Pothole Detection and Localization

By

Dana Abi Badra

Parveen Ayoubi

Markyves Menassa

A report submitted to the Faculty of Engineering in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Engineering

Faculty of Engineering

University of Balamand

May 2023

University of Balamand

Faculty of Engineering

This is to certify that I have examined this copy of a project by

Dana Abi Badra

Parveen Ayoubi

Markyves Menassa

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining jury have been made.

JURY MEMBERS:

Approved:
Dr. Ghattas Akkad
Supervisor

Approved:
Dr. Chafic Mokbel
Jury Member

Approved:
Dr. Nicolas Haddad
Jury Member

ACKNOWLEDGMENT

We would like to express our sincere gratitude to our moderators, Dr. Chafic Mokbel and Dr. Nicholas Haddad who have provided helpful feedback and constructive criticism. We are very thankful to our supervisor, Dr. Ghattas Akkad, whose guidance, whose advice and perceptive remarks greatly aided us in navigating the difficulties of this study and in defining its scope and direction. We are also very grateful to every member of the Faculty of Engineering at the University of Balamand for the valuable information they provided.

ABSTRACT

Potholes or chuckholes are pits or holes in a road surface that result from gradual damage caused by traffic and/or weather. Such road hazards, if left unpaved, can cause serious damage and can lead to fatal accidents. The main reasons for untreated potholes are the lack of awareness of the concerned authorities and minimal user reporting. There are many applications for the detection of objects on the road, with some of the most promising occurring in autonomous driving and surface defects. These applications are made possible through cameras that are mounted on moving vehicles. In order to address the challenges of detecting potholes from images and videos, there have been many methods proposed. These methods include processing images or videos captured with cameras from mobile phones, unmanned aerial vehicles, and drones. However, these devices are equipped with high-end processors and are expensive to deploy in bulk or on non-computerized vehicles/bicycles. Thus, this project focuses on designing and implementing a low-cost pothole detection and localization device. The machine learning-powered prototype is a portable device equipped with a low-power processor, a camera sensor a global positioning sensor, and a power source. The processor, an edge device, runs a pre-trained convolutional neural network that inputs the raw image data to detect potholes then saves and logs its location.

Keywords: pothole, detection, localization, machine learning, embedded systems

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Amazon Web Service	AWS
Artificial Intelligence	AI
Artificial Neural Networks	ANN
Circular Error Probability	CEP
Comma-Separated Values	CSV
Constant Current Constant Bus	CCCV
Convolutional Neural Networks	CNN
Deep Neural Network	DNN
Direct Current	DC
Faster Objects, More Objects	FOMO
Giga Bytes	GB
General Purpose Input Output	GPIO
Global Positioning System	GPS
Integrated Development Environment	IDE
Inter-Integrated Circuit	I^2C
Internet of Things	IoT
k-Nearest Neighbor	KNN
Liquid Crystal Display	LCD
Machine Learning	ML
Neural Networks	NN
Quasi-Zenith Satellite System	QZSS
Random Access Memory	RAM
Recurrent Neural Networks	RNN
Satellite Based Augmentation System	SBAS
Serial Peripheral Interphase	SPI
Surface Mountable Design	SMD
Serial Camera Control Bus	SCCB
Three dimensional	3D
Tiny Machine Learning	TinyML
Two dimensional	2D
Universal Asynchronous Receiver Transmitter	UART
Unmanned Aerial Vehicles	UAVs
You Only Look Once	YOLO

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Research Study

1.1 Introduction

A pothole is defined as a hollow in the road surface brought on by an asphalt road's structural failure, as shown in Figure 1-1.

Figure 1-1: Pothole on the Road

Potholes are not a recently occurring problem, however, a persistent one that countries all around the world suffer from. The problem arose in 1909 when cars and vehicles became more widespread. Although these dips frequently take an elliptical shape, they can also appear in abnormal forms. Their diameter is wider at the middle and bottom sections than at the top [1]. Environmental factors such as heat, rain, and ice play a major role in the formation of potholes. Potholes are inevitable [2]; underlying water, primarily by freezing action, deteriorates the soil's structure in the ground underneath the road no matter how tightly the surface is sealed. The pressure on the surface increases as this water expands and contracts when the temperature changes. Eventually, the road surface breaks down and forms a pothole as a result of traffic wandering over this weak area. The large weight of cars and trucks speeds up this process and

broadens the cracks leading to wider and deeper potholes. Once formed and left untreated, potholes can have various effects on the vehicle itself as well as the driver. First, when hitting a pothole, tires will suffer the most damage in a vehicle. Punctures can either cause the car to stop immediately, or the tires will deflate almost undetected until they flatten and require replacement. The damage extends beyond the tires to reach the rim [3]. Under pressure, the metal is easily bent, which poses a serious stability risk. The suspension may also be negatively impacted, which makes steering much more difficult. Since drivers are advised to slow done in order to avoid these potholes, traffic jam is more likely to occur in these areas. In addition to that, potholes pose a greater problem for motorbikes. It does not only damage the bike but could also injure the rider by causing them lose control [2]. Whether a vehicle is swerving to avoid them, or due to stability and control problems, potholes can lead to accidents that put the driver's life at risk. The situation of roads in Lebanon is hazardous, especially since the number of potholes is increasing [4]. The roads' poor infrastructure is causing a large number of accidents, resulting in both life and financial losses. From 2007 to the beginning of 2022, an average of 4259 accidents were reported in Lebanon on a yearly basis, killing an average of 519 people, and injuring 5760, with motorcyclists accounting for nearly 40% of those reported road injuries. Some countries, -mostly developed-, provide a way of reporting potholes through online forms or applications. For example, to report a pothole in Edmonton Canada, the user can just download an application called 311 and pinpoint the location of the pothole to request having it fixed as quickly as possible [5]. However, in Lebanon, such services are not yet available; people aren't provided autonomous and ready-to-use tools to easily report those potholes and pinpoint their location. If left unreported, these potholes are causing numerous accidents daily and presenting a threat to any driver on any Lebanese road.

1.2 Literature Review

1.2.1 Pothole detection design and techniques

There have been many methods and techniques proposed to perform reliable pothole detection. One method is the vision-based method which acquires a series of snapshots using a connected camera for automatic detection [6]. Another technique is the vibration-based technique. This method decides if there is a pothole by data acquired using the acceleration sensor in the car and can also approximate its depth. Another method is the 3D reconstruction-based technique which is based on stereo-vision technology. It approximates the shape of the pothole and calculates its volume once detected.

Table 1-1: Popular Pothole Detection Methods

Method	Vision-Based method [7]	Vibration-Based method (Sensor) [7]	Three dimensional (3D reconstruction-based method [7]
Input	Input two dimensional (2D) images and data	Takes data from the acceleration sensor	Static and dynamic images
Technique	Image processing and machine learning	Calculates depth of the pothole after detection	Predicts shape of the pothole Uses stereo vision technology and measures the volume after detection

Table 1-1 summarizes the three most popular pothole detection methods. All these three techniques offer their own advantages and some disadvantages. For example, the vibration-based method is the most cost-effective [7] out of the two others and it also requires the least storage. However, this technique's performance is affected by the sensor since it can get damaged due to the road conditions and the mode of transportation and is not always 100% accurate in providing the shape of the potholes. Moreover, it can detect false positives by analyzing a road joint as a pothole [7]. The vision-based method is not the most cost-effective technique, but it can determine the approximate shape of the potholes. However, it has

limitations over the two mentioned techniques, such as: calculating the volume and depth of the holes in the road and failure to accurately identify the pothole shape as a consequence of lighting effects. The last method, which is a 3D reconstruction-based technique, is the most accurate but at the same time, it is the most expensive since it uses a laser scanner and is in need of high computational requirements [7].

1.2.2 Pothole detection algorithms

In this section, we list some of the popular pothole detection algorithms. Table 1-2 shows various algorithms that are used for pothole detection. In different systems, a combination of multiple algorithms can be found. These algorithms include but are not limited to object detection, edge detection, and various other machine learning algorithms.

Table 1-2: Advantages and Disadvantages of Algorithms Used for Pothole Detection

Algorithms	Advantages	Disadvantages
You Only Look Once [8]	Identifies objects in images and/or videos	Difficulty identifying small objects Difficulty to detect close objects
K-nearest neighbor [9]	Does not require training New data can be added easily, without affecting accuracy, Easy to apply	Does not work well with large sets of data Easily affected by any outliers and/or missing data
Artificial neural network [10]	Optimizes the size of a network and the number of layers Greater speed in predictions after training	Long training time Fixed size Massive computational power

Baek et al. [8] implemented a pothole-detection system that uses object detection to record all objects. Then followed it by the YOLO algorithm that identifies potholes after all objects from the image are removed. To generate useful results, object detection algorithms frequently use machine learning or deep learning. Using a computer, object detection aims to simulate the way humans can quickly identify and pinpoint objects of interest when viewing photos or

videos. The KNN algorithm is used by [9] Ronghua Du et al. to separate the various types of abnormal pavement, including potholes and bumps in the road, and the modified Gaussian background model is utilized to extract the features of the abnormal pavement. Machine learning algorithms and neural networks have been used multiple times in projects to perform pothole detection. The algorithms implemented for these software and hardware devices will vary from one system to the next, but they all work on one key concept, identifying potholes in roads.

1.2.3 Pothole detection prototypes

In this section, we list various pothole detection prototypes. Table 1-3 shows three different prototypes that are compared in terms of algorithms used, hardware, and software.

Table 1-3: Pothole Detection Prototypes

Prototypes	Algorithm	Hardware	Software
Vehicles Potholes Detection based Blob method [11]	Blob detection - edge detection	Raspberry Pi 3G USB modem	Linux OS OpenCV Webpage for communication
Robot for Road Pothole Detection Using GPS [12]	Pothole classification	Ultrasonic and IR sensors A microcontroller (Arduino) Global positioning system Global system for mobile communication Motors	Arduino application C or C++
Abnormal Road Surface Recognition Based on Smartphone Acceleration Sensor [13]	Gaussian background model and KNN	Smartphone built in accelerometer Global positioning navigation system	Phone application for collection of data

In Table 1-3, the blob method-based prototype is a device that is mounted on the passenger vehicles that has the capability of capturing images continuously. When a pothole is detected

on the frame, the system will record the GPS coordinates and hence its location. In order to detect potholes, a blob detection [11] computation is performed on every frame taken. In this study, the Raspberry Pi is used as the main processor. Raspberry Pi is a low-cost Single Board Circuit computer. Raspberry Pi 2 has 1 GB of RAM which is sufficient to computer vision applications and communication tasks simultaneously. The robot in [12] prototype is one of the more expensive models designed. It requires many sensors, motors, an Arduino board, and a GPS. The Arduino board is not as performing as the Raspberry Pi. The smartphone acceleration sensor prototype [13] is the cheapest model and the least complex in hardware design. The algorithm used for this model though is more advanced and requires more computational power.

1.3 Project Proposal

Some techniques have already been implemented to try and reduce the number of potholes on the roads, such as online sites or applications but our goal is to design a low-cost auto-pothole detection and localization device. Contrary to the prototypes already out there, our devices will not be equipped with high-end processors, but with effective and capable processors which make it easy to deploy our device in bulk on any mode of transportation such as a car or a bicycle. As a result, our device will be a portable, low-power processor with a camera that detects these potholes and will automatically log its location with respect to the GPS coordinates to create a map of all potholes in Lebanon.

1.4 Design Criteria

The design will implement a machine learning algorithm to classify images and log the GPS coordinates of the potholes detected. It will consist of a microprocessor, a portable power source, a camera, and a memory. The camera is used to take pictures of the road in order to classify the image into two different categories: with pothole and without pothole. This design

needs to be inexpensive and portable, easily attached to different types of vehicles. The design is required to operate at a maximum voltage of 5V and to provide continuous runtime capabilities, i.e., last 24 hours on a single charge. It needs to have an accuracy greater than 90% to reliably classify images with potholes. Additionally, the GPS should be reliable enough and able to provide the current location within 30 seconds, including the refresh rate, down to +/- 15 meters. Finally, the prototype should be able to store the acquired image in a compressed format if a pothole is detected as well as its location.

1.5 Anticipated Outcomes

Convolutional neural networks that have operated on inputted user data will be implemented on TinyML, while using efficient, low-power usage microcontrollers that will be able to detect potholes on roads. High efficiency as well as capturing the picture of the pothole and logging its GPS coordinates to create a map of the potholes should be expected of the device that is being presented. The prototype will be powered by a battery and portable to be mounted on moving vehicles of different types (bicycles, scooters, cars, etc.). Once an image is acquired the device will detect whether it contains a pothole or not. Once a map is generated it can be linked to a satellite map to pinpoint pothole locations as a future project.

1.6 Scope of the Project

The project scope is defined by various restrictions to design an efficient pothole detection and localization system. These limitations can be divided into two major categories: general, technical, and realistic constraints.

1.6.1 General Constraints

This project focuses on designing and implementing a low-power pothole detection and localization device. Firstly, the main purpose behind it is to detect a pothole, capture an image, save it, and log the location coordinates. The prototype built is cost-effective, portable, and

user-friendly. A small-scale prototyping technique is used, where the production of the system is done in a reduced size. Moreover, to facilitate its transportation, a durable enclosure is required. Usability is also taken into consideration since the use of the device requires minimal effort while attaining the highest efficiency possible.

1.6.2 Technical Constraints

The pothole detection and localization system is shaped with respect to various technical constraints. First, it is a low-cost system that can be reproduced on both the hardware and software levels. Moreover, it could be expanded and improved. This system is a portable-size prototype based on a minimal power consumption processor, with continuous runtime capabilities; the prototype should function for a couple of days and operate at a maximum voltage of 5V, preferably at 3.3V. Since the prototype includes various components and lots of wiring, a durable enclosure is required to keep everything in place. This can also facilitate its transportation and keep the device safe. The global positioning sensor must indicate accurately the coordinates of the pothole it should be able to provide a valid location within 30 seconds and with a maximum error of +/-15 meters. To consider the system accurate and efficient, the classification precision must be greater than 90%. The system storage capacity should not be below 4GB and should be able to store the data for the full length of the experiment.

1.6.3 Realistic Constraints

The design's realistic constraints are divided into several categories:

1.6.3.1 Sustainability

To maintain sustainability, the plastic material used in this project is minimal. In addition to that, the power transmission protocols applied are low in power and in interference.

1.6.3.2 Manufacturability

The pothole detection and localization design is easy to reproduce. The hardware is not hard to build, the code can be rewritten, and the neural network can be trained. The design is done using open-source tools and platforms with easily available and replaceable components.

1.6.3.3 Health and safety

The project is safe for deployment and the user of this system is not exposed to any danger. This design does not require high voltage to run, therefore the risk of electric shocks is significantly low to zero. No hazardous material is used in designing the prototype.

1.6.3.4 Economic

The components used in this system are a low power microcontroller, a camera, a battery, a memory, and a GPS. After having a list of components that could be used for this prototype, obtainable and easy-to-find components were chosen to build this design. In addition to that, pricing was taken into consideration and expensive components were eliminated from the list. The design maximum budget is 100$.

1.6.3.5 Ethical

Since human intelligence and artificial intelligence are related, the trust between humans and the trust in artificial intelligence are analogous. The human trust was established based on formal or informal rules and laws that played a major role in the development of human morals and ethics. Therefore, trust in machine learning and artificial intelligence is based on obeying certain policies and laws. Moreover, these technologies must be powerful and secure enough to stand in front of attacks. The European Commission outlines seven key requirements for ethical, responsible, and trustworthy artificial intelligence. The first aspect is human agency and oversight,

where humans have control over any type of machine. The second factor is technical robustness and safety; these systems must be robust to prevent attacks or stand still facing hackers. The third requirement is privacy and data governance to avoid data breaches. Requirement number four is transparency: since algorithms make decisions, they must be interpretable and feasible to audit. The fifth factor covers diversity, fairness, and non-discrimination. The algorithms must be fair and unbiased, and not show any type of discrimination. Societal and environmental well-being represents the sixth requirement; these technologies must be eco-friendly and aim to improve society. The last factor is accountability. The party responsible for the actions, decisions, and predictions of the machine learning models should be known (producing company, society, or the user).

1.7 Gantt Chart

The Gantt chart below shows the time management of the project.

The Gantt chart includes the plan for the two semesters up till the completion of the project. Both the work done during Fall 2022 and Spring 2023 can be found in Appendix A.

1.8 Conclusion

A pothole is a hole in a road surface that can lead to significant damage if left unpaved. Despite the availability of applications and online forms, reporting remains minimal. The pothole detection and localization system designed in this project intends to detect a pothole, capture a picture, and save its coordinates to facilitate the reporting process. The last decade has witnessed various systems and prototypes able to identify and report potholes. However, these devices are expensive and not easily accessible. The system designed in this project relies on computer vision and artificial intelligence. It is based on a deep learning algorithm, a convolutional neural network.

CHAPTER 2

Literature Review

This project will employ a machine learning-powered prototype to detect potholes. The project runs a pre-trained convolutional neural network that inputs images to spot potholes and record their position. It will rely on artificial intelligence to extract unique image features from a given snapshot and decide whether it includes potholes or not. Our design is based on this neural network with supporting hardware. In this chapter, we will provide an introduction on neural networks, the different types, and an overview on embedded machine learning.

2.1 Artificial Intelligence, Machine Learning, and Neural Networks

The ability of machines to learn from experiences is known as artificial intelligence. The primary objective of AI is to create self-sufficient machines capable of thinking and taking actions/decisions similar to humans [14].

Figure 2-1: Relationship between AI, ML, and NN

While machine learning is a subset of AI that assists in the innovation of AI-driven applications, neural networks are a part of machine learning that utilizes large sets of data and advanced algorithms to train a model [14]. Figure 2-1 shows the relationship between the three terms.

2.2 Neural Networks

Deep learning techniques are built on neural networks, often known as artificial neural networks [14]. Each node layer in an ANN has an input layer, at least one hidden layer, and an output layer. Each node is interconnected and has its own weight and threshold. Each node implements its own linear regression model, made up of data, weights, a bias term, and a nonlinear activation function. The linear combiner formula is as shown in (2.1):

$$\mathbf{w}^T\mathbf{x} + \mathrm{b} = \sum_{i=1}^{m} w_i x_i + b \tag{2.1}$$

- The input vector **w** is the weight vector
- The input vector **x** is the input vector
- The variable m represents the total number of samples
- The subscript i is the index of a selected sample x or w from the input and weight vectors, respectively.
- The variable b is the bias

An activation function is known as the transfer function, it is used to get node output [15]. The more commonly used activation function is the non-linear since its nonlinearity helps the model to adjust to different types of data. A couple of nonlinear functions include the sigmoid function shown in equation (2.2), the hyperbolic tangent function shown in equation (2.3), and the Rectified Linear Unit function shown in equation (2.4).

$$\mathrm{S}(z) = \frac{1}{1+e^{-z}} \tag{2.2}$$

$$\tanh(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})} \quad (2.3)$$

$$f(z) = \max(0, z) \quad (2.4)$$

The most basic neural network, as well as the oldest, is the perceptron. It is composed of a single neuron. In Figure 2-2, there is a diagram of a simple neural network formed of 5 neurons in a three fully connected layers structure.

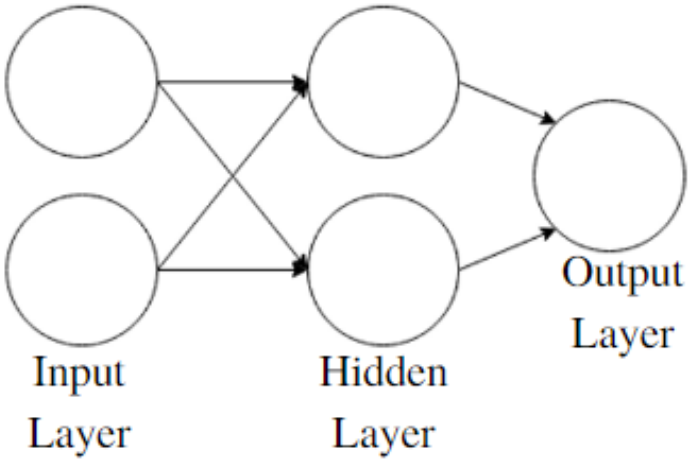


Figure 2-2: A simple fully connected neural network

There is the Loss function as well. This function is computed during the training phase and is used to evaluate the networks accuracy and control its optimization [15].

$$L(y, \hat{y}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \quad (2.5)$$

- y stands for the output we want or data label
- $\hat{y}$ stands for the output we received or predicted output

Training a neural network entails computing the best neuron weights and biases in an effort to minimize the loss function. This is done with respect to a feedback mechanism known as Gradient Backward Propagation. Training ANNs may be thought of as a function

optimization problem in which the goal is to find the optimum network variables that will reduce any error on a training dataset between the predicted and original value. Because it requires continuously assessing an objective function with multiple parameter values while iterating over a large data set, this training procedure is costly [15].

The next phase is the inference phase. This step is usually done offline after the training phase is complete. In this phase, the data is sent into the network and the output is defined using the rules set by our training phase. The data sets are usually divided between the training phase and inference phase, this kind of division could be an 80% to 20% division respectively. That is done to train our model using the larger chunk of data and then go on to test the last 20 percent using the rules the network has achieved. Neural networks are commonly found in various applications, such as classification, recognition, and detection.

2.3 Classification, Recognition, and Detection

In order to identify objects in an image or video the use of object recognition is required. Object recognition is a method used to spot objects in different forms of media. It is considered a key application of machine learning. The main concept is to teach machines to identify the various parts of an image as an average human would. Figure 2-3 shows us the relationship between object recognition, image classification, and object localization. While in image classification, a subset of object recognition, the system takes an input image and outputs the classification label. Another subset of object recognition is object localization [16]. This is an algorithm that finds the existence of an object in an image and outlines it with a bounding box. Object localization also takes an image as input and returns the location of the box. A combination of the two, image classification and object localization shown in Figure 2-3, make up the algorithm known as object detection. Object detection takes the input image and outputs the bounding boxes with each class label corresponding to the proper object [16].

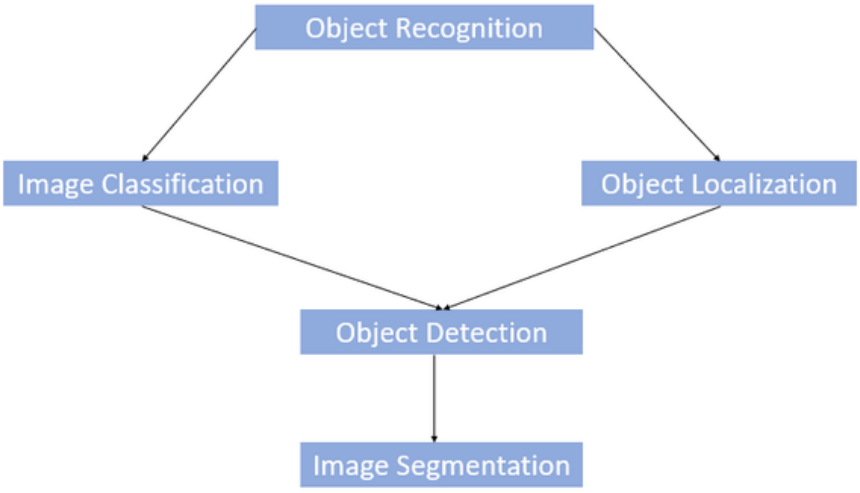


Figure 2-3: Object recognition tree [16]

In both the training and inference stages, neural networks rely on feature extraction. Features are unique parameters computed from a given input signal or image using classical signal and image processing algorithms. Feature extraction is a difficult and time-consuming process heavily dependent on extensive algorithms. However, with a goal to automate this process during the training phase and on raw input data, deep neural networks are employed.

2.4 Deep Neural Networks

Neural networks can be divided into multiple categories such as traditional and deep neural networks. What differentiates between these two networks is their depth i.e., the number of hidden layers.

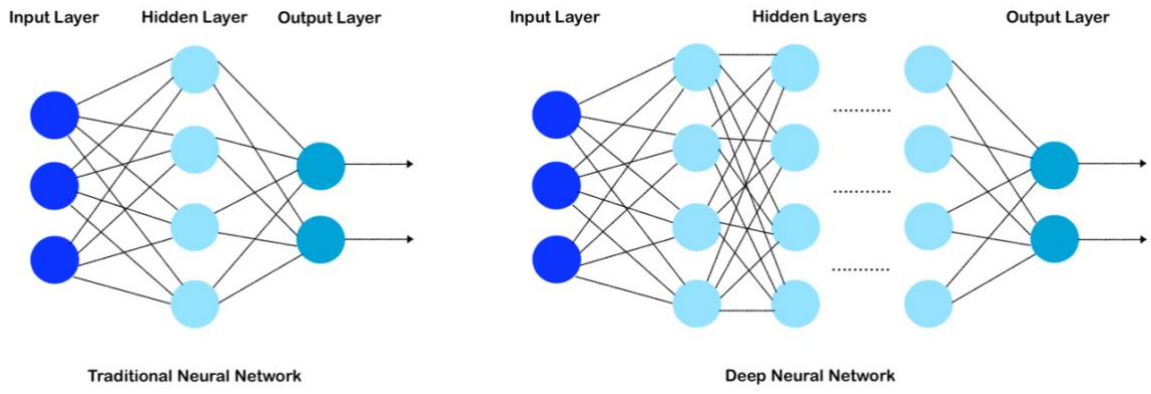


Figure 2-4: Traditional Neural Network vs. Deep Neural Network

As shown in Figure 2-4, while deep neural networks have a multi-layer architecture, traditional neural networks have a single or few hidden layers. Thus, DNN represents an improved version of the traditional neural network, characterized by its ability to learn and perform complex computations and operations [17]. Additionally, DNNs are used to perform automatic feature extraction on raw input data. Artificial Neural Networks, Recurrent Neural Networks, and Convolutional Neural Networks are three popular types of deep neural networks. At the inference stage, the inputs are only processed in a forward direction; this process is called Feed-Forward. ANN is inspired by the neurons in the human brain to make a machine operate in a human-like manner [18]. Using sequential data, RNN processes temporal information such as language translation, image captioning, and speech recognition [19]. CNN uses mathematical principles to perform video and image processing tasks such as object detection, image classification, etc. [20].

Deep learning is a subset of machine learning that consists of several techniques for learning in deep neural networks and training a model. Converting raw data into numerical features is called feature extraction. This process preserves the information in the original data set. Automated feature extraction does not require any human intervention since it employs automatically deep networks and algorithms to extract features from images or signals [21].

2.5 Convolutional neural networks and operations

Convolution is an operation specified by the user that allows the filtration of a specific dataset which would then produce a feature map that has multiple dimensions.

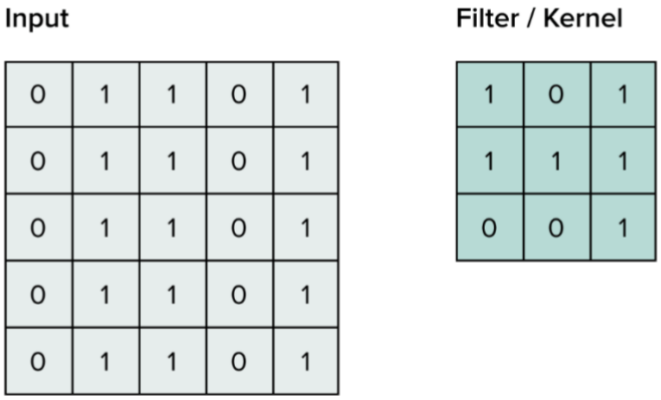


Figure 2-5: Filtered dataset [22]

The convolution operation is as follows [22]:

$$(f * g)(x) \triangleq \sum_{i=-\infty}^{\infty} f(i) \cdot g(x - i) \tag{2.6}$$

- $(f * g)$ is the function that is being convoluted
- x is the real number variable of functions f and g
- i is the index

To perform this convolution, the filter, which is also called the feature detector, analyses the input data, performs mathematical operations on it, and gives the filtered data set which is also the feature map [23]. The kernel iteratively traverses the input picture, doing matrix multiplication. The outcome for each receptive field is recorded in the feature map. A convolutional layer has many filters, each of which creates a filter map of its own. As a result, the output of a layer will be a collection of filter maps piled on top of one another.

2.6 Convolutional neural network's structure

Convolutional neural networks are mainly composed of three parts, convolutional, pooling, and a fully connected layer [24].

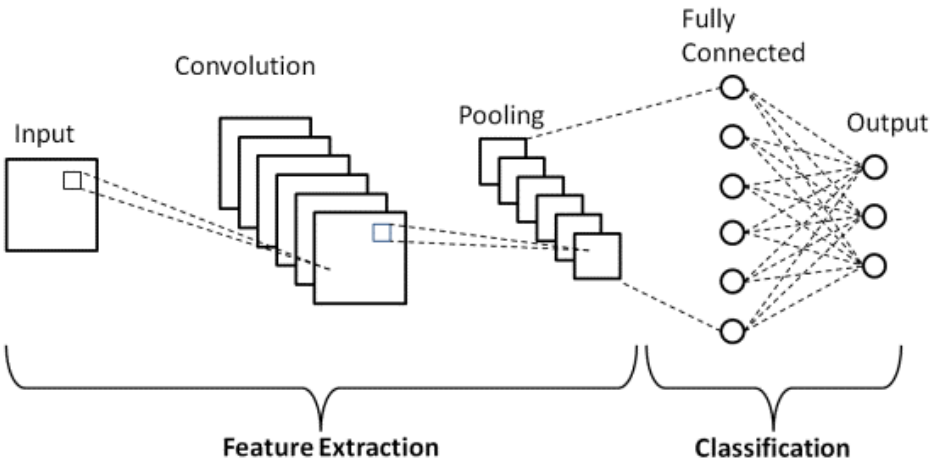


Figure 2-6: Convolutional neural network's structure [23]

- Convolutional layer: As explained in the part above, this layer is responsible for the computations that must be done on the inputted data set. It employs multiple filters with trainable weights to perform automatic feature extraction.
- Pooling layer: The main function of the pooling layer is to reduce the dimensions of the feature map explained before by reducing the number of parameters required and the number of computations needed [25]. By doing so it amplifies the values of interest.
- Fully connected layer: The fully connected layer is connected to both previous layers. Its' input is the output from the pooling layer. This layer is a classical neural network and is responsible for performing the final classification.

2.7 Implementing Convolutional Neural Networks on Embedded Devices

To implement these convolutional neural networks on embedded devices, i.e., TinyML, TensorFlow lite libraries are used so that microcontroller-based devices perform machine learning tasks.

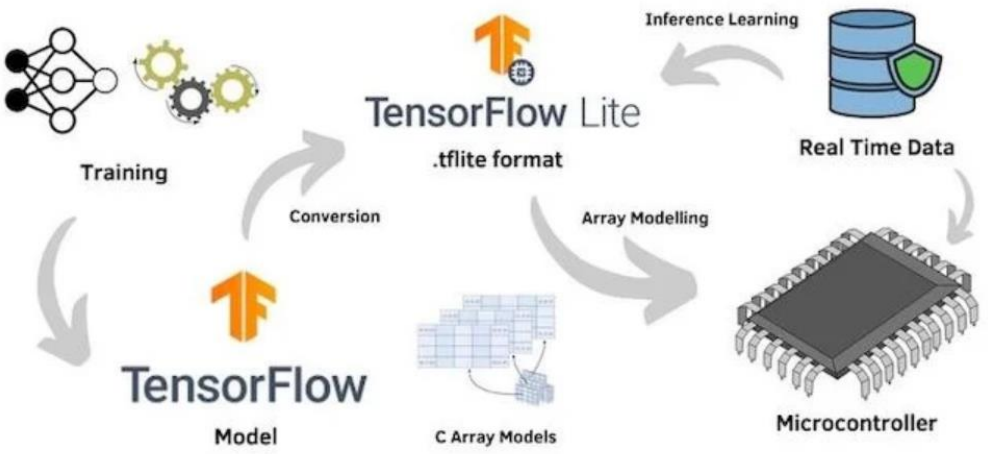


Figure 2-7: TensorFlow Light flowchart [26]

One of the many applications of TinyML is object recognition, which is what is needed for this project. The software that TinyML uses is TensorFlow lite, which will help in data acquisition and processing, so the inputted data will be analyzed by TensorFlow lite, as we can see in Figure 2-7 [26].

2.8 Pothole Detection Using TinyML

Table 2-1 presents two pothole detection systems based on tiny machine learning and a microcontroller.

Table 2-1: Comparison between Two Different Pothole Detection Prototypes using TinyML

Prototype	Hardware components	Tools
Pothole Detection using Edge Impulse's FOMO algorithm [27]	- OpenMV Cam M7	- Edge Impulse Studio
Sony Spresense Pothole Detection [28]	- Sony Spresense main board - Sony Spresense camera board - Sony Spresense LTE extension board - Flash Memory Card, SD Card	- Edge Impulse Studio - Arduino IDE - Amazon Web Services AWS IoT - Google Maps

The design in [27] runs on a 45x36x30mm microcontroller board with low power consumption equipped with a camera, OpenMV Cam M7. This design uses the Edge Impulse algorithm to build the pothole detection system: Faster Objects, More Objects. Triggering the I/O pins (to capture pothole pictures, etc.) is programmed in a high-level Python script. The model is created and trained using Edge Impulse Studio. The first step is gathering enough data and images to feed the neural network pothole patterns: the pothole dataset is downloaded from Kaggle and converted to Edge Impulse format. Then, these images are uploaded to Edge Impulse Studio to start training the machine learning model. Once the pothole model is created, it is deployed on the microcontroller. In order to make the model run and display what it sees on a liquid crystal display without being connected to a computer, an LCD shield and a lipo battery are connected to the OpenMV Cam M7. The design in [28] runs on the Sony Spresense main board which a Sony Spresense camera board is connected to. A Sony Spresense long-term evolution wireless data transmission LTE extension board with integrated GPS is also connected to the main board, eliminating the need for a location sensor. The first step to build the system is establishing the LTE connection. Then, the Amazon Web Service Internet of Things core is used to connect to MQ Telemetry Transport service which allows the network device to publish to the broker. The development of the model is done using Edge Impulse. Unlike design [27] which follows a FOMO algorithm, design [28] uses image classification. The dataset is extracted from Kaggle and loaded into the software. The model was then deployed as an Arduino library and the LTE extension board is connected to the GPS satellites. Once a pothole is detected, the GPS coordinates are saved with the image, and the results are then broadcasted to the MQTT broker. The CSV file containing the coordinates is then loaded into Google Maps and represented on the map with blue dots. While design [27] achieves an accuracy of 65.5%, design [28] reaches an accuracy of 80%.

2.9 Conclusion

This chapter discussed the difference between artificial intelligence and neural networks. It explained in detail neural networks, convolutional neural networks, and their formulas and described the differences between their design. Our project is established on a convolutional neural network thoroughly explained in this chapter to understand the coming chapters. In the next chapter, we will discuss the design's different components and show the block diagram in full detail.

CHAPTER 3

Project Design

3.1 Introduction

Component picking is one of the most important tasks of the project. In this chapter, different components will be compared to help decide which ones are the most efficient and useable. Moreover, a block diagram design will be created using these components as well as listing the protocols and standards.

3.2 General Schematic

In this section, we will introduce the general block diagram of our design. In Figure 3-1, you can see the top-level architecture of the design.

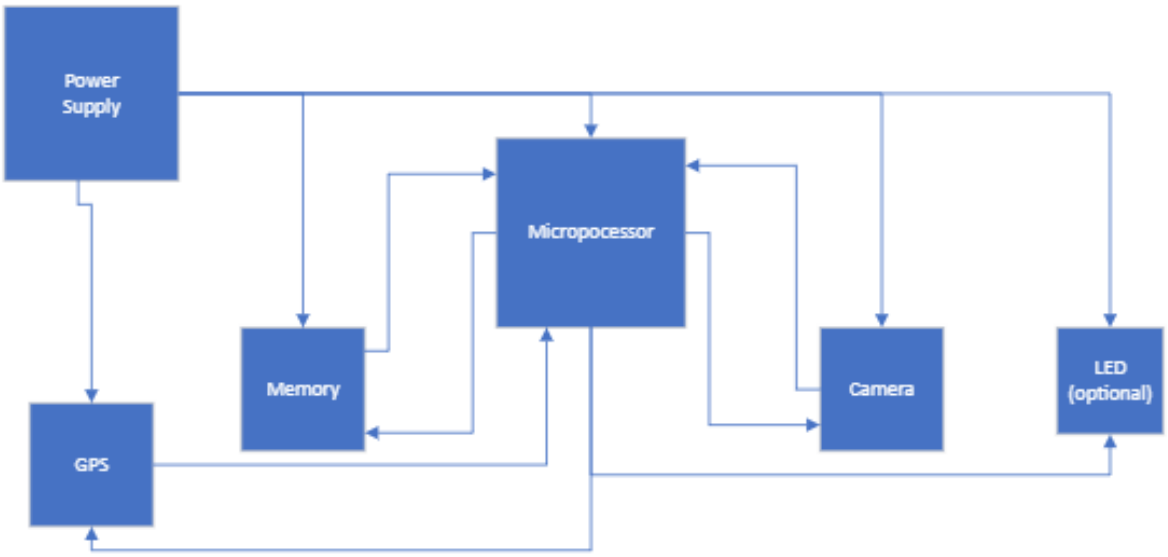


Figure 3-1: General Block Diagram

Figure 3-1 presents the connections to be made and the flow of data between the different parts. In this case, we assumed all power is being provided by the power supply. And all data goes into the microprocessor and comes out of the microprocessor. The components will be chosen based on the design criteria. In order to make sure that the design fits with our proposed idea. The microprocessor will be used to control the aspects of the project and do the

convolutions. The camera is used to capture any and all images. Furthermore, the GPS is used for marking where a pothole is detected. All the parts will work together to produce a working system that is lost cost, portable, and has high accuracy.

3.3 Component Selection

In this section, we will discuss the different components in more detail and compare each component with three different types to choose the best one for the project that meets the set design criteria and constraints.

3.3.1 Power Supply

Table 3-1 compares three types of batteries that are able to power the prototype.

Table 3-1: Comparison Between Three Types of Batteries

Features / Types	Cost ($)	Nominal Voltage	Capacity	Chemistry	Protocol
Duracell Button cell Battery [29]	1	3Volts	240 mAh	Lithium coin	CCCV charging
18650 [30]	3.5	3.7Volts	3000 mAh	Li-ion	CCCV charging
PAVAREAL Power Bank PB-80 [31]	15	5Volts	10000 mAh	Lithium Polymer	CCCV charging

The three power supplies follow a CCCV charging protocol. Duracell Button Cell Battery is a 1$ lithium coin with a nominal voltage of 3Volts. It is the cheapest among the three power supplies compared in Table 3-1 and has the smallest capacity (240mAh). The lithium-ion battery18650 has a voltage of 3.7Volts and 3000mAh capacity. It is slightly more expensive than the latter one with a cost of 3.5$. The PAVAREAL Power Bank PB-80 is a lithium

polymer power bank with the highest nominal voltage, capacity, and price. It has a 5Volts voltage, 10000mAh capacity, and costs 15$.

PAVAREAL Power Bank PB-80's nominal voltage is the maximum voltage that the prototype can operate at. Therefore, it is chosen as a power source for the project.

3.3.2 Microprocessor

A microprocessor is needed in order to run our network and give commands to other components for the detection to be possible. Table 3-2 shows three different microprocessors that can potentially be used for this project.

Table 3-2: Comparison Between Three Types of Microprocessors

Features / Type	Cost ($)	Processor	Power	Connectivity	Protocols	TensorFlow Lite Support
Arduino® Nano 33 BLE Sense [32]	42.95	64 MHz Arm® Cortex-M4F (with FPU)	I/O: 3.3v	Bluetooth® 5 multiprotocol radio	UART I^2C SPI	Yes
Raspberry Pi 4[33]	75.00	Quad core 64-bit ARM-Cortex A72 at 1.5 GHz	Input: 5 V/2.5 A DC Output: 3.3 and 5v	Wireless LAN Bluetooth 5.0	UART I^2C SPI	Yes
Arduino Uno WiFi R2 [34]	53.95	ATmega4809 16 MHz	DC current per I/O pin: 20 mA I/O voltage: 5v	Bluetooth Wi-Fi	UART I^2C SPI	No

Table 3-2 shows the different characteristics of three microprocessors. Comparing the different processors based on our design criteria, we can easily pick the Arduino Nano 33. This processor has the lowest cost which matches our criteria of low cost. It also supports TensorFlow Lite, and even though the Raspberry Pi 4 also supports it, it is more expensive than

Arduino. We also compared the power input and output in order to make sure we don't require a large input. The Nano 33 BLE Sense board has the lowest input voltage, and although does not provide 5V output, it is not necessarily needed. After comparing the three components, Arduino Nano 33 BLE Sense was chosen; it has a fast processor and enough power to handle our design.

3.3.3 Memory

A memory is needed to save the data detected by the prototype so a map of the potholes can be easily created. We compare three memory types in Table 3-3.

Table 3-3: Comparison Between Three Types of Memory

Features / Type	Cost ($)	Supports TensorFlow	Power	Size	Protocols	Power Consumption
CEG011000 [35]	0.90	Yes	4.5-5.5V	42x24mm	SPI	5-20mA
B09CPKSJ9M memory reader [36]	4.03	No	5V	14.99x10.92x 1.02mm	SPI	20-80mA
TenstarRobotTF [37]	1.2	Yes	4.5-5.5V	42x24mm	SPI	60mA

After comparing the three components, CEG011000 was chosen; it is the most efficient and the most logical memory that can be chosen. The second memory reader does not support TensorFlow so it cannot be used. The TenstarRobotTF is more expensive than the first memory reader and requires more power to function. CEG011000 supports TensorFlow and has the least power consumption out of the other two components and it is also the cheapest. Moreover, it is available on campus and an SD card can easily be implemented using this memory reader.

3.3.4 Camera

A camera is needed to take a picture of the pothole once detected. Table 3-4 shows three types of cameras that can be used in this project.

Table 3-4: Comparison Between Three Types of Cameras

Features / Types	Cost ($)	I/O Voltage	Power Consumption	Resolution	Output Format	Communication Protocol
OV3660 [38]	13.06	1.8V/ 2.8V	98mA	3MP	MJPEG/ YUY2	I^2C
OV5640 [39]	53.08	1.8V/ 2.8V	140mA	5MP	RGB565\ RGB555\ RGB444, YUV (422/420), YCbCr422, JPEG	I^2C
OV7675 [40]	6.94	1.71V ~ 3V	98mA	0.3MP	YUV422, Raw RGB, ITU656, RGB565	I^2C

The three cameras in Table 3-4 follow the I^2C protocol. OV3660 has a power consumption of 98mA and a 3MP resolution. The I/O voltage of OV3660 and OV5640 is the same, 1.8V or 2.8V, however, OV5640 has a higher price. While OV3660 costs 13.06$, the price of OV5640 is 53.08$. The latter's power consumption is 140mAh and its resolution is 5MP. OV7675 is the least expensive camera module with the lowest I/O voltage and resolution. It costs 6.94$ and has a 0.3MP resolution as well as an I/O voltage between 1.71V and approximately 3V.

After comparing the three components in Table 3-4, the OV7675 camera module was chosen since it meets the design criteria of this project. It has a low power consumption, it's cost-effective, and is available at the university.

3.3.5 GPS

A GPS is needed to track where the pothole is when detected by our system. We compare three different GPS modules in Table 3-5.

Table 3-5: Comparison Between Three Types of GPSs

Features / Type	Cost ($)	Power	Accuracy	Size (mm)	Protocol
NEO-6 Ublox [41]	20	Power consumption 111mW at 3.0V~37mA	Position 2.5m CEP 0.2 s refresh rate	12.2x16.0x2.4	UART SPI
GlobalTop Titan X1 [42]	44.95	GPS: min / typical / max: 20mA / 25mA /35mA	Without aid: 3m (50% CEP) 0.1 s refresh rate	12.5x12.5x6.8	GGA
MAX-M10S [43]	44.95	Current Consumption: ~6mA to ~25mA	Position GPS: 2m CEP 0.1 s refresh rate	10.7x9.8x2.7	UART I^2C

The three GPS components shown in Table 3-5 have good accuracy. When looking at our design criteria we can easily tell that the NEO-6 Ublox fits best with our budget. NEO-6 Ublox was chosen, it fits our design with a small module, low cost, low power consumption and has an accuracy that is greater than our minimum. When comparing all three, one thing to keep in mind is that all had multiple forms of tracking, all were small in size; as well as none needed high power. So, the two biggest things that played a role in choosing the appropriate module were cost and accuracy. The chosen module records GPS coordinates with a +/- 2.5meter error, less than +/- 15 meters specified in the design criteria.

3.4 Full Schematic

In Figure 3-2, the design schematic is redrawn with all chosen components.

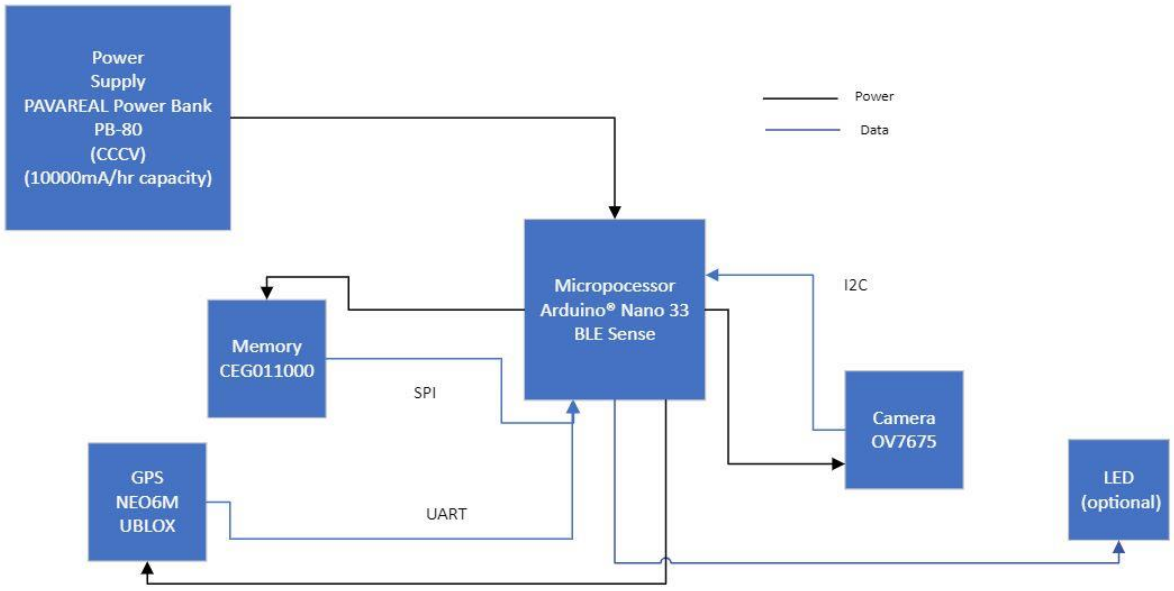


Figure 3-2: Full Schematic Design with All Official Parts

Figure 3-2 shows us all the components, their connection, and protocols used. The LED is optional since there is an on board if we wish to tell the user that a pothole has been detected. Otherwise, the data will be stored in memory only. The microprocessor gets power from the power supply while all the other components get power from the microprocessor. It being able to provide 500 mA of power, the camera, memory, and GPS use at most 150 mA, shown by the black connections. The protocols used by each part is also on the figure connected to the blue lines, lines of communication.

3.5 Protocols

The protocols used by the components in our design are I^2C, UART, and SPI. The connection and communication between the microprocessor and the GPS are linked via the SPI protocol. The camera's protocol is the I^2C [44]. I^2C transfers data in the form of messages by dividing it into data frames. I^2C is a serial protocol; data is sent bit by bit via a single cable. Because I^2C is synchronous, the bit output is synced to the bit sampling by a clock signal.

UART is the protocol used to connect the Arduino board to the computer. It is a hardware communication protocol that uses asynchronous serial transmission at a variable speed. The UART interface delivers data asynchronously and does not utilize a clock signal to synchronize transmitter and destination devices [45]. SPI is an interface bus that is often used to transport data between microcontrollers and tiny peripherals such as sensors, in our case GPS and SD cards. It employs separate clock and data lines, as well as a select line to pick the device that is needed for communication [46].

3.6 Standards

Standards are rules that are decided upon by a recognized authority as the most feasible and appropriate current solution to a recurrent problem.

- ISO/IEC FDIS 10918-5 [47]: JPEG is an image compression standard. It is a form of lossy picture compression. For coding transformation, JPEG compression employs the Discrete Cosine Transform technique.
- ISO 6709 [48]: The exchange of coordinates describing geographic point position. It defines the coordinate representation, including latitude and longitude, to be used in data transfer.
- IEEE 754-2019 [49]: This standard provides interchange and arithmetic formats, as well as techniques for binary and decimal floating-point arithmetic.

3.7 Conclusion

This chapter discussed various components that can be used to design the pothole detection and localization prototype. The most suitable ones were chosen based on the characteristics of the component and the needs of the design. The next chapter will go over the software side of the project. This will be done by getting the data and training the model.

CHAPTER 4

Network Development and Testing

4.1 Introduction

In this chapter, we will thoroughly explain the neural network and how we built our machine-learning model. This chapter consists mainly of the things we learned in previous chapters that were applied to our actual project and not just theoretical work. We will discuss the block configuration, layers in our system, training the system, and the confusion matrix. Each section will be discussed in detail, and we will discuss the process we took to create the machine-learning model.

4.2 Dataset

Our project's main function is to detect potholes on the street or in images. We utilized a specific dataset from Kaggle, known as the Pothole Detection Dataset. We used this dataset because it has the type of data that would match the images that our camera would be taking. The dataset is split into two categories, pothole and normal. There are 329 images specifically labeled as potholes and the remaining 352 represent normal streets. These images were taken using a phone camera and then a human annotator tagged the images as either normal or with a pothole. These images were also looked at by another annotator to ensure the quality of the images and no bias. The dataset was partitioned randomly in our project into a training set consisting of 83% of the images and a test set with the remaining 17%.

Even though the dataset was carefully evaluated, there is always a small chance of biases that could limit the machine learning models built on it. For this reason, we made sure to check the dataset's quality, verified the labeling, and spotted any limiters in the data. This ensured that the chosen dataset would suit our project's needs.

This dataset helped facilitate the training and testing of our model. We were able to train and test our model using the downloaded images. One thing that we had to keep in mind though while finding the dataset was the ethical considerations with respect to data privacy. We ensured the dataset's source was credited. We also made to keep our work according to the criteria and constraints when establishing the proper dataset.

4.3 Network Structure and Block Configuration

Our machine learning model consists of many different layers that various mathematical operations on inputted data. Our model consists of 5 main layers, one input layer, and an output layer. The layers included in our project are usually found in CNNs. This type of block configuration is said to be effective in image classification tasks.

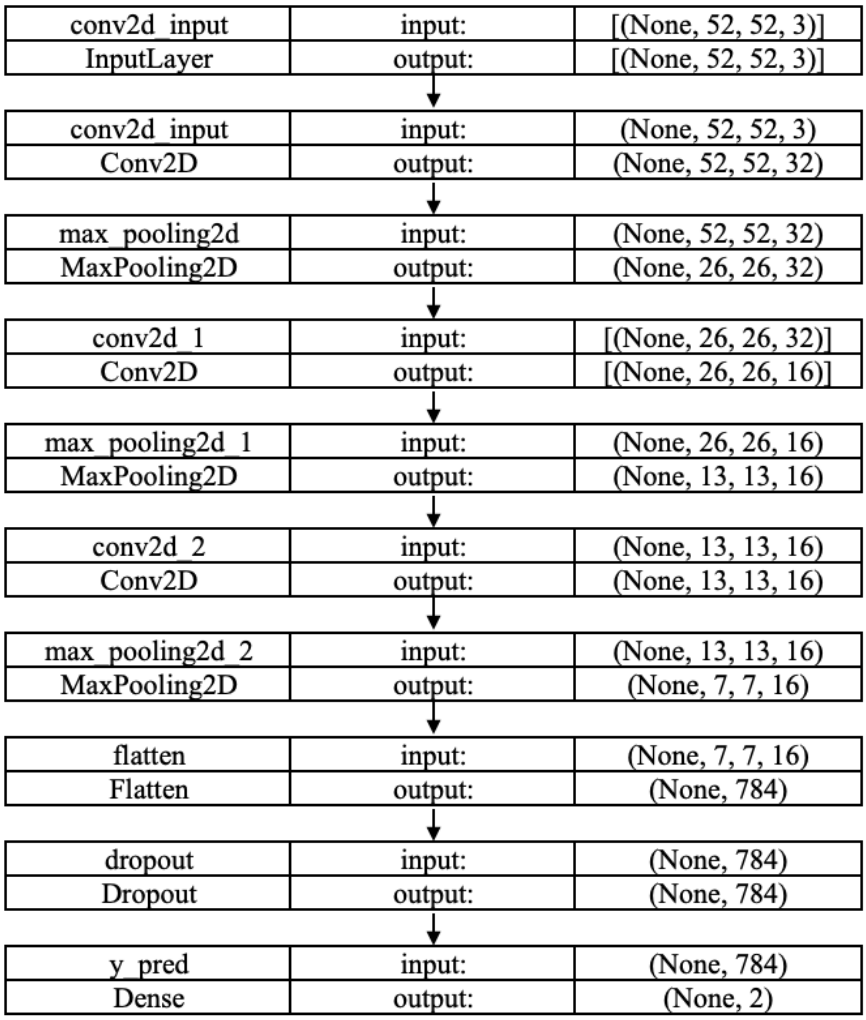


Figure 4-1: Network Architecture

The first layer, the input layer, is designed to take an image that is 52 * 52 RGB and feed it to the rest of the network. Following this layer are multiple 2D-convolution layers for creating feature maps and extracting data. Following each convolutional layer are 'pooling' layers.

The dimensions of the feature maps are even further decreased in the second and third convolutional layers. Each of these layers contains 16 filters with a kernel size of 3, which in turn take the extracted features and polish them. The convolutional layers used here will apply filters to the inputted data in which these filters will detect patterns or shapes in the image. We have multiple layers with many filters in order for our network to be able to detect more features. And our chosen kernel size will determine the region that is examined at a time with each filter.

The dropout layer is included in the network architecture with a rate of 0.25, to prevent overfitting. The technique used will randomly remove specific neurons during the training phase. With a learning rate of 0.0005, the system classifies between normal and pothole classes after the output has been flattened and had any dropouts removed. 100 cycles are employed during the training of the network.

The network is designed to take the image from the input layer and feed it to the layers below to detect the presence of a pothole. The block diagram of the neural network can be found in Figure 4-1. The diagram shows the dimensions, batch size, and channels in each layer. In the first convolutional layer, the layer input is (None, 52, 52, 3), which signifies that it has a shape of (batch size, height, width, channels). The pooling layers between each convolutional layer are used in order to downsize the feature maps that are created by the convolutional layers to be able to extract the most important features. Max Pooling 2D is frequently employed in

neural networks to improve translation invariance, which allows the model to detect features no matter where they are in the picture. The output consists of two classes, pothole and normal.

4.4 Image Size and Color Model

In this project, a 52x52 image (width and height) was chosen, which provided us with an image of 2,704 pixels, which is small compared to higher-resolution images but still provides enough detail for the model to learn relevant features. In this case, whether there is a pothole or not. Using larger images was not an option since our microprocessor didn't have the capability to process better-quality images. The fit shortest axis option was used to automatically resize all images to the same 52x52 dimensions. This is important so that the model can learn more effectively from the data inputted and it can help reduce memory and processing requirements. This all results in the classification of the images in two parts: normal images (images not containing a pothole) and pothole images (images containing a pothole).

For the parameters of the image, two options were available: grayscale images represent each pixel with a single value that gives the brightness of the pixel, rather than using three values for red, green, and blue color channels like in RGB (red, green, blue) images. Unlike RGB images, using grayscale images is not effective at capturing important features and patterns because these types of images only capture one aspect of the image (brightness) rather than multiple aspects that can be captured in RGB images. Moreover, using grayscale images will lower the accuracy of the neural network. However, RGB refers to the number of bits used to represent the color of each pixel in an image that has three color channels which are red, green, and blue. The RGB color depth that was used is 224, which can represent over 16 million possible colors which will greatly increase the accuracy of the model.

4.5 Neural Network Layers

As discussed before, layers are a fundamental component of neural networks, enabling them to learn and make accurate classifications. Those layers are represented in Figure 4-2.

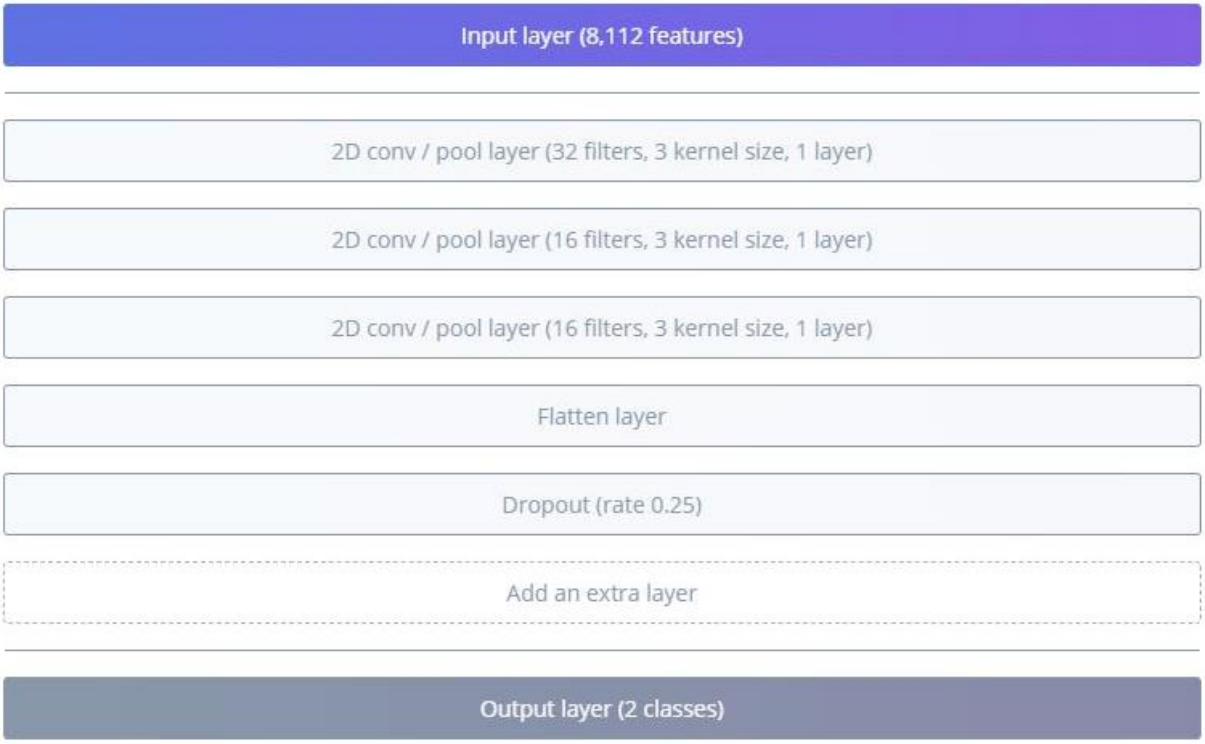


Figure 4-2: Network Layers

In this case, the input layer which is the first layer of neurons in the network is composed of 112 features and the batch size of the input data is 8. These numbers mean that the neural network is designed to process 8 data samples in each training iteration and 112 features in the inputted data. The second layer consists of a 2D conv / pool layer (32 filters, 3 kernel size, 1 layer). To improve the accuracy of the neural network, more than 1 pooling layer was used for better feature extraction. The "2D conv" part refers to the type of layer, which is a 2-dimensional convolutional layer in this case. It is designed to perform convolutional operations on the input data, to extract features from the input. "32 filters" means that the layer has 32 feature maps or filters. Each of these is a set of weights that is applied to the input data to

extract a particular feature. "3 kernel size" means that each filter has a kernel size of 3x3. The kernel is a sliding window that moves across the input data and performs a convolution operation at each position. "1 layer" means that there is only one instance of this type of layer in the network architecture. The flatten layer is used before the fully connected layer. Its main purpose is to transform the 2D images data to a 1-dimensional vector so in this case, the 52x52 image is transformed into a 2704-dimensional vector. This is important since the classification algorithm requires the input data to be a 1-dimensional vector.

To reduce the risk of the neural network memorizing the training data, a dropout technique is introduced. The rate is between 0 and 1, so for the case of this network, 25% of the neurons in the network will be dropped out during the training phase.

4.6 Network Size and Optimization

In order to enhance the pothole-detecting system's performance, this project uses model optimization. This technique works by preserving an acceptable level of accuracy while reducing the memory and processing power required. In the quantization approach of model optimization, the weights and activations of the neural network are represented as 8-bit integers rather than 32-bit floating point values. Table 4-1 compares two model optimization techniques.

Table 4-1: Comparison of Model Optimization Techniques

	Ram Usage	Latency	Flash Usage
Unoptimized (float32)	425.8K	1524ms	53.4K
Quantized (int8)	110.9K	476ms	36.0K

The RAM usage decreases from 425.8K before optimization to 110.9K after quantization; the quantized model uses approximately one-fourth of the memory used by the unoptimized

model. The Latency decreases significantly as well, from 1524ms in the unoptimized model to 476ms in the quantized model. Moreover, the quantized model uses less memory for storing on the Arduino Nano 33 BLE Sense. The flash usage decreases from 53.4K to 36.0K after optimization.

Therefore, the quantized model improves the performance of the system. It requires less RAM and Flash usage, which enables the system to better operate when deploying the model on Arduino Nano 33 BLE Sense. In addition to that, the quantized model's decreased latency allows the system to generate real-time predictions, enabling it to detect potholes rapidly and reliably. The quantized model was thus chosen so the pothole detection and localization system can be effectively deployed on a microcontroller with limited resources.

4.7 Training and Confusion Matrix

When training our pothole classification model, particular settings were employed to maximize the learning process. The model's performance may be considerably impacted by two key parameters. First, the frequency at which the training data is fed into the model during training is controlled by the number of training cycles. Moreover, the learning rate controls how much the parameters are altered during each iteration. The variables were selected based on testing and optimization to get the best performance while avoiding overfitting. These parameters are set to 100 training cycles and 0.0005 as the learning rate.

The validation set is a subset of the dataset that wasn't operated on during training. It assesses the model's performance on unobserved data and ensures that it does not overfit the training set. The confusion matrix is a table used in classification tasks to evaluate the performance of a machine-learning model, after passing the validation set to the model. For each class in the classification issue, it displays the proportion of accurate and inaccurate predictions provided by the model on a set of test data. It displays the percentage of predictions

that the model produced correctly and incorrectly on the validation set for each class (Normal, Pothole). Table 4-2 displays the confusion matrix of the system:

Table 4-2: Confusion Matrix

	Normal	Pothole
Normal	95.2%	4.8%
Pothole	3.9%	96.1%
F1	0.96	0.95

The confusion matrix displays the percentage of accurate and inaccurate predictions generated by the model. It includes true pothole classification (96.1%), true normal classification (95.2%), false pothole classification (4.8%), and false normal classification (3.9%). The model's total accuracy can be evaluated by the F1 score, which considers both precision and recall. The F1 score for the "Normal" class is 0.96, and 0.95 for the "Pothole" class. Therefore, 95.6% of the validation samples were properly categorized according to the model's accuracy on the validation set. The loss on the validation set is 0.43, which is the model's total inaccuracy in correctly predicting the class for each sample in the validation set. These values demonstrate the model's high level of accuracy in detecting normal road conditions, as well as roads with potholes.

The classification outcomes for the validation set and confusion matrix are represented visually in the data explorer in Figure 4-3. Data are shown as dots, with each dot representing a sample from the dataset. Each dot's color is determined by its expected class, and its position on the graph is determined by its features.

Figure 4-3: Data Explorer

The data explorer presents four clusters of dots based on the confusion matrix:

- Normal class classified correctly
- Pothole class classified correctly
- Normal class classified incorrectly
- Pothole class classified incorrectly

Some overlapping dots are spotted on the data explorer. These imply some degree of misclassification. The model's overall performance is assessed, and the potential improvement areas are determined by looking at the distribution of the dots and their spacing. The data explorer shows that the model can distinguish between the two groups since the data points for each class are separated by a decent distribution of dots. The overlap between one class's dots and those of other classes is hardly noticeable. This suggests that the model is effectively differentiating between the two classes.

4.8 Model Testing

Testing is the last step in the development process. It involves evaluating a machine learning model's performance with a set of untried data. This step includes testing the model on new data to see how effectively it generalizes and to ensure that it can predict results on new data. Table 4-3 presents a confusion matrix displaying the distribution of predictions among the classes. It was produced during the testing phase after the trained model was assessed on a batch of new pictures.

Table 4-3: Model Testing Results

	Normal	Pothole	Uncertain
Normal	87.9%	7.6%	4.5%
Pothole	10%	84%	6%
F1	0.90	0.87	

According to Table 4-3, 87.9% of the "Normal" photos and 84% of the "Pothole" photos were accurately identified. However, 7.6% of the normal" photos were incorrectly labeled as "Pothole”, and 4.5% were categorized as "Uncertain". Similarly, 10% of the "Pothole" photos were incorrectly categorized as "Normal," and 6% were categorized as "Uncertain". With an overall F1 score of 0.90 for the "Normal" class and 0.87 for the "Pothole" class during testing, the model performed well in differentiating the two classes. Overall, the model testing results show an accuracy of 86.21%. Figure 4-4 shows the feature explorer, which was developed based on the testing results.

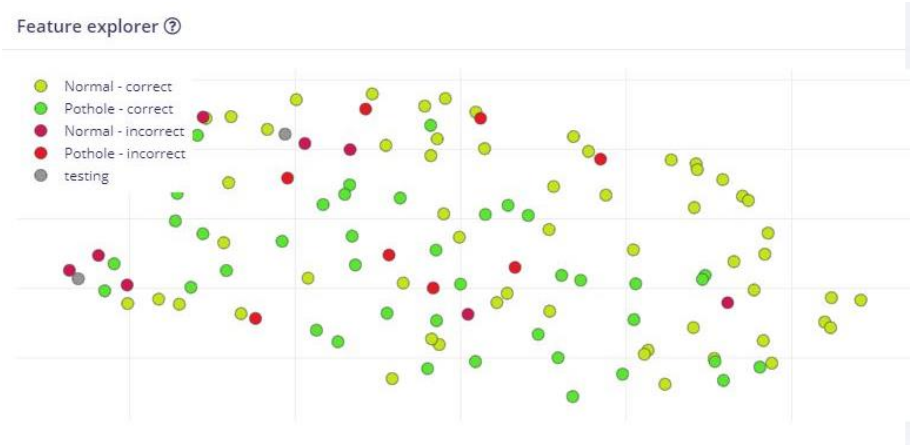


Figure 4-4: Testing Feature Explorer

Overall, despite some cases where the predictions were ambiguous, the model was able to differentiate between the two classes. Normal and pothole pictures were classified with a high level of accuracy. The uncertain or incorrectly classified images may be due to several factors, such as the lighting, the image quality, or other environmental circumstances. However, the F1 results show high performance for both classes, indicating that the model can be beneficial for detecting potholes in real-world situations.

Figure 4-5 shows the system's misclassification of a normal picture as a pothole. That may be due to the cracks in the road. This example's classification accuracy is 92%.

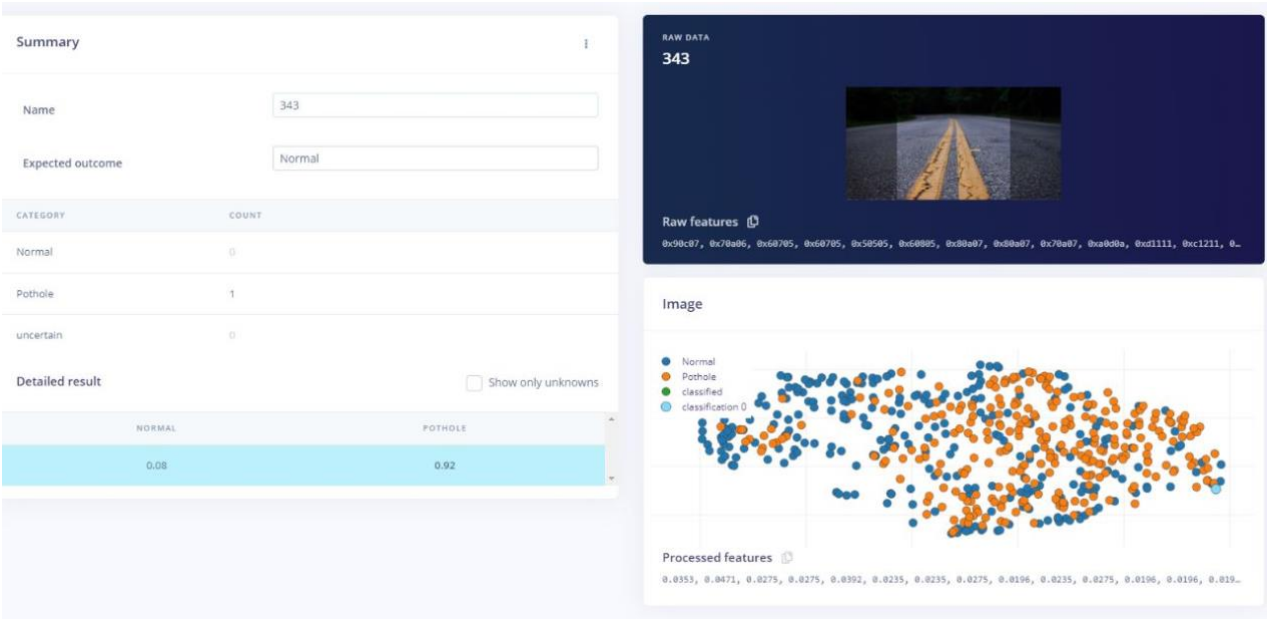


Figure 4-5: Incorrect Classification

Figure 4-6 shows an example of a correct picture classification as a normal road.

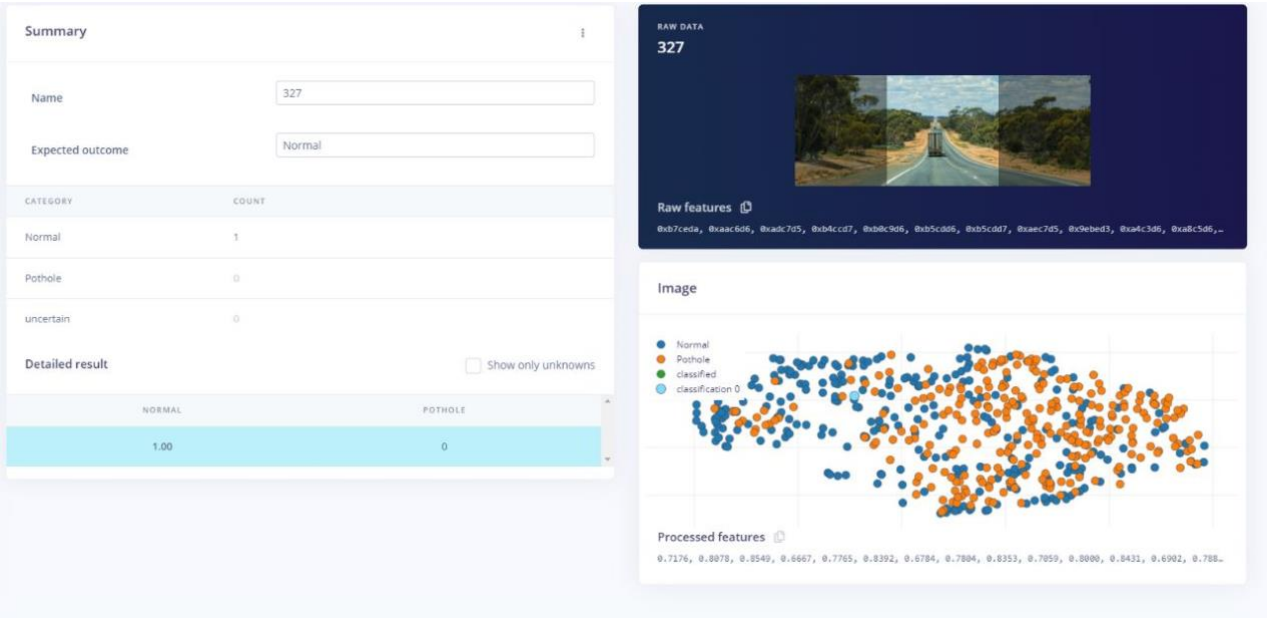


Figure 4-6: Correct Pothole Classification

Based on the features learned during training, the model accurately recognizes and classifies the road surface as "Normal". The classification accuracy, in this case, is 100%.

4.9 Conclusion

In conclusion, Edge Impulse is a powerful platform that creates machine-learning models implemented on microcontrollers. The parameters, block architecture, network structure, and dataset used to train the model affect the accuracy and efficiency of the model. The validation set as well as the confusion matrix offer cues for assessing the model's efficacy, while the data explorer visualizes the distribution of the data.

CHAPTER 5

Hardware Implementation and Field-Testing Results

5.1 Introduction

A key aspect of our project is creating a code that runs efficiently, without errors, and with high accuracy. In this chapter, we will illustrate how each part of the code for our project works; describe how each component is initialized as well as any other codes or processes needed for the completion of our project. Flow charts for the components, the main code, and the Python image viewer code will be provided. Other important aspects that will be discussed in this chapter are the results and analysis of our project post-testing.

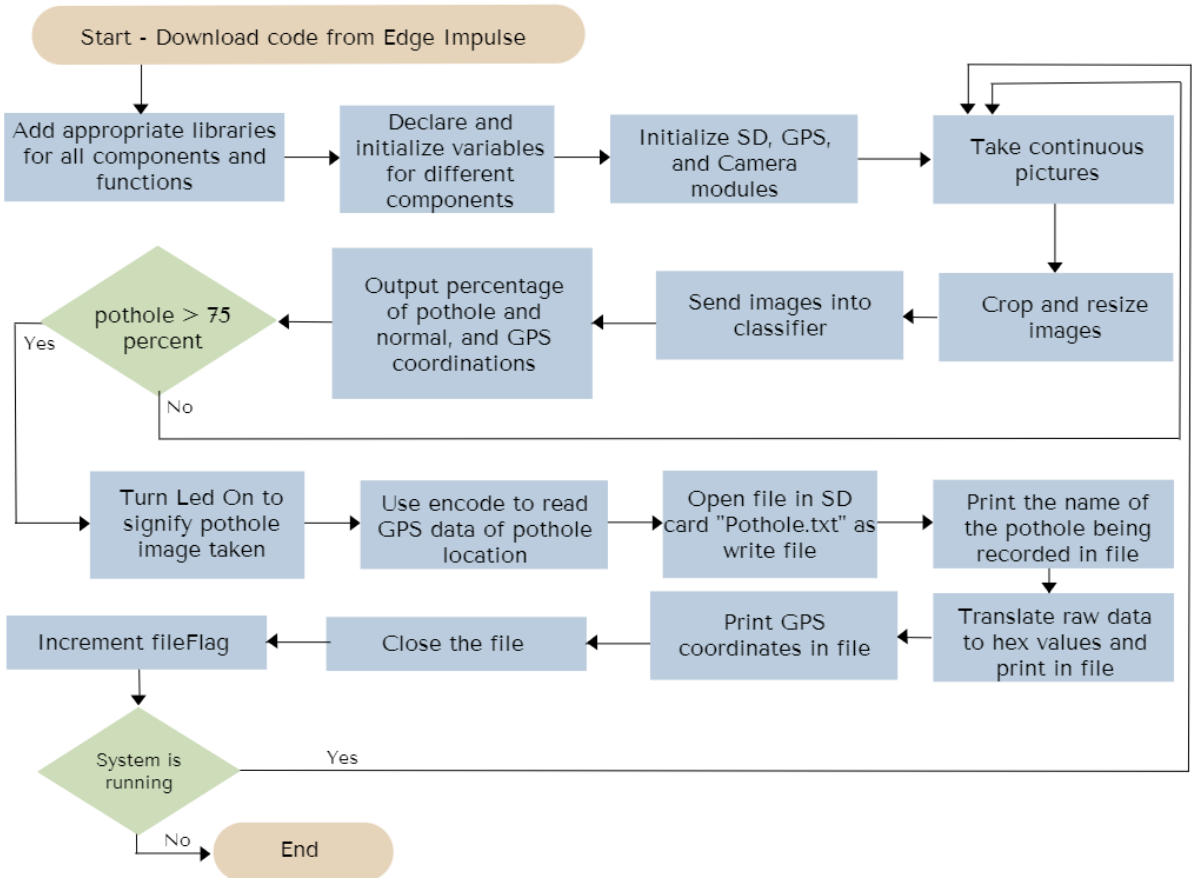


Figure 5-1: Main Flowchart

Figure 5-1 presents the main flowchart of the Arduino code, where the entirety of the project came together.

5.2 Main code

Although the main part of the code was created by our edge impulse model, the connection of the different components and the analysis and additions to the code was done according to our criteria and constraints.

As shown in Figure 5-1, the first step was declaring and initializing the appropriate variables. Next, the setup function was written to run once when the system is started. This consisted of initiating the camera, SD card, and GPS. The baud rate was set at a specific rate according to the GPS module we are using. The SD card also uses pin 7 for the chip select used for enabling and disabling the SD card.

After the setup was complete our code goes through a series of functions that were built into the file itself and outputs a classified result, pothole or no pothole. This is done when the camera takes an image, specified by the function ei_camera_capture, is resized, and sent into the edge impulse classifier, the output is then placed in an array.

If this output signifies there is a pothole, indicated by a value that is greater than 0.75, then we assume there is a pothole, the GPS reads Serial1 data and then the data and the hexadecimal values of the image are printed on a file in the SD card before closing the file and redoing the loop until the power is disconnected from the device. The full code can be found in Appendix B.

5.3 GPS

The flowchart explaining the code written for the GPS is shown in Figure 5-2. One of the initial steps in correctly programming the setup of the GPS module we are using was figuring out where each of the wires is to be connected.

One positive thing is that there is an RX and TX pins straight on the nano board. The GPS pins are connected ground to ground, as well as the Vcc on the GPS module to the 3.3 volts on

the microprocessor. The RX and TX pins on the GPS are connected to the TX and RX pins respectively on the Arduino board.

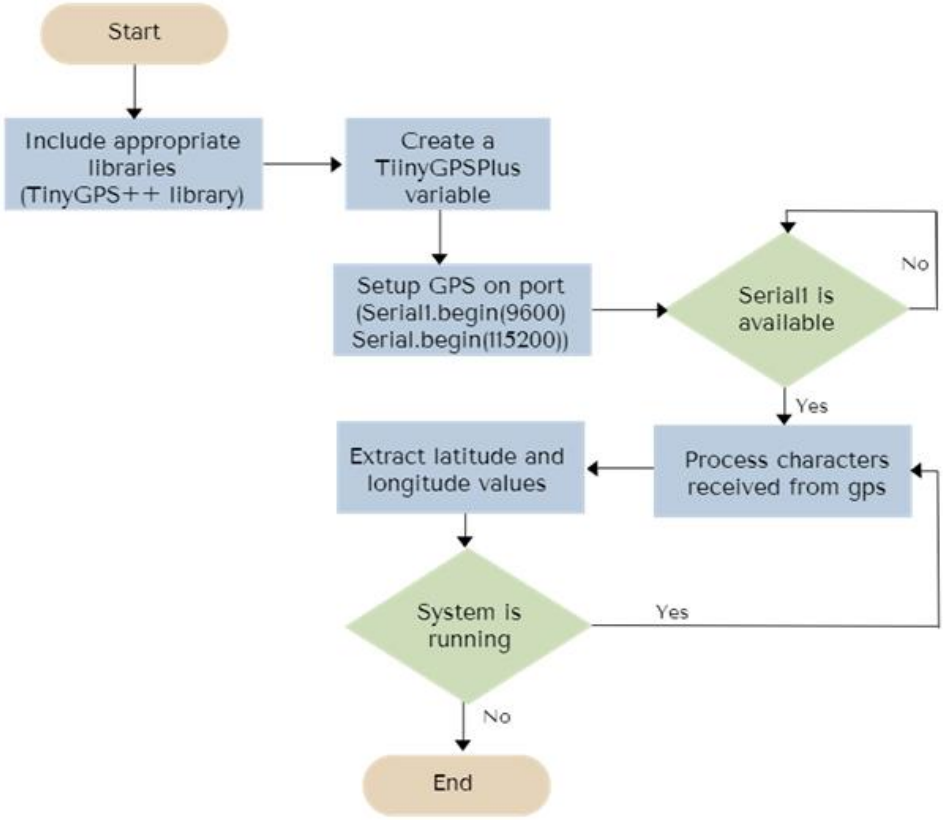


Figure 5-2: Flowchart Visualizing the GPS' Function

The code to setup the GPS sets the baud rate to 9600 for the serial communication between the board and the module. It then reads from Serial1 using the TinyGPSPlus library to encode the data gathered by the sensor to readable data. The GPS also needs to be placed in a viable place for connection or else the data read will be garbage.

5.4 SD Card Reader

After connecting the proper wires to the SD card, we initialized it to make sure it was working properly with our model, the correct partition(s) were made, as well as the correct formatting. Figure 5-3 shows the connections made to the Arduino Board.

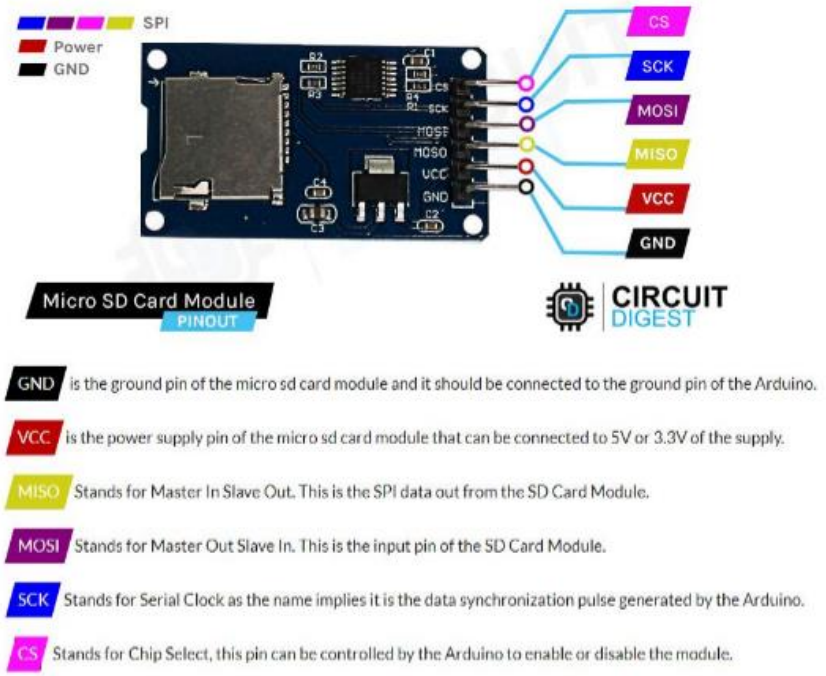


Figure 5-3: SD Card Reader Connections to Arduino [50]

The flow chart in Figure 5-4 signifies the process of initializing and working with the SD card.

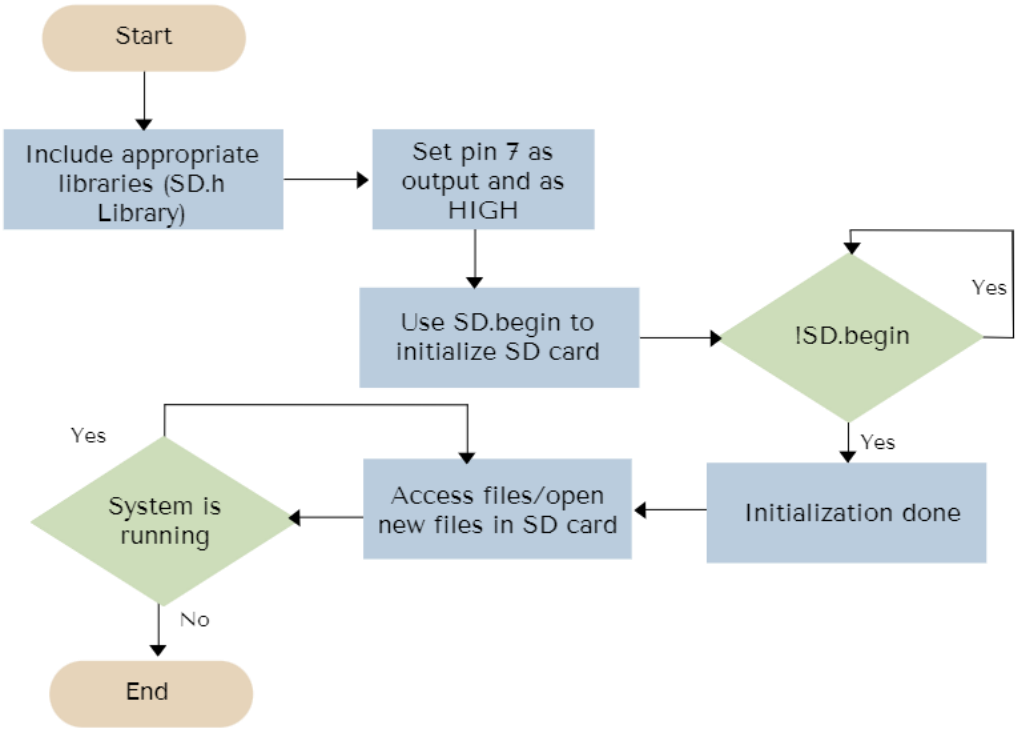


Figure 5-4: Flowchart Visualizing the SD Card Reader's Function

When the program starts, the SD card is setup using SD.begin() and setting its chip select to pin 7 on the Arduino board. Inside the code the SD card once initialized can constantly be accessed as long as the system is running. We can use the different SD card library functions to access the SD card, read and write to files.

5.5 Camera

The camera's wiring was the most complex of all the connections. Out of the 20 output pins we used 18 of them. The camera's connections are shown in Table 5-1.

Table 5-1: Connections Between the Camera and the Arduino Board [51]

Description	Camera Module Pin	Microcontroller Board Pin
VCC/3.3V	1	3.3V
GND	2	GND
SIOC/SCL	3	SCL/A5
SIOD/SDA	4	SDA/A4
VSYNC/VS	5	D8
HREF/HS	6	A1
PCLK	7	A0
XCLK	8	D9
D7	9	D4
D6	10	D6
D5	11	D5
D4	12	D3
D3	13	D2
D2	14	D0/RX
D1	15	D1/TX
D0	16	D10
NC	17	-
NC	18	-
PEN/RST	19	A2
PWDN/PDN	20	A3

After connecting the appropriate wires, we wrote the code to initialize the camera and start taking images. A flowchart explaining the code is shown in Figure 5-5. Our process was quite simple, initialize the camera and then take the raw data and convert it into a readable format.

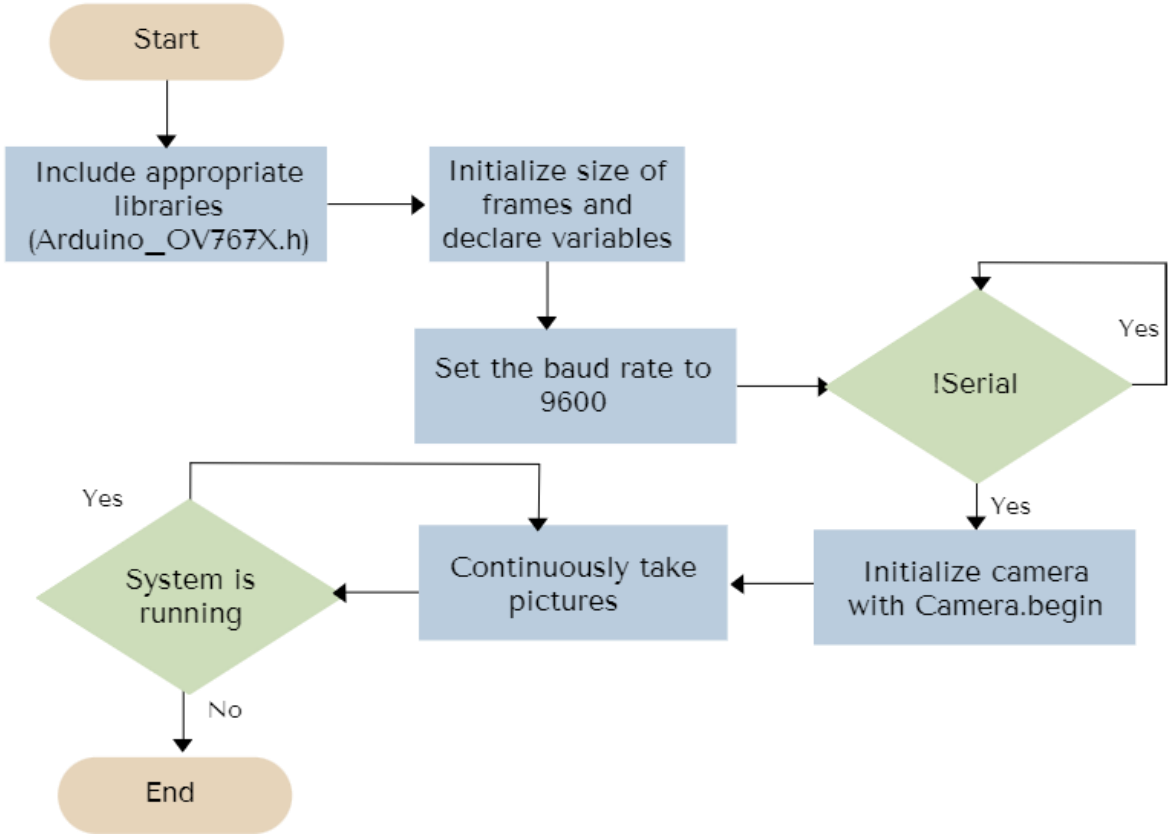


Figure 5-5: Flowchart Visualizing the Camera's Function

The camera will take images and these images, after specific resizing options, will be fed into our neural network.

5.6 Python Code to Retrieve the Pictures Captured

This code is used to convert the image data into an actual image as well as display the GPS coordinates on each image. The flowchart explaining the code is shown in Figure 5-6.

The code will read a text file containing the pothole information in order to count the number of potholes recorded, then go through the list of potholes to extract the hexadecimal values and convert the values to RGB pixel values and create a PNG image for each pothole.

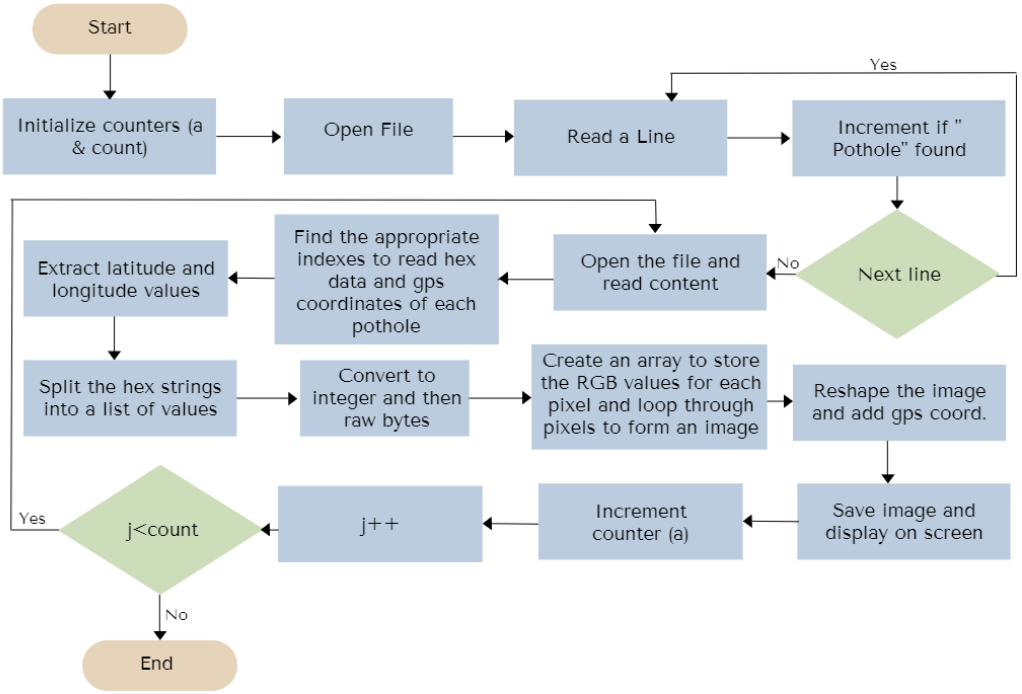


Figure 5-6:Flowchart Visualizing the Python Code

During this process, the program will also extract the GPS coordinates and save them in their appropriate variables. The image will have the latitude and longitude values displayed in the bottom right corner. This code then saves each of these photographs to a file with the pothole number as the name. The full Python code can be found in Appendix C.

5.7 Testing in Real-World Environment

Assessing the model's performance in real-world situations is crucial to ensure its accuracy and reliability. Many factors could have an impact on the system's performance since these factors differ from the conditions under which the model was designed and tested.

To assess the performance of the model in real-world circumstances, we walked on the street with the device. A camera, an SD Card, and a GPS module were connected to the Arduino microprocessor. The system was acquiring continuous pictures of the road while the car was driving around the city. Once a pothole was detected, the device took a picture, converted it to hex code, and saved it on the SD card along with the GPS coordinates.

After the data collection process, the data was retrieved from the SD card, and .png pictures were generated using Python code. Forty-two pictures were saved in total along with their GPS coordinates. Out of these pictures, 36 were correctly classified as “Pothole”, while 6 pictures were wrongly classified.

Figure 5-7 shows examples of the model's correct “pothole” classification.

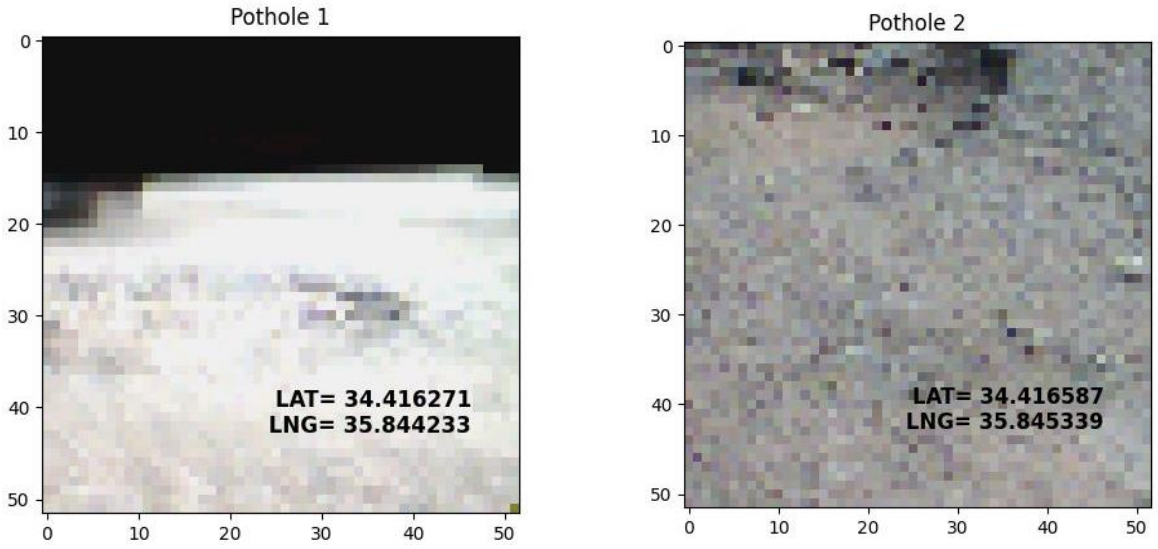


Figure 5-7: Correct Pothole Classification

Figure 5-8 shows examples of the model's misclassification.

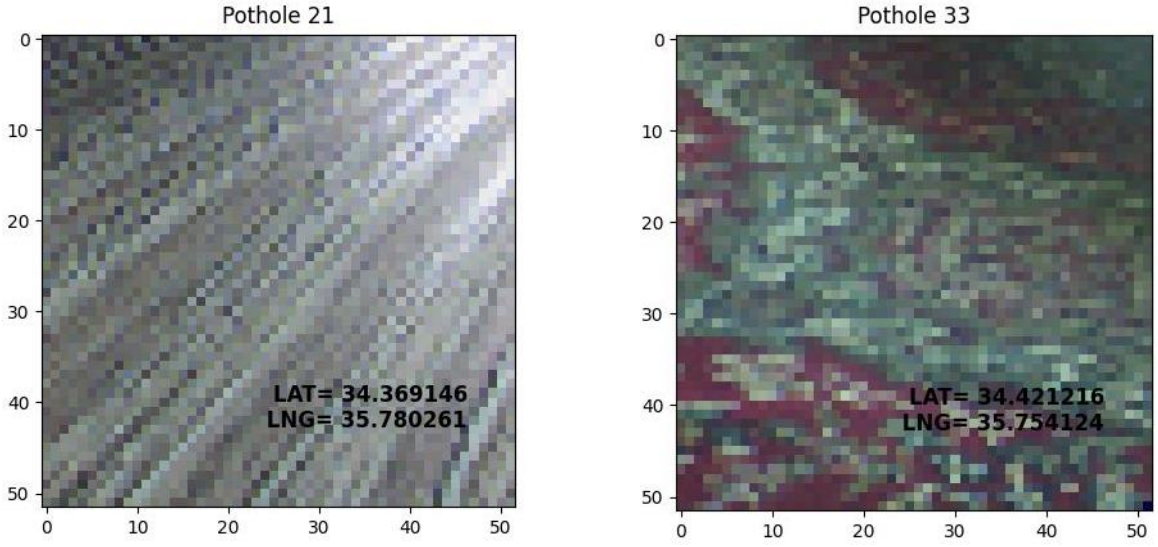


Figure 5-8: Incorrect Pothole Classification

Table 5-2 shows the testing results in percent.

Table 5-2: Testing Results in Real-World Settings

	Pothole
Normal	14.28%
Pothole	85.71%

With a misclassification rate of 14.28%, the system properly classified 85.71% of the pictures that were captured during testing. This indicates that the model's performance was consistent with the model testing on Edge Impulse (86.21%). The accuracy of the model's photo classification was used to evaluate its performance.

These results imply that the model is beneficial in pothole classification in real-world road settings. However, it is important to note that some external factors can influence the system's performance and therefore, affect its accuracy. It may be necessary to do more testing and optimization to maximize the efficiency and reliability of the model in real-world settings.

5.8 Conclusion

In conclusion, this chapter presented the flowchart explaining the Arduino code that runs the system. These flowcharts clarify the functions of the camera, the GPS, and the SD card reader. Moreover, the Python code that converts the hex of the image to a .png file and adds the GPS coordinates to it was also illustrated in a flowchart as well. The device was then tested, and the 42 pictures were captured and classified as "Pothole". After analyzing the results, an accuracy of 85.71% was detected. Overall, the technique was proven to be effective for finding potholes on the road and has the potential to be further developed for practices in road safety.

CHAPTER 6

Conclusion

6.1 Project Summary

This project attempted to assess the reliability and feasibility of a model that could detect potholes and save their coordinates each time a pothole was detected by the device. The work consists of six chapters. It proceeds with an introduction outlining the aim and the objectives then moves on to investigate neural networks and their implementations. Chapter three outlines the project design by describing the general schematic of the project and identifying the components that have been used such as the camera, and the type of batteries, along with the justification of the choice. Chapter four discusses network development, and its implementation and testing, whereas chapter five presents the hardware implementation and the testing of the system in a real-world environment. Finally, chapter six concludes the project, and discusses possible improvements.

6.2 Work Done

A brief summary of how the device works will be given in this paragraph. After researching the first few months to find out more about the topic picked, we finally gave our own proposal to create our own design. Additionally, we also took time to research neural networks to gain a better understanding of how our project will work and how accurate it will be. We were tasked to have an accuracy of more than 80% so we spent some time figuring out how to train our neural network to have the highest accuracy we can get. After that, our goal was to pick out the components which were most convenient for our design. After research and studies, we picked PAVAREAL Power Bank PB-80 as our power source, Arduino Nano BLE Sense as our microprocessor, OV7675 as our camera, CEG011000as our memory, and finally, NEO-6Ublox as our GPS. All selections were justified in chapter 3. Then, we moved on to the training of our neural network and its testing. After its configuration, we implemented the hardware and wrote

the code which allowed the camera to acquire continuous photos and of the road and classify them as a normal image or an image containing a pothole. The code will then allow our device to store the image of the pothole with its corresponding coordinates in a separate file. A Python code was used to retrieve the images from the SD card and add the GPS coordinates of the pothole detected on each picture.

6.3 Results

There are two main different types of implications emanating from this project: Theoretical and Practical.

On the theoretical front, this work extends modestly the literature concerning the use of neural networks to detect potholes. It builds on similar previous detection studies that use other methods to detect potholes such as the vehicles potholes detection based blob method, abnormal road surface recognition based on smartphone acceleration sensor, and robot for road pothole detection using GPS.

On the practical side, it is common knowledge that potholes present an expensive problem that both governments and individuals suffer from. From this perspective, coming up with a device to detect potholes can save large amounts of money, and presents a cost-effective method to remedy this problem. Additionally, results deriving from the use of this model can play a considerable role in setting future governmental plans, aiming at minimizing related physical risk and saving citizens' lives, as well as supporting the country’s economy.

The results we got from our project were what we wanted. With an accuracy of 85.71%, our neural network is able to classify an image containing a pothole or not, log its coordinates on a .txt file, and save its GPS coordinates on the picture.

6.4 Learning Experience

This project helped us better communicate and work as a group, since it was really challenging. Courses previously taken helped us a lot:

- "Instrumentation Lab" (ELEN201), "Electronics Lab" (ELEN304), and "Circuit Analysis Lab" (ELEN303) helped us with the practical part of the project.
- "Introduction to the engineering design fundamentals" (ELCP290) was extremely helpful when it came to designing Gantt charts and flowcharts. It also helped us better divide the work equally.
- "Microprocessors" (CPEN213) and "Programming for Engineering Solutions" (CPEN220) provided huge help for us when it came to the programming of our device, either for taking pictures with our camera or classifying whether this image contains a pothole or not.
- "Information Skills and Search Techniques" (LISP200) as well as English courses (ENGL203 & ENGL 204) were most helpful for our research and for writing the report.

Nonetheless, this project also helped us gain a better understanding of how to pick out components for a project, given its restraints and availability.

6.5 Future Work

Though this study presents a step forward in detecting potholes using image processing techniques, there are still several limitations that could be addressed in future research. Amongst these limitations is that the proposed algorithm was tested on a specific dataset; consequently, further validation using larger and more diverse datasets is needed. This would allow for a more vigorous assessment of the algorithm's functioning in diverse real-world scenarios.

There are many ways this project could be improved:

- While this work only focuses on classifying whether there is a pothole or not, future studies can explore ways to estimate more of their characteristics and attributes, in particular their severity and depth.
- Another thing we can do which can improve this project, is to save the data on the cloud, which would be easier for users to access.
- Linking our device to a location provider like GIS or Maps.
- A more complex thing we can do is design an application that can be linked to the location provider to further advance this project.

In conclusion, this project was effective in creating a device that is able to detect potholes using neural networks and image processing methods and log their GPS coordinates. The model can classify potholes with an accuracy of 85.71% and save the GPS coordinates of those images in a file. More features can be added in the future and some improvements can be made to improve the performance of the model. Nevertheless, the project is a significant step toward solving this pothole problem in Lebanon and has the potential to reduce physical risk and saving lives.

REFERENCES

[1] E. D. Elston, “Potholes: Their Variety, Origin and Significance,” in *The Scientific Monthly*, vol. 5, American Association for the Advancement of Science, pp. 554–567. [Online] Available: http://www.jstor.org/stable/22502/. [Accessed: Sep-2022].

[2] “Potholes: The cause, risk, and solution,” Claremont Asphalt, 16-Jun-2022. [Online]. Available: https://www.claremontasphalt.com.au/blog/potholes-the-cause-risk-and-solution/. [Accessed: Sep-2022].

[3] M. Bouchard, “The three dangers of potholes,” *Blackcircles.ca*, 13-Mar-2022. [Online]

Available: https://blog.blackcircles.ca/en/tips-guides/the-three-dangers-of-potholes/. [Accessed: Sep-2022].

[4] R. SALAME, “Behind the numbers: How Lebanon's crisis is felt on the roads,” *L'Orient Today*, 13-Jul-2022. [Online]. Available: https://today.lorientlejour.com/article/1305531/behind-the-numbers-how-lebanons-crisis-is-felt-on-the-roads.html. [Accessed: Sep-2022].

[5] “Whomp! see which Edmonton neighbourhoods complain the most about potholes | CBC news,” *CBCnews*, 08-Nov-2021. [Online]. Available: https://www.cbc.ca/news/canada/edmonton/whomp-see-which-edmonton-neighbourhoods-complain-the-most-about-potholes-1.6224551. [Accessed: 10-Oct-2022].

[6] K. R. Ahmed, “Smart pothole detection using deep learning based on dilated convolution,” *Sensors (Basel, Switzerland)*, 16-Dec-2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8704745/. [Accessed: 10-Oct-2022].

[7] Y.-M. Kim, Y.-G. Kim, S.-Y. Son, S.-Y. Lim, B.-Y. Choi, and D.-H. Choi, “Review of recent Automated Pothole-detection methods,” *MDPI*, 24-May-2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/11/5320. [Accessed: 10-Oct-2022].

[8] J.-W. Baek and K. Chung, “Pothole classification model using edge detection in road image,” *MDPI*, 23-Sep-2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/19/6662/htm. [Accessed: 15-Oct-2022].

[9] D. Jain, “KNN: Failure cases, limitations and strategy to pick right K,” *Medium*, 17-Jul-2020. [Online]. Available: https://levelup.gitconnected.com/knn-failure-cases-limitations-and-strategy-to-pick-right-k-45de1b986428?gi=5545ea3866f8. [Accessed: 15-Oct-2022].

[10] N. S. Gill, “Artificial Neural Networks Applications and algorithms,” *XenonStack*, 02-Sep-2022. [Online]. Available: https://www.xenonstack.com/blog/artificial-neural-network-applications. [Accessed: 15-Oct-2022].

[11] Djamaluddin, D., Parung, R., & Achmad, A. (2017, June 15). *PROTOTYPE OF VEHICLES POTHOLES DETECTION BASED BLOB DETECTION METHOD* . Retrieved September 25, 2022, from http://www.jatit.org/volumes/Vol95No11/18Vol95No11.pdf

[12] Mohamed, S. O. M. (2018, August). *Design and Implementation of a Robot for Road Pothole Detection Using GPS*. Retrieved September 26, 2022, from http://repository.sustech.edu/bitstream/handle/123456789/24062/Design%20and%20Implementation....pdf?sequence=1&isAllowed=y

[13] R. Du, G. Qiu, K. Gao, L. Hu, and L. Liu, “Abnormal road surface recognition based on smartphone acceleration sensor,” *MDPI*, 13-Jan-2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/2/451. [Accessed: 15-Oct-2022].

[14] S. M, “Discover the differences between AI vs. Machine Learning vs. Deep Learning [2022 edition],” *Simplilearn.com*, 13-Jul-2022. [Online]. Available: https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/ai-vs-machine-learning-vs-deep-learning. [Accessed: 17-Oct-2022].

[15] V. Bushaev, “How do we 'train' neural networks ?,” *Medium*, 22-Oct-2018. [Online]. Available: https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73. [Accessed: 17-Oct-2022].

[16] “Object detection vs object recognition vs image segmentation,” *GeeksforGeeks*, 28-Jun-2022. [Online]. Available: https://geeksforgeeks.org/object-detection-vs-object-recognition-vs-image-segmentation/. [Accessed: 17-Oct-2022].

[17] D. E. Kim, “COMPARISON OF SHALLOW AND DEEP NEURAL NETWORKS IN NETWORK INTRUSION DETECTION,” thesis, 2017. [Online]. Available: https://scholarworks.calstate.edu/. [Accessed: Oct-2022].

[18] B. Marr, “What are artificial neural networks - a simple explanation for absolutely anyone,” Forbes, 24-Sep-2018. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/09/24/what-are-artificial-neural-networks-a-simple-explanation-for-absolutely-anyone/?sh=69eaeea81245. [Accessed: Oct-2022].

[19] “What are recurrent neural networks?” IBM Cloud Education, 14-Sep-2020. [Online]. Available: https://www.ibm.com/cloud/learn/recurrent-neural-networks. [Accessed: Oct-2022].

[20] A. Pai, “Ann vs CNN vs RNN: Types of neural networks,” Analytics Vidhya, 19-Oct-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/. [Accessed: Oct-2022].

[21] “Feature extraction,” *MATLAB & Simulink*. [Online]. Available: https://www.mathworks.com/discovery/feature-extraction.html. [Accessed: Nov-2022].

[22] Gundersen, G. (n.d.). From convolution to neural network. Retrieved November 21, 2022, from https://gregorygundersen.com/blog/2017/02/24/cnns/

[23] Y. Gavrilova, “What are convolutional neural networks?,” *Serokell Software Development Company*, 03-Aug-2021. [Online]. Available: https://serokell.io/blog/introduction-to-convolutional-neural-networks. [Accessed: 16-Oct-2022].

[24] “Basic CNN architecture: Explaining 5 layers of Convolutional Neural Network,” *upGrad blog*, 03-Oct-2022. [Online]. Available: https://www.upgrad.com/blog/basic-cnn-architecture/#:~:text=other%20advanced%20tasks.-,What%20is%20the%20architecture%20of%20CNN%3F,the%20main%20responsibility%20for%20computation. [Accessed: 16-Oct-2022].

[25] “CNN: Introduction to pooling layer,” *GeeksforGeeks*, 24-Aug-2022. [Online]. Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/#:~:text=Pooling%20layers%20are%20used%20to,generated%20by%20a%20convolution%20layer. [Accessed: 16-Oct-2022].

[26] “What is tinyml? - technical articles,” *All About Circuits*. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/what-is-tinyml/. [Accessed: 18-Oct-2022].

[27] N. Shakhizat, “Pothole detection using Edgeimpulse's FOMO algorithm,” *Hackster.io*, 30-Jul-2022. [Online]. Available: https://www.hackster.io/shahizat/pothole-detection-using-edgeimpulse-s-fomo-algorithm-1df62d. [Accessed: Oct-2022].

[28] J. Lutz, “Sony SPRESENSE Pothole Detection!,” *Hackster.io*, 08-Aug-2022. [Online]. Available: https://www.hackster.io/justinelutz/sony-spresense-pothole-detection-75857e. [Accessed: Oct-2022].

[29] “25 button cell CR2032 Duracell,” *EKT*. [Online]. Available: https://www.katranji.com/item/339876. [Accessed: Nov-2022].

[30] “25 li-ion 3.7V 18650 2500mah tag,” *EKT*. [Online]. Available: https://www.katranji.com/item/215173. [Accessed: Nov-2022].

[31] “Pavareal Power Bank PB-80 - Gold Call,” Gold Call - We Deliver all over Lebanon, 04-Nov-2022. [Online]. Available: https://goldcall.co/product/pavareal-power-bank-pb-80/. [Accessed: Nov-2022].

[32] “Arduino nano 33 BLE Sense - Sparkfun Electronics.” [Online]. Available: https://cdn.sparkfun.com/assets/0/d/8/4/9/DS-15580-Arduino_Nano_33_BLE_Sense.pdf. [Accessed: 08-Nov-2022].

[33] DATASHEET - raspberry pi. (n.d.). Retrieved November 20, 2022, from https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[34] T. A. Team, “Uno WIFI REV2: Arduino documentation,” *Arduino Documentation | Arduino Documentation*. [Online]. Available: https://docs.arduino.cc/hardware/uno-wifi-rev2. [Accessed: 07-Nov-2022].

[35] “412 Arduino Module Micro SD Card,” *EKT*. [Online]. Available: https://www.katranji.com/Item/357843. [Accessed: 08-Nov-2022].

[36] “Amazon.com: Micro SD Storage Expansion Board Micro SD TF card memory ...” [Online]. Available: https://www.amazon.com/Storage-Expansion-Suitable-Arduino-Promotion/dp/B09CPKD53X. [Accessed: 08-Nov-2022].

[37] Alldatasheet.com, “TMPS05 datasheet(pdf) - traco electronic AG,” ALLDATASHEET.COM - Electronic Parts Datasheet Search. [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/1069877/TRACOPOWER/TMPS05.html. [Accessed: 21-Nov-2022].

[38] “OV3660 color CMOS GSXGA (3 megapixel) image sensor with OmniBSL technology,” *OmniVision,* May-2011. [Online]. Available: https://m5stack.oss-cn-shenzhen.aliyuncs.com/resource/docs/datasheet/unit/OV3660_CSP3_DS_1.3_sida.pdf. [Accessed: Nov-2022].

[39] “OV5640 Camera Board (A) USER MANUAL,” *Waveshare,* 12-Dec-2017. [Online]. Available: https://www.waveshare.com/w/upload/b/b8/OV5640_Camera_Board_%28A%29_User_Manual_EN.pdf. [Accessed: Nov-2022].

[40] “OV7675 CMOS VGA image sensor with OmniPixel3-HSTM technology,” *OmniVision,* Oct-2009. [Online]. Available:

https://www.uctronics.com/download/Image_Sensor/OV7675_DS.pdf. [Accessed: Nov-2022].

[41] “NEO-6 series - U-blox.” [Online]. Available: https://content.u-blox.com/sites/default/files/products/documents/NEO-6_ProductSummary_%28GPS.G6-HW-09003%29.pdf. [Accessed: 08-Nov-2022].

[42] “GNSS module Datasheet (tape-reel) - SparkFun Electronics.” [Online]. Available: https://cdn.sparkfun.com/assets/parts/1/2/2/8/0/GlobalTop_Titan_X1_Datasheet.pdf. [Accessed: 08-Nov-2022].

[43] “Max-M10s data sheet - SparkFun Electronics.” [Online]. Available: https://cdn.sparkfun.com/assets/7/5/9/a/a/MAX-M10S_DataSheet_UBX-20035208.pdf. [Accessed: 08-Nov-2022].

[44] S. Campbell, “Basics of the I2C communication protocol,” Circuit Basics, 14-Nov-2021. [Online]. Available: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/. [Accessed: 21-Nov-2022].

[45] E. Peňa and M. G. Legaspi, “UART: A hardware communication protocol understanding universal asynchronous receiver/transmitter,” *UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter | Analog Devices*. [Online]. Available: https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html#:~:text=By%20definition%2C%20UART%20is%20a,going%20to%20the%20receiving%20end. [Accessed: Nov-2022].

[46] “Serial peripheral interface (SPI),” Serial Peripheral Interface (SPI) - SparkFun Learn. [Online]. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all. [Accessed: 21-Nov-2022].

[47] “JPEG compression - javatpoint,” *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/jpeg-compression. [Accessed: 21-Nov-2022].

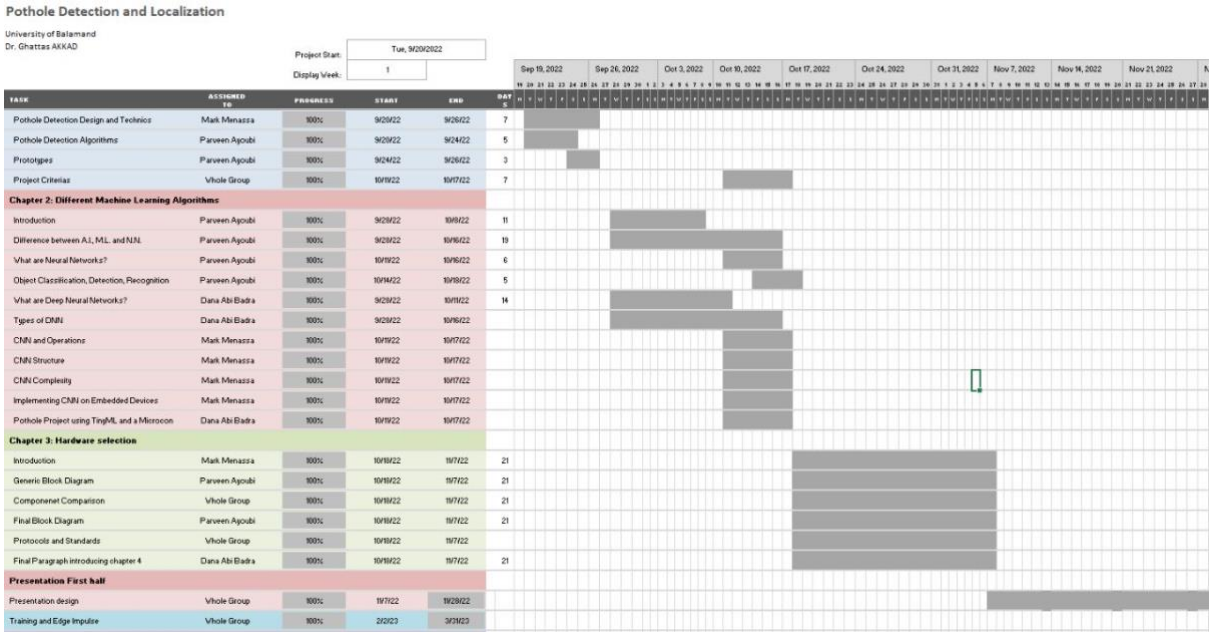
[48] “ISO 6709:2008,” ISO, 27-Sep-2022. [Online]. Available: https://www.iso.org/standard/39242.html. [Accessed: 21-Nov-2022].

[49] “IEEE SA - IEEE standard for floating-point arithmetic,” IEEE Standards Association. [Online]. Available: https://standards.ieee.org/ieee/754/6210/. [Accessed: 21-Nov-2022].

[50] D. Das, “How does a micro SD card module work and how to interface it with Arduino?,” Arduino Micro SD Card Module Tutorial - How SD Card Module Works and How to use it with Arduino, 25-May-2022. [Online]. Available: https://circuitdigest.com/microcontroller-projects/interfacing-micro-sd-card-module-with-arduino. [Accessed: 01-Mar-2023].

[51] ralphjy, “Arduino nano 33 ble sense with ov7670 camera,” element14 Community, 12-Sep-2021. [Online]. Available: https://community.element14.com/members-area/personalblogs/b/ralph-yamamoto-s-blog/posts/arduino-nano-33-ble-sense-with-ov7670-camera. [Accessed: 28-Feb-2023].

Appendix A

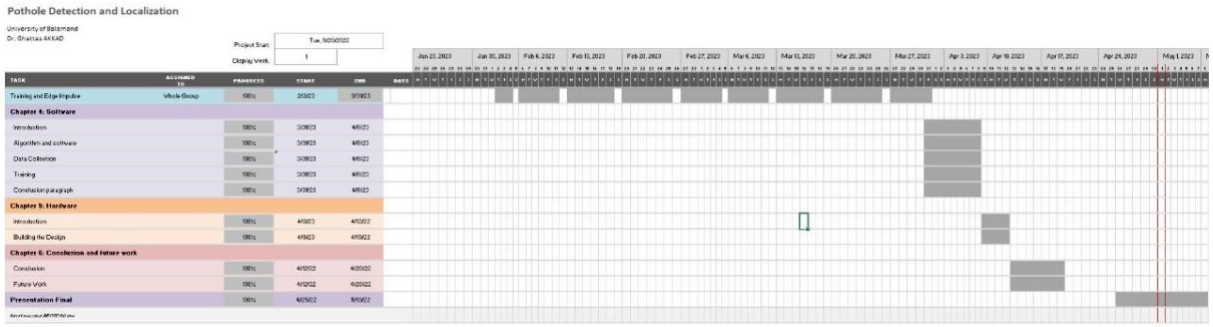


Pothole Detection and Localization

University of Balamand
Dr. Ghattas AKKAD

Project Start: Tue, 9/20/2022
Display Week: 1

TASK	ASSIGNED TO	PROGRESS	START	END	DAYS
Pothole Detection Design and Technics	Mark Menassa	100%	9/20/22	9/26/22	7
Pothole Detection Algorithms	Parveen Ayoubi	100%	9/20/22	9/24/22	5
Prototypes	Parveen Ayoubi	100%	9/24/22	9/26/22	3
Project Criterias	Whole Group	100%	10/11/22	10/17/22	7
Chapter 2: Different Machine Learning Algorithms					
Introduction	Parveen Ayoubi	100%	9/20/22	10/6/22	11
Difference between A.I, M.L. and N.N.	Parveen Ayoubi	100%	9/20/22	10/16/22	19
What are Neural Networks?	Parveen Ayoubi	100%	10/11/22	10/16/22	6
Object Classification, Detection, Recognition	Parveen Ayoubi	100%	10/14/22	10/18/22	5
What are Deep Neural Networks?	Dana Abi Badra	100%	9/20/22	10/11/22	14
Types of DNN	Dana Abi Badra	100%	9/20/22	10/16/22	
CNN and Operations	Mark Menassa	100%	10/11/22	10/17/22	
CNN Structure	Mark Menassa	100%	10/11/22	10/17/22	
CNN Complexity	Mark Menassa	100%	10/11/22	10/17/22	
Implementing CNN on Embedded Devices	Mark Menassa	100%	10/11/22	10/17/22	
Pothole Project using TinyML and a Microcon	Dana Abi Badra	100%	10/11/22	10/17/22	
Chapter 3: Hardware selection					
Introduction	Mark Menassa	100%	10/18/22	11/7/22	21
Generic Block Diagram	Parveen Ayoubi	100%	10/18/22	11/7/22	21
Componenet Comparison	Whole Group	100%	10/18/22	11/7/22	21
Final Block Diagram	Parveen Ayoubi	100%	10/18/22	11/7/22	21
Protocols and Standards	Whole Group	100%	10/18/22	11/7/22	
Final Paragraph introducing chapter 4	Dana Abi Badra	100%	10/18/22	11/7/22	21
Presentation First half					
Presentation design	Whole Group	100%	11/7/22	11/29/22	
Training and Edge Impulse	Whole Group	100%	2/2/23	3/31/23	



Pothole Detection and Localization

University of Balamand
Dr. Ghattas AKKAD

Project Start: Tue, 9/20/2022
Display Week: 1

TASK	ASSIGNED TO	PROGRESS	START	END	DAYS
Training and Edge Impulse	Whole Group	100%	2/2/23	3/31/23	
Chapter 4: Software					
Introduction		100%	3/28/23	4/6/23	
Algorithm and software		100%	3/28/23	4/6/23	
Data Collection		100%	3/28/23	4/6/23	
Training		100%	3/28/23	4/6/23	
Conclusion paragraph		100%	3/28/23	4/6/23	
Chapter 5: Hardware					
Introduction		100%	4/6/23	4/10/22	
Building the Design		100%	4/6/23	4/10/22	
Chapter 6: Conclusion and future work					
Conclusion		100%	4/10/22	4/20/22	
Future Work		100%	4/10/22	4/20/22	
Presentation Final		100%	4/25/22	5/1/22	

Appendix B

```
* Edge Impulse ingestion SDK
 * Copyright (c) 2022 EdgeImpulse Inc.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 */

/* Includes ---------------------------------------------------------------- */
#include <Bs_Project_inferencing.h>
#include <Arduino_OV767X.h>  //Click here to get the library: http://librarymanager/All#Arduino_OV767X

#include <stdint.h>
#include <stdlib.h>
#include <TinyGPS++.h>
#include <SD.h>
#include <SPI.h>
#include <JPEGDecoder.h>
#include <String.h>

long lat, lon;                  //Declaring variables to hold longitude and latitude from GPS coordiantes
TinyGPSPlus gps;                //Declaring a TinyGPSPlus variable
int bytesPerFrame = 52 * 52 * 2;  //Declaring variable for size of frame

/* Constant variables ------------------------------------------------------- */
#define EI_CAMERA_RAW_FRAME_BUFFER_COLS 160
#define EI_CAMERA_RAW_FRAME_BUFFER_ROWS 120

#define DWORD_ALIGN_PTR(a) ((a & 0x3) ? (((uintptr_t)a + 0x4) & ~(uintptr_t)0x3) : a)
```

```
/*
 ** NOTE: If you run into TFLite arena allocation issue.
 **
 ** This may be due to may dynamic memory fragmentation.
 ** Try defining "-DEI_CLASSIFIER_ALLOCATION_STATIC" in boards.local.txt (create
 ** if it doesn't exist) and copy this file to
 ** `<ARDUINO_CORE_INSTALL_PATH>/arduino/hardware/<mbed_core>/<core_version>/`.
 **
 ** See
 ** (https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-the-installed-cores-located-)
 ** to find where Arduino installs cores on your machine.
 **
 ** If the problem persists then there's not enough memory for this model and application.
 */

/* Edge Impulse ------------------------------------------------------------- */
class OV7675 : public OV767X {
public:
  int begin(int resolution, int format, int fps);
  void readFrame(void *buffer);

private:
  int vsyncPin;
  int hrefPin;
  int pclkPin;
  int xclkPin;

  volatile uint32_t *vsyncPort;
  uint32_t vsyncMask;
  volatile uint32_t *hrefPort;
  uint32_t hrefMask;
  volatile uint32_t *pclkPort;
  uint32_t pclkMask;

  uint16_t width;
  uint16_t height;
  uint8_t bytes_per_pixel;
  uint16_t bytes_per_row;
  uint8_t buf_rows;
  uint16_t buf_size;
```

```
  uint8_t resize_height;
  uint8_t *raw_buf;
  void *buf_mem;
  uint8_t *intrp_buf;
  uint8_t *buf_limit;

  void readBuf();
  int allocate_scratch_buffs();
  int deallocate_scratch_buffs();
};

typedef struct {
  size_t width;
  size_t height;
} ei_device_resize_resolutions_t;

/**
 * @brief      Check if new serial data is available
 *
 * @return     Returns number of available bytes
 */
int ei_get_serial_available(void) {
  return Serial.available();
}

/**
 * @brief      Get next available byte
 *
 * @return     byte
 */
char ei_get_serial_byte(void) {
  return Serial.read();
}

/* Private variables ------------------------------------------------------- */
static OV7675 Cam;
static bool is_initialised = false;

/*
** @brief points to the output of the capture
```

```
*/
static uint8_t *ei_camera_capture_out = NULL;
uint32_t resize_col_sz;
uint32_t resize_row_sz;
bool do_resize = false;
bool do_crop = false;

File myFile;
int fileFlag = 1;  //Used as counter for which pothole we are on

static bool debug_nn = false;  // Set this to true to see e.g. features generated from the raw signal

/* Function definitions ------------------------------------------------------- */
bool ei_camera_init(void);
void ei_camera_deinit(void);
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf);
int calculate_resize_dimensions(uint32_t out_width, uint32_t out_height, uint32_t *resize_col_sz, uint32_t
*resize_row_sz, bool *do_resize);
void resizeImage(int srcWidth, int srcHeight, uint8_t *srcImage, int dstWidth, int dstHeight, uint8_t *dstImage, int
iBpp);
void cropImage(int srcWidth, int srcHeight, uint8_t *srcImage, int startX, int startY, int dstWidth, int dstHeight,
uint8_t *dstImage, int iBpp);

/**
* @brief      Arduino setup function
*/
void setup() {

  Serial1.begin(9600);  //GPS on this port
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);  //Setting the built in led as an output to be used later

  // comment out the below line to cancel the wait for USB connection (needed for native USB)
  //while (!Serial)
   // ;
  Serial.println("Edge Impulse Inferencing Demo");

  // summary of inferencing settings (from model_metadata.h)
  ei_printf("Inferencing settings:\n");
  ei_printf("\tImage resolution: %dx%d\n", EI_CLASSIFIER_INPUT_WIDTH, EI_CLASSIFIER_INPUT_HEIGHT);
```

```
  ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
  ei_printf("\tNo. of classes: %d\n", sizeof(ei_classifier_inferencing_categories) /
sizeof(ei_classifier_inferencing_categories[0]));

  Serial.print("Initializing SD card...");
  pinMode(7, OUTPUT);
  digitalWrite(7, HIGH);
  if (!SD.begin(7)) {
    Serial.println("initialization failed!");
    while (1)
      ;
  }

  Serial.println("initialization done.");
}

/**
* @brief      Get data and run inferencing
*
* @param[in]  debug  Get debug info if true
*/
void loop() {
  bool stop_inferencing = false;

  while (stop_inferencing == false) {
    ei_printf("\nStarting inferencing in 2 seconds...\n");

    // instead of wait_ms, we'll wait on the signal, this allows threads to cancel us...
    if (ei_sleep(2000) != EI_IMPULSE_OK) {
      break;
    }

    ei_printf("Taking photo...\n");

    if (ei_camera_init() == false) {
      ei_printf("ERR: Failed to initialize image sensor\r\n");
      break;
    }

    // choose resize dimensions
```

```
    uint32_t resize_col_sz;
    uint32_t resize_row_sz;
    bool do_resize = false;
    int res = calculate_resize_dimensions(EI_CLASSIFIER_INPUT_WIDTH, EI_CLASSIFIER_INPUT_HEIGHT,
&resize_col_sz, &resize_row_sz, &do_resize);
    if (res) {
      ei_printf("ERR: Failed to calculate resize dimensions (%d)\r\n", res);
      break;
    }

    void *snapshot_mem = NULL;
    uint8_t *snapshot_buf = NULL;
    snapshot_mem = ei_malloc(resize_col_sz * resize_row_sz * 2);
    if (snapshot_mem == NULL) {
      ei_printf("failed to create snapshot_mem\r\n");
      break;
    }
    snapshot_buf = (uint8_t *)DWORD_ALIGN_PTR((uintptr_t)snapshot_mem);

    if (ei_camera_capture(EI_CLASSIFIER_INPUT_WIDTH, EI_CLASSIFIER_INPUT_HEIGHT, snapshot_buf) ==
false) {
      ei_printf("Failed to capture image\r\n");
      if (snapshot_mem) ei_free(snapshot_mem);
      break;
    }

    ei::signal_t signal;
    signal.total_length = EI_CLASSIFIER_INPUT_WIDTH * EI_CLASSIFIER_INPUT_HEIGHT;
    signal.get_data = &ei_camera_cutout_get_data;

    // run the impulse: DSP, neural network and the Anomaly algorithm
    ei_impulse_result_t result = { 0 };

    EI_IMPULSE_ERROR ei_error = run_classifier(&signal, &result, debug_nn);  //I think signal is the image itself,
but I dont know the type
    if (ei_error != EI_IMPULSE_OK) {
      ei_printf("Failed to run impulse (%d)\n", ei_error);
      ei_free(snapshot_mem);
      break;
    }
```

```
    // print the predictions
    ei_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.): \n",
            result.timing.dsp, result.timing.classification, result.timing.anomaly);
#if EI_CLASSIFIER_OBJECT_DETECTION == 1
    bool bb_found = result.bounding_boxes[0].value > 0;
    for (size_t ix = 0; ix < result.bounding_boxes_count; ix++) {
      auto bb = result.bounding_boxes[ix];
      if (bb.value == 0) {
        continue;
      }

      ei_printf("    %s (%f) [ x: %u, y: %u, width: %u, height: %u ]\n", bb.label, bb.value, bb.x, bb.y, bb.width,
bb.height);
    }

    if (!bb_found) {
      ei_printf("    No objects found\n");
    }
#else
    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
      ei_printf("    %s: %.5f\n", result.classification[ix].label,
              result.classification[ix].value);
    }

    //printGPS Coordinates
    gps.encode(Serial1.read());
    Serial.print("LAT= "); Serial.print(gps.location.lat(), 6);
    Serial.print("LNG= "); Serial.println(gps.location.lng(), 6);

    //Do a series of code if there has been a pothole detected
    if (result.classification[1].value > 0.75) {
      for (int LIGHT = 0; LIGHT < 5; LIGHT++) {       //Loop 5 times to blink light
        digitalWrite(LED_BUILTIN, HIGH);  //Turn LED on
        delay(1000);                      //Delay 1 second
        digitalWrite(LED_BUILTIN, LOW);   //Turn LED off
        delay(1000);                      //Delay 1 second
      }
      gps.encode(Serial1.read());  //read the data of the gps
```

```
    myFile = SD.open("Pothole.txt", FILE_WRITE);  //Open Pothole.txt as a write into file
    //Print on the file Pothole, its number and then a \n
    myFile.print("Pothole ");
    myFile.print(fileFlag, DEC);
    myFile.println("\n");

    //Translate raw data to hex values and print inside file (images)
    for (int i = 0; i < bytesPerFrame - 1; i += 2) {
     myFile.print("0x");
     myFile.print(snapshot_buf[i + 1], HEX);
     myFile.print(snapshot_buf[i], HEX);
     if (i != bytesPerFrame - 2) {
       myFile.print(",");
     }
    }

    //Print the Latitude and Longitude gps locations in file underneath picture
    myFile.print("\n\nLAT= ");
    myFile.println(gps.location.lat(), 6);
    myFile.print("LNG= ");
    myFile.println(gps.location.lng(), 6);
    myFile.println("\n");
    Serial.print("LAT=");
    Serial.print(gps.location.lat(), 6);
    Serial.print("LNG=");
    Serial.println(gps.location.lng(), 6);

    myFile.close();  //Close the file
    fileFlag++;      //increment counter
   }

#if EI_CLASSIFIER_HAS_ANOMALY == 1
    ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif
#endif

   while (ei_get_serial_available() > 0) {
    if (ei_get_serial_byte() == 'b') {
      ei_printf("Inferencing stopped by user\r\n");
```

```
        stop_inferencing = true;
      }
    }
    if (snapshot_mem) ei_free(snapshot_mem);
  }
  ei_camera_deinit();
}

/**
 * @brief      Determine whether to resize and to which dimension
 *
 * @param[in]  out_width     width of output image
 * @param[in]  out_height    height of output image
 * @param[out] resize_col_sz       pointer to frame buffer's column/width value
 * @param[out] resize_row_sz       pointer to frame buffer's rows/height value
 * @param[out] do_resize     returns whether to resize (or not)
 *
 */
int calculate_resize_dimensions(uint32_t out_width, uint32_t out_height, uint32_t *resize_col_sz, uint32_t
*resize_row_sz, bool *do_resize) {
  size_t list_size = 2;
  const ei_device_resize_resolutions_t list[list_size] = { { 42, 32 }, { 128, 96 } };

  // (default) conditions
  *resize_col_sz = EI_CAMERA_RAW_FRAME_BUFFER_COLS;
  *resize_row_sz = EI_CAMERA_RAW_FRAME_BUFFER_ROWS;
  *do_resize = false;

  for (size_t ix = 0; ix < list_size; ix++) {
    if ((out_width <= list[ix].width) && (out_height <= list[ix].height)) {
      *resize_col_sz = list[ix].width;
      *resize_row_sz = list[ix].height;
      *do_resize = true;
      break;
    }
  }

  return 0;
}
```

```
/**
 * @brief   Setup image sensor & start streaming
 *
 * @retval  false if initialisation failed
 */
bool ei_camera_init(void) {
  if (is_initialised) return true;

  if (!Cam.begin(QQVGA, RGB565, 1)) {  // VGA downsampled to QQVGA (OV7675)
    ei_printf("ERR: Failed to initialize camera\r\n");
    return false;
  }
  is_initialised = true;

  return true;
}

/**
 * @brief      Stop streaming of sensor data
 */
void ei_camera_deinit(void) {
  if (is_initialised) {
    Cam.end();
    is_initialised = false;
  }
}

/**
 * @brief      Capture, rescale and crop image
 *
 * @param[in]  img_width     width of output image
 * @param[in]  img_height    height of output image
 * @param[in]  out_buf       pointer to store output image, NULL may be used
 *                           when full resolution is expected.
 *
 * @retval     false if not initialised, image captured, rescaled or cropped failed
 *
 */
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf) {
  if (!is_initialised) {
```

```
  ei_printf("ERR: Camera is not initialized\r\n");
  return false;
}

if (!out_buf) {
  ei_printf("ERR: invalid parameters\r\n");
  return false;
}

// choose resize dimensions
int res = calculate_resize_dimensions(img_width, img_height, &resize_col_sz, &resize_row_sz, &do_resize);
if (res) {
  ei_printf("ERR: Failed to calculate resize dimensions (%d)\r\n", res);
  return false;
}

if ((img_width != resize_col_sz)
    || (img_height != resize_row_sz)) {
  do_crop = true;
}

Cam.readFrame(out_buf);  // captures image and resizes

if (do_crop) {
  uint32_t crop_col_sz;
  uint32_t crop_row_sz;
  uint32_t crop_col_start;
  uint32_t crop_row_start;
  crop_row_start = (resize_row_sz - img_height) / 2;
  crop_col_start = (resize_col_sz - img_width) / 2;
  crop_col_sz = img_width;
  crop_row_sz = img_height;

  //ei_printf("crop cols: %d, rows: %d\r\n", crop_col_sz,crop_row_sz);
  cropImage(resize_col_sz, resize_row_sz,
            out_buf,
            crop_col_start, crop_row_start,
            crop_col_sz, crop_row_sz,
            out_buf,
            16);
```

```
  }

  // The following variables should always be assigned
  // if this routine is to return true
  // cutout values
  //ei_camera_snapshot_is_resized = do_resize;
  //ei_camera_snapshot_is_cropped = do_crop;
  ei_camera_capture_out = out_buf;

  return true;
}

/**
 * @brief      Convert RGB565 raw camera buffer to RGB888
 *
 * @param[in]   offset       pixel offset of raw buffer
 * @param[in]   length       number of pixels to convert
 * @param[out]  out_buf      pointer to store output image
 */
int ei_camera_cutout_get_data(size_t offset, size_t length, float *out_ptr) {
  size_t pixel_ix = offset * 2;
  size_t bytes_left = length;
  size_t out_ptr_ix = 0;

  // read byte for byte
  while (bytes_left != 0) {
    // grab the value and convert to r/g/b
    uint16_t pixel = (ei_camera_capture_out[pixel_ix] << 8) | ei_camera_capture_out[pixel_ix + 1];
    uint8_t r, g, b;
    r = ((pixel >> 11) & 0x1f) << 3;
    g = ((pixel >> 5) & 0x3f) << 2;
    b = (pixel & 0x1f) << 3;

    // then convert to out_ptr format
    float pixel_f = (r << 16) + (g << 8) + b;
    out_ptr[out_ptr_ix] = pixel_f;

    // and go to the next pixel
    out_ptr_ix++;
    pixel_ix += 2;
```

```
    bytes_left--;
  }

  // and done!
  return 0;
}

// This include file works in the Arduino environment
// to define the Cortex-M intrinsics
#ifdef __ARM_FEATURE_SIMD32
#include <device.h>
#endif
// This needs to be < 16 or it won't fit. Cortex-M4 only has SIMD for signed multiplies
#define FRAC_BITS 14
#define FRAC_VAL (1 << FRAC_BITS)
#define FRAC_MASK (FRAC_VAL - 1)
//
// Resize
//
// Assumes that the destination buffer is dword-aligned
// Can be used to resize the image smaller or larger
// If resizing much smaller than 1/3 size, then a more rubust algorithm should average all of the pixels
// This algorithm uses bilinear interpolation - averages a 2x2 region to generate each new pixel
//
// Optimized for 32-bit MCUs
// supports 8 and 16-bit pixels
void resizeImage(int srcWidth, int srcHeight, uint8_t *srcImage, int dstWidth, int dstHeight, uint8_t *dstImage, int
iBpp) {
  uint32_t src_x_accum, src_y_accum;  // accumulators and fractions for scaling the image
  uint32_t x_frac, nx_frac, y_frac, ny_frac;
  int x, y, ty, tx;

  if (iBpp != 8 && iBpp != 16)
    return;
  src_y_accum = FRAC_VAL / 2;  // start at 1/2 pixel in to account for integer downsampling which might miss
pixels
  const uint32_t src_x_frac = (srcWidth * FRAC_VAL) / dstWidth;
  const uint32_t src_y_frac = (srcHeight * FRAC_VAL) / dstHeight;
  const uint32_t r_mask = 0xf800f800;
  const uint32_t g_mask = 0x07e007e0;
```

```
  const uint32_t b_mask = 0x001f001f;
  uint8_t *s, *d;
  uint16_t *s16, *d16;
  uint32_t x_frac2, y_frac2;  // for 16-bit SIMD
  for (y = 0; y < dstHeight; y++) {
    ty = src_y_accum >> FRAC_BITS;  // src y
    y_frac = src_y_accum & FRAC_MASK;
    src_y_accum += src_y_frac;
    ny_frac = FRAC_VAL - y_frac;        // y fraction and 1.0 - y fraction
    y_frac2 = ny_frac | (y_frac << 16);  // for M4/M4 SIMD
    s = &srcImage[ty * srcWidth];
    s16 = (uint16_t *)&srcImage[ty * srcWidth * 2];
    d = &dstImage[y * dstWidth];
    d16 = (uint16_t *)&dstImage[y * dstWidth * 2];
    src_x_accum = FRAC_VAL / 2;  // start at 1/2 pixel in to account for integer downsampling which might miss
pixels
    if (iBpp == 8) {
      for (x = 0; x < dstWidth; x++) {
        uint32_t tx, p00, p01, p10, p11;
        tx = src_x_accum >> FRAC_BITS;
        x_frac = src_x_accum & FRAC_MASK;
        nx_frac = FRAC_VAL - x_frac;  // x fraction and 1.0 - x fraction
        x_frac2 = nx_frac | (x_frac << 16);
        src_x_accum += src_x_frac;
        p00 = s[tx];
        p10 = s[tx + 1];
        p01 = s[tx + srcWidth];
        p11 = s[tx + srcWidth + 1];
#ifdef __ARM_FEATURE_SIMD32
        p00 = __SMLAD(p00 | (p10 << 16), x_frac2, FRAC_VAL / 2) >> FRAC_BITS;  // top line
        p01 = __SMLAD(p01 | (p11 << 16), x_frac2, FRAC_VAL / 2) >> FRAC_BITS;  // bottom line
        p00 = __SMLAD(p00 | (p01 << 16), y_frac2, FRAC_VAL / 2) >> FRAC_BITS;  // combine
#else                                                  // generic C code
        p00 = ((p00 * nx_frac) + (p10 * x_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // top line
        p01 = ((p01 * nx_frac) + (p11 * x_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // bottom line
        p00 = ((p00 * ny_frac) + (p01 * y_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // combine top + bottom
#endif                                                 // Cortex-M4/M7
        *d++ = (uint8_t)p00;                                  // store new pixel
      }                                                // for x
    }                                                  // 8-bpp
```

```
    else {                                              // RGB565
      for (x = 0; x < dstWidth; x++) {
        uint32_t tx, p00, p01, p10, p11;
        uint32_t r00, r01, r10, r11, g00, g01, g10, g11, b00, b01, b10, b11;
        tx = src_x_accum >> FRAC_BITS;
        x_frac = src_x_accum & FRAC_MASK;
        nx_frac = FRAC_VAL - x_frac;  // x fraction and 1.0 - x fraction
        x_frac2 = nx_frac | (x_frac << 16);
        src_x_accum += src_x_frac;
        p00 = __builtin_bswap16(s16[tx]);
        p10 = __builtin_bswap16(s16[tx + 1]);
        p01 = __builtin_bswap16(s16[tx + srcWidth]);
        p11 = __builtin_bswap16(s16[tx + srcWidth + 1]);
#ifdef __ARM_FEATURE_SIMD32
        {
          p00 |= (p10 << 16);
          p01 |= (p11 << 16);
          r00 = (p00 & r_mask) >> 1;
          g00 = p00 & g_mask;
          b00 = p00 & b_mask;
          r01 = (p01 & r_mask) >> 1;
          g01 = p01 & g_mask;
          b01 = p01 & b_mask;
          r00 = __SMLAD(r00, x_frac2, FRAC_VAL / 2) >> FRAC_BITS;              // top line
          r01 = __SMLAD(r01, x_frac2, FRAC_VAL / 2) >> FRAC_BITS;              // bottom line
          r00 = __SMLAD(r00 | (r01 << 16), y_frac2, FRAC_VAL / 2) >> FRAC_BITS;  // combine
          g00 = __SMLAD(g00, x_frac2, FRAC_VAL / 2) >> FRAC_BITS;              // top line
          g01 = __SMLAD(g01, x_frac2, FRAC_VAL / 2) >> FRAC_BITS;              // bottom line
          g00 = __SMLAD(g00 | (g01 << 16), y_frac2, FRAC_VAL / 2) >> FRAC_BITS;  // combine
          b00 = __SMLAD(b00, x_frac2, FRAC_VAL / 2) >> FRAC_BITS;              // top line
          b01 = __SMLAD(b01, x_frac2, FRAC_VAL / 2) >> FRAC_BITS;              // bottom line
          b00 = __SMLAD(b00 | (b01 << 16), y_frac2, FRAC_VAL / 2) >> FRAC_BITS;  // combine
        }
#else   // generic C code
        {
          r00 = (p00 & r_mask) >> 1;
          g00 = p00 & g_mask;
          b00 = p00 & b_mask;
          r10 = (p10 & r_mask) >> 1;
          g10 = p10 & g_mask;
```

```
            b10 = p10 & b_mask;
            r01 = (p01 & r_mask) >> 1;
            g01 = p01 & g_mask;
            b01 = p01 & b_mask;
            r11 = (p11 & r_mask) >> 1;
            g11 = p11 & g_mask;
            b11 = p11 & b_mask;
            r00 = ((r00 * nx_frac) + (r10 * x_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // top line
            r01 = ((r01 * nx_frac) + (r11 * x_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // bottom line
            r00 = ((r00 * ny_frac) + (r01 * y_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // combine top + bottom
            g00 = ((g00 * nx_frac) + (g10 * x_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // top line
            g01 = ((g01 * nx_frac) + (g11 * x_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // bottom line
            g00 = ((g00 * ny_frac) + (g01 * y_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // combine top + bottom
            b00 = ((b00 * nx_frac) + (b10 * x_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // top line
            b01 = ((b01 * nx_frac) + (b11 * x_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // bottom line
            b00 = ((b00 * ny_frac) + (b01 * y_frac) + FRAC_VAL / 2) >> FRAC_BITS;  // combine top + bottom
          }
#endif  // Cortex-M4/M7
          r00 = (r00 << 1) & r_mask;
          g00 = g00 & g_mask;
          b00 = b00 & b_mask;
          p00 = (r00 | g00 | b00);                  // re-combine color components
          *d16++ = (uint16_t)__builtin_bswap16(p00);  // store new pixel
        }                                           // for x
      }                                             // 16-bpp
    }                                               // for y
} /* resizeImage() */
//
// Crop
//
// Assumes that the destination buffer is dword-aligned
// optimized for 32-bit MCUs
// Supports 8 and 16-bit pixels
//
void cropImage(int srcWidth, int srcHeight, uint8_t *srcImage, int startX, int startY, int dstWidth, int dstHeight,
uint8_t *dstImage, int iBpp) {
  uint32_t *s32, *d32;
  int x, y;
```

```
 if (startX < 0 || startX >= srcWidth || startY < 0 || startY >= srcHeight || (startX + dstWidth) > srcWidth || (startY +
dstHeight) > srcHeight)
   return;  // invalid parameters
 if (iBpp != 8 && iBpp != 16)
   return;

 if (iBpp == 8) {
   uint8_t *s, *d;
   for (y = 0; y < dstHeight; y++) {
     s = &srcImage[srcWidth * (y + startY) + startX];
     d = &dstImage[(dstWidth * y)];
     x = 0;
     if ((intptr_t)s & 3 || (intptr_t)d & 3) {  // either src or dst pointer is not aligned
       for (; x < dstWidth; x++) {
         *d++ = *s++;  // have to do it byte-by-byte
       }
     } else {
       // move 4 bytes at a time if aligned or alignment not enforced
       s32 = (uint32_t *)s;
       d32 = (uint32_t *)d;
       for (; x < dstWidth - 3; x += 4) {
         *d32++ = *s32++;
       }
       // any remaining stragglers?
       s = (uint8_t *)s32;
       d = (uint8_t *)d32;
       for (; x < dstWidth; x++) {
         *d++ = *s++;
       }
     }
   }  // for y
 }   // 8-bpp
 else {
   uint16_t *s, *d;
   for (y = 0; y < dstHeight; y++) {
     s = (uint16_t *)&srcImage[2 * srcWidth * (y + startY) + startX * 2];
     d = (uint16_t *)&dstImage[(dstWidth * y * 2)];
     x = 0;
     if ((intptr_t)s & 2 || (intptr_t)d & 2) {  // either src or dst pointer is not aligned
       for (; x < dstWidth; x++) {
```

```
        *d++ = *s++;  // have to do it 16-bits at a time
       }
      } else {
       // move 4 bytes at a time if aligned or alignment no enforced
       s32 = (uint32_t *)s;
       d32 = (uint32_t *)d;
       for (; x < dstWidth - 1; x += 2) {  // we can move 2 pixels at a time
        *d32++ = *s32++;
       }
       // any remaining stragglers?
       s = (uint16_t *)s32;
       d = (uint16_t *)d32;
       for (; x < dstWidth; x++) {
        *d++ = *s++;
       }
      }
    }  // for y
  }    // 16-bpp case
} /* cropImage() */

#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_CAMERA
#error "Invalid model for current sensor"
#endif

// OV767X camera library override
#include <Arduino.h>
#include <Wire.h>

#define digitalPinToBitMask(P) (1 << (digitalPinToPinName(P) % 32))
#define portInputRegister(P) ((P == 0) ? &NRF_P0->IN : &NRF_P1->IN)

//
// OV7675::begin()
//
// Extends the OV767X library function. Some private variables are needed
// to use the OV7675::readFrame function.
//
int OV7675::begin(int resolution, int format, int fps) {
  pinMode(OV7670_VSYNC, INPUT);
  pinMode(OV7670_HREF, INPUT);
```

```
  pinMode(OV7670_PLK, INPUT);
  pinMode(OV7670_XCLK, OUTPUT);

  vsyncPort = portInputRegister(digitalPinToPort(OV7670_VSYNC));
  vsyncMask = digitalPinToBitMask(OV7670_VSYNC);
  hrefPort = portInputRegister(digitalPinToPort(OV7670_HREF));
  hrefMask = digitalPinToBitMask(OV7670_HREF);
  pclkPort = portInputRegister(digitalPinToPort(OV7670_PLK));
  pclkMask = digitalPinToBitMask(OV7670_PLK);

  // init driver to use full image sensor size
  bool ret = OV767X::begin(VGA, format, fps);
  width = OV767X::width();    // full sensor width
  height = OV767X::height();  // full sensor height
  bytes_per_pixel = OV767X::bytesPerPixel();
  bytes_per_row = width * bytes_per_pixel;  // each pixel is 2 bytes
  resize_height = 2;

  buf_mem = NULL;
  raw_buf = NULL;
  intrp_buf = NULL;
  //allocate_scratch_buffs();

  return ret;
} /* OV7675::begin() */

int OV7675::allocate_scratch_buffs() {
  //ei_printf("allocating buffers..\r\n");
  buf_rows = height / resize_row_sz * resize_height;
  buf_size = bytes_per_row * buf_rows;

  buf_mem = ei_malloc(buf_size);
  if (buf_mem == NULL) {
    ei_printf("failed to create buf_mem\r\n");
    return false;
  }
  raw_buf = (uint8_t *)DWORD_ALIGN_PTR((uintptr_t)buf_mem);

  //ei_printf("allocating buffers OK\r\n");
  return 0;
```

```
}
int OV7675::deallocate_scratch_buffs() {
  //ei_printf("deallocating buffers...\r\n");
  ei_free(buf_mem);
  buf_mem = NULL;

  //ei_printf("deallocating buffers OK\r\n");
  return 0;
}

//
// OV7675::readFrame()
//
// Overrides the OV767X library function. Fixes the camera output to be
// a far more desirable image. This image utilizes the full sensor size
// and has the correct aspect ratio. Since there is limited memory on the
// Nano we bring in only part of the entire sensor at a time and then
// interpolate to a lower resolution.
//
void OV7675::readFrame(void *buffer) {
  allocate_scratch_buffs();

  uint8_t *out = (uint8_t *)buffer;
  noInterrupts();

  // Falling edge indicates start of frame
  while ((*vsyncPort & vsyncMask) == 0)
    ;  // wait for HIGH
  while ((*vsyncPort & vsyncMask) != 0)
    ;  // wait for LOW

  int out_row = 0;
  for (int raw_height = 0; raw_height < height; raw_height += buf_rows) {
    // read in 640xbuf_rows buffer to work with
    readBuf();

    resizeImage(width, buf_rows,
                raw_buf,
                resize_col_sz, resize_height,
                &(out[out_row]),
```

```
            16);

    out_row += resize_col_sz * resize_height * bytes_per_pixel; /* resize_col_sz * 2 * 2 */
  }

  interrupts();

  deallocate_scratch_buffs();
} /* OV7675::readFrame() */

//
// OV7675::readBuf()
//
// Extends the OV767X library function. Reads buf_rows VGA rows from the
// image sensor.
//
void OV7675::readBuf() {
  int offset = 0;

  uint32_t ulPin = 33;  // P1.xx set of GPIO is in 'pin' 32 and above
  NRF_GPIO_Type *port;

  port = nrf_gpio_pin_port_decode(&ulPin);

  for (int i = 0; i < buf_rows; i++) {
    // rising edge indicates start of line
    while ((*hrefPort & hrefMask) == 0)
      ;  // wait for HIGH

    for (int col = 0; col < bytes_per_row; col++) {
      // rising edges clock each data byte
      while ((*pclkPort & pclkMask) != 0)
        ;  // wait for LOW

      uint32_t in = port->IN;  // read all bits in parallel

      in >>= 2;         // place bits 0 and 1 at the "bottom" of the register
      in &= 0x3f03;     // isolate the 8 bits we care about
      in |= (in >> 6);  // combine the upper 6 and lower 2 bits
```

```
      raw_buf[offset++] = in;

      while ((*pclkPort & pclkMask) == 0)
        ;  // wait for HIGH
    }

    while ((*hrefPort & hrefMask) != 0)
      ;  // wait for LOW
  }
} /* OV7675::readBuf() */
```

Appendix C

```
# Import the needed libraries
from matplotlib import pyplot as plt
import numpy as np
import struct
import string

# Initialize a counter variable
a = 1
lon_index = 0

#Count the number of potholes
count = 0
with open("POTHOLE.TXT", "r") as file:
    for line in file:
        count += line.count("Pothole")

# Loop through the file
for j in range(count):

  # Open the text file for reading
  with open("POTHOLE.TXT", "r") as file:
    contents = file.read()

  nextPothole = "Pothole " + str(a)

  # Find the index of the start of the next pothole section in the text file
  index = contents.find("Pothole", lon_index)

  # Find the index of the next newline character after the start of the section
  newline_index = contents.find("\n", index)

  # Find the index of the next newline character after the previous one
  next_newline_index = contents.find("\n", newline_index + 1)

  # Find the index of the next newline character after the previous one
  next_index_enter = contents.find("\n", next_newline_index + 1)

  # Find the index of the "LAT=" substring in the current section
```

```
lat_index = contents.find("LAT= ", next_index_enter)

# Find the index of the "LNG=" substring in the current section
lon_index = contents.find("LNG= ", lat_index)

# Extract the latitude and longitude values from the current section
LAT = contents[lat_index: lat_index+15]
LNG = contents[lon_index: lon_index+15]

# Extract the hexadecimal values for the image data as a string
hex_string = contents[newline_index + 2: next_index_enter - 1]

# Split the hexadecimal string into a list of values
hex_list = hex_string.split(",")

# Convert the hexadecimal values from strings to integers
hex_int = [0]*len(hex_list)
new_int = [0]*len(hex_list)
for b in range(len(hex_list)):
  hex_int[b] = int(hex_list[b], 16)
  new_int[b] = hex_int[b]

# Convert the integer values to raw bytes
raw_bytes = np.array(new_int, dtype="i2")

# Create an array to store the RGB values for each pixel
image = np.zeros((len(raw_bytes),3), dtype=int)

# Loop through all of the pixels and form the image
for i in range(len(raw_bytes)):
    # Read a 16-bit pixel from the raw bytes
    pixel = struct.unpack('>h', raw_bytes[i])[0]

    # Convert the RGB565 format to RGB 24-bit
    r = ((pixel >> 11) & 0x1f) << 3;
    g = ((pixel >> 5) & 0x3f) << 2;
    b = ((pixel >> 0) & 0x1f) << 3;
    image[i] = [r,g,b]

# Reshape the image array to match the QCIF resolution (52x52 pixels)
```

```
image = np.reshape(image,(52, 52,3))

# Show the image and add the latitude and longitude values as text in the top right corner
plt.imshow(image)
# Add LAT and LNG values to image
plt.text(0.9, 0.1, f'{LAT} {LNG}', horizontalalignment='right',
      verticalalignment='bottom', transform=plt.gca().transAxes, fontsize=12, color='white', weight='bold')

# Construct the filename for the plot based on the value of a
nameOfFile = 'Pothole' + str(a) + '.png'

# Set the plot title to "Pothole" followed by the value of a
plt.title('Pothole ' + str(a))

# Save the plot to the file with the constructed filename
plt.savefig(nameOfFile)

# Display the plot on the screen
plt.show()
a = a+1
```