Optimizing Quantized Neural Networks on FPGAs

Presented by:
Robert Green – Design Engineer

Copyright ASIC Design Services 2018
Core Deep Learning
an embedded FPGA solution

Outline

• Brief introduction to deep learning
• Convolutional neural networks on FPGAs
• Addressing the computation problem
• Addressing the memory problem
• Core Deep Learning framework
• Demo
Deep Learning Introduction

- Deep Learning Setup
- Deep Learning Overview
- Convolutional Neural Networks
- Applications
Deep Learning Setup

Training data

Training algorithm

Model

Prediction

Evaluate
Deep Learning Setup

Input data -> Trained Model -> Prediction

Copyright ASIC Design Services 2018
Deep Learning Overview

Traditional Image Processing Pipeline

Input

Hand-Crafted
SIFT, HOG, Gabor
Filters etc.

Feature Extractor

Trainable
Classifier

Output

Pedestrian
Car
Animal
Road

Deep Learning

Input

Trainable
Convolutional
Layers with
optional pooling
and activation
functions

Feature Extractor

Trainable
Classifier

Output

Pedestrian
Car
Animal
Road

Deep Learning Introduction

- Traditionally hand-crafted features
  - Time consuming design
  - Application Specific

- Deep Learning
  - Feature Learning
  - Trainable Feature Extractor
  - Requires lots of training data

- Became viable with improvement in
  - Improved Training Techniques
  - Availability of Training Data
  - Improved processing power

- Trainable Classifier generally used
Convolutional Neural Network

Input → Feature Extractor → Classifier → Output

Convolution Layer → Convolution Layer → Convolution Layer → Fully Connected Layer → Fully Connected Layer → Fully Connected Layer

Layer Input → Convolution Output → Subsampled Outputs

Multichannel Convolution → Subsampling (Pooling)
Feature Extractor

Input

Multichannel Convolution

Output of Preceding Layer is used as input

Non-Linear Activation Function

Activation function applied independently to each element

Feature Maps

Subsampling

Pooling

Output is used as input to the following layer

Copyright ASIC Design Services 2018
Applications

• Applications:
  • Consumer
  • Defence
  • Industrial
  • Medical
  • Surveillance

Deep Learning Introduction

Pose Estimation
Cao et al, 2017

Target Detection and Classification in SAR
Chen et al. 2016

Crowd Segmentation
Kang and Wang, 2014

Depth from Monocular Images
Lui et al, 2015

Copyright ASIC Design Services 2018
Training Stage

- Large Datasets
- Machine learning expert
- Optimization Algorithm

CNN

- Platform: Local GPUs or GPUs in the cloud
- Has required parallel processing power and memory for efficient training

Inference Stage

- Actual use of CNN in the field (deployment)

- Can continue to use CPUs or GPUs here
  - CPU - Inefficient and slow with CNNs
  - GPU – Large initial power budget, still general purpose, large space footprint, require control CPU

- **We suggest FPGAs** – low power, small, network specific optimised solution
Convolutional Neural Networks on FPGAs

- Why FPGAs
- Applications
- Complexity analysis
- Challenges
Why FPGAs

- Total solution size/footprint
- Flexibility
- Low-power
- Deep Learning algorithms running alongside other SOFT IP cores
- Security

Convolutional Neural Networks on FPGAs

Deep Learning
Applications

Convolutional Neural Networks on FPGAs

Moving the processing to the node

Solutions can be packaged into battery-operated consumer products

Copyright ASIC Design Services 2018
Complexity analysis

Convolutional Neural Networks on FPGAs

You Only Look Once: Redmon et al, 2016

Convolution layers

Fully connected layers
$\text{Conv}_{\text{time}} = O(N \times M \times K^2 \times R \times C)$

$\text{Pool}_{\text{time}} = O(N \times R \times C)$

$\text{Conv}_{\text{space}} = O(N \times M \times K^2)$
\[ FC_{time} = O(N \times M) \]

\[ FC_{space} = O(N \times M) \]
Complexity analysis

Convolutional Neural Networks on FPGAs

Convolution layers

You Only Look Once: Redmon et al, 2016

Fully connected layers

Copyright ASIC Design Services 2018
Complexity analysis

Convolutional Neural Networks on FPGAs

Layer time complexity

Layer space complexity

Copyright ASIC Design Services 2018
Challenges

Convolutional Neural Network Challenges
- Computational-intensive
- Frequent memory access
- Difficult to deploy on custom hardware platforms

FPGA limitations
- BRAM
- DSP resources
- Logic elements
- External memory bandwidth
Addressing the Computation and Memory Problem

- Sources of parallelism
- Tiling
- Loop optimizations
- Optimal math block configurations
- Quantization
- Double buffering
Sources of Parallelism

Kernel level parallelism

Weight set 1  Weight set 2  Weight set 3

Input feature maps

Output feature maps

Addressing the Computation and Memory Problem

S*C+K-S

S*R+K-S

N

M

R

K

K

Copyright ASIC Design Services 2018
Sources of Parallelism

Input feature map parallelism

Weight set 1  Weight set 2  Weight set 3

Input feature maps

Output feature maps

S*C+K-S

S*R+K-S

K

K

Copyight ASIC Design Services 2018
Addressing the Computation and Memory Problem

Sources of Parallelism

Output feature map parallelism

\[ S*C+K-S \]

Weight set 1  Weight set 2  Weight set 3

Input feature maps

Copyright ASIC Design Services 2018
Tiling

Addressing the Computation and Memory Problem

```
for (row=0; row<R; row++) {
    for (col=0, col<C; col++) {
        for (to=0; to<M; to++) {
            for (ti=0; ti<N; ti++) {
                for (i=0; i<K; i++) {
                    for (j=0; j<K; j++) {
                        output_fm[to][row][col] += weights[to][ti][i][j] * input_fm[ti][S*row+i][S*col+j]
                    }
                }
            }
        }
    }
}
```
Tiling

Addressing the Computation and Memory Problem

```c
for (row=0; row<R; row+=Tr) {
    for (col=0, col<C; col+=Tc) {
        for (to=0; to<M; to+=Tm) {
            for (ti=0; ti<N; ti+=Tn) {
                // Load Output feature maps
                // Load weights
                // Load input feature maps
                for (trr=row; trr<min(row+Tr, R); trr++) {
                    for (tcc=col; tcc<min(col+Tc, C); tcc++) {
                        for (too=to; too<min(to+Tm, M); too++) {
                            for (tii=ti; tii<min(ti+Tn, N); tii++) {
                                for (i=0; i<K; i++) {
                                    for (j=0; j<K; j++) {
                                        output_fm[too][trr][tcc] += weights[too][tii][i][j]*input_fm[tii][S*trr+i][S*tcc+j];
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

//Store Output feature maps

On chip data computation

External data transfer

Copyright ASIC Design Services 2018
Loop Pipelining and Unrolling

Addressing the Computation and Memory Problem

```
for (row=0; row<R; row+=Tr) {
    for (col=0, col<C; col+=Tc){
        for (to=0; to<M; to+=Tm){
            for (ti=0; ti<N; ti+=Tn){
                //Load Output feature maps
                //Load weights
                //Load input feature maps
                for (i=0; i<K; i++){  
                    for (j=0; j<K; j++){  
                        for (trr=row; trr<min(row+Tr,R); trr++){  
                            for (tcc=col; tcc<min(col+Tc,C); tcc++){  
                                for (too=to; too<min(to+Tm,M); too++){  
                                    for (tii=ti; tii<min(ti+N,N); tii++){  
                                        output_fm[too][trr][tcc] += weights[too][tii][i][j]*input_fm[tii][S*trr+i][S*tcc+j]  
                                    }  
                                }  
                            }  
                        }  
                    }  
                }  
            }  
        }  
    }  
}  

//Store Output feature maps
```
Addressing the Computation and Memory Problem

Computing Engine

Processing Element

MAC

Adder Tree

Weights

Data

Bias

Computing Engine

PE

PE

PE

Copyright ASIC Design Services 2018
DOT Product Mode

- Convolution implemented as a large number of multiply-accumulate operations
- Microsemi Math blocks can implement a DOTP mode
- Two multiply operations and an addition operation in a single clock cycle

\[ P = (B[8:0] \times A[17:9]) + (B[17:9] \times A[8:0]) \]
Cascading MACC units

Addressing the Computation and Memory Problem

$P = (B[8:0] \times A[17:9]) + (B[17:9] \times A[8:0]) + C$

Copyright ASIC Design Services 2018
Data quantisation

- High-precision multiply-accumulate
- Use dynamic fixed point per layer

- Significant bits selected through analysis of network using representative test set

- High-precision source networks
- Networks are retrained for lower precision to regain close to original performance
### Accuracy on a variety of datasets (Gysel, 2016)

<table>
<thead>
<tr>
<th>Network</th>
<th>Floating-point</th>
<th>8-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet</td>
<td>99.1</td>
<td>99.1</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>81.7</td>
<td>81.4</td>
</tr>
<tr>
<td>CaffeNet</td>
<td>56.9</td>
<td>56.0</td>
</tr>
</tbody>
</table>

### Top5 Accuracy in ImageNet (Guo et al, 2016)

<table>
<thead>
<tr>
<th>Network</th>
<th>Floating-point</th>
<th>8-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaffeNet</td>
<td>77.12</td>
<td>76.64</td>
</tr>
<tr>
<td>VGG16</td>
<td>88.10</td>
<td>87.60</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>88.82</td>
<td>88.64</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>79.72</td>
<td>79.16</td>
</tr>
</tbody>
</table>

### Accuracy on a variety of networks/applications (own work)

<table>
<thead>
<tr>
<th>Network</th>
<th>Floating-point</th>
<th>8-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet</td>
<td>99.12</td>
<td>99.13</td>
</tr>
<tr>
<td>Scene Labelling</td>
<td>73.89</td>
<td>73.31</td>
</tr>
<tr>
<td>VGG16</td>
<td>88.44</td>
<td>87.54</td>
</tr>
</tbody>
</table>
Local Memory Promotion

Addressing the Computation and Memory Problem

```c
for (row=0; row<R; row+=Tr) {
    for (col=0, col<C; col+=Tc){
        for (to=0; to<M; to+=Tm){
            for (ti=0; ti<N; ti+=Tn){
                //Load Output feature maps
                //Load weights
                //Load input feature maps
                for (i=0; i<K; i++){
                    for (j=0; j<K; j++){
                        for (trr=row; trr<min(row+Tr,R); trr++){
                            for (tcc=col; tcc<min(col+Tc,C); tcc++){
                                for (too=to; too<min(to+Tm,M); too++){
                                    for (tii=ti; tii<min(ti+Tn,N); tii++){
                                        output_fm[too][trr][tcc] += weights[too][tii][i][j]*input_fm[tii][S*trr+i] [S*tcc+j]
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```
Local Memory Promotion

Addressing the Computation and Memory Problem

for (row=0; row<R; row+=Tr) {
    for (col=0, col<C; col+=Tc) {
        for (to=0; to<M; to+=Tm) {
            for (ti=0; ti<N; ti+=Tn) {
                // Load Output feature maps
                // Load weights
                // Load input feature maps
                for (i=0; i<K; i++) {
                    for (j=0; j<K; j++) {
                        for (trr=row; trr<min(row+Tr, R); trr++) {
                            for (tcc=col; tcc<min(col+Tc, C); tcc++) {
                                for (too=to; too<min(to+Tm, M); too++) {
                                    for (tii=ti; tii<min(ti+Tn, N); tii++) {
                                        output_fm[too][trr][tcc] += weights[too][tii][i][j]*input_fm[tii][S*trr+i][S*tcc+j]
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

// Store Output feature maps

Copyright ASIC Design Services 2018
Double buffering

Addressing the Computation and Memory Problem

Ping-pong operation

<table>
<thead>
<tr>
<th>Load</th>
<th>Compute</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input buff 1</td>
<td>Input buff 0</td>
<td>Output buff 1</td>
</tr>
<tr>
<td>Input buff 0</td>
<td>Input buff 1</td>
<td>Output buff 0</td>
</tr>
<tr>
<td>Input buff 0</td>
<td>Input buff 1</td>
<td>Input buff 0</td>
</tr>
<tr>
<td>Input buff 1</td>
<td>Input buff 0</td>
<td>Output buff 1</td>
</tr>
</tbody>
</table>

Copyright ASIC Design Services 2018
Design Space Exploration

- Roofline model
- Design search
• Implementation can either be computation-bounded or memory-bounded
• Model performance to off-chip memory traffic

\[ \text{Att Perf} = \min \left\{ \frac{\text{Computational roof}}{\text{CTC ratio}} \times \text{BW} \right\} \]
Design Space Exploration

Design search

Optimal platform design parameters

\[ \text{Comp roof} = \frac{\text{# of operation}}{\text{# of execution cycles}} \]

\[ \text{CTC} = \frac{\text{# of operation}}{\text{# of external data access}} \]
Core generator features

- Full pipeline from convolutional neural network description to FPGA implementation
  - We only need the target platform or resource availability and the network architecture
- Network retraining for memory footprint minimisation
- Support for different network layers
  - Convolutional layer
  - Fully connected layer
  - Pooling layer
  - Activation layers
- Convolutional layers can implement filters of any size and stride
- Pooling layers supporting arbitrary kernel size
- Support for padding
- AXI memory interface for external RAM
Core Interface

Core Deep Learning
an embedded FPGA solution

Control IP

External Memory

APB Interface

AXI Interface

Input Buffers

Output Buffers

Controllers

Microcode

Computing Engine

PE

PE

.....

PE

Copyright ASIC Design Services 2018
User specified

- Platform (M2S090, MPF300T ...)
- Platform resources available for deep learning solution
  - MACC units
  - LSRAM memory blocks
  - uSRAM memory blocks
- Available memory bandwidth
Tiny-YOLOv2

- Fully Convolutional Neural Network - 9 Convolutional Layers
  - convolution operation + batch normalisation + activation + pooling
- Trained end-to-end on Pascal VOC dataset
- Quantized and finetuned from provided base network by Joseph Redmon
  - Tiny YOLO @ https://pjreddie.com/darknet/yolo/
- 5 fps on Microsemi M2S090
## Tiny-YOLOv2

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input image shape</td>
<td>416 x 416</td>
</tr>
<tr>
<td>Number of convolutional layers</td>
<td>9</td>
</tr>
<tr>
<td>Number of fully connected layers</td>
<td>0</td>
</tr>
<tr>
<td>GOPs (MULACC)</td>
<td>7</td>
</tr>
<tr>
<td>Runtime (ms)</td>
<td>216</td>
</tr>
<tr>
<td>Performance [GOPs/s]</td>
<td>32</td>
</tr>
<tr>
<td>Efficiency [GOPs/s/W]</td>
<td>18.82</td>
</tr>
<tr>
<td>Multiplier Efficiency [GOPs/s/Slice*]</td>
<td>0.381</td>
</tr>
</tbody>
</table>

*Slice – DSP Slice/Math Block
<table>
<thead>
<tr>
<th>4LUT</th>
<th>41131</th>
<th>48%</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFF</td>
<td>48310</td>
<td>56%</td>
</tr>
<tr>
<td>RAM64x18</td>
<td>72</td>
<td>64%</td>
</tr>
<tr>
<td>RAM1K18</td>
<td>72</td>
<td>66%</td>
</tr>
<tr>
<td>MACC</td>
<td>74</td>
<td>88%</td>
</tr>
</tbody>
</table>
## Tiny-YOLOv2 on Microsemi PolarFire

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input image shape</td>
<td>416 x 416</td>
</tr>
<tr>
<td>Number of convolutional layers</td>
<td>9</td>
</tr>
<tr>
<td>Number of fully connected layers</td>
<td>0</td>
</tr>
<tr>
<td>GOPs (MULACC)</td>
<td>7</td>
</tr>
<tr>
<td>Runtime (ms)</td>
<td>28</td>
</tr>
<tr>
<td>Performance [GOPs/s]</td>
<td>245</td>
</tr>
<tr>
<td>Efficiency [GOPs/s/W]</td>
<td>74.24</td>
</tr>
<tr>
<td>Multiplier Efficiency [GOPs/s/Slice*]</td>
<td>0.339</td>
</tr>
</tbody>
</table>

*Slice – DSP Slice/Math Block

---

Copyright ASIC Design Services 2018
## Tiny-YOLOv2 on Microsemi PolarFire

<table>
<thead>
<tr>
<th>Component</th>
<th>Usage</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4LUT</td>
<td>70039</td>
<td>23%</td>
</tr>
<tr>
<td>DFF</td>
<td>98703</td>
<td>33%</td>
</tr>
<tr>
<td>uSRAM (64x12)</td>
<td>1440</td>
<td>52%</td>
</tr>
<tr>
<td>LSRAM (20 k bit)</td>
<td>602</td>
<td>63%</td>
</tr>
<tr>
<td>MACC</td>
<td>723</td>
<td>78%</td>
</tr>
</tbody>
</table>

Copyright ASIC Design Services 2018