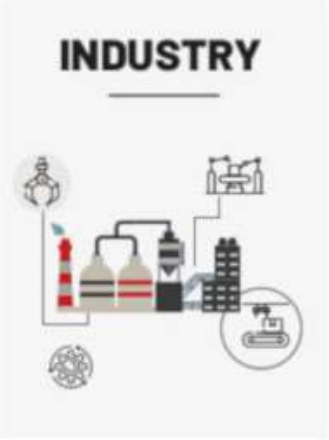
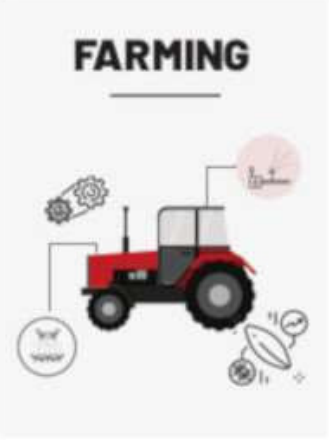
Introduction to

TinyML





Introduction

to

TinyML

By

AI Technology & Systems

ISBN (paperback): 9798839399129
ISBN (hardcover)): 9798840151297
First Edition: July 2022

Contents

1: Introduction 1

What are AI and ML 2

ML Disruption 3

Scalability and ML 4

What is TinyML? 5

TinyML Applications 5

TinyML Benefits 6

Summary 7

2: Understanding TinyML 9

Need for TinyML 11

Benefits of TinyML 12

Hardware for TinyML applications 14

List of TinyML Verticals 15

TinyML with Examples 16

Summary 17

3: American Sign Language 19

Introduction 19

What is ASL? 21

Where is ASL used? 21

Why use TinyML for ASL 22

Inputs And Outputs 23

ASL Application Steps 25

No-Code Deep Learning Model 25

Downloading device SDK 26
Flashing Arduino 27
The dataset 28
Model Training and Testing 29
Compile & Download 30
Low Code Deep Learning Model 30
The Dataset 31
Model Training 31
Testing & Validation 32
Saving the Model 32
Model Inference 33
Compilation 34
Model Analysis & OS Testing 34
Downloading and Deploying on Microcontroller 36
Summary 37
4: Audio Wake-Word Detection 39
Introduction 39
Algorithms For Wake Word Detections 40
Where Are Wake Words Used 40
TinyML and Wake Word Detection 41
Inputs of the Wake Word Model 42
No-Code Deep Learning Model 43
CAInvas Playground 43
The AuWW 44

The Dataset 44
Model Training 45
Testing 45
Compile and Download 46
Low Code Deep Learning Model 46
The Dataset 47
Model Training 48
Testing and Validation 49
Saving the Model 50
Compilation 50
Deploying onto Microcontroller 51
Summary 51
5: Visual Wake Word Detection 53
Introduction 53
What is Visual AI/ Computer Vision 54
Applications and the Scale of Computer Vision 54
Computer Vision and TinyML 55
Visual Wake Word vs. Audio Wake Word 56
Defining Inputs and Outputs of the VWW Application 57
A Bit About Relevant Camera Measures 57
Features of the Arducam 58
No-Code Deep Learning Model: 60
CAInvas Playground 60
The Dataset and Training 61

Testing 62
Compile and Download 62
Low Code Deep Learning Model 63
The Dataset 64
Data Augmentation 64
Model Training 65
Testing & Validation 66
Saving the Model 67
Compilation 67
OS Testing & Debugging 67
Downloading on Microcontroller 68
Summary 69
6: Predictive Maintenance 71
Introduction 71
Predictive Maintenance Solution Interface 72
Sensors and Analyses 74
Industry 4.0 Applications 75
TinyML and Predictive Maintenance 75
Sensors and Interface 77
Sensor Data and Equipment State 81
No-Code Deep Learning App 82
Data Collection 83
Signal Processing 84
Model Training 84

Application Deployment 84
on Device Signals 84
Dashboard 85
Low Code Deep Learning Model 85
Model Training 86
Testing & Validation 89
Saving the Model 90
Compilation 90
Model Inference & Integration 91
Deployment 91
AITS Differentiation 92
Summary 92
7: AutoML for TinyML 95
What is AutoML 95
Why Use AutoML? 96
Automation: 96
Consistency: 96
Scalability: 96
AI Empowerment 97
AutoML Features 97
AutoML frameworks 98
AutoML empowers TinyML 99
AITS AutoML - Stadium 100
Creating TinyML Applications Using Stadium 100

Model Testing....101
Steps for Creating an AI Vision App on Stadium....101
Step 1 - CAInvas Stadium....102
Step 2 - Dataset Collection....103
Step 3 - Testing....104
Step 4 - Compile & Download....105
Summary....105
8: DeepSea, A TinyML Compiler....107
What is a TinyML Compiler?....107
Types of Compilers....108
Interpreter....108
Ahead of Time Compiler....108
Benefits of Ahead of Time Compiler....108
Why Compiler?....109
The deepSea....110
deepSea vs TF Lite Micro....111
The deepSea Features....111
deepSea Technology....113
deepSea Tensors....113
Declaring and Using Tensors....114
deepSea Operators....115
Pythonic Operators....115
Neural Networks (NN) Operators in DeepSea....116
Putting It All Together in a Neural Network....117

Running a neural network on a board.....118
Manually Implementing ML models with DeepSea.....118
Applications of deepSea 120
deepSea Scripting Using Python 120
Implementing the KNN algorithm using DeepSea 120
Implementing the K-means Clustering Algorithm with DeepSea 126
deepSea for Low Level Language (C++) 132
Integrating deepSea with C++ 132
Microcontrollers Support in deepSea.....139
Appendix 1: List of TinyML Verticals 140
Appendix 2: List of TinyML Applications 143
Appendix 3: List of URLs 169

Preface

Artificial intelligence is a novel field of computer science. Despite being coined in the 1950s, the field saw numerous ups and downs until normal development started in the 2000s. The field has broken many barriers for computers once thought impossible. Today, you have smartphones that only open up when they see you. We have cars that no longer depend on drivers. Moreover, scientists now regularly rely on AI to unravel the secrets of our universe. Machine learning and Deep learning, the sub-fields in AI, continue to bring changes in our lives.

Machine learning and Deep learning have found applications in many frontiers and continue to emerge on various other frontiers. One such frontier is the field of Tiny Machine Learning (also called TinyML). The field intends to bring the power of machine learning to small devices. The field will further enable use cases in resource-limited scenarios. The potential for the field is huge as new technology continues to roll out, which makes devices and sensors smaller and more powerful.

This book is an effort by our company AI Technology & Systems to demystify the TinyML technology including market, applications, algorithms, tools and technology. We intended to dive deeper into the technology beyond common application and keep it light for the readers with varying background. The idea of developing a common solution of TinyML for users belonging to all domains led to the creation of the CAInvas platform. This book talks about the CAInvas platform, explaining how to use it to implement various AI applications.

We provide no-code solutions for all the applications discussed in this book. A low-cod variant of the applications has also been provided for experienced users (programmers and data scientists) with complete access to pre-written code of the applications. We provide guidance on using these pre-written models and tweak them according to their needs.

Apart from diving into the applications, we also have two special chapters in our book. One explains Automated machine learning on CAInvas. The other chapter dives into the DeepSea, an ahead-of-its-time, hardware agnostic Deep Neural Network library, compiler and framework. We discuss the various features of the DeepSea compiler and how it is the fastest and the most optimized solution available for TinyML.

We would like to thank all the members of the AITS developers' team, data science team, and product engineering team who put their efforts into providing relevant information to write this book. Time went into developing the models for applications discussed in the book. Moreover, models were made using the same low-code and no-code solutions of the applications shared in the book, which was an arduous task. Nonetheless, we overcame all the adversities in our way with a shared belief and passion for serving the user community with our novel platform CAInvas. It was this shared belief which led to the completion of this book.

Subject Matter

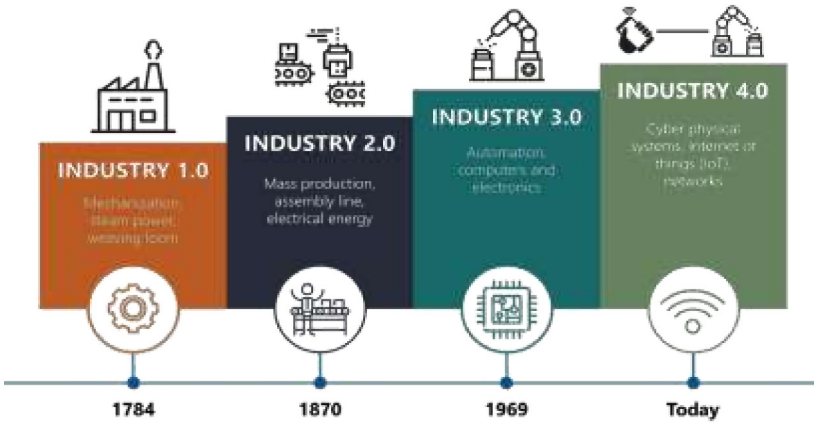
First chapter gently introduces foundational terms including AI, ML and TinyML along with their scalability, benefits and applications. Second chapter dives deeper into TinyML technology with examples. The next 4 chapters introduce 4 popular applications of the TinyML including American Sign Language, an audio application, a vision application and a maintenance application. The 4 application chapters introduce no-code and low code technology to develop the application. Chapter 7 is dedicated to the automated machine learning technology that powers no-code platform. Chapter 8 is devoted to in-depth discussion of tinyML compiler deepSea that rivals with Google's TensorFlow Lite Micro.

Conventions and Terms

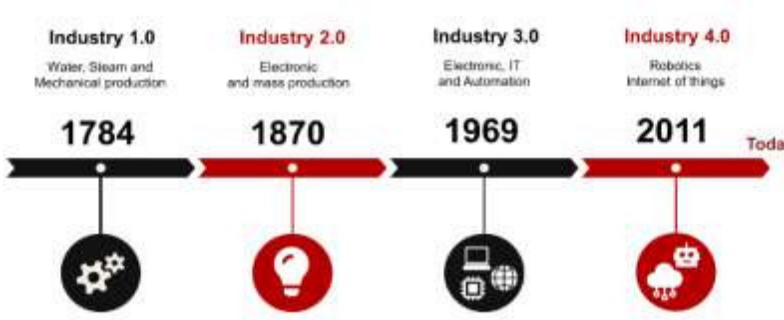
1. ***deepC*** is a vendor independent deep learning library, compiler and inference framework microcontrollers with an open-source repository: https://github.com/ai-techsystems/deepC
2. ***deepSea*** is a proprietary deep learning library, compiler and inference framework for TinyML devices at https://www.ai-tech.systems/deepsea/
3. ***deepCC*** is a ML model and hardware agnostic deep learning compiler for TinyML devices.
4. ***dnnc*** is python interface for machine leaning library and inference framework.
5. ***cAInvas*** is no-code and low-code platform for TinyML application development.
6. **Playground** and **Stadium** are sensor agnostic no-code application development platforms.

1: Introduction

Today, we live in a highly interconnected and automated world. The basis of this automation is the use of computers and their ever-increasing integration in our lives. Noticeable changes and disruptions are caused by advances in technologies in every walk of life, both in the industrial and domestic fields of life. Disruptive technologies have revolutionized the industries since the 17th century. In particular - Steam power, Electricity, Semiconductor and IoT technologies have marked the threshold of four industrial revolutions in that order.



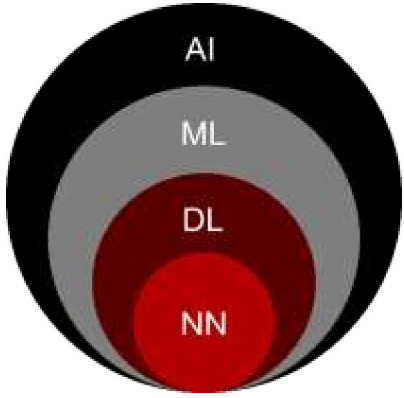
The next major frontier for the tech industry is the implementation and development of technologies that aid the fourth industrial revolution. Self-driving cars, smart contact lenses, Internet of things, and advanced healthcare are a few instances of the fourth industrial revolution. Artificial



Intelligence (AI) technologies are integral to almost all the instances above. Furthermore, the sub-field of AI, that is, deep learning, is the cornerstone of AI technologies today. Let us understand how these AI technologies are coming together to solve problems and automate tasks.

What are AI and ML

As defined by the renowned Institute of Electrical and Electronics Engineers (IEEE) (https://globalpolicy.ieee.org/wp-content/uploads/2019/06/IEEE18029.pdf), AI is "the theory and development of computer systems that are able to perform tasks. that normally require human intelligence, such as visual perception, speech recognition, learning, decision-making, and natural language processing".



Artificial Intelligence (AI) is one of the most amazing technologies in the modern history of computing. The ability of a machine to learn and adapt like a human being has led to revolutionary advances and developments in various industries such as agriculture, healthcare, transportation, and retail. AI is indeed the future of automation.

To explain machine learning (ML), we will use some basic computer terminologies. In general, ML is the use of computer algorithms that can learn and adapt from processing digital data. Several computer algorithms are used in the field of machine learning. The most famous ones include different types of regressions, K-means, Naive Bayes algorithm, etc. Though a sub-field of AI, ML itself is a large field. From the perspective of algorithms and techniques, ML itself has a subfield called Deep Learning (DL) that relies heavily on neural networks to process data and generate results.

Disruption

The use of AI and ML was driven by the limitation of traditional computer programming in solving problems that involved ambiguous parameters, extreme levels of processing, and situations with ever-changing conditions. For example, using an ML-based solution to classify a spam email is much more efficient than traditional coding. One may be tempted to write a line-by-line traditional code. However, it would not be easy to

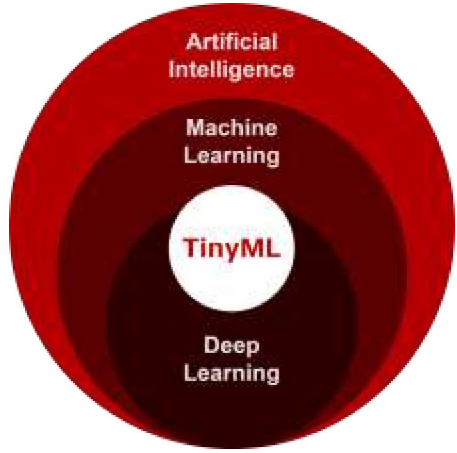
program such a code to identify a spam email from various sources, each having their own style of a spam email.

Scalability and ML

As evident from the problems that deep learning solves, it is usually a resource-intensive technology. Efficiently running ML algorithms requires large training data sets and high computer processing powers, usually available in high end computers with accelerators (such as GPUs and TPUs).. Therefore, there exists a limitation in scaling ML technologies. To implement the dedicated spam detection feature described previously, tech firms have high-powered computing machines working through a cloud infrastructure.

There has been an increasing potential and demand to avail ML solutions in small and resource constraint systems and edge devices. Deploying ML solutions requires high computing resources that are generally not available on the small edge devices. Edge devices could resort to using the cloud for compute resources. There are however scenarios where using the cloud is not an option either due to security reasons or due to lack of internet connectivity. Furthermore, the machine learning models sizes are oftentimes too large to load in the edge devices memory. This is where TinyML comes forth as a solution.

What is TinyML?



TinyML is a fast-growing sub-field of machine learning including algorithms, techniques and applications for resource constrained hardware capable of running small software applications. It sits at the intersection of machine learning and deep learning algorithms and enables a variety of always-on use-cases and targets battery-operated low power devices. As obvious from the definition, this technology is intended for small microcontroller units and embedded systems. Larger ML models are optimized and reduced to a small size to fit in the devices' small memory, The TinyML ecosystem is fueled by:

1. Emerging commercial applications on AIOT devices
2. Significant progress in optimizing algorithms, networks, and models down to 100 kB and below.
3. Low-power applications in vision and audio are becoming mainstream and commercially available.

TinyML Applications

Industries are using TinyML for many purposes like:

1. Predictive Maintenance
2. Smart Farming
3. Retail

4. Data Privacy and Security
5. Aero Defense

The list of the possible applications of TinyML is endless. Using TinyML, the industries can build more profitable and competitive products. Given the large potential and wide application areas that TinyML has, early adopters have a high chance of creating disruptive and world-class products.

Since resource constrained devices are inexpensive, the capital required for prototyping and scaling is not a barrier for adoption. TinyML devices also enable industries to build products that protect their customers' data privacy. It is because of the in-house processing capabilities of such devices.

Another bonus of using TinyML is the ability to run AI models on-device. The ability to run on-device AI models enables new ideas and facilitates numerous applications.

TinyML Benefits

TinyML is used because of the following advantages:

1. No internet connection required
2. Independent of Cloud
3. Low power consumption
4. Improved Latency
5. Battery-powered standalone solutions
6. Applications for Remote Areas with no Connectivity or Power
7. Stealth mode operation
8. Enhanced privacy
9. No headache of cyber security with on-device operation.
10. Reduction of elimination of bandwidth
11. Economic adoption

These benefits make TinyML solutions perfect for multiple applications in several industries and verticals over traditional connected IoT solutions.

Summary

In this chapter, we learned about artificial intelligence (AI) and its subfields machine learning (ML) and deep learning (DL). We also discussed the reasons and motivations behind these fields and their profound influence on the upcoming fourth industrial revolution. We found that deploying ML-based solutions is not easy. This is because many ML-based solutions rely on high levels of computing power to work properly. Therefore, it has always been a challenge to bring ML technology to smaller devices. TinyML was introduced to address these issues. The main aim of TinyML is to bring ML technology to small microcontroller units (MCUs). These MCUs are quite limited in their compute and memory resources. - Optimizing the ML applications to reduce the memory and power requirement is crucial for their deployment in small MCUs.

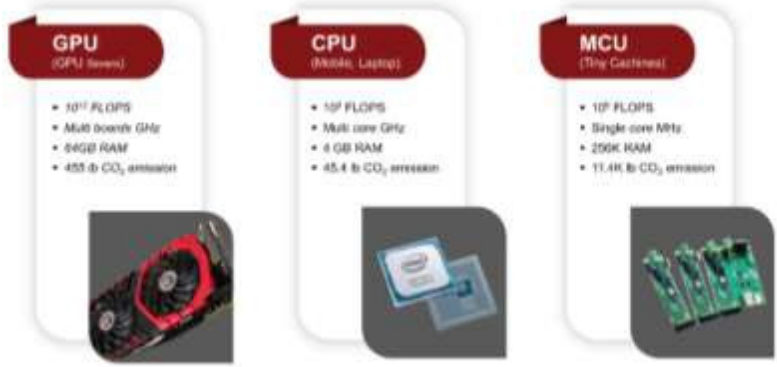
The possible applications of TinyML are numerous. It is currently being used in many industries. Moreover, new TinyML-related solutions continue to emerge. The devices that TinyML targets are not only cloud and internet independent but also power efficient enough to last several years. Such features go a long way in helping ML-based applications work in controlled and resource-constrained environments.

2: Understanding TinyML

Machine learning (ML) is a dynamic field of computer science. It has penetrated almost every digital thing we care about including social media, phones, cars, and even household items. However, ML isn't easy to run in resource constrained environments, since most state-of-the-art deep learning models require significant computer resources and energy consumption.

The demand for high-performance computing resources has limited many ML applications in the cloud, where data center computing is readily available. Using cloud computing risks privacy, security and mission critical nature of the application, forcing them to require unreasonably high bandwidth and high latency.

The issues of privacy and security are difficult to address with live connection to cloud machines and makes the user incline towards avoiding the cloud altogether. However, the compute demand of deep learning applications makes it difficult to host the application on the edge or IoT device with no cloud connection. Picture below shows a quick comparison of compute speed, compute capacity, compute storage and power consumption among mainstream GPUs, CPUs and MCUs.



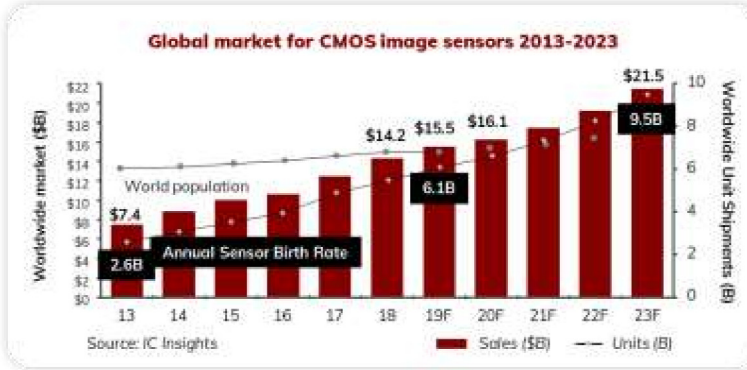
To extend the reach of ML and open a new era of process applications, we need to find ways to facilitate the inference of ML on a smaller hardware machine easy to deploy on the edge with or without cloud and internet. These efforts have led to a field known as Tiny machine learning or TinyML® (a term trademarked by the TinyML Foundation, which has become synonymous with technology).

TinyML is sometimes also described as a field of study of optimizing deep learning models and making them usable on resource constrained devices. It enables low-latency, low power, and low-cost deep learning by combining the power of models with microcontrollers, making it possible to leverage the advantages of these microcontrollers to train these models.

There is a growing momentum demonstrated by technical progress and ecosystem development in these areas. TinyML research serves as a flagship venue for related research at the intersection of deep learning applications, algorithms, software, and hardware in deeply embedded deep learning systems.

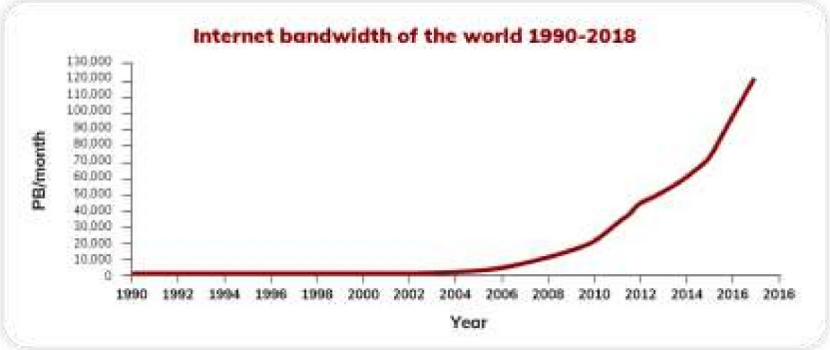
Need for TinyML

One may question the urgency and need for running applications on the edge or disconnected devices. Take image sensors for example - there are over 8 billion image sensors produced in 2021 as shown in the graph below.



Assuming 3 years as the average life of image sensors, we'll have over 20 billion image sensors in the world as of 2022 and it is growing as we speak. These image sensors will require bandwidth of 10^{20} bytes per second. Current bandwidth of the internet stands at approximately 10^{14} bytes per second. Ever increasing internet bandwidth helps but way too short to address the requirement as shown in the graph below.

Combined bandwidth of the world's networks falls short to support



generated pixels by a factor of 10^6 or 12 years. Data must be processed on the EDGE devices. So the vision applications must be hosted on the edge devices or devices disconnected from the internet to not require the data transport.

Similarly, there are other fields like health and biometric sensing that impose critical privacy concerns for the data sent over the internet being breached or stolen. In few cases like audio speech & command detection, deep learning models are already small enough (under 11KB code size) to be hosted in tiny devices.

Benefits of TinyML

TinyML enables the deployment and on-device processing of ML models into resource constrained edge devices. TinyML entails numerous benefits that are discussed below.

Low Latency

Data collected by edge device sensors is processed by an on-device ML model to generate the results. Tiny ML models are optimized to have a

reduced number of parameters and there is no time wasted in transmitting the data to a datacenter server or cloud for processing. This in turn results in a very low latency (i.e., fast turnaround time) from data collection to result generation. TinyML applications can work in real time due to their low latency for mission critical applications.

Privacy

Machine learning consumes data. Privacy has been a major concern against the adoption of machine learning technologies since consumers are oftentimes unwilling to share their data or transport it over the internet due to security concerns. TinyML enables on-device processing of data in real time. In TinyML applications, since data is neither transmitted out of device nor stored anywhere, there are no privacy concerns,

Fingerprints used for biometric authentication is an example of data with privacy value. Fingerprints are typically represented as 129 points. If this fingerprint data is compromised over the internet, it will be an irreparable loss. Passwords can be changed, fingers can't be! That's the reason sensitive biometric data must always be stored and processed on-device.

Low Power

TinyML applications are expected to run on resource constrained devices, process real time data and produce results within data sampling intervals. Driven by all these requirements TinyML models are optimized to run with fewer parameters, use reduced computations and consume low power.

TinyML applications mostly use on-device sensors' data. Majority of the sensor data (other than microphone audio data and camera video data) volumes are small (less than 1K bytes/sec). Consequently, machine learning models to process these data are also light.

Reduced Bandwidth

TinyML applications cater to small, resource constrained devices with zero or very low internet connectivity. On-device sensors capture the data, data is processed on-device and hence there is no data transmission bandwidth involved.

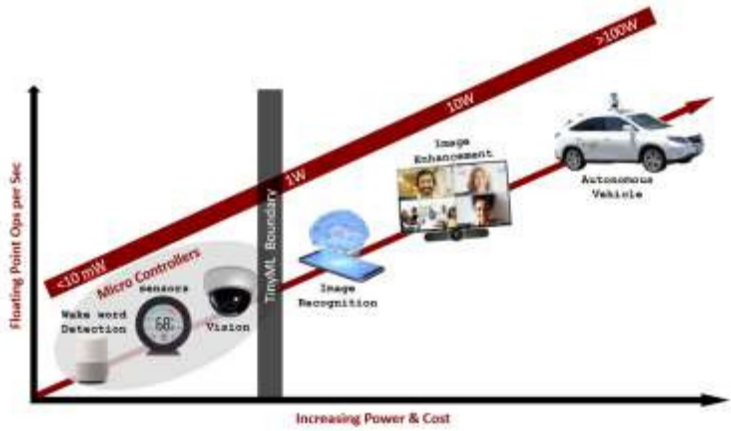
Sometimes TinyML applications use a hybrid cloud processing in order to achieve higher accuracy. For instance, a TinyML application does on-device processing of the video data from an always-on on-device camera sensor. Once on-device processing detects an object of interest it streams the information to cloud servers for more detailed processing.

Hardware for TinyML applications

TinyML is impressive because it aims to work with quite limited hardware capabilities. From a specific perspective, the real purpose of TinyML is to draw ML conclusions with the lowest possible strength.

The low power consumption, equal to one mW consumption, makes it a device that can run on a regular coin cell battery with a reasonable life of months to a year. So when you think of power sources for TinyML, think of button batteries, small Li-Po batteries, and energy storage devices.

From a computer perspective, TinyML does not rely on graphics processing units (GPUs), application-specific integrated circuits (ASICs), and



microprocessors like most ML applications demand. To achieve the one mW target, we are limited to less capable computer hardware such as microcontrollers (MCUs) and digital signal processors (DSPs). These devices are usually based on Cortex-M and can be expected to have no more than a few hundred kB of RAM, a similar flash volume, and a ten MHz clock. Other hardware you might expect from a TinyML device includes sensors (such as a camera or microphone) and possibly some Bluetooth Low Energy (BLE) connections. Next picture shows the compute performance (shown as floating-point operations per second), power (shown as Watts) and cost boundaries for TinyML hardware and applications.

List of TinyML Verticals

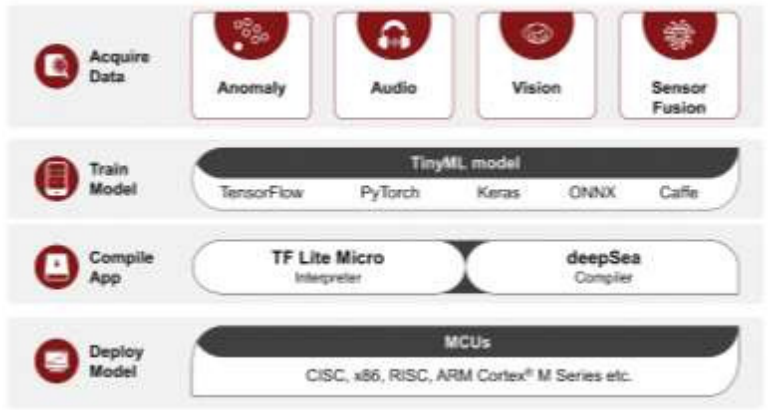
Here is a partial list of TinyML applications maintained on AITS CAInvas:

1. Smart Industrial IoT: https://cAInvas.ai-tech.systems/use-cases/verticals/details/industrial-iot/
2. Smart Aviation: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-aviation/
3. Smart Farming: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-farming/
4. Smart Transportation and Logistics:https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-transportation-and-logistics/
5. Smart Security: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-security/
6. Smart Oil and Gas: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-oil-and-gas/
7. Smart Environment: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-environment/

8. Smart Space: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-space/
9. Smart Home: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-home/
10. Smart City: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-city/
11. Smart Retail: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-retail/
12. Smart Energy: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-energy/
13. Smart Auto: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-auto/
14. Smart Society: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-society/
15. Smart Finance: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-finance/
16. Smart Health: https://cAInvas.ai-tech.systems/use-cases/verticals/details/smart-health/

TinyML with Examples

In subsequent chapters of this book, we will see TinyML in action. We will present you with applicable use-cases that you can try out on your small devices with little or no knowledge of coding or deep learning.



We will discuss design and implementation of end-to-end tinyML applications using AITS' CAInvas platform. The no-code platform lets users collect datasets, train models, compile them into an application for the hardware of the user's choice and download the application. The no-code platform allows users to click and develop the application without writing the code or understanding the technology. Low code features allow for flexibility for beginner and novice programmers to be more creative in modifying an existing solution. In particular, we would look into:

1. American Sign Language
2. Audio wake word
3. Visual wake word
4. Predictive Maintenance

Summary

This chapter discusses TinyML in greater detail. The limited scalability of ML applications and their reliance on cloud computing methods has become a halting factor in developing newer ML applications. Some users of the cloud-based ML application might face privacy and security issues.

Hence, we cannot rely on cloud-based ML technologies in all circumstances. TinyML devices might be constrained in their computing power. Still, it does not stop their ability to bring ML applications to the masses.

All in all, four benefits make TinyML attractive to the latest technology solutions. Since microcontroller units (MCUs) have their own sets of RAMs and CPUs, this guarantees lower latency in making calculations. TinyML targets a power consumption of below one mW. This tiny amount of power consumption enables them to work on small coin batteries for years. Being smaller in size, MCUs come in limited bandwidths (in the MHz ranges) or no bandwidth. This factor becomes especially useful for connectivity contested compute nodes in distributed networks. Finally, the independence from the internet and cloud infrastructure makes TinyML devices the best for situations where privacy is critical.

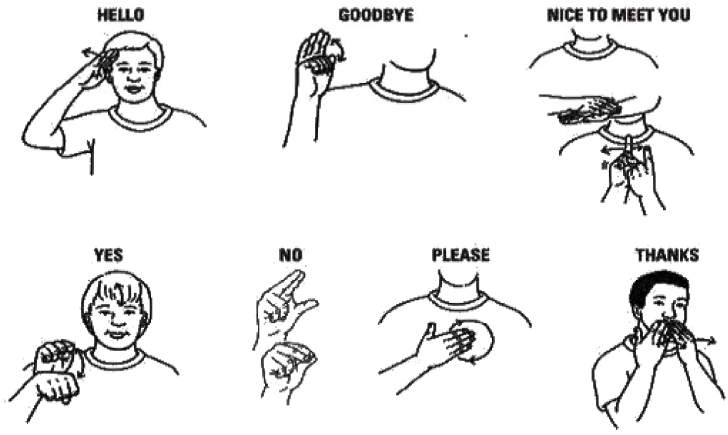
There is some confusion when discussing the precise class of devices used for TinyML. A better understanding of the hardware used in TinyML devices clears this confusion. Resource-constrained TinyML devices have a few hundred KBs of RAM and Cortex-M processors with clock speeds in MHz. In other words, all the components of an MCU must not consume more than one milliwatt of power. This distinction excludes handheld devices like smartphones and other smart gadgets.

To better understand how TinyML works, we will examine some applications of TinyML in the subsequent chapters of this book.

3: American Sign Language

Introduction

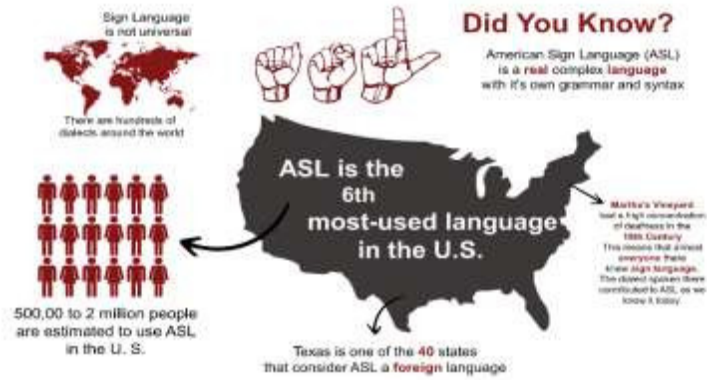
Mute population in the world today has crossed over 70 million. Mute people communicate using sign language consisting of gestures made by hands, arms and face. Students aren't taught sign language as part of a typical curriculum, so the majority of the population is disconnected from the mute population. How wonderful would it be for the vocal language speaking world to connect to sign language speaking one? It'd unlock billions of joyous conversations. ML applications to recognize sign language can facilitate such conversations. In this chapter, we will learn to create an application prototype that will decode and interpret sign language hand gestures. The device we will use will be an Arduino equipped with a gyro sensor, accelerometer, and a speaker. It will allow especially abled people to better communicate with others using hand gestures. Their gestures and movements will be decoded using accelerometer and gyroscope and will be translated/announced using speakers.



We must understand the word 'gesture' to better understand how gesture recognition works. Generally, the word gesture means non-verbal communication aimed at conveying a particular message. In the world of gesture recognition, gestures are defined as any physical movement, large or small, that a motion sensor can interpret. By extension, sign language is a means of communication through hands and arms, used when spoken communication is impossible or not desirable. Such practice is probably older than speech. Sign language may be as coarsely expressed as mere grimaces, shrugs, or pointing; or it may employ a delicately nuanced combination of coded manual signals reinforced by facial expression and perhaps augmented by words spelled out in a manual alphabet.

Computer gesture recognition identifies actions and perceptions. Gesture recognition is also used as an interface to carry out commands. Therefore, the general definition of gesture recognition is the ability of a computer to understand human gestures and perform commands. The feature of computer-based gesture recognition is used in many computer games and other non-touch electronics. Most of us are familiar with playing games using gestures on Wii Fit, X-box, and PlayStation games such as "Just Dance" and "Kinect Sports".

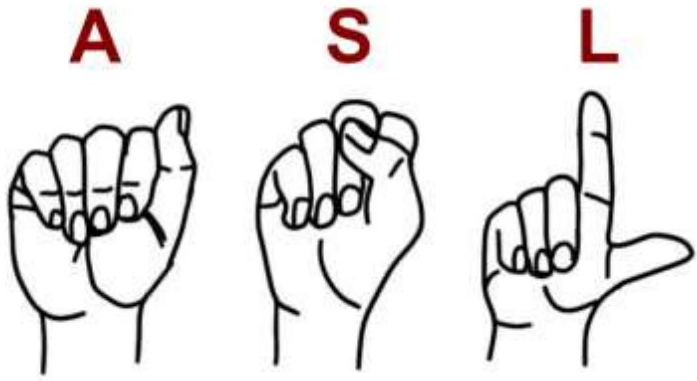
What is ASL?



American Sign Language (ASL) is a visual sign language used by most deaf communities in the United States and Canada. ASL is a natural language with a different structure than spoken English. It is not a tangible representation of the English language, nor is it a pantomime. However, ASL is a complete language with all the characteristics of spoken natural languages. Still, it has developed independently and differs from English. American sign language is the fourth most used language in the United States and Canada. It is recognized by 40 US states as a foreign language for academic purposes.

Where is ASL used?

As the name suggests, it is mostly used in America but is growing rapidly. Although no one knows the origin of this language, its popularity in the world is drawing to making it one of the most used sign languages.



Why use TinyML for ASL

To answer this question, let's consider how the benefits of using TinyML aid this application.

1. **Internet connectivity and dependence**: Our solution is independent of the internet and will work anywhere regardless of internet connection, hence no additional dependence.
2. **Electricity**: Since it is TinyML-based, it will work on battery, hence no issues with electricity. Moreover, the small electricity consumption of the device will enable it to run for days.
3. **Easy to Use**: The final device consists of small components easily available on the market and can be worn on hand like a glove. It doesn't need an internet connection, operates on a small battery, and a mute person can easily get started with using it within seconds.

In our ASL application, we place motion sensors (accelerometer and gyroscope) on the fingertips as part of a hand glove. When a gesture is

made, motion sensors capture the data for further processing.

Inputs And Outputs

We will be using our Arduino microcontroller for receiving input as well as producing the output. The Arduino attached to your hand will follow the movement as you move your hand. The readings sent by the gyroscope and accelerometer via Arduino will be input to a TinyML model. Before we dive further into how the input to the Arduino will be processed, it is better to understand the working of the gyroscope and the accelerometer.

Gyroscope: A gyroscopic sensor is a device that uses the gravity of the earth to determine orientation. It is a type of sensor found within the IMU (Inertial Measurement Unit) category. A gyroscope can measure the rotation of a certain axis. The device consists of a rotor that is simply a freely rotating disk. The rotor is mounted on a rotating shaft located in the center of another large wheel. The Coriolis force or Coriolis effect is used

to measure the momentum. The gyroscopic sensor works on the principle of maintaining momentum. The rotor or rotating wheel is attached to a pin in a gyroscopic sensor. The pivot pin causes the rotor to rotate on a certain axis called the Cardan shaft. Here we use two gimbals at the same time. One gimbal will be mounted on the other. This will give the rotor three degrees of freedom. Each time we turn the gyroscope rotor, the gyroscope stays in the same direction. Due to their axis determining abilities, gyroscopes are used in various applications. The applications range from simple compasses to complex handling of autopilots used in airplanes, missiles, and self-driving cars.

Accelerometer: An accelerometer is an electronic sensor that measures the acceleration forces that move an object. It is used to determine the object's position in space and control the object's movement. Acceleration, a vector number, is the speed at which an object's speed changes (speed is the movement of an object divided by a change in time). There are two types of acceleration forces: static forces and dynamic forces. Static forces constantly act on an object (such as friction or gravity). Dynamic forces are "moving" forces applied to an object at different speeds (such as vibrations or a force applied to a cue ball in a pool game). Accelerometers are used, for example, in car collision safety systems. When a strong dynamic force drives the car, the accelerometer (which senses the rapid deceleration) sends an electronic signal to the integrated computer, ejecting the airbags.

Once the gyroscope and accelerometer inputs are digitized, they are passed to a trained deep learning model already loaded into the device. The model will then process the inputs and, based on its training, will determine what you meant by the gesture. Finally, the speakers attached to the device will play the audio output.

ASL Application Steps

At a high level, ASL application is divided into the following 5 modules. Gesture motion data (from accelerometer and gyroscope) is fed to a well designed neural network. We'd learn how to generate an asl neural network using no-code and low-code approaches later in the chapter. Neural network classifies the word from the motion data, which is then fed to the speaker converting the sign gesture into a spoken language word.

No-Code Deep Learning Model

It is intuitive to think that one needs deep learning knowledge to create a deep learning model. AITS has created a no-code platform called cAInvas (http://cAInvas.ai-tech.systems/) for several applications to generate end-to-end deep learning solutions. No-code platforms allow programmers and non-programmers to create application software through graphical user interfaces instead of traditional computer programming. CAInvas (http://cAInvas.ai-tech.systems/) features **Playground** and **Stadium** are no code user interfaces that lets users

capture data and generate deep learning applications with no knowledge of deep learning or programming for that matter. This chapter explores the no-code feature to allow users with no programming experience to create their deep learning models capable of understanding ASL. All you have to do is log in to the CAInvas platform and create your deep learning models for the desired microcontroller.

Below we present a step-by-step procedure to implement an ASL interpreter application on a given microcontroller. We will use the Arduino Sense microcontroller, an efficient microcontroller unit (MCU) in a small form factor.

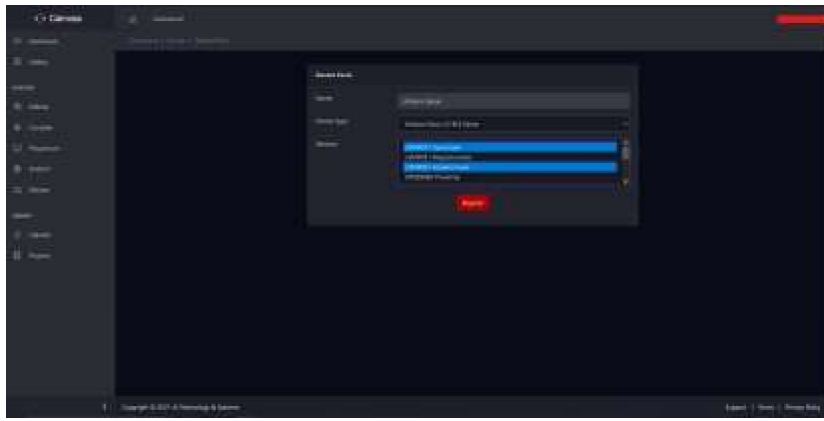
You need to follow three main steps to use the CAInvas platform for your no-code ASL detector. These steps are:

- downloading the MCU's Software Development Kit (SDK),
- flashing the SDK binary to the MCU,
- uploading a data set and training the ML model.

These three steps are explained below.

Downloading device SDK

1. After signing into the CAInvas platform, go to 'devices.'
2. Enter a device name as per your convenience
3. From the drop-down menu, select your device. For this example, we choose the Arduino sense.



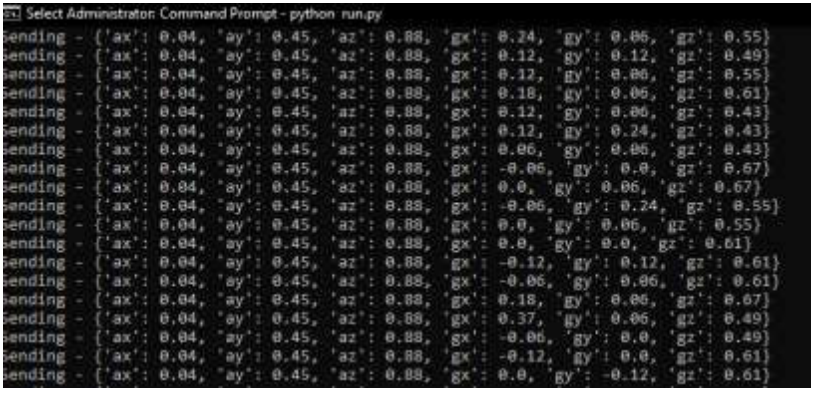
4. Now from sensors drop-down select gyroscope and accelerometer
5. Now Click on Register

6. Your device will get registered. Now click on Download below the SDK to download the SDK for your device.
7. This will provide you with all required files and packages so that you can acquire reading from your Arduino and send it to CAInvas by just running the program.

Flashing Arduino

In order to properly connect the Arduino, you need to flash it with the code you just downloaded from the CAInvas platform. To do so, follow these simple steps.

1. Unzip the SDK file
2. Plug in the Arduino
3. Open command prompt with admin privileges
4. Navigate to the extracted SDK folder
5. Type python run.py
6. It will automatically flash the Arduino and Arduino will be connected to the server, allowing you to capture data directly from the sensors

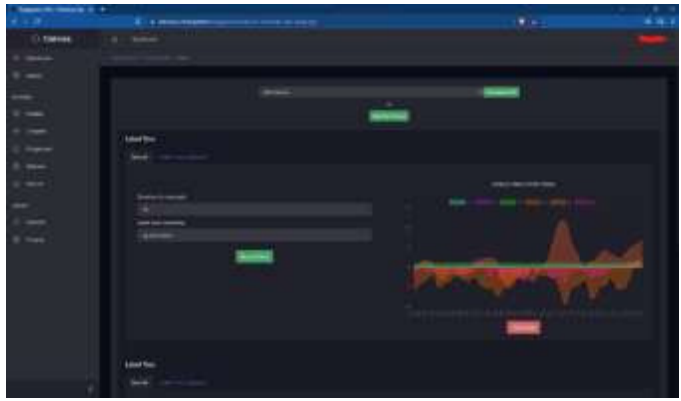


As you can see in the above image, ax, ay, az are the readings from the accelerometer and gx, gy, gz are the readings from the gyroscope.

The dataset

1. Click on playground then select ASL - American Sign Language

2. Click on the drop down above, select device and select the device we just created



3. Now In Label One
    a. Set the Duration in seconds
    b. Select the label (what you want to label sample as)
    c. Click on record data
4. Now repeat the same steps for label two

This completes the dataset collection step.

Model Training and Testing

Once you are done with the dataset collection, click the "**Train**" button to train a new model.

Once trained, the generated ML model will have the ability to detect the gesture for which it was trained. Similarly, users can record a variety of gestures and train the model with a bunch of different data sets to make their models more accurate in predictions of gestures.

Once training is finished, the UI screen changes to testing. You can record the gesture again and hit the button "Test" to test the accuracy of the deep learning model. compilation

Compile & Download

Once you are satisfied with test results, you can pick your hardware board of your choice from the pulldown menu at the bottom of your test results screen and hit the "**Compile**" button. This will kick-off compilation in the background and download the deep learning application binary on your laptop or desktop. You can locate the binary and flash it as per the instructions provided by the hardware vendor.

Low Code Deep Learning Model

This method for developing an ASL ML model is intended for the python developers without understanding deep learning theory. For Low Code, we will be using the Jupyter ipython notebook to collect the data, train, and compile the model. C++ python notebook is used to integrate the model in a C/C++ application. Following steps are used for creating a deep learning model.

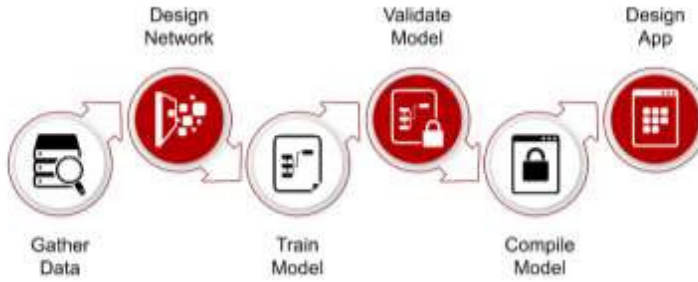


Figure: Steps used for creating an ASL deep learning model

The Dataset

You can create your dataset by capturing the motion of your desired gesture. However, make sure you save it in a CSV file format.

For example, you want to record the gesture **'Hi'**.To do so, place the Arduino on your hand and slowly make the gesture of **'Hi'** and move your fist back a few times to record the sensor data.

Take another gesture '**Sup**' and repeat the same for it. In this way, you can record data for various other gestures. Once you are done recording all the gestures, you have to upload the data in a CSV file (spreadsheet). Remember to put the correct data under the specified column.

Model Training

Training deep learning models is a process in which a neural network is fed with sufficient training data to learn from. Deep learning model training gives computers the ability to process data without the need for human interaction. This helps it detect patterns in the dataset and predict or classify an input sample that is not part of the dataset with a high degree of accuracy. A neural network is composed of multiple layers (a.k.a. operators) with several neurons (with model parameters known as weights

and biases). The model parameters are initialized randomly and iterated over a few hundred times. Each iteration with every training sample corrects the model parameters to minimize error measured in the output of the neural network.

For training, we first convert the dataset to tensors so that we can apply the operations quickly and process them efficiently. For that, we first parse the CSV files and transform them into a format that can be used to train the fully connected neural network.

If you've recorded additional gestures, update the GESTURES list with the names of the additional CSV files to increase the ASL vocabulary.

Using an 80:20 ratio, we split the data for testing and training. The training set will be used to train the model, and testing/validation data will be used to test how the model is performing.

Testing & Validation

We use 20% of the data from the recorded gesture and pass it to the model. Then we look at the prediction made by the model. After processing all the data from the testing data, we calculate how many of them were right and how many were wrong. Plotting its percentage helps in understanding how the model is actually performing. As a result, you can tune the model and finalize it according to your needs.

Saving the Model

We present a scenario to understand better the importance of saving the trained model when doing manual deep learning. Imagine a scenario with a

really good deep learning model application with a high degree of accuracy and precision, that took about 30 hours to train the model with a large dataset. If you have not saved the model now, you would have to repeat the entire model's training for 30 hours if you want to use it in other applications or with other datasets. For this reason, saving the model is a useful and important step. If you are programming the model, a few lines of additional code for saving the model can save you a lot of time and resources.

Since the model presented in the notebook for ASL is built using PyTorch, you can save the model using the **'torch.save'** method for pytorch or **model.save** for TensorFlow on model objects. Best practice while saving the model is to save the model when:

1. It has been thoroughly trained for a significant amount of time.
2. When it shows the highest accuracy during the testing phase after training.

Model Inference

Typically, data scientists are not necessarily tasked to deploy deep learning model inference systems. In the industry, successful deep learning deployments result from tight coordination between different teams, and new software technologies are often deployed to simplify the process. New systems known as "MLOps" are starting to put more structure and resources into bringing ML models into production and maintaining those models when changes are needed.

As of yet, the ASL model we have been training has been dealing with data tailored by us. From now onwards, it's time to give the model some real-time data and see its performance. This will tell us how the model will perform in a real-world scenario, allowing us to make the necessary tweaks in the dataset/ training of the model.

Compilation

After the model is saved, it needs to be compiled to run on a chosen hardware (like a microcontroller or an IoT device). AITS deepSea (https://www.ai-tech.systems/deepSea/) is used to compile saved models to an application binary. CAInvas provides a deepCC API to compile the saved model. Once the model is compiled as a library, CAInvas provides C++ notebook and compiler for integration of compiled libraries into a complete application. Complete application requires integration of other modules including motion sensor data capture and feeding to the neural network on the input side and decoding predicted word and feeding it to a speaker.

Model Analysis & OS Testing

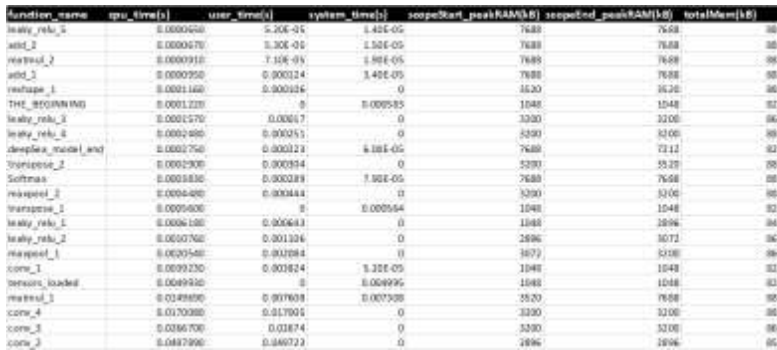
Occasionally, developers might need to understand and improve the performance of the model in order to design a better model suitable for the underlying hardware. Analysis and debugging are the process of analyzing how your program runs, how it generates data in order to find defects and issues in your code. deepSea provides features to analyze and debug the deep learning model library. Moreover, deepSea can provide the source code to facilitate this and continue debugging on an operating system like Linux or windows before flashing on to a microcontroller with sensors.

1. First step is to use deepSea to compile the library for the OS you're planning to analyze the model.
2. Next step to write a C++ wrapper to call deepSea API
    a. Find an input sample representative of sensor data and convert it into a deepSea vector or tensor as appropriate.
    b. Call deepSea API and store the result.

c. Compare and analyze the output with the expected golden result.

3. Compile the wrapper with appropriate debugging flags.

The deepSea compiler also provides options to generate a performance file to analyze model performance compiled on operating systems including ubuntu, raspberry pi etc. This performance report is generated as a csv file and can be opened in a spreadsheet application like Microsoft excel.



function_name	cpu_time(s)	user_time(s)	system_time(s)	scopeStart_peakRAM(kB)	scopeEnd_peakRAM(kB)	totalMem(kB)
leaky_relu_5	0.0000660	5.20E-05	1.40E-05	7688	7688	8028
add_2	0.0000670	5.30E-05	1.50E-05	7688	7688	8028
reshape_2	0.0000910	7.10E-05	1.90E-05	7688	7688	8028
add_3	0.0000950	0.000124	3.40E-05	7688	7688	8028
reshape_3	0.0001160	0.000126	0	3520	3520	8028
THE_BEGINNING	0.0001220	0	0.000585	1048	1048	8228
leaky_relu_3	0.0001570	0.00017	0	3200	3200	8668
leaky_relu_6	0.0002480	0.000251	0	3200	3200	8828
deepSea_model_end	0.0002750	0.000223	6.18E-05	7688	7712	8272
transpose_2	0.0002900	0.000334	0	3200	3520	8828
Softmax	0.0003830	0.000289	7.90E-05	7688	7688	8828
maxpool_2	0.0004480	0.000444	0	3200	3200	8828
transpose_1	0.0005600	0	0.000564	1048	1048	8228
leaky_relu_1	0.0006100	0.00063	0	1048	2896	8412
leaky_relu_2	0.0010760	0.001106	0	2896	3072	8676
maxpool_1	0.0020540	0.002084	0	3072	3200	8868
conv_1	0.0039230	0.003824	5.20E-05	1048	1048	8228
tensors_loaded	0.0049930	0	0.004995	1048	1048	8228
reshape_1	0.0149890	0.007608	0.007308	3520	7688	8828
conv_4	0.0170080	0.017001	0	3200	3200	8828
conv_5	0.0286700	0.03174	0	3200	3200	8868
conv_3	0.0497990	0.046723	0	2896	2896	8548

This report is helpful in root cause analysis of performance and provides following performance metric:

1. **Function Name:** it is the name of the layer used in the library and typically representative of the deep learning operator followed by an index. For example, conv_1 is the first convolution operator of the neural network.
2. **CPU time:** CPU time (or process time) is the amount of time for which a CPU was used for processing instructions of the layer or function name in the user space.
3. **User Time:** User time is the amount of time the CPU was busy executing code in user space.

4. **System Time:** System time is the amount of time the CPU was busy executing code in kernel space. This time is an indication of deep learning model parameter swapping.
5. **Start RAM:** Start peak RAM is the amount of RAM used by the process before execution of the function mentioned in the first column.
6. **End RAM:** End peak RAM is the amount of RAM used by the process before execution of the function mentioned in the first column.
7. **Total Mem:** Total memory is the peak RAM used by the process or operator indicated by the function name during its execution.

Using this report, a developer can easily spot performance or memory drag of the neural network caused by a specific operator, which can be addressed by a new neural network design including changing the type of operators.

Deploying on Microcontroller

After compiling and successfully debugging your trained ASL gesture recognition application, it is time to save the model and deploy it onto the microcontroller device. Assuming that you have used the notebook server provided in the CAInvas platform and compiled using the deepCC compiler, you should have compiled binary of the application in the .bin format. Once downloaded, locate the files in your computer, set microcontroller into flashing mode and run flashing script to flash the files onto the microcontroller attached to your computer.

Summary

In conclusion, there are two methods of implementing the ASL application using an MCU. Firstly, there is the no-code method using the Playground feature from the CAInvas platform. This method is quite straightforward and user-friendly for the most part.

The second method is the low-code, which requires that the user has good programming knowledge and demands more prerequisites, like high-quality data sets and efficient debugging for the method to work.

CAInvas provides an easy and flexible framework to develop tinyML models for resource constrained devices. It provides valuable features to design, optimize, debug, train, and analyze tinyML models.