A Cutting-Edge Memory Optimization Method for Embedded AI Accelerators

Arnaud Collard

Technical leader – Embedded AI

7 Sensing Software

Introduction to 7 Sensing Software

Our mission: develop AI based solutions for sensors

We develop sensing algorithms using machine learning

We possess comprehensive expertise spanning the entire development cycle, from cameras and sensors to deployment at the edge

Camera & Sensors

Utilize deep expertise across a diverse array of sensors, including **sensor modelling**

Data Generation

Apply advanced methods to acquire large datasets, including the **generation of synthetic human datasets**

AI Solutions

Create **advanced neural network** architectures to implement sensor fusion, multitasking…

Edge Deployment

Utilize both off-the-shelf and internal tools for deploying AI solutions on embedded systems

7 Sensing Software: application areas

AR/VR

- Eye-Tracking
- Depth Map Densification
- Spatial Light Source Estimation

Imaging

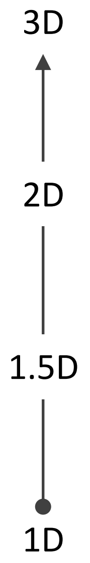
- ALS AWB
- AI-Accelerated Image Sensor

Human-Machine Interaction

- Gesture Recognition (dToF, PPG)
- Human Presence and Head Pose
- Optical Force Sensing

Vital Signs

- Respiratory Rate (at rest and with motion)
- Blood Pressure (PPG, ECG)

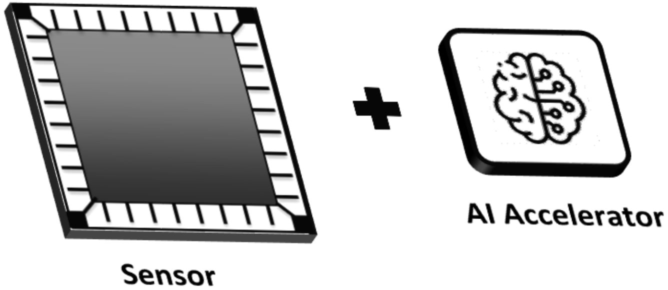


Sensor Data:
from time series up to RGB + depth (3D)

Creating smart sensors

Benefits and challenges of smart sensors

- There is value in integrating AI processing directly inside the sensor:
  - Avoiding the transfer of sensor data makes system design easier while enabling the reduction of overall power consumption

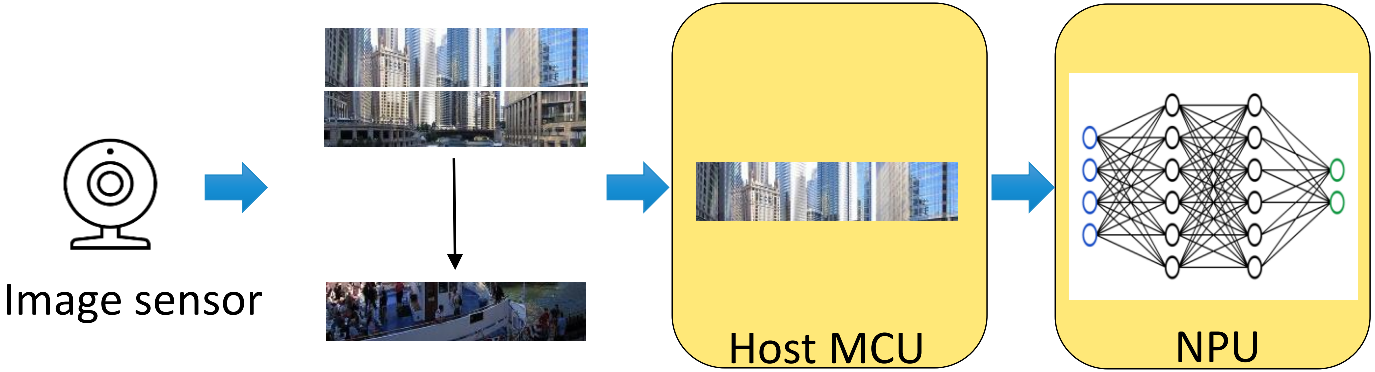


- However, such AI accelerator brings additional silicon area and therefore additional cost:
  - Optimizing for area, especially by reducing its memory footprint is of key interest
- We developed advanced AI methods to optimize AI processor integration in the case of an image sensor

Introduction to optimization method

Objectives and approach

- Two main objectives to address simultaneously
  - Reduce memory used by NPU to reduce cost and silicon area
  - Reduce latency between start of image acquisition and end of AI processing, and better load balancing of the AI accelerator (also called Neural Processing Unit (NPU))
- Our approach: Innovative optimization method allowing on-the-fly image acquisition and AI inference
  - Could be applied to a broader scope than sensor AI and image processing



© 2024 7 Sensing Software

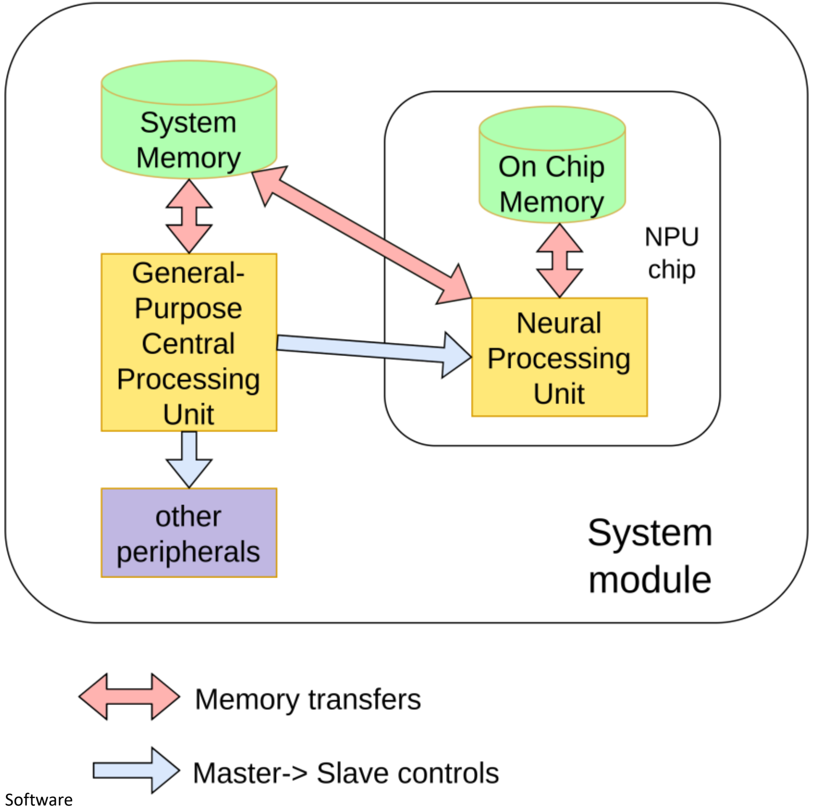
Main highlights of the method

- Designed to optimize AI on-chip memory (OCM) footprint and control the latency penalty (extra time to execute network on the AI accelerator)
- Relies on two well-known network optimization methods:
  - Processing by stripes
  - Processing by channels

  Complementary: both can be applied to the same model
- Implemented by a tool designed to explore network optimization parameters and find the best trade-off between memory footprint and latency
  - Stripes used at start of network, channels towards the end (last CNN layers)
- A patent application has been filed

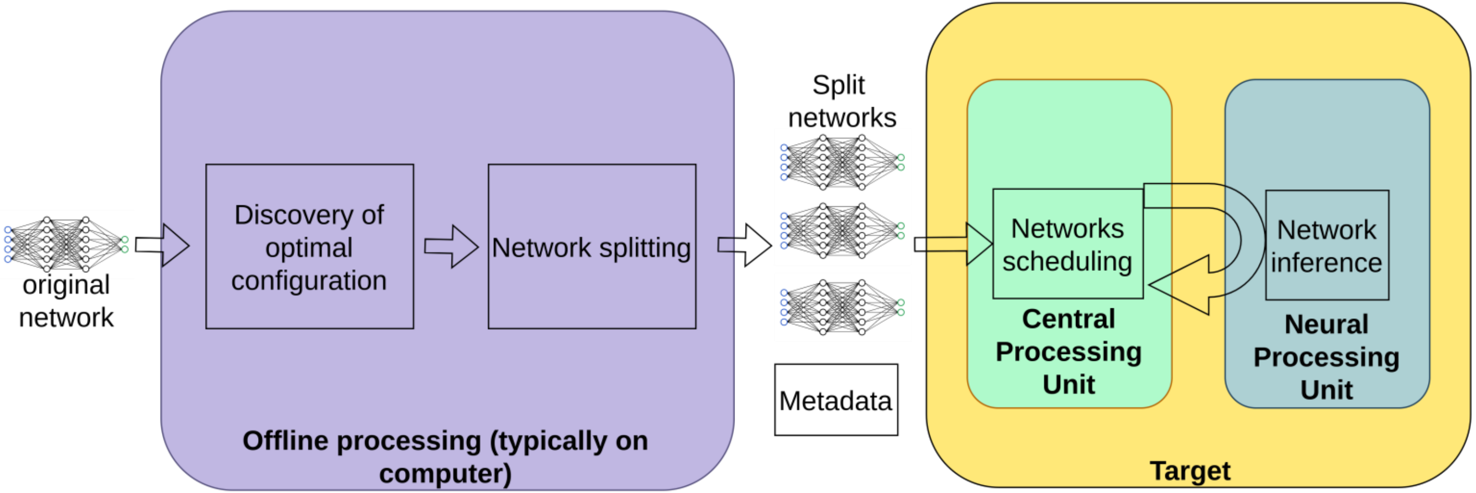
Targeted neural networks and architecture

- Neural networks
  - Focused on convolutional neural network applied to images
  - Could be extended to other network architectures and domains
- Embedded Neural Processing unit (AI accelerator)
  - Targets tiny embedded architectures with AI accelerator controlled by a general processing unit that schedules model inference
  - Both cores have their own memory
  - Priority is to optimize NPU on-chip memory (OCM)



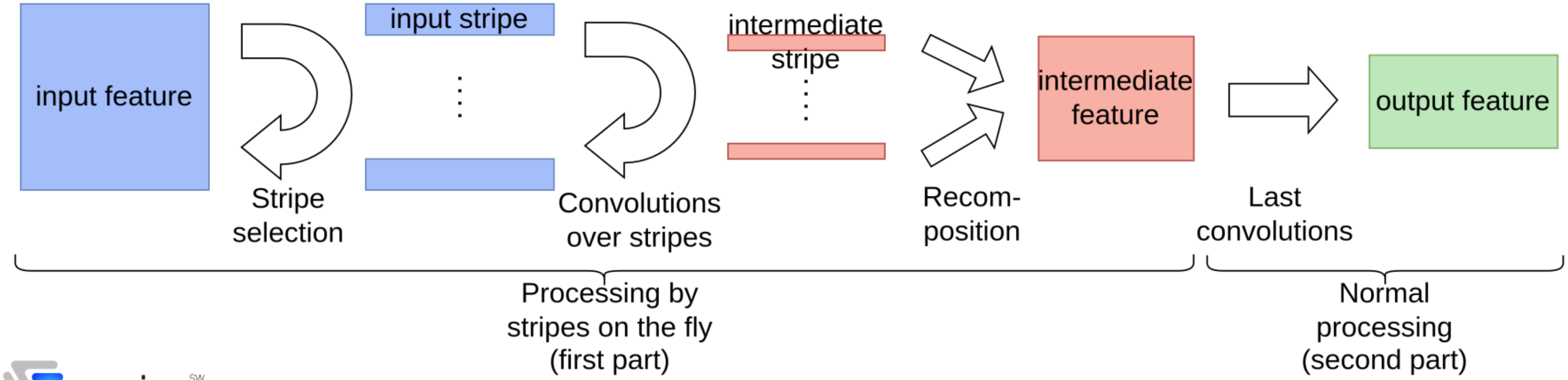
Process flow to split model and deploy on NPU

- General principle: the method splits an original network into multiple smaller networks with metadata to execute split models
  - Solution is NPU-agnostic



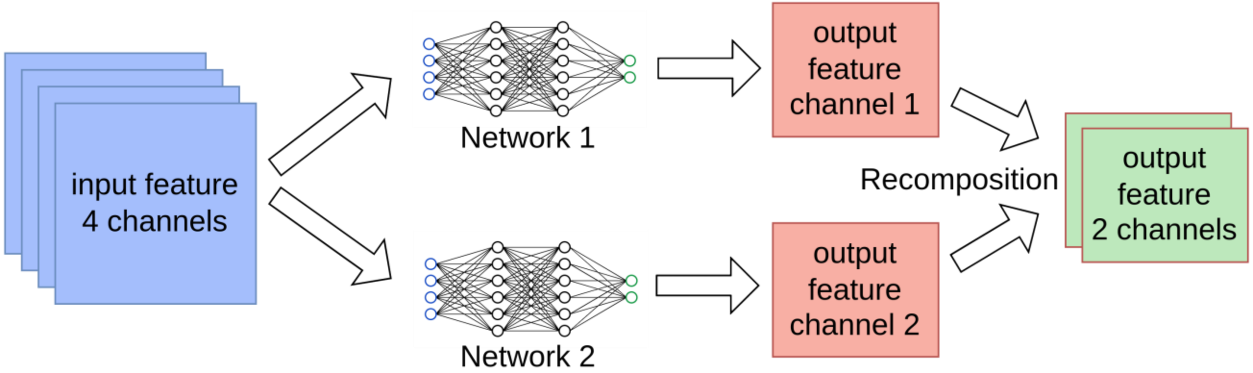
Processing by stripes

- Objective: reduces on-chip memory cost for input and output feature maps processed by CNN layers
- Number of stripes can decrease when moving forward in the network as feature map sizes decrease
  - Processes each stripe separately, can be processed by a varying number of cascaded layers
  - Various stripe configurations according to position in the network
- Allows on-the-fly processing as network input can be acquired stripe per stripe



Processing by channels

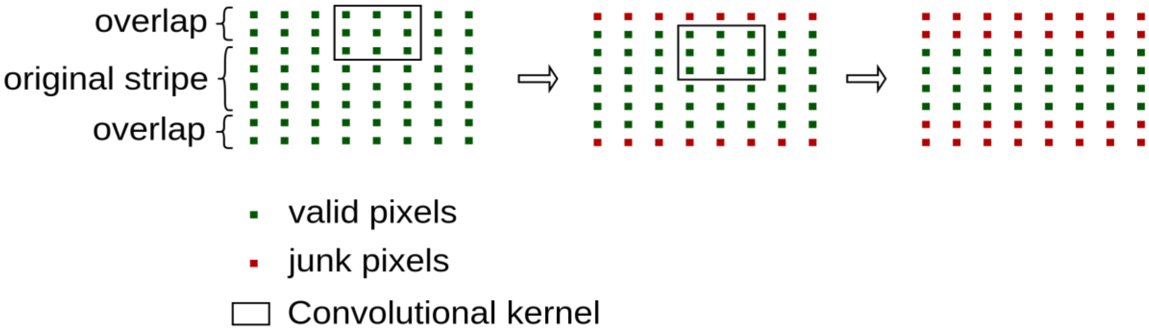
- Objective: reduces on-chip memory cost for weights of convolutional layers
- Cost for weights tends to increase when moving forward in the network
  - Applied only if processing by stripe is inactive at the end of network
- Divides a given convolutional layer by output channels
  - Several output channels may be grouped together for more efficiency



Addressing the challenges

Overlaps

- Applying convolutional kernel to feature map implies need for co-located data (overlap)
- Cascading convolutional layers implies cascading overlaps
  - increases drastically size of overlap for early layers and so latency
- Hard to determine size of overlap according to convolution parameters (stride, dilation, kernel size, padding) for all layers within a group
- Solution: implements 4-steps algorithm that resolves overlap configurations, remove junk data between split networks



overlap

original stripe

overlap

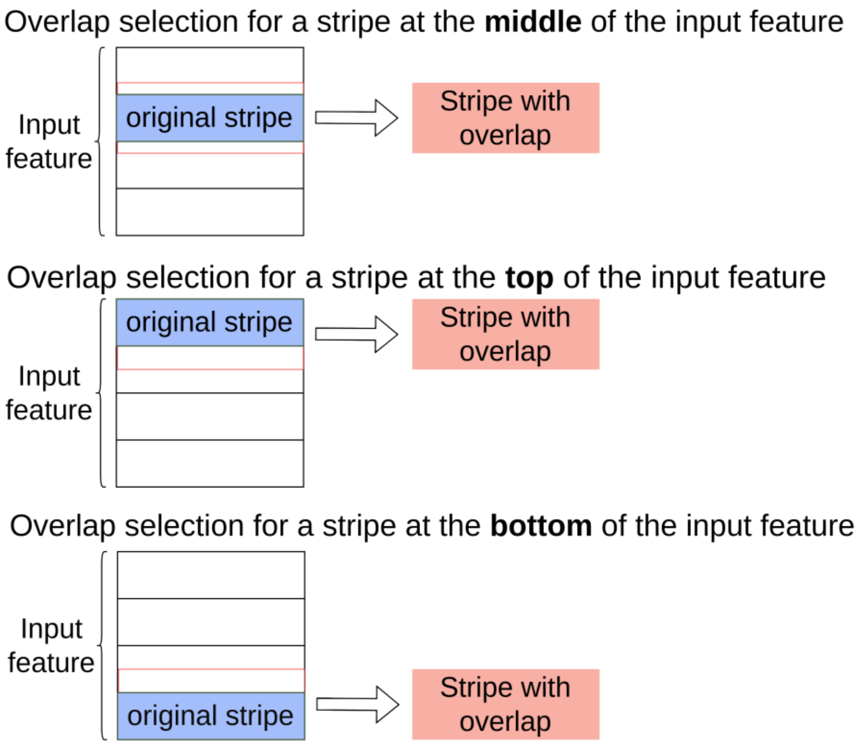
🟩 valid pixels

🟥 junk pixels

□ Convolutional kernel

Borders

- Problem: no co-located data available at the edge of the feature map
  - Breaks inference scheduling compared to normal stripe processing
  - Adding zero padding does not produce accurate results
- Solution: stick the overlap at the edge of the feature map (top or bottom) and rework stripe re-composition accordingly (eliminate duplicated data)



Overlap selection for a stripe at the **middle** of the input feature

Overlap selection for a stripe at the **top** of the input feature

Overlap selection for a stripe at the **bottom** of the input feature

Other challenges

- Huge number of configurations for processing by stripes and processing by channels
  - Solution: automatic algorithm for brute-force discovery on all stripes and channels configurations
    - Number of stripes only decreased when going through the network
    - Extending from stripes to tiles would be overkill (too complex and brings no value)
    - Coupling processing by stripes and processing by channel also overkill -> processing by channel only applied when stripe splitting not applied (end of network)
- Impacts on latency
  - Increase of MACs due to overlaps
  - Increase of memory transfers between CPU and NPU due to network splitting (by stripes and by channels)
  - Solution: simulates timing on NPU in the tool according to MACs and memory transfers, and takes maximum latency as an input parameter of the tool

Results

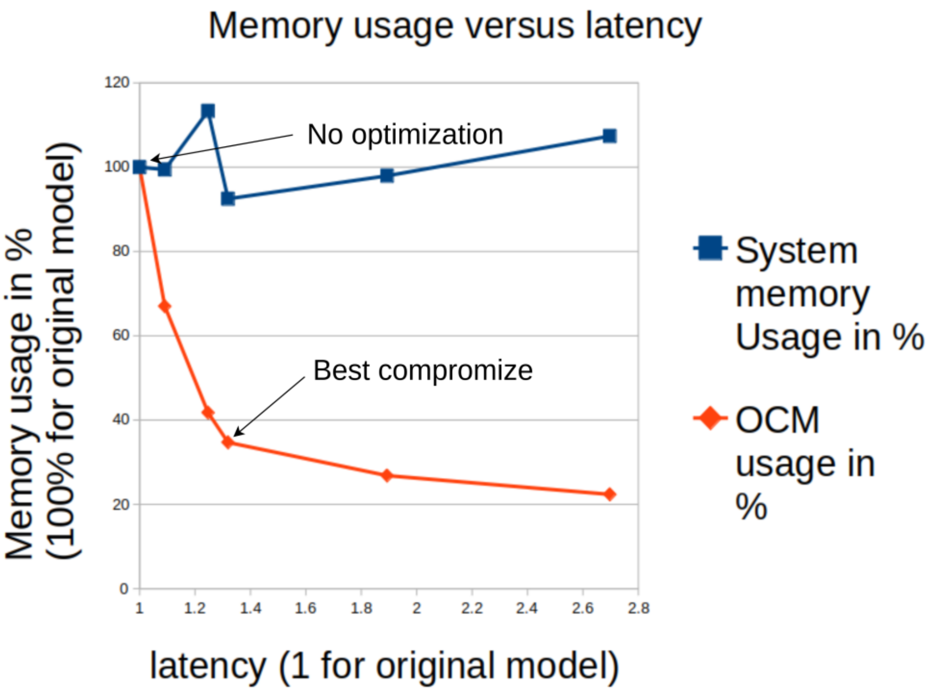
Face detection use case

- Basic face detector using MobileNet v1 without bounding box
- Best compromise
  - Memory footprint **reduced by a factor of 3** with reasonable impact on latency
- Validated on simulated hardware

	Original model	Optimized model	Gain
OCM (bytes)	301,200	104,512	-65.3%
System memory (bytes)	361,120	333,984	-7.5%
MACs	40,777,008	52,390,016	+28.5% overhead
Number of cycles	1,109,642	1,463,823	+31.9% overhead

Face detection use case – optimization exploration

- Various configurations applied to the face detector model to explore the balance between OCM and latency (1 for original model up to 3 times slower):
  - Huge reduction in OCM for an increase in latency up to 1.3x
  - Little additional gain in OCM for an increase of latency from 1.3x to 3x
  - Stable use of system memory



Other use cases

- Generalizes well on other network architectures
  - Good gains found for all models
- OCM reduction factor is dependent on network architecture and could be limited by:
  - Use of global operator (ex: global average pooling)
  - Number of skip connections

Model	Gain on OCM	Increase of cycles
Face segmentation	-37.5%	+2%
Face classification	-38.1%	+1.8%
Human detection	-62.2%	+11.8%
Face detection	-80.9%	+47.3%

Wrap up

Conclusions

- Significant decrease of on-chip memory footprint with reasonable impact on latency
- Automatic discovery of optimal configuration for various model architectures
- Proven with 5 different use cases
- Combining processing by stripes and processing by channels is a key point to significantly reduce OCM footprint
- NPU agnostic
- Internal tool, can be made available to customers in the context of a services project

Applying this method allows the integration of AI-processing capability in sensors at optimal cost

Additional resources

ams OSRAM

https://ams-osram.com

7 Sensing Software

https://7sensingsoftware.com/

My contact info

arnaud.collard@7sensingsoftware.com

https://www.linkedin.com/in/arnaudcollard/

2024 Embedded Vision Summit

Please meet us at booth 708 to learn how our expertise in sensors, synthetic data, AI algorithms and edge deployment can help you realize your product ambitions

THANK YOU!!