Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Unidade Acadêmica de Engenharia Elétrica

LUCAS FARIAS MARTINS

IMPLEMENTAÇÃO EM HARDWARE DE UM MÓDULO DE CONVOLUÇÃO QUANTIZADA PARA REDES NEURAIS

Campina Grande
Agosto de 2022

LUCAS FARIAS MARTINS

IMPLEMENTAÇÃO EM HARDWARE DE UM MÓDULO DE CONVOLUÇÃO QUANTIZADA PARA REDES NEURAIS

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr, Gutemberg Gonçalves dos Santos Júnior

Área de Concentração: Eletrônica

Campina Grande

Agosto de 2022

LUCAS FARIAS MARTINS

IMPLEMENTAÇÃO EM HARDWARE DE UM MÓDULO DE CONVOLUÇÃO QUANTIZADA PARA REDES NEURAIS

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do título de Bacharel em Engenharia Elétrica.

Trabalho Aprovado em: / /

_____

Gutemberg Gonçalves dos Santos Júnior, D.Sc.

Orientador

_____

Marcos Ricardo Alcântara Morais, D.Sc.

Avaliador

Campina Grande

Agosto de 2022

Dedico este trabalho aos meus pais e professores
Alexandre e Wanderleya.

AGRADECIMENTOS

À minha família, por todo amor e zelo demonstrados e proporcionados ao longo de toda minha vida.

A todos os professores e colaboradores do Departamento de Engenharia Elétrica da UFCG, que foram indispensáveis para minha formação, especialmente aos professores Alexandre Cunha e Marcos Morais, pela maior participação da minha formação e pelas lições tanto técnicas quanto éticas.

À Wendy, pelo relacionamento gratificante e pela ajuda na minha formação e nos momentos inesquecíveis que passamos.

Aos amigos George e Wendel, pelo companheirismo inestimável ao longo desses anos de curso e pela admiração que tenho por eles como pessoas.

Ao amigo Thiago, também pelo companheirismo e admiração, mas também pelo auxílio técnico na minha formação acadêmica.

À todos os demais amigos que passaram pela minha vida em Campina Grande, especialmente Marcos, Greg, Daniel e Matheus.

E ao professor Gutemberg, pela excelente participação como professor, orientador e coordenador de projeto.

A verdadeira viagem de descobrimento não consiste em procurar novas paisagens, e sim em ter novos olhos. (PROUST, Marcel).

RESUMO

O presente trabalho mostra uma implementação no fluxo de circuitos integrados de aplicação específica (ASIC) para um algoritmo de convolução para inteiros de 8 bits a nível de lógica digital própria de sistemas computacionais. Será apresentado todo o funcionamento do módulo, como lógica de endereçamento, de armazenamento de dados e controle de sinais, por meio de textos, diagramas - de bloco e de transição de estado - e exemplos ilustrativos do passo-a-passo das rotinas. Para isso, serão introduzidos todos os conceitos básicos para a atingir a compreensão do contexto e da relevância do trabalho como aplicação prática e teórica, de maneira a manter um fluxo de aprendizado linear e conciso. O módulo faz parte do projeto da Unidade de Processamento Neural eXtensível (XNPU), um acelerador escalável para redes neurais desenvolvidos por integrantes do Núcleo de Pesquisa Virtus e do Laboratório de Excelência em Microeletrônica do Nordeste (XMEN). O projeto continua em desenvolvimento, englobando atualmente parte dos módulos implementados em SystemVerilog e suas respectivas modelagens em alto nível na linguagem Python. O ambiente que o acelerador irá trabalhar é fortemente inspirado no framework do TensorFlow da Google, pois é de extrema utilidade poder traduzir informação de uma ferramenta demasiado difundida em pesquisas de aprendizado de máquina. Por fim, será mostrado o resultado de um exemplo demonstrativo comparado com um modelo em python, validando os resultados e fazendo uma análise final do andamento do bloco.

Palavras-Chave: Convolução, Hardware, Redes Neurais, Quantização.

ABSTRACT

The present work shows an implementation in the flow of application-specific integrated circuits (ASIC) for a convolution system for the entire set of 8 bits at the level of digital logic typical of computer systems. The module's entire functioning will be presented, such as addressing logic, data storage and signal control, through the use of texts, diagrams - block and state transition - and illustrative examples of step-by-step routines. For this, all the basic concepts will be introduced to achieve an understanding of the context and relevance of the work as a practical and theoretical application, in order to maintain a linear and concise learning flow. The module is part of the eXtensible Neural Processing Unit (XNPU) project, a scalable accelerator for integrated neural networks developed by researchers from Virtus research center and the Northeast Microelectronics Excellence Laboratory (XMEN). The project continues under development, currently encompassing part of the modules implemented in SystemVerilog and their respective high-level modeling in the Python language. The environment the accelerator will work in is strongly inspired by Google's TensorFlow framework, as it is extremely useful to be able to translate information from a tool that is widespread in machine learning research. Finally, the result of a demonstrative example will be shown compared with a model in python, validating the results and making a final analysis of the block's progress.

Keywords: Convolution, Hardware, Neural Networks, Quantization.

LISTAS DE ABREVIATURAS E SIGLAS

ADDR	Endereço
CNT	Contador
CNN	Redes neurais convolucionais
CQint8	Convolução quantizada em inteiros de 8bits
GPIO	Entradas e saídas de uso genérico
LIT	Sistemas lineares e invariantes no tempo
MEM	Memória
RTL	Nível de transferência de registradores
UART	Receptor/transmissor assíncrono universal
USB	Barramento universal serial
XMEN	Laboratório de Excelência em Microeletrônica do Nordeste
XNPU	Unidade de processamento neural extensível

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

No atual cenário de avanços tecnológicos fervorosos e altamente estimulados, sem dúvida uma das áreas de maior crescimento e aplicação é a de inteligência artificial. Muitos problemas da sociedade moderna já são simplesmente solucionados com a eletrônica e programação convencional, desde detecção de pedestres (Vargas *et al.,* 2019) até o auxílio no diagnóstico de câncer (Wurzel e Martins, 2022). Todavia a busca por desafios mais complexos, como classificação de imagens , mecanismos de segurança e investimentos na bolsa de valores, tem encontrado respaldo no rápido avanço de algoritmos inteligentes eficientes.

Uma inteligência artificial normalmente é criada a partir de uma rede neural, um sistema que tenta mimetizar o comportamento dos neurônios naturais. Tal realização se torna demasiado difícil de ser executada visto que o cérebro possui inúmeras conexões e elementos, contudo as tecnologias recentes como *machine* e *deep learning* apresentam resultados mais que satisfatórios para problemas que consegue até superar humanos, a exemplo de jogos de tabuleiro como o Alpha Go (Chen, 2016).

A limitação que mais atinge tal ciência é a capacidade de processamento e armazenamento computacional, uma vez que sua implementação exige a presença de muitos elementos interligados entre si. De acordo com Sze *et al.* (2017), em muitas aplicações é preferível um processamento local em vez da computação em nuvem devido a preocupações com privacidade ou latência, ou limitações na largura de banda. Redes Neurais Convolucionais ou CNNs são eficientes nesse quesito devido ao seu aspecto de extração de características de um certo conjunto de dados.

O presente trabalho visa implementar o módulo digital que realiza a operação linear da convolução para a Unidade de Processamento Neural Extensível (XNPU), um *hardware* dedicado com uma arquitetura de acelerador escalável para redes de aprendizado profundo de larga escala (Cruz *et al*., 2022). Tal projeto atualmente está sendo desenvolvido por integrantes do laboratório de Excelência em Microeletrônica do Nordeste (XMEN) em conjunto com pesquisadores do núcleo de pesquisa Virtus.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. Convolução

A convolução é uma ferramenta matemática própria de sistemas lineares e invariantes no tempo (LIT), uma classe de sistemas que, como descrito por Carvalho *et al.* (2015), tem sua relevância na engenharia elétrica pelo fato que podem ser usados para modelar fenômenos físicos reais de interesse prático nas mais diversas aplicações, considerando as limitações do escopo. Entre tais aplicações, pode-se citar o uso de sistemas analógicos para filtragem de sinais de áudio e controle de processos.

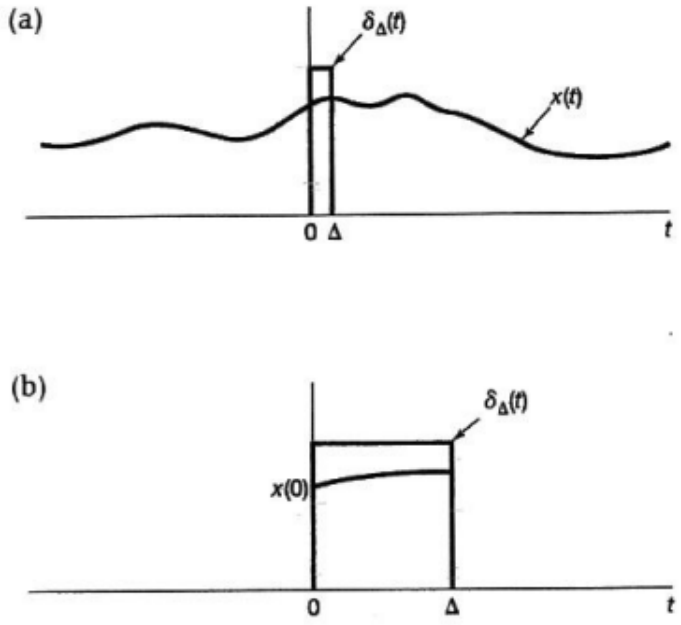
O princípio matemático da convolução implica que a saída de um sistema LIT é dada pelo somatório das sobreposições entre a entrada e a resposta ao impulso do sistema invertida, em regiões definidas por um deslocamento temporal onde ambos são não nulos. Entende-se a sobreposição como o produto entre os fatores. Se o sistema em questão pertencer ao domínio analógico, o somatório corresponde a uma integração no domínio do tempo. O símbolo algébrico de uma convolução entre dois sinais discretos no tempo é dado por Oppenheim e Willsky (2016) como:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]\, h[n-k] \tag{1}$$

Idealmente, um sistema cuja resposta ao impulso é um impulso unitário reproduz exatamente o sinal de entrada na saída, centrado no instante de tempo em que o impulso ocorre. Também chamada de Delta de Dirac ou amostra unitária, a função impulso ratifica esta constatação pela sua propriedade de peneiramento, ou amostragem. Multiplicá-lo com qualquer sinal resulta no valor do sinal apenas no instante onde a função é definida. A Figura 1 ilustra tal propriedade para uma função que aproxima a amostra unitária (Oppenheim e Willsky, 2016).

De posse dessas informações, é pertinente afirmar que um sistema LIT é plenamente definido pela sua resposta ao impulso unitário, e sua saída é obtida realizando a convolução entre a entrada e a resposta. Conhecer essas características de um sistema permite a implementação e elaboração de dispositivos que compensam certos comportamentos que levam à instabilidade, como controladores e filtros em sistemas de comunicação.

Figura 1: Propriedade da amostragem com uma aproximação da função Delta de Dirac.



Fonte: OPPENHEIM, Alan e V. WILLSKY, Alan S. Sinais e Sistemas.

2.2. Convolução Bidimensional

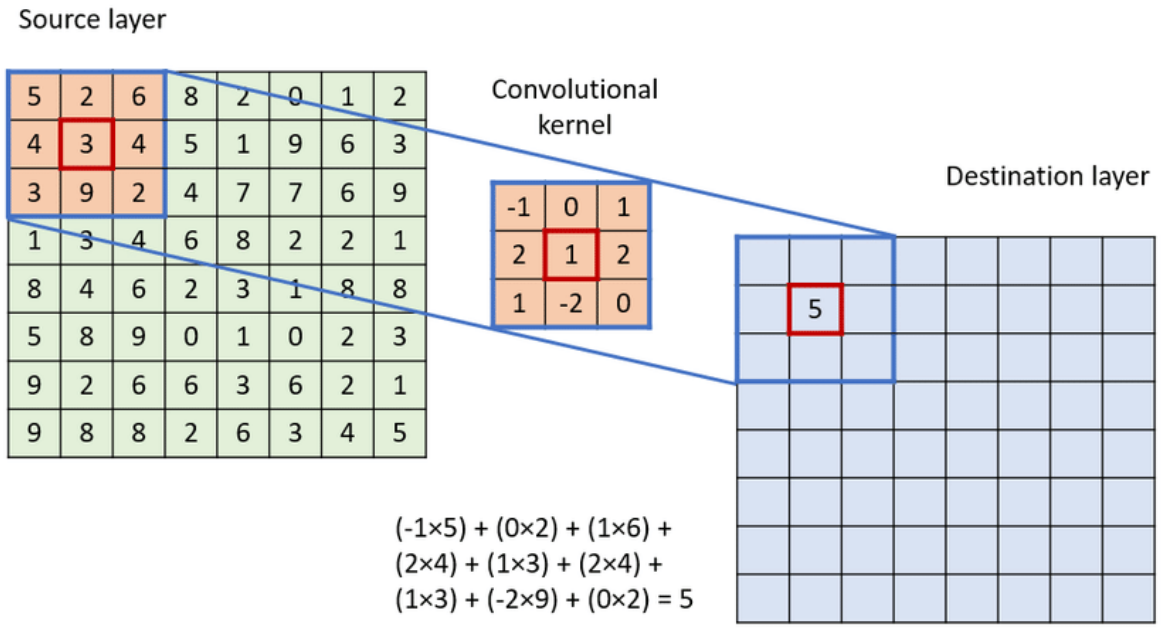
Em muitos fenômenos físicos e sistemas computacionais, o sinal recebido pode depender de mais de uma variável independente, e que podem ser interpretadas como as dimensões das funções do problema. As propriedades de sistemas LIT são mantidas independentemente da "profundidade" dos parâmetros dimensionais devido à sua linearidade, e por isso também é possível aplicar a convolução considerando que as operações algébricas são válidas para todas as variáveis no contexto.

Quando se fala em convolução unidimensional, a sobreposição de sinais equivale geometricamente ao valor da área da multiplicação dos sinais naquele intervalo, e se relaciona a um único instante do tempo da saída. Em outras palavras, o desenvolvimento é caracterizado por manipulações matemáticas entre grandezas de dimensões maiores (duas) que a grandeza final (uma). Em sistemas discretos, pode-se visualizar que cada valor do sinal de saída advém de um somatório entre dois vetores unidimensionais em que cada fator é igual à multiplicação entre os elementos de mesmo índice.

No caso bidimensional, a sobreposição se manifesta como o produto de regiões em um plano que quando são somadas - ou integradas - produzem uma saída igualmente bidimensional por meio de matrizes auxiliares de *features* (Shafkat, 2018). Como os fatores

são agora regiões, o valor numérico das multiplicações individuais pode ser interpretado geometricamente como o valor de um volume, e o conjunto desses produtos é normalmente chamado de tensor. A partir do tensor, são extraídos os volumes e atribuídos aos respectivos pontos do sinal de saída.

Figura 2: Exemplo de uma convolução bidimensional discreta.



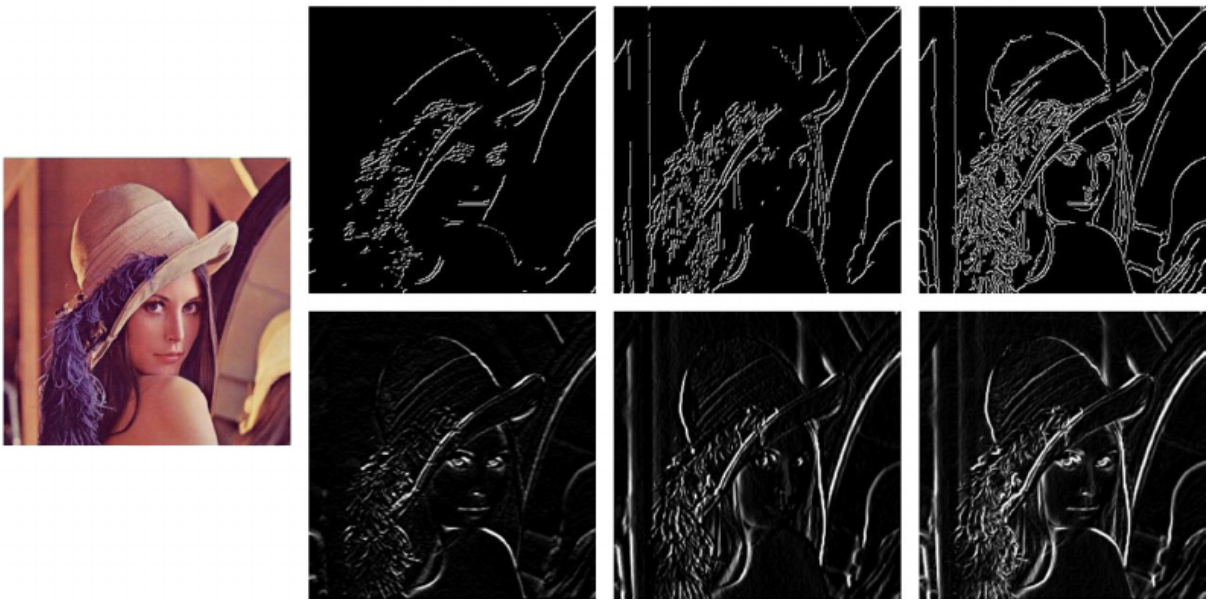
Fonte: Podareanu, Damian *et al*., 2019.

Uma das aplicações mais presentes da convolução na atualidade está no processamento digital de imagens, onde muitas vezes é desejado aplicar um filtro específico em uma foto, desenho, figura, etc. Diferente do que normalmente se imagina, um filtro ou kernel não é uma entidade bidimensional do mesmo tamanho da entrada, sendo na verdade muitas vezes bem menor em escala.

Para cobrir toda uma figura, o kernel invertido é inicialmente sobreposto em uma região de pixels da entrada com a quantidade de elementos igual ao primeiro, como mostrado na Figura 2 por Podareanu *et al.* (2019). Em seguida, desloca-se o kernel em uma das direções até atingir a extremidade da imagem, para assim repetir esse processo deslocando na outra direção. Se o deslocamento inicial se der no eixo horizontal, cada vez que se chega na extremidade terá sido processado uma linha da saída, e no caso para o eixo vertical, uma coluna. Esse processo é comumente nomeado como janela deslizante ou varredura da imagem.

Escolher apropriadamente o filtro aplicado no processo implica em uma versão da saída do sistema em relação à entrada. Os filtros de Sobel e Canny, por exemplo, são usados para detectar contornos em fotografias calculando diferenças finitas, fornecendo uma aproximação do gradiente da intensidade dos pixels da imagem. Mais especificamente, pode-se adequar esses tipos de kernels para identificar contornos ainda mais específicos, como horizontais, verticais e diagonais, como ilustrado por Machado *et al.* (2015) na Figura 3 com a popular imagem de teste de computação gráfica “Lenna”.

Figura 3: Aplicação de filtros Sobel (abaixo) e Canny (acima) na detecção de bordas.



Fonte: Machado *et al*. 2015.

2.3. Redes Neurais Convolucionais - CNN

Redes neurais são algoritmos computacionais capazes de simular o comportamento de neurônios e sinapses humanas. Essa abordagem possui uma construção diferente da elaboração tradicional de um programa de computador, que define um conjunto de regras e dados específicos para o processamento de ações ou resultados específicos. Em vez disso, são enviadas tanto entradas quanto as saídas desejadas para que assim o algoritmo elabore o conjunto de regras que venha convergir para aquele conjunto de dados.

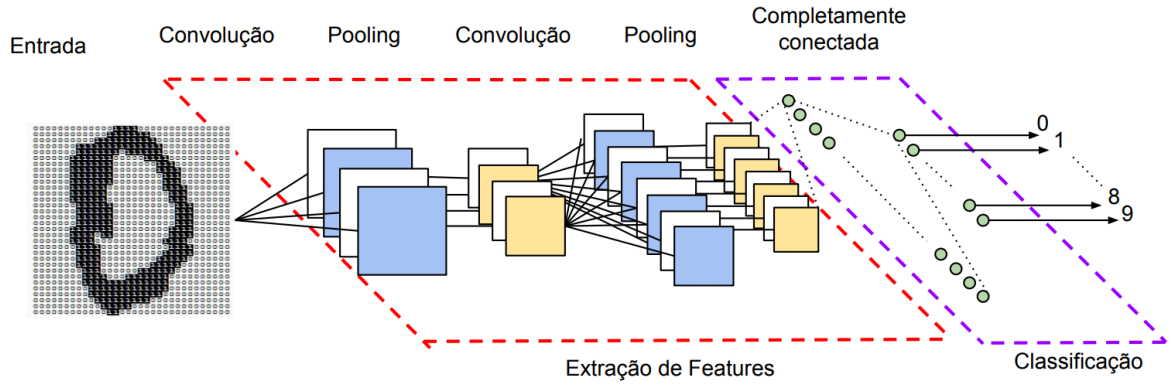
Apesar de parecer um trabalho de atividades complexas, mesmo com tantas aplicações atuais cada vez mais presentes e próximas, os fundamentos usados para confecção de tais redes obedecem a axiomas de álgebra linear e conceitos de lógica. Mitchel (1997) descreve esse paradigma da seguinte forma:

> Diz-se que um programa de computador aprende pela experiência E, em relação a algum tipo de tarefa T e alguma medida de desempenho P se, o seu desempenho P na execução da tarefa T, melhora com a experiência E.

2.3.1. Extração de características e tamanho da saída

A ferramenta matemática discutida na seção anterior oferece implicitamente um recurso extremamente útil para essas topologias em questão: a extração de características. O exemplo da detecção de bordas apresentado pode ser facilmente visto como uma maneira de evidenciar traços específicos como de rostos e formas geométricas. A CNN é capaz de aplicar filtros em dados visuais, mantendo a relação de vizinhança entre os pixels da imagem ao longo do processamento da rede (Vargas *et al.*, 2016). Em uma topologia de rede neural convolucional (CNN), a camada de convolução é usada com o intuito de criar um conjunto de dados reduzido para uma camada densa totalmente ou parcialmente conectada, ilustrado na Figura 4 por Vargas *et al.* (2016).

Figura 4: Diagrama de uma rede neural convolucional.



Fonte: Vargas *et al.*, 2016.

Dessa forma, outra peculiaridade da convolução bidimensional que se manifesta é o fato da saída ser menor que a entrada. Mais especificamente, o tamanho da imagem de saída é especificado por Zhang *et al.* (2021) como

$$(n_h - k_h + 1) \times (n - k_w + 1), \tag{2}$$

onde as grandezas escalares definidas pela letra "n" estão associadas a entrada, "k" ao kernel e "h" e "w" referenciam os eixos verticais e horizontais respectivamente (*height* e *width*). Vê-se que o aumento espacial do kernel implica numa diminuição na saída.

2.3.2. Correlação Cruzada

Anteriormente, foi dito que em uma convolução a resposta ao impulso ou filtro é invertida para poder realizar o deslocamento temporal e as subsequentes operações algébricas ou analíticas. Essa afirmação também é válida para a entrada do sistema em vez do filtro, não afetando o resultado final. Entretanto, no cenário de redes neurais, essa inversão da resposta ao impulso não ocorre, caracterizando uma correlação cruzada durante o processo de convolução em sua camada.

A correlação cruzada é, segundo Hinton (2002), uma função que fornece uma comparação estatística de duas sequências em função do deslocamento de tempo entre estas, usadas para detectar diferenças temporais. A abordagem matemática dessa operação é similar à da convolução, com a exceção da rotação de um dos sinais, e isso é notado na equação detalhada por Lathi (2004) como:

$$r_{xy}[k] = \sum_{k=-\infty}^{\infty} x[k]\, y[n + k]. \tag{3}$$

Com isso, percebe-se que uma convolução dita estrita nada mais é que uma correlação cruzada precedida de uma reflexão no eixo vertical, ou inversão temporal. É importante frisar a afirmação de Zhang *et al.* (2021) de que incorporar uma convolução estrita em uma CNN não interfere na sequência de dados resultante, uma vez que os kernels são "aprendidos" durante a fase de treinamento. A consideração que todavia deve ser tomada é de que o kernel aprendido também sofrerá uma rotação vertical e outra horizontal.

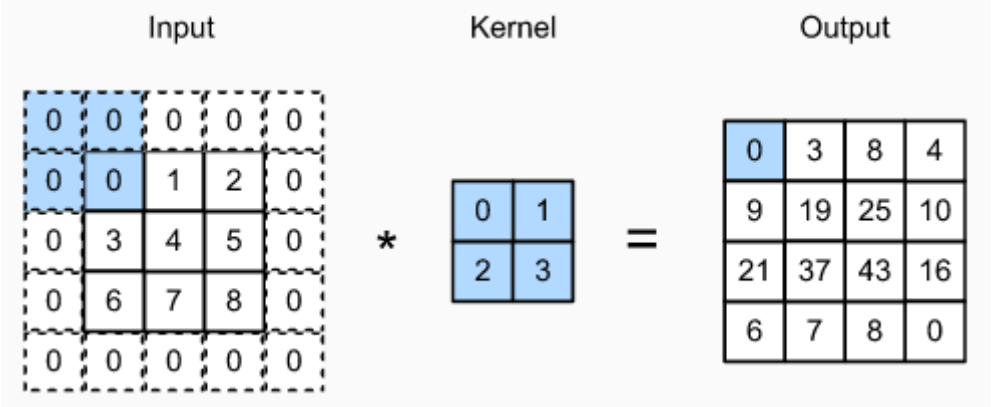
2.3.3. Padding e Striding

Outro fator que provoca a redução da saída é o passo do deslocamento da janela, ou *stride*, e existe uma variedade de situações que utilizam valores distintos. O deslocamento não precisa necessariamente ser tomado para janelas adjacentes como subentendido na seção anterior, levando em consideração que haverá uma quantidade inferior de elementos individuais atribuídos na construção da saída.

Não obstante, há cenários nos quais não é almejada a compressão do tamanho da imagem ao perder pixels do seu perímetro, e sim o seu tamanho original. Como exemplificado por Zhang *et al.* (2021), uma imagem 240x240 aplicada em uma rede com 10 camadas de convolução com filtros 5x5 revela uma saída 200x200, representando 70% da informação original.

A fim de evitar isso, a estratégia de preenchimento perimétrico de pixels ou *padding* é executada de modo que elementos também sejam adicionados no contorno da imagem convoluída. Geralmente o preenchimento é feito com valores nulos, mas pode variar de acordo com a abordagem, como por exemplo espelhar os valores dos elementos adjacentes. A Figura 5 exemplifica o *zero-padding* para uma imagem 3x3 e um filtro 2x2, e com esta se nota que os elementos centrais da saída permanecem inalterados.

Figura 5: Preenchimento com zeros de uma imagem 3x3.



Fonte: Zhang *et al.,* 2021.

Utilizando os recursos de preenchimento e aumento do salto, torna-se necessário reavaliar o tamanho da imagem final. A relação entre a área da entrada efetiva e a saída é diretamente proporcional, enquanto o valor do salto varia inversamente por diminuir a

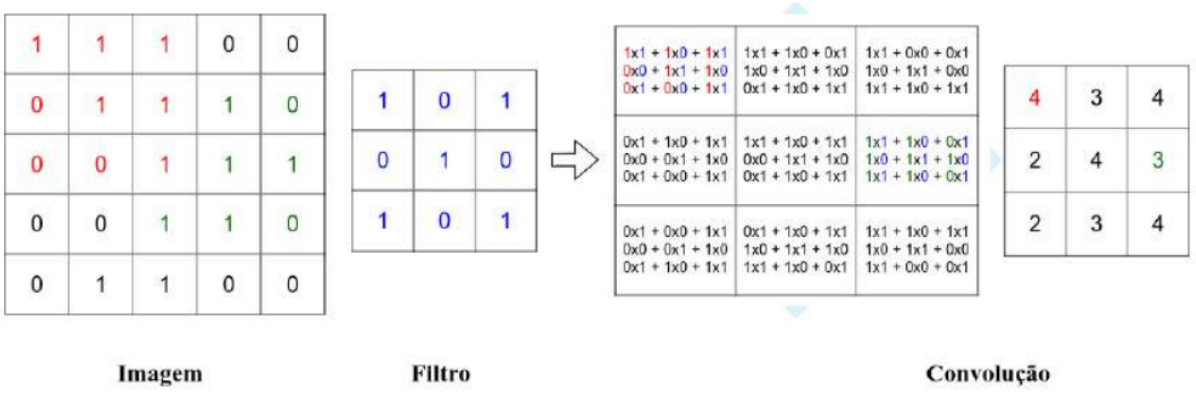
quantidade de janelas sobrepostas, ou elementos do tensor. Assim, a equação final considerando o padding e o stride para ambas as direções é para Zhang *et al.* (2021):

$$[(n_h - k_h + p_h + s_h)/s_h] \times [(n_w - k_w + p_w + s_w)/s_w]. \quad (4)$$

2.3.4. Exemplo demonstrativo

Para demonstrar o conhecimento exposto até então no presente trabalho, o exemplo de Ferreira (2017) é ilustrado na Figura 6, de modo que é possível observar cada operação de correlação cruzada sendo aplicado em cada janela de convolução, sem preenchimento em com passo unitário. O destaque de cores facilita a visualização dos elementos únicos da saída relacionados com as regiões da entrada.

Figura 6: Exemplo ilustrativo de uma convolução entre uma entrada 5x5 e um kernel 3x3.



Fonte: Ferreira, 2017.

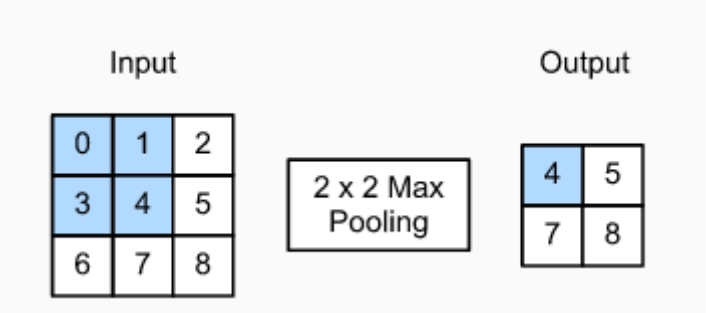
2.3.5. Pooling

A operação de agrupamento (*pooling*) é muitas vezes usada após uma camada de convolução e pode ser considerada como uma camada da rede por sua semelhança com o processo anterior a ela. Segundo Zhang *et al.* (2021), um operador de pooling consiste em uma janela fixa que percorre todas as regiões na entrada de acordo com seu passo, computando uma única saída para cada local percorrido pela janela fixa.

A diferença dessa camada para a camada de convolução é em como se estabelece a relação de vizinhança entre os pixels da região. Em contraste ao uso da correlação cruzada,

operadores de agrupamento determinam valores mínimos, médios ou máximos, nos quais o último é mais usado em redes convolucionais e foi proposto por Riesenhuber e Poggio (1999). Pode-se analisar um exemplo desenvolvido por Zhang *et al.* (2021) na Figura 7.

Figura 7: Exemplo de uma operação de agrupamento em uma imagem.



Fonte: Zhang *et al.,* 2021.

2.4. Métodos de Multiplicação

Existem diversos métodos para todas as operações matemáticas básicas, e são passados por gerações de centenas ou milhares de anos e que contam muito sobre a história do lugar e da sociedade de onde foi originado. Conforme O'Connor e Robertson (2011), há evidências de computação matemática que datam de cerca de 2000 anos antes de Cristo (AC).

Para multiplicação, um dos mais notórios é o método egípcio, que utiliza apenas uma sequência de adições e multiplicações por dois. West e Bellevue (2011) afirmam que esse método não requer memorização de todas as tabuadas de multiplicação, pois depende exclusivamente das habilidades das duas operações citadas.

O método egípcio consiste em multiplicar um dos fatores do produto por dois até atingir a maior potência de 2 menor que o segundo fator. Depois, fatora-se o segundo fator em potências de 2 para relacionar com os valores encontrados na primeira etapa correspondentes a cada potência encontrada. A Figura 8 exemplifica esse processo para a multiplicação entre 28 e 46. West e Bellevue (2011) ratificam que esse algoritmo funciona por causa da propriedade distributiva da multiplicação sobre a adição e a possibilidade de escrever um produto como a soma de potências de dois.

Figura 8: Método da multiplicação egípcia para 28 e 46.

$1 = 2^0$	$2^0 \times 46$	46
$2 = 2^1$	$2^1 \times 46$	92
$4 = 2^2$	$2^2 \times 46$	184
$8 = 2^3$	$2^3 \times 46$	368
$16 = 2^4$	$2^4 \times 46$	736
28		1288

Rewrite the problem as the sum of powers of 2:

$$28 \times 46$$
$$= (4 + 8 + 16) \times 46$$
$$= (4 \times 46) + (8 \times 46) + (16 \times 46)$$
$$= (2^2 \times 46) + (2^3 \times 46) + (2^4 \times 46)$$
$$= 184 + 368 + 736$$
$$= 1288$$

Fonte: WEST, Lynn e BELLEVUE, N. E., 2011.

Outro método de multiplicação que ficou muito popularizado é o denominado método do camponês russo, que ganhou fama na época antes da revolução russa pelo fato de muitos camponeses analfabetos conseguirem calcular mentalmente números suficientemente grandes (Tee, 2002). Esse método é considerado uma extensão do método egípcio, pois usa as mesmas propriedades fazendo operações com os dois fatores do produto.

Nesse método, enquanto um dos fatores é multiplicado por dois, o outro é dividido por 2 considerando apenas a parte inteira da divisão, repetindo esse processo até atingir o valor unitário na divisão. A partir daí, anulam-se os valores correspondentes em que a divisão resultou em valores pares e somam-se os valores restantes resultantes do fator multiplicado. A Figura 9 exemplifica o método para os mesmos valores usados anteriormente.

Figura 9: Método da multiplicação do camponês russo, para 28 e 46.

~~46~~	~~28~~
23	56
11	112
5	224
~~2~~	~~448~~
1	896
	1288

$$56 + 112 + 224 + 896 = 1288$$
$$\text{Therefore, } 46 \times 28 = 1288$$

Fonte: WEST, Lynn e BELLEVUE, N. E., 2011.

Ambos os métodos mostrados têm sua relevância por requerer operações simples sequenciadas, e West e Bellevue (2011) afirmam que usado como uma forma de andaime para estudantes que ainda não possuem domínio matemático. Não obstante, outro destaque desses métodos é a possibilidade de ser aplicado em sistemas computacionais, os quais são fundamentados em álgebra binária.

2.5. Quantização

Na confecção de redes neurais, os dados numéricos normalmente pertencem a intervalos fracionários, e quanto mais casas decimais mais precisa é a informação. Entretanto, esse aumento de precisão pode não melhorar a eficiência da rede de maneira considerável. A exemplo, redes neurais convolucionais em estado da arte não são adequadas para uso em dispositivos móveis (Jacob *et al.*, 2018).

Com o intuito de adaptar redes elaboradas com representação em ponto flutuante, pode-se quantizar a topologia para usar inteiros de oito bits tanto na fase de treinamento quanto na fase de teste ou em ambas. Essa abordagem facilita a criação de modelos que apresentam eficiências suficientemente satisfatórias, como foi feito por Chen *et al.* (2015) e Gong *et al.* (2014), mas tais trabalhos focam na melhora do armazenamento no dispositivo e menos com eficiência computacional, não sendo exatamente viáveis para implantações em hardware (Jacob *et al.*, 2018).

O esquema de quantização proposto por Jacob *et al*. (2018) mantém um alto grau de correspondência entre a aritmética de inteiros durante a inferência e a de ponto flutuante durante o treinamento. A base matemática do esquema gera um mapeamento de afinidade entre as representações pela equação:

$$r = S(q - Z), \tag{5}$$

onde "r" é a denotação do número real, "q" é o número quantizado e "S" e "Z" são a escala e o ponto zero (*zero point*), respectivamente. Aplicando essa abordagem na multiplicação matricial da equação de inferência, tem-se para um único neurônio:

$$S_y(y_q^{i,j} - Z_{py}) = \sum_k (S_w S_x(w_q^{i,j} - Z_{pw})(x_q^{i,j} - Z_{px})) + S_b(b_q^{i,j} - Z_{pb}), \tag{6}$$

onde "x" é a entrada da camada, "w" é a matriz dos pesos e "b" é o vetor do viés. O mesmo desenvolvimento é válido para camadas de convolução devido o uso da mesma operação linear, considerando que os pesos e as entradas são substituídos pelas janelas da imagem e pelo kernel respectivamente. Definido por Jacob *et al*. (2018), a escala do bias é definida como $S_b = S_w S_x$ e fazendo $Z_b = Z_w = 0$, pode-se simplificar a equação matricial para:

$$y_q^{i,j} = M\sum_k (w_q^{i,j} * x_q^{i,j}) - C, \quad (7)$$

onde:

$$C = M\sum_k (w_q^{i,j} * z_{px} + b_q^{i,j}) + Z_{py}, \quad (8)$$

$$M = M_0 * 2^{-n}. \quad (9)$$

Esta equação mostra que a saída de uma camada da rede é obtida a partir da operação de correlação cruzada adicionada de constantes proporcionais a dados de treinamento e de quantização. Assim, é possível mensurar as constantes M_0, C e N durante a fase de treinamento e enviar para um hardware apenas os valores da equação da saída da camada.

3. METODOLOGIA

3.1. XNPU - Unidade de Processamento Neural Extensível

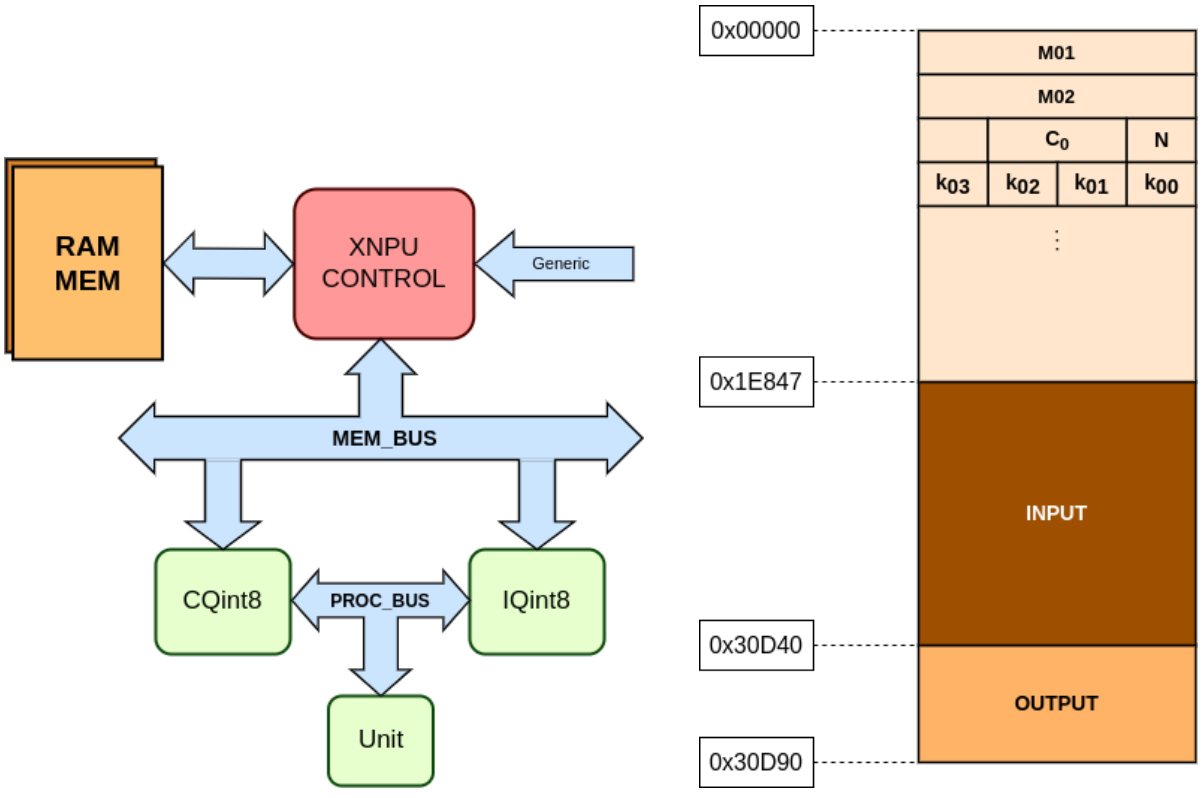
O XNPU é um projeto de um *chip* digital com uma arquitetura de processamento de dados apropriada para redes de aprendizado profundo (Cruz *et al.,* 2022) na fase de pós-treinamento em alto nível. O sistema propõe uma implementação completamente quantizada em inteiros de 8 bits, podendo efetuar tal processo no pós-treino a nível de hardware ou recebendo uma rede previamente treinada dessa forma. Atualmente o projeto engloba redes neurais profundas (DNN) e convolucionais (CNN).

A ideia principal do projeto é construir um sistema digital de processar uma pluralidade de topologias, baseadas apenas nos dados espaciais e quantitativos de uma rede. A ideia principal do projeto é construir um sistema digital de processar uma pluralidade de topologias, baseadas apenas nos dados espaciais e quantitativos de uma rede. É possível adquirir esses valores com dados do treinamento de uma rede a nível de software, em frameworks como o TensorFlow junto com o programa Netron, pela vantagem do uso difundido de tais ferramentas para soluções de problemas da mesma área.

As variáveis da rede são guardadas em uma memória e dispostas por camada, de acordo com sua necessidade de acesso. As constantes de quantização podem ser únicas por camada ou variar para cada neurônio, dependendo se o processamento da rede estiver na etapa da convolução ou da inferência. Assim, a escolha dos espaços da memória fica padronizada para qualquer tipo de camada.

A macroarquitetura da unidade de processamento é exposta no diagrama de blocos da Figura 10.a. Os blocos com sufixo int8 são os núcleos de processamento da rede, e se comunicam com a Unit através de um barramento OR de caminho de dados (PROC_BUS). O controlador de memória é responsável por todo o controle do chip, ativando os núcleos de acordo com o sequenciamento das camadas, armazenando dados de status e características da rede, mediando os acessos à memória e se comunicando com módulos de comunicação externa, como receptor/transmissor assíncrono universal (UART), barramento universal serial (USB), entradas e saídas de uso genérico (GPIO), etc. O bloco Unit é o responsável pelas operações algébricas comuns entre a convolução e a inferência, assim como a função de ativação.

Figura 10: (a) Arquitetura de topo do XNPU e (b) mapa de memória.



(a) (b)

A versão atual do projeto define o uso de uma memória de 1MB, oferecendo 250 mil posições de armazenamento de palavras de 32b. Essa divisão mostrada na Figura 10.b permite estabelecer a região da primeira metade dos endereços (0x00000 à 0x1E847) para o modelo, que no caso da convolução são as constantes de quantização e os kernels. Do endereço 0x1E848 ao 0x30D3F tem-se a região da entrada da camada e sobra para a sua saída a região de 0x30D40 a 0x3D090.

O funcionamento do chip é iniciado com um processo de *boot*, para que assim possa esperar a escrita dos dados do modelo e da entrada da primeira camada. Identificando a rede, escolhe-se o processo de convolução ou de inferência da camada, checa e armazena os resultados para processar a próxima camada, na qual a entrada desta será a saída da anterior.

3.2. Linearização de Dados

Informação digital é representada como uma sequência de bits, traduzida em um símbolo, como letras e números. Quando uma sequência (vetor) de símbolos compõem uma informação por si só, os bits são ligados por suas extremidades de modo que a ordem dos dados seja respeitada. Linguagens de descrição de hardware como Verilog e VHDL permitem gerar esses vetores a partir do número de bits dos elementos e a quantidade de desses (*packed*), mas ao sintetizar o sistema, os componentes físicos sempre resultarão em sequências unidimensionais (*unpacked*).

Essa noção é ainda mais relevante quando um sistema digital trata dados com mais de uma dimensão, como matrizes e tensores. Para poder armazenar e interpretar elementos bidimensionais, o dado precisa ser armazenado como uma sequência de vetores unidimensionais. Esse processo é chamado de linearização, e é usado por exemplo em arquivos binários de fotografias em que os pixels (bytes) são postos um ao lado do outro de uma diagonal da imagem até a diagonal oposta.

3.3. Arquitetura da Solução

O módulo CQint8 deve realizar a operação linear para inteiros de 8 bits se adaptando às dimensões da imagem e do kernel a nível de processamento, por meio do acesso à memória por meio do barramento. A Figura 11 mostra a visão de topo de bloco em questão, evidenciando as interfaces com o barramento e o módulo Unit, responsável pela correlação

cruzada. A Tabela 1 descreve os sinais da interface com o barramento, enquanto a Tabela 2 descreve a interface do bloco com a Unit.

Ao iniciar o processo, o bloco efetua o cálculo das dimensões da entrada e do kernel da camada para em seguida ler os dados das constantes de quantização e o filtro, os quais são guardados no início da memória até o fim do kernel, logo antes da posição do próximo, se houver. Após essa etapa, o módulo começa o janelamento alterando o endereço para o dado na primeira posição da região da entrada e variando de acordo com uma lógica de acesso às regiões da imagem. Nesse ciclo, quando o módulo identifica a presença de uma janela válida, o processo de leitura é interrompido para se comunicar com a Unit, espera o resultado dessa pelos sinais de *handshake* e armazena em um registrador interno.

Figura 11: Topo do bloco de convolução CQint8 e suas interfaces.

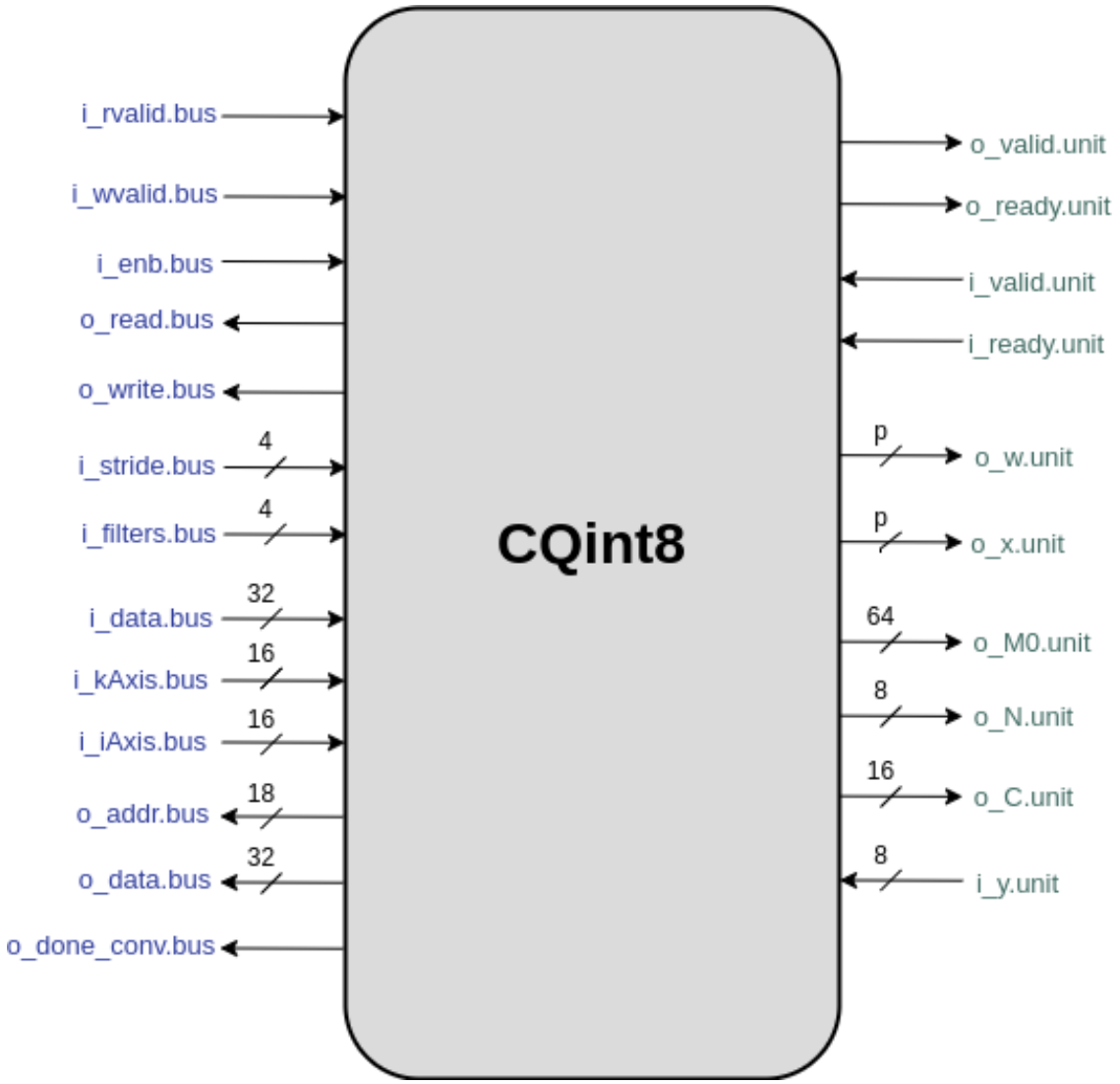


Tabela 1: Sinais da interface do módulo CQint8 com o barramento.

Nome do sinal	Tipo (I - entrada, O - saída)	Quantidade de bits	Descrição
i_rvalid	I	1	*Handshake* da memória indicando leitura realizada.
i_wvalid	I	1	*Handshake* da memória indicando escrita realizada.
i_enb	I	1	*Enable* para iniciar o processamento do bloco.
i_stride	I	4	Passo da convolução.
i_filters	I	4	Quantidade de filtros na camada corrente.
i_data	I	32	Dado lido da memória.
i_kAxis	I	16	Dimensões do(s) kernel(s) da camada.
i_iAxis	I	16	Dimensões da entrada da camada.
o_read	O	1	Solicitação de leitura na memória.
o_write	O	1	Solicitação de escrita na memória.
o_addr	O	18	Endereço para leitura ou escrita na memória via barramento.
o_data	O	32	Dado a ser escrito na memória.

Tabela 2: Sinais da interface do módulo CQint8 com a Unit.

Nome do sinal	Tipo (I - entrada, O - saída)	Quantidade de bits	Descrição
i_valid	I	1	*Handshake* da Unit indicando sinal válido do processamento.
i_ready	I	1	*Handshake* da Unit indicando que o módulo está pronto para receber.
i_y	I	8	Resultado da Unit.

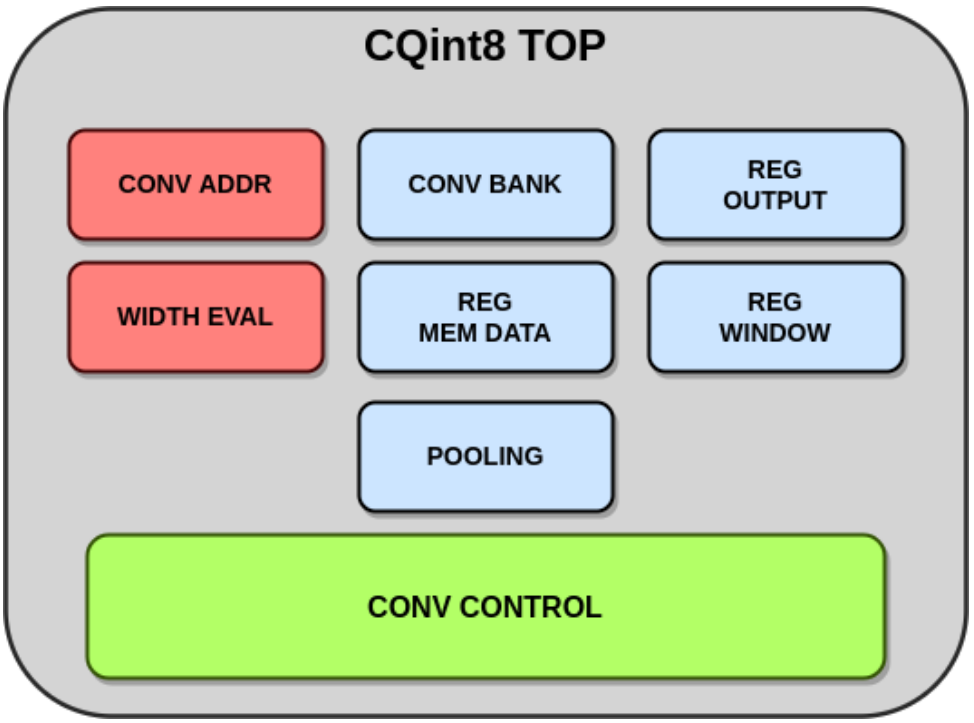
o_valid	I	1	*Handshake* de solicitação de ativação da Unit.
o_ready	O	1	*Handshake* da solicitação indicando que o módulo está pronto para receber o dado da Unit.
o_x	O	A depender	Vetor unidimensional *packed* com dados da entrada.
o_w	O	A depender	Vetor unidimensional *packed* com dados do kernel.
o_M0	O	64	Constante de quantização.
o_C	O	64	Constante de quantização.
o_N	O	64	Constante de quantização.

3.3.1. Macroarquitetura

Os submódulos que compõem a macroarquitetura do bloco são dispostos no diagrama da Figura 12 omitindo as conexões para evitar poluição visual. Cada módulo tem sua relevância em instantes específicos do processamento dos dados, sendo ativado apenas quando necessário ou selecionando um entre os comportamentos possíveis.

Os blocos em vermelho desenvolvem a lógica de endereçamento no decorrer da varredura, em azul tem-se o armazenamento de dados - constantes, kernel e entrada - e o controlador é realçado em tom de verde. Cada submódulo tem conexões específicas, mas todos se comunicam com o controle. O sub-módulo de pooling tem uma comunicação semelhante com o bloco externo Unit, porém como essa operação é majoritariamente precedida de camadas de convolução, escolheu-se inserí-lo dentro do módulo para compor uma ferramenta mais completa.

Figura 12: Macroarquitetura do CQint8.



3.3.2. Submódulo de Cálculo de Comprimentos

Como dito anteriormente, dados bidimensionais como as entradas, pesos e filtros, são guardados vetorizados em linha, da extremidade superior esquerda para a extremidade inferior direita. Com esse fato, torna-se necessário mensurar os dados na memória como vetores unidimensionais multiplicando as dimensões vindas do barramento. Esses dados são adquiridos a partir de arquivos contendo o treinamento de uma rede a nível de software, como citado em seções anteriores.

O comprimento do kernel é computado por meio do método da multiplicação do camponês (*peasant*), a fim de determinar o último endereço de palavra que o módulo deve ler para completar o vetor e enviá-lo para o submódulo de endereços. Se na camada houver mais de um filtro, o endereço logo após o calculado inicialmente é salvo visto que é onde se inicia os dados daquele. Como tais informações são únicas por camada, a região da memória para guardá-los é sempre a mesma.

3.3.3. Submódulo de Endereçamento

Este sub-bloco é acionado assim que os comprimentos dos vetores são obtidos. O endereçamento é feito de dois modos, conforme a etapa: no primeiro, o endereço varia

sequencialmente com passo unitário, apenas para carregar os dados da camada, enquanto o segundo varia segundo a necessidade do janelamento da entrada e por conseguinte, exige uma lógica mais complexa.

A lógica da etapa de janelamento considera um deslocamento de bytes e faz uso de três contadores com suas tabelas verdades dispostas nas Tabelas 3, 4 e 5, representando dimensões relevantes do kernel e da entrada usadas para disparar sinais para o controle e para os registradores. A Figura 13 demonstra o fluxo de dados obtidos para preenchimento do vetor que armazena cada janela, considerando uma imagem 5x5 e um filtro 3x3.

Como é possível armazenar quatro elementos em uma palavra, os bytes para o janelamento correto podem ficar estratificados na memória e pode ser necessário carregar uma palavra que começa com um elemento que não será considerado. Assim, o endereço efetivo para a memória desconsidera os dois bits menos significativos do endereço da convolução e é dividido por quatro, uma vez que, novamente, o endereço de convolução considera um deslocamento de bytes.

Em suma, o endereçamento e o deslocamento carregam x_k bytes de uma linha da entrada iniciando na primeira coluna, até que y_i linhas sejam lidas. Então, esse processo se repete para as próximas colunas da imagem e gerando a saída a partir de suas colunas. As Tabelas 6 e 7 mostram como os dois endereços discutidos se comportam.

Tabela 3: Tabela verdade do contador de colunas do kernel.

addr_conv != addr_mem	cnt_KC = x_k - 1	PRÓXIMO VALOR
0	X	cnt_KC
1	0	cnt_KC + 1
1	1	0

Tabela 4: Tabela verdade do contador de colunas da saída.

addr_conv != addr_mem	cnt_IR = y_i - 1	PRÓXIMO VALOR
0	X	cnt_OC
1	0	cnt_OC
1	1	cnt_OC + 1

Tabela 5: Tabela verdade do contador de linhas da entrada.

addr_conv != addr_mem	cnt_KC = x_k - 1	cnt_IR = y_i - 1	PRÓXIMO VALOR
0	X	X	cnt_IR
1	0	X	cnt_IR
1	1	0	cnt_IR + 1
1	1	1	0

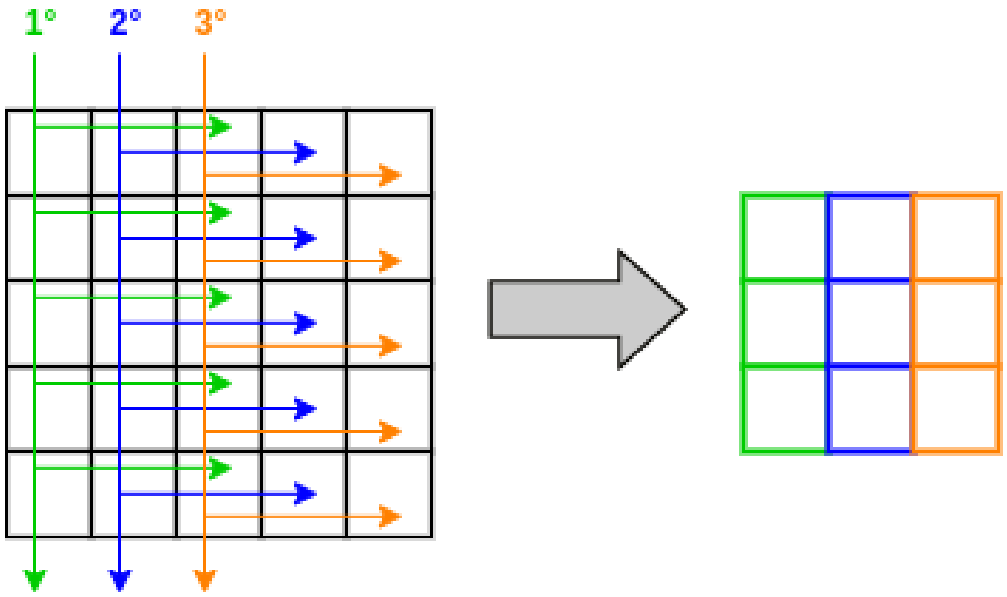
Tabela 6: Tabela verdade do endereço de memória (addr_mem), mapeado por palavra.

enable do endereço	cnt_KC = x_k - 1	PRÓXIMO VALOR
0	X	addr_mem
1	0	addr_mem + 1
1	1	addr_conv[15:3]

Tabela 7: Tabela verdade do endereço de convolução (addr_conv), mapeado por byte.

enable do endereço	addr_conv != addr_mem	cnt_KC = x_k - 1	cnt_IR = y_i - 1	PRÓXIMO VALOR
0	0	X	X	addr_conv
1	1	0	X	addr_conv
1	1	1	0	addr_conv + 1
1	1	1	0	addr_conv + x_ i - (x_k - 1)
1	1	1	1	cnt_OC

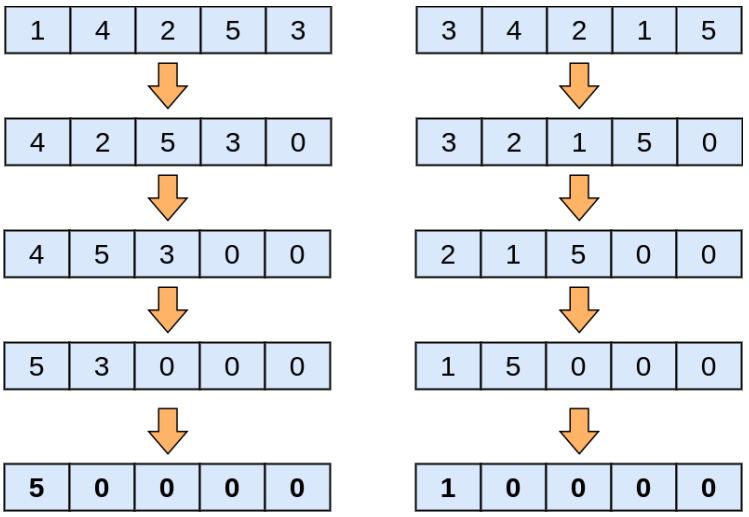
Figura 13: Diagrama de preenchimento de bytes na etapa do janelamento.



3.3.4. Submódulo de Pooling

Uma vez que a operação do pooling é análoga à realizar a convolução com a ausência de um filtro, foi feito um submódulo com esse intuito que pode ser acionado ao invés da Unit. Para que isso ocorra, é necessário identificar que a camada é de pooling, o que é facilmente resolvido atribuindo um valor nulo no sinal de quantidade de filtros i_filters que é enviada pelo controlador via barramento.

Figura 14: Diagrama da lógica de (a) Max Pooling e (b) Min Pooling.



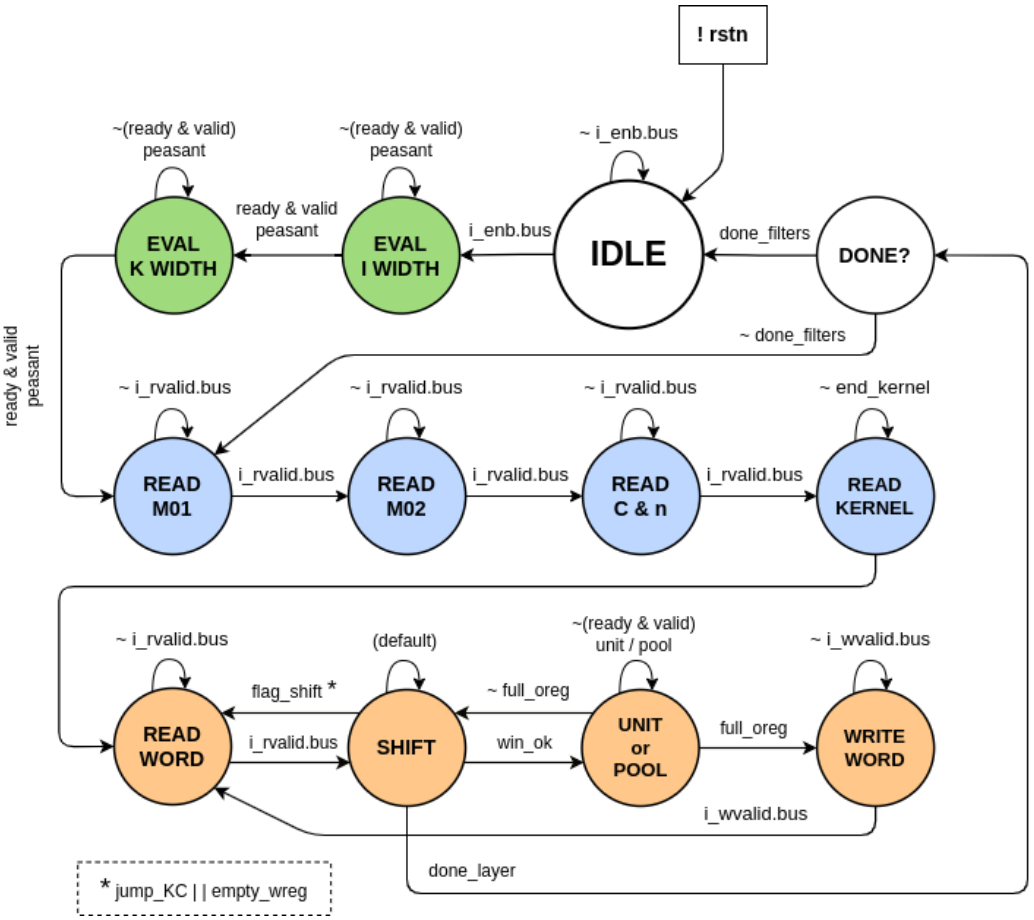
(a) (b)

O módulo é capaz de realizar o Max Pooling e o Min Pooling para os vetores de bytes que caracterizam cada janela da camada. Ao ser ativado, o bloco armazena o dado em um registrador e desloca todos os bytes da direita para a esquerda, sendo que o byte mais à esquerda possui a lógica de comparação “maior que” ou “menor que” com byte adjacente. A Figura 14 exemplifica esse procedimento para uma janela arbitrária de tamanho 5.

3.3.5. Submódulo de Controle

O controle do bloco recebe informações pontuais do módulo como um todo, identificando características do processamento do sistema e adapta os estados futuros para o sequenciamento exato de todas as execuções. Ao resetar o bloco, o controle mantém o circuito em estado ocioso (*idle*) esperando pelo *enable* do topo e, quando este é detectado, realiza a ordem estabelecida pelo diagrama da Figura 15. É possível separar o processamento em três grandes ciclos: o cálculo dos comprimentos; o carregamento de constantes de quantização e kernel; e o janelamento com auxílio da Unit.

Figura 15: Diagrama de transição de estados do controle do CQint8.



Cada estado também define os *enables* dos registradores do bloco no devido momento. Por exemplo, os registros das constantes são ativados na ordem do diagrama de estados durante sua respectiva etapa, assim como os de leitura e do janelamento. Consequentemente, também há sinais de ativação para a lógica do endereço.

3.3.6. Banco de registradores para Constantes e Kernel

A segunda etapa do CQint8 é a primeira etapa do janelamento, no qual o endereço varia sequencialmente e fornece os dados de constantes e do kernel. Os registradores do M0, do N e do C possuem 64, 8 e 16 bits respectivamente, definidos a nível de topo do XNPU levando em conta operações algébricas da quantização. O tamanho do vetor de registros do kernel também é definido no topo arbitrariamente, uma vez que um preenchimento com zero em bytes do vetor não afeta o resultado da correlação cruzada. Assim, pode-se definir o vetor com o número de elementos igual à maior camada da rede, lembrando de que a Unit deve conter esse mesmo número nas suas entradas correspondentes.

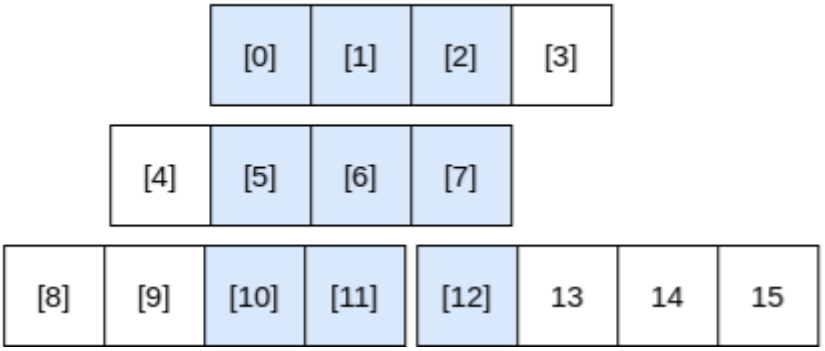
3.3.7. Registradores de Leitura e Escrita

As palavras lidas para carregar as janelas da entrada precisam de dois registradores com deslocamentos especiais: o registrador da palavra da memória e o vetor de bytes da janela. O primeiro é capaz de receber 32 bits do barramento ou deslocar seus bytes para o registro da janela. O segundo recebe em uma das suas extremidades os bytes vindo do primeiro, enquanto se desloca em 8 bits na mesma direção.

O registrador da palavra pode conter menos elementos necessários para completar uma linha da janela, assim como pode conter menos por causa da estratificação dos dados linearizados na memória. Um registrador de deslocamento de 4 bits indica quando a palavra foi enviada por inteiro, enquanto a flag do contador da linha da janela (jump_KC) indica se não precisa de mais bytes da palavra, podendo ler outra posição da memória. A Figura 16 mostra um exemplo de estratificação da primeira janela 3x3 de uma imagem 5x5.

Outro registrador é implementado para os dados da saída, recebendo cada byte da Unit quando o *handshake* acontece. Como deve-se armazenar palavras na memória, o registrador avisa quando está completamente preenchido para que possa solicitar uma escrita, limpar os dados e receber os próximos dados. O deslocamento é de 8 bits como nos casos já descritos.

Figura 16: Estratificação do acesso de bytes na memória.



4. RESULTADOS E DISCUSSÕES

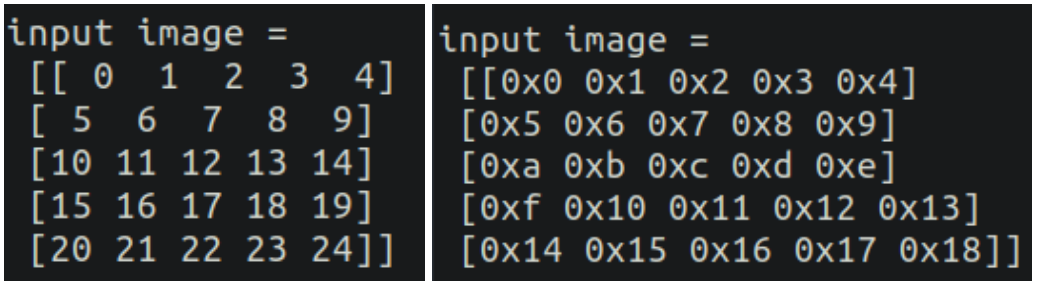
Para fins demonstrativos, foi testada a funcionalidade do bloco para a operação de convolução propriamente dita, sem preenchimento com zero e com passo único. O teste foi realizado para uma entrada 5x5 com valores sequenciais de 0 a 24 exposto na Figura 17.a (0 a 18 em hexadecimal, como consta a Figura 17.b) e também para o exemplo de Ferreira (2017) ilustrado na Figura 6. O modelo em alto nível foi testado em uma máquina local com o sistema Ubuntu Linux, enquanto foi usado o simulador Icarus Verilog junto com GT Waves para o RTL.

Como os valores estão dentro do alcance da representação em 8 bits, estes podem ser considerados como quantizados, e as constantes de quantização foram concedidas como C = 0, n = 0 e M_0 = 1. O filtro utilizado foi o mesmo do exemplo da Figura 6, que evidencia elementos das diagonais da janela. Verifica-se por meio da Figura 18 a saída esperada da convolução bidimensional, também com os valores decimais (18.a) e hexadecimais (18.b).

Fica evidente pelas Figuras 18 e 19 que os resultados obtidos tanto no modelo em alto nível quanto na implementação em descrição de *hardware* são equivalentes, o que confirma a validade da segunda. Isso é possível porque a linearidade dos sistemas tratados é mantida em todo o processo, e corrobora com a teoria exposta neste trabalho. O registrador que guarda a saída completa foi criado com intuito auxiliar de visualizar o resultado final, não sendo parte da implementação desejada.

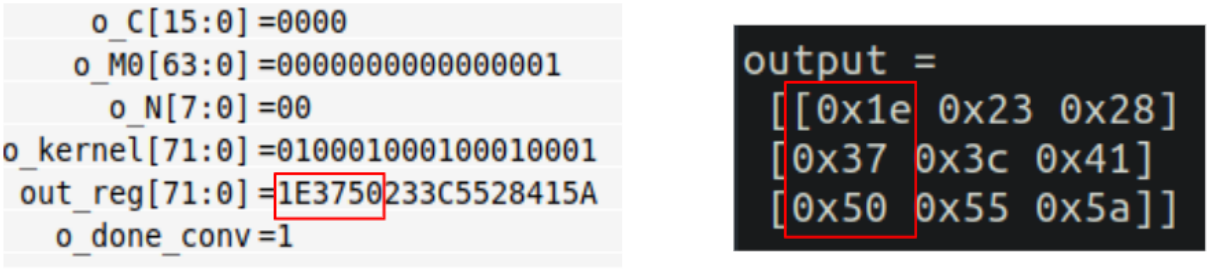
Também foi feito um ensaio para testar a funcionalidade do módulo de pooling, configurado para a função de máximo e usando a entrada sequencial. Como visto na Figura 20, a saída apresenta os bytes da janela inferior direita da entrada, o que é esperado.

Figura 17: Entrada do modelo para o teste sequencial (a) em decimal; (b) em hexadecimal.



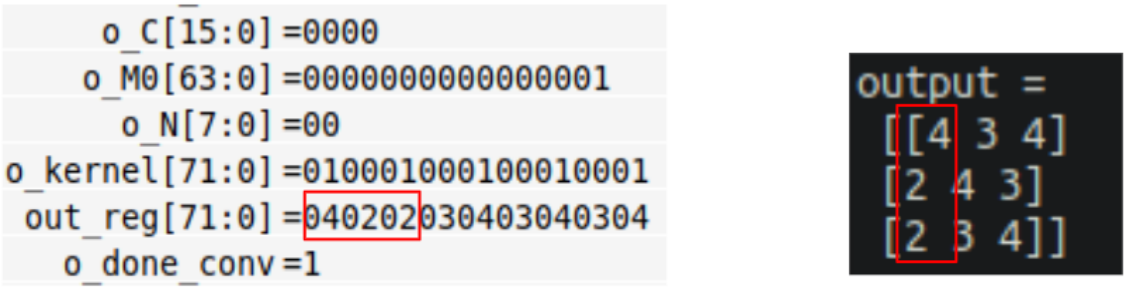
(a) (b)

Figura 18: Resultados para o sequenciamento (a) do CQint8; (b) do modelo em Python.



(a) (b)

Figura 19: Resultados para o exemplo de Ferreira (a) do CQint8; (b) do modelo em Python.



(a) (b)

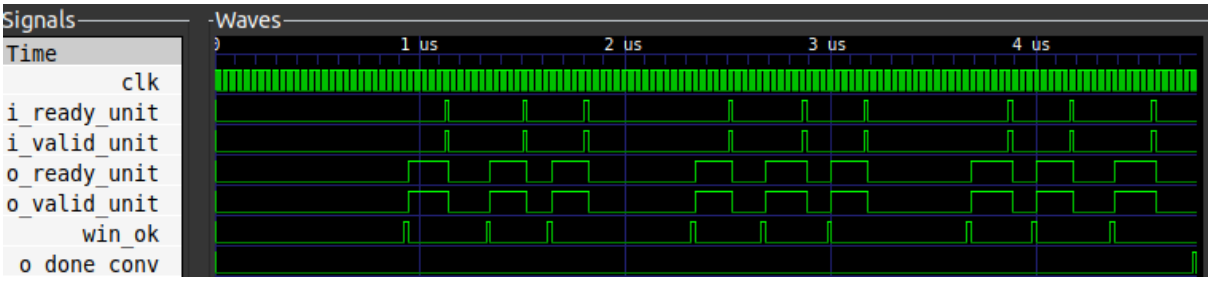
Figura 20: Resultados para o Max Pooling (a) do CQint8; (b) do modelo em Python.

(a) (b)

A congruência da resolução dos problemas é válida por causa do princípio de invariância de translação da convolução, o qual impõe que a janela da imagem e o filtro não pode depender das posições para obter a representação oculta de uma camada (Zhang *et al.*, 2021). Outro princípio anunciado por Zhang *et al.* (2021) e que é respeitado é o da localidade, que afirma que a sobreposição entre kernel e imagem deve estar dentro do domínio da imagem, e qualquer sobreposição fora desse domínio tem valor nulo.

A Figura 21 expõe as formas de onda para os sinais de *handshake* com a Unit. Pode-se observar que há nove pulsos de cada de ready e valid, o que é esperado pelo tamanho da saída. Após a quantidade descrita de pulsos, o módulo identifica a finalização da operação e ativa o sinal de término. Mesmo que as formas de onda apresentadas sejam de testes com a Unit, os sinais se comportam de forma idêntica com o uso do submódulo de pooling, considerando o sufixo para cada caso.

Figura 21: Formas de onda para os testes com Unit, no GTK Waves.



5. CONCLUSÃO

O presente trabalho mostra que é possível realizar uma operação de convolução a nível de processamento computacional, e que pode ser usado com muita vantagem em uma unidade de processamento neural com o objetivo de acelerar redes treinadas. Isso possibilita não só um maior desempenho comparado com implementações em alto nível, mas também demonstra que a área ocupada pelo bloco ou até pelo circuito integrado que o compõe pode atingir valores muito pequenos e também consumir pouca energia elétrica.

A versão atual do módulo ainda não possui todas as funcionalidades descritas no texto, como preenchimento com zero e passo maior que um. Ainda assim, foi demonstrado que ele realiza com facilidade e clareza a operação de convolução pura para dados linearizados em uma memória com dados maiores que os dados efetivos no processamento.

Torna-se importante frisar o fato de que a escolha de mapeamento de bytes por colunas da matriz de saída pode ter um resultado comumente interpretado como a transposição do real valor esperado, e de fato, torna-se um fator de influência no projeto. Assim como a rotina apresentada, é possível aplicar com facilidade o mesmo raciocínio para criar uma lógica de preenchimento de dados por linhas da imagem. Se necessário, atualizações do módulo substituirão essa rotina, ratificando a plausibilidade de afirmar que o fluxo de dados é suficiente para a completude do processo.

Também é notório o uso de algoritmos algébricos como a multiplicação do camponês russo, que além de oferecer uma opção fácil de aplicar manualmente, é ainda mais útil em sistemas computacionais que trabalham com aritmética binária. Todavia, apesar de estar apenas em uma versão limitada, o bloco já é capaz de receber uma boa variedade de topologias. Outra vantagem arquitetural do módulo é de não necessitar realizar divisões algébricas, como para o cálculo mostrada para o tamanho da saída.

Como o projeto do XNPU ainda está em andamento, futuras versões do módulo CQint8 serão capazes de fazer todas funcionalidades propostas, assim como será testado junto com os demais módulos do acelerador quando o projeto estiver mais desenvolvido. Mais futuramente, serão avaliadas as possibilidades de paralelizar algumas aplicações tanto do módulo quanto do chip.

REFERÊNCIAS BIBLIOGRÁFICAS

- CHEN, Jim X. The evolution of computing: AlphaGo. **Computing in Science & Engineering**, v. 18, n. 4, p. 4-7, 2016.
- CHEN, W., WILSON, J. T., TYREE S., WEINBERGER, K. Q., e Chen, Y. **Compressing neural networks with the hashing trick**. In: **International conference on machine learning**. PMLR, 2015. p. 2285-2294.
- CRUZ, T. S., GOMES, J. P., SUASSUNA, H. G., ALBUQUERQUE, D. W., DOS SANTOS JÚNIOR, G. G., PERKUSICH, A., & SANTOS, D. F. **XNPU-eXtensible Neural Processing Unity From a Programmable Logic Device**. In: **2022 IEEE International Conference on Consumer Electronics (ICCE)**. IEEE, 2022. p. 1-2.
- FERREIRA, Alessandro dos Santos. **Redes neurais convolucionais profundas na detecção de plantas daninhas em lavouras de soja**. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Computação - Universidade Federal do Mato Grosso do Sul, Campo Grande, 2017.
- GONG, Y., LIU, L., YANG, M., BOURDEV, L. Compressing deep convolutional networks using vector quantization. **arXiv preprint arXiv**:1412.6115, 2014.
- JACOB, B., KLIGYS, S., CHEN, B., ZHU, M., TANG, M., HOWARD, A., HARTWIG, A., KALENICHEKO, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. 2018. p. 2704-2713.
- MACHADO, P., ROMERO, J., NADAL, M., SANTOS, A., CORREIA, J., CARBALLAL, A. Computerized measures of visual complexity. **Acta psychologica**, v. 160, p. 43-57, 2015.
- MITCHELL, Tom M., and Tom M. Mitchell. **Machine learning**. Vol. 1. No. 9. New York: McGraw-hill, 1997.
- PODAREANU, D., CODREANU, V., SANDRA AIGNER, T. U. M., VAN LEEUWEN, G. C., WEINBERG, V. Best Practice Guide-Deep Learning. **Partnership for Advanced Computing in Europe (PRACE), Tech. Rep**, v. 2, 2019.
- RIESENHUBER, M.; POGGIO, T. Hierarchical models of object recognition in cortex. **Nature neuroscience**, v. 2, n. 11, p. 1019-1025, 1999.
- SHAFKAT, Irhum. Irhum. Intuitively understanding convolutions for deep learning. **Medium**. 2018. Disponível em:

https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1**.** Acesso em: 18 jul. 2022.

- SZE, V., CHEN, Y. H., EMER, J., SULEIMAN, A., & ZHANG, Z. Hardware for machine learning: Challenges and opportunities. In: **2017 IEEE Custom Integrated Circuits Conference (CICC)**. IEEE, 2017. p. 1-8.
- TEE, Garry J. **Russian peasant multiplication and Egyptian division in Zeckendorf arithmetic**. Department of Mathematics, The University of Auckland, New Zealand, 2002.
- VARGAS, A. C. G., PAES, A., VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In: **Proceedings of the xxix conference on graphics, patterns and images**, v. 1., n. 4, sn, 2016.
- WEST, Lynn; BELLEVUE, N. E. An introduction to various multiplication strategies. **Washington. USA. Merrill Prentice Hall**, 2011.
- WURZEL, P., e MARTINS, M. O. (2022). Classificação de imagens de mamografia com Machine Learning no auxílio de diagnósticos de câncer de mama. **Disciplinarum Scientia | Naturais e Tecnológicas**, v. 23, n. 2, p. 1-17, 2022.
- ZHANG, A., LIPTON, Z. C., LI, M., SMOLA, A. J. Dive into deep learning. **arXiv preprint arXiv:2106.11342**, 2021.