Performance Simulation of a Traffic Sign Recognition based Neural Network on Cadence's Tensilica Vision P6 DSP using Xtensa Xplorer IDE

NINAD PATIL[1], VANITA AGARWAL[2]
Department of Electronics and Telecommunication[1,2]
College of Engineering, Pune[1,2]
INDIA

Abstract: - Advanced Driver Assistance System (ADAS) technology is currently in an embryonic stage. Many multinational tech companies and startups are developing a truly autonomous vehicle that will guarantee the safety and security of the passengers and other vehicles, pedestrians on roads, and roadside structures such as traffic signal poles, traffic signposts, and other structures. However, these autonomous vehicles have not been implemented on a large scale for regular use on roads currently. These autonomous vehicles perform many different object detection/recognition tasks. Examples include traffic sign recognition, lane detection, pedestrian detection. Usually, the person driving the vehicle performs these detection/recognition tasks. The main goal of such autonomous systems should be to perform these tasks in real-time. Deep learning performs these object recognition tasks with very high accuracy. The neural network is implemented on the hardware device, which does all the computation work. Different vendors have many different hardware choices that suit the client's needs. Usually, these neural networks are implemented on a CPU, DSP, GPU, FPGA, and other custom-made AI-specific hardware. The underlying processor forms a vital part of an ADAS. The CNN needs to process the incoming frames from a camera for real-time object detection/recognition tasks. Real-time processing is necessary to take appropriate actions/decisions depending on the logic embedded. Hence knowing the performance of the neural network (in terms of frames processed per second) on the underlying hardware is a significant factor in deciding the various hardware options available from different vendors, which CNN model to implement, whether the CNN model is suitable to implement on the underlying hardware depending upon the system specifications and requirement. In this paper, we trained a CNN using the transfer learning approach to recognize german traffic signs using Nvidia DIGITS web-based software and analyzed the performance of this trained CNN (in terms of frames per second) by simulating the trained CNN on Cadence's Xtensa Xplorer software by selecting Cadence's Tensilica Vision P6 DSP as an underlying processor for inference.

Key-words - Traffic Sign Recognition (TSR), Convolutional Neural Networks (CNN, ConvNets), Multiply and Accumulate (MACs), frames per second (fps), Digital Signal Processor (DSP), Graphics Processing Unit (GPU), Xtensa Neural Network Compiler (XNNC), Xtensa Xplorer, Xtensa Processor Generator (XPG), Vector Floating Point Unit (VFPU), Graphical User Interface (GUI)

Received: March 14, 2021. Revised: January 15, 2022. Accepted: February 16, 2022. Published: March 24, 2022.

1 Introduction

ADAS enables vehicles to become more aware of their surroundings and, generally, safer to drive. ADAS covers a few functions of traffic sign recognition, automatic parking, adaptive cruise control, collision avoidance, lane departure, vehicle to vehicle communication, and Voice and Gesture Recognition [1]. As the sensors in the vehicle generate vast amounts of data per second, this data must be processed in real-time to enable quick and intelligent decision-making. Integrating all these functionalities and processing this vast amount of data is challenging and requires a unique design approach. Typically, the chips which perform these functionalities must have high compute performance (>1000GMA/s), high memory bandwidth (up to 1Gb to support high-resolution image/video streams), Hi-speed I/O, and providing optimal power, performance, and area ratio. The figure below shows an integrated ADAS SoC architecture[1].

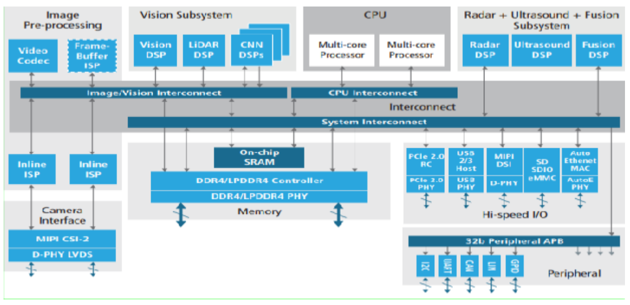


Fig.1 Integrated ADAS SoC Architecture[1]

To enable a safer driving experience, each vehicle will need to exchange vast amounts of data in real-time with other vehicles and roadside units present in the surroundings. Camera, ultrasound, radar sensors, and the LiDAR system are strategically placed inside and around the car. Front/Rear/Side view monitoring systems and driver monitoring systems are various safety systems[2]. These sensors are coupled with sophisticated AI/machine learning algorithms (deep learning) to aid quick and intelligent decision-making. Usually, an AI processor takes care of all the intense computation requirements of the algorithms.

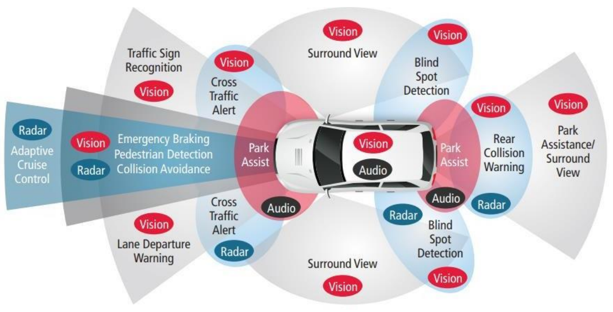


Fig.2 Various sub-systems within an ADAS environment[1]

The DSP used in this project is Tensilica's Vision P6 DSP. The figure below shows the internal block diagram of the DSP.

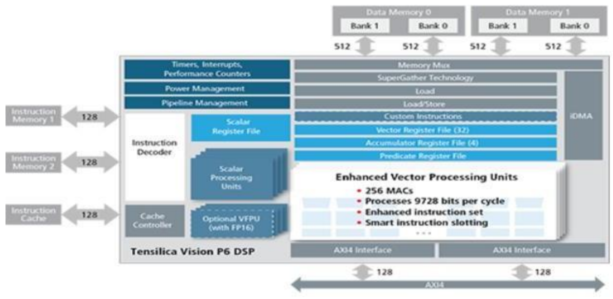
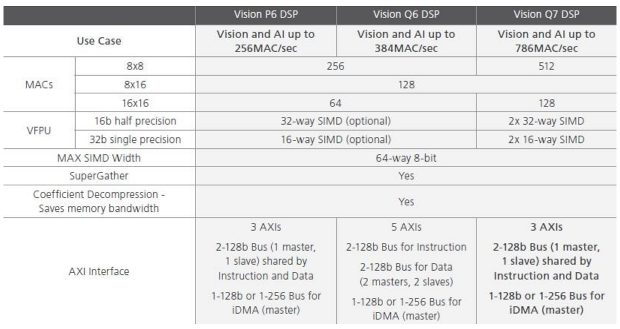


Fig.3 Vision P6 DSP Block Diagram[3]

The Vision P6 DSPs from Cadence is designed to perform image processing, computer vision & AI-specific operations efficiently[3]. Features that make this processor favorable for these tasks are as follows[3]: -

- Specially tailored Instruction sets for performing Image processing/AI-specific tasks (Add, Sub, Compare, Matrix Multiplication, Divide).
- New instructions which give an increased math throughput
- An increased number of MAC blocks for performing MAC operations is required for Image Processing/Object Recognition & AI-related tasks
- A high number of MAC operations per second
- 64-way 8-bit SIMD width
- More efficient in terms of performance and power consumption for a typical AI implementation as compared to GPU's
- With an improved 8-bit & 16-bit arithmetic, the Vision P6 DSP improves performance over convolution, FIR filtering & matrix multiplication
- 32-bit single-precision and 16-bit half-precision VFPU offers performance improvement



		Vision P6 DSP	Vision Q6 DSP	Vision Q7 DSP
Use Case		Vision and AI up to 256MAC/sec	Vision and AI up to 384MAC/sec	Vision and AI up to 786MAC/sec
MACs	8x8	256		512
	8x16	128		
	16x16	64		128
VFPU	16b half precision	32-way SIMD (optional)		2x 32-way SIMD
	32b single precision	16-way SIMD (optional)		2x 16-way SIMD
MAX SIMD Width		64-way 8-bit		
SuperGather		Yes		
Coefficient Decompression - Saves memory bandwidth		Yes		
AXI Interface		3 AXIs 2-128b Bus (1 master, 1 slave) shared by Instruction and Data 1-128b or 1-256 Bus for iDMA (master)	5 AXIs 2-128b Bus for Instruction 2-128b Bus for Data (2 masters, 2 slaves) 1-128b or 1-256 Bus for iDMA (master)	3 AXIs 2-128b Bus (1 master, 1 slave) shared by Instruction and Data 1-128b or 1-256 Bus for iDMA (master)

Fig.4 Features of Vision DSPs that enable object recognition and AI-related tasks to be performed efficiently[3]

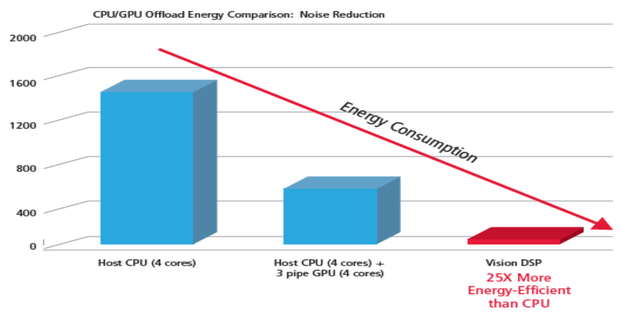


Fig.5 Energy consumption comparison between CPU, GPU, and Vision DSP[4]

A CPU or a GPU is used to implement a CNN in most cases. Compared to these traditional CPU/GPU, Vision DSPs consume significantly less energy/frame due to the improvements made in the architecture of the DSP in terms of memory bandwidth, Instruction set architecture, Hi-speed I/O and interconnects, and an increased number of MAC units which takes care of the high compute-intensive requirement of image processing/AI-specific tasks[4].

Cadence's Tensilica DSPs based on Xtensa architecture, which are configurable processors and can be custom-made (adding execution units, processor I/O interfaces, instruction sets) with the help of Xtensa Processor Developers Toolkit along with customized system design, software, and hardware (RTL) development[5][6]. As the hardware is automatically generated for the Tensilica Vision DSPs, we can synthesize these processors and implement them quickly on an FPGA. Due to the availability of software programmability, SoC designers can add elasticity to their design and improve performance.

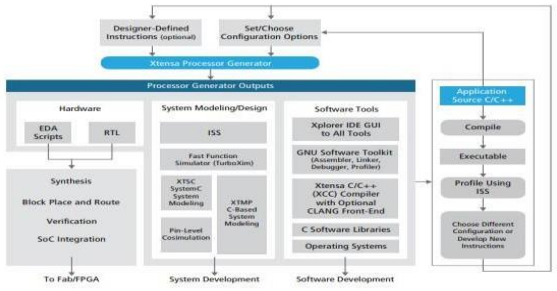

Designer-Defined Instructions (optional)
Set/Choose Configuration Options
Xtensa Processor Generator
Processor Generator Outputs
Hardware
EDA Scripts
RTL
Synthesis
Block Place and Route
Verification
SoC Integration
To Fab/FPGA
System Modeling/Design
ISS
Fast Function Simulator (TurboXim)
XTSC SystemC System Modeling
XTMP C-Based System Modeling
Pin-Level Cosimulation
System Development
Software Tools
Xplorer IDE GUI to All Tools
GNU Software Toolkit (Assembler, Linker, Debugger, Profiler)
Xtensa C/C++ (XCC) Compiler with Optional CLANG Front-End
C Software Libraries
Operating Systems
Software Development
Application Source C/C++
Compile
Executable
Profile Using ISS
Choose Different Configuration or Develop New Instructions

Fig.6 Flow diagram for XPG[6]

The Xtensa Processor Generator generates an RTL code, and along with the EDA scripts, various activities such as synthesis, block placement, routing, verification, SoC integration are performed. The processor also generates other outputs associated with the system and software development.

2 Literature Review

CNN is used for object detection/image recognition tasks owing to their accuracy in detecting/recognizing objects in the images and the speed with which they perform these tasks[7]. CNN consists of different layers connected in a cascaded manner. Some of the common types of layers are convolutional layer, fully connected layer, loss layer, pooling layer. CNN architectures are defined by the number of different layer types and how these layers are connected. Some of the common types of CNN architectures are ResNet, GoogleNet, AlexNet, MobileNet.

2.1 AlexNet Architecture

Input to AlexNet is an RGB image of size 224*224. AlexNet architecture has eight layers (5 convolutional layers followed by three fully connected layers in the end)[8]. Some of the convolution layers are followed by Max Pooling layers. Rectified Linear Unit (ReLU) f(x) = max(0,x) is used as an activation function. The three fully connected layers, in the end, feeds to the SoftMax classifier[8]. This classifier classifies the image into one of the classes depending upon the total classes available.

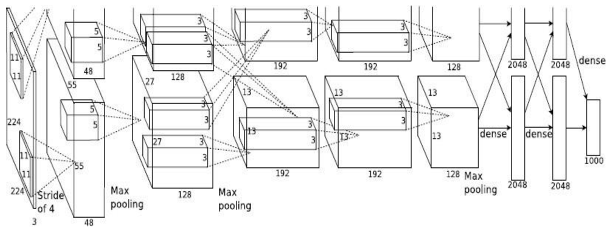


Fig.7 AlexNet Architecture[8]

2.2 Transfer Learning

It is a technique in which a neural network previously trained for some tasks is re-used for the task at hand. This technique saves much time and compute resources that would have been otherwise required for training the neural network from scratch. In this pre-trained network model, the weights have already been set. A new dataset is provided to the network. The only change is in the classifier, which is present at the end that classifies the image into one of the several labels present in the dataset. The classifier is changed depending on the number of classes present in the dataset. It also does some fine-tuning on the weights.

2.3 German Traffic Sign Dataset

Training of a neural network requires a large dataset. The dataset also needs to have various images of each class, taken in different light and weather conditions. Due to practical limitations in obtaining such a huge traffic sign dataset and variations, we downloaded the German Traffic Sign Recognition Benchmark (GTSRB) dataset used in the ICJNN competition[9]. The dataset was prepared from a video. The dataset consisted of 50,000 plus images of German road traffic signs divided into 43 classes.

2.4 Caffe

Caffe is a deep learning framework that the Berkeley Vision and Learning Centre developed at the University of California, Berkeley. Speed, expression, and modularity was the main goal behind creating the Caffe framework. It is prevalent among research scientists, startups, and academia. The network definition format is entirely different from other popular frameworks such as TensorFlow. No complex coding is required for optimization and defining the model and is done with the help of a configuration file. With the help of a single flag, we can switch between a CPU or GPU for training purposes. Once the model is trained, a .caffemodel file is generated. This file consists of floating-point weights/parameters. Other files required for training are:- 1) train_val. prototxt – this file is used to define the network architecture, the path to a binary proto file, the path to training, and the validation dataset 2) solver. prototxt – used to specify gradient descent parameters, learning rate, number of iterations, step-size, base learning rate, and other information required for training 3) mean. binaryproto – It has the mean value of images over a complete dataset for each channel that needs to be subtracted from each image[10].

2.5 Nvidia DIGITS

NVIDIA DIGITS provides a graphical user interface for training a neural network rather than a command line. DIGITS generates and modifies other training files. DIGITS is an interactive web-based tool that helps AI/ML engineers focus on training and designing the algorithms rather than debugging and other necessary pre-training tasks. DIGITS can train the neural network with high accuracy and rapidly for image recognition, image segmentation, object detection tasks. DIGITS take care of all the pre-processing steps that are performed on a dataset, selecting/uploading a pre-trained model for transfer learning, training a neural network from scratch, and providing various visualizations (in the form of graphs/charts) before, during, and after training[11]. This simplifies the training of a neural network. There are multiple image resizing options available in Nvidia DIGITS. For example, there is squash, fill (random noise), crop, half-crop.

2.6 Xtensa Neural Network Compiler

XNNC is a tool used to convert a floating-point neural network model into a fixed-point, optimized solution for Xtensa Processors. XNNC helps optimize the CNN by efficiently converting the floating-point weights to an optimized fixed-point weight without losing accuracy[12]. XNNC supports both standard and custom CNN layers and major CNN architectures. XNNC quantizes the input model. XNNC does not have a graphical user interface (GUI) and uses a command line to take inputs and generate output.

Files that are given to the input of the XNNC are:-

- Floating-point CNN definition (Accepted file formats include .caffemodel, .prototxt, .binaryproto, TensorFlow models)
- Calibration and Validation dataset

Files generated at the output of the XNNC: -

- CNN code optimized for target Xtensa DSP
- Accuracy report and other supporting files

The XNNC reads from a configuration file. In the file the path is mentioned for the following: 1) the path to the .caffemodel file is stored. 2) path where to store the output 3) path to the calibration and validation dataset. There are various other information present in the file such as name of the CNN architecture used, name of the Xtensa core, accuracy level to be used, etc., which helps to configure the XNNC. In order to generate an optimized, fixed-point version of the neural network model, the floating-point neural network goes through a 2-stage process. The first stage analyses the layer activations of the floating-point network. It evaluates quantization profiles (min/max ranges of the outputs from each layer) across a range of experimental distributions observed in the calibration and validation image sets given in the input. Based on this evaluation, scaling factors for fixed-point conversion of each layer are determined. The XNNC then takes these quantization parameters and generates an optimized, fixed-point neural network. The optimized code and other supporting software files are provided in a workspace with a .xws extension. This workspace file can be imported into Xtensa Xplorer to get a performance report[12].

2.7 Xtensa Xplorer IDE

The Xtensa Xplorer IDE acts as a GUI for the complete design process. It is the main center for custom processor development. Using Xtensa Xplorer IDE, we can do code creation for Xtensa based processors, perform system analysis, performance optimization (by addition of instruction and execution units), identify any hotspots in the systems, decide configuration options, profiling applications, and finally generate processor[6]. This reduces the time-to-market.

Features of Xtensa Xplorer IDE:-

- Efficient Xtensa C/C++ compiler (XCC)
- Xtensa assembler, debugger, linker, GNU profiler, and other utilities
- Performance, energy analysis, and project management tools
- Memory partitioning, sub-system simulation, debugging, and profiling of multi-processors
- Cycle-accurate Instruction Set Simulator
- Pipeline modeling
- Locating and vectorization of code loops with the help of a vectorization assistant

This makes it an all-in-one tool that integrates processor optimization, software development, and multi-processor SoC architecture design [6]. Xtensa Xplorer IDE is a part of the Cadence Tensilica Xtensa SDK. The various tools which are integrated inside speed up the software development process.

3 Methodology

3.1 Training of the CNN for traffic sign recognition task

A pre-trained AlexNet Caffe model was downloaded from the Caffe model zoo[13]. Using the transfer learning approach, the model was trained for traffic sign recognition tasks using the GTSRB dataset on Nvidia DIGITS[11]. The model was trained on 43 different classes of traffic signs on NVIDIA DIGITS. This model was then uploaded onto the DIGITS software along with the traffic sign dataset consisting of 43 image classes in jpeg format. The images were resized to a fixed resolution. After training was completed, a .caffemodel file was generated along with prototxt and binaryproto files.

3.2 Preparing calibration and validation dataset

The German Traffic Sign Recognition Benchmark (GTSRB) dataset, which was downloaded, had 43 traffic sign classes. The XNNC supports only .ppm, .jpeg, .pgm image file formats. As the images in the dataset were in png format, they were first converted from png to ppm format. These images were then divided into calibration and validation datasets. Each dataset had 11 images of each of the 43 classes. Hence each dataset had a total of 473 images.

3.3 Compilation using Xtensa Neural Network Compiler

The .caffemodel file generated after the training phase is fed to the XNNC software for compilation along with the traffic sign calibration and validation dataset[12]. After installing XNNC, the .caffe model file was given input to the XNNC along with the calibration and validation dataset. The image format was ppm, and the image resolution was 224*224*3. Both these datasets contain 11 images of each 43 classes. In total, there were 11*43 = 473 images in each dataset. There are a total of 2 stages in the compilation: -

3.1.1 Analysis Stage

The Analysis stage is executed using calibration images, validation images, and network definition. The Analysis stage maps the network layers to their XI-CNN library equivalents. It empirically evaluates the activations of each layer to determine appropriate quantization parameters for the model weights and library functions[12]. Analysis Stage Options:-

- Quantization Accuracy Levels:
  - Level 0 (default – better performance, lesser detection quality) – This level was selected for this task
  - Level 1 (Balanced)
  - Level 2 (better detection quality)
- Adjustable thresholds for min/max selection and outlier removal for quantization calibration
- Layer wise manual modification of min/max values for quantization
- Control of input/output data types for custom layers

3.1.2 Optimization Stage

After the analysis stage, the execution of the optimization stage takes place where it takes the quantized network and fixed-point parameters as inputs and generates the optimized code for the target Xtensa processor in the output[12]. The Optimization stage determines how local memory is partitioned, the storage formats of data and coefficients, and the DMA tiling strategy for the network[12]. Optimization Stage Options:-

- Layer-wise declaration of data storage formats (WHD, DWH)
- Memory Configuration Options
- Use default configuration from the target DSP's core parameters
- Treat dual local RAMs as contiguous or non-contiguous
- Adjust the size of memory reserved for stack and statically allocated data
- Batching (process a series of images to help reduce memory bandwidth)

When the compilation process is complete, optimized code is generated for the target Xtensa processor, along with a summary of the expected detection quality of the quantized, fixed-point network. Caffe model is required to compile the network model using the XNN compiler[12]. Alternatively, we can use a TensorFlow model, but this approach will require one additional step, converting the TensorFlow model into the Caffe model. If we use Caffe, we can bypass this conversion step, as shown in the image below.

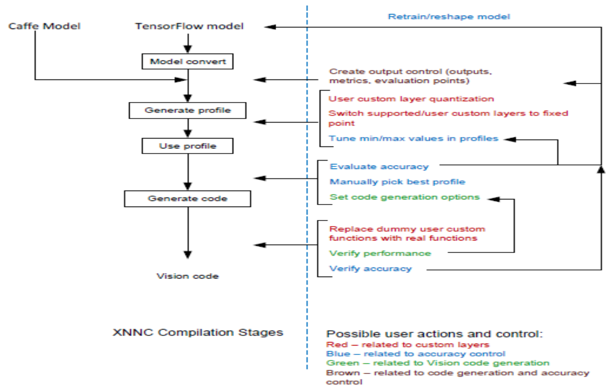

Caffe Model
TensorFlow model
Retrain/reshape model
Model convert
Create output control (outputs, metrics, evaluation points)
User custom layer quantization
Switch supported/user custom layers to fixed point
Generate profile
Tune min/max values in profiles
Use profile
Evaluate accuracy
Manually pick best profile
Set code generation options
Generate code
Replace dummy user custom functions with real functions
Verify performance
Verify accuracy
Vision code
XNNC Compilation Stages
Possible user actions and control:
Red – related to custom layers
Blue – related to accuracy control
Green – related to Vision code generation
Brown – related to code generation and accuracy control

Fig.8 XNNC Compilation Stages and User Control[12]

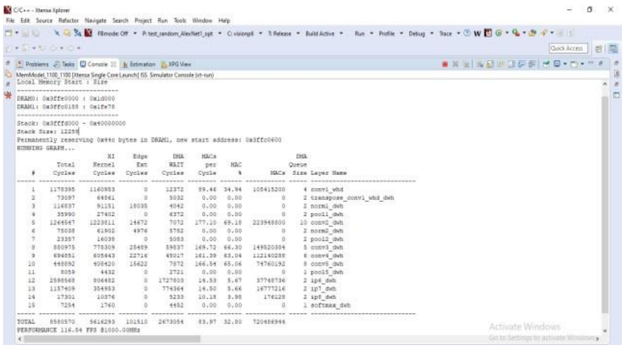
3.4 Viewing the performance report on Xtensa Xplorer IDE

The optimized code generated by the XNNC, along with other supporting software files needed to get the performance report, is given as input to the Xtensa Xplorer IDE in the form of a workspace[12]. The extension of this workspace file is .xws. To analyze the CNN's performance, Xtensa Xplorer IDE needs to be configured by importing the Vision P6 DSP core. Once the Xtensa IDE is configured for Vision P6 DSP, then the performance report of the neural network can be generated by importing the previously generated workspace from XNNC and running the simulation. The simulated result would be approximately equal to the actual result if the CNN was implemented on Vision P6 DSP hardware.

4 Results and Discussions

After training the neural network on Nvidia DIGITS a .caffemodel is generated. This file is given as input to the Xtensa Neural Network Compiler. A workspace file with the extension .XWS is generated at the output of XNNC which is then given to the Xtensa Xplorer for generating the performance report. The performance report for the trained CNN (AlexNet) was generated by running the simulation on the Xtensa Xplorer software. The figure below shows the performance report generated on the Xtensa Xplorer software. In the figure, below the toolbar option we can see that in the P: section, the neural network used is AlexNet, and in the C: section, the software has been configured to run the AlexNet on Vision P6 DSP. The CNN (AlexNet) performance was 116.54 FPS at a clock speed of 1000.00 MHz. The total number of MAC operations required to process one frame is 720 million. The number of cycles required to process one frame is around 8.5million (precisely 8580570). The processor clock speed is 1000 MHz, i.e., 1000000000 clock cycles in 1 second. Thus 1000000000/8580570 comes out around 116.54 fps which indicates it can process one frame in 8.58ms. It can be concluded from the "Layer Name" column that the neural network used for the TSR task is AlexNet. There are convolutional layers (conv1, conv2, conv3, conv4, conv5), fully connected layers (ip6, ip7, ip8) after convolutional layers, and at the end, a SoftMax layer. Convolutional layers 1 and 2 are followed by batch normalization and max pooling. The convolutional layers 3,4,5 are cascaded, followed by max-pooling in the end.

We can conclude from this result that convolutional & fully connected layers perform most of the MAC operations. Other related information has been provided in the various columns. Some of the columns indicate the total number of cycles, DMA wait cycles (Number of cycles the request must wait to access the memory directly via a system bus), DMA Queue Size (the number of DMA requests queue reserved in local memory), number of MACs per cycle, and the total number of MAC operations for each layer.


C/C++ - Xtensa Xplorer
File Edit Source Refactor Navigate Search Project Run Tools Window Help
FBmode Off
P: test_random_AlexNet1_opt
C: visionp6
T: Release
Build Active
Problems
Tasks
Console
Estimation
XPG View
MemModel_1100_1100 [Xtensa Single Core Launch] ISS Simulator Console (xt-run)

```
Local Memory Start : Size
---------------------------
DRAM0: 0x3ffe0000 : 0x1d000
DRAM1: 0x3ffc0188 : 0x1fe78
---------------------------
Stack: 0x3fffd000 - 0x40000000
Stack Size: 12288
Permanently reserving 0x440 bytes in DRAM1, new start address: 0x3ffc0600
RUNNING GRAPH...
                       XI     Edge       DMA     MACs                        DMA
          Total    Kernel      Ext      WAIT      per    MAC                Queue
    #    Cycles    Cycles   Cycles    Cycles    Cycle      %        MACs  Size Layer Name
----- ---------- ---------- -------- --------- -------- ------ ----------- ----- ---------------------------
    1   1170395   1160953        0     12372    89.46  34.94   105415200     4 conv1_whd
    2     73097     64561        0      5032     0.00   0.00           0     2 transpose_conv1_whd_dwh
    3    116837     91151    18035      4042     0.00   0.00           0     2 norm1_dwh
    4     35990     27402        0      6372     0.00   0.00           0     2 pool1_dwh
    5   1264547   1223811    14672      7072   177.10  69.18   223948800    10 conv2_dwh
    6     75038     61902     4976      5782     0.00   0.00           0     2 norm2_dwh
    7     23357     16038        0      5083     0.00   0.00           0     2 pool2_dwh
    8    880975    778309    25489     59837   169.72  66.30   149520384     5 conv3_dwh
    9    694851    605443    22716     45017   161.39  63.04   112140288     6 conv4_dwh
   10    448892    408420    15622      7872   166.54  65.06    74760192     8 conv5_dwh
   11      8059      4432        0      2721     0.00   0.00           0     1 pool5_dwh
   12   2598568    806482        0   1727803    14.53   5.67    37748736     2 ip6_dwh
   13   1157409    354953        0    774364    14.50   5.66    16777216     2 ip7_dwh
   14     17301     10376        0      5233    10.18   3.98      176128     2 ip8_dwh
   15      7254      1760        0      4452     0.00   0.00           0     1 softmax_dwh
----- ---------- ---------- -------- --------- -------- ------ ----------- ----- ---------------------------
TOTAL   8580570   5616293   101510   2673054    83.97  32.80   720486944
PERFORMANCE 116.54 FPS @1000.00MHz
```

Fig.10 Performance Report of AlexNet trained for TSR on Cadence Tensilica Vision P6 DSP

5 Conclusion

The simulation concluded that the AlexNet CNN processed the input video frames at 116.54 FPS at a processor clock speed of 1000 MHz on Vision P6 DSP. Convolutional and fully connected layers perform a significant portion of the total MAC operations. The total number of MAC operations required per frame is around a 720million. The total cycles required for the processing of one frame are 8580570.

6 Future Works

In the future, more efficient and optimized CNN architectures such as ResNet and GoogleNet can be trained on the GTSRB dataset, improving the performance results significantly. Performance comparison of different CNN architectures can also be made, which will set a benchmark for the performance of a CNN and can also serve as a reference model for future research. It can also help us decide which architecture to implement in an actual self-driving car. Current results assume ideal conditions that there will be no error while recognizing the traffic sign. The overall system can be more robust by considering different weather conditions and other scenarios and training the neural network on these datasets.

7 Acknowledgment

I want to thank Late Prof A.B.Patki from the E&TC department, Centre of Excellence (CoESIP) & VLSI design lab staff for all the guidance and help provided in completing this project. I would also like to thank Mr. Sarang Shelke and Mr. Abhishek Belkonikar of Cadence Design Systems, Inc, who helped me complete this project and solve queries regarding Cadence's XNNC and Xtensa Xplorer software.

References:

[1] Cadence Design Systems, Inc. (2017). Developing Smarter, Safer Cars with ADAS IP [White Paper]. Cadence Design Systems, Inc. https://ip.cadence.com/uploads/1195/cdn-wpt-auto-ip-sys-design-enablement-pdf

[2] Cadence Design Systems, Inc. (2019). Computer Vision and AI for Automotive Safety: Staying Alert on the Road [White Paper]. Cadence Design Systems, Inc. https://ip.cadence.com/uploads/1253/13044_VisionQ7_Automotive_TB_FINAL-pdf

[3] Cadence Design Systems, Inc. (2019). Tensilica Vision DSP Family [White Paper]. Cadence Design Systems, Inc. https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/ip/tensilica-ip/TIP-PB-Vision-DSP-FINAL.pdf.

[4] Cadence Design Systems, Inc. (2015). Choosing the Right DSP for High-Resolution Imaging in Mobile and Wearable Applications [White Paper]. Cadence Design Systems, Inc. https://ip.cadence.com/uploads/899/TIP_WP_Vision_P5_Final-pdf

[5] Cadence Design Systems, Inc. (2014). Xtensa Processor Developer's Toolkit [White Paper]. Cadence Design Systems, Inc. https://ip.cadence.com/uploads/102/HWdev-pdf

[6] Cadence Design Systems, Inc. (2014). Tensilica Software Development Toolkit (SDK) [White Paper]. Cadence Design Systems, Inc. https://ip.cadence.com/uploads/103/SWdev-pdf

[7] Cadence Design Systems, Inc. (2015). Using Convolutional Neural Networks for Image Recognition [White Paper]. Cadence Design Systems, Inc. https://ip.cadence.com/uploads/901/TIP_WP_cnn_FINAL-pdf

[8] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In NIPS, pp. 1106–1114, 2012.

[9] Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural Networks, 32, 323-332.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093, 2014.

[11] NVIDIA Corporation. (2017). NVIDIA DIGITS [White Paper]. NVIDIA Corporation. https://docs.nvidia.com/deeplearning/digits/pdf/DIGITS-User-Guide.pdf

[12] Cadence Design Systems, Inc. (2018). Xtensa Neural Network Compiler User Guide [White Paper]. Cadence Design Systems, Inc

[13] S. Lapuschkin. (2019) Model Zoo. [Online]. Available:https://github.com/BVLC/caffe/wiki/Model-Zoo