# **Data Processor for Fully Connected Layer Operations of a CNN**

**Julia Shan**

722572854, jsha485

# Introduction

In recent times, there has been a significant increase in machine learning based applications [franco]. Deep learning, a subset of machine learning has made signification breakthroughs in artificial intelligence. Most commonly in the form of a deep neural network, deep learning methods are able to classify, recognise and predict various forms of data. The convolutional neural network (CNN) is one of the most widely used deep learning methods due to its high performance and accuracy in computer vision and image classification. They have also been used for audio classification such as environment sound classification and audio generation [1]. They require relatively little pre-processing compared to other networks such as the recurrent neural network (RNN) and the multi-layer perceptron (MLP).

This report details research undertaken to implement operations of the classification stage of a CNN on an application specific processor data processor (DP-ASP). This data processor is to be implemented as part of a heterogeneous multi-processor system on chip (HMPSoC) connected to a time-domain multiple access multistage interconnect network. The DP-ASP will communicate through this network-on-chip (NoC) to a ReCOP, a reactive and concurrent processor to a non-time critical network of general-purpose processors such as a Nios II. This DP-ASP will be pipelined with others to build a full multi-layer neural network.

This neural network will be designed for audio classification. The overall SoC will take an audio input which will then be converted in a spectrogram image representation to be passed into the neural network processors. For audio recognition and to make full use of environment signals, data will be streamed through an analogue to digital converter before being converted into a pixel image format that CNN perform operations on.

The purpose of this data processor is to implement and execute the mathematical algorithms involved in the classification stage of a convolutional neural network. Generally, this comprises of a dense/fully connected layer, with an activation function for classification. This processor is focused on implementing the matrix multiplication and other mathematical functions, without training. Hence training, backpropagation and updating weights are out of scope. This is purely an implementation of the algorithms for inference without learning.

The aim of this individual research component is to implement an acceleration of algorithms in the fully connected layer(s) of a convolutional neural network (DP-D). This includes taking a weighted set of neurons as a vector input, performing a dense operation to connect all input neurons to each neuron in an output vector and using an activation function for either further operations or classification. The scope of the project includes implementing mathematical representations of fully connected layer algorithms through register transfer level (RTL) design. The processor will not be concerned with the training of the network and hence will not rely on a dataset. The DP-ASP unit will be optimised for performing CNN operations that are also included at the training stage without any backpropagation and learning of weights. Hence output values will be arbitrary but will ideally demonstrate the successful implementation of the algorithm operations for the classification stage.

# Literature Findings

The CNN model was first proposed in 1989 by Yann LeCun. Significant developments have been made since its inception to the complexity and applicability of the convolutional neural network, with a focus on hardware acceleration. Today, graphical processing units (GPU) are the preferred hardware implementation for training neural networks due to their high computational performance and well-established programming interface, however they suffer from poor energy efficiency and inadequate parallelism functionality. Hence there has been a shift towards more special purpose solutions such as field programmable gate arrays (FPGAs) as they have a high degree of parallelism, reconfigurability and energy efficiency [2].

The most effective hardware accelerators were focused on hardware-level design, which was shown to improve the efficiency and acceleration; this requires an in-depth understanding of CNN algorithm structure and the FPGA system architecture [3]. Taking advantage of the reconfigurability of an FPGA by optimising on the hardware level gave the most accurate results. The biggest issues with CNN acceleration the high computational intensity of convolution operations and the memory intensive classification operations, impact the ability the achieve real-time performance with high energy efficiency [3,4].

The major focus of existing research on FPGA acceleration for CNNs is running operations on a trained network. Different approaches include using high level synthesis tools for performance optimisation and register transfer level design. As the weights and biases for this implementation are not being iteratively updated with meaningful, literature reviews were undertaken into the effect of using random or arbitrary initial values as well as CNN inference without training or a data set. Researchers found that the structure of a network can capture a large amount of low-level image information prior to any learning [5]. This was proven by applying randomly initialised weight values to an untrained CNN. Alternatively, using a fixed Hadamard matrix for the final classification layer did not hinder the performance of a classifier [6].

A big bottleneck in performance is due to the softmax function. The computational intensity of calculating and storing multiple values in memory. There is also an added difficulty of representing the Euler’s number, hence researchers have found ways of approximating this function. One such method is the Maclaurin series, which reduces the number of logic resources. Research into activation functions concluded that rectified linear unit activation for hidden layers and softmax for output were the best options for a multi-class network [7, 8].

# Implemented Algorithms

The algorithms implemented in this DP-ASP are those that are found in the classification stage of the neural network indicated by fc-3 and fc-4 as seen in figure 1.



Figure 1: Convolutional Neural Network Architecture

The output from the previous pooling operation and convolutional layer is taken in as an input. This input is flattened into a 1D vector, before producing a dot-product output with a weight matrix. The matrix weight contains weights for every connection from an input node to an output node (as seen in figure 2). A bias matrix is then summed with the dot product output. An activation function is then applied to the output vector. This operation is expressed in the equation below:

(1)

represents the activation function, used to introduce non-linearity or to make a multi-class prediction

is a 2D weight matrix with dimensions [input\_size, output\_size]

is a flattened 1D vector of inputs

is a 1D vector of bias values, with the same length as the output vector

The dot product operation connected every value in the input vector to every value in the output vector through a weight matrix. The effect of this operation is shown below:

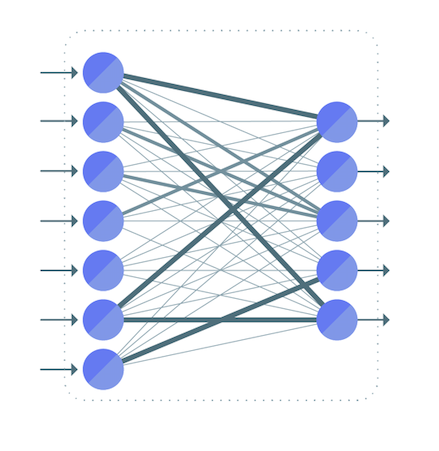
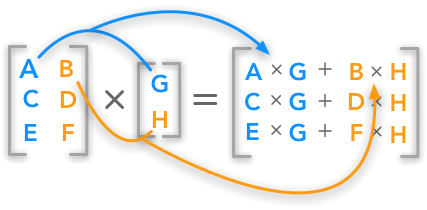


Figure 3: Fully connected layer connections

Figure 2: Matrix-vector dot product

Activation Functions

Different activation functions are required depending on which layer of the network is being processed. For hidden (i.e non-output layers) layers, a rectified linear activation function (ReLu) will be used to determine the ‘firing strength’ of each neuron. This function represented in equation 2 returns the value of an input neuron if it is greater than zero, otherwise for values of 0 or less, 0 will be

(2)

For the final output layer, the values must represent a probability distribution over several predicted output classes. The probabilities of each output must sum to 1, and hence a softmax function is implemented to perform the multi-class classification. The equation of the softmax function as seen in (3), takes the Euler’s number of each element of the vector z, divided by the sum of all Euler’s numbers for the whole vector.

(3)

An approximation for the Euler’s number can be calculated using the McLaurin series expansion. A series of 5 terms gives a high accuracy without large trade offs in resource use [9].

(4)

(5)

# Design

DP\_D performs one fully connected layer operation; it can be configured for a hidden layer or an output layer. Signals and information for configuration of the DP-ASP will be passed through the NoC, however as the size of the data exceeds the 32-bit data interface limit, inputs and outputs for each DP-ASP will be passed through a scratchpad memory with point-point connections between each DP-ASP. Input data for DP-D will be streamed through this memory and stored in the component. The data being streamed to the input of the processor will be in chunks of 32 unsigned integers in the form of 16-bit standard logic vectors. There are 32 integers in each row of the 2D matrix output from the previous DP-ASP. This will be passed directly into a 1D array. The input array size N and output array size M are declared as generics; these values be configured by the data messages that are passed through the NoC. The output array size will vary depending on if the operation is for a hidden layer or the output layer (for which the output size will be the number of classes). The output port signal for streaming data out of the DP-ASP will be the same as input size.

Diagram

Description automatically generated

Figure 4: Control unit

As seen in figure 4, the control unit receives, decodes and sends messages between the DP-ASP and the NoC. The recv input contains configuration information which is then sent to the data path. For example, a data\_read signal is passed to the data path, indicating that data from the point-to-point memory connect is ready to be streamed, through the data\_in line. Conversely, the data\_write signal notifies the control unit that the data path is ready to write output data to data\_out. The data\_read and data\_write signals are represented by std\_logics, data\_in and data\_out are arrays of 32 16-bit std\_logic\_vectors. Layer is a configuration signal send by the control unit to the datapath.

Diagram, schematic

Description automatically generated

Figure 5: Data path

Figure 5 represents the data path of the DP-ASP. The input data stream is stored into a 1D input vector of size N, depending on the data\_read control signal. This stored input array is then multiplied with a weight matrix with N x M dimensions, where M is the output vector size. This is broken down to the multiplication of individual array elements, before being summed together in the output[M] vector. The result of the dot product operation is stored and then passed into a multiplexer to determine where the data gets sent. The step were a bias vector is added to the result of the dot product is omitted, as the biases would not add any real value to the results of the network. However, it would be a simple change to implement this, just adding each value at the same array element for the output [M] vector and a bias vector of the same dimensions. The mux takes in the ’layer’ control signal, which decides what part of the over neural network structure is. As there can be multiple-fully connected layers in a CNN, the DP-ASP must also be able to make this distinction in order to select the right activation function. If the ‘layer’ control signal is 1, this means that this operation is for the output layer of the overall network, requiring the softmax activation function. The output[M] vector is passed to the upper branch, where an approximation of is calculated for every element in the output array and stored in exp[M]. Each approximation value is also summed together. Then for an element k in exp[M], the of that value is divided by the summed total of all the approximations. This then gets stored back in output[M] before being passed through the data\_out signal and writing to the data\_write signal. If the ‘layer control signal is 0, this means that the current operation is for a hidden layer. Hence, we select ReLU activation function instead. The output[M] vector is passed through the max(output[k], 0). For each element k of the output vector, if the value is greater than 0, then the value is passed through. Otherwise, the value for that element is set to zero. This is stored back into output[M] before being pass to the output streaming stage, with the same operation as in the softmax path.

Below figure 6 shows a visual representation of the internal arrays implemented in the ASP. A N dimension vector is multiplied by a N x M weigh matrix to give an out of a M dimensional output vector. The input must be a 1D vector in order to produce a 1D output vector, which is required for the final output layers. N and M are defined as generics in the current implementation.

Table

Description automatically generated

Figure 6: Data array implementation for dot product operation

Graphical user interface, application, table, Excel

Description automatically generated

Figure 7: TDAM-MIN message protocol for NoC

Figure 7 provides the NoC interface for passing messages through the TDMA-MIN. Bits 30-29 (111) represent the id of this DP-D data processor. Bits 27 to 20 are reserved for ReCOP configurations. An enable signal is implemented in bit 19, while the layer control signal is represented by bit 18. Row, IN and OUT are the values that are either written by the ReCOP or the previous DP-ASP in the pipeline to determine the input size, for streaming and storing data, and to determine the number of outputs, which will vary depending on the current layer or number of classes.

# Discussion

During the implementation, there were number of challenges that arose. Firstly, initialising and working with large sizes of data not being passed through NoC messages meant that Quartus was unable to complete a full synthesis and analysis. I found that compiling just the main DP-ASP VHDL file containing the processor entity would result in error due to the lack of required pins. However, while the overall compilation failed, the analysis and synthesis of the DP-ASP was able to determine the number of required combinational ALUTs and logic registers. With a top level added, essentially block the use of these ports, the overall project compiles, however much of the logic is optimised and hence no statistics are available for the synthesis of the processor. This also meant that the data processor would not be able to undergo timing analyses and hence I could not find out what the fmax value was. From the Quartus analysis and synthesis summary, we can see that there is a high number of combinational ALUTs (288) and logic registers (144) used. This was not unexpected, given our data sizes and computationally intensive operations. This bottleneck is a key point for future consideration. Finding ways of reducing resources used by looking at alternative ways execute array arithmetic. A multi-cycle approach could also be implemented in the future for higher efficiency.

Text

Description automatically generated with medium confidence

Figure 8: Quartus analysis and synthesis report summary

The DP-ASP implementation was tested and validated through ModelSim. Due to the large size of the data, as well as the complex Euler’s constant operation, valid output could not be accurately determined. There were also some issues with the arithmetic implementation, as all the algorithms included operations that weren’t suited to the std\_logic\_vector type. Hence, converting all these vectors into integers also posed some issues. A computer screen capture

Description automatically generated with low confidence

Figure 9: Modelsim Results

As seen from the simulation results, I can verify that the data steam (initiated with a value of 255) are stored in the DP-ASP correct, indicating a correct response to the data\_read signal. The data\_write signal is also set and a defined output (of 0s) can be observed.

**Conclusion**

Throughout my research I found that the mathematical algorithms and operations were not difficult in themselves but implementing them in such a way that can cater to a large array data type posed some issues. Converting std\_logic\_vectors to integers resulted in some meta values, and large data arrays posed memory problems. A deeper understanding of register transfer level design and parallel optimisation would have resulted in a comparable output.

There are also several other considerations for future improvement of the DP-ASP. Firstly, to achieve a higher accuracy, high precision data types could be used, such as floating point. However, this also has the risk of increasing the computational intensity. While having extremely high precision is not typically necessary during classification to achieve good accuracy levels [3] , it is a key point to note that most CNNs have lot of floating point values. Another consideration would be to reduce the memory access of the ASP. By optimising the number of signals and looking into how large amounts of data can better be stored, this would lead to an increase in efficiency. With a successful integration of the DP-ASP into the overall HMPSoC, the implementation of this neural network functionality could provide the basis for a full application in the future, running a trained model through the architecture with appropriate datasets for a more accurate classifier.

# References

[1] K. Palanisamy, D. Singhania, and A. Yao, ‘Rethinking CNN Models for Audio Classification’. arXiv, Nov. 13, 2020. Accessed: May 24, 2022. [Online]. Available: <http://arxiv.org/abs/2007.11154>

[2] S. Zhai, C. Qiu, Y. Yang, J. Li, and Y. Cui, ‘Design of Convolutional Neural Network Based on FPGA’, *J. Phys.: Conf. Ser.*, vol. 1168, no. 6, p. 062016, Feb. 2019, doi: [10.1088/1742-6596/1168/6/062016](https://doi.org/10.1088/1742-6596/1168/6/062016).

[3] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J. Seo, ‘ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler’, *Integration*, vol. 62, pp. 14–23, Jun. 2018, doi: [10.1016/j.vlsi.2017.12.009](https://doi.org/10.1016/j.vlsi.2017.12.009).

[4] S. Sahin, Y. Becerikli, and S. Yazici, ‘Neural Network Implementation in Hardware Using FPGAs’, in *Neural Information Processing*, Berlin, Heidelberg, 2006, pp. 1105–1112. doi: [10.1007/11893295\_122](https://doi.org/10.1007/11893295_122).

[5] D. Ulyanov, A. Vedaldi, and V. Lempitsky, ‘Deep Image Prior’, *Int J Comput Vis*, vol. 128, no. 7, pp. 1867–1888, Jul. 2020, doi: [10.1007/s11263-020-01303-4](https://doi.org/10.1007/s11263-020-01303-4).

[6] A. Rosenfeld and J. K. Tsotsos, ‘Intriguing Properties of Randomly Weighted Networks: Generalizing While Learning Next to Nothing’. arXiv, Feb. 02, 2018. Accessed: May 28, 2022. [Online]. Available: <http://arxiv.org/abs/1802.00844>

[7] ‘Fully Connected Deep Networks - TensorFlow for Deep Learning [Book]’. <https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html> (accessed May 24, 2022).

[8] ‘Fully Connected Layer: The brute force layer of a Machine Learning model’, *OpenGenus IQ: Computing Expertise & Legacy*, Mar. 02, 2019. <https://iq.opengenus.org/fully-connected-layer/> (accessed May 24, 2022).

[9] M. A. M. Alshahrani, ‘FPGA IMPLEMENTATION OF APPROXIMATE SOFTMAX FUNCTION FOR EFFICIENT CNN INFERENCE’, *Open Access*, vol. 03, no. 09, p. 11.