“DeepMaker - Deep Learning Accelerator on Commercial Programmable Devices”
Masoud Daneshtalab – Mälardalen University
Sweden Area Group – June 17, 2021
tinyML Talks Sponsors

Additional Sponsorships available – contact Olga@tinyML.org for info
Arm: The Software and Hardware Foundation for tinyML

Stay Connected
@ArmSoftwareDevelopers
@ArmSoftwareDev

Resources: developer.arm.com/solutions/machine-learning-on-arm
WE USE AI TO MAKE OTHER AI FASTER, SMALLER AND MORE POWER EFFICIENT

Automatically compress SOTA models like MobileNet to <200KB with little to no drop in accuracy for inference on resource-limited MCUs

Reduce model optimization trial & error from weeks to days using Deeplite's design space exploration

Deploy more models to your device without sacrificing performance or battery life with our easy-to-use software

BECOME BETA USER bit.ly/testdeeplite
TinyML for all developers

- **Dataset**
  - Acquire valuable training data securely

- **Impulse**
  - Enrich data and train ML algorithms

- **Test**
  - Test impulse with real-time device data flows

- **Edge Device**
  - Real sensors in real time
  - Open source SDK
  - Embedded and edge compute deployment options

- **C++ library**
- **Arduino library**
- **WebAssembly**

[www.edgeimpulse.com](http://www.edgeimpulse.com)
The Eye in IoT

Edge AI Visual Sensors

- CMOS Imaging Sensor
  - Ultra Low power CMOS imager
  - AI + IR capable

- Computer Vision Algorithms
- IoT System on Chip
  - Machine Learning edge computing silicon
  - <1mW always-on power consumption
  - Computer Vision hardware accelerators

- Machine Learning algorithm
- <1MB memory footprint
- Microcontrollers computing power
- Trained algorithm
- Processing of low-res images
- Human detection and other classifiers
Enabling the next generation of **Sensor and Hearable products** to process rich data with energy efficiency
LatentAI

Adaptive AI for the Intelligent Edge

Latentai.com
Maxim Integrated: Enabling Edge Intelligence

**Advanced AI Acceleration IC**

The new MAX78000 implements AI inferences at low energy levels, enabling complex audio and video inferencing to run on small batteries. Now the edge can see and hear like never before.

[www.maximintegrated.com/MAX78000](http://www.maximintegrated.com/MAX78000)

**Low Power Cortex M4 Micros**

Large (3MB flash + 1MB SRAM) and small (256KB flash + 96KB SRAM, 1.6mm x 1.6mm) Cortex M4 microcontrollers enable algorithms and neural networks to run at wearable power levels.

[www.maximintegrated.com/microcontrollers](http://www.maximintegrated.com/microcontrollers)

**Sensors and Signal Conditioning**

Health sensors measure PPG and ECG signals critical to understanding vital signs. Signal chain products enable measuring even the most sensitive signals.

[www.maximintegrated.com/sensors](http://www.maximintegrated.com/sensors)
Qeexo AutoML
 Automated Machine Learning Platform that builds tinyML solutions for the Edge using sensor data

**Key Features**

- Supports 17 ML methods:
  - Multi-class algorithms: GBM, XGBoost, Random Forest, Logistic Regression, Gaussian Naive Bayes, Decision Tree, Polynomial SVM, RBF SVM, SVM, CNN, RNN, CRNN, ANN
  - Single-class algorithms: Local Outlier Factor, One Class SVM, One Class Random Forest, Isolation Forest

- Labels, records, validates, and visualizes time-series sensor data

- On-device inference optimized for low latency, low power consumption, and small memory footprint applications

- Supports Arm® Cortex™-M0 to M4 class MCUs

**End-to-End Machine Learning Platform**

For more information, visit: www.qeexo.com

**Target Markets/Applications**

- Industrial Predictive Maintenance
- Smart Home
- Wearables
- Automotive
- Mobile
- IoT
Qualcomm AI research

Advancing AI research to make efficient AI ubiquitous

Power efficiency
- Model design, compression, quantization, algorithms, efficient hardware, software tool

Personalization
- Continuous learning, contextual, always-on, privacy-preserved, distributed learning

Efficient learning
- Robust learning through minimal data, unsupervised learning, on-device learning

Perception
- Object detection, speech recognition, contextual fusion

Reasoning
- Scene understanding, language understanding, behavior prediction

Action
- Reinforcement learning for decision making

A platform to scale AI across the industry

Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.
Pre-built Edge AI sensing modules, plus tools to build your own

**Reality AI solutions**
- Prebuilt sound recognition models for indoor and outdoor use cases
- Solution for industrial anomaly detection
- Pre-built automotive solution that lets cars "see with sound"

**Reality AI Tools® software**
- Build prototypes, then turn them into real products
- Explain ML models and relate the function to the physics
- Optimize the hardware, including sensor selection and placement

Add Advanced Sensing to your Product with Edge AI / TinyML

https://reality.ai  info@reality.ai  @SensorAI  Reality AI
Build Smart IoT Sensor Devices From Data

SensiML pioneered TinyML software tools that auto generate AI code for the intelligent edge.

- End-to-end AI workflow
- Multi-user auto-labeling of time-series data
- Code transparency and customization at each step in the pipeline

We enable the creation of production-grade smart sensor devices.

sensiml.com
SynSense builds sensing and inference hardware for ultra-low-power (sub-mW) embedded, mobile and edge devices. We design systems for real-time always-on smart sensing, for audio, vision, IMUs, bio-signals and more.

https://SynSense.ai
**Syntiant**

**Syntiant Corp**, is moving artificial intelligence and machine learning from the cloud to edge devices. Syntiant’s chip solutions merge deep learning with semiconductor design to produce ultra-low-power, high performance, deep neural network processors. These network processors enable always-on applications in battery-powered devices, such as smartphones, smart speakers, earbuds, hearing aids, and laptops. Syntiant’s Neural Decision Processors™ offer wake word, command word, and event detection in a chip for always-on voice and sensor applications.

Founded in 2017 and headquartered in Irvine, California, the company is backed by Amazon, Applied Materials, Atlantic Bridge Capital, Bosch, Intel Capital, Microsoft, Motorola, and others. Syntiant was recently named a [CES® 2021 Best of Innovation Awards Honoree, shipped over 10M units worldwide](https://www.syntiant.com), and [unveiled the NDP120](https://www.syntiant.com) part of the NDP10x family of inference engines for low-power applications.

[www.syntiant.com](http://www.syntiant.com)  
[@Syntiantcorp](https://twitter.com/Syntiantcorp)
Focus on:
(i) developing new use cases/apps for tinyML vision; and (ii) promoting tinyML tech & companies in the developer community

Submissions accepted until August 15th, 2021
Winners announced on September 1, 2021 ($6k value)
Sponsorships available: sponsorships@tinyML.org
https://www.hackster.io/contests/tinyml-vision
Successful tinyML Summit 2021:

- **5** days of tutorials, talks, panels, breakouts, symposium
  - **4** tutorials
  - **6** keynotes & 6 plenary tinyTalks (more in breakouts)
  - **2** panel discussions
  - **5** disruptive news presentations
  - **17** breakout/partner sessions
  - **6** Best Product and Innovation Award Finalists & Presentations
  - **89** Speakers

- **5006** registered attendees representing:
  - **104** countries, **1000+** companies and **400+** academic institutions

- **26** Sponsoring companies

tinyML Summit-2022, January 24-26, Silicon Valley, CA
Local Committee in Sweden

Ali Balador, Senior Researcher RISE, Assistant Professor MDH, ali.balador@ri.se

Johan Malm, AI Engineer, PhD, Imagimob AB, johan.malm@imagimob.com

Magnus Melander, Evangelist and Co-founder THINGS, magnus@wbird.se

Åke Wernelind, Business Development, Imagimob AB, ake.wernelind@imagimob.com
# Next tinyML Talks

<table>
<thead>
<tr>
<th>Date</th>
<th>Presenter</th>
<th>Topic / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuesday, July 6</td>
<td>Shivy Yohanandan, Xailient</td>
<td>Cracking a 600 million year old secret to fit computer vision on the edge</td>
</tr>
</tbody>
</table>

Webcast start time is 8 am Pacific time

Please contact talks@tinyml.org if you are interested in presenting
Reminders

Slides & Videos will be posted tomorrow

tinyml.org/forums  youtube.com/tinyml

Please use the Q&A window for your questions
Masoud Daneshtalab is professor at Mälardalen University (MDH) in Sweden and an adjunct Professor at Tallinn University of Technology (TalTech) in Estonia. He is co-leading the Heterogeneous System research group (www.es.mdh.se/hero/). Since 2016 he has been in the Euromicro board of Directors, a faculty member of the HiPEAC network, and a permanent associate editor of Elsevier MICPRO. His research interests include hardware/software co-design and deep learning acceleration.

He has published 2 books and over 200 refereed international journals and conference papers.
This research group tackles the following scientific areas:

- Deep learning modeling and acceleration
- Hardware/software co-design and integration
- Interconnection and time-sensitive networking (TSN)
- Model-based development of heterogeneous systems

Members: 10 PhD students + 1 postdoc

Projects on Deep Learning:
- **AutoDeep**: Automatic Design of Safe, High-Performance and Compact Deep Learning Models for Autonomous Vehicles
- **SafeDeep**: Dependable Deep Learning for Safety-Critical Airborne Embedded Systems
- **HERO**: Heterogeneous systems: software-hardware integration,
- **DeepMaker**: Deep Learning Accelerator on Commercial Programmable Devices
Agenda

● Challenges on deep learning

● Introduction on DeepMaker

● Neural Architecture Search (Frontend)

● FPGA implementation of DNNs (Backend)
Challenges:

- How the computation-intensive DNN could be customized and deployed on the resource-limited hardware platform
- How much to rely on their outputs.

Increasing the complexity of DL algorithms for achieving better accuracy [1].

● provides resource-efficient DNNs with a required performance-level for (embedded) computing platforms.

● Algorithmic techniques based on evolutionary multi-objective optimization to generate optimal DNN models (topology and hyper-parameters) in terms of:
  ● Accuracy
  ● Execution time (Flops)
  ● Network complexity (Size)

● Using quantization techniques to reduce the huge memory footprint and bandwidth requirements.

Neural Architecture Search (NAS)

- NAS is to design DNN models automatically without human intervention.
- NAS methods try to design neural architectures with better accuracy.
  - But other criteria can also be considered in addition to accuracy:
    - Execution time (Flops)
    - Network complexity (Size)
Macro NAS

- Macro NAS search spaces with a chain structure.
- $O_i$ is an operation and the $i$-th operation in the chain structure.
- The input goes through a series of operations to get the final output.

Micro NAS

- Micro NAS search space often only needs to search a few small cell structures, and then repeatedly stack such cells to form the final architecture.
Popular Search Strategies

- **Random Search**
  - SMASH [Brock et al. 2017]

- **Evolutionary Algorithm**

- **Reinforcement Learning**
  - REINFORCE: [Zoph and Le 2017], [Pham et al. 2018 1]
  - Proximal policy optimization (PPO): [Zoph et al. 2018]

- **Bayesian Optimization (BO)**
  - GP with string kernel
    - Vizier [Chen et al. 2018]
    - Guided ES [Liu et al. 2018 2]
  - Hyperband Bayesian optimization [Wang et al. 2018]
  - BO and optimal transport [Kandasamy et al. 2018]

- **Gradient Based Optimization**
  - SMASH [Brock et al. 2017]
  - ENAS [Pham et al. 2018 2]
  - DARTS [Liu et al. 2018 1]
  - ProxylessNAS [Cai et al. 2018]
Designing Optimal CNN Architecture

- **Goal:** Designing Energy-efficient CNN for resource-limited devices
- Using a template-based design space inspired by DenseNet architecture.
- Using NSGA-II as the multi-objective search engine
  - FLOPS
  - Accuracy

(a) A genome type representing NAS hyperparameters. (b) Crossover & mutation operators. (c) Inspired template architecture of a generated network.

## Classification Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Approach</th>
<th>Solutions</th>
<th>#Params ($\times 10^6$)</th>
<th>Error Rate (%)</th>
<th>Compression Rate$^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MNIST</strong></td>
<td>Hand-Crafted</td>
<td>Wan et al. [25]</td>
<td>–</td>
<td>0.21</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>RL</td>
<td>MetaQNN [2]</td>
<td>5.59</td>
<td>0.35</td>
<td>0.023x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>* ADONN-Arch.3 [17]</td>
<td>0.13</td>
<td>0.41</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>Our M.Net.1</td>
<td>0.065</td>
<td>0.71</td>
<td>2x</td>
</tr>
<tr>
<td></td>
<td>Hand-Crafted</td>
<td>SimpleNet [8]</td>
<td>5.48</td>
<td>4.68</td>
<td>0.53x</td>
</tr>
<tr>
<td></td>
<td>Hand-Crafted</td>
<td>DenseNet (k = 12)-40 [12]</td>
<td>1.0</td>
<td>7.0</td>
<td>3.1x</td>
</tr>
<tr>
<td></td>
<td>Hand-Crafted</td>
<td>ResNet-20 [9]</td>
<td>0.27</td>
<td>8.75</td>
<td>11.48x</td>
</tr>
<tr>
<td></td>
<td>Hand-Crafted</td>
<td>ResNet-110 [9]</td>
<td>1.7</td>
<td>6.43</td>
<td>1.82x</td>
</tr>
<tr>
<td></td>
<td>Hand-Crafted</td>
<td>Gastaldi et al. [6]</td>
<td>26.4</td>
<td>2.86</td>
<td>0.117x</td>
</tr>
<tr>
<td></td>
<td>RL</td>
<td>Block-QNN-22L [29]</td>
<td>39.6</td>
<td>3.54</td>
<td>0.078x</td>
</tr>
<tr>
<td></td>
<td>RL</td>
<td>MetaQNN [2]</td>
<td>6.92</td>
<td>11.18</td>
<td>0.45x</td>
</tr>
<tr>
<td></td>
<td>RL</td>
<td>NAS-x1/v3 [30]</td>
<td>4.2/37.4</td>
<td>5.70/3.05</td>
<td>0.7x/0.083x</td>
</tr>
<tr>
<td></td>
<td>RL</td>
<td>Block-QNN-S [29$^*$]</td>
<td>6.1</td>
<td>4.88</td>
<td>0.5x</td>
</tr>
<tr>
<td></td>
<td>EC</td>
<td>Real et al. [21]</td>
<td>5.4</td>
<td>5.4</td>
<td>0.57x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>NSGA-Net [19]</td>
<td>3.3</td>
<td>3.85</td>
<td>0.94x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>ADONN-Arch.3 [17]</td>
<td>0.14</td>
<td>14.1</td>
<td>22.14x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>Lou et al. [18]</td>
<td>0.56</td>
<td>13.8</td>
<td>5.5x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>Our CI10-Net.1</td>
<td>0.065</td>
<td>16.49</td>
<td>47.7x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>Our CI10-Net.2</td>
<td>0.21</td>
<td>11.05</td>
<td>14.7x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>Our CI10-Net.3</td>
<td>1.0</td>
<td>6.81</td>
<td>3.1x</td>
</tr>
<tr>
<td><strong>CIFAR-100</strong></td>
<td>RL</td>
<td>MetaQNN [2]</td>
<td>11.18</td>
<td>27.14</td>
<td>0.28x</td>
</tr>
<tr>
<td></td>
<td>RL</td>
<td>Block-QNN-S [29$^*$]</td>
<td>6.1</td>
<td>20.65</td>
<td>0.5x</td>
</tr>
<tr>
<td></td>
<td>Hand-Crafted</td>
<td>DenseNet (k = 12)-40 [12]</td>
<td>1.0</td>
<td>27.55</td>
<td>3.1x</td>
</tr>
<tr>
<td></td>
<td>Hand-Crafted</td>
<td>DenseNet (k = 12)-100 [12]</td>
<td>7.0</td>
<td>23.79</td>
<td>0.44x</td>
</tr>
<tr>
<td></td>
<td>Hand-Crafted</td>
<td>SimpleNet [8]</td>
<td>5.48</td>
<td>26.58</td>
<td>0.53x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>NSGA-Net [19]</td>
<td>3.3</td>
<td>26.71</td>
<td>0.94x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>Our CI100-Net.1</td>
<td>1.1</td>
<td>26.83</td>
<td>2.82x</td>
</tr>
<tr>
<td></td>
<td>MO$^2$.EC</td>
<td>Our CI100-Net.2</td>
<td>1.89</td>
<td>24.87</td>
<td>1.64x</td>
</tr>
</tbody>
</table>

$^*$ The baseline for comparing the compression rate.

$^1$ The values more than 1.0 indicate improvement. Best results are in **bold.**

- **MNIST** (Compare to MetaQNN by Google): 43x compression rate, **0.06%** accuracy loss
- **CIFAR-10** (Compare to the most accurate): 26.4x compression Rate, **4%** accuracy loss
- **CIFAR-10** (Compare to MetaQNN by Google): 6.8x compression Rate, 4.3% better accuracy
- **CIFAR-100** (Compare to MetaQNN by Google): 10x compression Rate, **0.5%** accuracy loss
- **CIFAR-100** (Compare to the most accurate and light-weight network, CondensNet): 2.8x compression Rate, **9%** accuracy loss
## Hardware Implementation Results

(Compared to the ADONN with 0.14M parameters)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Dataset</th>
<th>Network</th>
<th>NID (×10^6)</th>
<th>GPU Energy efficiency</th>
<th>Speedup (Kernel)</th>
<th>Speedup: D2H/H2D (Comm.)</th>
<th>Intel® CPU Energy efficiency</th>
<th>Speedup (Inference Time)</th>
<th>ARM processor Energy efficiency</th>
<th>Speedup (Inference Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td></td>
<td>ADONN-Arch.3</td>
<td>7.06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MNIST</td>
<td>M_Net.1</td>
<td>15.25</td>
<td>1.55x</td>
<td>1.49x</td>
<td>1.59x/2.95x/1.49x</td>
<td>1.51x</td>
<td>1.56x</td>
<td>1.08x</td>
<td></td>
</tr>
<tr>
<td>CIFAR-10</td>
<td></td>
<td>ResNet-20</td>
<td>3.37</td>
<td>4.2x</td>
<td>4.25x</td>
<td>0.62x/0.79x/0.75x</td>
<td>0.81x</td>
<td>1.05x</td>
<td>0.87x</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ResNet 110</td>
<td>0.55</td>
<td>1.53x</td>
<td>1.61x</td>
<td>0.108x/0.15x/0.093x</td>
<td>0.05x</td>
<td>0.16x</td>
<td>0.87x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>DcnscNet (k = 12)-100</td>
<td>0.136</td>
<td>0.25x</td>
<td>0.027x</td>
<td>0.06x/0.073x/0.099x</td>
<td>0.112x</td>
<td>0.07x</td>
<td>0.116x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>*ADONN-Arch.3</td>
<td>0.13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>C10_Net.1</td>
<td>13</td>
<td>1.54x</td>
<td>1.51x</td>
<td>1.61x/1.57x/1.5x</td>
<td>1.59x</td>
<td>1.52x</td>
<td>1.32x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>C10_Net.3</td>
<td>0.93</td>
<td>0.14x</td>
<td>0.154x</td>
<td>0.25x/0.313x/0.3x</td>
<td>0.372x</td>
<td>0.3x</td>
<td>0.37x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CIFAR-100*</td>
<td>ResNet-110</td>
<td>0.43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CIFAR-100*</td>
<td>DenseNet (k = 12)-100</td>
<td>0.109</td>
<td>0.017x</td>
<td>0.016x</td>
<td>0.44x/0.46x/0.92x</td>
<td>1.09x</td>
<td>0.94x</td>
<td>0.74x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CIFAR-100*</td>
<td>C100_Net.1</td>
<td>0.66</td>
<td>0.11x</td>
<td>0.1x</td>
<td>2.28x/2x/3.2x</td>
<td>3.46x</td>
<td>1.80x</td>
<td>2.88x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CIFAR-100*</td>
<td>C100-Net.2</td>
<td>0.4</td>
<td>0.06x</td>
<td>0.06x</td>
<td>1.45x/1.39x/2.45x</td>
<td>2.58x</td>
<td>1.35x</td>
<td>1.83x</td>
<td></td>
</tr>
</tbody>
</table>

* The baseline for comparing the energy efficiency of different architectures. The values more than 1.0 indicate improvement. Best results are in **bold**.
Industrial Use Case #1: Runway Detection for Auto-landing System

- All the results have been achieved by running on NVIDIA GTX 1080ti.
- 4x better inference time
- 4.22x more compact
- 4.15% more accurate

### Classification Results

<table>
<thead>
<tr>
<th>Solution</th>
<th>RetinaNet</th>
<th>DeepMaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG. Accuracy</td>
<td>86.95%</td>
<td>91.1%</td>
</tr>
<tr>
<td>Inference Time (ms)</td>
<td>63</td>
<td>16</td>
</tr>
<tr>
<td>Frame/Second</td>
<td>15</td>
<td>62</td>
</tr>
</tbody>
</table>
What is Stereo Matching?

- A piece of Stereo Vision process which aim to estimate depth of contents by getting the images of stereo camera.
- The main process in this part is finding equal pixels in stereo pictures and estimating their distances to generate **Disparity Map**.
- Although CNN-based stereo matching algorithms provides superior accuracy, they suffer from **huge memory footprint** and **computational cost**.
- For example, full training a stereo matching model may costs under loading 4 GPUs for more than **6 hours**!
DenseDisp: Optimizing CNN Architecture for Depth Estimation

- To reduce the computational demand of depth estimation task, we utilized multi-objective Simulated Annealing (SA) to design a hardware-friendly architecture.
- We selected the Siamese neural model due to high flexibility.
- Using SA to decrease the search cost achieving **24x reduction** compared to state-of-the-art.

- We used Matrix design space to generate discrete chain macro NAS structure.
**DenseDisp Experimental Results: Disparity Estimation Performance**

- **KITTI-2015** is a dataset of real-world images that consist of 200 training and 200 test stereo pairs with the dimensions of $376 \times 1240$ pixels.

- Our DenseDisp architecture achieves **92.03%** accuracy with **0.6 Sec.** inference time and **1.03M parameters / 1.56M multiply-adds (FLOPs)**, achieving a cutting-edge accuracy-FLOPs trade-off.

### Classification/Complexity Results

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Accuracy (%)</th>
<th>Params $\times 10^6$</th>
<th>FLOPS $\times 10^6$</th>
<th>Exploration Method</th>
<th>Compression Rate $\dagger$ (%)</th>
<th>Exploration Cost (GPU Days)</th>
<th>GPU Latency (Sec.)</th>
<th>H.W. Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoDispNet-BOHB-C $\dagger$</td>
<td>97.82</td>
<td>37</td>
<td>61</td>
<td>RL</td>
<td>1</td>
<td>42</td>
<td>-</td>
<td>1x GTX 1080ti</td>
</tr>
<tr>
<td>Content-CNN</td>
<td>95.46</td>
<td>7</td>
<td>2</td>
<td>Hand-Crafted</td>
<td>35.5</td>
<td>-</td>
<td>1</td>
<td>1x Titan Xp</td>
</tr>
<tr>
<td>GA-Net-deep</td>
<td>98.19</td>
<td>-</td>
<td>-</td>
<td>Hand-Crafted</td>
<td>-</td>
<td>-</td>
<td>1.8</td>
<td>1x Tesla P40</td>
</tr>
<tr>
<td>DenseDisp (Ours)</td>
<td>90.03</td>
<td>1.03</td>
<td>1.56</td>
<td>Meta-heuristic</td>
<td>39.1</td>
<td>2</td>
<td>0.56</td>
<td>1x GTX 20800</td>
</tr>
<tr>
<td>DenseDisp + Median Filter (Ours)</td>
<td>92.01</td>
<td>1.03</td>
<td>1.56</td>
<td>-</td>
<td>39.1</td>
<td>2</td>
<td>0.6</td>
<td>1x GTX 2080</td>
</tr>
</tbody>
</table>

$\dagger$ The baseline for comparing the compressing rate. Our considerable results are in green cells.

### FPGA implementation Results.

<table>
<thead>
<tr>
<th>Work</th>
<th>Method</th>
<th>Disparity Range</th>
<th>FPS</th>
<th>Accuracy (%) (D1-all)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Schumacher &amp; Greiner, 2014)</td>
<td>SGM</td>
<td>160</td>
<td>199</td>
<td>81.94</td>
</tr>
<tr>
<td>Virtex-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Rahnama et al., 2018)</td>
<td>ELAS (Geiger et al., 2010)</td>
<td>-</td>
<td>9.5</td>
<td>86.1</td>
</tr>
<tr>
<td>Zynq ZC706 (CPU+FPGA)</td>
<td>SGM+ELAS</td>
<td>-</td>
<td>52</td>
<td>91.3 $\ddagger$</td>
</tr>
<tr>
<td>(Rahnama et al., 2019)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zynq ZCU104 (CPU+FPGA)</td>
<td>CNN</td>
<td>128</td>
<td>6</td>
<td>95.46</td>
</tr>
<tr>
<td>Content-CNN (Luo et al., 2016)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zynq UltraScale+</td>
<td>CNN</td>
<td>128</td>
<td>12</td>
<td>92.3 (94.3 $\ddagger$)</td>
</tr>
<tr>
<td>(Ours)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\dagger$ The Efficient Large-Scale Stereo (ELAS), is the fastest triangulation-based disparity estimation algorithm on CPU.

$\ddagger$ With using median filter.

DenseDisp was the only network we managed to deployed in Intel NCS2 (Movidius)
DeepMaker - Backend

- **Backend:**
  - Efficiently map the optimized DNN architecture to target devices:
    - GPU
    - TPU
    - Intel NCS2 (Movidius)
    - FPGA

**Acceleration:**
- Running the DNN on the FPGA or ASIC to achieve a **higher power performance gains**
- Gap: DNN design is performed by high-level software/data specialists, whereas the hardware is implemented by low-level hardware ones
- HLS is a promising tool to fill the gap between DNN designers and the programmable logic (FPGA)/ASIC.

**High-level synthesis:**
- Generates a circuit for a design described in high-level languages (mainly C or C++).
- Easy to use but required to be an expert using directives (without directives, results will be worth than CPU)
- Multiple architectures (area-latency) through simply adding just a few **directives** (e.g. loop unrolling, pipelining)
DeepMaker - DeepHLS

Keras Implementation (e.g. LeNet Network)

```python
model.add(Conv2D(filters=6, kernel_size=(5,5),
                   activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters=16, kernel_size=(5,5),
                  activation='relu'))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Extraction of the layers' information

HLS Implementation in C / C++

<table>
<thead>
<tr>
<th>CNN</th>
<th>BRAM</th>
<th>FF</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG</td>
<td>616</td>
<td>7991</td>
<td>21636</td>
</tr>
<tr>
<td>AlexNet</td>
<td>326</td>
<td>4200</td>
<td>11257</td>
</tr>
<tr>
<td>LeNet-5</td>
<td>16</td>
<td>2348</td>
<td>5763</td>
</tr>
</tbody>
</table>
The impact of the fixed-point size on accuracy and resource utilization (AlexNet)
DeepMaker – DeepHLS

Features summary

A full toolchain including the **Keras to C conversion** and **conversion accuracy verification**.

The generated C has easy-to-use **knobs** to switch the HLS implementation from **floating point to fixed point** with arbitrary quantization levels.

The generated C code is in **ANSI C**, and thus, it can be used in almost **all** the open-source and commercial **HLS tools**.

The C code is **flat**, allowing the HLS users to easily **add directives** such as pipeline and unroll in a fine-grained way. This also enables the **intra-layer optimizations**.

For verification and accuracy evaluation, a **file-to-memory interface** is implemented that allows the toolchain to be used for **large networks** with many parameters and large datasets.
DeepMaker – Backend - Second stage

An automatic flow for DNN implementation using HLS

Why do we need DSE? Why not fully unroll and pipeline every loop?

- Fails: if the unroll factor results in a very large circuit that exceeds the processing capacity of the HLS tool for scheduling and routing
- Fails: if the required resources are more than the available resources on the specified chip
- Fails: if the concurrent memory accesses needed cannot be provided due to the limited bandwidth of the specified memory unit
- Not optimal: two different directive set may result in the same performance improvement but different resource utilization
DeepMaker – Backend - Second stage (step 1)

Array Partitioning

- Is applied to intermediate data elements (layer outputs)
- Helps simultaneous access to multiple elements
For-loops without parents mark the beginning of a new partition.
DeepMaker – Backend - Second stage (step 3)

More than 137 billion possibilities

Choose a partition

Store the results for this partition

Run hill-climbing on the chosen partition

i = 0

Apply directives and run HLS

i = i + 1

Choose the set of i directives with the minimum latency

Fix the chosen i directives

End processing this partition if no improvement

Partition #1 Done!
Partition #2 Done!
Partition #3 Under DSE!
DeepMaker – Backend - Second stage (step 3)

Resource-aware DSE

- Automatic conversion
- DNN in C/C++
- Applying Quantization
- Applying HLS Array Directives (Array Partitioning)
- Per-partition directive impact (Latency - Utilization)
- Pareto-frontier (Latency - Utilization)
- RTL Implementation
- Directive Set (Minimal Latency)

- Latency and resource utilization for all possible partition directive set combinations

- Pareto-frontier
  - Helps designer choose a specific point in the explored space based on the required latency and available resources.

- Directive Set (Minimal Latency)
  - The point at which all the directives are applied, and the minimum latency is achieved.

- Directive set impact predictor
  - Based on the impacts of the directives related to a specific partition
  - Is needed to avoid excessive HLS runs, which is the most time-consuming part of the flow
DeepMaker – Backend - Results

Comparison with existing HLS based works and other HLS implementations

<table>
<thead>
<tr>
<th>Row</th>
<th>Implementation</th>
<th>Latency (ms)</th>
<th>FF</th>
<th>LUT</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No directives (1)</td>
<td>24.34</td>
<td>2,099</td>
<td>5,884</td>
<td>32-bit Floating</td>
</tr>
<tr>
<td>3</td>
<td>No directives (2)</td>
<td>7.15</td>
<td>1,219</td>
<td>4,651</td>
<td>16-bit Fixed</td>
</tr>
<tr>
<td>4</td>
<td>[14]</td>
<td>3.58</td>
<td>33,585</td>
<td>32,589</td>
<td>11-bit Fixed</td>
</tr>
<tr>
<td>This work</td>
<td>HLS DSE</td>
<td>0.55</td>
<td>157,927</td>
<td>183,746</td>
<td>16-bit Fixed</td>
</tr>
</tbody>
</table>

Comparison with other implementations

<table>
<thead>
<tr>
<th>Row</th>
<th>Title</th>
<th>Description</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2018 - An FPGA Based Hardware Accelerator for Classification of Handwritten Digits</td>
<td>Custom Letnet Implementation</td>
<td>5.58</td>
</tr>
<tr>
<td>2</td>
<td>2016 - From high-level deep neural models to Fpgas</td>
<td>DNNWeaver: Template based</td>
<td>2.41</td>
</tr>
<tr>
<td>3</td>
<td>2017 - Throughput-optimized fpga accelerator for deep convolutional neural networks</td>
<td>Computation/Memory analysis</td>
<td>1.35</td>
</tr>
<tr>
<td>This work</td>
<td>Proposed method</td>
<td></td>
<td>0.55</td>
</tr>
</tbody>
</table>

6.51x higher performance

Automatic and easy to use for DNN designers without HW knowledge

4.5x faster than 2 (DNNWeaver)
- Representing the floating-point weights and/or activations with fewer bits
  - Reduces memory footprint
  - Improves computational efficiency
  - Essential for deploying deep models onto IoT devices
- Two Popular types of network quantization
  - **Binary quantization**: 1-bit values \{-1, 1\}
  - **Ternary quantization**: 2-bit values \{-1, 0, 1\}

---

Comparing different quantization methods:

Ternary neural network (TNN) vs. Binary neural network (BNN)

<table>
<thead>
<tr>
<th>Criteria</th>
<th>TNN</th>
<th>BNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparsity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Energy Consumption</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Footprint</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goal: Increasing the sparsity rate and energy efficiency of TNNs
- Introduced a novel ternarized training to reduce the accuracy gap compared to floating-point training
- Proposing a novel multiplier for ternary neural networks to improve network sparsity, energy efficiency, and silicon footprint
- Proposing novel ternarized activation function to improve model accuracy

We propose a novel TNN that ternarizes both weights and activations at training time.
To clip weights into the \([-1, 1]\) interval, we use Eq. 1 [10] to obtain the suitable value of both activations and weights depending on their values:

\[
W_t = \begin{cases} 
+1 & \text{if } W \geq \Delta, \\
0 & \text{if } |W| \leq \Delta, \\
-1 & \text{if } W \leq -\Delta.
\end{cases}
\]

\[\Delta = \frac{0.7}{n} \sum_{i=1}^{n} |X_i| \quad (1)\]

Where \(\Delta\) is a threshold parameter (the sparsity level), \(n\) is the total number of activations or weights in a filter, and \(X\) denotes either activations or weights.
Note that, \(\Delta\) changes dynamically for each filter and the activation during training.
We eliminate \(|X| < \Delta\) values which are ineffective in the computation.

We use the standard sign and magnitude encoding to encode ternary values.

**Sign and magnitude encoding**

<table>
<thead>
<tr>
<th>alt. 1</th>
<th>alt. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>01 or 10</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

TOT-Net increases **sparsity** due to adding zero activation.

## DeepMaker – Quantization – ToT-NET – Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy %, (Neural Network Architecture)</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>ImageNet §</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-Precision [6]</td>
<td></td>
<td>89.67</td>
<td>64.32</td>
<td>80.2/56.6, (AlexNet) +</td>
</tr>
<tr>
<td>BC [9]</td>
<td></td>
<td>NA</td>
<td>NA</td>
<td>61/35.4, (AlexNet) +</td>
</tr>
<tr>
<td>BNN [4]</td>
<td></td>
<td>NA</td>
<td>NA</td>
<td>50.4/27.9, (AlexNet) +</td>
</tr>
<tr>
<td>TOT-Net</td>
<td></td>
<td>87.53, (NIN)</td>
<td>61.61, (NIN)</td>
<td>68.20/42.99, (AlexNet) **+</td>
</tr>
</tbody>
</table>

* The experiments have been run by us (trained by 20 epochs).
+ Presenting both top-5 and top-1 accuracy, respectively.

The impact of tweaking learning rate on the TOT-Net accuracy

Recurrent Neural Networks
- Long Short Term Memory (LSTM) is the most popular and effective RNN
- Sequence learning task

Health care application and medical diagnoses:
- ElectroEncephaloGraphy (EEG),
- Electrocardiogram (ECG) signal classification,
- Electromyography (EMG) signal classification
The DeepMaker Framework (cont.), RNN/LSTM (cont.)

- Ternary Compression Method
  - **Ternarize** weights, inputs, and all internal parameters of LSTM cell ternarized to \{-1,0,1\} values
  - Considering the proposed Scaling Factor to reduce accuracy loss
  - In our micro-architecture the multiplication of activation function, and ternarizing function are merged and implemented by couple of XNOR gates
  - The experiments on the ECG and EMG signals show
    - reducing **Latency**: $\sim 110 \times \:-120 \times$ and **Silicon for**: $\sim 29 \times \:-33 \times$ respectively.
    - Gain **Memory footprint**: 17\times and **bandwidth** 31\times compared to full precision signal classification.
    - $\sim 0.88\:-2.04\%$ **accuracy loss** in comparison to the full-precision counterparts


Conclusion

- DeepMaker deals with
  - Optimizing DNN architectures with NAS considering the following criteria
    - Complexity
    - Accuracy
  - Currently looking for hardware-aware NAS
    - Complexity
    - Accuracy
    - Energy and resource utilization
    - Robustness
  - DeepHLS and DSE-HLS for finding the best possible directive sets
    - Using predictive models to speed up the process
  - Ternary NN, looking for optimal boundaries for -1, 0, 1
  - Find papers in google scholar (Masoud Daneshtalab) or my MDH homepage for preprints
Copyright Notice

This presentation in this publication was presented as a tinyML® Talks webcast. The content reflects the opinion of the author(s) and their respective companies. The inclusion of presentations in this publication does not constitute an endorsement by tinyML Foundation or the sponsors.

There is no copyright protection claimed by this publication. However, each presentation is the work of the authors and their respective companies and may contain copyrighted material. As such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

tinyML is a registered trademark of the tinyML Foundation.

www.tinyML.org