ASL Interpreter Model

Sebastian Alberdi, Alice Yu
6.5940 TinyML
MIT
salberdi@mit.edu, alicey@mit.edu

Abstract

This paper introduces a groundbreaking American Sign Language (ASL) interpretation model, developed by our group, which stands out for its compact and efficient design. Utilizing advanced techniques like sparse pruning and channel pruning, we have significantly reduced the model's size without compromising its interpretative accuracy. Additionally, the implementation of neural architecture search (NAS) has further refined the model, optimizing it for performance on the discovery kit. This integration not only highlights the model's adaptability but also its potential to enhance accessibility for the deaf and hard-of-hearing community by providing real-time ASL interpretation. Although integration was unsuccessful, the pursuit of this goal revealed such technology could be very easily made available in the near future

1 Motivation

Our motivation to develop an easily accessible American Sign Language (ASL) video interpreter is deeply rooted in the desire to address the significant challenges faced by the deaf and hard-of-hearing community, particularly in areas where access to skilled interpreters is limited. Recognizing that good interpreters are often hard to find and even harder to access in remote or under-served areas, we sought to create a technological solution. In this way, we can bridge this gap in accessibility. By implementing our cutting-edge ASL interpretation model onto a small, convenient edge device, we aim to provide a portable and user-friendly tool that can serve as a reliable alternative to human interpreters. This innovation is not just about technological advancement; it's about making communication more accessible and inclusive. Our hope is that this device will empower individuals in remote communities, enabling them to engage in effective and seamless communication, thus fostering greater independence and connection in their daily lives.

To ensure that our model could be seamlessly integrated into various devices, including those with limited processing capabilities, we employed three distinct methods: sparse pruning, channel pruning, and neural architecture search. These techniques were meticulously chosen and implemented to significantly reduce the size of the model without compromising its functionality or accuracy. By doing so, we made it possible for the model to be compatible with a wide range of devices, from high-end computers to more basic discovery kits, thus ensuring its accessibility.

2 Pruning

2.1 Model Establishment

In the development of our American Sign Language (ASL) interpretation model, we began by adapting an existing neural network architecture. We chose the VGG model from Lab 1 for our purposes. However, to tailor it to our specific needs for ASL interpretation, we made a crucial modification to the classifier layer. VGG from Lab 1 is originally designed to categorize images into

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

10 classes. We expanded it to accommodate 29 categories in order to represent the 26 letters of the English alphabet along with three additional symbols for 'null', 'space', and 'delete,' essential for ASL communication. This modification was a critical step in customizing the model to accurately recognize and interpret the unique gestures of ASL.

Another significant adaptation involved the preprocessing of input images. The model was originally trained on images sized 32 by 32 pixels, but the dataset we found on Kaggle are all 200 by 200 pixels. To adapt the training database to our model, we resized these images down to 32x32 pixels. This resizing played a crucial role in making the model more suitable for deployment on less powerful devices, such as the STM32F7 discovery kit. By reducing the resolution of the input images, we were able to significantly decrease the computational resources required for the model to process and interpret ASL signs, without notably sacrificing the accuracy of its interpretations.

2.2 Sparse Pruning

Our main challenge was to significantly reduce the model's size without compromising its interpretive accuracy to fit on any edge device. With a 1MB of storage, we need to prune around 97% of the weights. To achieve this, we employed a magnitude-based fine-grained pruning function. This method involves a systematic process of identifying and removing less critical information from the neural network. The original size of our model was approximately 35MB, which was impractically large for our intended use. Through the application of this pruning technique, we were able to shrink the model down to a mere 970KB. This drastic reduction in size was a pivotal achievement in making our model more accessible and deployable on a wide range of devices.

The magnitude-based fine-grained pruning process we adopted consists of several key steps. In the first step, we calculated the importance of each weight tensor in the model. We shall not remove any important nodes as we need to maintain the accuracy of our predictions. We utilized PyTorch's torch.abs() and related APIs to achieve this. This step was crucial to identify which weights were essential for the model's performance and which could be discarded without significant loss of accuracy.

In the next two steps of the pruning process, we determined the pruning threshold and subsequently calculated the pruning mask based on this threshold. The threshold was set such that all synapses with an importance value smaller than the threshold would be removed. To find this value, we used PyTorch's torch.kthvalue(), torch.Tensor.kthvalue(), and torch.topk() APIs. Once the threshold was established, we created a pruning mask, where a value of 1 indicated that the synapse would be kept and 0 meant it would be removed. We heavily pruned the convolution layers as they take up the most amount of storage and has numerous insinificant weights. These final steps were critical in ensuring that our pruning was precise and effective, allowing us to retain the model's essential functionality while significantly reducing its size. Our model retained around 94.84% accuracy with a size of 970KB.

2.3 Channel Pruning

Sparse pruning has a great compression rate and maintains accuracy of the model. However, it is hard to implement in terms of hardware and the latency in computation persists. In addition to sparse pruning, we also attempted at channel pruning to adapt to more potential edge devices. Our team diligently applied channel pruning to our ASL interpretation model, which resulted in a substantial reduction of the model's size to 950KB. This is a mere 2.69% of the original model size, a remarkable achievement that underscores the efficacy of our pruning methodology. The table accompanying our results clearly illustrates the impact of our efforts. Notably, the latency, which is the time the model takes to make a prediction, was reduced from 24.3 milliseconds to just 1.3 milliseconds, an 18.8 times decrease. Furthermore, the reduction in the number of Multiply-Accumulate operations (MACs) from 606 million to just 7 million, an 89.9% decrease, signifies a dramatic reduction in computational load. This drastic reduction in latency and computation is particularly beneficial for edge devices, which require fast, real-time processing capabilities.

The initial trade-off for this significant size reduction was a drop in accuracy to 4.46%. To mitigate this, we engaged in fine-tuning the pruned model. This fine-tuning was aimed at restoring, as much as possible, the original level of accuracy while maintaining the reduced model size. Our efforts were successful, and we were able to increase the accuracy back to approximately 90.86%. This recovery

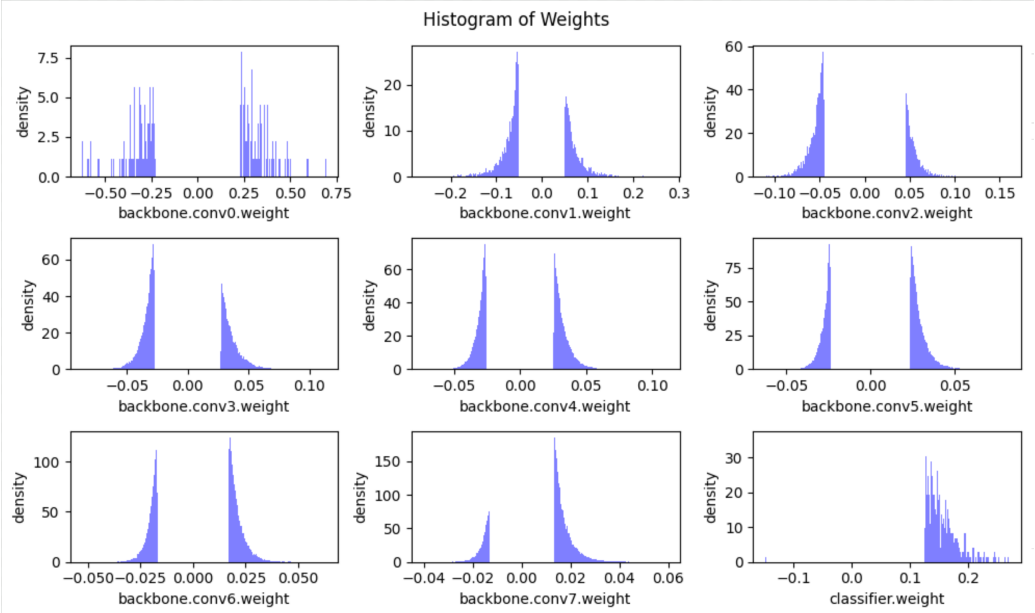

Figure 1: Weights after sparse pruning.

Table 1: Channel Pruning Results

	Original	Pruned	Reduction Ratio
Latency (ms)	24.3	1.3	18.8
MACS (M)	606	7	89.9
Params (M)	9.24	0.25	37.1

of accuracy demonstrates that even with aggressive compression techniques, it is possible to retain high performance in neural network models.

The implications of these results are significant for the deployment of ASL interpretation models on edge devices. Edge devices, characterized by their limited computational resources and the need for low latency, benefit immensely from models that are both lightweight and performant. The pruned model, now 37.1 times smaller in parameter size, will be less demanding on storage and memory, making it an ideal candidate for integration into smartphones, tablets, and other portable technology that may be used by individuals requiring ASL interpretation services. The ability to run complex models locally, without the need for a constant connection to cloud services, enhances the usability and reliability of such applications, paving the way for broader accessibility and support for the deaf and hard-of-hearing community in a variety of settings.

3 Neural Architecture Search

Realizing transforming a model pruned in the way we had to a format conducive to launching on an MCU would be a daunting task, we decided Neural Architecture Search would be best to discover a model that would fit within the constraints of our microcontroller. We took inspiration from the previous year's Lab 4 and this year's Lab 3, as well as the MCUNet project[1]. The plan was to use evolutionary search on the Once-For-All network to extract a subnet with total size less then 1Mb, then modify it for our purposes. We used evolutionary search informed by the analysis on evolutionary hyperparameters within Lab 3. The search for such an architecture was inconclusive, although we uncovered one not much larger than 2Mb. With more time, we would quantize this model and proceed.

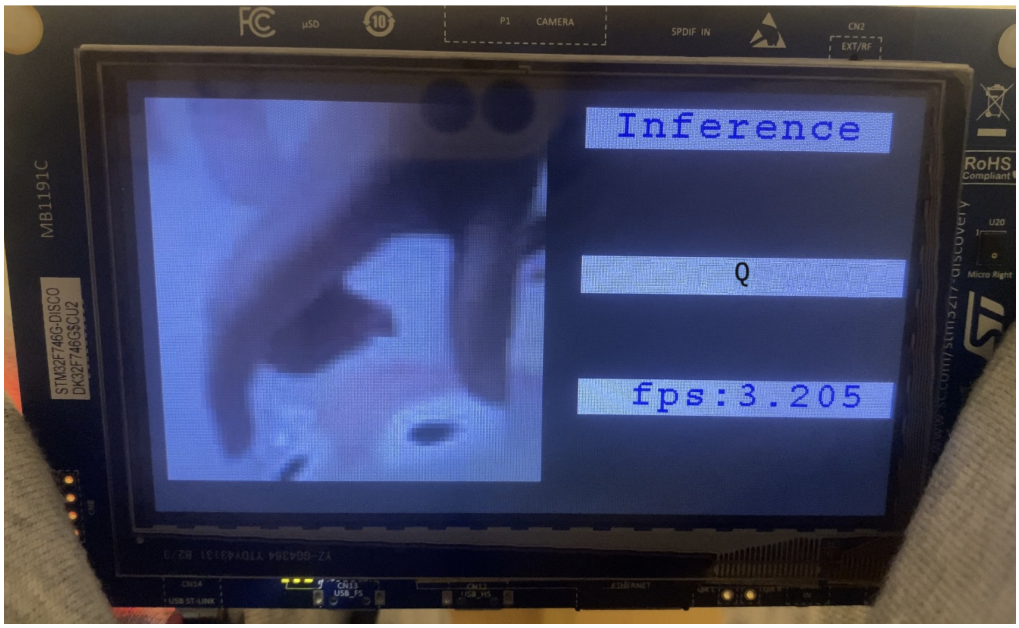

Figure 2: Design goal for ASL integration (unrealized).

4 STM32F7 Discovery Kit and ArduCam Implementation

Our innovative ASL interpretation was to be implemented on the STM32F7 Discovery Kit, a development board that boasts a powerful ARM Cortex-M7 processor capable of speeds up to 216MHz, and a 32-bit microcontroller. This kit is particularly suited for our application due to its LCD display, which is perfect for providing visual feedback from the ASL interpreter. To interface with the physical world, we utilized an ArduCam, a camera module capable of capturing images. There were functions within the MCUNet Lab that could convert a Tflite model to C source code. This conversion was critical as it allowed us to deploy a high-level neural network model onto a resource-constrained embedded system, enabling real-time ASL interpretation without the need for internet connectivity or external computing resources. The successful deployment of this model on the STM32F7 Discovery Kit would represent a significant step toward making ASL interpretation widely accessible and convenient for users in a variety of environments.

5 Conclusion and Acknowledgements

Although the project is unfinished, we have strong intuition for how to proceed. With more time, we would quantize the smallest accurate model possible found using the aforementioned NAS method. Additionally, we learned a lot about the process of model compression and the trials involved in the field of TinyML. We would like to thank all the staff for their help and wonderful instruction, the class was very well-taught!

References

[1] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning, 2021.