

Florida Institute of Technology

Scholarship Repository @ Florida Tech

Theses and Dissertations

5-2020

Model Optimization For Edge Devices

Adolf Anthony D'costa

Follow this and additional works at: <https://repository.fit.edu/etd>



Part of the [Computer Engineering Commons](#)

Model Optimization For Edge Devices

by

Adolf Anthony D'costa

Bachelor of Engineering

Electronics And Telecommunication Engineering

Mumbai University

2014

A thesis

submitted to the College of Engineering and Science at

Florida Institute of Technology

in partial fulfillment of the requirements

for the degree of

Master of Science

in

Computer Engineering

Melbourne, Florida

May, 2020

We, the undersigned committee, hereby approve the attached thesis.

Model Optimization For Edge Devices

by

Adolf Anthony D'costa

Carlos Otero, Ph.D.
Major Advisor
Associate Professor, Computer Engineering

Veton Kepuska, Ph.D.
Committee Member
Associate Professor, Electrical and Computer Engineering

William Allen, Ph.D.
Outside Committee Member
Associate Professor, Computer and Science

Philip Bernhard, Ph.D.
Department Head
Associate Professor, Computer Engineering and Sciences

Abstract

Title:

Model Optimization For Edge Devices

Author:

Adolf Anthony D'costa

Thesis Advisor:

Carlos E. Otero, Ph.D.

Edge devices are undergoing groundbreaking computing transformation, which lets us tap into artificial intelligence, quantum computing, 5th generation network capability, fog networking, and computing complex algorithms. Edge systems have substantial advantages over the conventional system in terms of scalability, optimized resources, reliability, and security. The proliferation of such resource-constrained devices in recent years has resulted in the generation of a large quantity of data; these data-producing devices are attractive targets for applications of machine learning. Machine learning models, especially deep learning neural networks, produced models that have high accuracy and prediction capability, but it comes at the cost of computation power and memory consumption. There is a large scope

to optimize machine learning models over certain criteria such as size, efficiency, latency, and accuracy. Optimization of machine learning models and running them over a constrained environment will help revolutionize edge devices and help develop exceptional results in real-time.

Table of Contents

List of Figures	viii
List of Tables	xi
Acknowledgements	xii
1 Introduction	1
2 Literature Review	4
2.1 Traditional Machine Learning	5
2.2 Speech Recognition	5
2.3 Converging Speech Recognition And Machine Learning	6
2.4 Sound Classification using Machine Learning on Edge Device	7
2.5 Optimizing Machine Learning Model For Edge Devices	9
2.5.1 Architecture Redesign	10

2.5.2	Pruning	10
2.5.3	Quantization Awareness Training	11
2.5.4	Quantization	13
3	Proposed Approach	15
4	Implementation	19
4.1	Conventional Edge Devices	19
4.1.1	Microprocessor	20
4.1.2	Microcontroller	20
4.2	Hardware Selection	22
4.3	Pre-requisites	23
4.4	Segregating and Data Collection	26
4.5	Signal processing parameters	27
4.6	Pre-processing	27
4.7	Data Organization and Conversion from 1D to 2D	34
4.8	Model	37
4.9	Quantization Aware Training	41
4.10	Magnitude Based Weight Pruning	43

4.11	Quantization	46
4.12	Conversion to C array	47
5	Results	48
5.1	Logging Data	48
5.2	Parameters and Duration for Testing	49
5.3	Output	50
5.4	Confusion Matrix	52
5.4.1	Results in Confusion Matrix	53
5.5	Plots Weights and Biases	55
5.5.1	GPU Usage	55
5.5.2	Epochs	56
5.5.3	Loss, Accuracy and Learning Rate	57
5.5.4	Model Comparison	59
6	Conclusion	61
6.0.1	Future Work	61
	Bibliography	62

List of Figures

2.1	Edge Eco-System [11]	5
2.2	Casper system architecture [44]	7
2.3	Fire Module [28]	10
2.4	Pruning [43]	11
2.5	Quantization Aware Training [16]	12
2.6	Quantization AlexNet [41]	13
3.1	Flow Chart For Proposed Approach	16
3.2	Typical Machine Learning System For Audio Signals	17
4.1	Casper system architecture [44]	19
4.2	Basic Microprocessor Architecture [5]	20
4.3	Basic Microcontroller Architecture [5]	22
4.4	Anaconda	23

4.5	Raw Wave Form ESC-50 Knocking (1-81001-A-30.wav) [39]	29
4.6	Standard Scoring Normalization	30
4.7	Processed Data 1st Window	31
4.8	Processed Data 2nd Window	32
4.9	Processed Data 3rd Window With Padding	33
4.10	Converting 1D Array to 2D [21]	35
4.11	Model Flowchart [28]	39
4.12	System Generated Model Architecture [28]	40
4.13	Quantization using TensorFlow lite Operation [7]	41
4.14	Quantization Aware Training [41]	42
4.15	Quantization Aware Training Weight Adjustments [41]	43
4.16	Pruning Technique [26]	43
4.17	Pruning Model Architecture	45
4.18	Quantization AlexNet [41]	46
5.1	ShrinkLog Text File	51
5.2	Original Model Accuracy	52
5.3	TensorFlow Lite Model Accuracy	52
5.4	Confusion Matrix Example Keyboard Typing	54

5.5	GPU Usage	55
5.6	Epochs	56
5.7	Validation Accuracy	57
5.8	Validation Loss	58
5.9	Learning Rate	59
5.10	Model Comparison	60

List of Tables

2.1	Microcontroller Compatible With Machine Learning	9
4.1	List of Software	24
4.2	List of Dependent Python Packages	25
4.3	List of Sounds	26
5.1	Confusion Matrix Calculation Table	54

Acknowledgements

I wish to express my deepest gratitude to my advisor and mentor Dr. Carlos E. Otero; he has been an inspiration to me. His wisdom, expertise, and support were a gift to me for the past two years.

I am deeply grateful to my parents Anthony Dcosta and Rayna Dcosta, along with my relatives for the motivation, encouragement, and support they have given me through this journey.

Finally, I would like to thank my colleague David Elliott for spending time with me to resolve issues related to QAT, data handling, and also engaged me with innovative ideas. I would like to appreciate the help from Nikita John and Tapas Joshi, they have helped me correct my thesis document, by providing valuable feedback, also like to thank Ravi Pandhi and Rosalin Dash for their support.

Chapter 1

Introduction

With the introduction of "Edge Devices," which can be traced back to the 1990s, the number of devices has been exponentially increasing. There are approximately 20.4 billion edge devices today [3]. These devices are embedded system which has a variety of sensors connected to them depending upon the application they were designed. Edge devices are small, power-efficient, and application-oriented, which generates a large amount of data. These massive amounts of data are currently transmitted over to data centers for processing, as these embedded systems have a constraint on processing, power, and storage. This leads to other challenges; it isn't cost-effective to bring the data to the cloud for real-time inference. Network latency is present in bringing the data from edge devices to data centers for processing, which affects the amount of time required to provide a result from the time the data was generated, which could be crucial in specific applications. Sending data from the edge to the cloud raises a scalability problem as the number of connected devices on the network increases. There is a substantial potential of a security breach to occur when data is transmitted over the internet; this could be critical if the data is classified information.

Machine learning is a method that lets a computer learn explicitly without programming. One of the most significant advantages of machine learning is its capacity to improvise over time. Machine learning can efficiently process intricate patterns and also identify minute changes. Once a machine learning model is trained over a wide array of data, and it has archived an excellent accuracy, it can be used to review, classify, and process significant volumes of data. No human intervention is required. It can handle multi-dimensional and multi-variety of data, which broadens its scope of application. Classification, Clustering, Natural Language Processing, and Forecasting are some of the models that are compatible with machine learning. Machine learning is resource hungry; it needs a massive amount of computational resources to function. Engineers generate a highly efficient, accurate, and effective model by using a large amount of data, large computation power like Graphic Processor Unit (GPU) or Tensor Processing Unit (TPU), and large high-speed memory Solid State Drive (SSD).

We proposed an approach to optimize a machine learning model in an efficient manner to harness limited computational resources from the edge devices. Machine learning is effective in handling advance, complex, and dynamic processes. The aim is to optimize the model so that it can run under-restricted computational resources efficiently and effectively; this will revolutionize edge computing. The ultimate objective is to develop a model over machine learning that is extremely small in size and yet has high accuracy in prediction, which will intern decentralize data processing. Electronics are undergoing constant evolution, peripherals and sensors are getting smaller; a large array of a variety of sensors are now being incorporated into embedded systems. This data can be processed and evaluated in real-time using the ability of machine learning.

Chapter 2 contains the literature review of all the work done in the past on edge devices, speech processing, and machine learning optimization. Chapter 3 includes details of a proposed approach to implement the system. Chapter 4 comprises of implementation and details about how the system was built. Chapter 5 contains details of the results achieved by implementing the system. Chapter 6 will conclude the thesis and talk about future scope and improvements.

Chapter 2

Literature Review

Edge devices have gone from being a theoretical concept to a key component. Edge devices are now growing exponentially, expecting a staggering growth from 15 billion today to 150 billion by 2025, [17]. Edge devices that have internet connectivity are IoT devices. By the year 2025 IDC (International Data Corporation), is expecting approximately 75.44 billion IoT devices [15], which will generate 79.4 zettabytes of data [18]. This thesis aims at processing and computing data at the edge. Figure 2.1 shows an ecosystem for Edge devices.

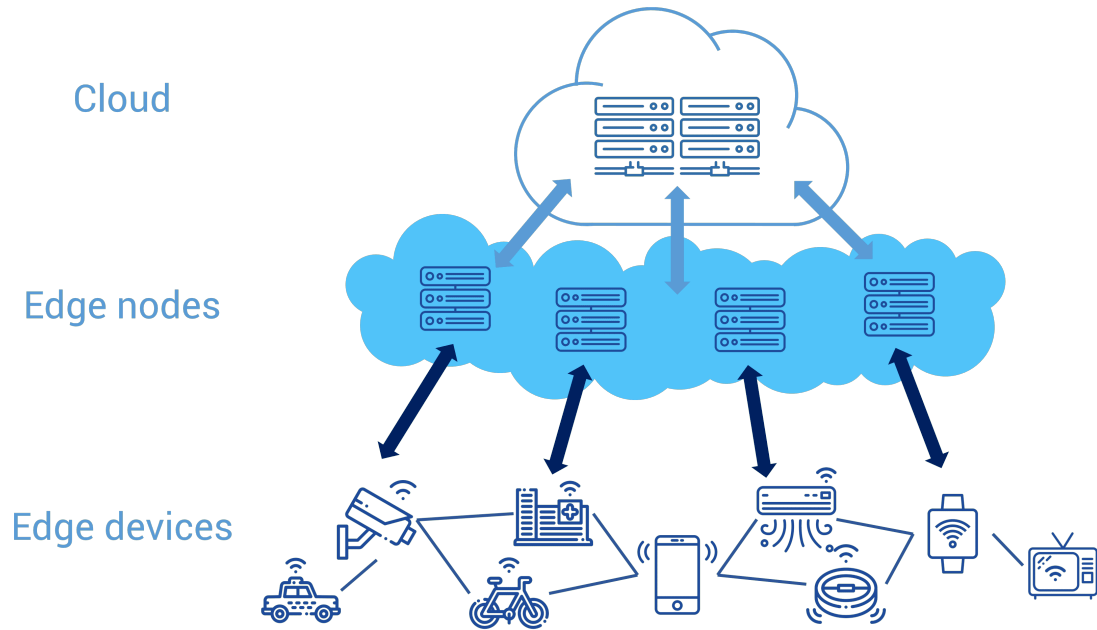


Figure 2.1: Edge Eco-System [11]

2.1 Traditional Machine Learning

Machine learning was termed by Arthur Samuel in the year 1959 [42]. In conjunction, Tom M Mitchell defined it as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E ." [35]. In the year 1963, Donald Michie uses reinforcement learning to create the first applicative algorithm of machine learning, which can play tic-tac-toe [19].

2.2 Speech Recognition

In conjunction with Machine learning, there is intensive research and efforts put into the development of speech recognition. "Audrey" was the first documented

speech recognizer; it was invented in 1952 by Davis, Biddulph, and Balashek at Bell Laboratories. The system was analog and could only detect digits, which ran on a huge relay rack [12]. Increase in the insistence in the field of speech recognition, lead to several innovations. "Dragon Dictate" world's first speech recognition software was developed in 1990 for consumers; software wasn't efficient enough as the user had to pause between each word [34].

2.3 Converging Speech Recognition And Machine Learning

Apple introduced "Casper" to its benefit during the era of innovation in speech recognition. It was the first continuous speech recognition software that could detect approximately 20,000 words. It was also the first voice recognizer that used a decision tree to predict a continuous speech and perform synthesis [44]. The architecture of Casper is shown in Figure 2.2. It was running on a Macintosh machine which had ample computing power; it was equipped with 68030 processor clocked at 25MHz, 4MB of RAM and 80MB of hard disk capacity [33].

The Casper system architecture can be pictured as follows:

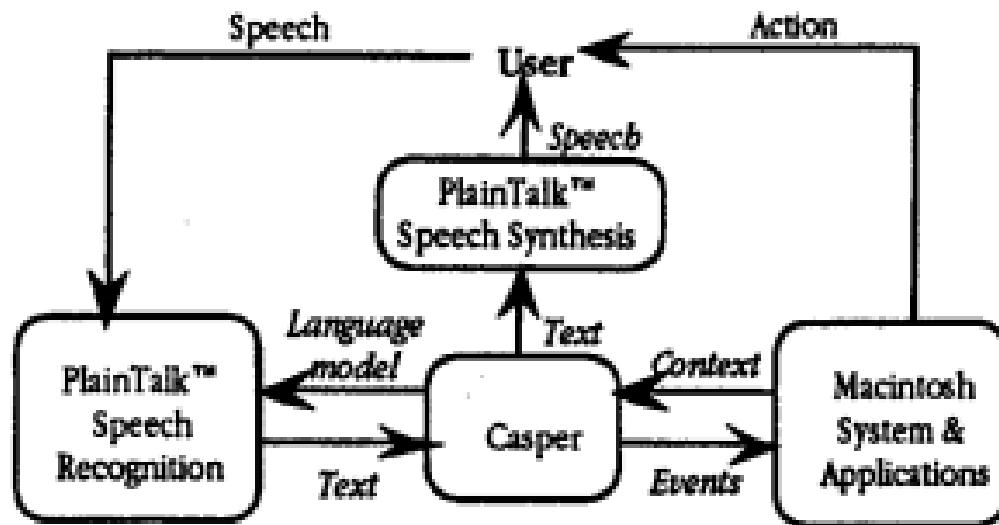


Figure 2.2: Casper system architecture [44]

2.4 Sound Classification using Machine Learning on Edge Device

Google introduced a breakthrough in machine learning when it introduced TensorFlow Lite in November 2017 [24]. It was a lightweight artificial intelligence (AI) solution for edge devices. The (.tflite) file is a flat-buffer generated from Keras model; its smaller in size and has quick initialization. It incurred cross-platform ability, where it could run on a wide range of hardware and software. It's optimized for edge devices and also supported hardware acceleration.

Related research was performed at the Florida Institute of Technology by a team lead by David Elliott under Dr. Carlos Otero, and Dr. Anthony Smith, where they introduced astounding techniques in Environmental sound classification (ESC) fo-

cused on office sound classification. This research provided spectacular results with an accuracy of 96.4% and a size of 20.6MB. This system was designed and implemented on a mobile device [21]. The objective of this thesis is to take things one step further and implement environmental sound classification on microcontrollers. A microcontroller has more stringent computational resources compared to microprocessors.

Microcontrollers are low energy consumption devices; they are small in size so that it can be embedded easily in any environment, at the cost of small memory, reduced processing power, and low storage capacity. Selecting the right microcontroller had to be done wisely to optimize machine learning. Machine learning isn't supported in all microcontrollers; here is a list of devices in Table 2.1, shortlisted, as TensorFlow lite is currently supported on ARM Cortex M processor family only.

Table 2.1: Microcontroller Compatible With Machine Learning

Name	Processor	Clock Speed	Memory	Ram	Cost \$
Arduino Nano 33 BLE Sense	32-bit ARM® Cortex™-M4	64MHz	1MB	256KB	33.40
SparkFun Edge - Apollo3 Blue	32-bit ARM® Cortex™-M4F	96MHz	1 MB	384Kb	14.95
ESP-EYE	Tensilica LX6	160MHz	4MB	8MB	25.01
STM32F746 DiscoveryKit	Arm® Cortex®-M7	32KHz	1MB	340KB	55.13
Adafruit EdgeBadge	ATSAMD51	120MHz	512KB	192KB	35.95
Adafruit - Bluefruit	Cortex M4 processor	64 MHz	2MB	256KB	24.95

2.5 Optimizing Machine Learning Model For Edge Devices

Optimization is a key constituent when executing an intensive process like machine learning on a device that has stringent computational resources. "Cyber-Physical Analytics: Environmental Sound Classification at the Edge"[21] is one of the essential papers to build this research on. The researchers of this paper successfully implemented sound classification on an android mobile device.

ESC-50 is an ideal data set used by the team to implement sound classification. Amplitude feature extraction was a technique proposed by the researchers of "Environmental Sound Classification at edge" [21]. It comprised of using Standard scoring normalization on each window of the data set; it helps in reducing latency in classification as there is little computational power needed in feature extraction.

2.5.1 Architecture Redesign

In 2017 authors of SqueezeNet designed a model architecture that was as accurate as Alex-Net but had 50X fewer parameters. A couple of ingenious strategies were used in designing the architecture of the Machine learning model [28].

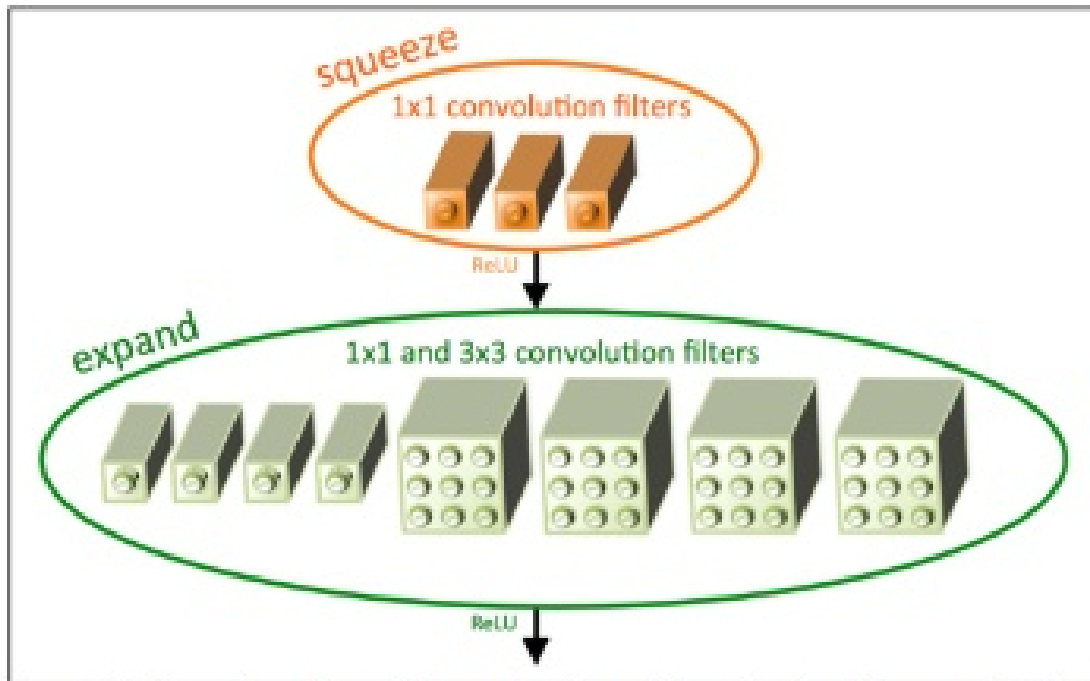


Figure 2.3: Fire Module [28]

The fire module is shown in Figure 2.3; it consists of a series of squeezes and expands. This technique helps to reduce the size of the model considerably, but it was designed for images. It will need alteration implemented to function effectively on sound signals [28].

2.5.2 Pruning

Magnitude based weight-pruning is a method in reducing the size of the model by five times. It uses a principle of eliminating unnecessary values in weighted tensor

[4]. Neural network parameters are set to zero in order to trim the low weight values from between the layers of the neural network [45]. Pruning benefits the use of computational resources optimally by removing unnecessary computation that contains zero value [4] [27]. Visualization of pruning in layers of a neural network is shown in Figure 2.4. This technique perhaps can trim down the model size.

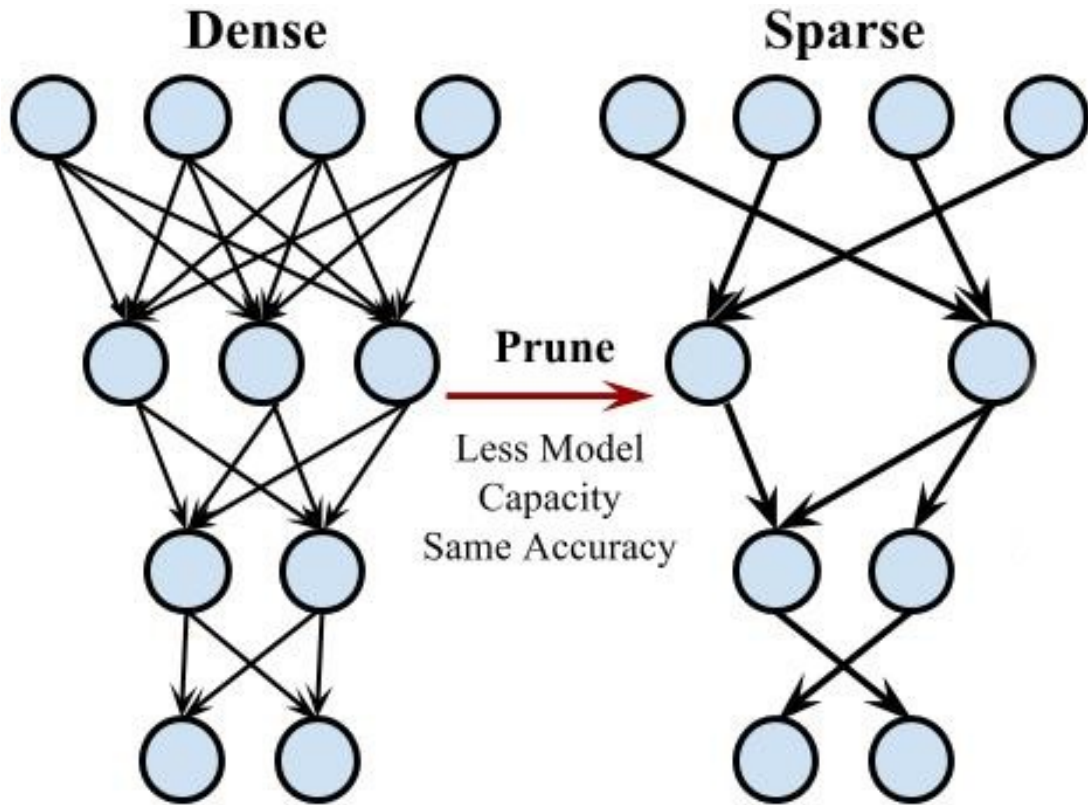


Figure 2.4: Pruning [43]

2.5.3 Quantization Awareness Training

A process of approximation of model parameters, to reduce the loss of accuracy during the quantization of the model. In Figure 2.5, A is the activation, and B is Bias. After the model is trained with quantization nodes, it is used to quantize the model to the desired precision (Eg. Float16, Int8, etc.) Quantization brings many

advantages, including reduced power, low latency, and reduced model size, which are incredibly useful for inferencing in edge devices [29].

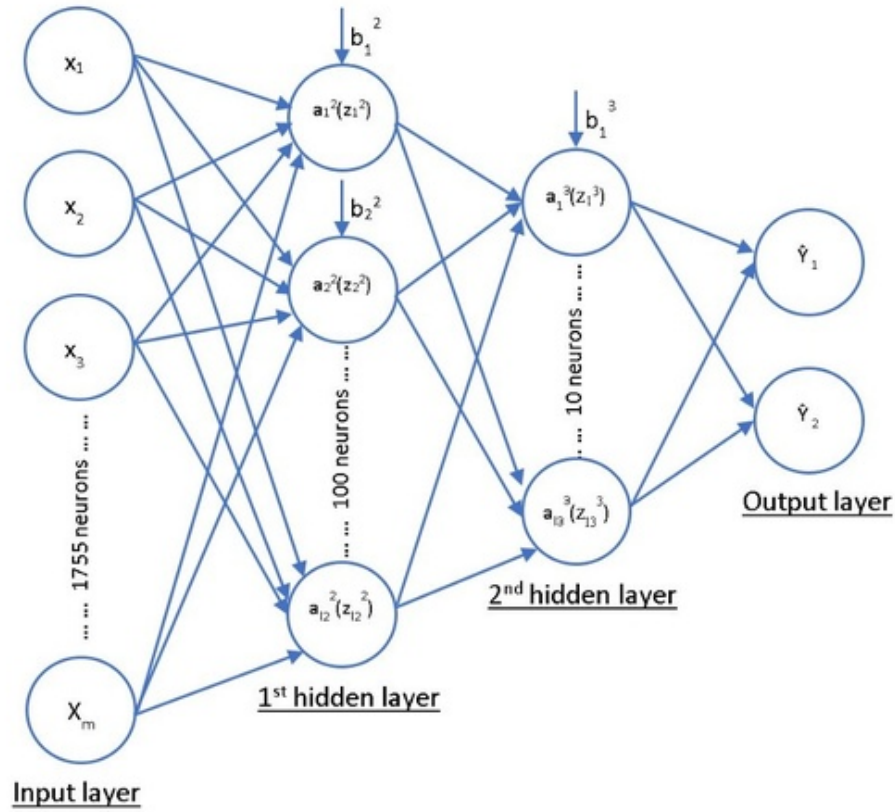


Figure 2.5: Quantization Aware Training [16]

As the Figure 2.5 illustrates there are fake nodes added to each layer, to visualize the result of quantizing through forward and backward passes and to learn ranges for each layer separately during the training cycle [14], bi-directional propagation's computes gradient error, which helps

As the Figure 2.5 illustrates, there are fake nodes added to each layer, to visualize the result of quantizing through forward and backward passes and to learn ranges for each layer separately during the training cycle [14]. Both forward and backward pass simulate activation and quantization of weights. Parameters are updated at high

precision during a backward pass, which ensures significant precision adjustments to parameters [24]. Maximum and minimum values are determined during training. This allows the model that is trained with quantization in a loop to be converted to a fixed point inference model. It also eliminates the requirement of calibration.

2.5.4 Quantization

Quantization is the process of representing a large set of values into a relatively small set of symbols or integers [13]. Quantization plays a key factor in model optimization. With a paradigm shift of Machine Learning and AI from a cloud-based system that has beefy computing power to an edge-based Machine learning and AI system, quantization has gained lots of contextual value. Tensorflow Lite converts the entire model into a flat buffer [21]. A computer uses a 32-bit floating-point to represent a real number for most applications; the innovational concept of quantization is to convert these 32-bit floating-point values to 8-bit integers, with a slight reduction of accuracy. This gives a distinctive result in model size of approximately 4X times compression [41] [29].

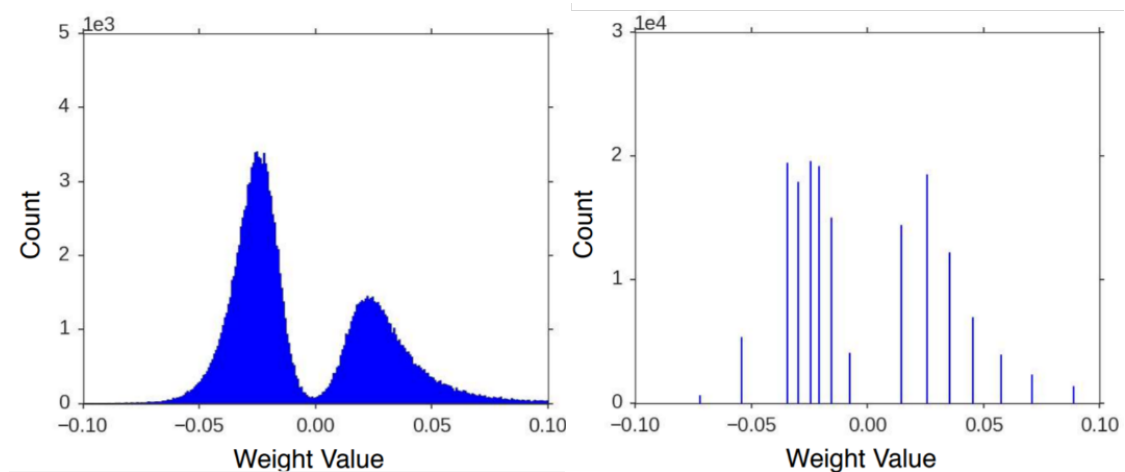


Figure 2.6: Quantization AlexNet [41]

Figure 2.6 is an example of AlexNet, with the original histogram on the left. Quantization discretizes to record important values while round-off the rest. Models are trained using very tiny gradient updates, for which we do need high precision [41]. Quantization can be one of the techniques that can be used for model optimization for edge devices.

Chapter 3

Proposed Approach

Analysis needs an adequate amount of computational power, with technology advancing at a rapid pace, electronic devices are getting smarter, smaller, and faster. We propose to design a system where Machine learning can harness the power from edge devices efficiently and effectively. Complex computation can be performed at the edge with minimal resources. This has given rise to a new era called "Edge Based ML" [31].

Figure 3.1 is a proposed flow chart for designing a system that generates optimized models for edge devices. This flow chart focuses on developing a system that generates an optimized model to interpret sound or speech signals over the edge with minimal utilization of computational resources.

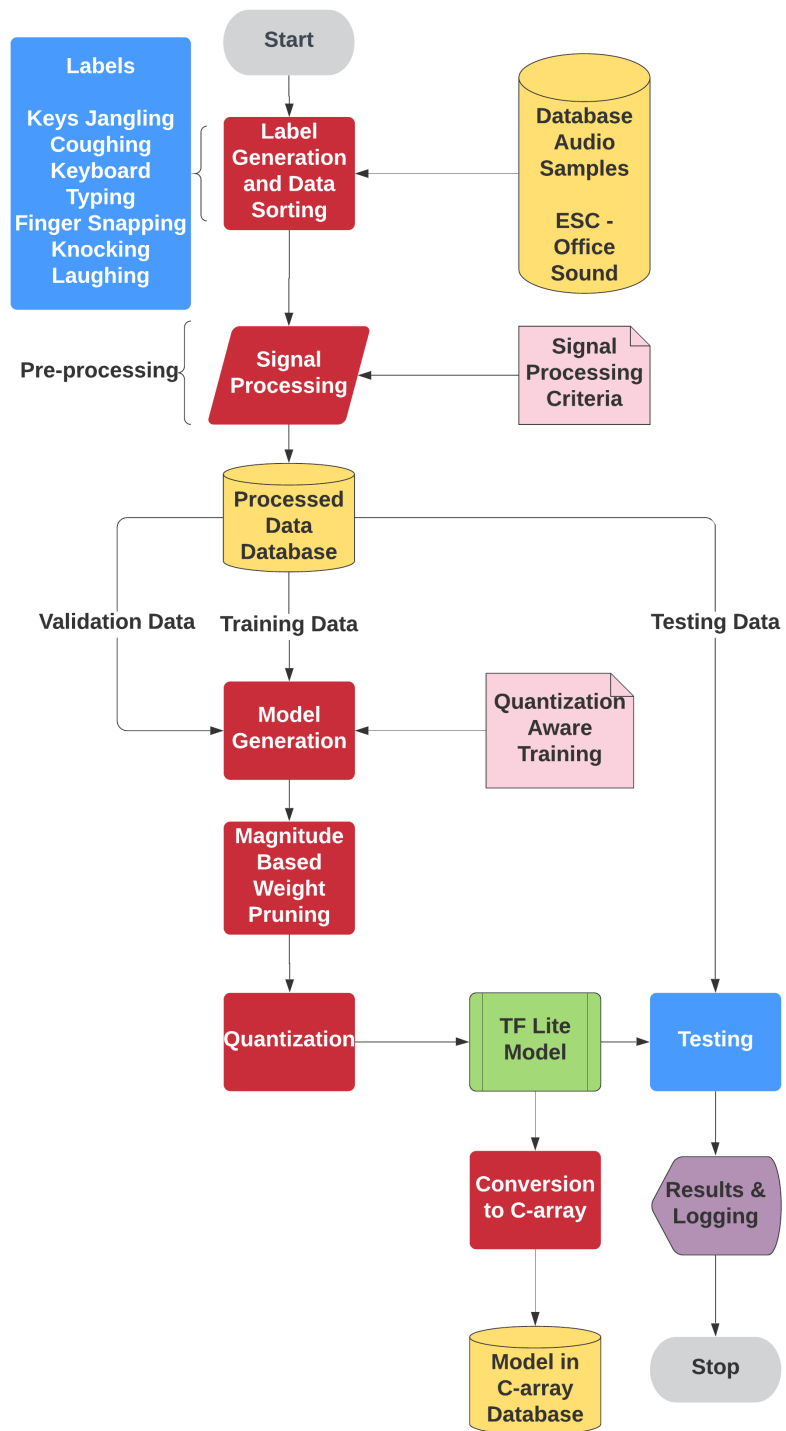


Figure 3.1: Flow Chart For Proposed Approach

Reliable and quality data plays a crucial roll in advance data learning technology; attention to data collection, labeling, and preparation before analysis is critical [25]. ESC-50 (Environmental Sound Classification) [39] and DCASE 2018 [22] consists of audio recordings; these recordings mark a benchmark and are ideal data set for machine learning; they comprise of (.wav), which are lossless and accurate [9]. The system further categorizes these sounds appropriately by attaching labels to them. Data processing develops raw data into a usable form. Machine learning relies on extensive amounts of data. The data should be provided to the system in the appropriate format to achieve adequate analysis. The four steps are shown in Figure 3.2 generalized machine learning system for audio signals.

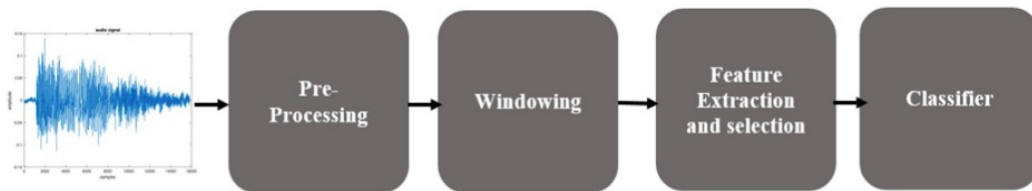


Figure 3.2: Typical Machine Learning System For Audio Signals

The categorized raw (.wav) files are fed into the system, as shown in Figure 3.2, a system is initialized with data pre-processing. Noise cancellation and silence removal are part of pre-processing the audio data, which isn't required for our data set because ESC-50 and DCASE 2018 was developed in a controlled environment which eliminates any noise or silence.

We selected a 1 second window with a 0.5 second hop length, thus results in overlapping windows. The last window is padded with zeros to meet the desired size [21]. Feature extraction is invoked by using Standard scoring normalization on each window in our system [21]. Due to this, lightweight system edge devices will be able to perform classification seamlessly.

We proposed in enforcing Quantization awareness training as soon as the model architecture is loaded. Quantization awareness training will help maintain accuracy when the model is quantized. As the model is generated, it's passed through a pruning algorithm; this algorithm discards unwanted low or zero weight tensors; there should be a reduction in size observed in the pruned model. Quantization is implemented on the pruned model using TensorFlow-lite, which converts the model to a flat buffer and also converts 32bit floating-point values to 8-bit integers. This results in a tremendous reduction of size and generates a (.tflite) model that's lightweight and small, ideal for edge devices. This TensorFlow-lite model is altered into a C-Array for microcontrollers as most of them run on C++ programming language.

The database provides a couple of segregated testing data that the model hasn't witnessed; it is used by the system to evaluate the prediction accuracy for each category of sound. This data helps to plot a confusion matrix that can be the model's assessment criteria.

Chapter 4

Implementation

4.1 Conventional Edge Devices

Edge devices are categorized into two major types Micro Controller (MCU), and Micro Processor (MPU) based system. You can use either of the two as a tool at the edge. We will discuss the in-sites of these tools in Sections 4.1.1 and 4.1.2.

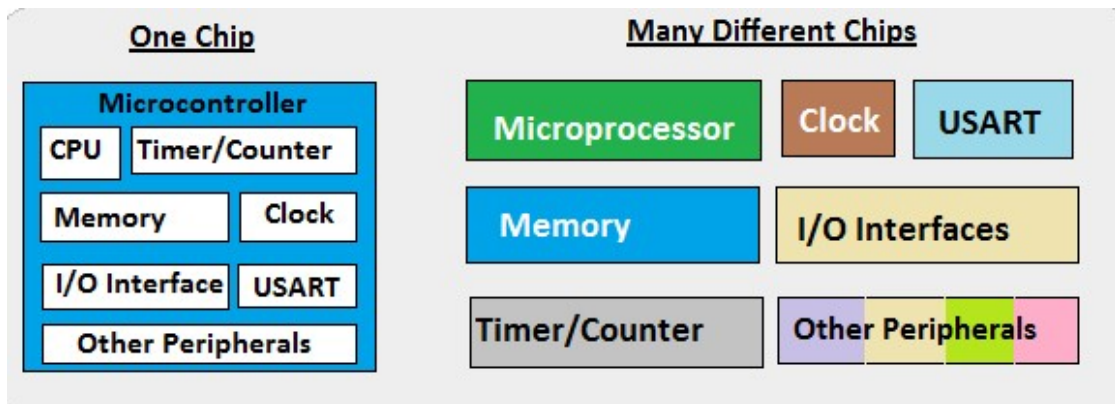


Figure 4.1: Casper system architecture [44]

4.1.1 Microprocessor

A microprocessor is defined as an integrated circuit in a digital computer system that performs the central processing function [38]. The first Microprocessor was invented by Intel in the year 1971. It consisted of only 2300 transistors; it was a 4-bit processor that was a 16-pin package [2]. A microprocessor is one part of the computer system; a microprocessor cannot function alone, it must be connected to other units, generally on a shared or 'party line' basis, so as to provide all the functions of a complete computing system [20]. Microprocessor is connected RAM, ROM and I/O interface unit using the address and data bus. Figure 4.2, shows the basic block diagram of a microprocessor.

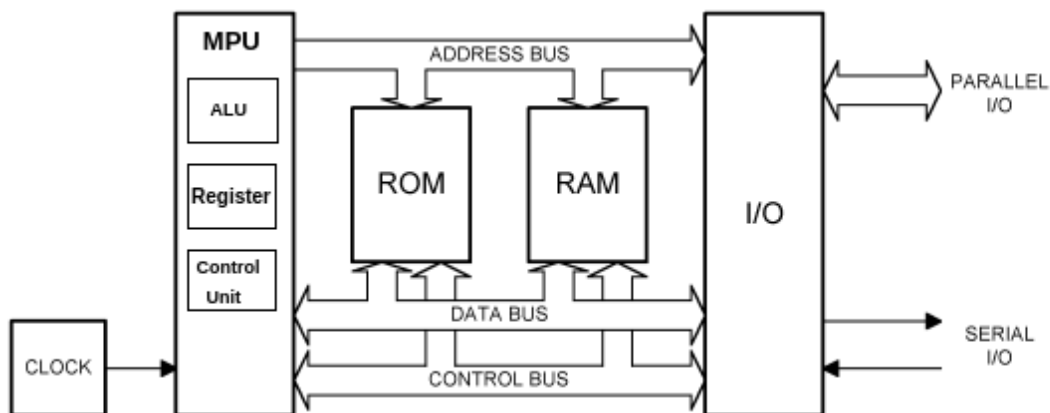


Figure 4.2: Basic Microprocessor Architecture [5]

4.1.2 Microcontroller

Microcontroller are really tiny yet really powerful; with the invention of microcontroller it innovated embedded systems. A microcontroller is a very large scale integrated circuit (VLSI) that contains [40].

- Central Processing Unit (CPU)
- (ROM – Read Only Memory)
- (RAM – Random Access Memory)
- Timers and Counters
- I/O Ports (I/O – Input/Output)
- Serial Communication Interface
- Clock Circuit (Oscillator Circuit)
- Interrupt Mechanism

A microcontroller is also known as a single chip computer; it can operate stand-alone unlike microprocessors. As microcontrollers got innovated, there were several resources added to the single chip computer, it was introduced with Analog to Digital converter (ADC), Digital to Analog converter (DAC), Serial Universal Asynchronous Transmitter and Receiver, Comparators, Pulse Width Modulation module, Master Synchronous Serial Port for SPI (Serial Peripheral Interface)/I2C (Inter Integrated Circuit) communications, USB port, ethernet port, on-chip oscillators, along with a host of other peripherals [30]. Figure 4.3 shows the basic block diagram of a microcontroller.

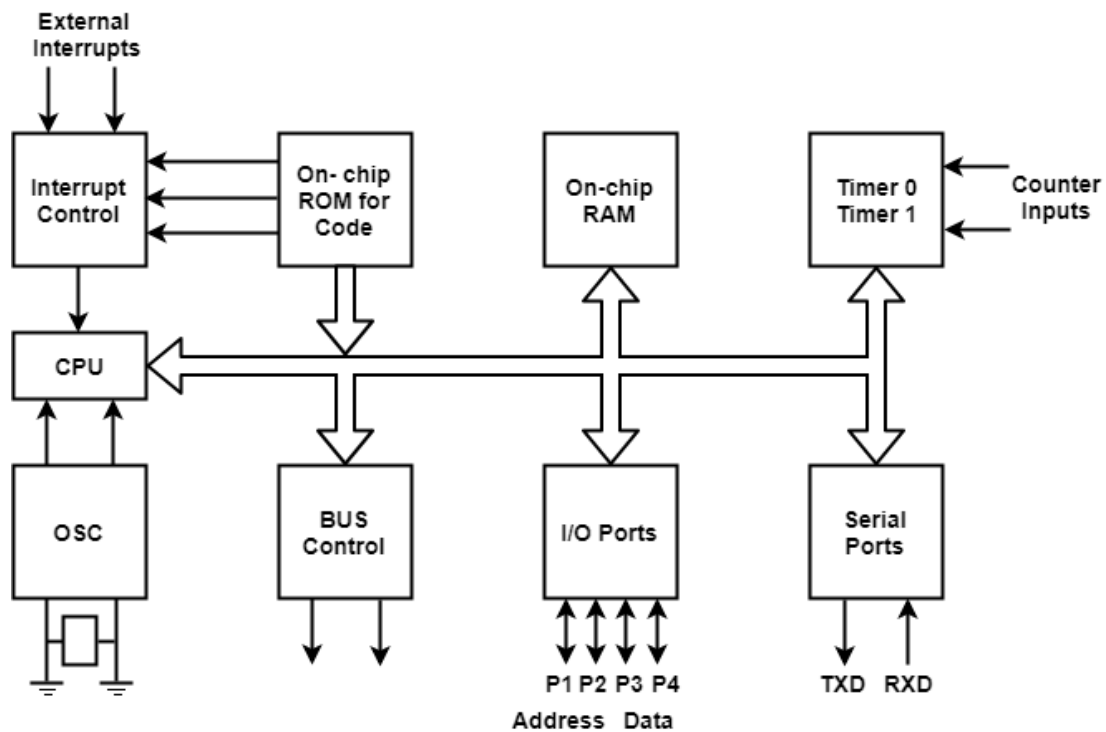


Figure 4.3: Basic Microcontroller Architecture [5]

4.2 Hardware Selection

After carefully scrutinizing Sections, 4.1.1 and 4.1.2 observed that microcontroller has the edge over microprocessor for edge devices.

- Cost Effective
- Small in size
- Low power consumption
- Reliable

4.3 Pre-requisites

There are a couple of hardware and software pre-requisites that are essential in building a sound classification model for an edge device. Our development and designing are implemented on Ubuntu 18.04.4 (Bionic Beaver); this platform is chosen because of the flexibility the OS provides, along with being an open-source system that made it an ideal operating system for development.

As python deals with lots of packages and dependencies, it was pragmatic to create a virtual environment. A virtual environment is lightweight directories isolated from the system directory and have their own Python binary [1]. Anaconda was a quintessential match to develop the system; it's an opensource application that handles package dependency, and it's armed with a bunch of opensource channels. It handles the virtual environment smoothly and efficiently, as showing in Figure 4.4.

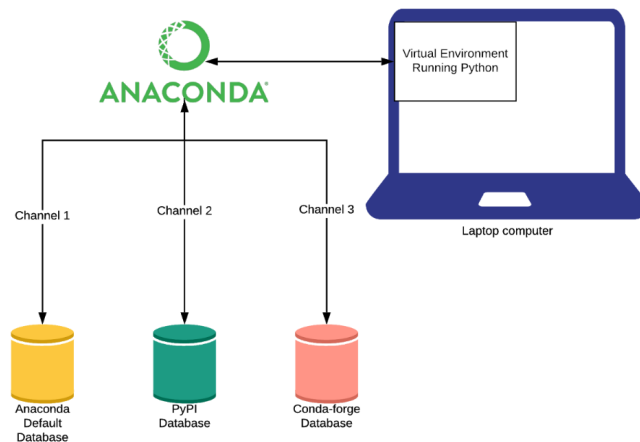


Figure 4.4: Anaconda

Table 4.1 provides a list of all necessary software for developing the model and uploading it to Arduino Nano Ble 33 Sence.

Table 4.1: List of Software

Software	Application
Pycharm/ Jupiter Notebooks	Integrated Development Environment for Python
Anaconda	Opensource software that handles virtual environment and python packages.
Arduino IDE	IDE is a cross-platform and a link between board and computer .
Python 3.6	Higher-level programming language.

Table 4.2 provides a list of all python packages along with their versions, necessary to generate the model. Choosing the right package version is essential due to their inter-dependency.

Table 4.2: List of Dependent Python Packages

Package Name	Application	Version
Tensorflow-gpu	Opensource Machine Learning Framework	1.14.0
Cuda toolkit	Creates a developing environment to Enforce Machine Learning on High-Performance GPU.	10.1.243
Keras-GPU	High-Level Neural network API that runs on TensorFlow.	2.0.8
librosa	Library for Audio and Music Processing.	0.7.2
numpy	Python array computing kit.	1.18.1
wandb	Command-line program and library for Weights and Biases API interactions.	0.8.30
TensorFlow-model-optimization	A library in TensorFlow to perform constrained optimization.	0.2.1
multiprocessing	Backport of the multiprocessing package to Python.	default
matplotlib	Plotting Package for Python.	3.1.3
scikit-learn	It features various classification, clustering, and regression algorithm.	0.22.1
os	Interfaces python with the operating system.	default
seaborn	Statistical visualization of the data.	0.10.0
tqdm	Smart, progressive meter.	4.42.0

4.4 Segregating and Data Collection

Carefully selected and shortlisted ESC-50 Data (Environmental Sound Classification), DCASE Data set, coupled with few lab collected data to generate a model that can classify sound on edge devices. The table below shows the collection of (.wav) files. The (.wav) file format was selected because it's highly accurate and lossless format of sound, as discussed in chapter 3.

Table 4.3: List of Sounds

Label	DCASE	ESC-50	Collected	Combined
Knocking	270	40	88	407
Laughing	290	40	68	398
Typing	119	40	43	202
Coughing	243	40	3	286
Keys Jangling	99	40	7	146
Snap	77	40	52	169
Total	1107	240	261	1608

Data is a critical aspect of machine learning; it must be organized and sorted before it is being processed. To handle this, we developed a code to organize data and folders that will be used during execution. The code uses (OS) library; it helps in interfacing python with the operating system. The system generates essential directories for organizing data, logging, and output.

4.5 Signal processing parameters

- **Sampling rate** - The human audible sound range is from 20Hz to 20KHz with a maximum intensity of 20dB. A microphone captures an audio signal, its analog in nature. It can be represented in digital form by sampling the audio signal and discretizing the signal in the time domain. It should meet the minimum criteria, which are at least two samples per cycle, also defined as the Nyquist rate [23]. The system is designed to dynamically accept the sampling rate from the user to process the sound signal depending on its application.
- **Audio length** - Audio length is used to convert the raw audio signal to fixed predefined length in milliseconds. Detailed processing of an audio signal is explained in Section 4.6.
- **Channel** - It is a passage in which a sound signal is transported. For simplicity, the system is designed to convert the raw signal to a single channel. A single channel throws a monophonic output.

4.6 Pre-processing

The raw data directory contains the categorized (.wav) files stored in their appropriate folders; each folder name corresponds to its label. As shown in the Section 4.3, there are 1608 sound files stored in 6 categories. The "get label" function browses through the raw file directory and retrieves the label names and label indices.

We enable multiprocessing to process the data as it saves time when executing a complex mathematical calculation over a wide range of values. The system retrieves all raw, categorized audio files, and loads them into an audio buffer using Librosa. Librosa handles audio files seamlessly and efficiently, also helps integrates the audio data into Python.

The raw wave file is plotted in Figure 4.5 is an example of a visual representation of data; ESC-50 Knocking (1-81001-A-30.wav) was used. In order to comprehend and visualize the pre-processing easily, we have sampled the data at 16000 and intend to generate clusters of 3 seconds. The original file is 705bits and Sampled at 44100 and is 5 seconds long as shown below

Note: that the parameters mentioned above were for explanatory and visualization purposes only. The actual training was conducted with the following parameters. Data were sampled at a rate of 30000 per second. The window size was 1second. Window Hop was 0.5 seconds.

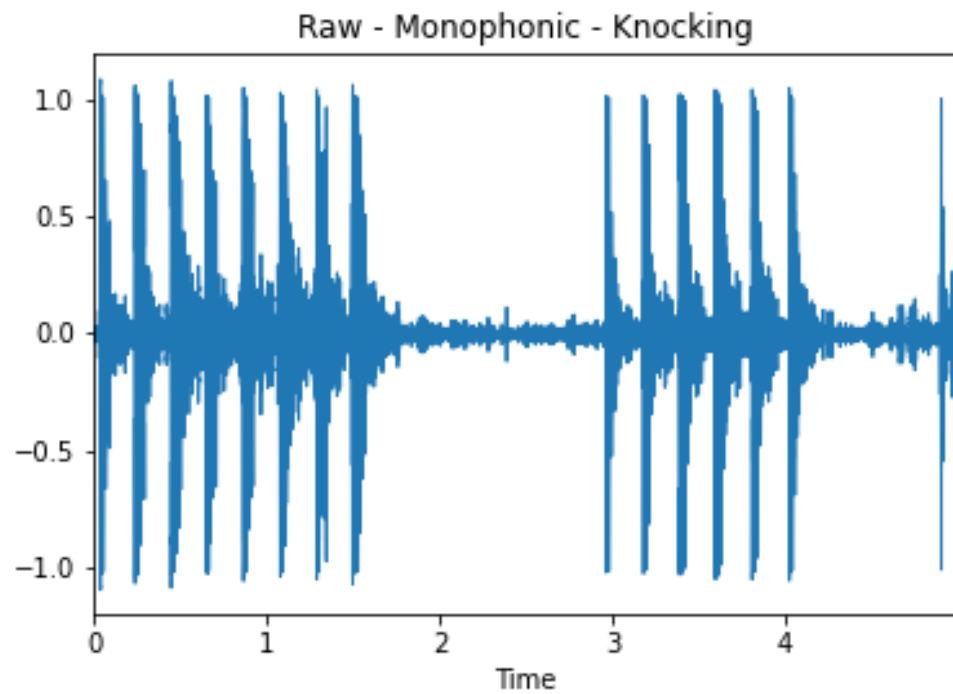


Figure 4.5: Raw Wave Form ESC-50 Knocking (1-81001-A-30.wav) [39]

The mean (μ) and standard deviation (σ) are computed for the given audio file. Standard Scoring Normalization(Z-Score) is applied to the signal [32].

$$Z = (X - \mu)/\sigma$$

X = Raw data

(μ) = Mean

(σ) = Standard Deviation

Figure 4.6 shows the visualization of the sample data once Standard Scoring Normalization is applied to the raw wave file.

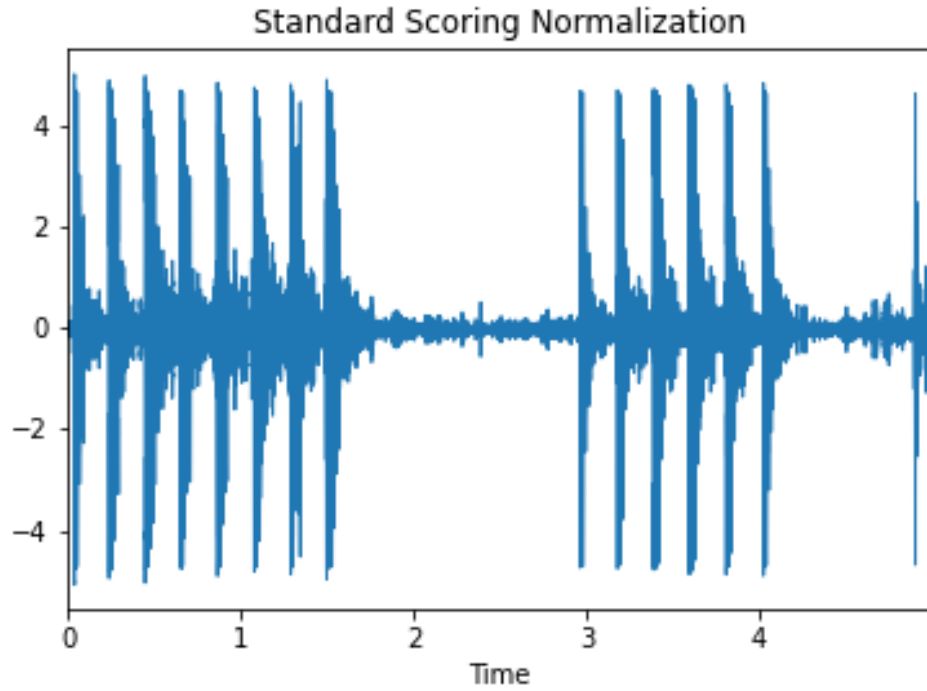


Figure 4.6: Standard Scoring Normalization

The window size and its hop length are critical in the generation of clusters. Most neural networks accept variable length size of waveform, but this system is designed to generate fixed-length clusters, and cluster length is equal to the window length. Fixed length was chosen as that technique proves to be more reliable as stated in the paper Environmental sound classification [21].

$$Window\ Hop = (0.5 \times Window\ length)$$

Therefore our example window size is 3 seconds, and the hop length is 1.5 seconds, i.e., the window is slid to the right by 1.5 seconds. As sounds signals are continuous, it is essential to capture the transition of sound in the time domain as each state

is dependent on the previous state, and the next state will depend on the current one.

Figure 4.7 shows the first 3 second window captured after processing, from 0 seconds to 3 seconds. Figure 4.8 shows the second, 3 second window captured after a hop of 1.5 seconds to the right, as mentioned in Formula above, this window starts from 1.5 seconds to 4.5 seconds.

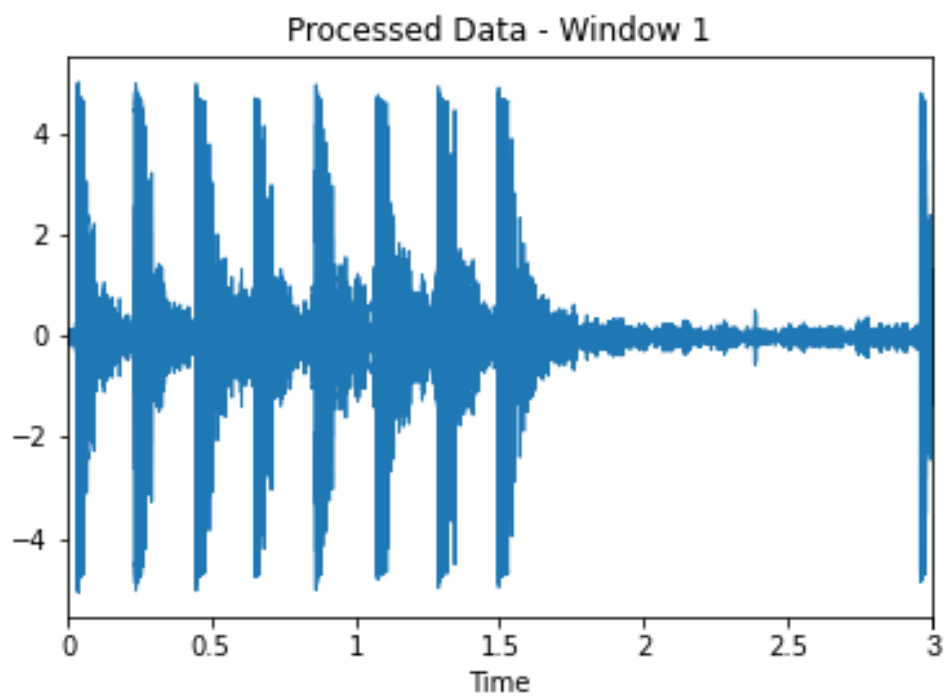


Figure 4.7: Processed Data 1st Window

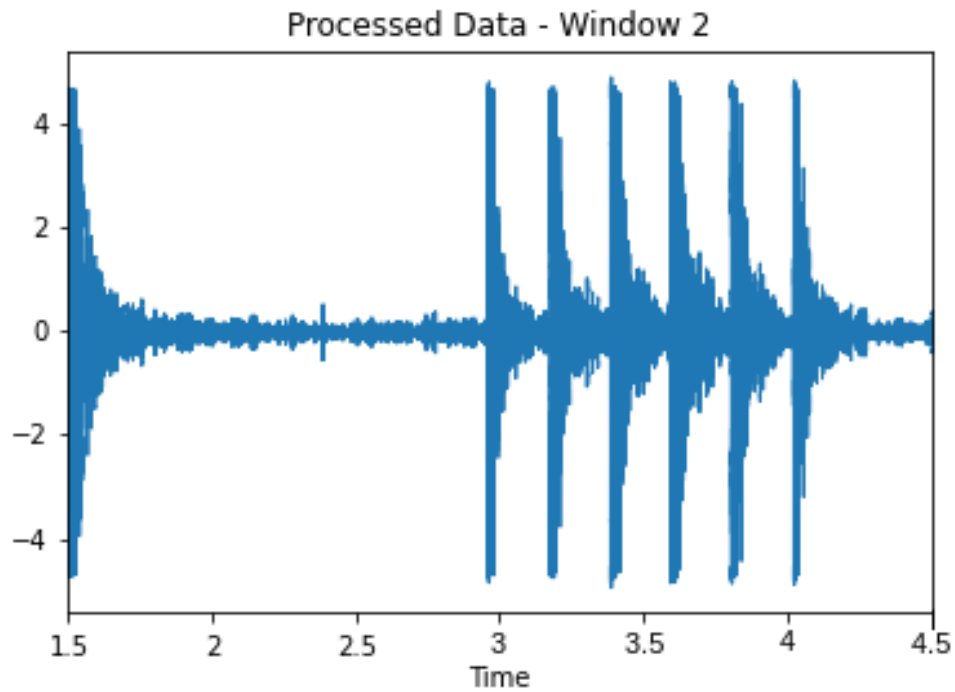


Figure 4.8: Processed Data 2nd Window

After a hop of another 1.5 seconds, the third window will capture data from 3rd second to 6th second. But the raw file contains data only until 5 seconds, as shown in Figure 4.5. To keep the window size consistent, the system pads the data with zeros for the final window until the desired window length is achieved, as shown in Figure 4.9. Padding is done to get a uniform length on each window; it helps to prevent losing data at the corners and also prevents from losing essential data while downsizing in the neural network.

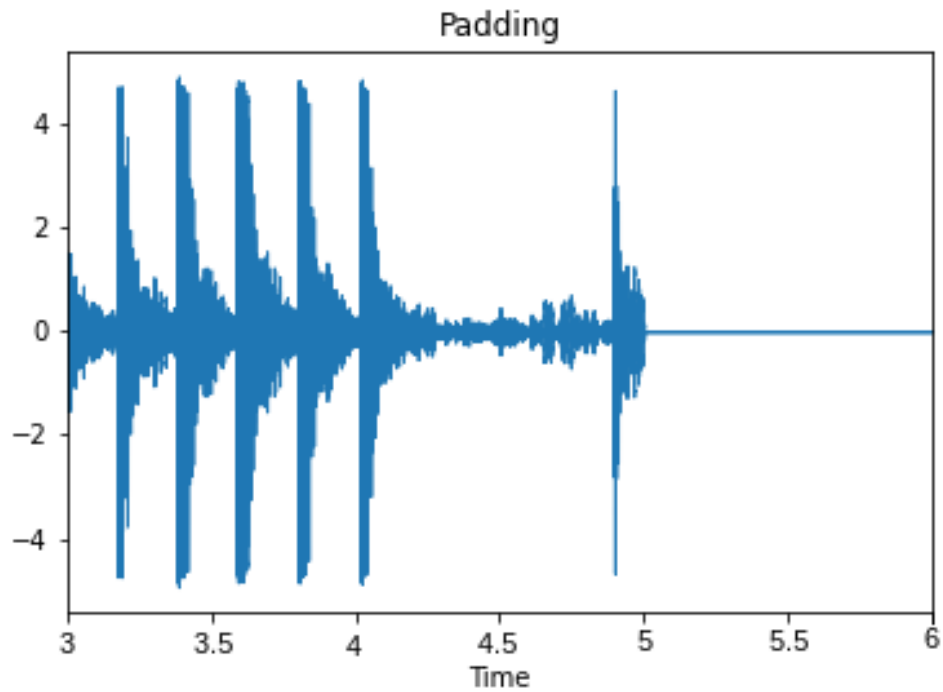


Figure 4.9: Processed Data 3rd Window With Padding

Window data is stored in the form of a numpy array. As there are several windows, there are multiple numpy arrays created. Each numpy array is appended to the previous one. This pre-processing is performed on the entire data set of a particular label category. Eventually, the system generates a huge list of appended numpy arrays which is stored in a file of (.npy) format.

4.7 Data Organization and Conversion from 1D to 2D

The program loads the processed data according to the labels from the (.npz) files stored in the directory. The program uses a 90% Split Ratio where the entire data set is segregated, 81% of the processed data is allocated to Training, 9% to Validation, and 10% to Testing. Function "train test split" is used for allocating and categorization of data. Split ratio value is a parameter taken from the user.

Note: After concatenating all (.npz) files, data shape generated after training with actual parameters is (19400, 30000, 1). In which the data contains 19400 arrays of sound, and each one has an audio length of 30000.

While implementing Quantization aware training, explained in Section 4.9, we came across several hurdles and learned that Quantization aware training could only be implemented on 2-dimensional layers. Moreover, TensorFlow lite has a limited support subset for microcontroller implementation [8], which only includes 2-dimensional layers. Converting processed data from one dimension to two dimension was essential.

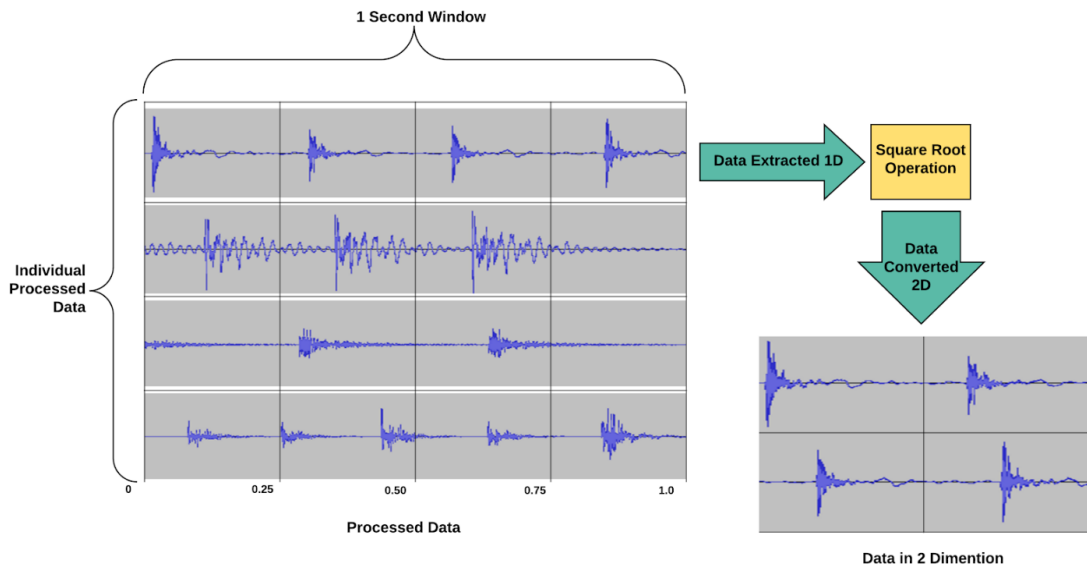


Figure 4.10: Converting 1D Array to 2D [21]

Figure 4.10 describes the conversion from one dimension to two dimensions. The processed data is extracted from the (.npy) file. Each row contains a unique sample of a fixed length. Each sample is processed over a math square root operator, where a one dimensional array is converted to a two dimensional array. In Figure 4.10, original data has a shape of (4,4,1) after it's processed over a math square root operator; its shape is (4,2,2,1). This data can be effectively fed into a two dimensional neural network.

Note: Once the actual processed data of shape (19400, 30000, 1) is generated its fed to a math square root operator, it performs square root operation on a fixed audio length of 30000 samples per second for each array of sound and converts the data to a 2-dimensional array. This generates data of shape (19400,173,173,1). After passing this data to the split ratio function, which is tuned to a 90% split ratio, the actual data is split as follows.

- Training Data Shape - (15714, 173, 173, 1)
- Validation Data Shape - (1746, 173, 173, 1)
- Testing Data shape - (1940, 173, 173, 1)

4.8 Model

With innovation in electronics, a fine mesh of computational node is placed over the edge; this intern helps in reducing latency of processing data over the cloud; edge devices have limited processing power and storage. There has always been a trade-off between accuracy and model size. The system developed has a model architecture that is similar to SqueezeNet architecture. With SqueezeNet they are able to compress the model to 0.5Mb, which is impressive but isn't enough for our application; hence we had to refer to the design architecture and generate our own model with reference to squeeze net.

The design is based on using fire modules, as shown in Figure 2.3. While designing our model three strict guidelines were maintained in the design strategy

- **Strategy 1** - Replace 3X3 filters with 1x1 - Choose 1x1 convolution filters over 3x3 because of 9X fewer parameters [28].
- **Strategy 2** - Decrease the number of input channels to 3X3 filter - Design a layer that holds few parameters. Fewer parameters lead to smaller model size[28] .

$$Parameters = (Input\ Channels) \times (Num\ Filters) \times (Size\ of\ Filter)$$

- **Strategy 3** - Down sample late in the network so that convolution layers have large activation maps [28].

The fire module shown in Figure 2.3 comprises two segments, squeeze and expand. The squeeze layer only includes S(1X1) filters [28]. The expand convolution layer comprises of E(1X1) and E(3X3) convolution filters. The squeeze layer fulfills

the 1st strategy as it limits the number of input parameters. The expansion layer consists of a few $E(1 \times 1)$, and $E(3 \times 3)$ filter; hence the total number of filters in expansion module is $E(1 \times 1) + E(3 \times 3)$; consequently the squeeze layer helps to limit the channel input to the expand layer. This technique, in turn, fulfills strategy two by reducing the total number of parameters. Max pooling is placed relatively late in the architecture, which is the main factor for downsizing, and it satisfies strategy three [28].

Figure 4.11 contains model architecture design based on squeeze net's strategy while designing this architecture; we excluded a few layers. Channels were changed from "channels first " to "channels last," adjusting the model to implement the arrangement.

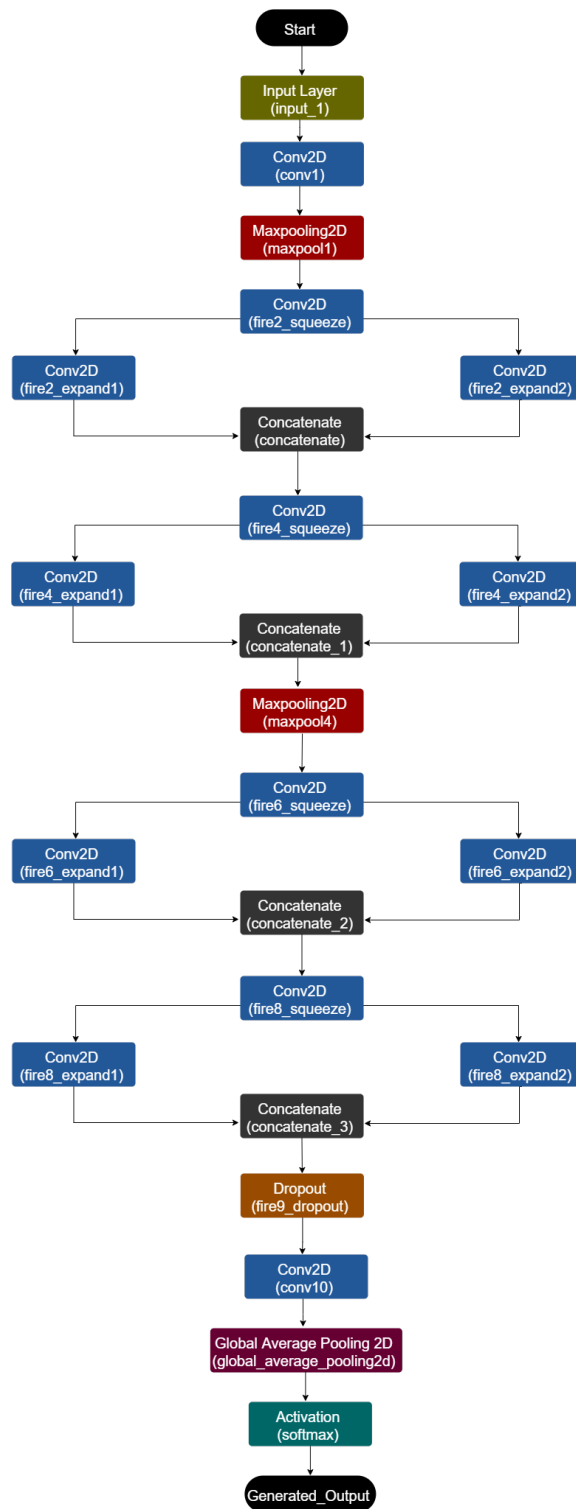


Figure 4.11: Model Flowchart [28]

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 173, 173, 1)]	0	
conv1 (Conv2D)	(None, 87, 87, 96)	4800	input_1[0][0]
maxpool1 (MaxPooling2D)	(None, 43, 43, 96)	0	conv1[0][0]
fire2_squeeze (Conv2D)	(None, 43, 43, 16)	1552	maxpool1[0][0]
fire2_expand1 (Conv2D)	(None, 43, 43, 64)	1088	fire2_squeeze[0][0]
fire2_expand2 (Conv2D)	(None, 43, 43, 64)	9280	fire2_squeeze[0][0]
concatenate (Concatenate)	(None, 43, 43, 128)	0	fire2_expand1[0][0] fire2_expand2[0][0]
fire4_squeeze (Conv2D)	(None, 43, 43, 32)	4128	concatenate[0][0]
fire4_expand1 (Conv2D)	(None, 43, 43, 128)	4224	fire4_squeeze[0][0]
fire4_expand2 (Conv2D)	(None, 43, 43, 128)	36992	fire4_squeeze[0][0]
concatenate_1 (Concatenate)	(None, 43, 43, 256)	0	fire4_expand1[0][0] fire4_expand2[0][0]
maxpool4 (MaxPooling2D)	(None, 21, 21, 256)	0	concatenate_1[0][0]
fire6_squeeze (Conv2D)	(None, 21, 21, 48)	12336	maxpool4[0][0]
fire6_expand1 (Conv2D)	(None, 21, 21, 192)	9408	fire6_squeeze[0][0]
fire6_expand2 (Conv2D)	(None, 21, 21, 192)	83136	fire6_squeeze[0][0]
concatenate_2 (Concatenate)	(None, 21, 21, 384)	0	fire6_expand1[0][0] fire6_expand2[0][0]
fire8_squeeze (Conv2D)	(None, 21, 21, 64)	24640	concatenate_2[0][0]
fire8_expand1 (Conv2D)	(None, 21, 21, 256)	16640	fire8_squeeze[0][0]
fire8_expand2 (Conv2D)	(None, 21, 21, 256)	147712	fire8_squeeze[0][0]
concatenate_3 (Concatenate)	(None, 21, 21, 512)	0	fire8_expand1[0][0] fire8_expand2[0][0]
fire9_dropout (Dropout)	(None, 21, 21, 512)	0	concatenate_3[0][0]
conv10 (Conv2D)	(None, 21, 21, 6)	3078	fire9_dropout[0][0]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 6)	0	conv10[0][0]
softmax (Activation)	(None, 6)	0	global_average_pooling2d[0][0]
Total params: 359,014			
Trainable params: 359,014			
Non-trainable params: 0			

Figure 4.12: System Generated Model Architecture [28]

4.9 Quantization Aware Training

Quantization is a lossy process. “Quantization is the process of transforming an ML model into an equivalent representation that uses parameters and computations at a lower precision.[7]” Quantization is a lossy process because it moves from a higher precision to a lower precision. Figure 4.13 shows the representation of floating-point values is represented into a fixed int value range.

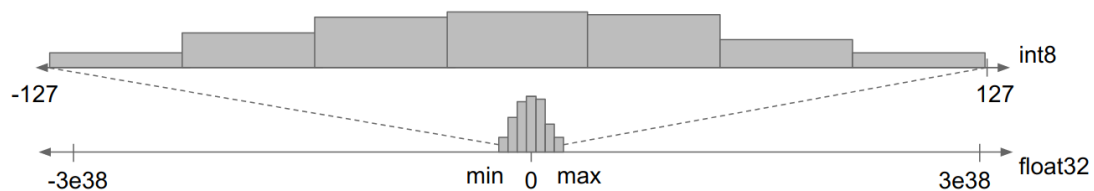


Figure 4.13: Quantization using TensorFlow lite Operation [7]

The fundamental precept of quantization aware training is to simulate a low-precision inference-time computation in the forward pass. There are fake nodes introduced; these nodes convert the floating-point numbers to low precision values and also reverses the process by converting the low precision numbers back to floating-point values. It ensures that losses from quantization are introduced into the computation that emulates low precision. Each floating-point value is mapped to one low precision number in a tensor. This will help in reducing losses. In Figure 2.5, `wt quant` and `act quant` introduces quantization losses in the forward pass inference [41]. This technique prevents loss of accuracy while quantizing the model that's explained in Section [41].

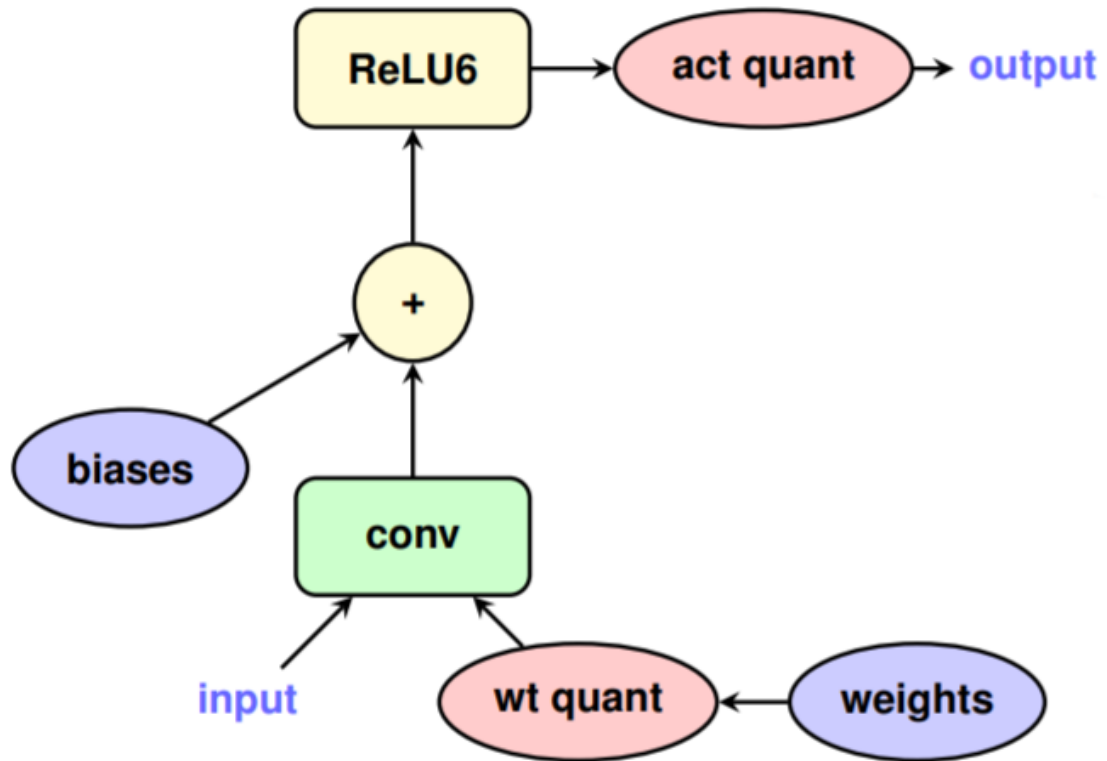


Figure 4.14: Quantization Aware Training [41]

Another important guideline that needs to be followed is Quantization aware training can only be inferred on two dimensional Convolution networks; hence Section 4.7 shows the conversion of data from one dimension to two dimensions.

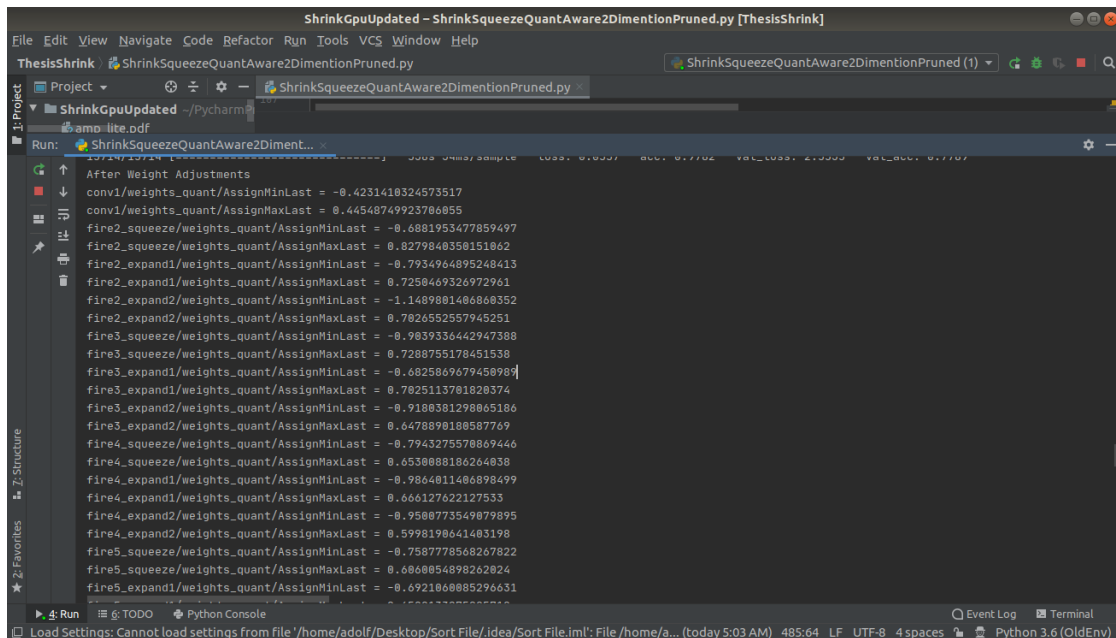


Figure 4.15: Quantization Aware Training Weight Adjustments [41]

4.10 Magnitude Based Weight Pruning

"Magnitude Based Weight Pruning means, eliminating unnecessary values in the weight tensor [4]."

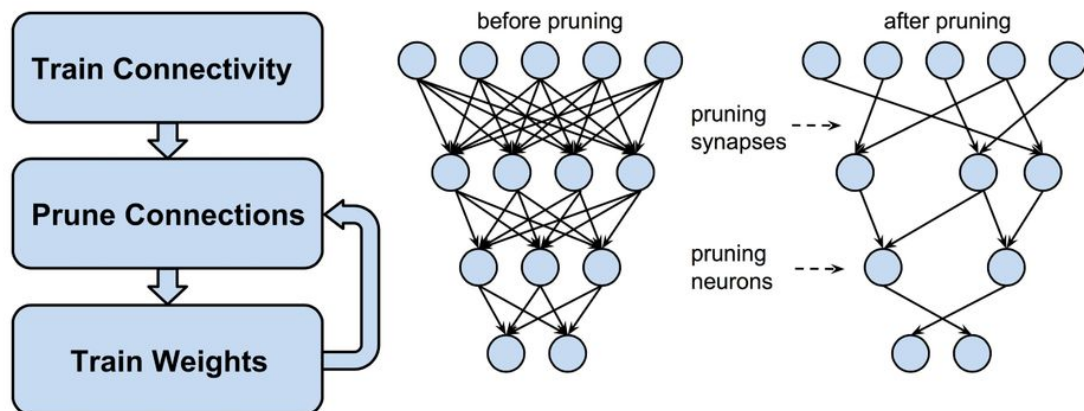


Figure 4.16: Pruning Technique [26]

Pruning reduces the model size up to five times. This process can aid in reducing the size of the model with which is critical on memory constraint edge devices. The saved model file is loaded, and the architecture is integrated with pruneable hyper-parameters.

- **Sparsity** - Initial and Final Sparsity, we start at initial sparsity and gradually train the model to reach final sparsity. Our system is set at an initial sparsity of 50% and a final sparsity of 80%. These inferences the amount of pruning done.
- **Frequency** - Frequency is the rate of pruning. It is the time given for the model to recover from the pruning. Our system is set to a frequency of 50.
- **Begin step** - Once the model acquires the desired accuracy, pruning begins. Begin step is set to 0 so that we train for the desired number of pruning epochs.

Figure 4.17 is the implementation of pruning on our model. It contains 359,014 prunable parameters out of the total 716,523 parameters .

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 173, 173, 1) 0		
prune_low_magnitude_conv1 (Prun	(None, 87, 87, 96)	9506	input_1[0][0]
prune_low_magnitude_maxpool1 (P	(None, 43, 43, 96)	1	prune_low_magnitude_conv1[0][0]
prune_low_magnitude_fire2_squee	(None, 43, 43, 16)	3090	prune_low_magnitude_maxpool1[0][0]
prune_low_magnitude_fire2_expan	(None, 43, 43, 64)	2114	prune_low_magnitude_fire2_squeeze
prune_low_magnitude_fire2_expan	(None, 43, 43, 64)	18498	prune_low_magnitude_fire2_squeeze
prune_low_magnitude_concatenate	(None, 43, 43, 128)	1	prune_low_magnitude_fire2_expand1 prune_low_magnitude_fire2_expand2
prune_low_magnitude_fire4_squee	(None, 43, 43, 32)	8226	prune_low_magnitude_concatenate[0]
prune_low_magnitude_fire4_expan	(None, 43, 43, 128)	8322	prune_low_magnitude_fire4_squeeze
prune_low_magnitude_fire4_expan	(None, 43, 43, 128)	73858	prune_low_magnitude_fire4_squeeze
prune_low_magnitude_concatenate	(None, 43, 43, 256)	1	prune_low_magnitude_fire4_expand1 prune_low_magnitude_fire4_expand2
prune_low_magnitude_maxpool4 (P	(None, 21, 21, 256)	1	prune_low_magnitude_concatenate_1
prune_low_magnitude_fire6_squee	(None, 21, 21, 48)	24626	prune_low_magnitude_maxpool4[0][0]
prune_low_magnitude_fire6_expan	(None, 21, 21, 192)	18626	prune_low_magnitude_fire6_squeeze
prune_low_magnitude_fire6_expan	(None, 21, 21, 192)	166082	prune_low_magnitude_fire6_squeeze
prune_low_magnitude_concatenate	(None, 21, 21, 384)	1	prune_low_magnitude_fire6_expand1 prune_low_magnitude_fire6_expand2
prune_low_magnitude_fire8_squee	(None, 21, 21, 64)	49218	prune_low_magnitude_concatenate_2
prune_low_magnitude_fire8_expan	(None, 21, 21, 256)	33026	prune_low_magnitude_fire8_squeeze
prune_low_magnitude_fire8_expan	(None, 21, 21, 256)	295170	prune_low_magnitude_fire8_squeeze
prune_low_magnitude_concatenate	(None, 21, 21, 512)	1	prune_low_magnitude_fire8_expand1 prune_low_magnitude_fire8_expand2
prune_low_magnitude_fire9_dropo	(None, 21, 21, 512)	1	prune_low_magnitude_concatenate_3
prune_low_magnitude_conv10 (Pru	(None, 21, 21, 6)	6152	prune_low_magnitude_fire9_dropout
prune_low_magnitude_global_aver	(None, 6)	1	prune_low_magnitude_conv10[0][0]
prune_low_magnitude_softmax (Pr	(None, 6)	1	prune_low_magnitude_global_averag
Total params: 716,523			
Trainable params: 359,014			
Non-trainable params: 357,509			

Figure 4.17: Pruning Model Architecture

4.11 Quantization

“Quantization is a conversion technique that can reduce model size while also improving CPU and hardware accelerator latency, with little degradation in model accuracy” [6]. In Section 4.9, we performed Quantization aware training to reduce the effect of quantization on the model accuracy. TensorFlow lite converts the entire model into a flat buffer [21]. A computer uses a 32-bit floating-point to represent a real number for most applications; the innovative concept of quantization is to convert these 32-bit floating-point values to 8-bit integers, with a slight reduction of accuracy. This gives a distinctive result in the model size of approximately four times compression [24].

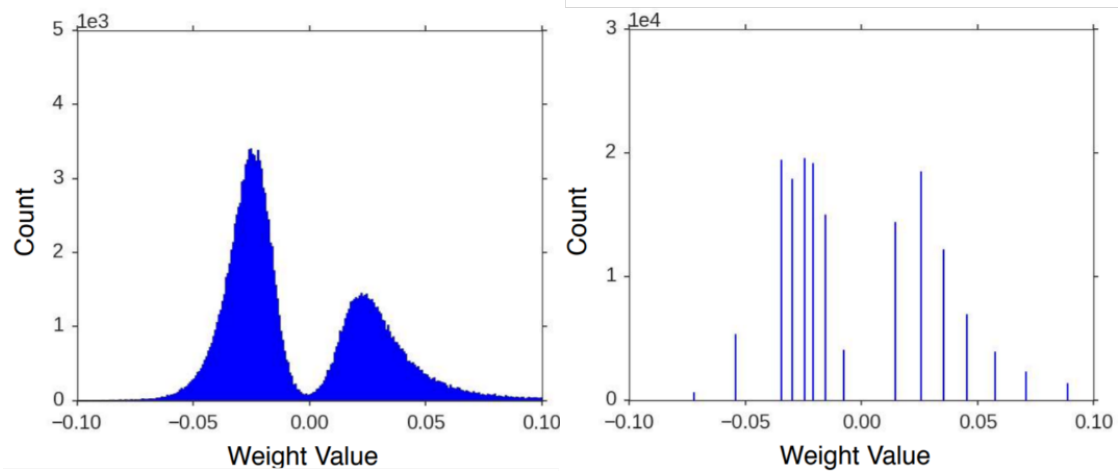


Figure 4.18: Quantization AlexNet [41]

Figure 2.6 is an example of AlexNet, with the original histogram on the left. Quantization discretizes to record important values while round-off the rest. Models are trained using very tiny gradient updates, for which we do need high precision [41]. Quantization distinctly reduces the size of models making it compatible for edge

devices. It's implemented by calling optimization and TensorFlow to TensorFlow lite conversion functions that implement this algorithm.

4.12 Conversion to C array

Converting from TensorFlow lite to C-array is essential, and the final step because micro-controllers can read the model file only in a C-array. Machine learning on Microcontroller is the forefront of innovation; therefore TensorFlow lite for micro-controllers has a limited subset of supported operations. The model architecture was carefully scrutinized so that all operation implemented is supported on the microcontroller. In Linux xxd -i is used as it creates a hex dump of the given file and converts the binary data in the file to ASCII value; the output is in C include file style [8].

Chapter 5

Results

5.1 Logging Data

This work was to design a system that can generate compressed models that can perform sound classification for edge devices. There is always a trade-off between accuracy and model size. The goal of this project was to achieve a model small enough for edge devices yet has decent accuracy. Edge devices have huge constraints on computational power; therefore, we only used Standard Scoring Normalization, a non-vigorous pre-processing method. Logging file size and accuracy was the most critical data required for this thesis; hence we wrote a code that does it dynamically for us. In the directory model, the system generates a file ShrinkLog.txt. This file logs all the parameters and hyper-parameters fed by the user along with a timestamp. It also logs the model accuracy along with the size. Figure 5.1 reviles the contents of the ShrinkLog.txt file.

For a graphical representation of various parameters of model training, Weights and Biases drew our attention, it's a pioneering webpage that can interact with your

system during training; by using wand python package. The below parameters were logged by "Weights and Biases"[10].

- Accuracy
- Loss
- Epochs
- Learning Rate
- Validation Accuracy
- Validation Loss
- GPU Memory Usage

5.2 Parameters and Duration for Testing

We trained the model for 13hrs with the following parameters mentioned below, and achieved great results. We also discovered that smaller window size also affects the accuracy as some of the windows may only contain noise signals after preprocessing.

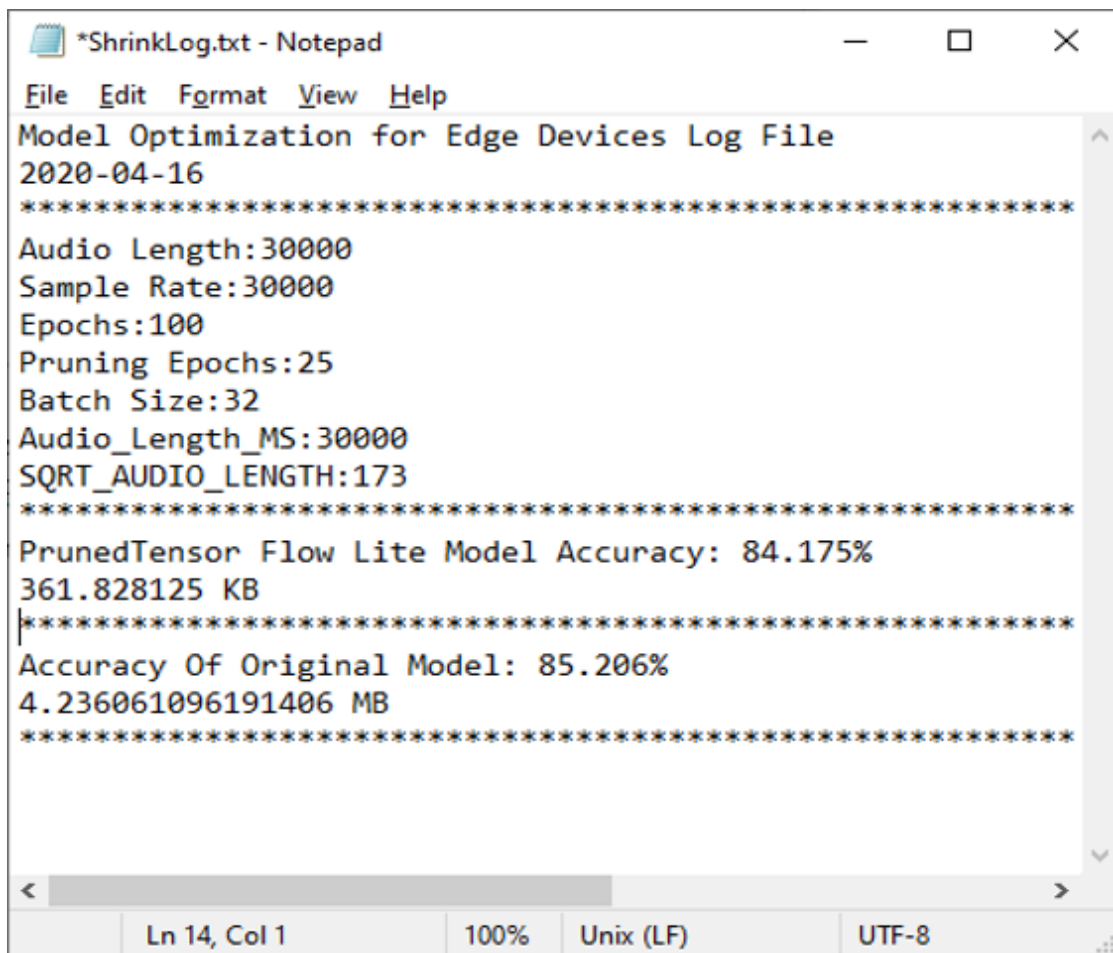
- SAMPLE RATE = 30000
- AUDIO LENGTH MS = 1.0 (1 Second)
- AUDIO LENGTH = (SAMPLE RATE \times (AUDIO LENGTH MS))
- SQRT AUDIO LENGTH = (math.sqrt(AUDIO LENGTH))
- Channel = 1 (Mono)

- Epochs = 100
- Epochs prune = 25
- Batch size = 32
- Verbose = 1
- Number of classes = 6

5.3 Output

We achieved 84.79% accuracy with a model size of 361kB, small enough for most microcontrollers. The original model for edge devices (microprocessors) developed by authors of the paper Cyber-Physical Analytics [21], was impressive; they achieved 89% accuracy on a model size of 1.8Mb. The focus of this thesis was based on reducing the model's size by losing minimum accuracy.

We have achieved 80.37% reduction in size by losing only 4.21% on accuracy on the same data set. Figure 5.1 shows the log generated after training.



```
*ShrinkLog.txt - Notepad
File Edit Format View Help
Model Optimization for Edge Devices Log File
2020-04-16
*****
Audio Length:30000
Sample Rate:30000
Epochs:100
Pruning Epochs:25
Batch Size:32
Audio_Length_MS:30000
SQRT_AUDIO_LENGTH:173
*****
PrunedTensor Flow Lite Model Accuracy: 84.175%
361.828125 KB
*****
Accuracy Of Original Model: 85.206%
4.236061096191406 MB
*****
Ln 14, Col 1    100%    Unix (LF)    UTF-8
```

Figure 5.1: ShrinkLog Text File

5.4 Confusion Matrix

Original

True	KT	405	5	3	7	5	2
	CO	9	218	2	2	11	50
	FS	2	7	80	2	5	1
	KJ	12	3	0	192	9	8
	KN	19	16	4	9	248	13
	LA	10	57	3	4	7	510
		KT	CO	FS	KJ	KN	LA
		Predicted					

Figure 5.2: Original Model Accuracy

Tf-lite Non-Pruned

True	KT	403	6	2	7	6	3
	CO	9	217	2	2	13	49
	FS	3	7	80	2	4	1
	KJ	11	6	0	194	7	6
	KN	18	19	4	9	241	18
	LA	11	55	3	3	9	510
		KT	CO	FS	KJ	KN	LA
		Predicted					

Figure 5.3: TensorFlow Lite Model Accuracy

5.4.1 Results in Confusion Matrix

Confusion matrix can be used to calculate the performance of a machine learning model [36]. There are several parameters that are derived from a confusion matrix.

- **Accuracy:** Number of all correct predictions divided by the total number of the data set [36].

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

- **Precision:** Out of all the positive classes we have predicted correctly, how many are actually positive [36].

$$Precision = \frac{(TP)}{(TP + FP)}$$

- **Recall:** Out of all the positive classes, how much we predicted correctly [36].

$$Recall = \frac{(TP)}{(TP + FN)}$$

- **Overall Accuracy:** Complete model accuracy [36].

$$Overall Accuracy = \frac{(Correctly\ classified\ Values)}{(Total\ number\ of\ values)} \times 100$$

Figure 5.4 is an example of how True positive, True negative, False positive, and False negative, is calculated for the example Keyboard Typing.

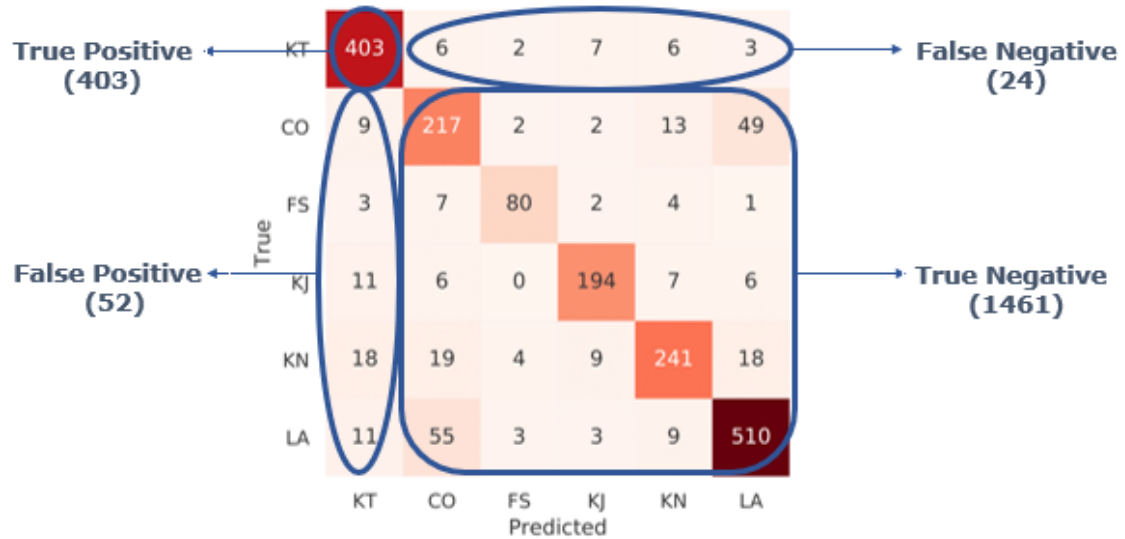


Figure 5.4: Confusion Matrix Example Keyboard Typing

Table 5.1 presents accuracy, precision, recall and F1 score for all labels. The overall model accuracy was also calculated and we achieved 84.79% accuracy.

Table 5.1: Confusion Matrix Calculation Table

Label	Accuracy in %	Precision	Recall	F1-Score
Knocking	94.48	0.86	0.78	0.82
Laughing	91.8	0.87	0.86	0.87
Typing	96.08	0.89	0.94	0.91
Coughing	91.34	0.70	0.74	0.72
Keys Jangling	97.27	0.89	0.87	0.88
Snap	98.56	0.88	0.82	0.85

Overall Model Accuracy = 84.79%

5.5 Plots Weights and Biases

5.5.1 GPU Usage

Figure 5.5 shows that the GPU was processing and training the model for approximately 13Hrs to generate the results. There is a spike at the 11th hour, where the system transitioned from training to pruning. It was concluded that pruning requires less computational resources compared to training a model.

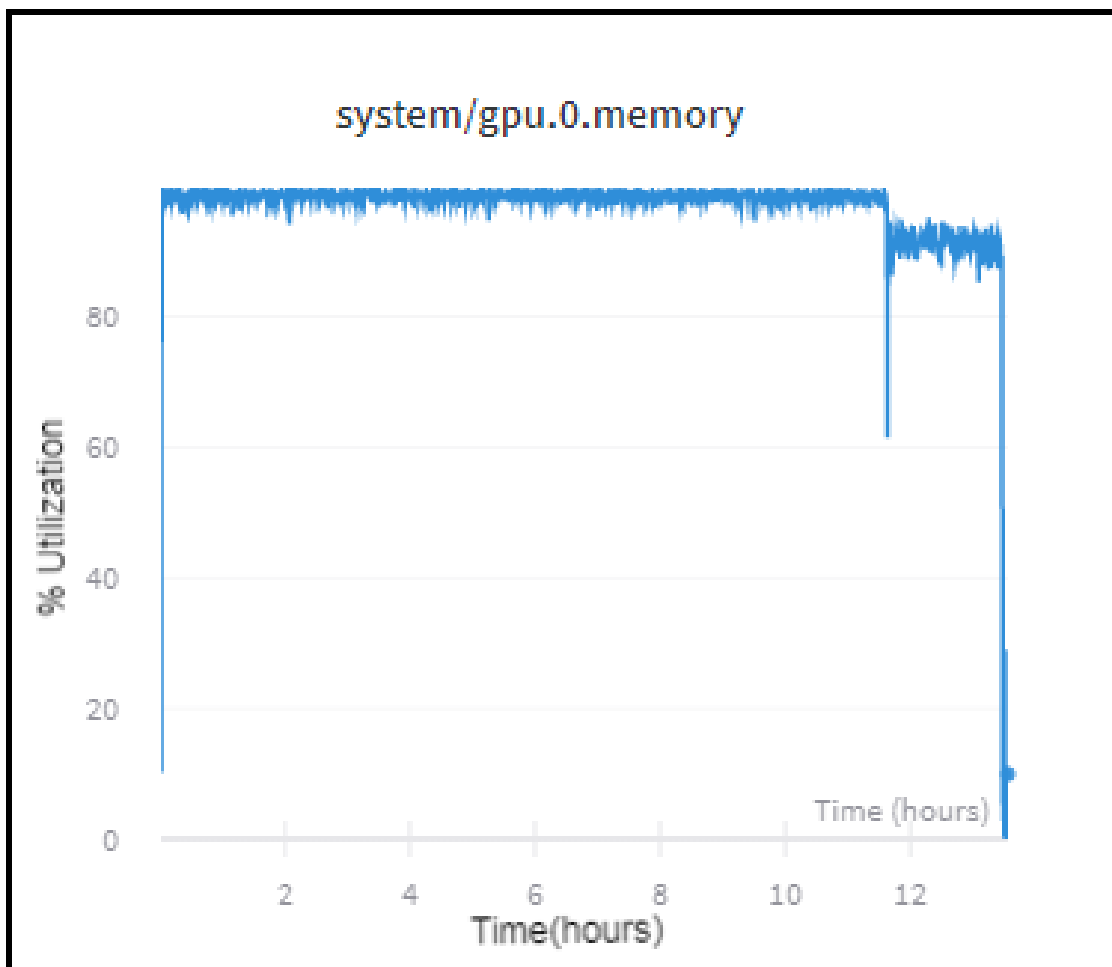


Figure 5.5: GPU Usage

5.5.2 Epochs

Epochs termed as the number of passes through the entire data set. We trained the model with a total of 125 Epochs, 100 for training, and 25 for pruning; it's represented in Figure5.6.

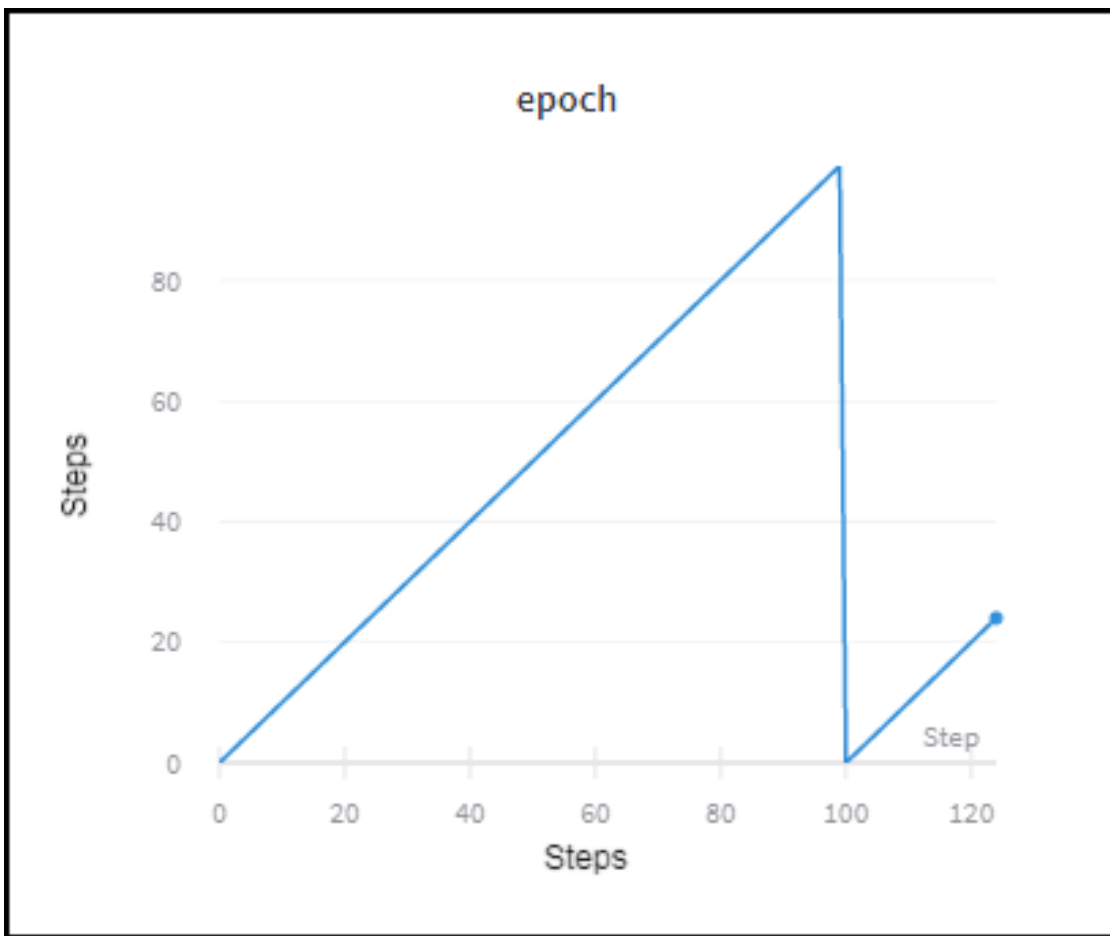


Figure 5.6: Epochs

5.5.3 Loss, Accuracy and Learning Rate

Figure 5.7 and Figure 5.8 represent validation accuracy and validation loss of the model. We integrated training with learning rate and reduced the learning rate gradually during the training period, its represented in the graph shown in Figure 5.9.

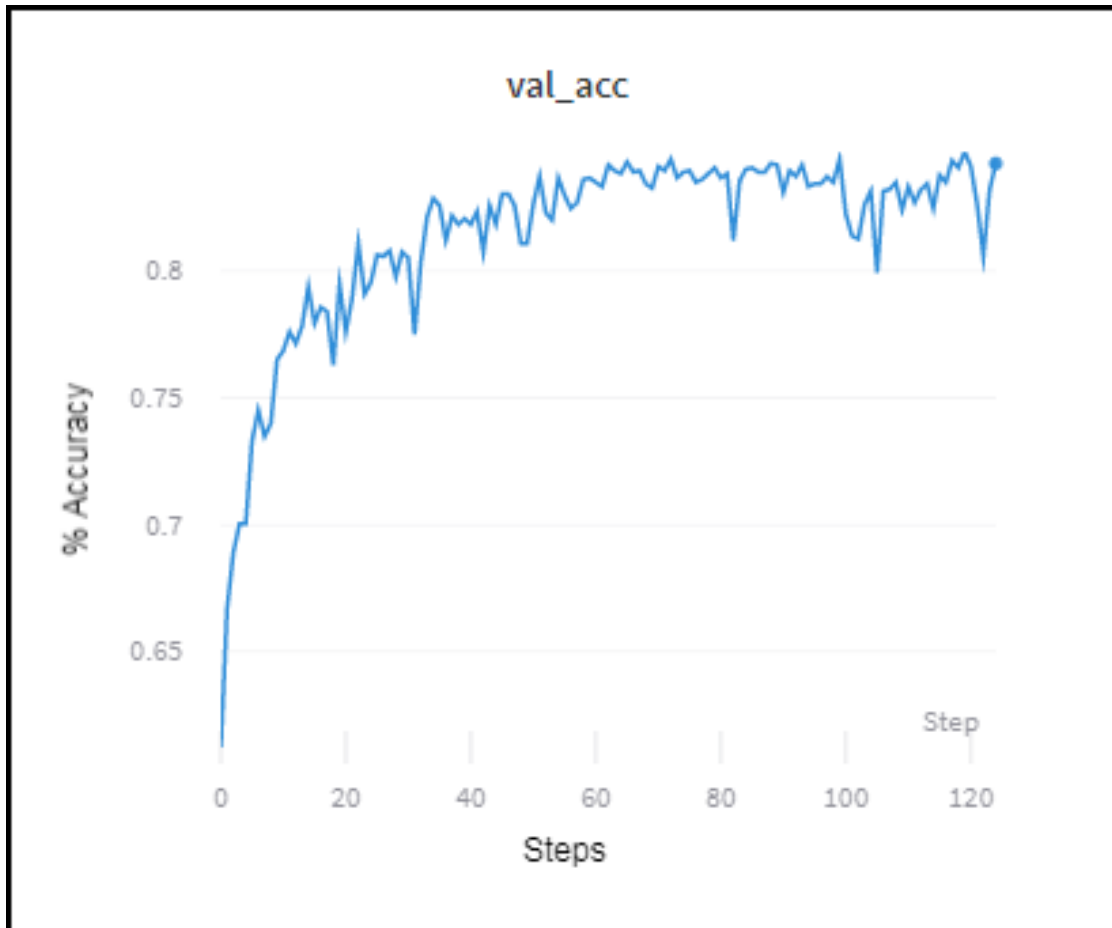


Figure 5.7: Validation Accuracy

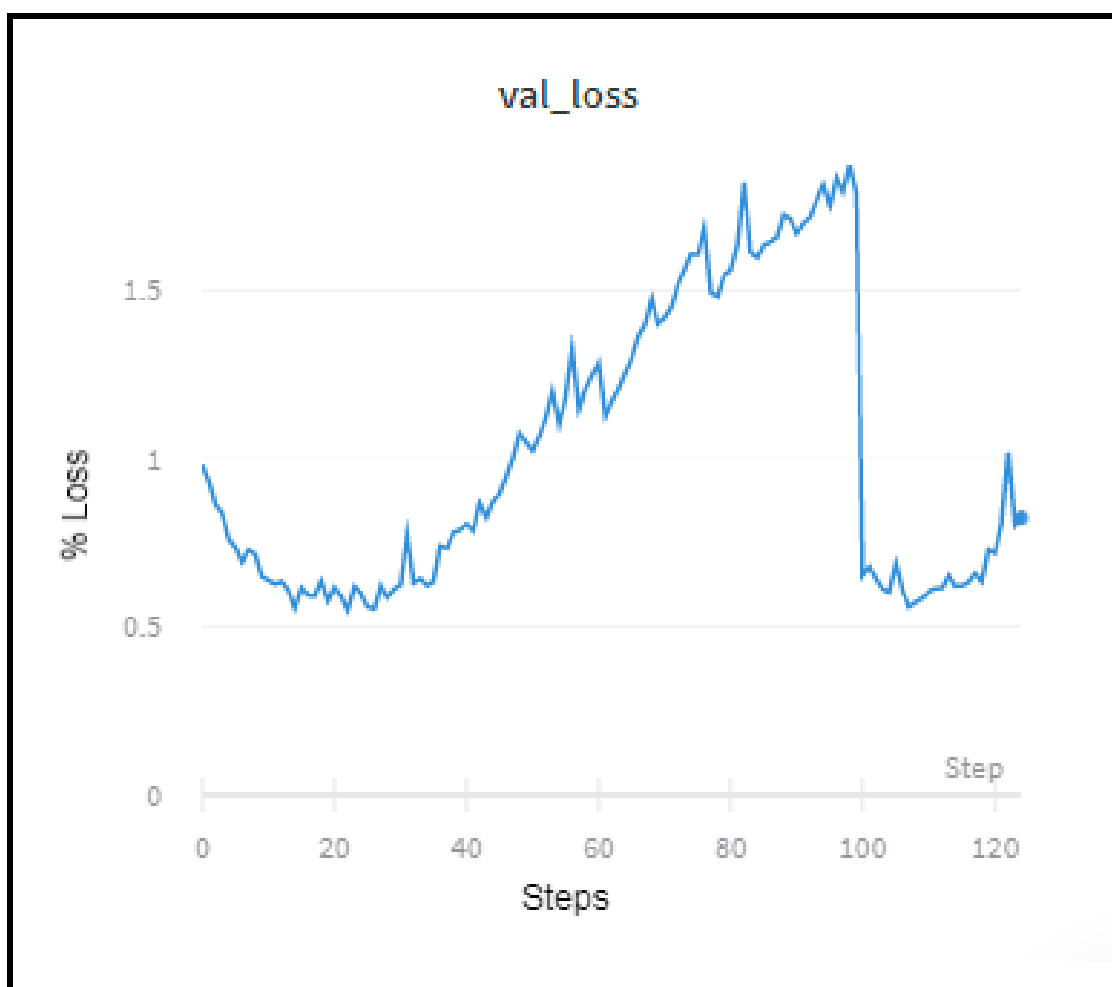


Figure 5.8: Validation Loss

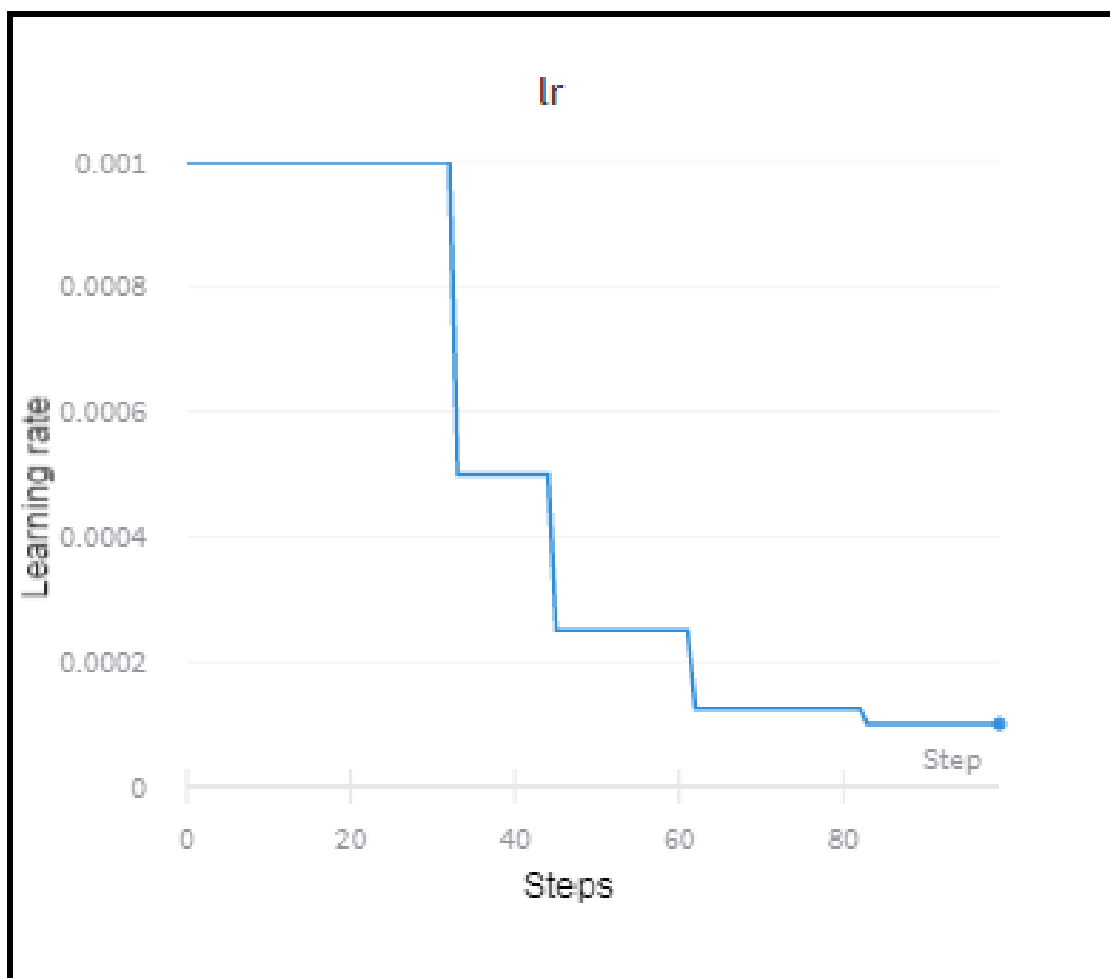


Figure 5.9: Learning Rate

5.5.4 Model Comparison

Figure 5.10 shows the comparison between the different models in terms of their accuracy and size. The architecture we designed on the basis of SqueezeNet strategy proved to be the most space-optimized model among the three. The accuracy of M11 was better, but it was considerably larger.

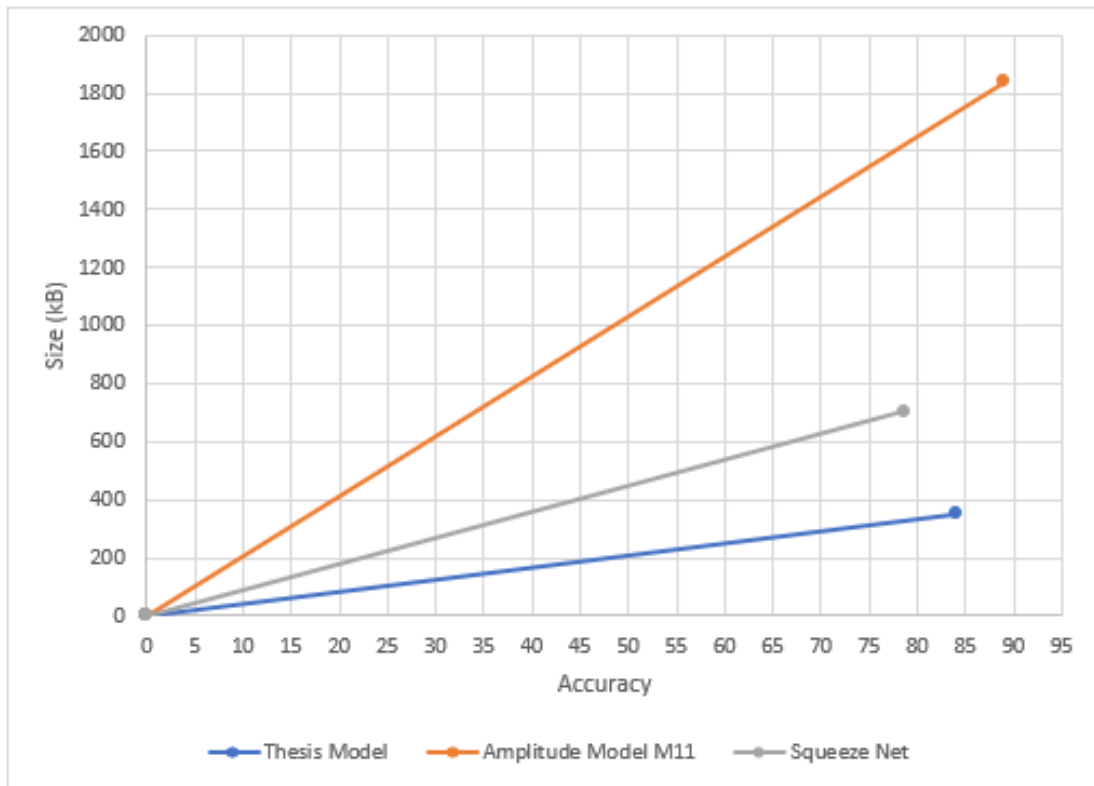


Figure 5.10: Model Comparison

Chapter 6

Conclusion

In this thesis, we conducted a literature review that helped us study the insights of machine learning and speech synthesis. We also gained knowledge about types of hardware used on edge. Based on this finding, we proposed an approach to combine office sound classification, which used machine learning with edge devices; the idea was to incorporate a model small and light enough for microcontrollers at the edge of the network. It would open a new paradigm and possibilities toward edge computing. We achieved high accuracy for a tiny model size. We could now process data in real-time over the edge, and eliminate any kind of cyber-attacks on sensitive data.

6.0.1 Future Work

Dr. Andrew Ng says," Artificial Intelligence is the new electricity.[37]" The next leap is to generate models that can perform natural language processing over the edge. We also can incorporate transfer learning to gain accuracy from a well-trained model. We could tap into using reinforcement learning to make our models learn from their mistakes to get better accuracy.

Bibliography

- [1] 12. *Virtual Environments and Packages - Python 3.8.2 documentation.*
- [2] 1971: Microprocessor integrates cpu function onto a single chip.
- [3] Gartner 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016, December.
- [4] Magnitude-based weight pruning with keras.
- [5] Microprocessor systems. <http://www.key2study.com/microsys.htm>.
- [6] Post-training quantization : Tensorflow lite.
- [7] Quantization aware training with tensorflow model optimization toolkit - performance with accuracy.
- [8] Tensorflow lite for microcontrollers. <https://www.tensorflow.org/lite/microcontrollers>.
- [9] Wave file format. <https://web.archive.org/web/20080113195252>.
- [10] Weights biases. <https://www.wandb.com/>.
- [11] What is edge computing? advantages of edge computing. <https://www.alibabacloud.com/knowledge/what-is-edge-computing>, September 2012.
- [12] Audrey: The first speech recognition system, Oct 2014.

- [13] Quantization. <https://en.wikipedia.org/wiki/Quantization>, Mar 2020.
- [14] Lusine Abrahamyan. Guide on quantizing and converting model to tensorflow lite, Jan 2020.
- [15] Tanweer Alam. A reliable communication framework and its use in internet of things (iot). 3, 05 2018.
- [16] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.
- [17] Marina Brady. OMF: Edge computing changes everything. <https://www.ibm.com/blogs/industries/rob-high-edge-computing/>, December 2019.
- [18] David Reinsel Carrie MacGillivray. Idc. June 2019. Available at <https://www.idc.com/getdoc.jsp?containerId=prUS45213219>.
- [19] Oliver Child. Menace: the machine educable noughts and crosses engine, Oct 2016.
- [20] J. C. Cluley. *Interfacing to microprocessors*. MacMillan Education, 1986.
- [21] David Elliott, Evan Martino, Carlos E Otero, Anthony Smith, Adrian Peter, Benjamin Luchterhand, Eric Lam, and Steven Leung. Cyber-physical analytics: Environmental sound classification at the edge.
- [22] Eduardo Fonseca, Manoj Plakal, Frederic Font, Daniel P. W. Ellis, Xavier Favory, Jordi Pons, and Xavier Serra. General-purpose tagging of freesound audio with audioset labels: Task description, dataset, and baseline, 2018.
- [23] Yves Geerts, Michiel Steyaert, and Willy Sansen. *Design of Multi-Bit Delta-Sigma A/D Converters*. Springer US, 2003.
- [24] GitHub. tensorflow. <https://github.com/tensorflow/tensorflow/tree/r1.13/tensorflow/contrib/quantize>, Dec 2018.

- [25] Guha and Ghosh. The impact of data quality in the machine learning era, Jun 2018.
- [26] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.
- [27] Qiangui Huang, Kevin Zhou, Suyu You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks, 2018.
- [28] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size, 2016.
- [29] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.
- [30] Aimee Kalnoskas, Henry.G, Gabriel Urach, and Gabriel Urach. A beginner's guide to microcontrollers.
- [31] Fotis Konstantinidis and Fotis Konstantinidis. Why and how to run machine learning algorithms on edge devices, Feb 2020.
- [32] Erwin Kreyszig. *Advanced engineering mathematics*. Wiley, 2006.
- [33] Kyle Media LLC. Macintosh IIci Specs.
- [34] Mark A Mandel. A commercial large-vocabulary discrete speech recognition system: Dragondictate. *Language and speech*, 35(1-2):237–246, 1992.
- [35] Thomas M. Mitchell. *Machine learning*. McGraw-Hill, 1997.
- [36] Sarang Narkhede. Understanding confusion matrix, Aug 2019.
- [37] Andrew NG. Artificial intelligence: the new electricity.
- [38] DikshaTewariCheck out this Author's contributed articles. and DikshaTewari. Introduction of microprocessor, Aug 2019.

- [39] Karol J. Piczak. ESC: Dataset for Environmental Sound Classification. In *Proceedings of the 23rd Annual ACM Conference on Multimedia*, pages 1015–1018. ACM Press.
- [40] Ravi. Basics of microcontrollers: History, structure, applications, Dec 2017.
- [41] Manas Sahni. 8-bit quantization and tensorflow lite: Speeding up mobile inference with low precision, Feb 2020.
- [42] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.
- [43] Ranjeet Singh. Pruning deep neural networks, Aug 2019.
- [44] Robert Strong. Casper: A speech interface for the macintosh. In *Third European Conference on Speech Communication and Technology*, 1993.
- [45] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017.