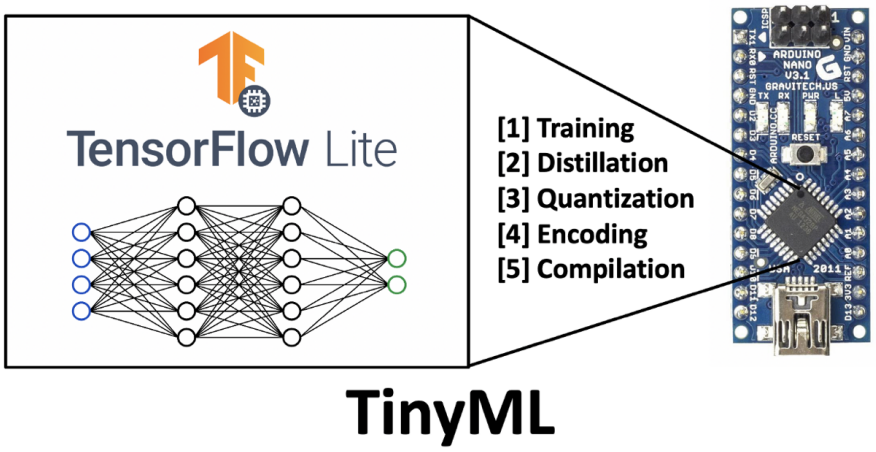
Tiny Machine Learning: The Next AI Revolution

The bigger model is not always the better model

Matthew Stewart, PhD Researcher Follow
Oct 2 · 16 min read ★

> *Miniaturization of electronics started by NASA's push became an entire consumer products industry. Now we're carrying the complete works of Beethoven on a lapel pin listening to it in headphones.* — **Neil deGrasse Tyson, astrophysicist and science commentator**

> *[…] the pervasiveness of ultra-low-power embedded devices, coupled with the introduction of embedded machine learning frameworks like TensorFlow Lite for Microcontrollers will enable the mass proliferation of AI-powered IoT devices.* — **Vijay Janapa Reddi, Associate Professor at Harvard University**



Overview of tiny machine learning (TinyML) with embedded devices.

This is the first in a series of articles on tiny machine learning. The goal of this article is to introduce the reader to the idea of tiny machine learning and its future potential. In-depth discussion of specific applications, implementations, and tutorials will follow in subsequent articles in the series.

Introduction

Over the past decade, we have witnessed the size of machine learning algorithms grow exponentially due to improvements in processor speeds and the advent of big data. Initially, models were small enough to run on local machines using one or more cores within the central processing unit (CPU).

Shortly after, computation using graphics processing units (GPUs) became necessary to handle larger datasets and became more readily available due to introduction of cloud-based services such as SaaS platforms (e.g., Google Colaboratory) and IaaS (e.g., Amazon EC2 Instances). At this time, algorithms could still be run on single machines.

More recently, we have seen the development of specialized application-specific integrated circuits (ASICs) and tensor processing units (TPUs), which can pack the power of ~8 GPUs. These devices have been augmented with the ability to distribute learning across multiple systems in an attempt to grow larger and larger models.

This came to a head recently with the release of the GPT-3 algorithm (released in May 2020), boasting a network architecture containing a staggering 175 billion neurons — more than double the number present in the human brain (~85 billion). This is more than 10x the number of neurons than the next-largest neural network ever created, Turing-NLG (released in February 2020, containing ~17.5 billion parameters). Some estimates claim that the model cost around $10 million dollars to train and used approximately 3 GWh of electricity (approximately the output of three nuclear power plants for an hour).

While the achievements of GPT-3 and Turing-NLG are laudable, naturally, this has led to some in the industry to criticize the increasingly large carbon footprint of the AI industry. However, it has also helped to stimulate interest within the AI community towards more energy-efficient computing. Such ideas, like more efficient algorithms, data representations, and computation have been the focus of a seemingly unrelated field for several years: **tiny machine learning**.

Tiny machine learning (tinyML) is the intersection of machine learning and embedded internet of things (IoT) devices. The field is an emerging engineering discipline that has the potential to revolutionize many industries.

The main industry beneficiaries of tinyML are in edge computing and energy-efficient computing. TinyML emerged from the concept of the internet of things (IoT). The traditional idea of IoT was that data would be

sent from a local device to the cloud for processing. Some individuals raised certain concerns with this concept: privacy, latency, storage, and energy efficiency to name a few.

Energy Efficiency. Transmitting data (via wires or wirelessly) is very energy-intensive, around an order of magnitude more energy-intensive than onboard computations (specifically, multiply-accumulate units). Developing IoT systems that can perform their own data processing is the most energy-efficient method. AI pioneers have discussed this idea of "data-centric" computing (as opposed to the cloud model's "compute-centric") for some time and we are now beginning to see it play out.

Privacy. Transmitting data opens the potential for privacy violations. Such data could be intercepted by a malicious actor and becomes inherently less secure when warehoused in a singular location (such as the cloud). By keeping data primarily on the device and minimizing communications, this improves security and privacy.

Storage. For many IoT devices, the data they are obtaining is of no merit. Imagine a security camera recording the entrance to a building for 24 hours a day. For a large portion of the day, the camera footage is of no utility, because nothing is happening. By having a more intelligent system that only activates when necessary, lower storage capacity is necessary, and the amount of data necessary to transmit to the cloud is reduced.

Latency. For standard IoT devices, such as Amazon Alexa, these devices transmit data to the cloud for processing and then return a response based on the algorithm's output. In this sense, the device is just a convenient gateway to a cloud model, like a carrier pigeon between yourself and Amazon's servers. The device is pretty dumb and fully dependent on the speed of the internet to produce a result. If you have slow internet, Amazon Alexa will also become slow. For an intelligent IoT device with onboard automatic speech recognition, the latency is reduced because there is reduced (if not no) dependence on external communications.

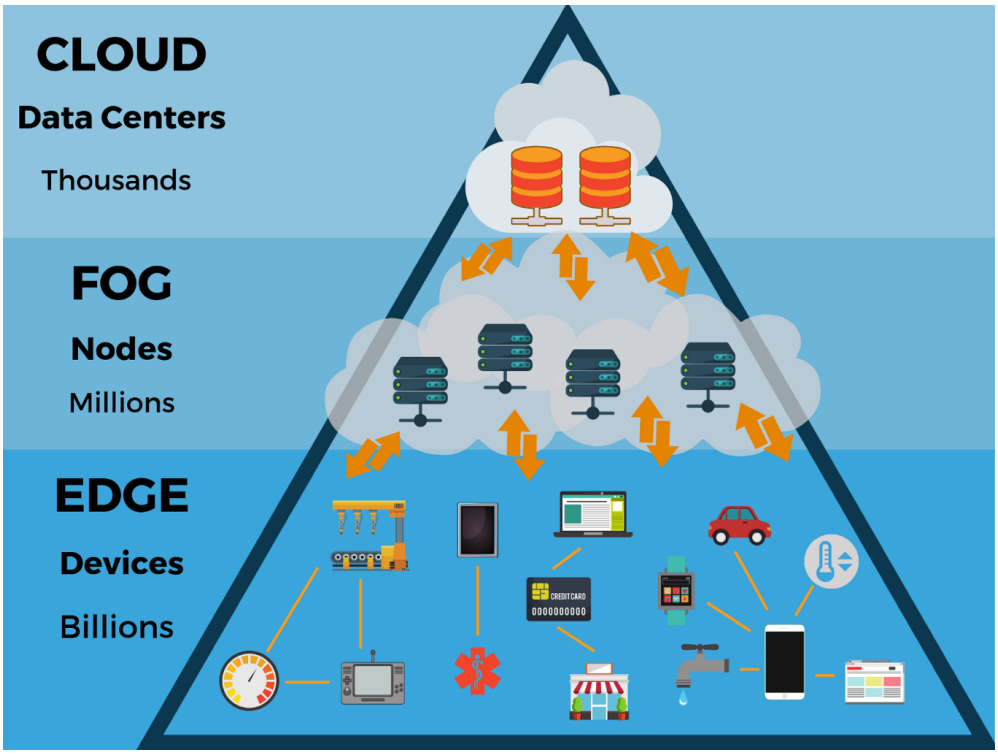
These issues led to the development of edge computing, the idea of performing processing activities onboard of edge devices (devices at the "edge" of the cloud). These devices are highly resource-constrained in terms of memory, computation, and power, leading to the development of more efficient algorithms, data structures, and computational methods.

Such improvements are also applicable to larger models, which may lead to efficiency increases in machine learning models by orders of magnitude with no impact on model accuracy. As an example, the Bonsai algorithm

developed by Microsoft can be as small as 2 KB but can have even **better** performance than a typical 40 MB kNN algorithm, or a 4 MB neural network. This result may not sound important, but the same accuracy on a model 1/10,000th of the size is quite impressive. A model this small can be run on an Arduino Uno, which has 2 KB RAM available — in short, you can now build such a machine learning model on a $5 microcontroller.

We are at an interesting crossroads where machine learning is bifurcating between two computing paradigms: compute-centric computing and data-centric computing. In the compute-centric paradigm, data is stockpiled and analyzed by instances in data centers, while in the data-centric paradigm, the processing is done locally at the origin of the data. Although we appear to be quickly moving towards a ceiling in the compute-centric paradigm, work in the data-centric paradigm has only just begun.

IoT devices and embedded machine learning models are becoming increasingly ubiquitous in the modern world (predicted more than 20 billion active devices by the end of 2020). Many of these you may not even have noticed. Smart doorbells, smart thermostats, a smartphone that "wakes up" when you say a couple of words, or even just pick up the phone. The remainder of this article will focus deeper on how tinyML works, and on current and future applications.



The hierarchy of the cloud. (Source: eBizSolutions)

Examples of TinyML

Previously, complex circuitry was necessary for a device to perform a wide range of actions. Now, machine learning is making it increasingly possible to abstract such hardware "intelligence" into software, making embedded devices increasingly simple, lightweight, and flexible.

The challenges that machine learning with embedded devices presents are considerable, but great progress has already been achieved in this area. The key challenges in deploying neural networks on microcontrollers are the low memory footprint, limited power, and limited computation.

Perhaps the most obvious example of TinyML is within smartphones. These devices perpetually listen actively for '**wake words**', such as "Hey Google" for Android smartphones, or 'Hey Siri" on iPhones. Running these activities through the main central processing unit (CPU) of a smartphone, which is 1.85 GHz for the modern iPhone, would deplete the battery in just a few hours. This level of degradation is not acceptable for something that most people would use a few times a day at most.

To combat this, developers created specialized low-power hardware that is able to be powered by a small battery (such as a circular CR2032 "coin" battery). These allow the circuits to remain active even when the CPU is not running, which is basically whenever the screen is not lit.

These circuits can consume as little as 1 mW and can be powered for up to a year using a standard CR2032 battery.

It may not seem like it, but this is a big deal. Energy is a limiting factor for many electronic devices. Any device that requires mains electricity is restricted to locations with wiring, which can quickly get overwhelming when a dozen devices are present in the same location. Mains electricity is also inefficient and expensive. Converting mains voltage (which operates around 120 V in the United States) to a typical circuit voltage range (often ~5 V) wastes large amounts of energy. Anyone with a laptop charger will probably know this when unplugging their charger. The heat from the transformer within the charger is wasted energy during the voltage conversion process.

Even devices with batteries suffer from limited battery life, which requires frequent docking. Many consumer devices are designed such that the battery lasts for a single workday. TinyML devices that can continue operating for a year on a battery the size of a coin mean they can be placed in remote environments, only communicating when necessary in order to conserve energy.

Wake words are not the only TinyML we see seamlessly embedded in smartphones. Accelerometer data is used to determine whether someone has just picked the phone up, which wakes the CPU and turns on the screen.

Clearly, these are not the only possible applications of TinyML. In fact, TinyML presents many exciting opportunities for businesses and hobbyists alike to produce more intelligent IoT devices. In a world where data is becoming more and more important, the ability to distribute machine learning resources to memory-constrained devices in remote locations could have huge benefits on data-intensive industries such as farming, weather prediction, or seismology.

It is without a doubt that empowering edge devices with the capability of performing data-driven processing will produce a paradigm shift for industrial processes. As an example, devices that are able to monitor crops and send a "help" message when it detects characteristics such as soil moisture, specific gases (for example, apples emit ethane when ripe), or particular atmospheric conditions (e.g., high winds, low temperatures, or high humidity), would provide massive boosts to crop growth and hence crop yield.

As another example, a smart doorbell might be fitted with a camera that can use facial recognition to determine who is present. This could be used for security purposes, or even just so that the camera feed from the doorbell is fed to televisions in the house when someone is present so that the residents know who is at the door.

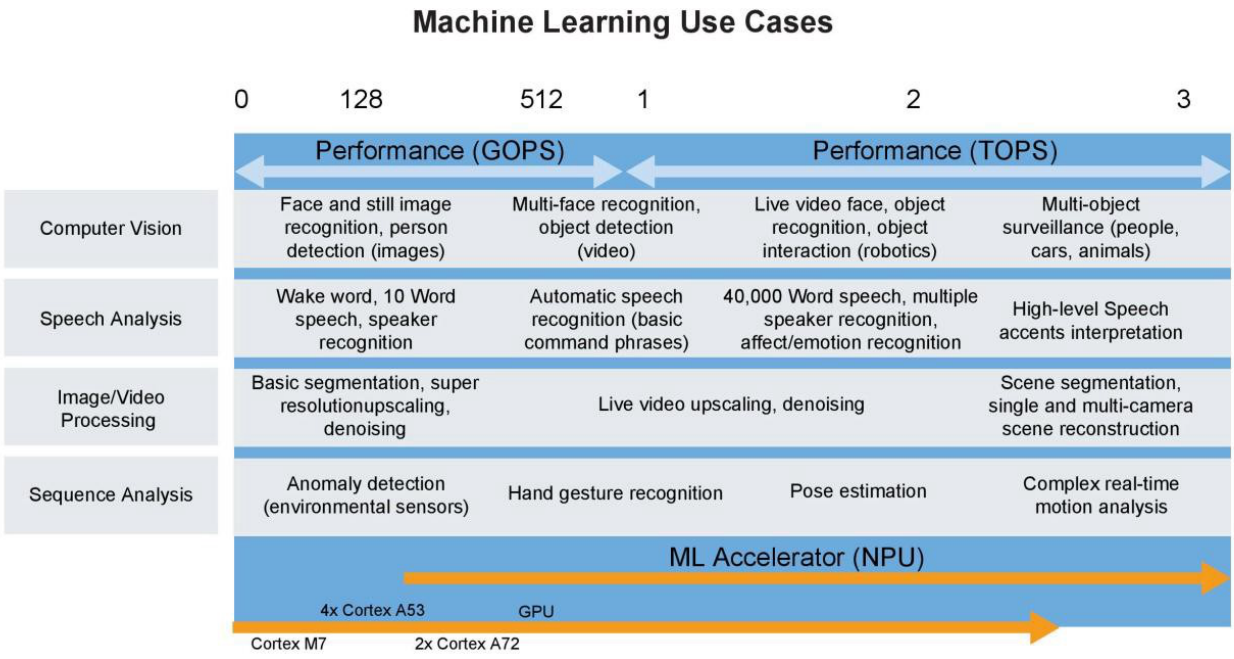
Two of the main focus areas of tinyML currently are:

Keyword spotting. Most people are already familiar with this application. "Hey Siri" and "Hey Google" are examples of keywords (often used synonymously with hotword or wake word). Such devices listen continuously to audio input from a microphone and are trained to only respond to specific sequences of sounds, which correspond with the learned keywords. These devices are simpler than automatic speech recognition (ASR) applications and utilize correspondingly fewer resources. Some devices, such as Google smartphones, utilize a cascade architecture to also provide speaker verification for security.

Visual Wake Words. There is an image-based analog to the wake words known as visual wake words. Think of these as a binary classification of an image to say that something is either present or not present. For example, a smart lighting system may be designed such that it activates when it detects the presence of a person and turns off when they leave. Similarly, wildlife

photographers could use this to take pictures when a specific animal is present, or security cameras when they detect the presence of a person.

A more broad overview of current machine learning use cases of TinyML is shown below.



Machine learning use cases of TinyML (Source Image: NXP).

How TinyML Works

TinyML algorithms work in much the same way as traditional machine learning models. Typically, the models are trained as usual on a user's computer or in the cloud. Post-training is where the real tinyML work begins, in a process often referred to as **deep compression**.

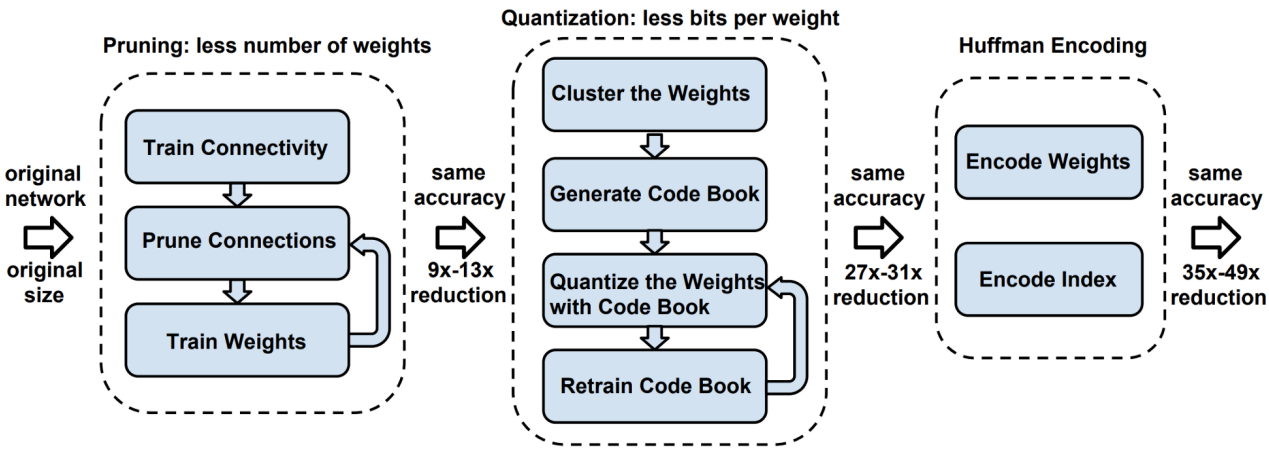


Diagram of the deep compression process. Source: ArXiv.

Model Distillation

Post-training, the model is then altered in such a way as to create a model with a more compact representation. **Pruning** and **knowledge distillation** are two such techniques for this purpose.

The idea underlying knowledge distillation is that larger networks have some sparsity or redundancy within them. While large networks have a high representational capacity, if the network capacity is not saturated it could be represented in a smaller network with a lower representation capacity (i.e., less neurons). Hinton et al. (2015) referred to the embedded information in the teacher model to be transferred to the student model as "**dark knowledge**".

The below diagram illustrates the process of knowledge distillation.

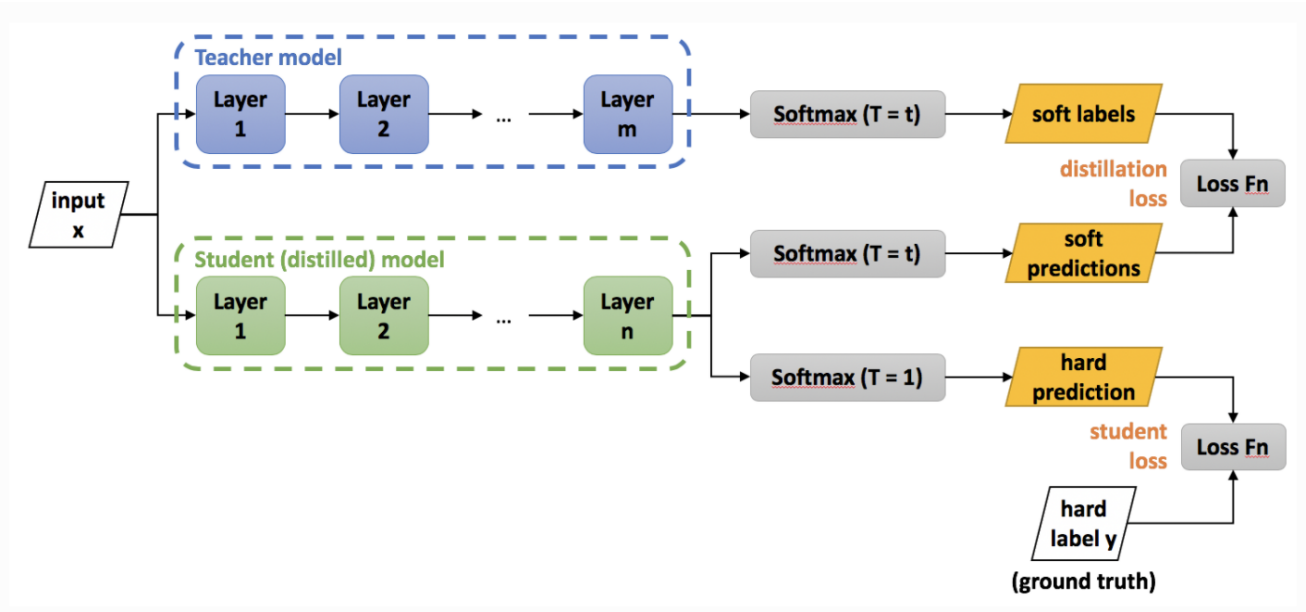


Diagram of the deep compression process. In this diagram, the 'teacher' is a trained convolutional neural network model. The teacher is tasked with transferring its 'knowledge' to a smaller convolutional network model with fewer parameters, the 'student'. This process is known as knowledge distillation and is used to enshrine the same knowledge in a smaller network, providing a way of compressing networks such that they can be used on more memory-constrained devices. Source: ArXiv.

In this diagram, the ‘teacher’ is a trained neural network model. The teacher is tasked with transferring its ‘knowledge’ to a smaller network model with fewer parameters, the ‘student’. This process is used to enshrine the same knowledge in a smaller network, providing a way of compressing the knowledge representation, and hence the size, of a neural network such that they can be used on more memory-constrained devices.

Similarly, pruning can help to make the model’s representation more compact. Pruning, broadly speaking, attempts to remove neurons that provide little utility to the output prediction. This is often associated with small neural weights, whereas larger weights are kept due to their greater importance during inference. The network is then retrained on the pruned architecture to fine-tune the output.

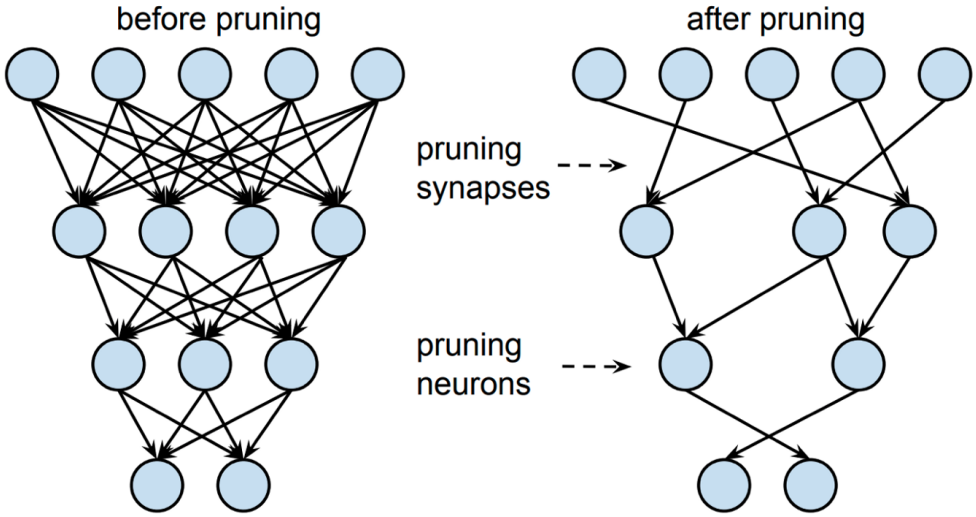


Illustration of pruning for distilling a model’s knowledge representation.

Quantization

Following distillation, the model is then quantized post-training into a format that is compatible with the architecture of the embedded device.

Why is quantization necessary? Imagine an Arduino Uno using an ATmega328P microcontroller, which uses 8-bit arithmetic. To run a model on the Uno, the model weights would ideally have to be stored as 8-bit integer values (whereas many desktop computers and laptops use 32-bit or 64-bit floating-point representation). By quantizing the model, the storage size of weights is reduced by a factor of 4 (for a quantization from 32-bit to 8-bit values), and the accuracy is often negligibly impacted (often around 1–3%).

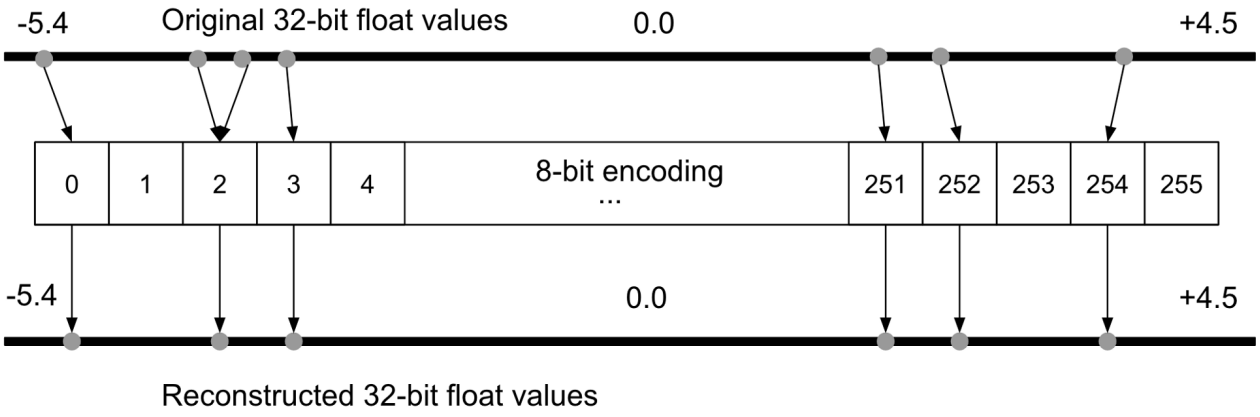


Illustration of quantization error during 8-bit encoding (which is then used to reconstruct 32-bit floats). (Source: TinyML book)

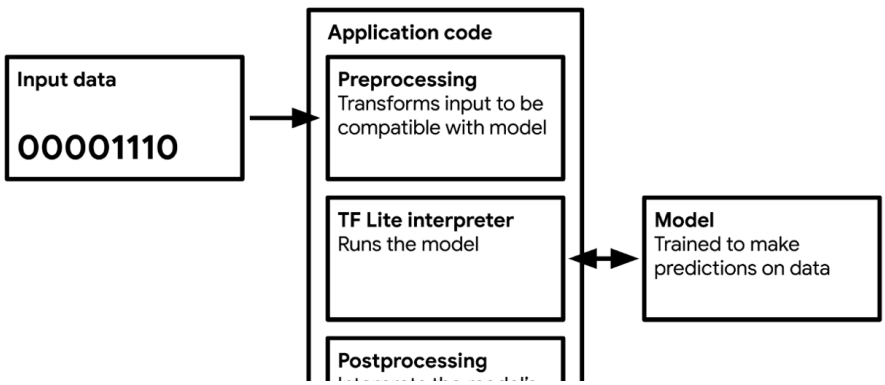
Some information may be lost during quantization due to quantization error (for example, a value that is 3.42 on a floating-point representation may be truncated to 3 on an integer-based platform). To combat this, quantization-aware (QA) training has also been proposed as an alternative. QA training essentially constrains the network during training to only use the values that will be available on the quantized device (see Tensorflow example).

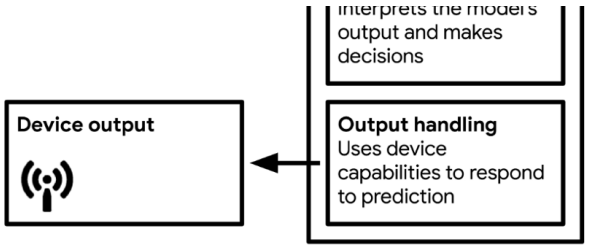
Huffman Encoding

Encoding is an optional step that is sometimes taken to further reduce the model size by storing the data in a maximally efficient way: often via the famed Huffman encoding.

Compilation

Once the model has been quantized and encoded, it is converted to a format that can be interpreted by some form of light neural network interpreter, the most popular of which are probably TF Lite (~500 KB in size) and TF Lite Micro (~20 KB in size). The model is then compiled into C or C++ code (the languages most microcontrollers work in for efficient memory usage) and run by the interpreter on-device.





The workflow of TInyML application (Source: TinyML book by Pete Warden and Daniel Situnayake)

Most of the skill of tinyML comes in dealing with the complex world of microcontrollers. TF Lite and TF Lite Micro are so small because any unnecessary functionality has been removed. Unfortunately, this includes useful abilities such as debugging and visualization. This means that it can be difficult to discern what is going on if there is an error during deployment.

Additionally, while the model has to be stored on the device, the model also has to be able to perform inference. This means the microcontroller must have a memory large enough that it can run (1) its operating system and libraries, (2) a neural network interpreter such as TF Lite, (3) the stored neural weights and neural architecture, and (4) the intermediate results during inference. Thus, the peak memory usage of a quantized algorithm is often quoted in tinyML research papers, along with memory usage, the number of multiply-accumulate units (MACs), accuracy, etc.

Why not train on-device?

Training on-device brings about additional complications. Due to reduced numerical precision, it becomes exceedingly difficult to guarantee the necessary level of accuracy to sufficiently train a network. Automatic differentiation methods on a standard desktop computer are approximately accurate to machine precision. Computing derivatives to the accuracy of 10^-16 is incredible, but utilizing automatic differentiation on 8-bit values will result in poor results. During backpropagation, these derivatives are compounded and eventually used to update neural parameters. With such a low numerical precision, the accuracy of such a model may be poor.

That being said, neural networks have been trained using 16-bit and 8-bit floating-point numbers.

The first paper looking at reducing numerical precision in deep learning was the 2015 paper *"Deep Learning with Limited Numerical Precision"* by Suyog Gupta and colleagues. The results of this paper were interesting, showing that the 32-bit floating-point representation could be reduced to a 16-bit fixed-point representation with essentially no degradation in accuracy. However, this is the only case when **stochastic rounding** is used

because, on average, it produces an unbiased result.

In 2018, Naigang Wang and colleagues trained a neural network using 8-bit floating point numbers in their paper *"Training Deep Neural Networks with 8-bit Floating Point Numbers"*. Training a neural network using 8-bit numbers rather than inference is significantly more challenging to achieve because of a need to maintain fidelity of gradient computations during backpropagation (which is able to achieve machine precision when using automatic differentiation).

How about compute-efficiency?

Models can also be tailored to make them more compute-efficient. Model architectures widely deployed on mobile devices such as **MobileNetV1** and **MobileNetV2** are good examples. These are essentially convolutional neural networks that have recast the convolution operation to make it more compute-efficient. This more efficient form of convolution is known as **depthwise separable convolution**. Architectures can also be optimized for latency using **hardware-based profiling** and **neural architecture search**, which are not covered in this article.

The Next AI Revolution

The ability to run machine learning models on resource-constrained devices opens up doors to many new possibilities. Developments may help to make standard machine learning more energy-efficient, which will help to quell concerns about the impact of data science on the environment. In addition, tinyML allows embedded devices to be endowed with new intelligence based on data-driven algorithms, which could be used for anything from preventative maintenance to detecting bird sounds in forests.

While some machine learning practitioners will undoubtedly continue to grow the size of models, a new trend is growing towards more memory-, compute-, and energy-efficient machine learning algorithms. TinyML is still in its nascent stages, and there are very few experts on the topic. I recommend the interested reader to examine some of the papers in the references, which are some of the important papers in the field of tinyML. This space is growing quickly and will become a new and important application of artificial intelligence in industry within the coming years. Watch this space.

References

[1] Hinton, Geoffrey & Vinyals, Oriol & Dean, Jeff. (2015). Distilling the Knowledge in a Neural Network.

[2] D. Bankman, L. Yang, B. Moons, M. Verhelst and B. Murmann, "An always-on 3.8μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28nm CMOS," *2018 IEEE International Solid-State Circuits Conference — (ISSCC)*, San Francisco, CA, 2018, pp. 222–224, doi: 10.1109/ISSCC.2018.8310264.

[3] Warden, P. (2018). Why the Future of Machine Learning is Tiny. Pete Warden's Blog.

[4] Ward-Foxton, S. (2020). AI Sound Recognition on a Cortex-M0: Data is King. EE Times.

[5] Levy, M. (2020). Deep Learning on MCUs is the Future of Edge Computing. EE Times.

[6] Gruenstein, Alexander & Alvarez, Raziel & Thornton, Chris & Ghodrat, Mohammadali. (2017). A Cascade Architecture for Keyword Spotting on Mobile Devices.

[7] Kumar, A., Saurabh Goyal, and M. Varma. (2017). Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things.

[8] Zhang, Yundong & Suda, Naveen & Lai, Liangzhen & Chandra, Vikas. (2017). Hello Edge: Keyword Spotting on Microcontrollers.

[9] Fedorov, Igor & Stamenovic, Marko & Jensen, Carl & Yang, Li-Chia & Mandell, Ari & Gan, Yiming & Mattina, Matthew & Whatmough, Paul. (2020). TinyLSTMs: Efficient Neural Speech Enhancement for Hearing Aids.

[10] Lin, Ji & Chen, Wei-Ming & Lin, Yujun & Cohn, John & Gan, Chuang & Han, Song. (2020). MCUNet: Tiny Deep Learning on IoT Devices.

[11] Chen, Tianqi & Moreau, Thierry. (2020). TVM: An Automated End-to-End Optimizing Compiler for Deep Learning.

[12] Weber, Logan, and Reusch, Andrew (2020). TinyML — How TVM is Taming Tiny.

[13] Krishnamoorthi, Raghuraman. (2018). Quantizing deep convolutional

networks for efficient inference: A whitepaper.

[14] Yosinski, Jason & Clune, Jeff & Bengio, Y. & Lipson, Hod. (2014). How transferable are features in deep neural networks?.

[15] Lai, Liangzhen & Suda, Naveen & Chandra, Vikas. (2018). CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs.

[16] Chowdhery, Aakanksha & Warden, Pete & Shlens, Jonathon & Howard, Andrew & Rhodes, Rocky. (2019). Visual Wake Words Dataset.

[17] Warden, Pete. (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition.

[18] Zemlyanikin, Maxim & Smorkalov, Alexander & Khanova, Tatiana & Petrovicheva, Anna & Serebryakov, Grigory. (2019). 512KiB RAM Is Enough! Live Camera Face Recognition DNN on MCU. 2493–2500. 10.1109/ICCVW.2019.00305.