Advanced Topics in TinyML

Throughout this course, you have developed multiple tinyML projects on your local microcontroller. You have designed a keyword spotting algorithm that was trained using your own customized dataset. You have designed a visual wake words algorithm and then, through transfer learning, adapted this to create a mask detection algorithm. You have even created your own gesturing magic wand!

Although you have now come to the end of this class, your journey as a tinyML engineer is only just beginning. You are now equipped with a set of tools and techniques that will be invaluable in the coming years as the field of tinyML reaches ubiquity and is incorporated beyond academia into commercial markets. The exponential increase of IoT devices, increasing prevalence of sensor networks, and transition towards industry 4.0 will continue to drive further development of tinyML at an accelerating pace. Consequently, keeping up-to-date with state-of-the-art methods, as well as advanced and novel techniques will be a necessity for your success as a tinyML engineer.

In this reading, we will touch on some of the major topics that we consider “advanced” as well as potential future developments and applications which may become relevant in the near future. Such predictions are, at best, informed guesses about how the field may evolve, and may deviate depending on the conflicting needs of industry, end users, and academia. However, we believe that these topics may act as a helpful stepping stone to you in your continuing development as a tinyML engineer.

Profiling and Benchmarking

Profiling is used in software engineering to optimize software performance. Similarly, in tinyML, profiling can be used to optimize a system for its deployment. This might include finding bottlenecks, computational inefficiencies, along with other properties which may negatively impact algorithmic performance. Profiling is typically used to determine where speedup can be achieved to aid in minimizing the computational costs of running an algorithm.

To accompany this, **benchmarking** is used to provide like-for-like comparisons between an algorithmic implementation running on different hardware, different implementations running on the same hardware, and so on. Benchmarking typically involves the use of comparative metrics, which for tinyML might include inference time, operations per second, and memory utilization.

Benchmarking and profiling will become increasingly important for commercial applications where large numbers of devices are running the same code, since inefficiencies lead to higher costs and other undesirable effects. TinyMLPerf is a recent attempt to help provide a set of benchmarks for performing comparative measurements of typical tinyML tasks.

ML Operations (MLOps)

MLOps is very closely related to DevOps. DevOps focuses on seamless integration of automated testing and benchmarking capabilities to a software engineering environment, allowing updated code and algorithms to be automatically checked for errors and assessed for performance. MLOps would apply this to our tinyML algorithms, which would significantly increase the productivity of developers by providing real-time feedback which can be tailored to the deployment environment visible to the end user. Several companies focusing on MLOps already exist, and there will likely be more as market penetration of tinyML applications increases.

Edge Impulse is one such example, which makes it easy to gather your dataset, train models, optimize and deploy them. Now that you know what is underneath the hood. You can go ahead and use such tools with good progress.

Cross-platform Interoperability

Algorithm interoperability across various hardware platforms remains an important barrier to tinyML. Microcontroller devices built upon different instruction sets may have (1) different sensor capabilities, (2) be bare-metal as opposed to having a lightweight operating system, (3) different memory capabilities, with or without a memory controller, and (4) different computational capabilities (e.g., clock speed, word lengths). These differences lead to a panoply of challenges for creating algorithms that can run on various hardware platforms.

Several attempts at overcoming this issue have already been attempted. OctoML provides a compiler stack which acts as a middleman that provides interoperability between different deep learning platforms and microcontroller devices. This challenge will become vital to overcome once the tinyML ecosystem begins to propagate, since multiple vendors will have to have products that can work together seamlessly, similar to how different computers and devices are able to interact on the internet today.

Neural Architecture Search

As we have seen throughout the course, it is difficult to find the optimal network configuration for a given application and dataset. **Neural architecture search** (NAS) is a common method used in machine learning to search and test multiple architectures in an attempt to find an optimal configuration automatically. However, we are more resource-constrained with tinyML devices, which must be incorporated into our NAS. Therefore, for tinyML devices, we must recast NAS to produce an optimal configuration given the hardware constraints imposed by our system, such that we can eke out the maximal performance.

Several attempts have been made so far to do this, such as with MCUNet and MicroNets. The commercial drivers for this are strong since devices with higher performance capabilities are preferable in any given application.

No doubt there will be more items that could be added to this list in the near future, but we hope this will provide you with a valuable starting point for finding additional topics that may be useful for you in future applications.