Karlsruher Institut für Technologie

Person Detection on Jetson Nano

Embedded Machine Learning Lab Challenge

Benedikt Engel, Mario Guijarro | September 13, 2026



Introduction & Motivation

The Goal:

- We want to detect **only persons**.
- Original Tiny Yolo v2 is a "generalist" that can detect 20 classes of objects (dogs, cars, etc.) $\Rightarrow$ it wastes resources extracting unnecessary features.
- We want to create a specialized **person-detection expert** to run on a **Jetson Nano** (limited computational resources).

The Challenge:

1. **Speed:** Real-Time Processing at high rate ($>$ 20 FPS).
2. **Accuracy:** Don't lose ability to spot people, i.e. maintain sufficient AP.

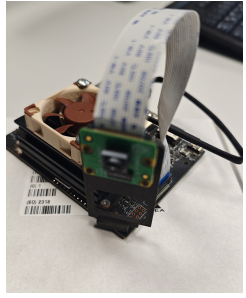


Figure: Jetson Nano & Camera Module

Pipeline Overview

Our Strategy:

1. Batch Norm Fusion
2. Reduce number of output classes ($20 \rightarrow 1$)
3. Retraining with Data Augmentation on persons only
4. Structured Pruning
5. Quantization
6. Export model to ONNX Format
7. Run with NVIDIA TensorRT

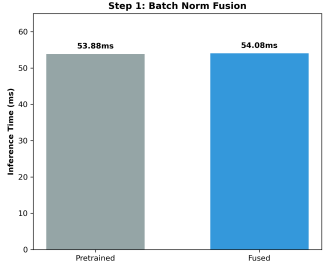
Design Objectives:

- **Performance Constraint:** Real-time inference on Jetson Nano
$$f_{fps} \geq 20\,\text{Hz}$$
- **Optimization Target:** Maximize Person-class Average Precision $\max(AP_{person})$
- **Trade-off Criteria:** balance between inference time Δt_{lat} and AP.
- **Objective Function:** Multi-part Yolo Loss L_{total} and AP_{person}

Steps 1 & 2: Preparation

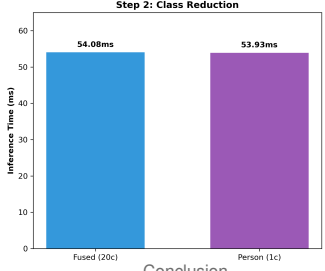
1. Fuse Batch Norm:

- Merge the Batch Norm layer into the corresponding Convolutional layer.
- **Result:** No visible performance gain



2. Person Class Only:

- Number of output filters decreased from 125 to 30.
- *Result:* Almost no speed difference.
- **Initial** AP_{person}**:** 0.60; 10FPS demo



Step 3a: Fine-Tuning Strategy

Transition: 20-Class Generalist → 1-Class Specialist

- **Knowledge Preservation:** Re-using pre-trained weights from VOC backbone to maintain feature extraction capabilities.
- **Differential Learning Rates w.r.t. AP:**
  - **Backbone (10^{-6}):** Minimal updates to frozen features to prevent *Catastrophic Forgetting*.
  - **Detection Head (10^{-4}):** High rate for the re-initialized Layer 9 to adapt to the new person-only objective.
- **Head vs Backbone:** Trained Head only for 15 Epochs ⇒ 30 Epochs Backbone and Head
- Early Stopping if $|AP_{Epoch} - AP_{Epoch-1}| \leq 10^{-3}$

Key Finding

Unfreezing with differential rates outperformed Head-only training.

Step 3b: Data Augmentation Pipeline

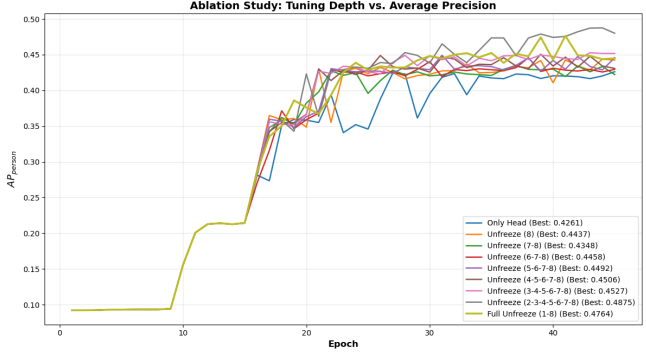
Goal: Counteracting small sample size & sensor variability

- **Geometric Invariance (Spatial Awareness):**
  - *Horizontal Flip*: Direction independence.
  - *BBox-Safe Crop*: Scale invariance and handling partial occlusions.
- **Environmental Robustness (Lighting):**
  - *Photometric Jittering*: Randomized brightness, contrast, and HSV shifts for changing light conditions.
- **Edge Hardware Adaptation:**
  - *Gaussian Noise*: Emulating low-cost sensor characteristics of the Jetson Nano camera modules.

Pre-Processing: ImageNet normalization for stable gradient flow.

Step 3c: Results

- **Key Result:** *Unfreezing Multiple layers (2-3-4-5-6-7-8-9)* outperformed "Head-only" training, achieving **0.4876 AP** on **Augmented Data** by allowing the entire feature hierarchy to adapt.



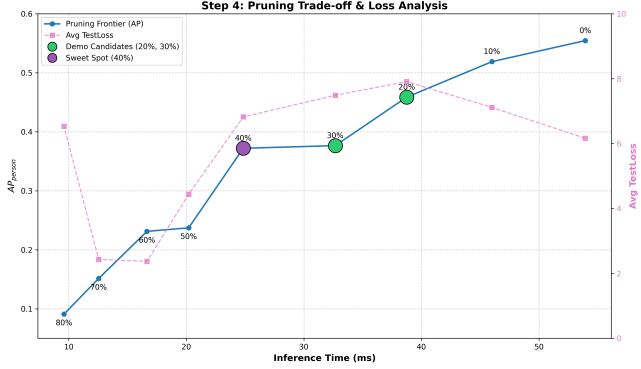
Step 4: Structured Pruning

Strategy

- **Method:** Iterative *L1-Structured Pruning* & retraining
- **Range:** 10% to 80% Sparsity.
- $|AP_{Epoch} - AP_{Epoch-1}| \leq 10^{-3}$

Observation

- **The "Cliff":** Accuracy stable up to 40%; crashes afterwards; model loses "understanding" of persons.
- **Efficiency:** Linear reduction in MACs $\rightarrow$ smaller size (60MB to 22MB)
- **Sweet Spot:** 40% offers the best theoretical balance (0.37 AP).



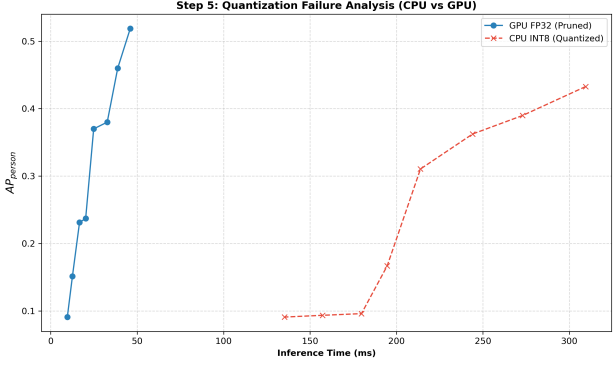
Step 5: INT8 Quantization Analysis

The Challenge

- **Hardware Gap:** The Jetson Nano (Maxwell GPU) lacks native INT8 acceleration (No Tensor Cores/DP4A).
- **Fallback:** CPU-based INT8 inference.

Results & Trade-offs

- **Accuracy Cliff:** Cliff stays at 40% but faster AP drop-off.
- **The Verdict:** INT8 on CPU is a major bottleneck ($\sim$ 4 FPS; 40% pruned).



Step 6: ONNX Runtime & 16 bit quantization

Optimization Strategy:

- **Export:** Sweet-Spot (40% sparsity) converted to ONNX format.
- **Graph Optimizations:** ONNX performs *Constant Folding* and *Operator Fusion*
- **Static Memory:** Pre-allocated buffers reduce overhead during CUDA execution.
- **FP16 quantization** to reduce overhead

Final Performance:

- **Latency:** Reduced to **12 ms** ($\approx$ 83 FPS).
- **Comparison:** Efficiency increased 4.5x compared to the native PyTorch GPU implementation.

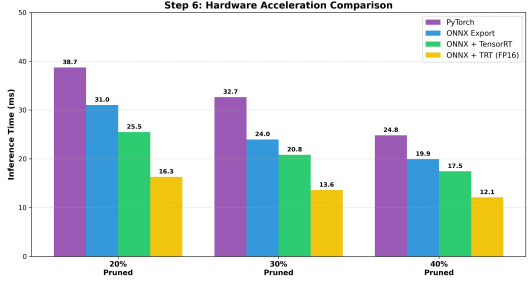


Figure: Inference Speed PyTorch vs. ONNX Runtime

Final Summary of Results

Pipeline Stage	Model	Accuracy (AP)	Latency	Jitter	FPS	Size	#Params	device
Batch Norm Fusion & Person Only	Baseline (20 classes)	0.60	53.88 ms	0.4198 ms	18.56	60.55 MB	15861773	CUDA
	Fused (20 classes)	0.60	54.08 ms	1.45 ms	18.49	60.5 MB	15858717	CUDA
	Person-Only (1 class)	0.55	53.93 ms	0.8164 ms	18.54	60.13 MB	15761342	CUDA
Retrain & Augmentation	Unfreeze Head	0.53	53.9 ms	0.8109 ms	18.55	60.13 MB	15761342	CUDA
	Unfreeze 2-3-4-5-6-7-8-9	0.56	54.06 ms	1.0029 ms	18.5	60.13 MB	15761342	CUDA
L1 Structured Pruning	20% Pruned	0.46	38.74 ms	0.738 ms	26.62	38.62 MB	10121932	CUDA
	30% Pruned	0.38	32.67 ms	0.7152 ms	32.8	32.09 MB	8409824	CUDA
	40% Pruned	0.37	28.27 ms	0.8121 ms	35.37	26.07 MB	6832970	CUDA
Quantization	40% Pruned int8	0.31	213.87 ms	14.46 ms	4.68	5.52 MB	5724533	CPU
ONNX Export	20% Pruned	0.46	25.48 ms	1.0437 ms	39.25	38.62 MB	10121932	CUDA
	30% Pruned	0.38	20.84 ms	0.6732 ms	47.99	29.64 MB	8409824	CUDA
	40% Pruned	0.37	17.47 ms	0.5101 ms	57.24	21.85 MB	6832970	CUDA
	70% Pruned	0.15	8.72 ms	0.2448 ms	114.7	5.57 MB	-	CUDA
ONNX & TensorRT & FP16	20% Pruned	0.46	16.26 ms	0.589	61.5	38.62 MB	10121932	CUDA
	30% Pruned	0.38	13.57 ms	0.462	73.7	29.64 MB	8409824	CUDA
	40% Pruned	0.37	12.06 ms	3.184	82.92	21.85 MB	6832970	CUDA

Conclusion & Future Work

Project Achievements:

- ✓**Performance:** Exceeded real-time requirement by **>336%** (83 FPS vs. 19 FPS).
- ✓**Efficiency:** Reduced inference latency from 54 ms (Baseline PyTorch) to $\approx$ 12 ms (FP16 40% ONNX)
- ✓**Accuracy:** Successful adaptation from generalist to person-only specialist.

Future Work & Optimization Potential:

- **Quantization Aware Training:** Recovering accuracy lost during post-training quantization.
- **Data Diversification:** Expanding the dataset to improve robustness.

Thank you for your attention!

Live Detection Demo

We will showcase 3 versions to demonstrate the trade-off:

- **1. Default Baseline:** Slower but highly accurate ($\sim$ 10 FPS).
- **2. High Accuracy Focus (20% Pruned):** Prioritizes detection reliability ($\sim$ 25 FPS).
- **3. High Speed Focus (40% Pruned FP16):** Max system performance ($\sim$ 45 FPS).

**Note: While pure inference runs at 83 FPS, end-to-end system speed is bottlenecked by camera I/O and drawing bounding boxes.*

References I

1 Redmon, J. (2017). YOLO9000. (The paper explaining how YOLO works).

2 PyTorch Documentation.

3 ONNX Runtime Docs.

4 Lab Manual (Task Description).

5 Average Precision definitio: `https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173`

Appendix

Appendix: Additional Plots

