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

Ashwin Bhat, Adou Sangbone Assoa
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia, USA
(ashwinbhat, aassoa3)@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 broad field, ”feature attribution” encompasses post-hoc methods that back propagate gradients to assign relevance scores to the input features and quantify their contribution to the model's output. Designing flexible hardware for multiple such algorithms is challenging due to design complexity and large compute/memory overhead. In this work, we propose to design a HLS based flexible accelerator to support real time feature attribution heatmap generation on edge devices. Tiling based convolution is performed to utilize the constrained on-chip resources and sequential scheduling allows maximal reuse of dedicated compute kernels. We show that the overhead of supporting feature attribution in addition to inference manifests largely in terms of latency. Our design achieves a 10x performance improvement compared to a baseline CPU implementation with minimal memory overhead.**

***Index Terms*—Explainable-AI (XAI), Domain Specific Hardware Accelerators, FPGA, High-Level Synthesis (HLS)**

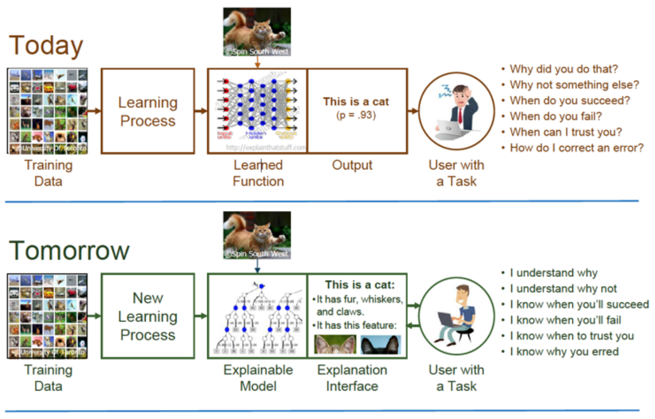


Fig. 1. A comparison between the current practice in the field of Machine Learning of using Models as Black-Box functions and the future trend on developing explainable models. However, post-hoc methods are being utilized to shed light on the decisions of present-day black box DNN models.

I. Introduction

There has been an exponential surge in the field of machine learning and artificial intelligence in the past decade. These algorithms have been adopted in widespread domains like robotics, autonomous driving, finance and medical applications. However, as shown in Fig.1, these models are treated as black boxes with no insight into why they work. To address this issue, explainable AI (XAI) techniques are being developed to shed light into the model's decisions. This is important from the perspective of developing trust in the model while also ensuring robustness, fairness and transparency.

Feature attribution methods are a subset of the XAI field. These are post-hoc explanation methods that work on existing black-box models. The basic idea is to assign relevance scores to each feature in the input space to quantify its contribution to the model's decision. The naive way to do is to evaluate the gradient of the classification probability with respect to each individual feature. A large gradient value implies that the model's output is highly sensitive to small changes in the feature value. Other methods within this category, such as DeConvNet and Guided Backpropagation modify the calculated gradients to define a better attribution rule.

The dataflow of feature attribution methods involves a forward pass (FP) through the network to calculate the inference result. This is then followed by a backward pass (BP) through the network in order to evaluate the gradients, modify them according to the chosen algorithm and assign relevance scores to each input feature. The backward pass poses significant overhead in terms of compute complexity and memory overhead. The compute overhead stems from the fact that BP requires twice the number of MAC operations compared to running inference only. The memory overhead is because we need to store the indices of sub-sampling layers such as max-pool in order to accurately oversample the gradient values during the backward pass. The hardware substrate should also be flexible enough to support a variety of gradient-based feature attribution methods.

In this work, we design a hardware accelerator using High-Level Synthesis (HLS) for feature attribution methods relying on gradient backpropagation. Our accelerator, targeted for edge applications, utilizes a tiling-based architecture. The novelty of our design and contributions are summarized below:

- Reuse of on-chip compute blocks designed for FP phase during the BP phase to ensure high utilization of resources
- A low memory overhead design to support feature attribution (BP phase) over an inference accelerator (FP

phase)

- A reconfigurable dataflow to support three different feature attribution methods: (1) Saliency Map (2) DeconvNet and (3) Guided Backpropagation

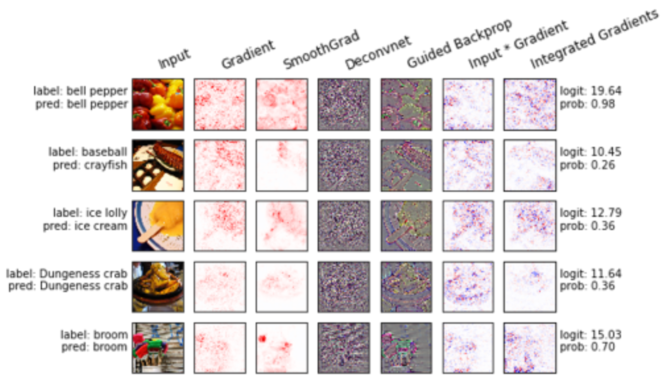


Fig. 2. An illustration of post-hoc feature attribution methods. The generated relevance score are visualized as heatmaps. These heatmaps are visually validated to be highlighting those pixels that are relevant to the model's output decision

II. Feature Attribution

Currently, Deep Neural Network (DNN) models are used as a black-box. The large number of parameters involved and inherent non-linearity of these models make it difficult for the user to understand their performance. Feature attribution methods (Fig.2) are algorithms that provide post-hoc explanations for existing DNN models that have been trained for a specific task. These algorithms require a forward pass (FP) through the network to evaluate the inference output followed by a backward pass (BP) to calculate the relevance scores. The algorithms differ in how they handle non-linear activations like ReLU in the backward pass as shown in Fig.3.

A. Saliency Map

In classification tasks, the output layer has as many neurons as there are classes in the dataset. The class corresponding to the output neuron with the highest activation is chosen as the decision of the overall model. The simplest way to evaluate feature importance ($R_i(x)$) is to evaluate the gradient of the classification probability ($S_c(x)$) of the chosen output neuron with respect to the input features (x_i). This is done by adding a gradient backpropagation step to the inference. A large value of the gradient implies that small changes in the value of the input feature would produce large changes in the model's classification score; thereby indicating the higher relevance of that particular input feature. Saliency maps do not differentiate between positive and negative contributions of the input features by considering the absolute value of the gradient, computed by:

$$R_i(x) = \frac{\partial S_c(x)}{\partial x_i} \tag{1}$$

B. DeconvNet

This approach is similar to evaluating gradients as in Saliency Map with the only difference being in how they deal with ReLU non-linearity. When encountering a ReLU activation, Saliency Map calculation zeroes out gradient values corresponding to the activations that were negative in the forward pass. However, DeconvNet applies the ReLU activation on the gradient value itself without considering the activations during the forward pass. By doing so, relevance scores are assigned to only those input features that positively contribute to the models output.

C. Guided Backpropagation

This method combines the ideas of both Saliency Map and DeconvNet. At a ReLU activation, it zeroes out gradient values that are negative (DeconvNet) as well as the values that correspond to negative activation values during the forward pass.

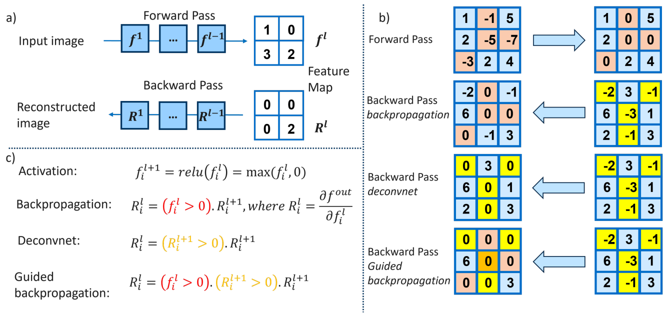

a)
Forward Pass
Input image
f^1
...
f^{l-1}
f^l
Feature Map
Backward Pass
Reconstructed image
R^1
R^{l-1}
R^l
b)
Forward Pass
Backward Pass backpropagation
Backward Pass deconvnet
Backward Pass Guided backpropagation
c)
Activation: $f_i^{l+1} = relu(f_i^l) = \max(f_i^l, 0)$
Backpropagation: $R_i^l = (f_i^l > 0).R_i^{l+1}, where\ R_i^l = \frac{\partial f^{out}}{\partial f_i^l}$
Deconvnet: $R_i^l = (R_i^{l+1} > 0).R_i^{l+1}$
Guided backpropagation: $R_i^l = (f_i^l > 0).(R_i^{l+1} > 0).R_i^{l+1}$

Fig. 3. Comparison of the dataflow of commonly used feature attribution methods (a) The two stages of a feature attribution algorithm namely forward pass to evaluate the model output followed by backpropagation to evaluate the attributions (b) mathematical expression for layer-by-layer computation involved in different algorithms (c) an example dataflow showing the differences in how the ReLU layer is handled by different algorithms

III. Hardware Design

In this work, we are targeting edge applications which have tight resource constraints in terms of storage, memory bandwidth and available on-chip resources. This necessitates a tile-based design in order to maximally extract available parallelism from the constrained hardware. The following blocks are implemented to support the kernels in the feature attribution algorithms.

A. Convolution Block

The primary operation in a Convolutional Neural Network (CNN) during both the FP and BP phase is convolution. On-chip BRAM buffers are dedicated to store the input and weight block tiles. These values are then fed to the compute units. The computation loops are unrolled along the height and width dimension of the feature maps and the BRAM buffers are partitioned accordingly. The convolution block utilizes 64 DSP units to compute 64 MACs in parallel. The tile scheduling is done in an output stationary manner. The output values are updated in the output buffer. Once the output values for a tile of the output feature map are completely evaluated, the output buffer values are written back into the DRAM.

B. Vector-Matrix Multiplication Block

The convolutional layers of a CNN are typically used for feature extraction. They are appended with fully connected (FC) layers at the end to combine the extracted features. To support FC layers, we design a dedicated Vector-Matrix Multiplication (VMM) block. On chip vector buffers are dedicated for the input vector, the matrix weight values and the output. The tiling is done over the matrix weight values. The output is again evaluated using a output stationary architecture. This enables us to utilize the available parallelism by partitioning the buffers and unrolling the loops. 16 DSP units are consumed by the VMM block.

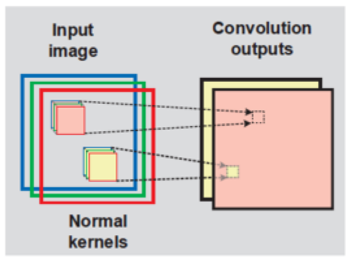
C. Miscellaneous

The primary computation blocks are the convolution and VMM engines. Apart from that, the operations that need to be supported include max-pooling, non-linear activation (ReLU) and upsampling. Maxpooling block is implemented as a comparator that reads in the required values from DRAM, loops over the required values to pick the largest value and store it back to DRAM. For non-linear activations like ReLU, in-place modification is done in the on-chip output buffers before storing the result back into DRAM. In order to support the gradient calculation during BP, mask bits are stored for both max-pooling as well ReLU. In case of max-pooling, a 2b index value is stored corresponding to each value of the output feature map. This value indicates the position of the value in the input feature map that got picked during maxpooling. In case of ReLU, a 1b mask value is stored corresponding to each output value indicating whether its input was positive or negative.

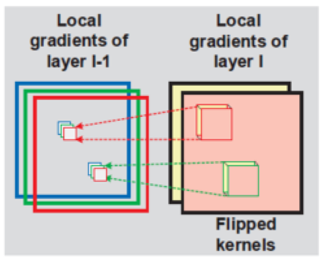
D. Scheduling

Since the target application is edge device, the architecture reuses the compute blocks that are designed to support the kernels. The different layers of the model are scheduled on the compute blocks in a sequential manner. The output of a layer is written back into DRAM from the output buffers and then reloaded into the input buffers while executing the next layer.

During the backward pass, we evaluate the gradients with respect to the activation values only and not the layer parameters. Thus, the dataflow of feature attribution algorithms (FP + BP) is somewhat in between the dataflows for training (FP + BP + Weight update) and inference (FP). The gradients are propagated back sequentially utilizing the chain rule for differentiation. In case of fully connected layers, the gradient computation is a matrix-vector multiplication. However, since the compute kernel is designed for a vector-matrix product, we load the buffers in a transposed manner from the DRAM to be able to reuse the same hardware. Similarly, for the convolutional layers, the gradient computation is also a convolution with similar dimensions. The only difference being, the input channel and output channel dimensions are switched (transpose) as compared to FP while the kernel dimensions are flipped (rotated by 180 degrees). The layer parameters are loaded from the DRAM to the on-chip buffers in a transposed flipped fashion to reuse the hardware resources during the BP phase.



(a) Feedforward convolutions.



(b) Backward convolutions.

Fig. 4. Convolution operations and changes in kernels during FP and BP

E. Reconfigurable H/W Substrate

The HLS based design of the accelerator is done in order to make it reconfigurable and support different types of back-propagation based feature attribution methods. Currently, the implemented library of HLS functions supports Saliency Map, DeconvNet and Guided Backpropagation. The key difference in these algorithms lies in the backward pass and how they deal with the gradient values. For the Saliency Map, we need to store the mask bits for non-linear layers like Mapool and ReLU activation during FP phase and utilize them to make in-place updates to the gradient values before storing them into the DRAM. In case of DeconvNet, no mask bits are needed since it only applies a ReLU on the gradient signal. Thus, DeconvNet has the lowest memory overhead of all three. GuidedBackpropagation is implemented as a combination of both Saliency Map and DeconvNet. Due to this, the gradient values have the highest amount of sparsity for this case which can be a future research direction ie. optimizing the architecture for sparse BP phase.

TABLE I
CNN STRUCTURE

Input Shape	Layer (type)	Output Shape	Number of parameters
[1,3,32,32]	Conv2d	[1,32,32,32]	896
[1,32,32,32]	Conv2d	[1,32,32,32]	9248
[1,32,32,32]	MaxPool2d	[1,32,16,16]	
[1,32,16,16]	Conv2d	[1,64,16,16]	18496
[1,64,16,16]	Conv2d	[1,64,16,16]	36,928
[1,64,16,16]	Maxpool2d	[1,64,8,8]	
[1,64,8,8]	FC	[1,128]	524416
[1,128]	ReLU	[1,128]	
[1,128]	FC	[1,10]	1290

IV. Results

The design was synthesized on the PYNQ-Z2 board based on using the ZYNQ XC7Z020 FPGA. We trained a neural network on the CIFAR-10 data-set using a 16-bit fixed point precision and achieved 88% accuracy in 20 epochs. The CNN structure is summarized in Table I. Table II shows the resources utilization. Besides the LUT utilization (82%) the BRAM (8%), DSP (11%) and FF (19%) usage is low. Low BRAM and FF usage is credited to the use of the tiled-based convolution and matrix-vector multiplication. As Table II shows, LUT utilization is the main limitation of the optimization. Indeed, reusing the compute blocks requires loading them with parameters from different layers which results in a large number of multiplexers. Furthermore, for the backward pass, the features and weights are loaded in a transpose fashion compared to the forward pass. These require the use of several different memory loading functions. Performance of our FPGA designed is compared to a CPU implementation on the 650MHz dual-core Cortex-A9 processor. As shown in table III, our accelerator has an overall latency of 0.109s which is an 10X improvement compared to the 1.02 s latency performance of the CPU.

V. Discussion and Related Work

There have been several designs that have been proposed for accelerating CNN Inference both on FPGAs as well as ASICs. These designs vary in scale from large datacenter designs optimized for throughput to edge applications optimized for low latency and energy efficiency. In our work, we are studying the hardware requirements to support feature attribution in addition to inference. Our design shows that the same compute units can be repurposed in order to utilize them for the backward pass that is needed to compute activation gradients.

On-device training is gaining traction in order to preserve data privacy for the end user. However, training DNN models has a very large compute and memory overhead which makes it challenging and close to impractical for larger models. For training, the activations during the forward pass need to be stored in order to compute the gradients with respect to model parameters during the backward pass. Popularly used software Machine Learning (ML) frameworks such as Tensorflow and PyTorch utilize automatic differentiation to implement the gradient computation during training. The same method is used for feature attribution in software implementations leading to a large memory overhead. We try to answer the question that even though on-device training has significant hurdles, can we at least support feature attribution on edge platforms to build user trust in black box models? Our design and implementation shows that by using mathematical expressions for gradient computation, we are able to reduce the memory overhead significantly and the overhead lies purely in terms of latency. Reusing pre-existing compute blocks for inference enables deployment of feature attribution on the edge.

During the forward pass, the mask bits for non-linear layers (ReLU, MaxPool) are saved. Even though, the masks have same shape as the activation layers, they are quantized to 1-bit and 2-bit for the ReLU and MaxPool respectively, resulting to a small memory overhead. In contrary, the compute overhead is large. Fig.5 shows the normalized latency breakdown of the different phases. The backward pass takes as much time than the forward pass, hence a factor of 2 overhead latency. This is expected since the hardware reuses the compute blocks. Overhead of adding feature attribution to an inference accelerator is mainly latency and not memory.

TABLE II
Resources Utilization

Resources	BRAM	DSP	FF	LUT
USED	25	25	20312	43787
TOTAL	280	220	106400	53200
% UTILIZATION	8	11	19	82

TABLE III
Performance comparison with CPU

Device	FPGA	CPU
Latency (s)	0.109	1.02

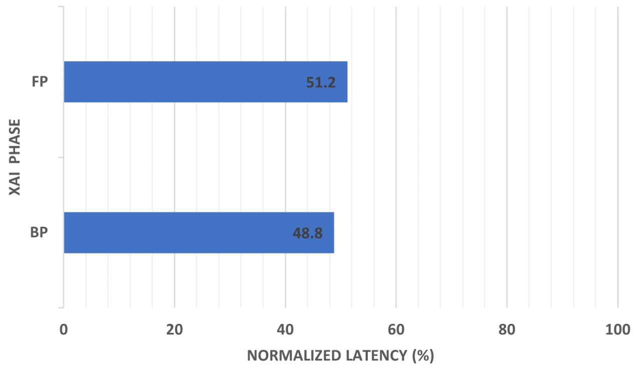


Fig. 5. Normalized latency breakdown

VI. Conclusion

The work presented in this paper proposed a HLS based flexible accelerator for real time feature attribution for edge applications. Our design shown the successful implementation of commonly used layers in CNN with a reconfigurable hardware which can support both inference and activation gradient backpropagation. We demonstrated the reuse of on-chip compute blocks for both FP and BP phase while having a low memory overhead. The dataflow is flexible enough to support different feature attribution algorithms. Performance comparison with CPU shows 10x improvement in terms of latency.

References

[1] Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., & Kagal, L. (2019). Explaining explanations: An overview of interpretability of machine learning. Proceedings - 2018 IEEE 5th International Conference on Data Science and Advanced Analytics, DSAA 2018, 80–89. https://doi.org/10.1109/DSAA.2018.00018

[2] Ancona, M., Ceolini, E., Öztireli, C., & Gross, M. (2017). Towards better understanding of gradient-based attribution methods for Deep Neural Networks. 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings, 1–16. http://arxiv.org/abs/1711.06104

[3] Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. 3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings, 1–14.

[4] Zeiler, M. D., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 8689 LNCS (Issue PART 1, pp. 818–833). https://doi.org/10.1007/978-3-319-10590-1_53

[5] Venkataramanaiah, S. K., Ma, Y., Yin, S., Nurvithadhi, E., Dasu, A., Cao, Y., & Seo, J. S. (2019). Automatic compiler based FPGA accelerator for CNN training. Proceedings - 29th International Conference on Field-Programmable Logic and Applications, FPL 2019, 166–172. https://doi.org/10.1109/FPL.2019.00034

[6] Wu, X., Ma, Y., Wang, M., Wang, Z. (2022). A Flexible and Efficient FPGA Accelerator for Various Large-Scale and Lightweight CNNs. IEEE Transactions on Circuits and Systems I: Regular Papers, 69(3), 1185–1198. https://doi.org/10.1109/TCSI.2021.3131581