DNPU: An Energy-Efficient Deep Neural Network Processor with On-Chip Stereo Matching

Dongjoo Shin, Jinmook Lee, Jinsu Lee, Juhyoung Lee, and Hoi-Jun Yoo

Semiconductor System Laboratory
School of EE, KAIST
About DNPU v1 – Designed in 2015

- **Deep Neural network Processing Unit**
- Embedded Deep Neural Network Processing in Mobile Platforms

- **Heterogeneous Architecture** for Convolutional Layers vs MLP-RNN

- **Convolution Processor**
  - Mixed workload division method
  - Layer-by-layer dynamic-fixed-point operation with on-line adaptation

- **MLP-RNN Processor**
  - LUT-based multiplication with weight quantization (Q-table)

- **Stereo Matching Processor** for Depth Map Generation
- **RGB-D 4-ch Processing Support**

MLP: Multi-layer Perceptron
RNN: Recurrent Neural Network
Targets of DNPU

- Embedded Deep Neural Network Processing in Mobile Platforms

- Platform: Mobile
  - Low-power and high energy efficiency

- Task: Vision
  - 4.7GB/min (HD 30fps) → Bottleneck for cloud computing

- Operation: Real-time & low-latency
  - High throughput and embedded computing

- Smart Machines & Intelligence-on-Things (IoT)
  - Robot, drone, smartphone, wearable devices, home appliances

DNN-dedicated SoC
Why both CNN & RNN?

- **CNN:** Visual feature extraction and recognition
  - Face recognition, image classification...
- **RNN:** Sequential data recognition and generation
  - Translation, speech recognition...
- **CNN + RNN:** CNN-extracted features → RNN input

**Previous works**
- Optimized for convolution layers only: [1], [2], [3], [4]
- Optimized for MLP and RNN only: [5]

Hardware for Deep Neural Networks

- **CPU**
  - Intel Xeon E7-8894 v4
  - 14 nm, 2.4 GHz
  - 11 GOPS/W (FP 32)

- **GPU**
  - Nvidia P100
  - 16 nm, 1.3 GHz
  - 64 GOPS/W (FP 16)
  - 2.3 TOPS/W (INT 8)

- **ASIC (Server)**
  - Google TPU
  - 28 nm, 0.7 GHz
  - 4.2 TOPS/W (INT 8)

- **ASIC (Mobile)**
  - KAIST DNPU
  - 65 nm, 0.1 GHz

Dedication: Low to High
Energy Efficiency: Low to High
**DNPU: DNN-dedicated SoC**

- **CPU**
  - Control
  - ALU
  - ALU
  - ALU
  - Cache
  - Complex control logic
  - Low compute density
  - High programmability

- **GPU**
  - Parallel computation
  - High compute density
  - Floating point data type
  - Parallel Processing

- **ASIC (server)**
  - Matrix multiply arch.
  - Dedicated memory arch.
  - Fixed point data type

- **DNPU (mobile)**
  - Heterogeneous arch.
  - Fixed computation pattern
  - Dedicated data type

**Key Features**
- Higher Energy Efficiency (Operations/W)
- Higher Programmability
**DNPU: DNN-dedicated SoC**

- **CPU**
  - Complex control logic
  - Low compute density
  - High programmability

- **GPU**
  - Parallel computation
  - High compute density
  - Floating point data type

- **ASIC (server)**
  - Matrix multiply arch.
  - Dedicated memory arch.
  - Fixed point data type

- **DNPU (mobile)**
  - Heterogeneous arch.
  - Fixed computation pattern
  - Dedicated data type

---

**DNN itself has high adaptability for various applications**
Heterogeneous Characteristics

- Convolution Layer (CNN): Computation >> Parameter
- MLP (of CNN), RNN: Computation ≈ Parameter
Heterogeneous Architecture

- **Architecture for Convolution Processor**
  - Image, kernel, convolution reuse architecture
  - Dynamic fixed-point with on-line adaptation (Feature map reduction)
  - Distributed memory
  - On-chip memory portion: Input > weight > output

- **Architecture for MLP-RNN Processor**
  - Matrix multiply architecture
  - Weight quantization (Weight reduction)
  - LUT-based multiplication (Quantization-table or Q-table)
  - On-chip memory portion: Output > weight > input
**Overall Architecture**

**Convolution Processor**
- **Kernel**: Support any size
  - Maximum utilization @ 3xn, 6xn, 9xn, …: 100%
  - Minimum utilization @ 1x1: 33%
- **Stride**: 1, 2, 4
- **Channel**: Support any size
  - Optimized for 16, 32, 64, 128, 256, 512, 1024
- **Pooling**: 2 x 2
- **Activation**: ReLU

**MLP-RNN Processor**
- **Channel**: Support any size
- **Activation**: ReLU, sigmoid, tanh

**Stereo Matching Processor**
- **Depth level**: 64
- **Input image**: QVGA
Convolution Processor

Convolution Core
- Image Memory (16KB)
- Weight Mem. (1KB)
- PE Array (12x4)
- FSM Ctrlr.
- Partial Sum Registers

Aggregation Core
- Image Memory (16KB)
- FSM Ctrlr.
- Pooling
- Relu
- Partial Sum Registers

Data NoC
- Instruction Network
- Partial Sum Network

CNN Controller
Mixed Workload Division Method

- Limited on-chip memory size → Workloads should be divided

**Overhead #1**: Overlapped region is needed

**Overhead #2**: Multiple off-chip accesses for kernel weights

**Overhead #1**: Multiple off-chip accesses for partial output

**Overhead #2**: Cannot achieve advantages from pooling and ReLU

- Image Division
- Channel Division
Mixed Workload Division Method

<table>
<thead>
<tr>
<th>Input Layer Division Method</th>
<th>Image Division</th>
<th>Channel Division</th>
<th>Mixed Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Division</td>
<td><img src="image" alt="Image Division" /></td>
<td><img src="image" alt="Channel Division" /></td>
<td><img src="image" alt="Mixed Division" /></td>
</tr>
<tr>
<td>Multiple off-chip accesses for weight</td>
<td><img src="image" alt="Multiple off-chip accesses for weight" /></td>
<td><img src="image" alt="Multiple off-chip accesses for partial output" /></td>
<td>Use both divisions</td>
</tr>
<tr>
<td>Having advantage: image &gt;&gt; weight</td>
<td><img src="image" alt="Having advantage: image &gt;&gt; weight" /></td>
<td><img src="image" alt="Having advantage: image &lt;&lt; weight" /></td>
<td></td>
</tr>
</tbody>
</table>

Off-chip Access (W/O Compression Scheme)

- **Input Image**
  - \( W_i \times H_i \times C_i \)
  - \( W_i \times H_i \times C_i \)
  - \( W_i \times H_i \times C_i \)

- **Weight**
  - \( W_f \times H_f \times C_i \times C_o \times Img. Div. # \)
  - \( W_f \times H_f \times C_i \times C_o \times Img. Div. # \)
  - \( W_f \times H_f \times C_i \times C_o \times Img. Div. # \)

- **Output Image**
  - \( W_o \times H_o \times C_o \times Ch. Div. # \times 2 \)
  - \( W_o \times H_o \times C_o \times Ch. Div. # \times 2 \)
  - \( W_o \times H_o \times C_o \times Ch. Div. # \times 2 \)

**VGG-16 Off-chip Memory Access Analysis**

- Image Division
- Channel Division
- Mixed Division

→ Mixed division can take lower points
Layer-by-Layer Dynamic Fixed-point

- Data distribution in each layer

![Layer-by-Layer Data Distribution](image1)

- Floating-point implementation
  - Large data range, but high cost

- Fixed-point implementation
  - Low cost, but limited data range
Layer-by-Layer Dynamic Fixed-point

- Each layer has different WL and FL
  - Having floating-point characteristic via layers
- WL and FL are fixed in a same layer
  - Enabling fixed-point operation

Example)
- Conv. Layer 1 – WL: 8, FL: 1
- Conv. Layer 2 – WL: 8, FL: 2
Off-line Learning-based Approach (Previous)

- Off-line (off-chip) learning-based FL selection
- FL is trained to fit with given image data set
- Selected FL is used for every image at run time

Image Data Set

Fraction Length (FL) Sets
Set1 – L1: 0, L2: 0, L3:-4 …
Set2 – L1: 1, L2: -1, L3: -5 …
Set3 – L1: 1, L2: -3, L4: -6 …

Finding FL set which shows the minimum error with given image data set
Proposed On-line Adaptation

- On-line adaptation-based FL selection

- **On-line (on-chip)** learning-based FL selection
- FL is *dynamically fit* to *current input image*

→ No off-chip learning, lower required WL
Performance Comparisons

- Image classification results

![Graph showing performance comparisons]

- Proposed Dynamic-Fixed-Point with Adaptation
- Dynamic Fixed-Point
- Fixed-Point

Baseline: 32-bit floating point

Used Model: VGG-16
Used Dataset: ImageNet
MLP-RNN Processor

Weight Mapping Table

Q-table Construction FIFO

External injection

Weight Index Buffer

Q-table 0
8-to-8

ADD Tree 0
8-to-1

ACC Reg.

Internal Data Buffer (8 KB)

Sigmoid & tanh

Element Mult. 0

Element Mult. 1

Element Mult. 7

Vector Multiplication

Element-wise Multiplication
\[ o_m = W_{0m} i_0 + W_{1m} i_1 + \ldots + W_{nm} i_n + b_m \]

\[
\begin{bmatrix}
1 & i_0 & i_1 & \ldots & i_n
\end{bmatrix}
\begin{bmatrix}
b_0 & b_1 & \ldots & b_m \\
W_{10} & W_{11} & \ldots & W_{1m} \\
W_{n0} & W_{n1} & \ldots & W_{nm}
\end{bmatrix}
\]
MLP-RNN Processor

RNN - LSTM

\[ i_t = \sigma(W_{xi}x_t + W_{yi}y_{t-1} + W_{ci}c_{t-1} + b_i) \]
\[ f_t = \sigma(W_{xf}x_t + W_{yf}y_{t-1} + W_{cf}c_{t-1} + b_f) \]
\[ o_t = \sigma(W_{xo}x_t + W_{yo}y_{t-1} + W_{co}c_t + b_o) \]
\[ z_t = \tanh(W_{xz}x_t + W_{yz}y_{t-1} + b_z) \]
\[ c_t = f_t c_{t-1} + i_t z_t \]
\[ y_t = o_t \tanh(c_t) \]
MLP-RNN Processor: Weight Quantization

**FC Layer Weight Quantization**

<table>
<thead>
<tr>
<th></th>
<th>Top-1 Error</th>
<th>Top-5 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>32bits</td>
<td>42.78%</td>
<td>19.73%</td>
</tr>
<tr>
<td>4bits</td>
<td>42.79%</td>
<td>19.73%</td>
</tr>
<tr>
<td>2bits</td>
<td>44.77%</td>
<td>22.33%</td>
</tr>
</tbody>
</table>

ImageNet Classification Test

**LSTM Layer Weight Quantization**

<table>
<thead>
<tr>
<th></th>
<th>Perplexity (Lower is better)</th>
<th>BLEU (Higher is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32bits</td>
<td>15.680</td>
<td>55.7 / 37.4 / 24 / 15.7</td>
</tr>
<tr>
<td>4bits</td>
<td>15.829</td>
<td>56.7 / 38 / 24.4 / 15.7</td>
</tr>
<tr>
<td>2bits</td>
<td>19.298</td>
<td>58.4 / 38.6 / 24 / 14.8</td>
</tr>
</tbody>
</table>

Flickr 8K Image Captioning Test
MLP-RNN Processor: Q-table Construction

- Pre-computation with each quantized weight

Multiplications between input and quantized weights

Multiplication results are also quantized to 16 values.

Weight Mapping Table

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>$w_0$</td>
<td>$w_1$</td>
<td>$w_2$</td>
<td>...</td>
<td>$w_{15}$</td>
</tr>
</tbody>
</table>

Input Registers

$Q$-table for $I_0$

$Q$-table for $I_1$

$Q$-table for $I_7$
MLP-RNN Processor: Multiplication with Q-Table

- Decode index to load the pre-computed result

Quantization:
: 16-bit weight → 4-bit index

Off-chip Access:
: 75% reduction for weight

<table>
<thead>
<tr>
<th></th>
<th>16-bit Fixed-point</th>
<th>Q-table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>1890</td>
<td>1380</td>
</tr>
<tr>
<td></td>
<td>(per 1 Mult.)</td>
<td>(per 1 Mult.)</td>
</tr>
<tr>
<td>Power</td>
<td>0.32mW</td>
<td>0.068mW</td>
</tr>
<tr>
<td></td>
<td>(per 1 Mult.)</td>
<td>(per 1 Mult.)</td>
</tr>
<tr>
<td>Latency</td>
<td>7.1ns</td>
<td>0.49ns</td>
</tr>
<tr>
<td></td>
<td>(Unconstrained)</td>
<td>(Unconstrained)</td>
</tr>
</tbody>
</table>

Simulation results on 65nm @ 200MHz, 1.2V
Motivation: Why RGB-D?

- Higher Accuracy than RGB
  - Object detection
  - Image segmentation
  - 3D face recognition
  - Car detection
  - Gesture recognition
  - Visual attention
2011-2022 Market Forecast for 3D Imaging & Sensing Devices
(Source: 3D Imaging & Sensing 2017 report, April 2017, Yole Developpement)

2016
US $1.3 billion

2022
US $9 billion
Big Data and Transfer Learning

- ImageNet DB: 14,197,122 images, 21,841 synsets indexed
- Rich visual features can be trained with ImageNet
  
  **Transfer Learning**

- Enabling learning for other vision tasks with small dataset

- Consumer 3D imaging devices (Smartphone): Big data
- Transfer learning on RGB-D dataset

→ Amplification on RGB-D DNNs will be coming soon
Hierarchical Bit-line is used to Minimize the Bit-line Switching
Proposed 4-Square Integral Image

- 4 independent region ➔ 4-way *parallel processing*
- $\frac{1}{4}$ Maximum range ➔ data bit width *2-bit reduction*
- No additional computation & Mem access cost

4-way Integral Image Generation

\[
S = A - B - C + D
\]

\[
S = A - B + C - D
\]

\[
S = A + B + C + D
\]

\[
S = A + B - C - D
\]
Chip Photograph and Summary

<table>
<thead>
<tr>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
</tr>
<tr>
<td>Die Area</td>
</tr>
<tr>
<td>SRAM Size</td>
</tr>
<tr>
<td>Supply Voltage</td>
</tr>
<tr>
<td>Operating Frequency</td>
</tr>
<tr>
<td>Power Consumption</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- **DNPU**: Enabling **embedded DNNs** in mobile platforms

- **Heterogeneous Architecture**
  - Convolution Processor
  - MLP-RNN Processor

- Stereo Matching Processor
- **RGB-D 4-ch** Operation

- Next version DNPU for **large input image & large kernel**