ESP32-WROOM COMPRESSION

Group #5

Dominik Kornak, Alp Berke Ardic, Damien Karpen, Anand Pudi

Introduction: Constraints of Edge AI on ESP32

- **Objective:** Efficiently deploy Convolutional Neural Networks (CNNs) on low-resource microcontrollers like the ESP32 WROOM.
- **Hardware Bottleneck:**
  - Although the 4 MB Flash memory meets the storage requirements, the system is dramatically bottlenecked by the ~312 KB DRAM capacity.
- **Infeasibility of Standard Models**:
  - As an example, the parameter count for a full precision ResNet 18 is around 46.8 million. The network weights require approximately 46.8 MB (at FP32 precision) of memory.

Our Approach: Aggressive Compression Pipeline

Goal: To drastically reduce parameters with the goal of fitting the model into DRAM while sacrificing only a small percentage of classification accuracy.

Key Optimization Methods:

1. Structured Pruning
2. Adaptive Average Pooling (AAP)
3. 8 bit Quantization
4. Low Level C Optimization

Pre-Deployment: Requirements

- Need a model for **CIFAR-10** dataset.
- **Memory Constraints** for ESP32-WROOM:
  - Flash: 4 MB, DRAM: 312 KB
- Initial Design Choice: ResNet-18
  - 11.69 million parameters - 46.8 MB in FP32
- ResNet-18 would need compression **in orders of magnitude**
  - Need for a **simpler** yet **effective** model

Pre-Deployment: Initial Model Specifications

- **Conv1:** 3 input channels, 32 filters, 3×3 kernel, stride 1, padding 1
- **Maxpool1:** 2×2
- **Conv2:** 32 input channels, 32 output channels, 3×3 kernel, stride 1, padding 1
- **Maxpool2:** 2×2
- **Flatten:** Converts 8×8×32 = 2048 features
- **FC1:** 2048 inputs, 32 outputs
- **FC2:** 32 inputs, 10 outputs (number of classes)

Pre-Deployment: Final Model Specifications

- **Conv1:** 3 input channels, 32 filters, 3×3 kernel, stride 1, padding 1
- **Maxpool1:** 2×2
- **Conv2:** 32 input channels, 64 output channels, 3×3 kernel, stride 1, padding 1
- **Maxpool2:** 2×2
- **AdaptiveAvgPool2d:** Output size 2×2
- **FC1:** 64 × 2 × 2 = 256 inputs, 128 outputs
- **FC2:** 128 inputs, 10 outputs (number of classes)
- **Accuracy (Quantized + Dequantized):** 72.63%

Model Compression: Structured Pruning

- Entire convolution layers are removed
  - Depending on the **importance** of layers
- **Importance Score:** sum of absolute values of all weights for a given layer

$$s_c^{(1)} = \sum_{i,j,u,v} \left| W_{c,i,u,v}^{(1)} \right|, \qquad c = 1, \ldots, C_1$$

- Channels/neurons with higher amplitude: More important
- **Pruning ratio:** 20% (initially it was 40%)

Model Compression: Adaptive Average Pooling

- **Problem:** Fully connected layers create the most number of weights
- **Need:** Compressing the fully connected layers aggressively
- **Solution:** Adaptive Average Pooling (2×2)
- Dividing each channel into 4 regions and averaging them
- Pruning + AAP: 94.84% parameter reduction
  - 545098 to 28576 parameters

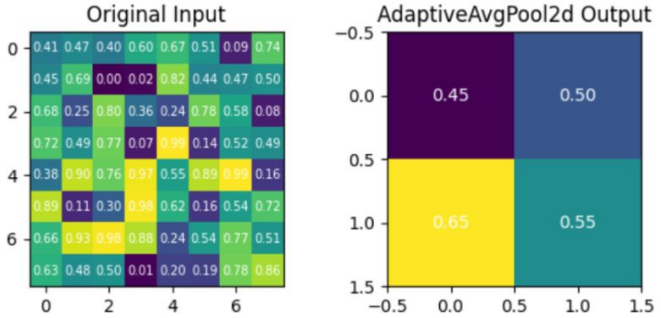


Fig. 1. Adaptive Average Pooling Example with Output Size (2,2)

https://mlmadesimple.in/demystifying-pooling-layers-in-cnns-maxpool-avgpool-and-more/

$$\text{AAP} : \mathbb{R}^{64\times8\times8} \rightarrow \mathbb{R}^{64\times2\times2}$$

Pre-Deployment: Fine Tuning

- Non-pruned, Non-quantized Accuracy: 76.03%
- Pruned Accuracy: 44.84%
- **Need:** Recovering performance of the pruned model
- **Solution:** Fine Tuning
  - Lower learning rate, 10 epochs
  - 72.63% final accuracy (pruned, quantized, de-quantized)

Pre-Deployment: Quantization

- **Problem:** FP32 model not fitting into DRAM memory
- **Need**: Quantization of weights into 8 bit integers (INT8)

- *x*: FP32 weight
- *b* = 8: number of bits
- $q_{\max}$: maximum representable integer value
- $q_{\max} = 2^{b-1} - 1 = 127$

$$\text{scale} = \frac{\max|x|}{q_{\max}} = \frac{\max|x|}{127}$$

- $$q = \text{round}\left(\frac{x}{\text{scale}}\right)$$

$$\hat{x} = \text{scale} \cdot q$$

Pre-Deployment: Comparisons

TABLE I
MODEL SIZE AND ACCURACY UNDER DIFFERENT QUANTIZATION, POOLING, AND PRUNING SCHEMES

Model and Quantization	Size (MB)	Accuracy
Original (FP32)	2.204	76.03%
Original (INT8)	0.551	75.12%
Avg. Pooling w/o Pruning (FP32)	0.329	73.96%
Avg. Pooling w/o Pruning (INT8)	0.082	73.05%
Avg. Pooling with Pruning (FP32)	0.207	73.12%
Avg. Pooling with Pruning (INT8)	0.052	72.63%

Pre-Deployment: Comparisons

TABLE II
STATISTICAL SUMMARY OF INFERENCE MEASUREMENTS (CPU)

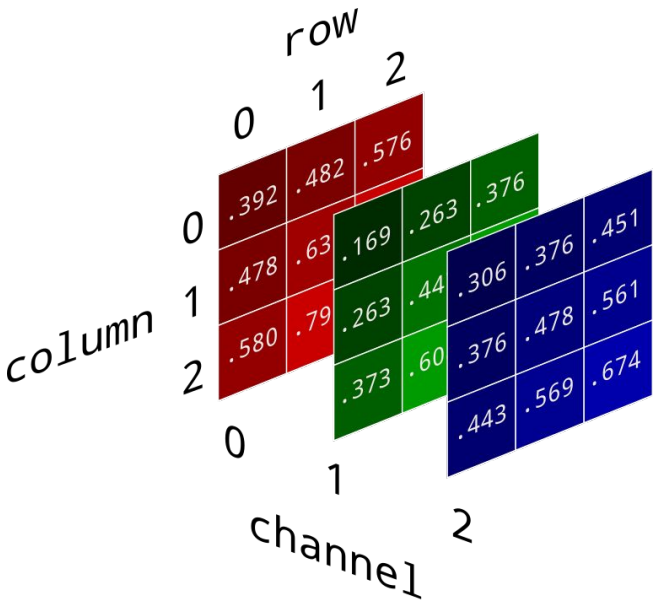
Metric	Value
One-shot Inference Time	0.419 ± 0.0801 ms
Energy per Inference (One-shot)	340.927 ± 37.10 μJ
Estimated CPU Power (One-shot)	0.700 ± 0.0735 W

Input Compression ESP32

Input Image Array

- 1080x1080 jpg -> 32x32x3 int8 array
- 32x32x3 = 3072 int8 pixels = 3072 bytes
- .h file 11KB -> .bin file 3KB
- 3.647x memory reduction

int_8 image[]={red channel, green channel, blue channel}



https://brohrer.github.io/images/image_processing/three_d_array.png

Weight Compression ESP32

Weights/Bias

- 96KB .h file -> 28KB .bin file
- 3.43x memory reduction
- Use size of each weight/bias arrays as pointers to respective section in .bin

.h file

```
…
int 8_t conv1bias_size=4
int8_t conv1bias[] = {8,6,
-9,-5}

int 8_t conv1weight_size=5
int8_t conv1weight[] = {3,1,
-2,-30,6}

int 8_t conv2bias_size=3
int8_t conv1bias[] = {5,1,4}
…
```

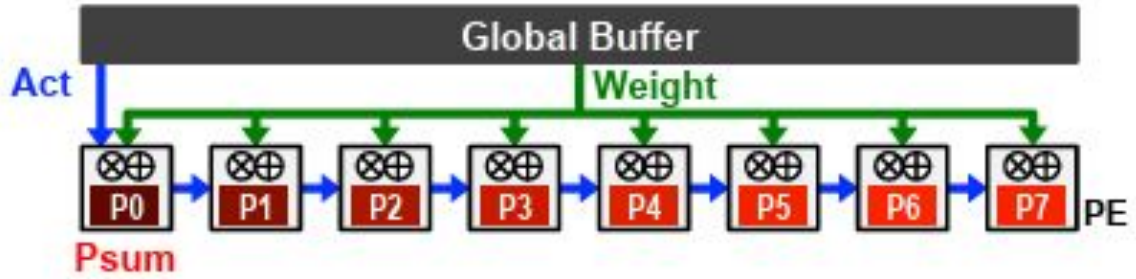
.bin file

8,6,-9,-5,3,
1,-2,-30,6,
5,1,4, …

Convolution Optimizations ESP32

Output Stationary Dataflow

- Keep each output element stationary in a buffer while its corresponding input activations and weights are streamed through to compute the final output pixel
- Best for ESP32 because minimizes buffer allocation
- Ex: Row stationary needs 3 buffers, 1 to store row of input, 1 for weight row, 1 temporary buffer, so needs 2 buffers more then OS

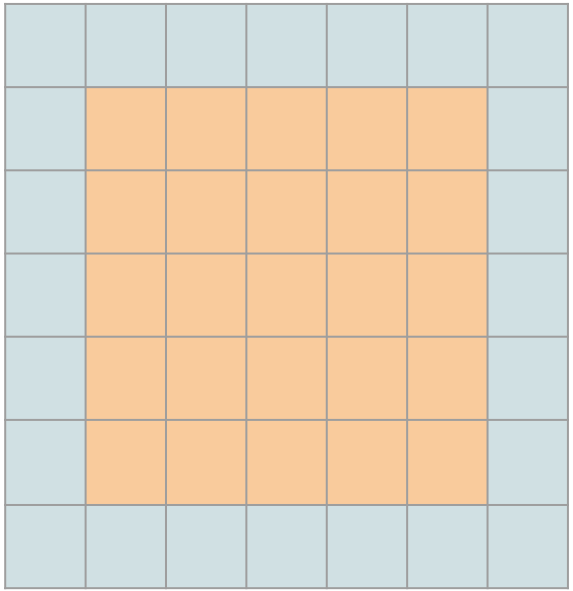


Efficient Processing of Deep Neural Networks: A Tutorial and Survey

Convolution Optimizations ESP32 Continued

- **Problem:** Padding utilizing unnecessary memory
- **Need:** No Boundary check for interior image
- **Solution:** Split the convolution into two regions:
  - **Interior region**: pixels where the kernel fully fits inside the ifmap.
  - **Boundary region**: pixels at the edges that require checks.
  - Example: 7x7 input image with 3x3 filter:
    No boundary check region(orange) is 5x5=25 pixels; boundary check region (blue) is perimeter of ifmap (24 pixels) so 51% of pixels don't need a boundary check

Ex: 7x7 ifmap



ESP32 Inference Metrics

Inference Time

- 313.33 ms inference time
- Conv1 takes up 55.1 ms
- Conv2 takes up 245.98 ms (4x Conv1)
- Convolution takes up 96.08% of the total inference time on ESP32

Accuracy

- Ran inference on ESP32 for 10 randomly selected JPG images chosen from Google Images (1 random image per class)
- 9/10 images detected correctly
- The 1 undetected image came second in Top-3

ESP32 Memory Summary

- Conv1 temporary buffer = 100KB
- All temp buffers=152.14KB; make up 48.75% of DRAM capacity
- For comparison, Input and Weights/Biases make up 9.93%
- 85.4% of runtime memory is used by temp buffers
- We have 138KB of DRAM/IRAM remaining

ESP32 Memory Summary (Initial vs Runtime)

	Total Memory(KB)	Remaining Runtime Memory(KB)
DRAM	312.10	133.92
IRAM	372.97	196.59

Shared IRAM/DRAM region approximately 128KB

Memory Simulation Methodology

Simulation Setup

- Tool: Energy consumption in the memory subsystem was evaluated with the CACTI 7.0 simulation tool
- Hardware Model: A 128 KB SRAM scratchpad was modeled with a 45 nm process technology
- Block Size: Set to 64 bytes (corresponding to 16 single precision floats)

SRAM Memory Characteristics

- Dynamic Read Energy: 0.0919 nJ per access
- Dynamic Write Energy: 0.1815 nJ per access
- Leakage Power: 65.41 mW
- Access Time: 0.73 ns

Parameter	Value
Technology Node	45 nm
Memory Size	128 KB
Area	0.44 mm^2

Convolution Energy Analysis

Layer-wise Energy Consumption

Layer	Physical accesses	Energy (μJ)
Conv1	43,200	8.23
Conv2	183,600	33.89

Key Findings

- Channel Depth Impact: Energy wise, Conv2 which has a smaller spatial resolution (16 *16 vs 32 * 32) is consuming ~4.1x more than Conv1
- Bottleneck: The main reason for the memory energy consumption to grow so much that the time for Conv2 to complete increased by ~4.

Power Consumption

Current Draw

Idle State:	**36.4 mA**
Active Inference:	**43.0 mA**
Delta (Δ):	**+6.6 mA**

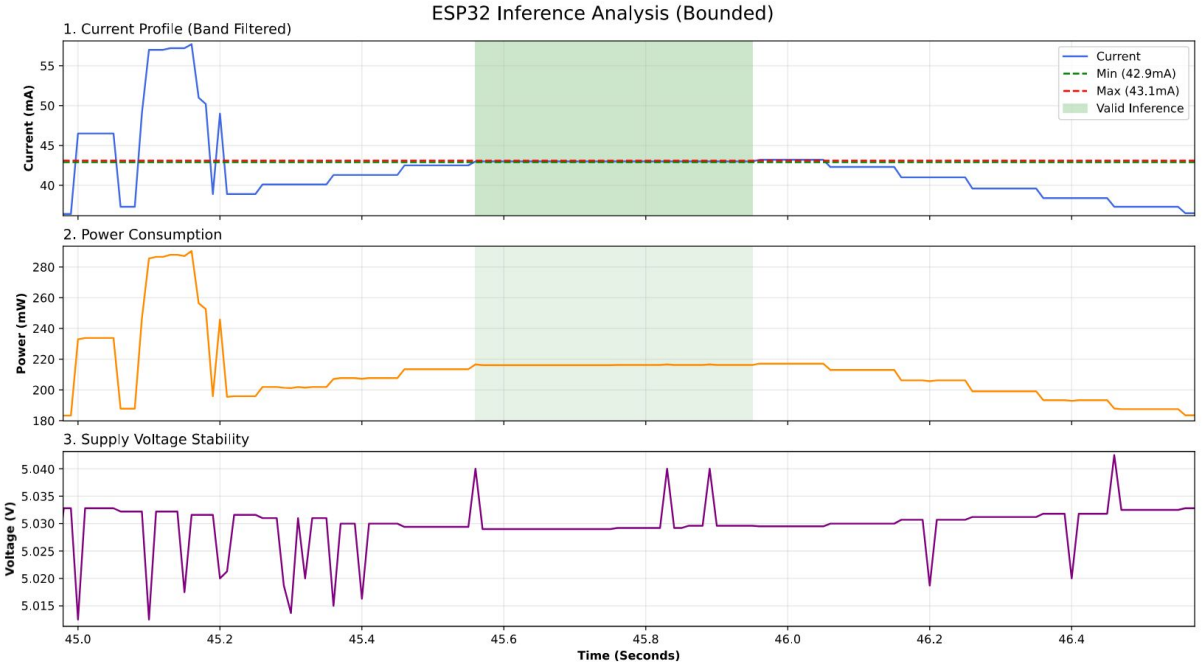
σ < 0.1 mA (CV < 0.23%)

Power

Idle:	**183.1 mW**
Active:	**216.3 mW**
Increase:	**+18.1%**

Energy

Duration:	**0.32 s**
Range:	**0.29 - 0.42 s**
Avg Energy:	**69.2 mJ**



$$Active(t) = \begin{cases} 1 & if\ I_{min} < I(t) < I_{max}\ \&\ t > t_{boot}\ \&\ \tau > \tau_{min} \\ 0 & otherwise \end{cases}$$

Battery Life Estimates

Battery Specifications

Nominal Voltage:	**5.0 V**
Capacity:	**1.0 Ah**
Total Energy:	**5.0 Wh**

$\eta_{conv} = 85\%$

$\eta_{discharge} = 90\%$

Usable: 3.83 Wh

Energy Breakdown

Per typical inference ($\tau = 0.32s$)

Continuous Inference

Operating Time
17.7 hours

Total Inferences
198,952

Inference Rate
3.12 Hz

Assumes $P_{active} = 216.3$ mW, $\tau = 0.32$s

Duty-Cycled Operation

1 inference per minute

Operating Time
20.9 hours

Total Inferences
1,252

Average Power
183.3 mW

Duty cycle D = 0.53%, dominated by idle power

Demo

Pick a class from CIFAR-10:

airplane, car, bird, cat, deer, dog, frog, horse, ship, truck