Improving Performance Estimation for FPGA-based Accelerators for Convolutional Neural Networks

Martin Ferianc[1], Hongxiang Fan[2], Ringo SW Chu[1], Jakub Stano[3] and Wayne Luk[2]

1 UCL 2 Imperial College London 3 ETH zürich

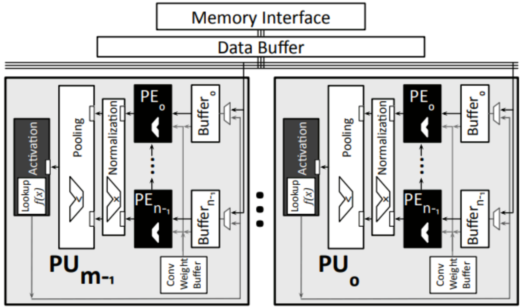
uceemfe@ucl.ac.uk github.com/martinferianc

Outline

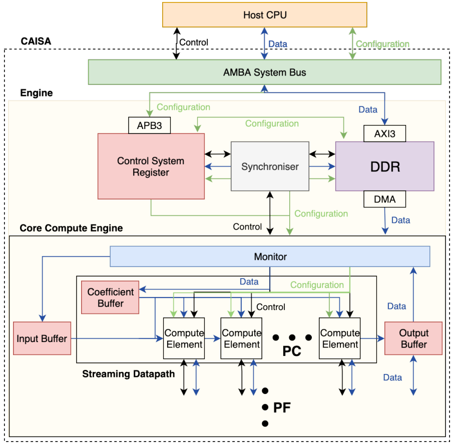
- **Background** Slides: X to Y
- **Method** Slides: X to Y
- **Experiments** Slides: X to Y
- **Conclusion and Summary** Slides: X to Y

Background: Problem

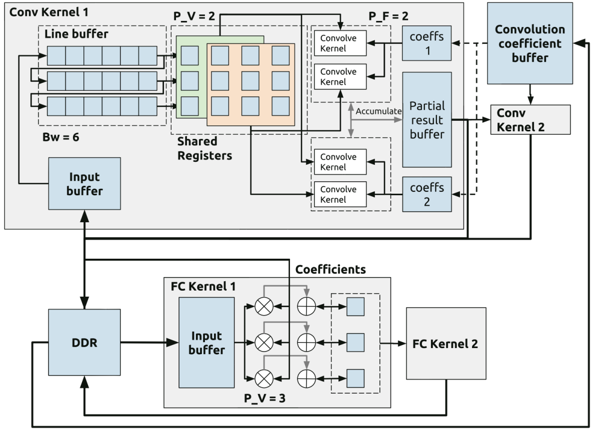
FPGA-based multi CNN Accelerators



[1]



[2]



[3]

Design space exploration (DSE)

$$\min_{SW,\,HW\ Variables}/\max \quad Objective$$

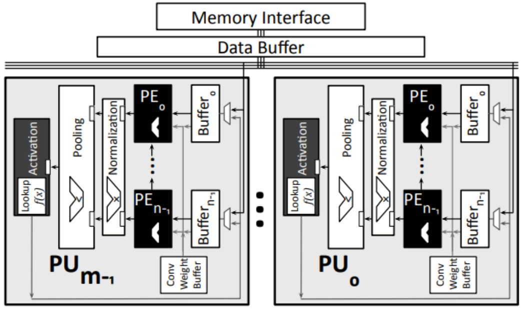
$$Hardware\ consumption(SW, HW\ variables) \leq Resources$$

$$Accelerator\ specific\ constraints$$

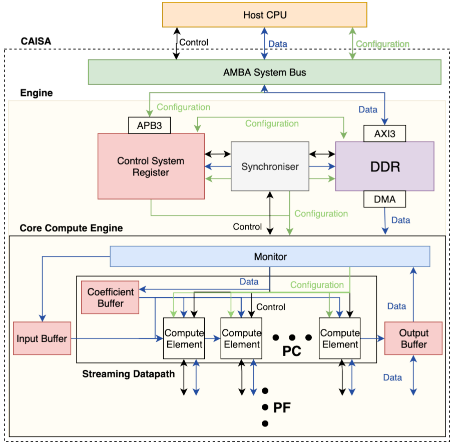
$$Network\ constraints$$

Background: Problem

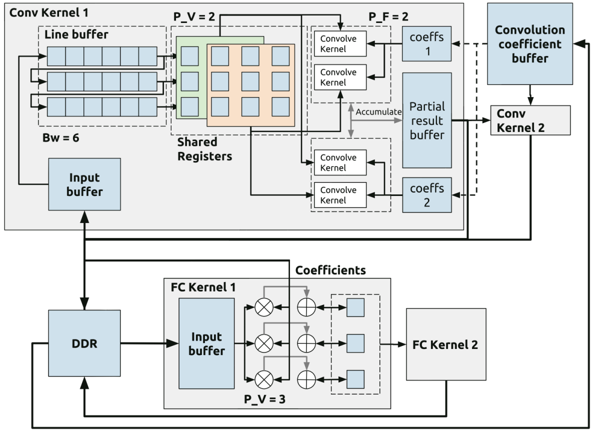
FPGA-based multi CNN Accelerators



[1]



[2]



[3]

Design space exploration (DSE)

$$\min_{SW,HW\ Variables}/\max \quad Objective$$

$$Hardware\ consumption(SW, HW\ variables) \leq Resources$$

$$Accelerator\ specific\ constraints$$

$$Network\ constraints$$

Estimate the influence of attributes to save computational resources

Background: Challenges

Design space exploration (DSE)

$$\min_{SW,\,HW\ Variables}/\max \quad Objective$$

$$Hardware\ consumption(SW, HW\ variables) \leq Resources$$

$$Accelerator\ specific\ constraints$$

$$Network\ constraints$$

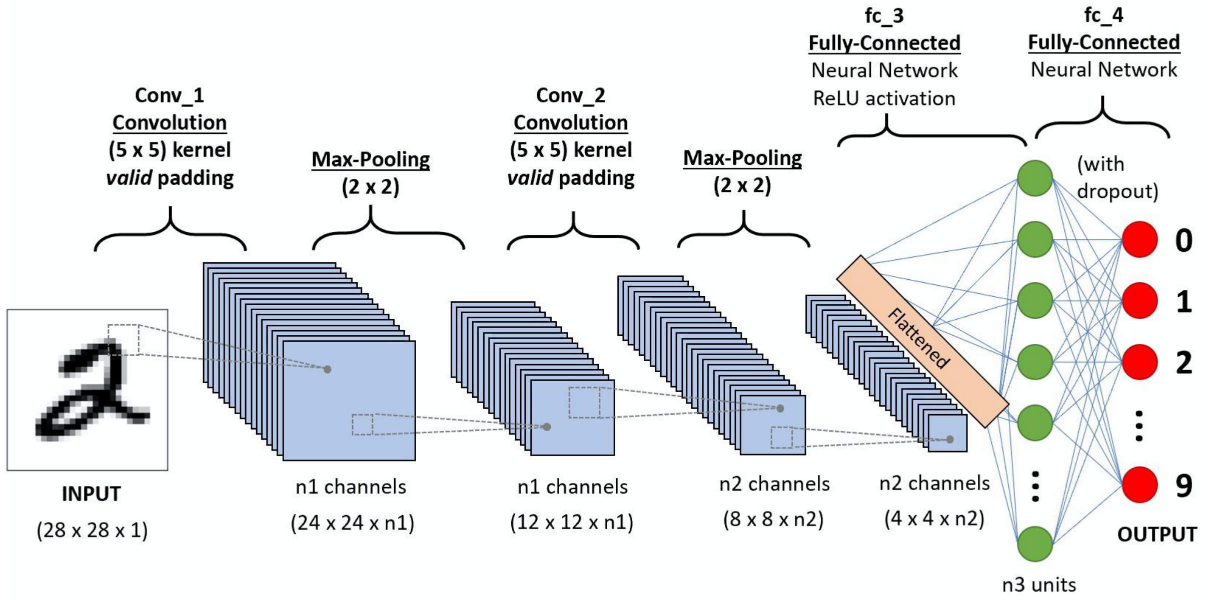
Estimate the influence of attributes to save computational resources

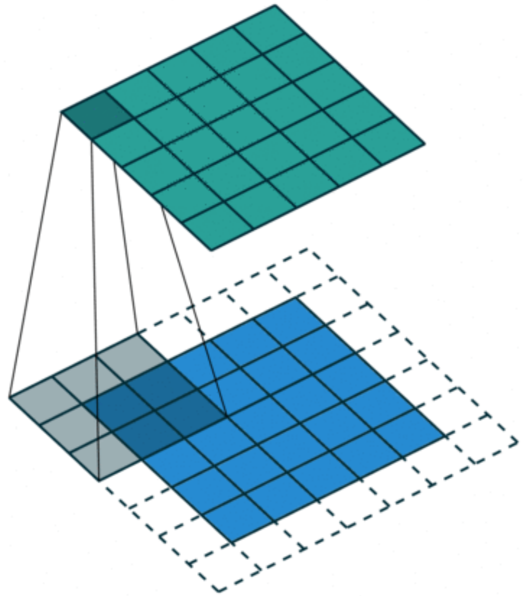
- **Challenges**
  - **Scheduling** which might be complicated if it supports Complicated operations
  - **I/O and Interruptions** which might or might not be regular or irregular

Background: Convolutional neural nets

Building block:
2D Convolution

LeNet-5 For classifying hand-written images





Background: Standard way - Convolution

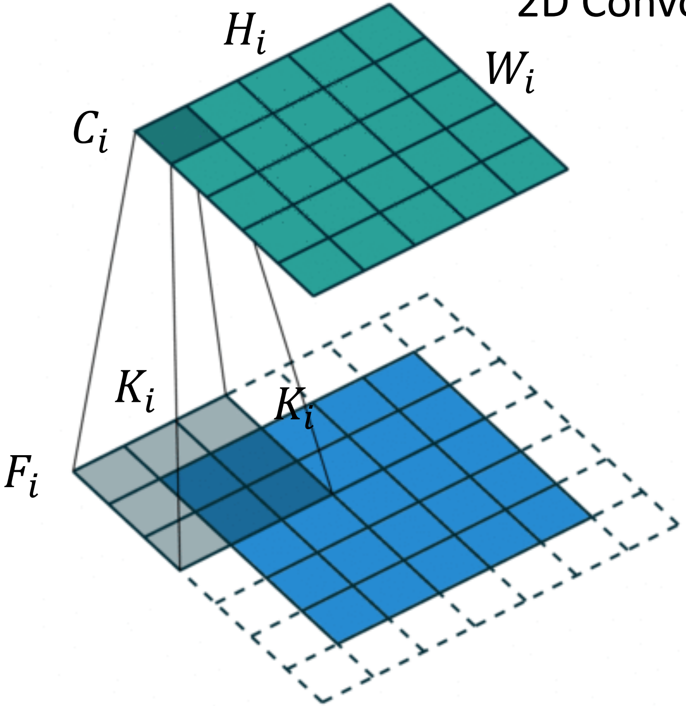
$$T_{compute_i} = \frac{F_i \times C_i \times H_i \times W_i \times K_i \times K_i}{Clock\ cycle\ time}$$

$$T_{store_i} = \frac{H_{O_i} \times W_{O_i} \times F_i}{Memory\ bandwidth}$$

$$T_{load_i} = \frac{C_i \times H_i \times W_i}{Memory\ bandwidth}$$

$$\boldsymbol{T_i} = \frac{\mathbf{max}(\boldsymbol{T_{compute_i}}, \boldsymbol{T_{load_i}}, \boldsymbol{T_{store_i}})}{\boldsymbol{Parallelism}}$$

2D Convolution



H_{O_i}, W_{O_i} Height and width of output

Background: Standard way

$$Latency = \sum_{i}^{N} \boldsymbol{T_i}$$ For each layer

$$T_{compute_i} = \frac{Number\ of\ operations}{Clock\ cycle\ time}$$

$$T_{load_i} = \frac{Input\ Size}{Memory\ bandwidth}$$

$$T_{store_i} = \frac{Output\ Size}{Memory\ bandwidth}$$

$$\boldsymbol{T_i} = \frac{\mathbf{max}(\boldsymbol{T_{compute_i}}, \boldsymbol{T_{load_i}}, \boldsymbol{T_{store_i}})}{\boldsymbol{Parallelism}}$$

Background: Standard way

$$Latency = \sum_{i}^{N} \boldsymbol{T_i}$$ For each layer

$$T_{compute_i} = \frac{Number\ of\ operations}{Clock\ cycle\ time}$$

$$T_{load_i} = \frac{Input\ Size}{Memory\ bandwidth}$$

$$T_{store_i} = \frac{Output\ Size}{Memory\ bandwidth}$$

$$\boldsymbol{T_i} = \frac{\mathbf{max}(\boldsymbol{T_{compute_i}}, \boldsymbol{T_{load_i}}, \boldsymbol{T_{store_i}})}{\boldsymbol{Parallelism}}$$

- Does not model I/O, interrupts, memory alignment, sync., control…
- But it can act as a **prior** knowledge and a **regulariser**

Method: Gaussian process

- Given that we have some **measurements** $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3 \ldots\}$
  For which we know the **features** such as the **software**
  Or **hardware parameters** such as: *Convolution parameters, Clock cycle time…*
- We have also recorded the objective such as **latency** as
  $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3 \ldots\}$
- We can then use the **measurements** and the **standard way** for estimation

Method: Gaussian process

- **Gaussian process** enables us to combine both
  - **Measurements** which give us a way to learn the estimation
  - **Standard heuristic** gives us a heuristic and a prior knowledge that can be followed to avoid **overfitting**

Method: Gaussian process

- **Gaussian process** enables us to combine both
    - **Measurements** which give us a way to learn the estimation
    - **Standard heuristic** gives us a heuristic and a prior knowledge that can be followed to avoid **overfitting**

$$p(\boldsymbol{y}_{\boldsymbol{pred}}|\boldsymbol{X},\boldsymbol{y},\boldsymbol{X}_{\boldsymbol{pred}}) = N(\mathrm{E}[\boldsymbol{y}_{\boldsymbol{pred}}|\boldsymbol{X},\boldsymbol{y},\boldsymbol{X}_{\boldsymbol{pred}}], \mathrm{V}[\boldsymbol{y}_{\boldsymbol{pred}}|\boldsymbol{X},\boldsymbol{y},\boldsymbol{X}_{\boldsymbol{pred}}])$$

$$\mathrm{E}[\boldsymbol{y}_{\boldsymbol{pred}}|\boldsymbol{X},\boldsymbol{y},\boldsymbol{X}_{\boldsymbol{pred}}] = \boldsymbol{m}(\boldsymbol{X}_{\boldsymbol{pred}}) + k(\boldsymbol{X}_{\boldsymbol{pred}},\boldsymbol{X})(k(\boldsymbol{X},\boldsymbol{X}) + \sigma^2\boldsymbol{I})^{-1}(\boldsymbol{y} - \boldsymbol{m}(\boldsymbol{X}))$$

Method: Gaussian process

$$p(\boldsymbol{y}_{pred}|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{X}_{pred}) = N(\mathrm{E}[\boldsymbol{y}_{pred}|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{X}_{pred}], \mathrm{V}[\boldsymbol{y}_{pred}|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{X}_{pred}])$$

$$\mathrm{E}[\boldsymbol{y}_{pred}|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{X}_{pred}] = \boxed{m(\boldsymbol{X}_{pred})} + \underline{k}(\boldsymbol{X}_{pred}, \boldsymbol{X})\underline{(k}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}(\boldsymbol{y} - \boxed{m(\boldsymbol{X})})$$

$\boldsymbol{X}, \boldsymbol{y}$: Training data, $\boldsymbol{X}_{pred}, \boldsymbol{y}_{pred}$: Target data

mean function

kernel function *e.g.: a Gaussian*

Key message

Replace m with T_i

$$\boldsymbol{T}_i = \frac{\mathbf{max}(\boldsymbol{T}_{compute_i}, \boldsymbol{T}_{load_i}, \boldsymbol{T}_{store_i})}{\boldsymbol{Parallelism}}$$

Experiments: Our accelerator

$$T_{weights_i} = \frac{K_i \times K_i \times F_i \times C_i}{PF \times M_{CLK} \times S \times M_{EFF}} \qquad T_{compute_i} = \frac{F_i \times C_i \times H_i \times W_i \times K_i \times K_i}{PF \times PC \times L_{CLK}}$$

$$T_{data_i} = \frac{H_i \times W_i \times C_i \times DW}{PF \times M_{CLK} \times S \times M_{EFF}} \qquad T_{store_i} = \frac{H_{O_i} \times W_{O_i} \times F_i \times DW}{PF \times M_{CLK} \times S \times M_{EFF}}$$

$$T_i = \begin{cases} T_{i=1} = T_{load_i} + T_{compute_i} \\ T_{i \neq 1, i<N} = \max(T_{weights_i}, T_{compute_i}) \\ T_{i=N} = \max(T_{weights_i}, T_{compute_i}) + T_{store_i} \end{cases}$$

In total we had **15 features** corresponding to **software** and **hardware** parameters

Experiments: Results for latency

- 3 CNNs (ResNet, SSD, Yolo), quantised to 8-bits with PC, PF=64

	Heuristic	GP	**GP and Standard**	GP and ANN	LR	GTB	ANN
Leave-one-out cross-validation (LOO) [ms]	**0.450**	0.521	**0.312**	0.692	0.450	0.607	1.257
Training Time	None	$O(N^3)$	$O(N^3)$	-	$O(N^2)$	$O(N_{trees} * N * \log(N))$	-
Inference Time	1	$O(N^3)$	$O(N^3)$	$O(N^3 * L)$	$O(P)$	$O(P * \log(P))$	$O(P * L)$

N stands for the number of training points, P for the number of input features, in this case 15, L stands for a complexity of hidden layers in a neural network. - stands for a variable complexity which depends on the architecture of artificial neural network as well as the optimiser method.

Intel Arria GX 1150

$$\frac{0.312}{0.450} \sim$$

31% lower error

uceemfe@ucl.ac.uk

Conclusion

	GP and Standard
LOO [ms]	**0.312**
Training Time	$O(N^3)$
Inference Time	$\boxed{O(N^3)}$

- Not limited to only estimating latency
- Reuses previously developed knowledge
- Works if the data is scarce (152 samples)

Scalability if large dataset, should not have a lot evaluations

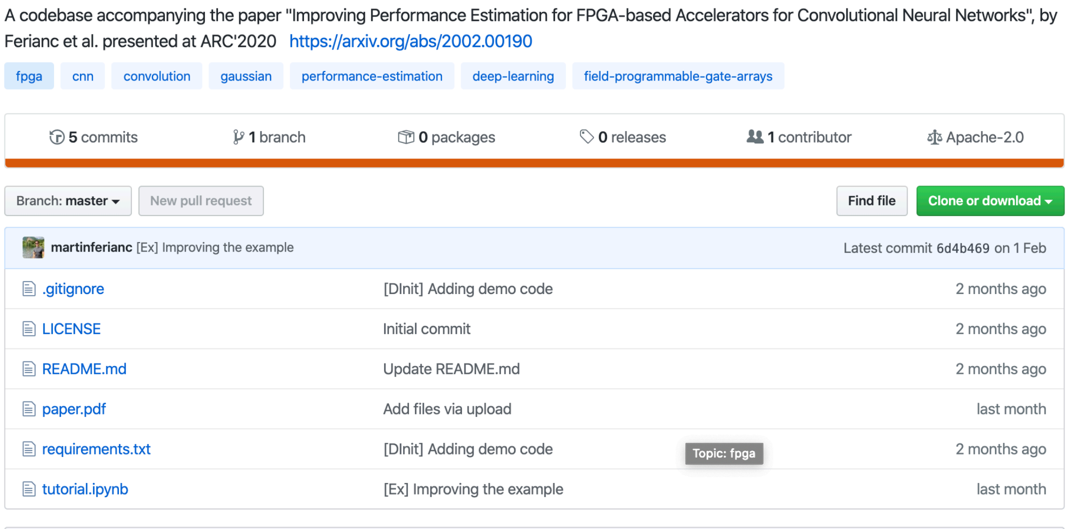
Future work

- **Multiple objectives:** Energy, throughput

- **Multiple functions:** 3D Convolution or other arch.

- **Co-design of accelerator and neural network Architecture** through neural architecture search

Summary



QR Code for an online tutorial

https://github.com/martinferianc/Improving_Per_Esti_for_FPGA-based_Acc_for_CNNs-ARC2020

References

[1] Njikam, A. N. S. and Zhao, H. (2016). A novel activation function for multilayer feed-forward neural networks. In *Applied Intelligence*, volume 45, pages 75–82, New York, NY, USA. Springer.

[2] Corerain Technologies Ltd. (2019). Rainman accelerator. https://bit.ly/2H5jFO0.

[3] Zhao, R., Niu, X., Wu, Y., Luk, W., and Liu, Q. (2017a). Optimizing CNN-based object detection algorithms on embedded FPGA platforms. In *Proceedings of 2017 International Symposium on Applied Reconfigurable Computing*, pages 255–267, Belfast, UK. Springer.

.