KI DELTA LEARNING
Scalable AI for Automated Driving

Analyzing and optimizing AI for embedded applications

Domenik Helms, Arunachalam Thirunavukkarasu, Adrian Osterwind

Embedded Systems

Goals

Develop methodologies to reduce the resource requirements on AI, when running on embedded Systems.

Prediction

Predict the hardware execution time of an AI early in the design and training phase of the AI so that the AI designer can optimize for these constraints.

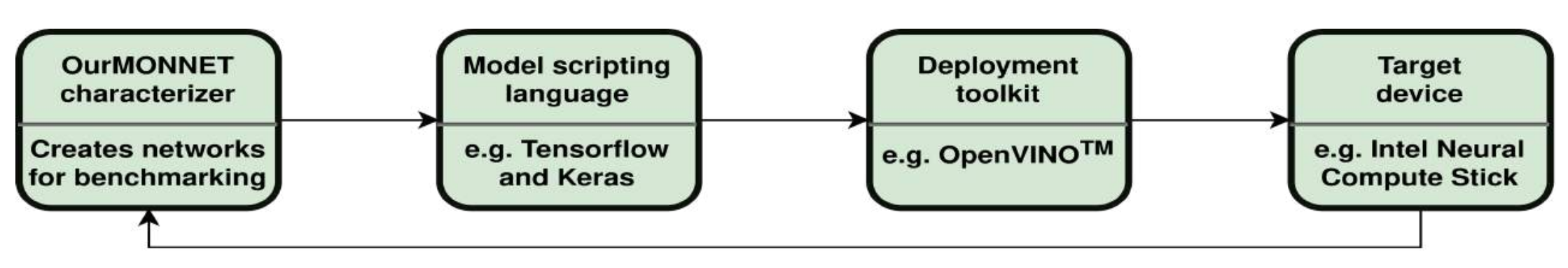

Figure 1: General Toolchain. First is the MONNET Characterizer (the result of this work), which creates a Tensorflow-Keras description of the neural network. This network is compiled using the OpenVINO toolchain and using this toolchain characterized on the Neural Compute Stick 2

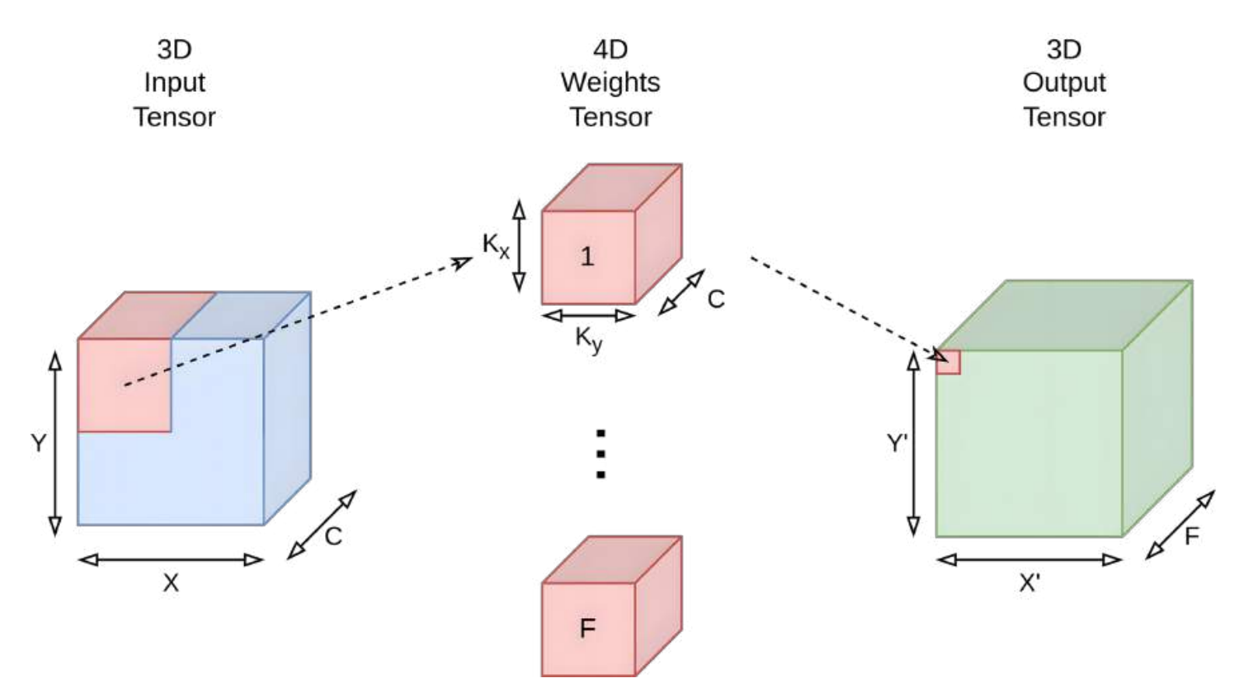

Figure 2: Description of the Convolutional 2D operation.

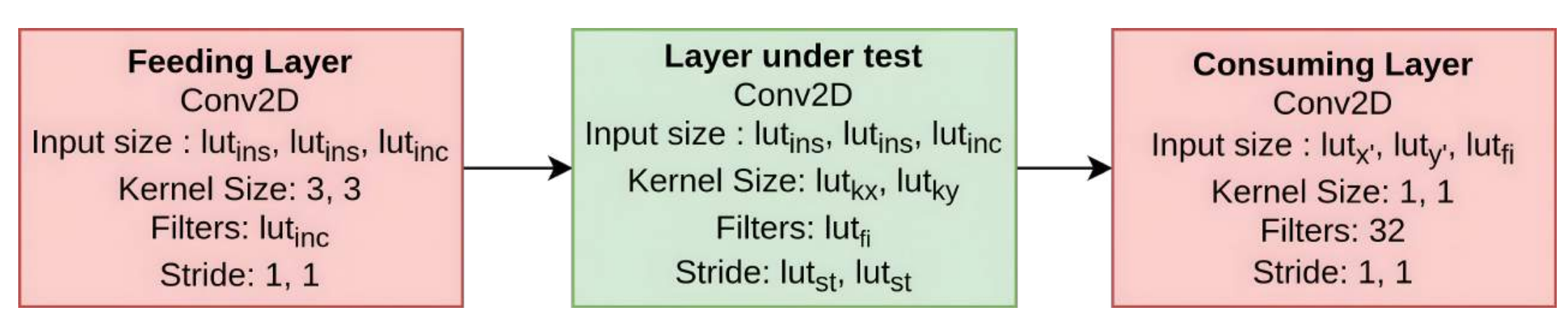
Feeding Layer
Conv2D
Input size : lut_{ins}, lut_{ins}, lut_{inc}
Kernel Size: 3, 3
Filters: lut_{inc}
Stride: 1, 1
Layer under test
Conv2D
Input size : lut_{ins}, lut_{ins}, lut_{inc}
Kernel Size: lut_{kx}, lut_{ky}
Filters: lut_{fi}
Stride: lut_{st}, lut_{st}
Consuming Layer
Conv2D
Input size : $lut_{x'}$, $lut_{y'}$, lut_{fi}
Kernel Size: 1, 1
Filters: 32
Stride: 1, 1

Figure 3: Preparation of the characterization process. A layer must be characterized without a „real“ network surrounding it. This leads to inaccuracies in the measured execution time due to optimizations and caching in the network compilation

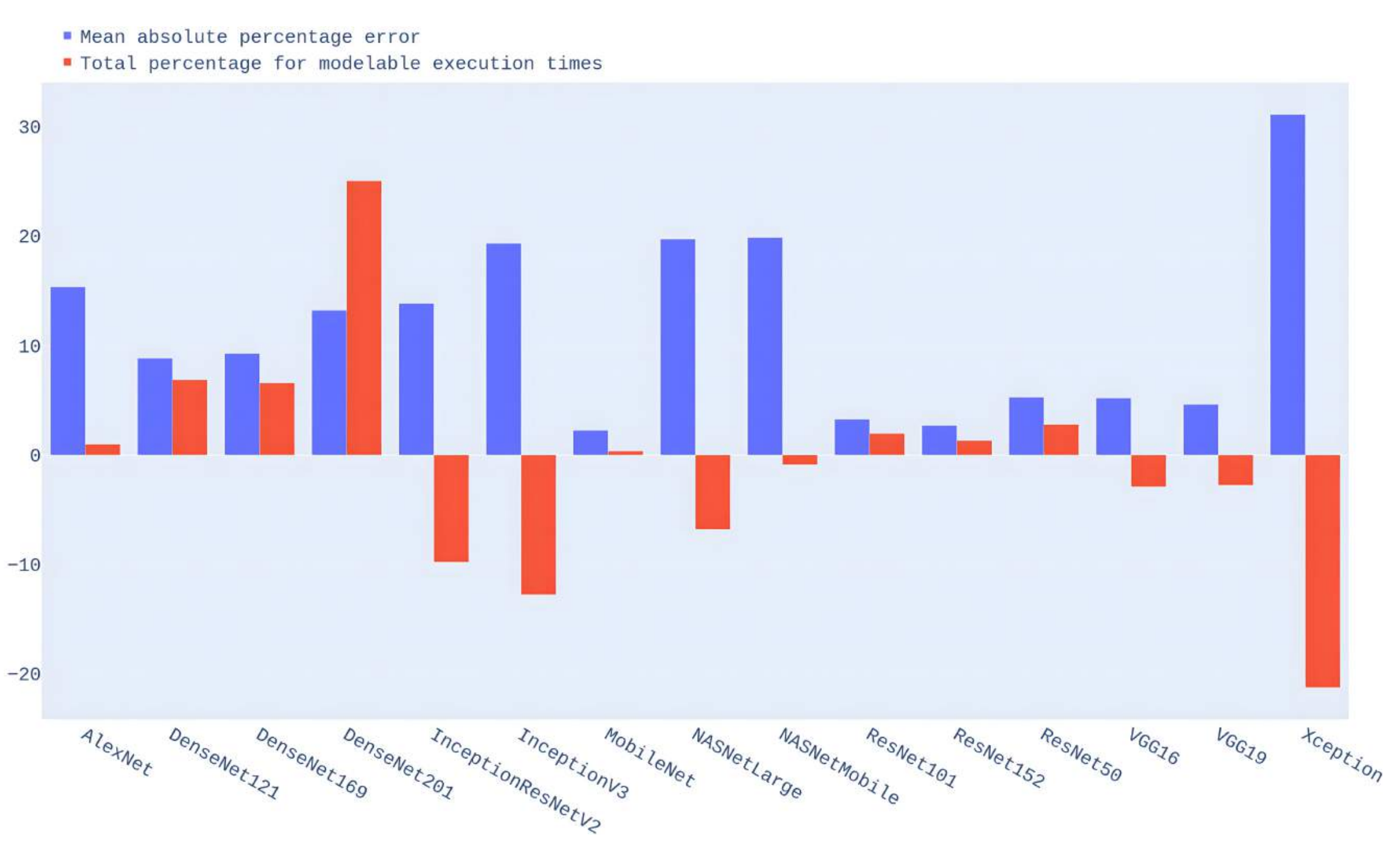

Figure 4: Result of the execution time prediction. Most results stay within 20% error. Exceptions are DenseNet201 and Xception. The former has a layer which repeats often within the network, which is badly characterized. The latter has many Separational Convolutional2D layers, which we characterized less than Convolutional2D layers

Results

A prediction methodology, returning quick and accurate hardware timing estimates directly inside Tensorflow or PyTorch. A methodology to reduce the size of the AI tensors with only a minimal impact onto the overall accuracy of the AI.

Optimization

Apply mathematical transformations to the tensors of the AI layers to reduce their size and thus memory footprint and execution time.

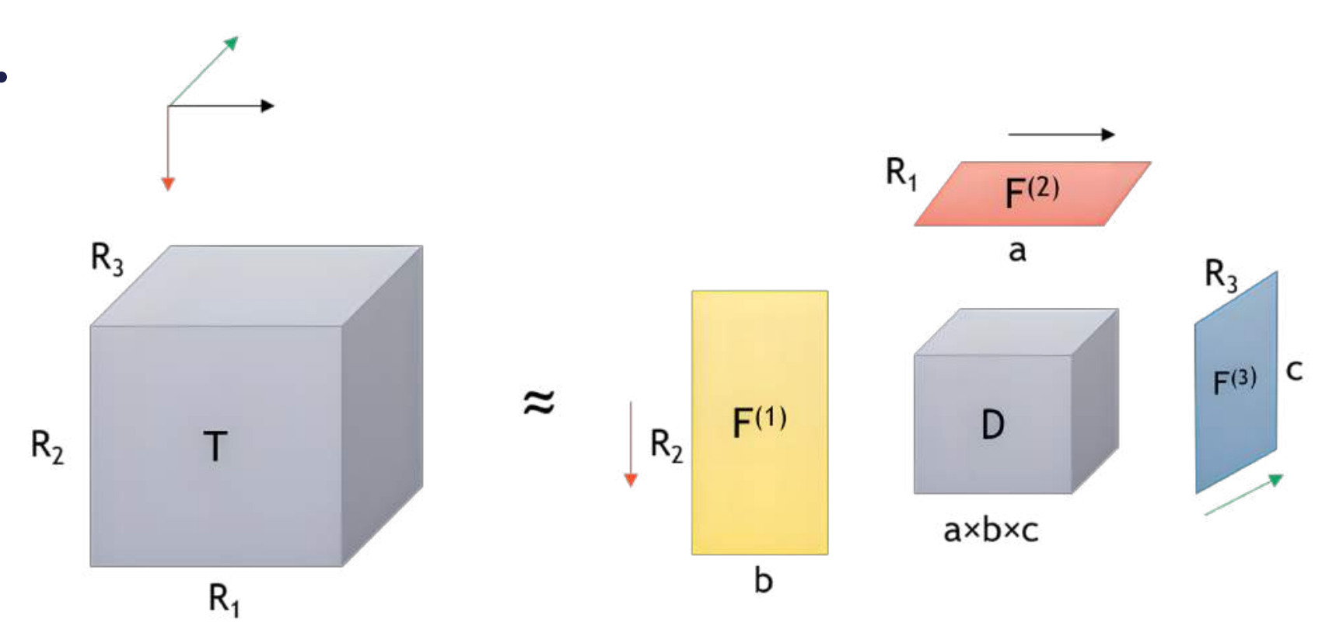

*Figure 6: Basic principle of Tucker Decomposition. A Convolutional 2D layer can be approximated by one core tensor of size a*b*c and three factor matrizes along each direction with the core tensor size and the original tensor size along the corresponding axes.*

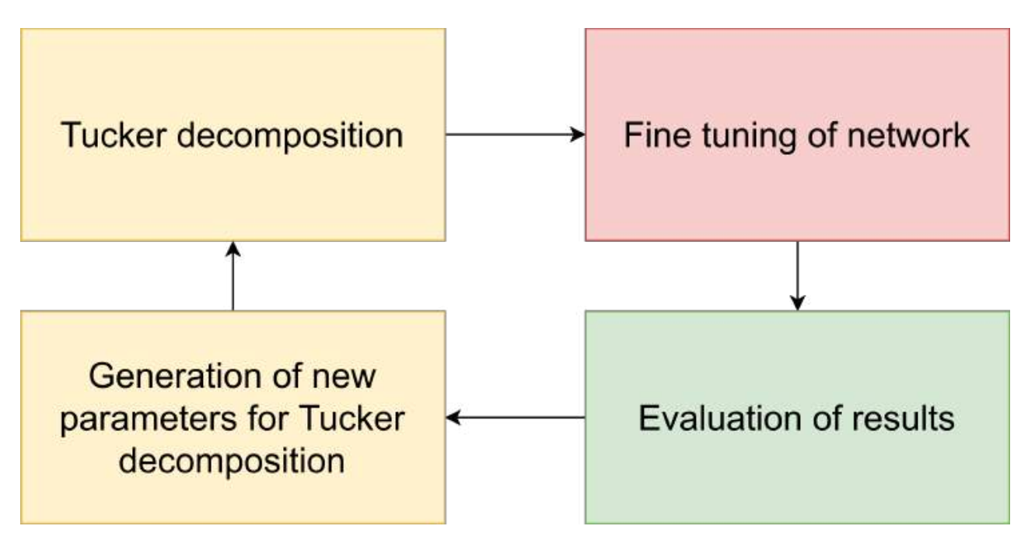

Figure 7: Tucker decomposition NAS cycle. First a Tucker decomposition is done using default values. After that the network is shortly trained again on the original dataset (fine tuning) Afterwards the results are evaluated regarding size and accuracy and from this new parameters for the tucker decomposition are generated

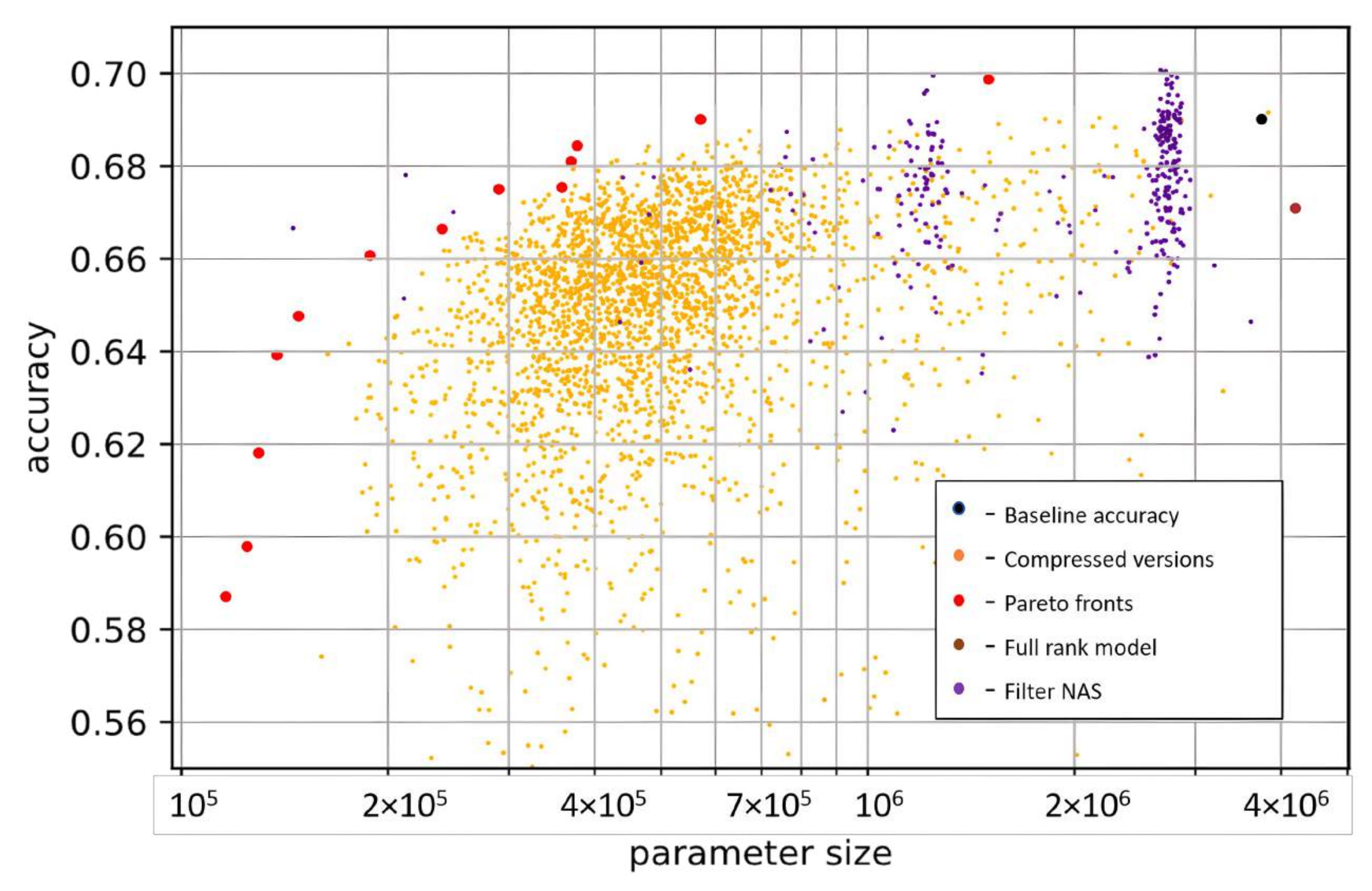

Figure 8: Results of the Tensor Compression NAS on AlexNet compared to SOTA filter NAS method. Tensor compression results in smaller networks often wih with little loss in accuracy. Filter NAS can sometimes result in smaller networks, but takes a longer time to reach such a result which is not always guaranteed

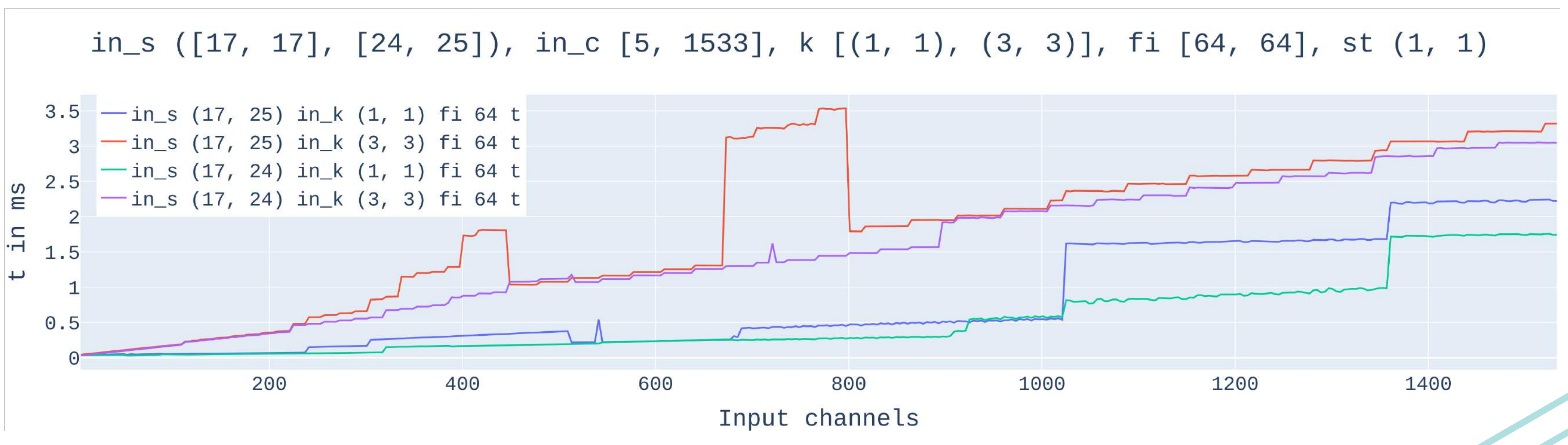

Figure 5: Sweep for different Conv2D layer combinations. it shows a non-linearity for certain parameter combinations. While the regular jumps could be predicted with a metaheuristic, the areas where the execution time increases and decreases suddenly cannot be reliably predicted

Partners

External partners

For more information contact:

domenik.helms@dlr.de

KI Delta Learning is a project of the KI Familie. It was initiated and developed by the VDA Leitinitiative autonomous and connected driving and is funded by the Federal Ministry for Economic Affairs and Climate Action.

KI FAMILIE

Supported by:
Federal Ministry for Economic Affairs and Climate Action
on the basis of a decision by the German Bundestag

www.ki-deltalearning.de   @KI_Familie   KI Familie