2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC) | 978-1-6654-9005-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/VLSI-SoC54400.2022.9939601

Gradient Backpropagation based Feature Attribution to Enable Explainable-AI on the Edge

Ashwin Bhat, Adou Sangbone Assoa, Arijit Raychowdhury
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA, USA
(ashwinbhat, aassoa3)@gatech.edu, arijit.raychowdhury@ece.gatech.edu

***Abstract*—There has been a recent surge in the field of Explainable AI (XAI) which tackles the problem of providing insights into the behavior of black-box machine learning models. Within this field, *feature attribution* encompasses methods which assign relevance scores to input features and visualize them as a heatmap. Designing flexible accelerators for multiple such algorithms is challenging since the hardware mapping of these algorithms has not been studied yet. In this work, we first analyze the dataflow of gradient backpropagation based feature attribution algorithms to determine the resource overhead required over inference. The gradient computation is optimized to minimize the memory overhead. Second, we develop a High-Level Synthesis (HLS) based configurable FPGA design that is targeted for edge devices and supports three feature attribution algorithms. Tile based computation is employed to maximally use on-chip resources while adhering to the resource constraints. Representative CNNs are trained on CIFAR-10 dataset and implemented on multiple Xilinx FPGAs using 16-bit fixed-point precision demonstrating flexibility of our library. Finally, through efficient reuse of allocated hardware resources, our design methodology demonstrates a pathway to repurpose inference accelerators to support feature attribution with minimal overhead, thereby enabling real-time XAI on the edge.**

***Index Terms*—Convolution Neural Network, Explainable Machine Learning, Back-propagation, Hardware Accelerator, FPGA, High-Level Synthesis (HLS)**

I. Introduction

There has been an exponential surge in the field of machine learning (ML) and artificial intelligence (AI) in the past decade. ML techniques, especially Deep Neural Networks (DNN) have found widespread adoption in various domains such as computer vision, speech recognition, autonomous driving and bio-medical applications. However, one major hurdle currently is the inability to interpret the output of these models since they are treated as a "black-box". The lack of transparency in the model's decision making process severely limits its applicability (Fig 1). In order to address this issue, several techniques have been proposed recently to interpret these models [1]. Explainable-AI (XAI) methods shed light into the workings of "black-box" models and thereby identify failure modes, establish trust in the end user and would eventually enable machine teaching [2].

XAI techniques can be broadly classified into three categories namely (1) visualization (2) model distillation and (3) training interpretable models [3]. Among these three, visualization is the only post-hoc explanation method that can be directly applied on existing pre-trained models [4], and hence is the focus of this work. Visualization comprises of assigning relevance scores to the input features of the model in order to quantify their importance to the output of the black-box model. In the case of image classification using Convolutional Neural Networks (CNN), the feature attribution scores can be visualized as a heatmap of the input pixels. This would highlight regions that contributed most for that particular input-output mapping produced by the model.

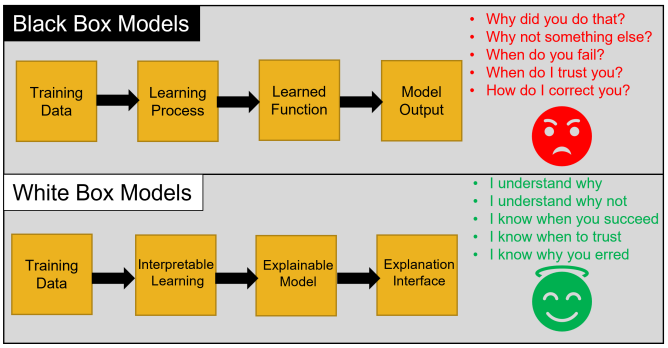

Fig. 1: Pitfalls of using models as black-box functions v/s advantages of developing Explainable AI.

The different feature attribution methods comprise of two common steps. First, a forward pass (FP) through the model to determine the inference result. The second step is a backpropagation (BP) through the model to evaluate the relevance scores for input features and generate the heatmap (Fig. 2). Compared to neural network training, feature attribution does not require calculating gradient with respect to the model parameters for the weight update (WU) step. Thus, the dataflow of feature attribution algorithms (FP+BP) lies in between that of inference (FP) and training (FP+BP+WU).

While inference accelerators have been designed for edge applications, supporting on-device training is challenging because of the large compute and memory overheads. WU is the most expensive step in training. It requires storing all intermediate activations during FP (memory overhead) and calculating gradient with respect to each model parameter (compute overhead). However, feature attribution only involves calculating local activation gradients layer by layer (BP).

This work was supported by Semiconductor Research Corporation (SRC) Task 2969.001.

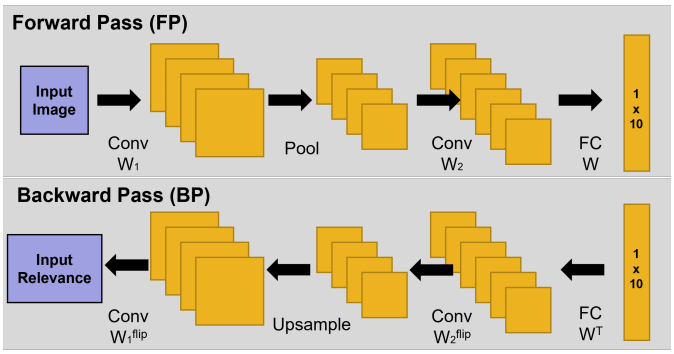

Fig. 2: The two phases of feature attribution algorithms for CNNs. First, a forward pass to determine inference output. Second, a backward pass to compute activation gradients.

XAI has been deployed for applications such as hardware security [5], medicine [6] and finance [7]. However, challenges remain that prohibit end-users from accessing explanations in real time [7]. In this work, we try to answer the question whether real-time XAI can be supported on edge devices. Specifically, by studying the hardware mapping of gradient backpropagation based feature attribution methods, the paper makes the following key contributions:

- Dataflow analysis of three gradient backpropagation based feature attribution methods: (1) Saliency Map, (2) DeconvNet, and (3) Guided Backpropagation to determine their h/w resource overhead compared to inference.
- We propose a hardware design that efficiently reuses compute blocks and on-chip buffers (designed for inference) during the BP step for feature attribution. The design can be configured to support any of the three feature attribution methods.
- We prototype our proposed design on a tiny, resource-constrained FPGA using High-Level Synthesis (HLS), thereby enabling real-time XAI on edge devices.

II. Feature Attribution

Feature attribution methods visualize the contribution of input features to the model's output decision in the form of a heatmap (Fig. 3). Higher relevance scores imply that those corresponding features create maximum response or stimulation influencing the model's output. These post-hoc methods can be applied to any off-the-shelf DNN model. After the inference step (FP) to evaluate the output, a backpropagation step (BP) is applied to evaluate gradient signals and pass them from output to input in a layer by layer fashion. We study the dataflow of three different commonly used feature attribution methods: (1) Saliency Map [8] (2) DeconvNet [9] and (3) Guided Backpropagation [10]. These methods differ in the handling of the gradient signals when it encounters a ReLU activation (Fig. 4) layer in the DNN. Equation 1 describes how the network activations are computed during FP when it encounters a ReLU activation at layer L.

$$f_i^{L+1} = \text{ReLU}(f_i^L) = \max(f_i^L, 0) \tag{1}$$

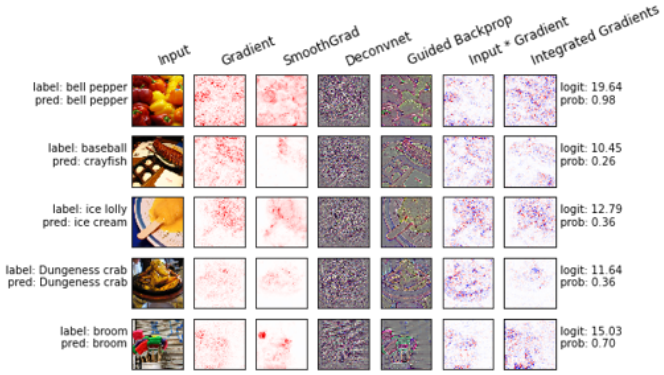

Fig. 3: An illustration [11] of post-hoc feature attribution methods. The generated relevance score are visualized as heatmaps. These heatmaps are visually validated to be highlighting those pixels that are relevant to the model's output decision. In this work, we focus on gradient, deconvnet and guided backprop.

A. Saliency Map

Saliency map is a baseline gradient based approach which assigns relevance scores ($R_i(x)$) to input features based on the partial derivative of the model's output ($f_c(x)$, where c is the output class) with respect to each input feature (x_i) as shown in Equation 2. A large value of the gradient implies that small changes in the value of that input feature would produce large change in the model's output, thereby indicating higher sensitivity. If we consider absolute value of the gradients, the positive and negative contributing features cannot be differentiated.

$$R_i(x) = \frac{\partial f_c(x)}{\partial x_i} \tag{2}$$

During BP, when a ReLU activation is encountered, the gradient signals are zeroed out corresponding to the negative values of activations during FP as shown in Equation 3. Thus, we need to store the indices of the negative activation values in order to support BP for a ReLU activation.

$$R_i^L = (f_i^L > 0) \odot R_i^{L+1} \text{ , where } R_i^L = \frac{\partial f^{out}}{\partial f_i^L} \tag{3}$$

B. DeconvNet

Deconvolution was originally designed to reconstruct the input of a CNN starting from the network outputs, in an unsupervised manner. It has been widely adopted as an XAI technique owing to its visualization power of most discriminative features. DeconvNet consists of inverse operations of the FP through a CNN. During BP, the convolutional layers are replaced with deconvolutions and max-pooling layers are replaced with unpooling layers. Deconvolution can be viewed as a transposed convolution and hence, DeconvNet boils down to evaluating gradients of the output with respect to the input features. The primary difference compared to vanilla gradients is in the handling of ReLU layer as shown in Equation 4.

$$R_i^L = (R_i^{L+1} > 0) \odot R_i^{L+1} \tag{4}$$

During BP, DeconvNet applies the ReLU function on the gradient values itself. Thus, it does not incur the memory overhead of storing indices of negative activation values during FP. However, by doing so, only those features that have a positive contribution to the model's output are highlighted.

C. Guided Backpropagation

This method combines the ideas of vanilla backpropagation in Saliency Maps with DeconvNet to make more accurate reconstructions starting from deeper layers of the network. As shown in Equation 5, Guided Backpropagation zeroes out values corresponding to negative activations (during FP) as well as negative gradients (during BP) when it encounters a ReLU layer. Similar to DeconvNet, the heatmap would only highlight features that positively contribute to the model's output decision.

$$R_i^L = (f_i^L > 0) \odot (R_i^{L+1} > 0) \odot R_i^{L+1} \tag{5}$$

In this case, we need to store indices of negative values of the activations during FP to guide the gradient computation across ReLU layers during BP. Thus, the incurred memory overhead is same as that of Saliency Maps.

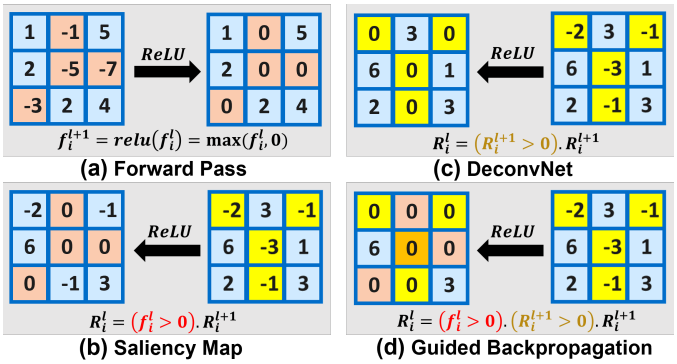

Fig. 4: Comparison of the dataflow of feature attribution methods at ReLU activation layer (a) Forward pass through a ReLU activation (b) Saliency map performs vanilla gradient computation across ReLU using indices of negative values during forward pass (c) DeconvNet applies ReLU on gradient values (d) Guided Backpropagation combining (b) and (c)

III. Hardware Implementation

Edge devices have limited hardware resources. There is a tight constraint on available memory, bandwidth, compute units as well as the power consumption. This necessitates a tile-based design in order to maximally extract the available parallelism in the algorithm while adhering to the hardware constraints.

A. Design Overview

We design a HLS library to support feature attribution for CNNs which typically consist of convolutional layers, pooling, fully connected (FC) layers and non-linear activations such as ReLU. The design is optimized to fit on a small FPGA. The CNN model parameters (weights) as well as the input image are stored in DRAM. Computation is performed in a layer wise manner for both FP and BP phases. Each layer is broken into multiple tiles based on its size. The tiles are loaded into the on-chip buffers from the DRAM using AXI interface. The output tile is stored back into the DRAM once computation is complete. This serves as the input for the next layer of the network.

B. Convolution Block

The primary compute kernel in CNNs during both BP and FP is the convolution operation. On-chip buffers are allocated to store the input and filter kernel tiles. The multiply-and-accumulate (MAC) operation is performed on these tiles utilizing the dedicated DSP units on the FPGA. Loop unrolling is performed in order to execute several MACs simultaneously in the same cycles. The loops along the height and width dimension of the input feature maps are unrolled and the corresponding buffers are partitioned accordingly. The unroll factors are configurable at design time. An output stationary dataflow is employed to perform the convolution. The output values are accumulated in-place in the output buffer while iterating over the input tiles. After the output tile computation is complete, it is stored back into DRAM.

C. Vector-Matrix Product Block

While the convolutional layers in a CNN are used as feature extractors, the FC layers at the end combine those features together to generate the output. FC layers can be mathematically expressed as a vector-matrix multiplication (VMM). To support FC layers, we design a VMM compute block. In order to have a tiled design, on-chip buffers are allocated for the input vector, the weight matrix and the output vector. The input vector as well as the weight matrix are split into tiles and loaded from the DRAM into the corresponding on-chip buffers. Output stationary is employed to accumulate the result in the buffers. The output tile is stored back into DRAM once it is completely evaluated. Partitioning the buffers and unrolling the loop performing the MAC operation enables the design to utilize the available parallelism.

D. Non-linear Layers

Apart from convolutional and FC layers, CNNs typically also comprise of other layers such as pooling and non-linear activations. Our HLS library currently supports max-pooling and ReLU activation. The implementation of these layers is designed to support both FP as well as BP.

ReLU. The Rectified Linear Unit (ReLU) activation zeroes out negative values going into the layer. ReLU is implemented via in-place modification of the value in the on-chip output buffers before storing those values back into DRAM. This

reduces the data movement between DRAM and on-chip buffers. To support BP, we observe that the gradient of a ReLU activation is a step-function. The gradient value is 1 for positive inputs and 0 for negative. Thus, at a ReLU layer during FP, a 1-bit mask is stored in the on-chip BRAM. This mask has the same dimension as the input feature map to the ReLU layer. This mask is utilized during BP to propagate the local activation gradients.

Max-pooling. Pooling layers reduce the size of feature maps via sub-sampling. A max-pooling layer picks the largest value in the sampling window and passes it to the next layer during FP (Fig. 5a). The window size is typically 2x2 and a stride of 2 ensures no overlap. The implementation of max-pooling is absorbed into the output store operation of the layer that it follows. After the computation of an output tile is complete, the values are scanned based on the pooling window and the stride and only the maximum value is written back into the DRAM.

Unpooling. Unpooling layers increase the size of feature maps via up-sampling. During BP phase, the gradient across a max-pooling layer is the unpooling operation. The index of the maximum value within the pooling window during FP is stored on-chip. For a 2x2 pooling window, each index is a 2 bit value. The size of the entire index mask is same as the dimension of the output feature map of the max-pooling layer. Unpooling operation uses the cached index value to perform the gradient routing (Fig. 5b) with the remaining values in the pooling window set to 0.

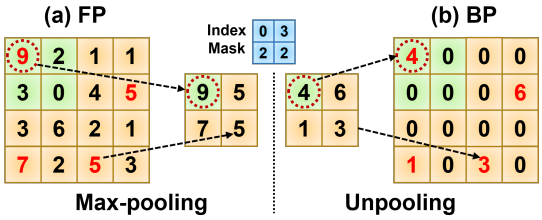

Fig. 5: Illustration of (a) Maxpooling with 2x2 window and stride of 2 during FP (b) Unpooling of gradient signal at same layer. The 2b index routes the gradient during BP

E. Gradient Computation

The BP phase for feature attribution requires computation of activation gradients sequentially in a layer by layer manner. The gradient values with respect to input features are evaluated using chain rule for derivatives. Our design is able to reuse the compute blocks designed for FP to perform gradient computation during BP as well. Thus, we have minimal area overhead to support feature attribution in addition to inference.

FC layers. Since the FP FC layer is a VMM product, its gradient ends up being a matrix-vector product. Thus, in order to re-use the VMM block, the on-chip buffers are loaded in a transpose manner from the DRAM during BP.

Convolution layers. The gradient computation with respect to activations for a convolution layers remains a convolution of similar dimension as that during FP. The only difference being (1) the input and output channel dimensions of the layer weight parameters are transposed (Fig. 6) and (2) the values of each kernel are flipped by 180°. We term this as a flipped-transpose convolution during the BP phase. The DRAM access pattern is modified during BP to handle loading the buffers in the required manner.

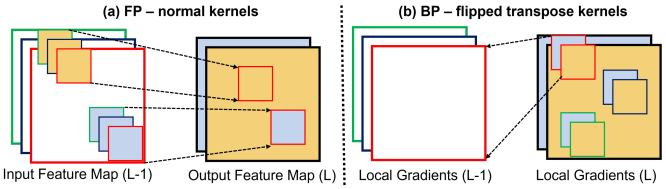

Fig. 6: (a) Feedforward convolutions (FP) with 3 input channel and 2 output channels with normal kernels (b) Backward convolutions (BP) with 2 input channels and 3 output channels with transposed kernels

F. Scheduling

The HLS library comprises of template functions that are required to support FP and BP phase for different layers of a CNN. Based on the CNN architecture, appropriate template functions are chosen from the library to execute the model. The layers are scheduled in a sequential manner. The output of each layer is stored into DRAM and is treated as the input for the next layer. The on-chip buffers and compute blocks are fixed at design time. Each layer of the CNN is broken up into tiles based on its size. Starting with the FP phase, the network output is evaluated. Mask values are stored on-chip at non-linear layers. The maximum output value at the last layer is chosen as the output of the network. The BP phase begins at this output value. The DRAM loading functions into the convolution (Table I) and VMM buffers are chosen according to operation phase. Gradient signals are propagated back to the input features. The feature relevance values and the inference output are evaluated for one input at a time (batch size = 1).

TABLE I. Buffer re-use across computational phase

Phase	Input	Weight	Output
FP	Activations (Layer L)	Normal Kernel	Activations (Layer L+1)
BP	Activation Gradient (Layer L+1)	Flipped + Transposed Kernel	Activation Gradient (Layer L)

G. Reconfigurable Design

The HLS library is designed to be reconfigurable and support different visualization based XAI methods which rely on gradient backpropagation. The memory overhead and the dataflow at the ReLU layers depends on the choice of feature attribution algorithm at design time (Table II). Currently, the library supports three different dataflows: (1) Saliency Map (2) DeconvNet and (3) Guided Backpropagation. Of the three, DeconvNet has the smallest memory overhead while Guided

Backpropagation introduces the largest amount of sparsity in intermediate gradient signals.

TABLE II. Memory overhead comparison at non-linearities

Attribution Method	Saliency Map	Deconv Net	Guided Backpropagation
ReLU Mask	Yes	No	Yes
Pooling Mask	Yes	Yes	Yes

IV. Results

A. Experimental Setup

To evaluate our hardware design, we train a representative CNN for CIFAR-10 dataset similar to [12] by stacking commonly used layers. The structure of the CNN is shown in Table III. The model size (2.26 MB) is comparable to SqueezeNet [13], a commonly used DNN model for edge applications. We use PyTorch to train the network achieving 88% accuracy after 20 epochs.

TABLE III. CNN structure

Input Shape	Layer (type)	Output Shape	# parameters
[3,32,32]	Conv2d	[32,32,32]	896
[32,32,32]	Conv2d	[32,32,32]	9248
[32,32,32]	MaxPool2d	[32,16,16]	
[32,16,16]	Conv2d	[64,16,16]	18496
[64,16,16]	Conv2d	[64,16,16]	36,928
[64,16,16]	Maxpool2d	[64,8,8]	
[64,8,8]	FC	[128]	524416
[128]	ReLU	[128]	
[128]	FC	[10]	1290

The FPGA accelerator is designed for this network using the HLS library and synthesized using Xilinx Vitis HLS tool at a target frequency of 100 MHz. The configurable data precision is set to 16-bit fixed point for activations, weights and gradient values. To demonstrate the flexibility of our HLS library, we synthesize our design on three different FPGAs: (1) Pynq-Z2 (2) Ultra96-V2 (3) ZCU104. While (1) is based on Xilinx Zynq-7000 SoC, (2) & (3) are based on Xilinx Zynq Ultrascale+ MPSoCs. The resource availability and power consumption of these platforms are comparable with edge devices. The hardware configuration of the synthesized design (which determines the resource utilization and latency) are chosen according to the target FPGA platform.

B. Analysis

Design Configuration. The configurable parameters for the synthesized design are the buffer sizes for the convolution and VMM compute blocks. The input/output buffers of the convolution block are partitioned along the height (width) dimension with a factor N_{oh} (N_{ow}). The DSP utilization for the convolution block is $N_{oh} \times N_{ow}$ owing to parallel MAC operation. For the VMM block, the buffer size is set to 16/32 based on available resources and the DSP utilization is equal to the same. Table IV shows the configurations chosen for the target FPGA boards. The hardware configuration remains same for both FP and BP phase of feature attribution for CNNs since we efficiently reuse the compute blocks.

Resource Utilization. Table IV shows the breakdown of the hardware resource utilization for inference (only FP) and feature attribution (FP+BP) for different configurations. We observe that the utilization of BRAM (memory) and DSP (compute) shows negligible change when we add a BP phase to support feature attribution. Thus, our design efficiently reuses the inference hardware to support feature attribution. DSP utilization is in accordance to the design configuration parameters chosen prior to synthesis. The increase in the FF and LUT utilization on adding the BP phase is attributed to the additional logic required for scheduling the layers. Feature attribution would make the scheduler go through the network layers twice compared to just once during inference. High LUT consumption, which is the limiting factor for further speedup in each configuration, is attributed to two reasons (1) partitioning of on-chip buffers for parallel read/write access, and (2) multiplexers for loading the on-chip buffers from different layer parameters in the DRAM. (1) is required to extract speedup from parallelism and (2) is required to efficiently reuse the hardware by changing DRAM access patterns during FP and BP.

Latency Table IV shows the latency for different hardware configurations obtained via simulation of the synthesized design at a 100 MHz clock. Larger loop unroll factors lead to higher parallelism in the MAC computation. As expected, the latency is lower for larger unroll factors on FPGAs with more available resources. The latency breakdown is provided for running inference (FP) and feature attribution (FP+BP). The overhead of supporting feature attribution in addition to inference manifests in the end-to-end latency of the running the entire network and varies from 50%-72% depending on the hardware configuration. On larger FPGAs, the FP and BP phases can be pipelined to improve the throughput of the design by $\approx$1.6$\times$ at the cost of separate compute blocks.

V. Discussion & Related Work

Software. Commonly used DNN software frameworks for CPU/GPU platforms such as Tensorflow and PyTorch implement BP via automatic differentiation. This incurs a large memory overhead since activation values during FP are cached to recursively evaluate gradients during BP using chain rule for derivatives. For the chosen network architecture (Table III), the memory overhead is 3.4 Mb. Our design avoid this issue by computing activation gradients in an analytic manner requiring mask bits only at non-linear layers. This reduces memory footprint to 24.7 Kb (137$\times$ lower). This optimization is specific to feature attribution since it does not involve calculating gradients with respect to weight parameters.

Inference Hardware. Optimized hardware architectures have been proposed to accelerate DNN inference [14] and prototyped on FPGA platforms. These designs only support FP phase but XAI techniques require additional computations in the form of gradient backpropagation. Our work highlights how to repurpose inference accelerators to support BP with low resource overhead for visualization based XAI.

TABLE IV. Evaluation of the hardware design on different target FPGA platforms.

FPGA	Operating Phase	Unroll Factors		Resource Utilization				Latency (ms)
		N_{oh}	N_{ow}	*BRAM*	*DSP*	*FF*	*LUT*	
Pynq-Z2	FP	4	4	10 (3%)	32 (14%)	18.6K (17%)	38.4K (72%)	43.53
	FP+BP			11 (3%)	33 (15%)	26.7K (25%)	52.9K (99%)	66.75
	Overhead			1	1	8.1K	14.5K	23.22
Ultra96-V2	FP	4	8	10 (2%)	48 (13%)	19.2K (13%)	47.8K (67%)	24.56
	FP+BP			11 (2%)	49 (13%)	25.6K (18%)	62.9K (89%)	39.96
	Overhead			1	1	6.4K	15.1K	15.4
ZCU104	FP	8	8	10 (1%)	96 (5%)	27.2K (5%)	68.1K (29%)	15.32
	FP+BP			11 (1%)	97 (5%)	34.9K (7%)	85.7K (37%)	26.37
	Overhead			1	1	7.7K	17.6K	11.05

Training Hardware. Hardware architectures designed to accelerate DNN training can directly support vanilla gradient based feature attribution [15]. However, modifications are required to support other visualization algorithms. These designs are optimized for evaluating weight gradients (WU phase) and incur memory overhead of caching activations during FP. This work highlights that these overheads can be avoided for feature attribution.

XAI Hardware. XAI being a relatively nascent field, there is still a dearth of hardware architectures that are specialized for these algorithms. [16] focuses on model distillation based explanation. Model distillation requires training a new interpretable model which mimics the original model's input-output behavior locally. Their target platform is server class TPU/GPU whose energy efficiency make them unsuitable for edge applications. We focus on visualization methods which does not require training new models and is suitable for real-time edge applications. [12] implements a cyclic weight storage to allow for normal (FP) and transpose (BP) access with no memory overhead. It incurs a logic overhead of address translator block. We observe that this technique is necessary only when all weights are stored in on-chip buffers. Our optimization relies on the data movement that occurs between DRAM and on-chip buffers in tiling based designs. The DRAM access patterns are modified during different compute phases. Thus, our design incurs low logic overheads while also simplifying the hardware design.

VI. Conclusion

In this paper, we present a HLS based FPGA accelerator for end-to-end XAI for CNNs. The dataflow of multiple feature attribution algorithms is analyzed to design a configurable HLS library supporting three different algorithms. By identifying the difference in the dataflow of inference, feature attribution and training, the gradient computation is optimized to minimize memory overhead. The accelerator is synthesized at 16-bit fixed point precision on multiple FPGAs with varying resource constraints demonstrating flexibility of our HLS library. Analysis of our implementation shows that supporting feature attribution in addition to inference incurs a latency cost while the resource overhead is minimal. Our methodology of reusing allocated memory and compute resources by modifying memory access patterns in inference accelerators and repurpose them to support feature attribution paves the way to enable real-time XAI on edge devices.

References

[1] A. Adadi and M. Berrada, "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52 138–52 160, 2018.

[2] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[3] G. Ras, N. Xie, M. van Gerven, and D. Doran, "Explainable deep learning: A field guide for the uninitiated," *Journal of Artificial Intelligence Research*, vol. 73, pp. 329–397, 2022.

[4] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Towards better understanding of gradient-based attribution methods for deep neural networks," in *6th International Conference on Learning Representations (ICLR)*, no. 1711.06104, 2018.

[5] A. Golder, A. Bhat, and A. Raychowdhury, "Exploration into the explainability of neural network models for power side-channel analysis," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, 2022, pp. 59–64.

[6] L. Ibrahim, M. Mesinovic, K.-W. Yang, and M. A. Eid, "Explainable prediction of acute myocardial infarction using machine learning and shapley values," *IEEE Access*, vol. 8, pp. 210 410–210 417, 2020.

[7] U. Bhatt, A. Xiang, S. Sharma, A. Weller, A. Taly, Y. Jia, J. Ghosh, R. Puri, J. M. Moura, and P. Eckersley, "Explainable machine learning in deployment," in *Proceedings of the 2020 conference on fairness, accountability, and transparency*, 2020, pp. 648–657.

[8] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *In Workshop at International Conference on Learning Representations*. Citeseer, 2014.

[9] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[10] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[11] M. Alber, S. Lapuschkin, P. Seegerer, M. Hägele, K. T. Schütt, G. Montavon, W. Samek, K.-R. Müller, S. Dähne, and P.-J. Kindermans, "innvestigate neural networks!" *J. Mach. Learn. Res.*, vol. 20, no. 93, pp. 1–8, 2019.

[12] S. Kolala Venkataramanaiah, Y. Ma, S. Yin, E. Nurvithadhi, A. Dasu, Y. Cao, and J.-S. Seo, "Automatic compiler based fpga accelerator for cnn training," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 166–172.

[13] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[14] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[15] J. Lee and H.-J. Yoo, "An overview of energy-efficient hardware accelerators for on-device deep-neural-network training," *IEEE Open Journal of the Solid-State Circuits Society*, 2021.

[16] Z. Pan and P. Mishra, "Hardware acceleration of explainable machine learning," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 1127–1130.