



Model Compression on SPU-001

FemtoseNSE
Document version 1.0

5/12/2024

Debayan Ghosh, Raphael Abbou,
Rohit Pidaparathi, Scott Reid

Introduction

The Femtosense SDK enables model developers to compress their deep neural networks with sparsity techniques and then deploy them efficiently in the tiny, Femtosense Sparse Processing Unit (SPU). This document provides a high-level overview of sparse neural network compression, and then provides links to relevant documentation and tutorials so that model developers can quickly learn to use the Femtosense SDK and deploy their own sparse models.

Overview of Sparsity and the SPU

The Sparse Processing Unit (SPU) enables sparse deep learning algorithms to run efficiently in a tiny embedded form factor. With sparsity, large neural networks can be compressed so that they can be run at lower power and with less memory. However, sparse models do not run efficiently on any hardware accelerator – the hardware needs special instructions and memory formats to fully reap the benefits in memory, latency, and efficiency – otherwise its potential benefits are lost in overhead. With first-class ISA-level support for sparsity, the SPU can fully benefit from sparse model compression.

- Memory footprint scales as δ_{param} (parameter density)¹
- Energy and latency scale roughly as $\delta_{param} \cdot \delta_{activation}$ (product of parameter and activation density) through zero-skipping of both parameters and activations

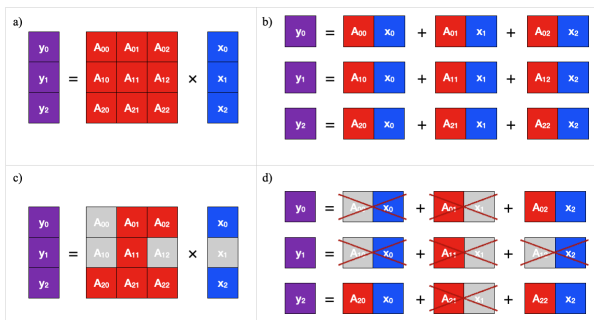


Figure 1: Sparsifying a matrix-vector product reduces necessary work.

a), b) - A dense matrix-vector product is expanded to highlight the individual multiply-accumulate operations.

c), d) - By sparsifying the matrix and vector entries (gray entries are set to 0), the work actually necessary to compute the result can be dramatically reduced.

¹ Density and sparsity are measured as the fraction of elements of a tensor that are nonzero and zero, respectively. Sparsity = 1 - Density.

The SPU fully supports dense operations while unlocking the potential savings in memory, latency, and energy from sparse operations with its dedicated sparse hardware data formats and sparse matrix-vector product instructions. The SPU's sparse-matrix data format stores only nonzero components of a given matrix. The sparse matrix-vector product operator exploits sparsity in both the matrix and the vector to skip over unnecessary operations (Figure 1).

Case Studies

We provide two examples where sparse models outperform their dense counterparts in terms of performance per parameter. Beyond these two examples from our own internal model development, there is a vast body of literature studying the efficacy of sparsity as a model compression tool.

Sparse Google Speech Commands Keyword Spotting

In Figure 2 below, we compare sparse and dense models across a wide range of model sizes, showing that sparse models outperform their dense counterparts at a given parameter budget. We trained our models on a real-time streaming version of the Google Speech Commands benchmark task, in which 10 keywords must be recognized (Up, Down, Left, Right, Stop, Go, On, Off, Yes, No). Sparse LSTMs outperform dense models in terms of accuracy per parameter count.

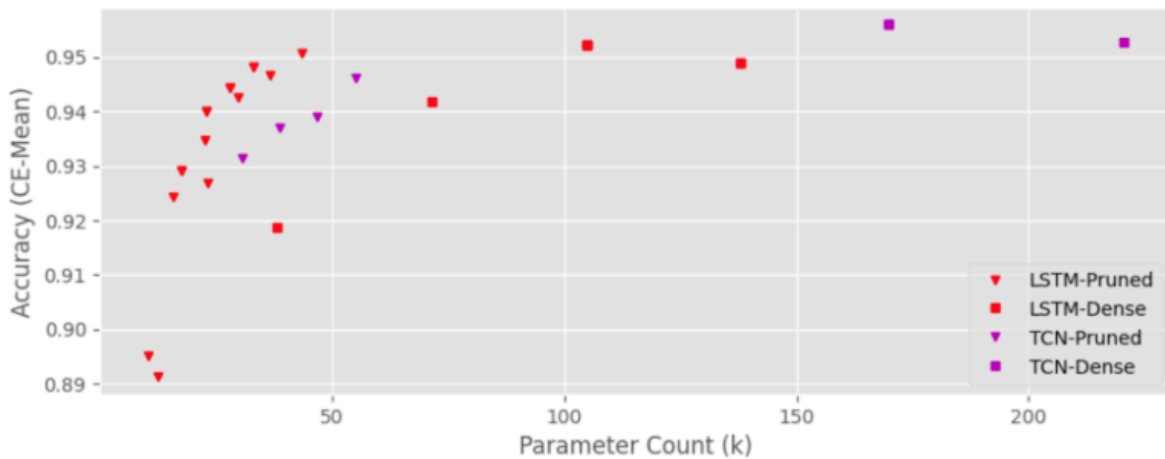


Figure 2: Accuracy on a streaming-version of the Google Speech Commands Task for various model sizes and architectures, with and without pruning. The pruned LSTM models (red triangles) achieve the highest accuracy at their given parameter count, achieving similar accuracy as dense models more than twice their size.

Sparse Speech Enhancement

When developing our low-latency AI Noise Reduction algorithms, we employ sparsity to compress large models to fit on SPU and run efficiently. We have found after 85% pruning and quantization (with quantization aware training), our compressed AINR algorithm matches the performance of a floating-point model, with a 27x lower memory footprint and compute budget.

In Table 1 below, we compare the SISDRi (Scale Invariant Signal to Distortion Ratio Improvement) of the dense, sparse, and sparse+quantized model. The SISDRi drops negligibly with both 85% pruning and quantization, making use of mixed precision int8 weights and int16 activations. The quantized model underwent Quantization Aware Training (QAT).

Table 1: SISDRi performance of dense, sparse, and sparse+quantized low-latency AI Noise Reduction algorithm in a cafe environment with speech babble background noise. Background SNR levels are sampled uniformly over the dB range -6 dB to +6 dB.

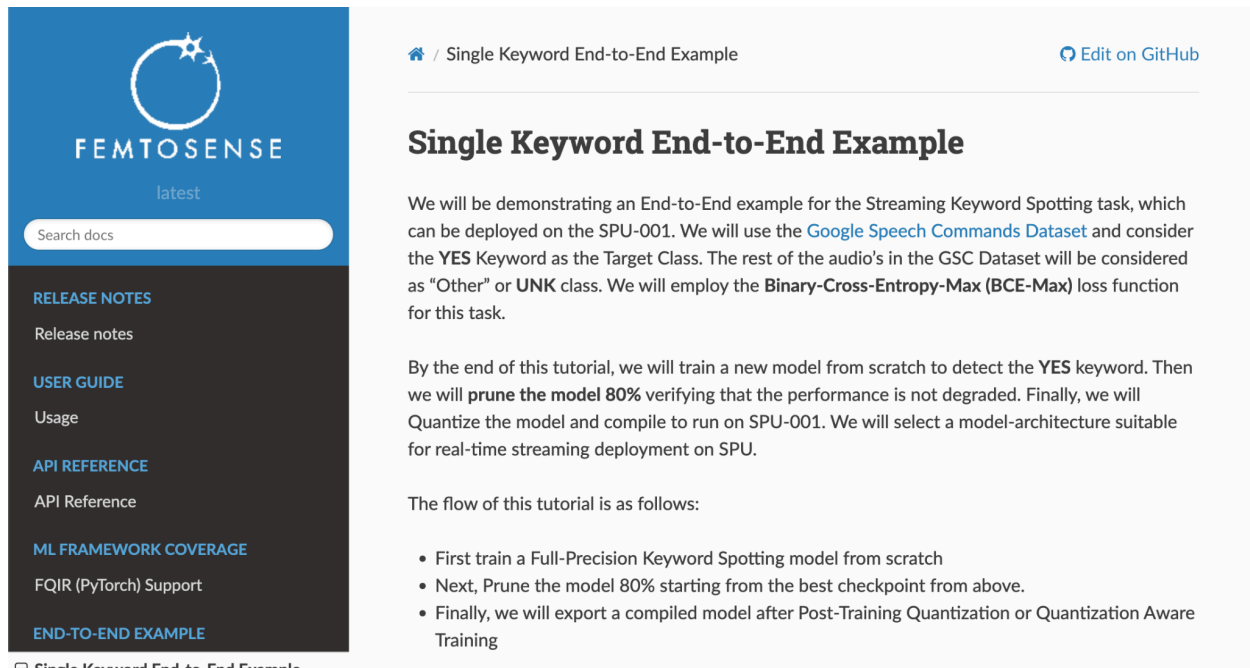
Model	Sparsity	Precision	Babble Background SISDRi
dense	0%	float32	8.09 dB
sparse	85%	float32	8.06 dB
sparse+quantized	85%	int8 weights int16 activations	7.86 dB

Technical Documentation and Tutorials

Below, we link to various documentation resources demonstrating how to deploy models to SPU using *fmot* and *femtocrux*. The documentation provides API examples and practical guidance for how to compress models with sparsity.

[Single Keyword End-to-End Example](#)

Learn how to train, prune, quantize, and deploy a single-keyword model. This is a great starting place and launching-off point for custom development for SPU.



The screenshot shows the Femtosense documentation website. On the left is a dark blue sidebar with the Femtosense logo and a search bar. The main content area is white and features the title 'Single Keyword End-to-End Example' with an 'Edit on GitHub' link. The text describes the task of training a model for keyword spotting on SPU-001, using the Google Speech Commands Dataset and BCE-Max loss. It outlines the process of training, pruning (80%), and quantizing the model for real-time deployment. A list of steps follows: training a full-precision model, pruning it, and finally exporting a compiled model after quantization.

fmot API Documentation

fmot is the PyTorch frontend tool to deploy models to SPU. Below are relevant *fmot* documentation pages covering pruning and quantization compression techniques, as well as providing example model architectures.

- Documentation on [parameter pruning](#)
- Documentation on [activation pruning](#)
- Documentation on [quantization and QAT](#)
- A collection of template [model architectures](#)

Change Log

Version	Release Date	Description
1.0	2024-05-12	Initial release