DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
INDIA

September 18-19, 2024
Radisson Blu, Marathahalli, Bangalore

DVCON Processor Design Stage 1

Ratnayake R.M.L.H., Rupasinghe N.P.S.S., Jayakumar W.S.
Team WSL, Submission 5406
University of Moratuwa

Table of Contents

List of Tables

List Of Figures

List of Symbols

ViT – Vision Transformer
MLP – Multi Layer Perceptron
MSA – Multi head Self Attention
GEMM – GEneral Matrix Multiplier
MAC – Multiply Accumulate
PE – Processing Elements

Abstract

This document focuses on the design of an accelerator for inference on a vision transformer. It comprises of background research, goals, and design methodology. The vision transformer was trained, and inference was run on the trained network. It was this model that needed to be accelerated. In order to design a suitable accelerator, we first looked into existing accelerators and focused our attention on how such architectures can be implemented for our purpose. We have also stated the methodology we would be using to test the processor and benchmark it.

I. Introduction

In this project, we aim to increase the speed and efficiency of running a vision transformer on a custom designed accelerator. We propose to use a custom IP block for this purpose which is specifically designed based on neural networks and the computations done when running inference on a neural network. We commence this report by providing background research on the subject. Next, we take a look at the goals we are trying to achieve. Afterwards, we focus on the proposed design and thought process for arriving at the said design in such a way that our goal would be met. Finally, we present the expected results of the design.

II. Background Research

Initially the model was trained for 50 epochs on a cloud computer from Azure Machine Learning. We selected the first 10000 images from each category to perform the training. This was so that during inference, the model could be tested on completely new images. The following results were obtained for accuracy and loss. Based on the figures, it can be seen that the model performs fairly well with an accuracy of around 0.97 in the validation samples.

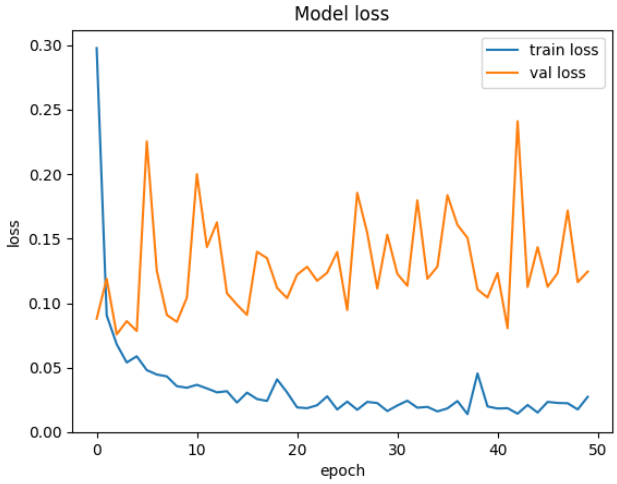

Figure 1

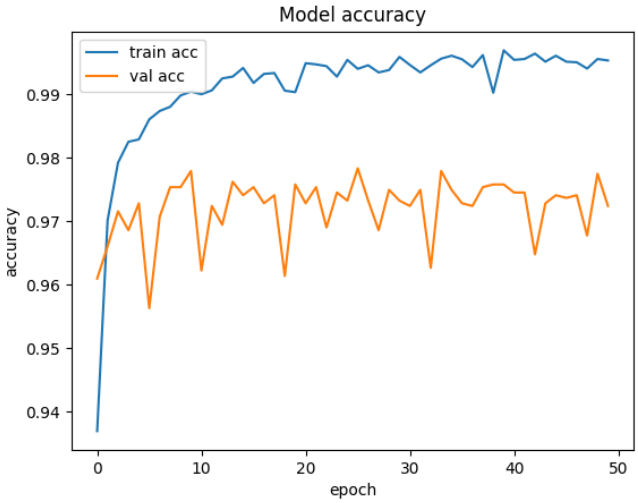

Figure 2

In addition to this, we also researched into the architectures used in current AI accelerators such as the Google TPU, Microsoft Brainwave [1], HPIPE [2] and Nvidia GPU's. It was noted that these chips use a highly parallel architecture with pipelining and interleaving in order to have the maximum performance.

The main components are fast memory access, parallel multipliers, and accumulators or fused multiply and add units. In addition to the hardware, the instructions present in these accelerators were also studied. Specifically, instructions that are used to speed up inference were studied. One such operation was Stassen's matrix multiplication method. This technique can reduce the number of multiplications in exchange for increasing the number of additions. We also studied how temporal and spatial reuse can be employed in order to reduce the number of memory reads. Specifically, we studied weight stationary, activation stationary and output stationary architectures.

Additionally, we analyzed the code provided and came up with the flow chart provided below. This flow chart contains the steps in running inference on the model. The layers used for inference are different from the layers used during training as techniques such as dropout would be avoided during inference. This diagram provided us with the basic structure we would need to develop our architecture.

We also looked into software optimization techniques such as quantization. Additionally, we studied the importance of the sensitivity of this model and added a SoftMax layer to the final layer of the model.

(It is discussed under V. Results and Discussions)

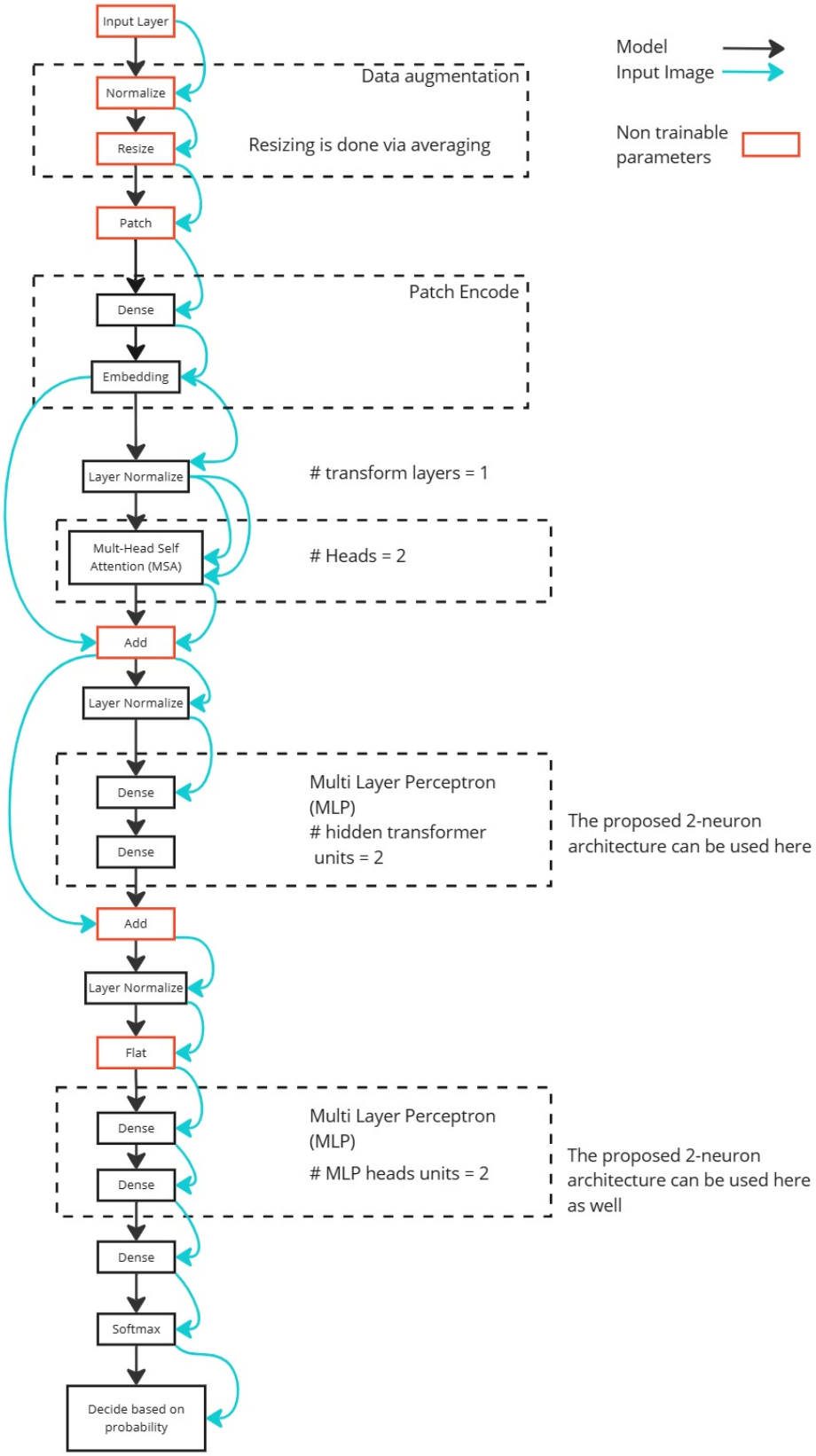
Input Layer
Normalize
Resize
Data augmentation
Resizing is done via averaging
Patch
Dense
Embedding
Patch Encode
Layer Normalize
transform layers = 1
Mult-Head Self Attention (MSA)
Heads = 2
Add
Layer Normalize
Dense
Dense
Multi Layer Perceptron (MLP)
hidden transformer units = 2
The proposed 2-neuron architecture can be used here
Add
Layer Normalize
Flat
Dense
Dense
Multi Layer Perceptron (MLP)
MLP heads units = 2
The proposed 2-neuron architecture can be used here as well
Dense
Softmax
Decide based on probability
Model
Input Image
Non trainable parameters

Figure 3

III. Goal and Objectives

Our goal is to design a suitable processor for running inference on the trained model. For this purpose, we propose four subgoals:

1. Design a processor that can run inference within a short time.
2. Design a processor that is energy efficient.
3. Maintain high sensitivity of the model.
4. Adding required minimum amount of extra complexity to the design for meeting expected goals.

For the first subgoal, it is necessary to focus mainly on improving both the computation capabilities of the proposed accelerator and on the memory access speeds while reducing the frequency of memory access.

For the second subgoal, reduction in silicon area, less frequent memory access and more efficient instructions are necessary.

It might seem that the first two subgoals are conflicting with each other and hence it is necessary to find a good balance between both sub goals.

The model would need to, in most cases detect a malware as malware. The goal is that in malware is tested, it would need to be accurately detected while benign ware being detected as benign ware is not that strict of a requirement.

Given the limited timeline, the proposed system would need to simple as possible. This needs to be done while not affecting the other goals.

IV. Design Process

In this section we present the design process of the proposed design.

i. Problem Statement

Design of a Fast and Power-Efficient FPGA-Based Accelerator for Vision Transformer Inference

ii. Functional Specification

Perform matrix multiplications efficiently: This can be done using Strassen's algorithm and hardware specifically designed for matrix multiplication with fused multiply adders.

Efficient data movement between on-chip memory and RAM. We propose to explore different spatial reuse techniques.

Optimize dataflow and pipelining strategies to maximize parallelism and minimize latency.

iii. Proposed Design

Our proposed design is similar to that of existing architectures of specialized processors for Artificial Intelligence.

The proposed design has a structure as shown below.

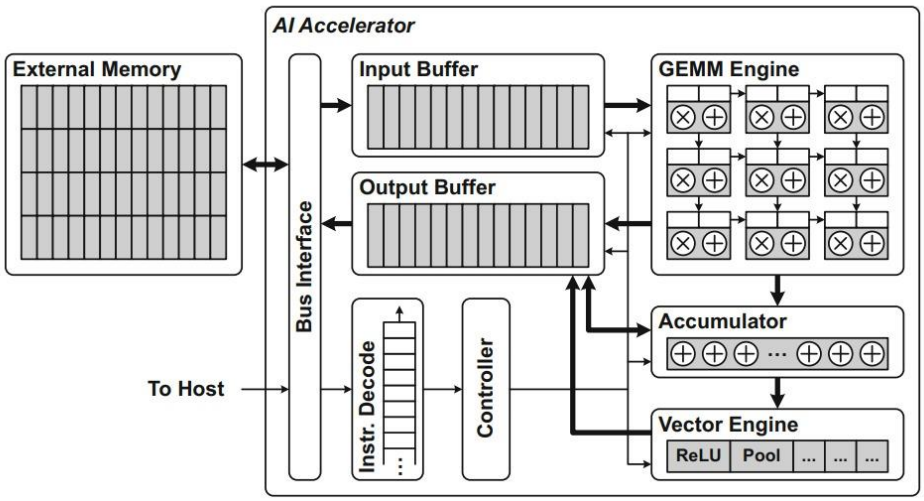

Figure 4

The details of the architecture are given below.

GEMM engine

This is a collection of highly parallelized multiply accumulators for high-speed inference.
GEMM has interconnected parallel PEs in it.
PE has 2 registers in it.
The GEMM engine is the main component that is responsible for calculations, the main calculation being matrix multiplications. Based on the maximum number of calculations per layer, we would decide on the number of parallel units needed. Additionally, this is pipelined so that we would be able to interleave operands if necessary. Regarding the GEMM engine, we had to make following 3 decisions:

Decision 1: Number of GEMM engines

Increasing the number of GEMM engines would be beneficial to the performance of the inference as matrix multiplications can be done in parallel. However, the inclusion of more GEMM engines would increase the silicon area requirements and consequentially, power draw as well.
They are scarce resources in edge devices. Therefore, a proper balance would need to be sought.

Decision 2: Number of parallel PEs in a GEMM engine

This would depend on the number of parallel calculations required per layer. Having the number of PEs as the number of parallel calculations required per layer is beneficial to reduce computation time. Additionally, this is pipelines and hence operands could be interleaved.

Decision 3: Spatial reuse

For the purpose of spatial reuse, we propose an output stationary model. This is because the outputs of one layer are sent as activations to the next layer in a neural network. Having such an architecture improves the performance as the processing elements would less frequently require data from the memory. This additionally saves power as well.

Decisions 1 and 2 would need to be decided after extensive testing.

Memory Hierarchy
(External Memory, IO Buffers, Registers in PE)

Regarding memory, the design would contain a 3-level hierarchy. These are external DDR memory, input and output buffers, and registers. The DDR memory is used to store the weights as the on-chip memory would not be sufficient to do so. The input and output buffers are used to mitigate the latency effects of reading and writing to and from slower DDR memory. This is used to get the weights of the next layer and to store the

outputs of a layer to be used later. Registers are placed close to the MAC units to enable very fast access to data.

Accumulator

An accumulator is needed to sum up the products of activations with weights. This needs to be of high speed. Additionally, it is possible to pipelining and interleave operations to mitigate pipeline stalls. This can be done especially for the MLP layers. Additionally, the accumulator would be needed to sum up the product between weights and activations and also to add the bias to the activations.

Vector Engine

Gelu is the activation function used. Additionally, SoftMax is used for the output layer, though not mentioned in the code, it is usually used in the output layer for classification problems and hence it was added. Using SoftMax ensures the output follows a probability distribution and that the values are normalized between 0 and 1. This further allows to define detection thresholds to ensure higher sensitivity of the output. Both activation functions can use the same vector engine as mentioned in [5]. This is beneficial as the silicon area used won't increase by much. Additionally, this unit has been specifically designed for vision transformers.

Inter layer optimization.

In addition to these optimizations, we propose to use the 2-neuron architecture as proposed in [4]. All above optimizations contribute to improvements within a layer. However, the 2-neuron architecture would improve inter layer performance. This is done by calculating each activation and then immediately sending it to the next layer.

Decision 4

Division Unit

ViT model perform division in few stages.

i. Resizing images based on averaging.
ii. Layer normalization

Division is quite a computation-intensive task. We have few options regarding that.

i. Use existing IP blocks.
ii. Consider using already existing DSP blocks in FPGA.
iii. Implement a division unit in our Accelerator IP

Rather than dividing in the VEGA processor, these methods will improve performance, but it will increase the power consumption, complexity, and silicon area of the design.

AXI interface

The accelerator would communicate with external memory and VEGA processor over AXI.

All decisions are associated with balancing our main goals. Especially, since it is an edge device, goals such as silicon area need to be considered. We would need to take decisions such that all goals would be balanced. Power consumption also needs to be considered using power estimators from FPGA manufacturers.

The processor should perform at an optimum level, that is, it should neither be compute bound nor memory bound. As shown in figure 5, the performance of the neural network should be at the cross over point between being memory bound and compute bound.

iv. Analysis of Final Design

The proposed design consists of what is required to run inference on a model efficiently. This would greatly improve the speed when running inference and directly supports the first goal as stated in our list of goals. Additionally, by only including the necessary components and not designing a general-purpose processor, silicon area is minimized. This leads to a more power efficient design.

Considering the vision transformer design, it is possible to implement a processor that can run inference on many types of neural networks. However, such a generalization would adversely affect both the main goals of our proposed architecture. Therefore, the architecture has been designed only to run inference on vision transformers. This enables us to achieve our goals more easily.

v. Testing and Implementation of Final Design

The final design would need to be tested on an FPGA. When testing we plan on testing the speed of inference and the efficiency of the processor as these are the first two main goals of our processor. Elements such as the size of the memory buffers and number of Processing Elements in General Matrix Multiply units would need to be tested to obtain an efficient processor. It would need to be tested against existing accelerated systems such as GPUs and TPUs. Additionally, it would need to be benchmarked against a non-accelerated system. A non-accelerated system would include modern desktop CPU's, enterprise CPU's and a Raspberry Pi. Benchmarking would be done both for speed and power efficiency using power estimators.

Simulations would need to be done to find the optimal clock cycle for the accelerator. Input will be a bit stream related to the Vit instructions and it will go through our IP, and we will validate our design by analyzing the outputs.

Timeline: The provided timeline of nearly 2 months would be sufficient to design the processor. 40% of the time would be allocated to designing the processor and the remaining 60% for testing, simulation, validation, and architectural improvements.

V. Results and Discussion

The proposed system would accelerate inference operations similar to most modern-day neural network accelerators such as GPUs and TPUs. Comparing the performance against non-accelerated systems and other accelerators would serve as a benchmark for our processor.

In the proposed system, we plan on implementing an optimized solution. This means that the processor performs at the transition point between being memory bound and compute bound. In order to achieve this, we would need to fine tune the memory bandwidth and number of processing elements (PEs).

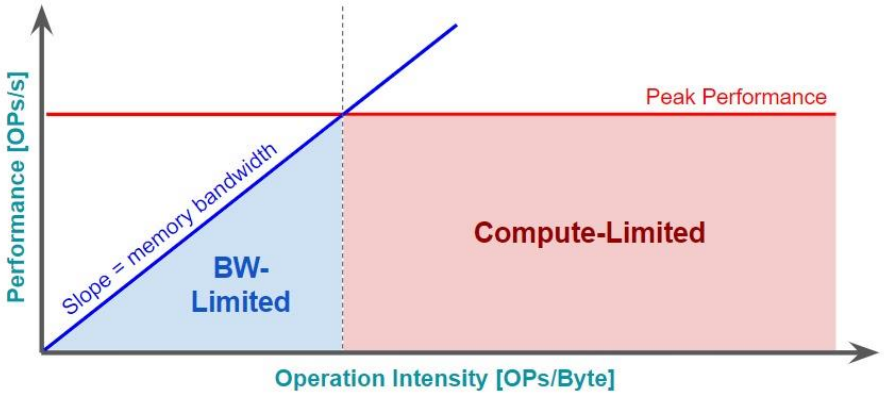

Figure 5

We quantized the model using TensorFlow lite to enable faster inference. This converted the parameters of the model to int8 and int32 datatypes. This quantized model had to be tested for accuracy.

In order to test this model, we used images that weren't used for the testing purpose. The accuracy was 96.19% for the quantized model which in line with the unquantized model's 96.15% accuracy. This concludes that the model hasn't lost accuracy due to the quantization.

Additionally, based on the application, we also tried to tune the inference for obtaining high sensitivity (proportion of malware correctly identified by the model) as the consequences of incorrectly detecting malware as benign ware are far more serious than the consequences of incorrectly labelling benign ware as malware. To facilitate this, the output layer of the model was given a sigmoid activation in order to get a probability distribution. This would enable us to set a probability threshold as we wish. For inference, the sensitivity of the model would be high while not compromising on overall accuracy. We used a threshold of 60 with the total range being 255, which resulted in a sensitivity of 96.3% while maintaining the accuracy at 97.02%. This would indicate that the model is biased toward predicting images as benign ware due to the trained data.

VI. Conclusion

The proposed design is specifically optimized to run inference related tasks on a vision transformer and other neural networks. The overall design has considered both hardware optimizations as well as software optimizations. Due to this, we are confident that the proposed system would meet our goals of having fast inference performance as well as being energy efficient.

VII. References

[1] E. Lee, V. Chandra, A. Shrivastava, M. J. Moudgill, "Brainwave: Latency-Tolerant Real-Time AI Inference," in Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, 2018, pp. 1-13. [Online]. Available: ISCA18-Brainwave-CameraReady.pdf (microsoft.com)

[2] M. Hall and V. Betz, "HPIPE: Heterogeneous Layer-Pipelined and Sparse-Aware CNN Inference for FPGAs," in Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2020

[3] A. Mishra, J. Cha, H. Park, and S. Kim, Eds., *Artificial Intelligence and Hardware Accelerators*. Cham, Switzerland: Springer, 2023.

[4] S. Chen and Z. Lu, "Hardware Acceleration of Multilayer Perceptron Based on Inter-Layer Optimization," 2019 IEEE 37th International Conference on Computer Design (ICCD), Abu Dhabi, United Arab Emirates, 2019, pp. 164-172, doi: 10.1109/ICCD46524.2019.00028.

[5] Li, T., Zhang, F., Xie, G., Fan, X., Gao, Y., & Sun, M., "A high speed reconfigurable architecture for softmax and GELU in vision transformer," *Electronics Letters*, vol. 59, 2023, doi: 10.1049/ell2.12751.