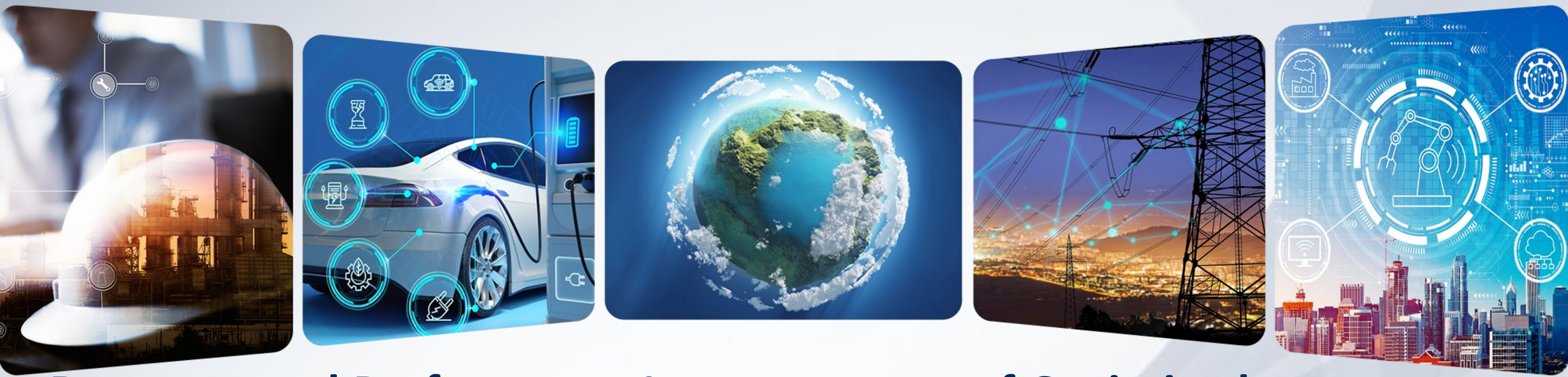


Resource and Performance Improvements of Optimized Convolutional Neural Networks for FPGA Implementations of Automatic Modulation Recognition

Joshua Rothe, Dr. Haya Shajaiah – Johns Hopkins University, USA

59th Annual Conference on Information Science and Systems

IEEE
Advancing Technology for Humanity

Outline

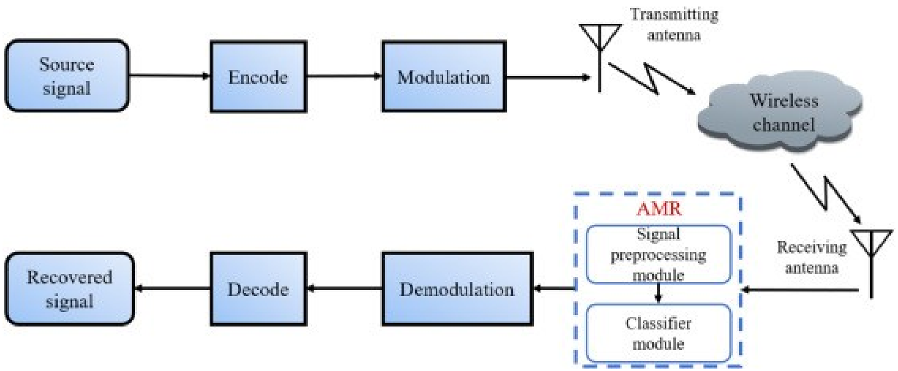
- Introduction
- Other/Prior Work
- Motivation
- Proposed Work
- Methodology
- Results
- Conclusion/Future Work

Introduction

Background

- Automatic Modulation Recognition (Detection) – detects the modulation of a radio frequency (RF) signal

- Two stage system - feature extraction and classification

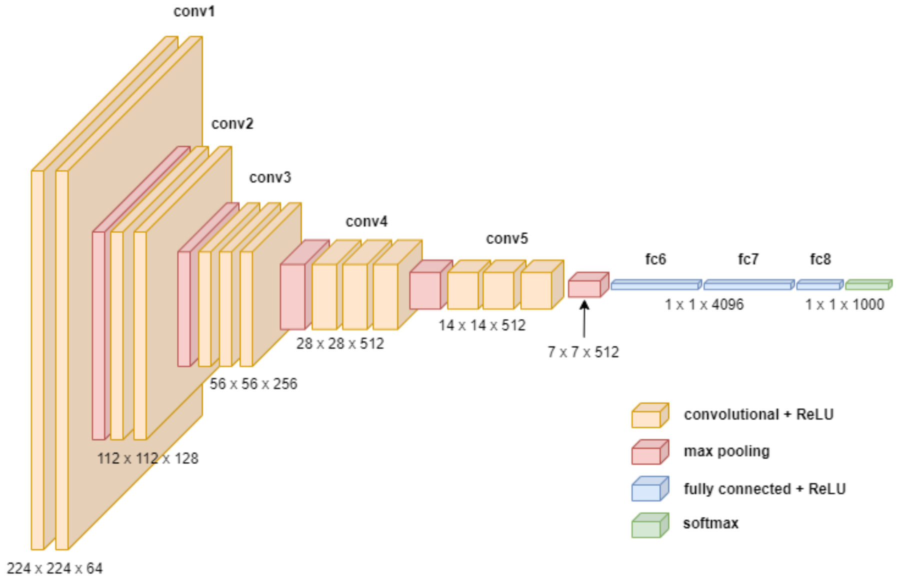


Juan Wang, Guan Gui, Hikmet Sari, Generalized automatic modulation recognition method based on distributed learning in the presence of data mismatch problem, Physical Communication, Volume 48, 2021, 101428, ISSN 1874-4907, https://doi.org/10.1016/j.phycom.2021.101428.

Introduction

Convolutional Neural Networks

- Convolutional layer – produces a feature map
- Rectified Linear Unit layer – introduces non-linear properties to the system, helps detect complex features
- Max pooling layer – reduces special dimensions of a layer to reduce computational load, make model more resilient to distortions/noise
- Fully connected layer – takes the data from previous layers and classifies them

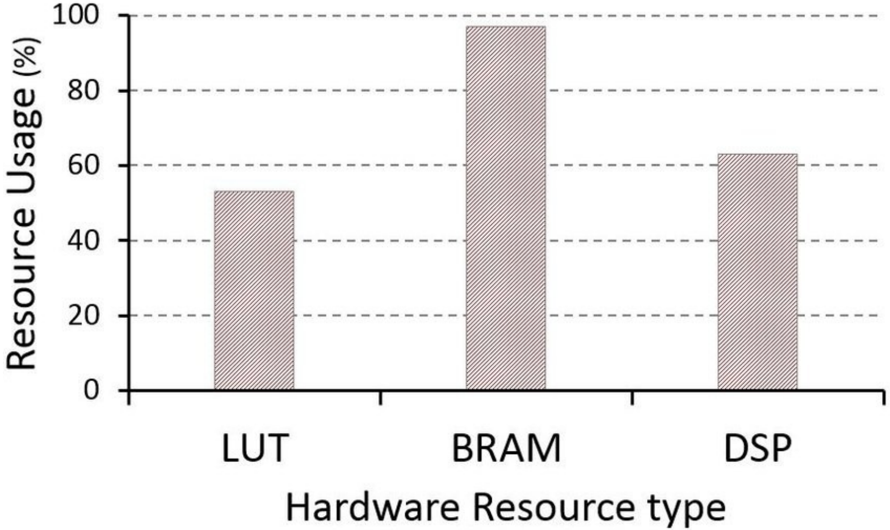


K. Leung, “How to Easily Draw Neural Network Architecture Diagrams | TDS Archive,” Medium, Aug. 23, 2021. https://medium.com/data-science/how-to-easily-draw-neural-network-architecture-diagrams-a6b6138ed875.

Introduction

Problem

- CNNs are resource intensive
- Lowering resource utilization while maintaining accuracy is critical
- Example of utilization on Xilinx PYNQ board – TinyCNN, modular CNN accelerator for Embedded FPGAs



Jahanshahi, Ali & Zamani Sabzi, Hadi. (2020). TinyCNN: A Tiny Modular CNN Accelerator for Embedded FPGA.

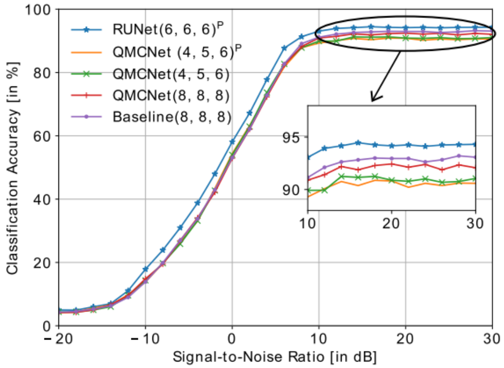
Other Work

"Automatic Modulation Recognition: An FPGA Implementation"

- Quantization
- Pruning
- Rectified Linear Units (ReLUs)
- Synthesizes using FINN in Vivado

Algorithm 1 Iterative Pruning Based Training Method

Input: Total number of pruning iteration $k = 50$, Pruning Amount per iteration $\delta = 0.2$, Total number of training Epochs = $E = 30$, Minimum Accuracy = 58%
Output: Pruned model
for *Pruning Iterations=1:k* **do**
  Best Accuracy = 0
  for *Epochs =1:E* **do**
    Train the Model and find Test Accuracy
    if *Test Accuracy* $\geq$ *Best Accuracy* **then**
      save model Weights
      Best Accuracy = Test Accuracy
    end
    if *Test Accuracy* $<$ *Best Accuracy for 10 consecutive Epochs* **then**
      Stop Training
    end
  end
  if *Test Accuracy* $<$ *Minimum Accuracy* **then**
    Stop Pruning Iteration
  end
  Prune the model weight to δ
end



S. Kumar, R. Mahapatra and S. Anurag, "Automatic Modulation Recognition: An FPGA Implementation," IEEE Communications Letters, vol. 26, no. 9, pp. 2062-2066, 2022.

Prior Work

Quantization and Pruning of Convolutional Neural Networks for Efficient FPGA Implementation

- Overlaps with current work – CNNs of various model sizes were created and trained
- Performance evaluated on GPU using PyTorch and CUDA
- Implemented varying levels of Quantization and Pruning, as well as ReLU and Max Pooling layers, and varying model sizes

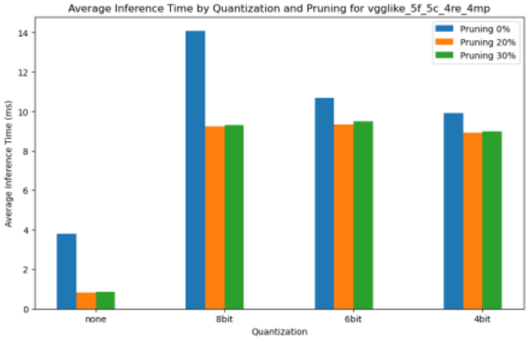


Figure 11. VGGLike_5f_5c_4re_4mp Inference Time (ms) at Various Quantization and Pruning Rates.

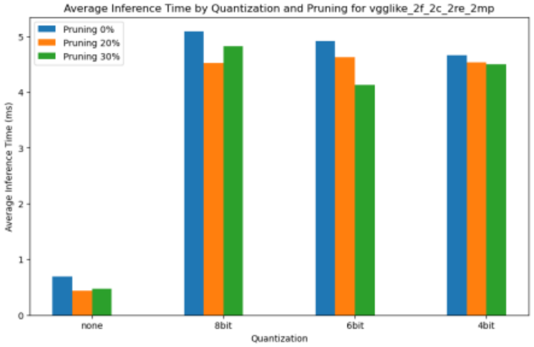


Figure 12. VGGLike_2f_2c_2re_2mp Inference Time (ms) at Various Quantization and Pruning Rates.

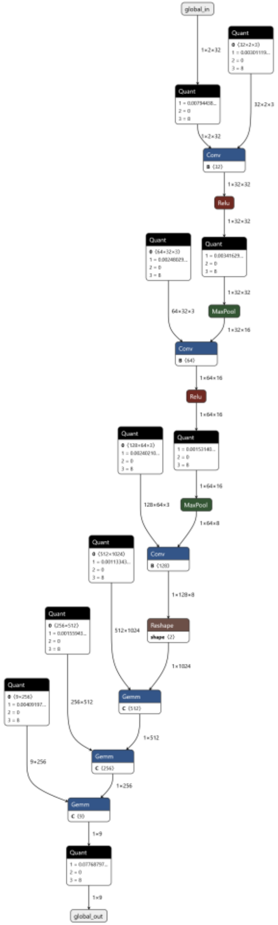
J. A. Rothe, Quantization and Pruning of Convolutional Neural Networks for Efficient FPGA Implementation of Digital Modulation Detection Firmware,
Johns Hopkins University, 2024.
https://jscholarship.library.jhu.edu/handle/1774.2/69928

Motivation

- Various methodologies can be used to reduce utilization, combining several methods may be particularly useful
- Other works usually focus on one or two, controlling the other variables for more noteworthy results (e.g. large model sizes, so quantization is more impactful)
- Goal is to combine all methodologies and analyze what works best

Proposed Work

- Combination of Methodologies
  - **Bit Quantization**
  - **Pruning**
  - **Shrinking Model Architectures**
  - ReLU Layers
  - Max Pooling Layers
  - One-Dimensional CNNs
  - Data Preprocessing (Normalized I and Q values)
- Benchmarking
  - Done post-synthesis using Xilinx Vivado with the help of the FINN library

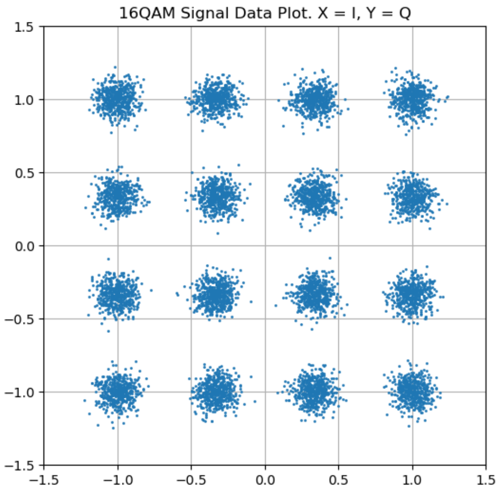



```yaml
- name: vgglike_3f_3c_2re_2mp
  layers:
    - conv1: {in_channels: 2, out_channels: 32, kernel_size: 3, stride: 1, padding: 1}
    - relu1: {return_quant_tensor: True}
    - maxpool1: {kernel_size: 2}
    - conv2: {in_channels: 32, out_channels: 64, kernel_size: 3, stride: 1, padding: 1}
    - relu2: {return_quant_tensor: True}
    - maxpool2: {kernel_size: 2}
    - conv3: {in_channels: 64, out_channels: 128, kernel_size: 3, stride: 1, padding: 1}
    - fc1: {in_features: 1024, out_features: 512}
    - fc2: {in_features: 512, out_features: 256}
    - fc3: {in_features: 256, out_features: 9}
  quantization:
    - bits: 32
      learning_rate: 0.0005
      weight_decay: 0.0001
      loss_threshold: 0.003
      num_epochs: 500
    - bits: 8
      learning_rate: 0.0003
      weight_decay: 0.0001
      loss_threshold: 0.003
      num_epochs: 500
    - bits: 6
      learning_rate: 0.0003
      weight_decay: 0.0001
      loss_threshold: 0.003
      num_epochs: 500
    - bits: 4
      learning_rate: 0.0002
      weight_decay: 0.0001
      loss_threshold: 0.006
      num_epochs: 500
  pruning:
    - percentage: 0
      prune_loss_threshold: 0.008
    - percentage: 0.1
      prune_loss_threshold: 0.008
    - percentage: 0.2
      prune_loss_threshold: 0.008
    - percentage: 0.3
      prune_loss_threshold: 0.012
```

Methodology

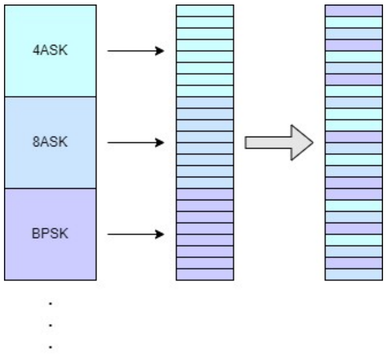
Signal Generation

- Generate normalized I and Q signal pairs for various modulation types (4ASK, 8ASK, BPSK, QPSK, 8PSK, 16PSK, 8QAM, 16QAM, 32QAM)
- Associate pairs together, label them, split them into sets of 32 and shuffle
- After generating all data types, shuffle the sets again amongst themselves
- Different generated datasets for training and evaluation



Algorithm I: CNN Dataset Generation
Inputs: Noise values N_a, N_p, chunk size c_s, quantity V_d.
Output: A data frame D_f consisting of V_d/c_s labeled, chunked datasets.

1. Initialize qty D_m list of siggen functions.
2. $i = 0$.
3. **while** $i \le D_m$ **do**:
4. Generate V_d quantity I, Q pairs and shuffle.
5. Create an empty list L_c.
6. $n = 0$.
7. **for** $j \leftarrow j + c_s$ **do**:
8. Append qty c_s I and Q pairs to entry n in L_c.
9. Label entry n with associated label D_m.
10. $n = n + 1$.
11. **end for**
12. $i = i + 1$.
13. **end for**
14. Combine L_{c_i} for $i = 1$ thru D_m.
15. Convert L_{c_i} into dataframe D_f.
16. Return D_f.



Methodology

Model Generation

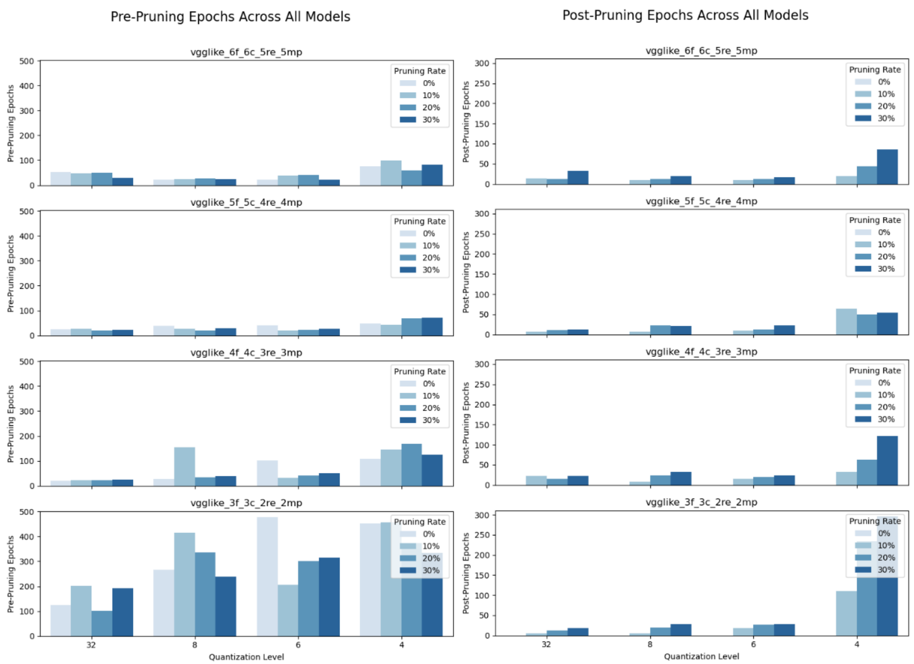
- Create 1-D CNNs of various layer counts
- Train them using the training dataset with appropriate parameters (learning rate, loss function threshold)
- Model trains repeatedly over the dataset until a desired loss function is achieved
- If a loss function never converges low enough, the model is retrained (would result in poor accuracy)

```
for i, layer_config in enumerate(config['layers']):
    layer_type = list(layer_config.keys())[0]
    layer_params = layer_config[layer_type]
    if layer_type.startswith('conv'):
        layer = QuantConv1d(
            in_channels=layer_params['in_channels'],
            out_channels=layer_params['out_channels'],
            kernel_size=layer_params['kernel_size'],
            stride=layer_params['stride'],
            padding=layer_params['padding'],
            weight_quant_type=QuantType.INT,
            weight_bit_width=quantization_bits
        )
    elif layer_type.startswith('fc'):
```

Results

Performance (Training)

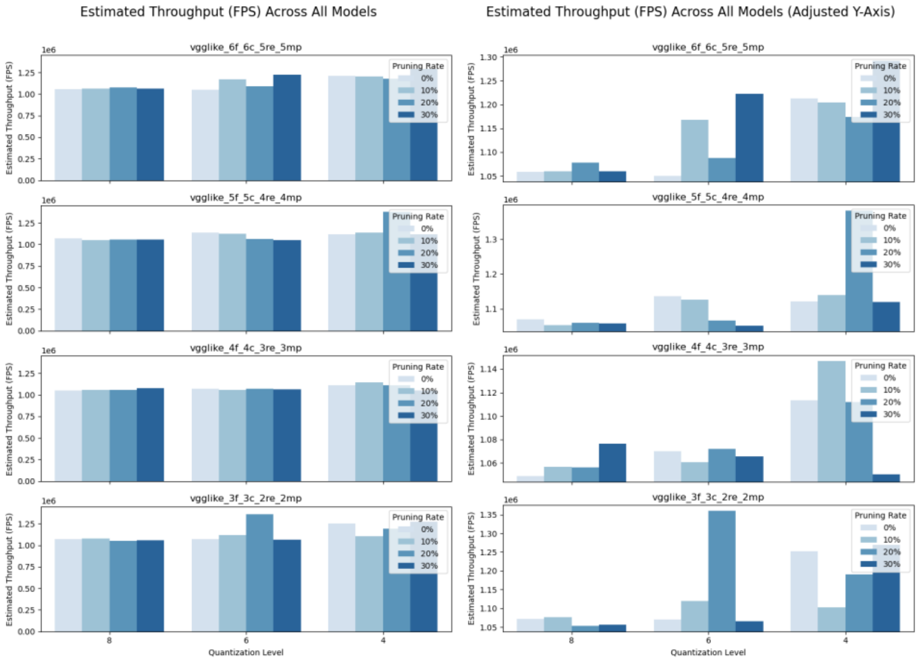
- Training
  - Smaller or more optimized models are harder to train and require more tweaking
  - Retraining and multiple attempts needed for most aggressively shrunk models
  - Generally, lower learning rates and higher loss thresholds for smaller models, especially for post-pruning training



Results

Performance (Evaluation)

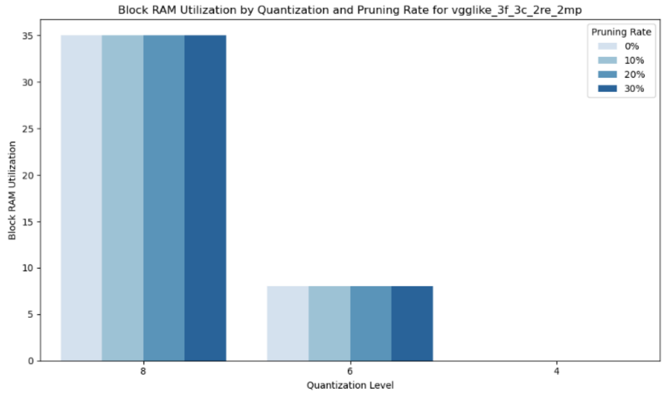
- Throughput
  - For FINN implementations, you can set target throughput and synthesizer will try to match
  - Quantization and pruning seem to apply after the synthesizer optimizes for the target performance, as gains above target are seen
  - Synthesizer optimizes for utilization – thus, performance gains could be taken away to favor utilization gains

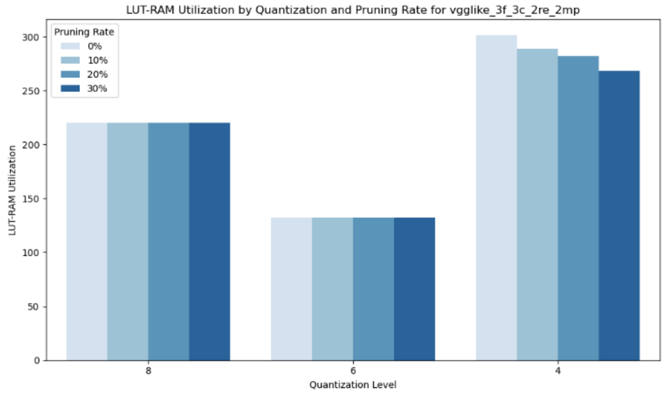


Results

Performance (Utilization)

- BRAM and LUT-RAM
  - Synthesizer optimizes for utilization as well as power, pushes storage from BRAM onto LUT-RAM when possible
  - Only takes advantage of pruning at the smallest quantization rate – inconsistent, but shows the synthesizer does recognize pruning
- Model size did not affect results here
  - All model sizes were the same graph – due to “folding”
  - FINN library will store weights and values and load them into the FPGA each clock cycle, prioritizing utilization





Results

Performance (Increased Target Throughput)

- Increased Clock Speed
  - Synthesizer could not go over ~120 MHz
- Increased Target Throughput
  - Gains from smaller models begin to become apparent, synthesizer will still prioritize utilization

TABLE I. 8-BIT QUANTIZATION, 20% PRUNING UTILIZATION

Model	Metrics				
	LUTs	*FFs*	*BRAM*	*Carry*	*Throughput*
VGGlike_6	9,212	23,582	32	1,168	2887202.762
VGGlike_5	9,267	23,728	32	1,165	3089509.262
VGGlike_4	9,058	23,469	32	1,161	3071404.000
VGGlike_3	9,131	23,578	29	1,171	3023267.063

Conclusion

Design Considerations

- For any design, utilization is balanced against performance and accuracy
- FINN library and Vivado synthesizer will prioritize utilization while meeting (designer-defined) performance goals. Workflow is to set performance goals FIRST, synthesizer handles resource optimization after
- Quantization gives best gains – 20-30% drop in utilization with 2-bit removal
- Pruning benefits minimal, best at lower quantization values
- Aggressive pruning combined with aggressive quantization allowed all BRAM usage to move to LUT-RAM – conserving valuable resources (and possibly increasing performance)

Backup Slides

Other Work

Ristretto

- Varying quantization values of weights per each layer, strove to find the "optimal" quantization value for each weight in a model
- Automated this search and ran on GPUs with CUDA
- Combined with ReLU layers and Max Pooling layers for greater utilization savings

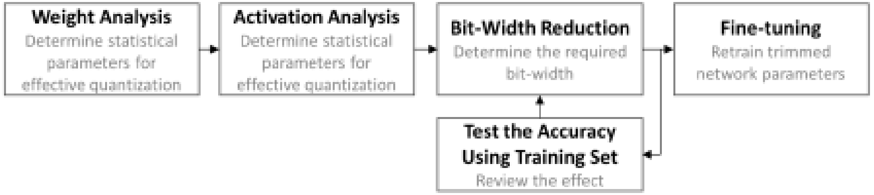


Figure 9.1. Network approximation flow with Ristretto.

P. M. Gysel, Ristretto: Hardware-Oriented Approximation of Convolutional Neural Networks, University of California Davis, 2016.

Other Work

Quantized Deep Neural Networks for Automatic Modulation Recognition

- Similar to current work – used FINN library to implement quantified CNNs onto Xilinx FPGAs for AMR

- Evaluated different quantization levels

- Future work – wanted to combine this with other quantization techniques or pruning

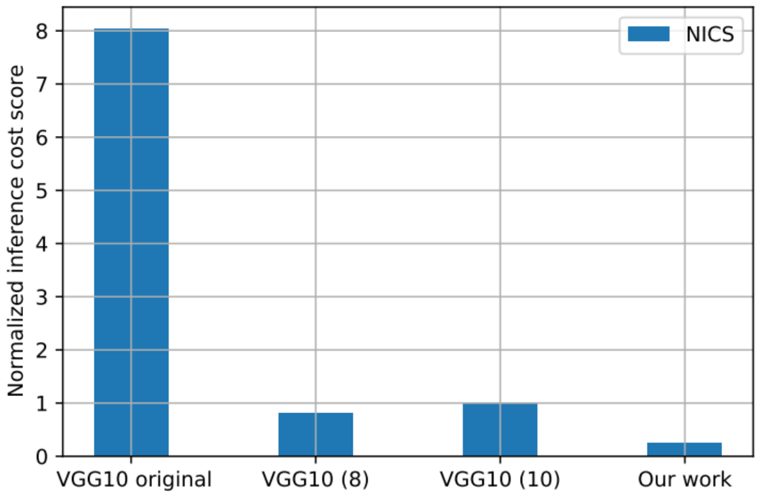


Figure 9. Comparison of the quantized VGG10 1D-CNN model versus the non-quantized.

D. Góez, P. Soto, S. Latré, N. Gaviria and M. Camelo, "A Methodology to Design Quantized Deep Neural Networks for Automatic Modulation Recognition," Algorithms, vol. 15, p. 441, 2022.