 Individual Classifiers

As we have stated earlier, we have 27 street cameras across the city of Barcelona. Instead of developing a single classifier for all the cameras, we tried developing an individual classifier for each camera. The motivation behind this approach was the fact that since each camera was positioned differently and captured images at a different zoom, a common classifier might not be as effective as an individual classifier.

So, we generated a new training dataset with only the images from a single camera. The disadvantage of this method was the fact that we had very less data for training or testing the classifier this way. For example, when using the camera placed at 'Plaza Catalunya', We had only 2,168 images for training and 58 images for testing.

We got an accuracy of 65% on the test set. However, since we had a small number of test images we cannot be sure whether it will scale up with the same accuracy when size of the training and testing data is increased.
6.2.2 Regression Modelling

In order to solve the problem of estimating traffic density, we considered the problem as a classification problem wherein each image had to be classified into one of 5 different labels, from 1 to 5, according to the level of traffic in the image. However, another approach to solve the same problem would be to consider the problem as a regression problem wherein the image is given a numerical score, in a continuous interval from 1-5.

We tested the regression method only on the SqueezeNet model. To implement a regression-based approach, we made a few changes to the model. We used mean squared error as the loss function instead of cross entropy as cross entropy is not suitable for regression problems. The final layer was a single neuron, 1 sized dense layer, instead of a 5 way softmax classifier as we wanted a single numeric value as the output.

We trained the model for 30 epochs. A low loss did not mean that the predictions were well distributed. When we used the labels from 1-5 we did not obtain any predictions from 0 to 1 and more than 5. So, we rescaled the labels from 1-5 to 0-4 in order to obtain a more continuous set of predictions from 0.

For example, the model that achieved the lowest loss of 0.45 was a model that predicted all values to be 1.47. So, in order to make sense of the input, we had to classify those predictions again. Since, lowest loss did not mean best predictions, we stored the predictions of the model after all epochs in different files and compared them. When we classified all predictions from < 0.5 as ‘0’, 0.5-1.5 as ‘1’, 1.5-2.5 as ‘2’, 2.5-3.5 as ‘3’ and > 3.5 as ‘4’, the lowest classification error we got was 36.2% which meant an accuracy of 63.8%.

Though, a general trend we observed was that none of the predictions were above 4 or below 0. In our best prediction set, none of the predictions were above 3. By altering the limits used for classifying the numerical values into labels, we realized that it will not result in a significant lowering of the wrong predictions. The reason can be explained with an example. If we classify the predictions between 1.5 - 2.5 as belonging to label ‘2’, some images labelled as ‘2’ have been predicted to be <1 and some have been predicted to be >3. It would be naive of us to
extend the limits to include all such predictions as it will lower the accuracy of the other classes too, which suffer from the same anomalies.

As the regression model did not outperform our classification model, we did not use the regression model for final deployment.
Chapter 7

Deployment

The platform chosen to be deploy the project was the Raspberry Pi. We had a working Raspberry Pi 2. We used the free monitors available at the library in the campus to work with the Pi. Deployment began with the initial setting up of the Raspberry Pi 2 which included installing the Raspbian OS on the Pi and then installing Python and all its required packages. Some packages like TensorFlow were not inherently built for Raspbian OS and thus had a few dependencies which had to be sorted out by installing specific third-party wheels, special Python installation packages, for those particular packages.

Once the environment was set up, we developed a simple application that could load the classifier model (which was previously trained on the supercomputer), the image and predict the classification of the image using the model. All the functions to do the same were available in the Keras package. We displayed the prediction along with the image on a webpage. A screenshot of the webpage has been given in Figure 29.

![Traffic Status Prediction](image)

*Figure 29: A screenshot of the webpage with the prediction*
Afterwards, we deployed the same on a Web Server using Flask, an open source Python library for micro-web development.

The Raspberry Pi hosts a web server on its network, which can be accessed by any client which has access to the network. Thus, the web page with results can be viewed from any laptop which is connected to the same network as the Raspberry Pi. We have deployed the server on a local network. We can also deploy the server on the Internet, which would enable any client with access to Internet to be able to access our results page. This Internet-based client server model is better understood with the image in Figure 30.

We conducted a basic test of performance once we deployed the SqueezeNet model on the Raspberry Pi 2. On an average, our classifier was able to classify 32 images in 4 seconds. Thus, it can classify an image in roughly 0.12 seconds. In a minute, it is capable of classifying of 480 images. This level of response is more than sufficient for real-time usage.
Chapter 8

Conclusion & Future Work

8.1 Conclusion

We successfully developed a classifier model which can estimate traffic density from street camera images using convolutional neural networks. The advantage of this model over current technologies is that it leverages the use of machine learning and artificial intelligence, and is also cheaper and more efficient. We were able to achieve an accuracy of 72% with our best model.

We also deployed the model on a Raspberry Pi Mini Computer as planned. On the Raspberry Pi, we could not run our best model due to computational constraints. We ran a model that achieves 69% accuracy on the Pi. The Raspberry Pi hosts a web server that can be accessed to view the image processed and the prediction made. We designed a basic webpage for the same. We achieved a response time of 0.12 seconds per image, thus making our model highly suitable for real-time usage.

8.2 Future Work

There is a scope for further improvements to the accuracy of the model. The following methods can be experimented on.

- Relative zooming of images

Instead of a common zoom for all the images, which is what we have used, there can be a relative zoom for each image i.e. since every camera captures an image of the street at different
angle and zoom, every image can be rotated and zoomed by different measures so that the only the street is visible to the classifier.

- Getting more data for Individual Classifiers

While, we did consider using Individual Classifiers, we were not able to get satisfactory results because we lacked enough data for training individual classifiers. With more time and effort, more data can be gathered for this approach and it could turn out to be very efficient and accurate.

- Manual curating of labels

An innate problem with the training dataset we used was the mismatch of labels and images as they were obtained from different sources. Instead of mapping such labels from different sources, which will always have some percentage of error, we can employ Human Assisted Curating of images to give a perfectly labelled dataset. This method is commonly used for High-Scale projects and datasets like the ImageNet. Amazon Ltd. provides an easy interface to do the same via Amazon Mechanical Turk. However, the down side of this method is that it requires more resources in terms of time and money.

This project can also be extended in terms of the scale of deployment. Instead of using the model only for Barcelona, the model can be used with traffic image sets of other cities too. This way, the model can be further generalized and made more accurate.

### 8.3 Personal Conclusion

At the outset, I would like to place on record my gratitude to Professor Javier Bejar for giving me the opportunity to work on such an exciting and enriching project. This project was at just the right difficulty for me, wherein, neither was it too easy being a walk through nor too hard, seemingly impossible. It was challenging and piqued my curiosity to put my best efforts into it. Being my first foray into the domain of Machine Learning and Neural Networks, I learnt a lot thanks to this project, both academically and practically. The background knowledge that I
had to acquire before beginning the project gave me a thorough theoretical exposure to convolutional neural networks. Reading through so many research papers in this domain, I obtained a very good awareness of the latest advances, best practices and conventions in this field. After starting the work on project, I was exposed to the practical side of implementing neural networks and along with it, I got a good experience of Image Processing and File Handling in Python as well. Working with such a comprehensive codebase taught me how important modularity and writing clean code was, when building software. More importantly, the work that I did has inspired me to delve deeper into the domain and pursue higher studies in the same domain. Apart from all these technical insights, I gained a lot of valuable non-technical skills too. Running hundreds of experiments overnight and waiting for the results, taught me how important patience and discipline were, in research. Also, being an exchange student in Barcelona, I was able to live and relish a beautiful city with a great culture. The successful completion of this project, as a one-man team, in a new environment, in a new city has boosted my confidence and made me ready to take on bigger challenges. However, this would not have been possible without the constant support and expert guidance of Professor Javier Bejar, to whom I am eternally grateful. So, on the whole, this project has given me a wonderful experience which has improved my skills and character. I take with me a lot of knowledge and cherishable memories, as I move forward in life.
Chapter 9

References


Appendix A: Requirements

The following tools are required for the project.

**Software**

- **Python** was chosen as the language of implementation as it is one of the best and most famous programming/scripting languages, aptly suited for this project. Python enables us to use a highly modular framework with a huge library of packages which we can utilize for various tasks. It can also be run on Raspberry Pi, which we will use for deployment of the project.

**Python Packages**

- To get the data from the server we use **Requests**, which is a HTTP API built for python.
- To process the data, we use **PIL** (Python Imaging Library).
- To store processed data as datasets, we use **h5py**.
- To implement the convolutional neural network model, we use **Keras**. Keras is a high-level neural networks library, written in Python and capable of running on top of either **TensorFlow** or **Theano**, both of which are well-known comprehensive machine learning libraries in Python. It was developed with a focus on enabling fast experimentation.
- To build the deployment server on Raspberry Pi, we use **Flask**, which is a micro web development framework for Python.

- We also use **Git**, the best way for project management and version control. GitHub gives us a git repository on the cloud, which can be used for synchronization of different files and security of the project.
• **PyCharm**, a Python IDE developed by JetBrains is used for working on the project. It has automatic synchronization with Git and helpful features like code completion, automatic error checking, suggestions, package control, logging, etc.

**Hardware**

• **RaspberryPi**, a mini computer that will be used to capture the image and run the neural network model to classify the image, at the site after deployment.

• **Laptop**, our primary workstation. This is bundled with the basic software (Ubuntu 16.10, Google Chrome, Libre OpenOffice, Sublime Text Editor) and internet connection.
Appendix B: Background Knowledge

Convolutional Neural Networks

Convolutional networks (LeCun, 1989), also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology. Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representations.

- Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means every output unit interacts with every input unit. Convolutional networks, however, typically have sparse interactions (also referred to as sparse connectivity or sparse weights). This is accomplished by making the kernel smaller than the input.

- Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. As a synonym for parameter sharing, one can say that a network has tied weights, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere. In a convolutional neural net, each member of the kernel is used at every position of the input.
• In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation. To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function $f(x)$ is equivariant to a function $g$ if $f(g(x)) = g(f(x))$. [9]

Convolutional network (ConvNet) architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the number of parameters in the network.

Left (Figure 31): A regular 3-layer Neural Network. Right (Figure 32): A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels). [10]
Layers of a Convolutional Neural Network

Convolutional Layer

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

Pooling Layer

Pooling Layers use a pooling function on the input and return the result of the function as output. A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the max pooling operation reports the maximum output within a rectangular neighborhood. Other popular pooling functions include the average of a rectangular neighborhood, the L2 norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel. In all cases, pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.
**Fully Connected Layer (Dense Layer)**

The high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

**Softmax Activation Layer**

In mathematics, the softmax function, or normalized exponential function, is a generalization of the logistic function that "squashes" a K-dimensional vector $\mathbf{z}$ of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range $(0, 1]$ that add up to 1. The function is given by

$$
\sigma(\mathbf{z}) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, \ldots, k
$$

The output of the softmax function can be used to represent a categorical distribution – that is, a probability distribution over K different possible outcomes. The softmax function is often used in the final layer of a neural network-based classifier. Thus, the softmax activation layer is a dense layer with K neurons activated by the softmax function. It is the final layer which outputs the probability of the input belonging to each of the K classes.

**Dropout Layer**

Because a fully connected layer occupies most of the parameters, it is prone to overfitting. One method to reduce overfitting is dropout. At each training stage, individual nodes are either "dropped out" of the net with probability $1-p$ or kept with probability $p$, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights.
In the training stages, the probability that a hidden node will be dropped is usually 0.5; for input nodes, this should be much lower, intuitively because information is directly lost when input nodes are ignored.

**Gradient Descent Algorithm**

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost). Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm. In Neural Networks, Gradient Descent is used to optimize the weights of the network.

**Intuition**

Think of a large bowl like what you would eat cereal out of or store fruit in. This bowl is a plot of the cost function (f). A random position on the surface of the bowl is the cost of the current values of the coefficients (cost). The bottom of the bowl is the cost of the best set of coefficients, the minimum of the function. The goal is to continue to try different values for the coefficients, evaluate their cost and select new coefficients that have a slightly better (lower) cost. Repeating this process enough times will lead to the bottom of the bowl and you will know the values of the coefficients that result in the minimum cost.

**Procedure**

The procedure starts off with initial values for the coefficient or coefficients for the function. These could be 0.0 or a small random value.

\[
\text{coefficient} = 0.0
\]

The cost of the coefficients is evaluated by plugging them into the function and calculating the cost.

\[
\text{cost} = f(\text{coefficient})
\]
The derivative of the cost is calculated. The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.

\[ \text{delta} = \text{derivative(cost)} \]

Now that we know from the derivative which direction is downhill, we can now update the coefficient values. A learning rate parameter (alpha) must be specified that controls how much the coefficients can change on each update.

\[ \text{coefficient} = \text{coefficient} - (\alpha \times \text{delta}) \]

This process is repeated until the cost of the coefficients (cost) is 0.0 or close enough to zero to be good enough.

**Raspberry Pi**

The Raspberry Pi is a credit-card-sized computer that plugs into your TV and a keyboard. It is a capable little computer which can be used in electronics projects, and for many of the things that your desktop PC does, like spreadsheets, word processing, browsing the internet, and playing games. It also plays high-definition video.

All models feature a Broadcom system on a chip (SoC), which includes an ARM compatible central processing unit (CPU) and an on-chip graphics processing unit (GPU, a Video Core IV). The Raspberry Pi primarily uses Raspbian, a Debian-based Linux operating system.