Designing Bespoke CNNs for Target Hardware

Woonhyun Nam
Director, Algorithms
StradVision
September, 2020
• We develop deep learning-based perception software for Advanced Driver Assistance Systems (ADAS) and Autonomous Vehicles

For more detailed information, please refer to https://youtu.be/tK7F8yvpiGA and https://stradvision.com/
Agenda

• Development Process of Deep Learning-based Software
• Scalability Challenge
• Layer Transformations
• Case Studies
• Conclusions
Development Process of Deep Learning-based Software
Development Process

Repeated for each task (e.g., object detection) and target platform

(Pros)
• **Reuse** existing CNN designs
• **Tweak** some layers

(Cons)
• **Retrain from scratch** even for small changes
• **Revalidate from scratch** if output changes
Scalability Challenge
Scalability challenge

- Number of networks to maintain grows rapidly
- More than 100 networks become difficult to manage

# of edges = # of networks
Main idea

- Grouping target platforms based on similar computational characteristics (or capabilities) with respect to a given CNN transformation

![Diagram showing task to target platform relationship with number of edges equal to number of networks and same group highlighted.]

© 2020 StradVision
Scalable Development Process

- Design/Train CNNs for each group
- Transform CNNs without retraining within same group
Scalable Development Process

Cost-effective method

• Increasing computational efficiency
  • E.g., lower latency, higher throughput, cheaper hardware

• Increasing engineering efficiency
  • E.g., shorter development time, less engineers, more projects

• Increasing economic efficiency
  • E.g., lower usage of cloud services for training, lower production-cost
Layer Transformations
Layer Transformations

- Basic operations
- Case studies
  - YUV Input
  - Fixed-size Kernel Acceleration in Convolution
  - No FC Support
  - No Custom Layer Support in Quantization
  - Block-level Accumulation in Convolution
Layer Transformations

Basic operations

• (Linear) layer fusion
  • Conv & Conv → Conv

• Memory layout change
  • For Input, Weight, and Output tensors
  • Reshape, Transpose

• Zero-filling

\[ y = ax + b \quad \text{and} \quad z = cy + d \]

\[ z = (ac)x + (bc + d) \]

New weight \quad New bias
Case Studies
Hypothetical target hardware

- YUV input, instead of RGB
- YUV2RGB equation is given

→ Fusion of YUV2RGB and the first Conv

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} = \begin{bmatrix}
w_{RY} & w_{RU} & w_{RV} \\
w_{GY} & w_{GU} & w_{GV} \\
w_{BY} & w_{BU} & w_{BV}
\end{bmatrix} \begin{bmatrix}
Y + b_{Y} \\
U + b_{U} \\
V + b_{V}
\end{bmatrix}
\]

YUV2RGB equation
Equations for new weights and new bias

\[ Z = \begin{bmatrix} w_R & w_G & w_B \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + b \]

Conv1 equation

\[ \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} w_{RY} & w_{RU} & w_{RV} \\ w_{GY} & w_{GU} & w_{GV} \\ w_{BY} & w_{BU} & w_{BV} \end{bmatrix} \begin{bmatrix} Y + b_Y \\ U + b_U \\ V + b_V \end{bmatrix} \]

YUV2RGB equation

\[ Z = \begin{bmatrix} w_R & w_G & w_B \end{bmatrix} \begin{bmatrix} w_{RY} & w_{RU} & w_{RV} \\ w_{GY} & w_{GU} & w_{GV} \\ w_{BY} & w_{BU} & w_{BV} \end{bmatrix} \begin{bmatrix} Y + b_Y \\ U + b_U \\ V + b_V \end{bmatrix} + b + \begin{bmatrix} w_R & w_G & w_B \end{bmatrix} \begin{bmatrix} w_{RY} & w_{RU} & w_{RV} \\ w_{GY} & w_{GU} & w_{GV} \\ w_{BY} & w_{BU} & w_{BV} \end{bmatrix} \begin{bmatrix} b_Y \\ b_U \\ b_V \end{bmatrix} \]

New Conv1 equation

New weight

New bias
Hypothetical target hardware

- Only 5x5 kernel acceleration supported
- Applying zero-filling for smaller kernels
- Computational utilization
  - 1x1 Conv ~ 4%, 3x3 Conv ~ 36%
Case Study – Fixed-size Kernel Acceleration in Convolution

→ Convert 1x1 Conv into 4(kh)x1(kw) Conv

<table>
<thead>
<tr>
<th>Input</th>
<th>Weight</th>
<th>Output reshape</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ic)x(ih)x(iw)</td>
<td>(oc)x(ic)x(1)x(1)</td>
<td>(oc)x(oh)x(ow)</td>
</tr>
</tbody>
</table>

Reshape

Input reshape (ic/4,4,ih*iw) (ic/4)x(4)x(ih*iw)

Weight reshape (oc,ic/4,4,1) (oc)x(ic/4)x(4)x(1)

Output (oc)x(1)x(oh*ow)
Case Study – Fixed-size Kernel Acceleration in Convolution

→ Convert 1x1 Conv into 4(kh)x1(kw) Conv

Reshape

Input

Reshape

Weight

Reshape

Output

iw=3
ih=2
ic=4

1 2 3
4 5 6
10 11 12
16 17 18
22 23 24

Reshape

1w′=ih\times iw=6

Reshape

1h′=ic=4

Reshape

1c′=ic/4=1

Reshape

1ow′=iw′=6

Reshape

1oh′=1

iw=input width
ih=input height
ic=input channel
kw=kernel width
kh=kernel height
ow=output width
oh=output height
oc=output channel
Hypothetical target hardware

- No support for FC layers, or
- Only single-batched FC layers supported

Matrix Multiplication in FC layer

<table>
<thead>
<tr>
<th>batch=2</th>
<th>ic=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>4 5 6</td>
</tr>
</tbody>
</table>

* 

<table>
<thead>
<tr>
<th>oc=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4</td>
</tr>
<tr>
<td>2 5</td>
</tr>
<tr>
<td>3 6</td>
</tr>
</tbody>
</table>

= 

<table>
<thead>
<tr>
<th>batch=2</th>
<th>oc=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td></td>
</tr>
<tr>
<td>3 4</td>
<td></td>
</tr>
</tbody>
</table>
→ Convert FC into 1x1 Conv

\[
\text{Input (batch) x (ic)} \times \text{Weight (ic) x (oc)} = \text{Output (batch) x (oc)}
\]

\[
\begin{align*}
\text{Input: } & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} & \text{Weight: } & \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} & \text{Output: } & \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}
\end{align*}
\]

\[
\text{Transpose (Input)} \times \text{Reshape (ic, 1, batch)} \times \text{Transpose (Weight)} \times \text{Reshape (oc, ic, 1, 1)} = \text{Transpose (Output)} \times \text{Reshape (oc, 1, batch)}
\]

\[
\begin{align*}
\text{Input: } & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 4 \\ 5 & 6 & 1 \end{bmatrix} & \text{Weight: } & \begin{bmatrix} 4 & 2 \\ 5 & 3 \\ 6 & 1 \end{bmatrix} & \text{Output: } & \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{Transpose (Input)} \times \text{Reshape (ic, 1, batch)} \times \text{Transpose (Weight)} \times \text{Reshape (oc, ic, 1, 1)} = \text{Transpose (Output)} \times \text{Reshape (oc, 1, batch)}
\end{align*}
\]
Case Study – No Custom Layer Support in Quantization

Hypothetical target hardware

• Quantization should be done by a tool provided by hardware vendor
• No support for custom layers
• The tool accepts only image files as quantization inputs

→ Convert input tensors of custom layers into image files
• Useful for two-stage detection networks (E.g., Faster RCNN)
Case Study – No Custom Layer Support in Quantization

Convert 4D tensors into image files

Input (batch)x(c)x(h)x(w)

```
+-----+-----+-----+-----+
|     |     |     |     |
|     |     |     |     |
| 1   | 2   | 3   | 9   |
| 4   | 5   | 6   | 15  |
| 10  | 11  | 12  | 21  |
| 16  | 17  | 18  | 23  |
| 22  | 23  | 24  |     |
+-----+-----+-----+-----+

w=3 h=2 c=4 batch=2
```

2(batch)x4(c)x2(h)x3(w)

Tensor in Int8 format

Input reshape(batch*c,h*w) + 128

```
[1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48]
```

8(h’)x6(w’)

Gray image in Uint8 format
Case Study – Block-level Accumulation in Convolution

Hypothetical target hardware

• Accumulating intermediate results of Conv in 8(oc)x8(ic) channels

• For example,

\[
\text{Conv (Input } \begin{array}{c} 24(\text{ic})\times32(\text{ih})\times32(\text{iw}) \end{array}, \text{ Weight } \begin{array}{c} 8(\text{oc})\times24(\text{ic})\times1(\text{kh})\times1(\text{kw}) \end{array})
\]

\[
= \text{Conv}^{8(\text{oc})x8(\text{ic})} (\text{Input}[0:8,:,:), \text{ Weight}[;0:8,:,:,])
\]

\[
+ \text{Conv}^{8(\text{oc})x8(\text{ic})} (\text{Input}[8:16,:,:), \text{ Weight}[;8:16,:,:,])
\]

\[
+ \text{Conv}^{8(\text{oc})x8(\text{ic})} (\text{Input}[16:24,:,:), \text{ Weight}[;16:24,:,:,])
\]
Hypothetical target hardware

- Accumulating intermediate results of Conv in 8(oc)x8(ic) channels
- For example,
Case Study – Block-level Accumulation in Convolution

→ 8(oc)x8(ic) block-level sparsification

\[
\text{Weight}_{\text{sparse}}(i\text{-th block}) = \begin{cases} 
0 & \text{if } \sum |\text{Weight}_{\text{dense}}(i\text{-th block})| < \theta \\
\text{Weight}_{\text{dense}}(i\text{-th block}) & \text{otherwise}
\end{cases}
\]

1x1 Conv Weight (dense)

1x1 Conv Weight (53% sparse)
Conclusions
Conclusions

• Layer transformation techniques can reduce production-cost especially for retraining of networks while supporting a diverse array of target platforms

• It may also be beneficial for deep learning hardware manufacturers to support a wide variety of networks not yet natively supported by their hardware
Resources

• Sparsification

  • “Exploring the Granularity of Sparsity in Convolution Neural Networks”, CVPR2017