Fast semantic segmentation CNNs for FPGAs

Simon Wezstein[12], Muen Jin[1], Michael Stelzl[2], and Michael Heizmann[1]

[1] Karlsruhe Institute of Technology, Institute of Industrial Information Technology,
Hertzstraße 16, 76187 Karlsruhe, Germany
[2] MSTVision GmbH,
Im Weiherfeld 10, 65462 Ginsheim-Gustavsburg, Germany

Abstract In this contribution small semantic segmentation CNNs are evaluated against traditional segmentation approaches and state of the art segmentation CNNs. The CNNs are optimized for the implementation on frame grabber FPGAs. A dataset of industrial burner flames and a dataset of transparent plastic granules is used to assess the segmentation performance of the models. VisualApplets by Basler AG is used to implement the models on an FPGA. The implemented models reach foreground IoU values of up to 96.7 %. The inference of a 552 x 552 pixel image takes slightly more than 1 ms. The latency between the start of an input line to the output of the line is 0.1 to 1.9 ms for streaming an 8192 pixel wide image.

Keywords Image signal processing, FPGA, CNN, segmentation

1 Introduction

Segmentation is a common task in image processing. There are many methods of segmentation available, from simple global thresholds to deep neural networks. One common use case in industrial image processing is to combine semantic segmentation with a Binary Large Object (BLOB) analysis to form an object detection algorithm. There are many more applications, often dependent on segmentation: measurement of objects in images, classification of objects, motion detection and

DOI: 10.58895/ksp/1000174496-11 erschienen in:
Forum Bildverarbeitung 2024
DOI: 10.58895/ksp/1000174496 | https://www.ksp.kit.edu/site/books/m/10.58895/ksp/1000174496/

tracking, etc. Semantic segmentation may be seen as pixel-wise classification in an image. With semantic segmentation an image's pixels may be classified into various classes. Semantic segmentation with neural networks (NNs) recently gained big attention for complex tasks like autonomous driving and many other tasks with high variance regarding the imaging scene. any networks for semantic segmentation use convolutional filters, they are called Convolutional Neural Networks (CNN). In our work, we only refer to a binary segmentation, thus the classification in foreground and background.

Our former work on hybrid image processing with Field Programmable Gate Arrays (FPGAs) for low latency and high throughput applications ([1], [2]) was concentrated on balancing the computing load between CPUs, GPUs and FPGAs for optimized real time capability and image resolutions in sensor-based sorting. In our current architecture the FPGA is used for object detection and tracking and the GPU for object classification. The semantic segmentation in the object detection stage is realized by a global threshold operation. With this concept we are able to reduce the load on the PC host which allows low latencies and high raw image data throughput. Prior investigation and the correspondence with potential customers showed that simple rule-based approaches are often not powerful enough to fulfill the task. Employing CNNs for segmentation in a GPU would break our system architecture and running the whole raw image data through an NN would break the tight latency constraints (5 ms camera to actuator).

In our system design a frame grabber with an FPGA is always present. The approach is to develop simple yet sophisticated enough NNs to fit on this FPGA hardware as a drop-in replacement for the currently used global threshold. Many NNs are designed to fulfill more complex tasks than most of those in sensor-based sorting or industrial image processing in general. We seek to fill this gap. In industrial image processing, the imaging scene can be well controlled, which should allow the usage of simpler models in terms of parameters and operations, than the common state of the art ones. We want to optimize them for line scan cameras under low latency and high throughput demands.

2 Resources and Methods

Compared to GPU or CPU based development, on an FPGA the defined operations are configured in hardware instead of being broken down into machine code and being executed sequentially. Therefore all operations and parameters must fit into the FPGA's resources. The FPGA design is built with VisualApplets (VA), a proprietary platform by Basler AG for their frame grabbers [3]. Due to its exclusive use for the FPGA implementations at MSTVision GmbH, the set of possible operations is limited to the ones available in VA. The absolute hardware constraints lead to the unusual development strategy: "Which operations can be used and how many of them". We seek to find a sweet spot between model accuracy, hardware occupation and throughput/latency.

All currently available Basler frame grabber FPGA hardware is limited to integer arithmetics, forcing us to use quantized models. We use the Basler imaFlex CXP-12 Quad frame grabber for our experiments [4]. It is equipped with a Xilinx Ultrascale+ KU3P FPGA and 1.5 GB DRAM [5]. It has 160679 lookup tables (LUTs), 323224 flip flops (FFs), 720 block ram (BRAM) cells with 18 KiB each and 1368 48 bit digital signal processing (DSP) units.

For quantization aware training of our networks, QKeras [6] in conjunction with Keras [7] and TensorFlow [8] is used. The models are trained on an Nvidia RTX3080 GPU.

2.1 Available operations and limitations

The operator set of VA is limited to basic image and signal processing operations. These include: base arithmetics, convolution, image upscaling, lookup tables (LUTs), histograms, counters, BLOB detector, etc. Additionally there are many operations for data flow control like first in first out (FIFO) buffers, pipeline synchronisation, etc. Common operations in CNNs like matrix multiplication, activation functions, pooling, etc. are missing. If needed, they have to be implemented from scratch using the available operations. A complete list of available operators can be retrieved from [9].

2.2 Proposed models

Due to the limitations in hardware resources, implemented operations in VA and possible computation latency, we aim to build the models as simple and lightweight as possible. Our models need to be able to be trained from scratch to avoid legal problems with foreign datasets prohibiting industrial usage. For example the ImageNet dataset is restricted to non-commercial use [10].

Our most simple model (fig. 1) is a two layer convolutional model:

1. Convolution with 5x5 kernel, from 1 channel to 16 channels
2. Quantized ReLu
3. Convolution with 5x5 kernel, from 16 channel to 1 channel
4. Quantized ReLu

This forms the baseline of complexity and parameter count.

All other models are simple convolution only encoder-decoder-structures (fig. 2(a) and 2(b)). They consist of 2D convolutions, ReLu activations, max pooling, 2D transposed convolutions and upsampling with nearest neighbor interpolation. All models are quantized to 8 bit integer representation. The various tested models have varying filter sizes and encoder/decoder layer count. One part of the networks runs upsampling before transposed convolution, the other part after. Using upsampling after transposed convolution reduces the bandwidth to be processed in transposed convolution. Using smaller filter sizes reduces the amount of parameters. Tuning these parameters allows greater depth. The models, except the baseline model, use 8 channels and pooling/upsampling by the factor of 2 in the intermediate layers. All models are listed with their parameters in tab. 1. The model shown in fig. 2(a) consists of the following operations:

1. Convolution block with 5x5 kernel, 1 input and 8 output channels
   a) Convolution with 8 bit integer mask and input
   b) Offset with 16 bit values
   c) Rounding to 8 bit integers
   d) ReLu

2. Max pooling
3. Convolution block with 5x5 kernel and 8 channels
4. Max pooling
5. Upsampling
6. Transposed convolution block with 5x5 kernel and 8 channels
7. Upsampling
8. Transposed convolution block with 5x5 kernel and 1 channel

To save FPGA resources, all convolutions except the first and last layer of a model are carried out sequentially. This should pose only a minor impact on throughput and latency due to the reduced data rate after the max pooling operation. Other common operations like skip connections, fully connected layers, etc. were not considered due to their hardware requirements and/or implementation complexity.

Table 1: The models evaluated. ”Type” denotes the structure of the model with the abbreviations : ”c” for convolution, ”e” for encoder layer, ”d” for decoder layer and ”d (c)” for a decoder layer with convolutions instead of transposed convolutions. ”K” denotes the kernel size, e.g. 5 by 5. ”Ups.” denotes the order of upsampling operations: before the transposed convolution or after it. ”Para Cnt” denotes the number of model parameters.

Model	Type	K.	Ups.	Para Cnt
Base	2 Conv.	5	-	817
4 layer 5 a	2 e, 2 d	5	before	3625
4 layer 5 b	2 e, 2 d	5	after	3625
6 layer 5 a	3 e, 3 d	5	before	6841
6 layer 3 a	3 e, 3 d	3	before	2489
6 layer 5 b	3 e, 3 d	5	after	6841
6 layer 3 b	3 e, 3 d	3	after	2489
6 layer 3 a c	3 e, 3 d (c)	3	before	2489
6 layer 3 b c	3 e, 3 d (c)	3	after	2489

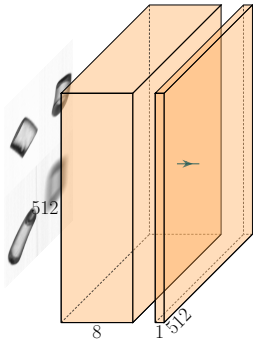


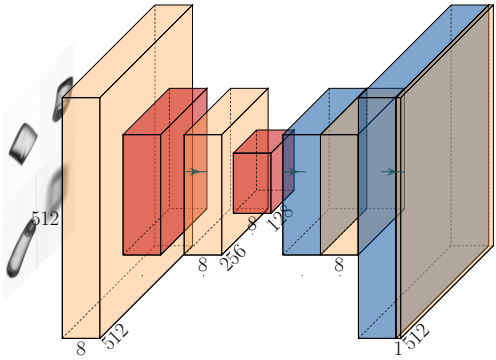
Figure 1: 2 layer baseline network. Generated with [11].

2.3 Data

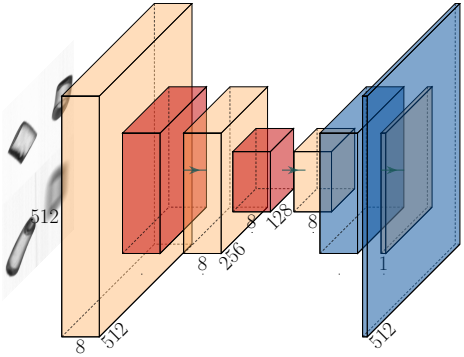
We use two industrial datasets to compare the proposed model's performances. The first dataset is the refined industrial burner dataset published in [12], [13]. It contains images with a resolution of 552×552 pixels. We use the dataset without augmentation to compare our results with theirs. They provide two datasets, "DataA" and "DataB", we use the first for our experiments, see fig. 3 for an example image. We swapped the test (160 images) and train (40 images) folder as they seem to be accidentally swapped.

The dataset has no predefined test subset, we use the validation set for testing.

The second dataset is a transparent plastic granule dataset based on our own data. The raw data was generated with a 16384 pixel wide line scan camera in a transmitted light setup. The granules to scan were poured on a slide while the camera was triggered at a line rate of 100 kHz. The raw data was filtered with a global threshold to remove most of the empty images. Segment Anything Model (SAM) [14] was used to generate masks for the granules. The masks were manually refined, the objects cropped to single 256×256 pixel images and randomly stitched to 512×512 pixel images (fig. 4). For training we use a 60/20/20 percent split. The stitched dataset, which is used for training, consists of 1004 images.



(a) 4 layer encoder-decoder-architecture with upsampling before transposed convolution.



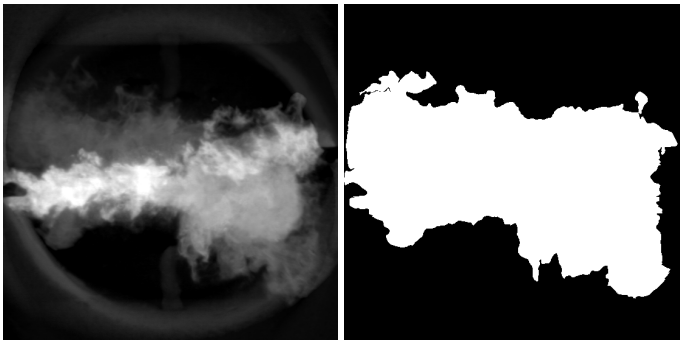
(b) 4 layer encoder-decoder-architecture with upsampling after transposed convolution.

Figure 2: 4 layer examples of the proposed encoder-decoder-architecture. The evaluated models vary in convolution kernel and pooling/upsampling sizes and in layer count. Generated with [11].

2.4 Evaluation workflow

For both datasets, each model is trained 10 times to gain statistics about the reached model performance. The models are trained from scratch with their default initialization defined by QKeras.

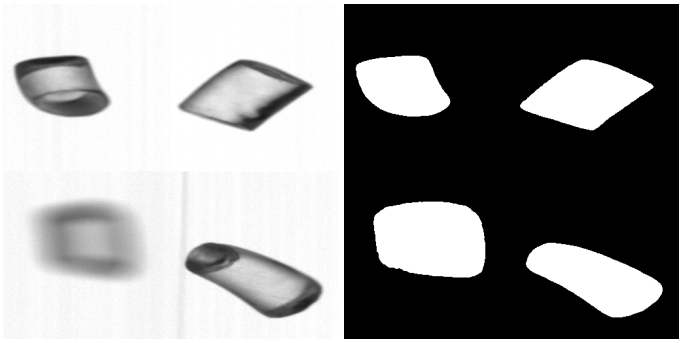
Training parameters:



(a) Industrial burner dataset image data example. Resolution: 552 x 552 pixel.

(b) Industrial burner dataset ground truth example. Resolution: 552 x 552 pixel.

Figure 3: Example image pair of the industrial burner dataset. [13, "DataA", image/mask 172]



(a) Image data example, consisting of four cropped granule samples. Resolution: 512 x 512 pixel.

(b) Segmentation ground truth example, consisting of four cropped granule samples. Resolution: 512 x 512 pixel.

Figure 4: Example image pair of our granule dataset.

- Batch size: 32
- Epoch count: 1500 for granules, 5000 for burner flames

- Optimizer: Adam
- Loss function: Binary Cross Entropy

Based on the captured statistics and the estimated performance, the best performing parameters are picked and implemented with VisualApplets. Timing measurements are implemented, too. The FPGA simulation results are compared to the results of QKeras. After synthesis, a 8192 x 512 pixel test image is uploaded to the FPGA and processed. The timing data is then evaluated.

3 Results

The test results of the best training run for each model are shown in table 2. All models perform better than the global threshold experiment, which yielded a foreground class intersection over union (IoU) of 81.4 % (test set of ”Data A”) and 80.2 % (test set of ”Data B”) for the burner flames and 51.5 % for the granules, except the baseline model which is below for the burner flames. We perform a grid search like [12] did. We consider our result of ”Data B” for our comparison, because of the same result in [12, tab. 1].

Due to problems in the implementation of transposed convolutions with VA, all models were trained with normal convolutions in the decoder layers. The models which were not implemented for the FPGA use transposed convolutions and are listed for comparison. The results for the FPGA implementations are listed in table 3.

Our best model on the burner dataset is ”6 layer 3 b c” with a mean IoU of 92.8 % and a foreground class IoU of 89.7 %. We implemented this architecture on the FPGA, see tab. 3. Our best model on the granules dataset is ”6 layer 5 a” with a mean IoU of 97.9 % and a foreground class IoU of 96.5 %. The best model which we could implement on the FPGA is ”6 layer 3 b c” with a mean IoU of 96.7 % and a foreground class IoU of 94.7 %. In comparison to the reference results of [12, tab. 1], most of our models perform better than their traditional machine learning models, with 86.6 % at best for an MLP. Our models perform worse compared to their neural networks with their worst results at 91.9 % (U-Net (MN)) and their best at 92.3 % (DL3+ (RN101)). We only consider their results for training from scratch as we did. The results

for the granule dataset show even better IoU values compared to the burner dataset results. This shows the potential of our models for segmentation in granule sorting.

In terms of inference time, the baseline model and the two 6 layer models take roughly 1 ms while the others take more time. This is due to the sequential calculation of the 8 channels while the amount of data is only reduced to 1/4 of the input bandwidth. The inference of the big image drops the throughput to around half the bandwidth. This behavior requires further investigation. Performing the upsampling operation after the convolution has positive effects for the IoU and for throughput. The simulation shows small differences between PC inference and FPGA inference. We suspect rounding problems as root case.

Table 2: The intersection over union (IoU) results of the models segmentation performance. "B" denotes the industrial burner dataset. "G" denotes the granule dataset. "FG" denotes the foreground class, "Mean" the mean IoU of background and foreground class. All values in %.

Model	Mean IoU B	IoU FG B	Mean IoU G	IoU FG G
Global Threshold	86.3	80.2	69.8	51.5
Base	87.7	82.4	79.7	67.4
4 layer 5 a	92.2	88.8	93.7	89.9
4 layer 5 b	92.7	89.5	96.1	93.6
6 layer 5 a	92.6	89.4	**97.9**	**96.5**
6 layer 3 a	92.3	89.0	94.8	91.5
6 layer 5 b	92.6	89.5	97.7	96.3
6 layer 3 b	92.7	89.6	96.8	94.8
6 layer 3 a c	92.5	89.3	95.5	92.6
6 layer 3 b c	**92.8**	**89.7**	96.7	94.7

4 Conclusion

We showed the potential of low parameter models for the usage in semantic segmentation with FPGAs. The models perform better than initially expected, superseding the traditional machine learning methods of [12] while having more throughput and lower latencies. Having an additional latency between 0.105 ms and 1.904 ms, the models and

Table 3: The throughput/latency and resource occupation results of the FPGA implementations. "L2L" denotes the time between processing the first pixel of a line and retrieving the first processed pixel of that line using the 8192 pixel wide test image. "Time B" denotes the inference time for a single 552 x 552 pixel image of the burner dataset. "LUT, FF, DSP and BRAM" show the relative resource consumption of the model on the FPGA.

Model	L2L [ms]	Time B [ms]	LUT [%]	FF [%]	DSP [%]	BRAM [%]
Base	0.105 ± 2.3e-3	1.041	48.83	32.33	2.41	37.78
4 layer 5 a	1.780 ± 0.14	2.755	28.29	31.27	85.31	78.89
4 layer 5 b	1.904 ± 0.16	2.074	28.41	31.3	85.31	61.25
6 layer 3 a c	1.393 ± 0.13	1.067	65.46	40.89	37.79	38.61
6 layer 3 b c	1.669 ± 0.17	1.058	65.68	40.68	37.79	32.5

implementations are considerable candidates for line scan applications. Future work will target the rounding problems in the FPGA implementation. Because of the usage of quantization aware training, we suspect the FPGA to have exactly the same output as computed on the PC. In addition the throughput decrease for large images needs will be investigated, too. We expect to be able to increase the throughput and parameter count further with bigger FPGAs, hopefully available in the near future. Future work will target the implementation of more sophisticated CNN-operations, too.

References

1. S. Wezstein, M. Stelzl, and M. Heizmann, "Latency evaluation of an FPGA-based sorting system," in *9th Sensor-Based Sorting & Control 2022*, K. Greiff, H. Wotruba, A. Feil, N. Kroell, X. Chen, D. Gürsel, and V. Merz, Eds., 04 2022, pp. 143–160.
2. S. Wezstein, O. Gräff, M. Stelzl, and M. Heizmann, "Latency evaluation of a CNN enhanced FPGA-based sorting system," in *10th Sensor-Based Sorting & Control 2024*, ser. Sensor-Based Sorting & Control, K. Greiff, A. Feil, L. Weitkämper, N. Kroell, T. Scherling, D. Gürsel, and V. Merz, Eds., vol. 10. Düren: Shaker Verlag, 03 2024, pp. 67–88.
3. Basler AG, "VisualApplets Graphical FPGA Programming," https://www.baslerweb.com/en/software/visualapplets/, 2024, online, accessed 21-September-2024.

4. ——, “imaFlex CXP-12 Quad,” https://www.baslerweb.com/en/shop/imaflex-cxp-12-quad/, 2024, online, accessed 21-September-2024.
5. ——, “VisualApplets User Manual,” https://docs.baslerweb.com/visualapplets/files/manuals/content/device_resources.html, 2024, online, accessed 21-September-2024.
6. C. N. C. J. au2, A. Kuusela, S. Li, H. Zhuang, T. Aarrestad, V. Loncar, J. Ngadiuba, M. Pierini, A. A. Pol, and S. Summers, “Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors,” 2021. [Online]. Available: https://arxiv.org/abs/2006.10159
7. F. Chollet *et al.*, “Keras,” https://keras.io, 2015.
8. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/
9. Basler AG, “VisualApplets User Manual,” https://docs.baslerweb.com/visualapplets/files/manuals/content/operator_documentations.html, 2024, online, accessed 21-September-2024.
10. Stanford Vision Lab, Stanford University and Princeton University, “ImageNet,” https://www.image-net.org/, 2021, online, accessed 21-September-2024.
11. H. Iqbal, “Harisiqbal88/plotneuralnet v1.0.0,” Dec. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.2526396
12. S. Landgraf, M. Hillemann, M. Aberle, V. Jung, and M. Ulrich, “Segmentation of industrial burner flames: A comparative study from traditional image processing to machine and deep learning,” Tech. Rep., 2023.
13. S. Landgraf, M. Hillemann, M. Ulrich, M. Aberle, and V. Jung, “Dataset for the segmentation of industrial burner flames,” 2023. [Online]. Available: https://publikationen.bibliothek.kit.edu/1000159497
14. A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, “Segment anything,” *arXiv:2304.02643*, 2023.