Em Proc da 33ª ACM Int Conf on
Supercomputing (ICS), junho de 2019

# Otimização full-stack para acelerar CNNs usando Potências de dois pesos com validação FPGA

Bradley McDanelÿ
Universidade de Harvard
mcdanel@fas.harvard.edu

Sai Qian Zhangÿ
Universidade de Harvard
zhangs@g.harvard.edu

HT Kung
Universidade de Harvard
kung@harvard.edu

Xin Dong
Universidade de Harvard
xindong@g.harvard.edu

## ABSTRATO

Apresentamos uma estrutura de otimização full-stack para acelerar a inferência de CNNs (Redes Neurais Convolucionais) e validar o abordagem com uma implementação de array de portas programáveis em campo (FPGA) . Ao otimizar conjuntamente modelos CNN, arquiteturas de computação, e implementações de hardware, nossa abordagem full-stack alcança desempenho sem precedentes no espaço de trade-off caracterizado por latência de inferência, eficiência energética, utilização de hardware e precisão de inferência. Uma implementação FPGA é usada como veículo de validação para nosso projeto, alcançando uma latência de inferência de 2,28ms para o benchmark ImageNet. Nossa implementação brilha porque tem Eficiência energética 9x maior em comparação com outras implementações enquanto alcança latência comparável. Um destaque da nossa abordagem que contribui para a alta eficiência energética alcançada é uma arquitetura Seletor-Acumulador (SAC) eficiente para implementação CNNs com potências de dois pesos. Comparado a uma implementação FPGA para um MAC tradicional de 8 bits, o SAC reduz substancialmente recursos de hardware necessários (4,85x menos tabelas de pesquisa) e energia consumo (2,48x).

## CONCEITOS DE CCS

**• Metodologias de computaçãoÿ Redes Neurais;• Informática organização de sistemas** ÿ **matrizes sistólicas; Redes neurais; • Hardware** ÿ **Codesign de hardware-software.**

### PALAVRAS-CHAVE

matrizes sistólicas; redes neurais; escassez; otimização conjunta; co- design; potências de dois pesos

**Formato de referência ACM:**
Bradley McDanel, Sai Qian Zhang, HT Kung e Xin Dong. 2019. Otimização full-stack para acelerar CNNs usando potências de dois pesos com validação FPGA. Na Conferência Internacional de Supercomputação de 2019 (ICS '19), 26 a 28 de junho de 2019, Phoenix, AZ, EUA. ACM, Nova York, NY, EUA, 12 páginas. https://doi.org/10.1145/3330345.3330385

ÿContribuição Igualitária.


ICS '19, 26 a 28 de junho de 2019, Phoenix, AZ, EUA

ACM ISBN 978-1-4503-6079-19/01/06. . . US$ 15,00
https://doi.org/10.1145/3330345.3330385

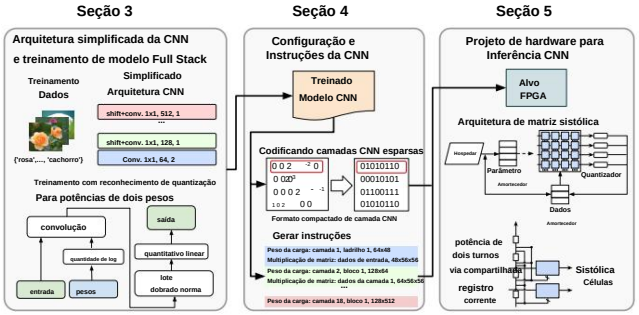



**Figura 1: Uma visão geral da estrutura de otimização full-stack proposta para acelerar a inferência de dados esparsos e CNNs quantizadas. A seção 3 detalha o treinamento CNN, que inclui restrições de dispersão e quantização para corresponder ao arquitetura de computação proposta. A seção 4 descreve o processo de conversão de uma CNN treinada em uma representação compactada para implementação de matriz sistólica e como o FPGA instruções são geradas para cada camada de convolução. A seção 5 descreve a arquitetura proposta com células sistólicas baseadas em seletor-acumulador (SAC) que são usadas para realizar inferência para todas as camadas de convolução no FPGA.**

## 1. INTRODUÇÃO

Devido ao amplo sucesso das Redes Neurais Convolucionais (CNNs) em uma variedade de domínios, houve extraordinários esforços de pesquisa e desenvolvimento direcionados para melhorar a latência de inferência, a eficiência energética e a precisão dessas redes. Geralmente, esses esforços de pesquisa podem ser vistos a partir de dois aspectos distintos. perspectivas: (1) profissionais de aprendizado de máquina que se concentram na redução da complexidade das CNNs por meio de convolução mais eficiente operações [45], quantização de peso e ativação [22] e peso poda [13] e (2) especialistas em arquitetura de hardware que projetam e construir aceleradores CNN com o objetivo de minimizar o consumo de energia, consumo de memória e custo de E/S [8, 23, 43, 48].

No entanto, abordar o problema a partir de apenas um destes dois pontos de vista podem levar a soluções abaixo do ideal. Por exemplo, conforme discutido na Seção 2.4, muitos métodos de quantização de peso de baixa precisão métodos podem omitir fatores de custo significativos em uma implementação ponta a ponta, como o uso de pesos e dados de precisão total para o primeiro camada [1, 50] ou usando normalização de lote de precisão total [21]. No lado do hardware, a maioria dos aceleradores CNN são projetados para suportar alguns direcionar CNNs (por exemplo, AlexNet [26] e VGG-16 [39]) em 8 ou 16 bits precisão para pesos e dados [7, 12]. Portanto, esses aceleradores geralmente não são diretamente aplicáveis a muitos dos avanços recentes

no projeto de CNN, incluindo estruturas CNN eficientes (usando, por exemplo, filtros separáveis [16]), poda de peso (usando, por exemplo, Lasso [40]), e quantização de baixa precisão.

Para resolver essa disparidade, neste artigo propomos um full-stack estrutura de otimização, onde o design do modelo CNN é otimizado em conjunto (co-projetado) com as arquiteturas de computação e implementações de circuitos nos quais ele será executado. A Figura 1 fornece uma visão geral do método proposto em três etapas, abordadas em três seções. A seção 3 descreve a etapa de treinamento, que utiliza um gráfico de quantização com reconhecimento de hardware para facilitar o treinamento de uma CNN. A CNN treinada pode permitir a implementação direta de quantizados cálculos em um FPGA sem qualquer sobrecarga adicional. A seção 4 descreve o processo de geração de instruções para executar inferência dada tanto a CNN treinada quanto a matriz sistólica de algum tamanho fixo implementado no FPGA de destino. Também cobre como a CNN esparsa e quantizada treinada é codificada para eficiência uso de memória FPGA. A seção 5 descreve a matriz sistólica serial de bits arquitetura que inclui o uso de esparsos livres de multiplicação células sistólicas, baseadas na arquitetura Seletor-Acumulador (SAC) para a operação de acumulação multiplicadora (MAC).

As novas técnicas do artigo são as seguintes:

- Uma **estrutura de otimização full-stack** onde a arquitetura de hardware informa a estrutura da CNN no treinamento.
- **Seletor-acumulador** (SAC) que fornece eficiência inferência livre de multiplicação para CNNs treinadas com potências de dois pesos. Substituímos o hardware MAC tradicional de 8 bits com SAC barato facilitado por simples cadastro compartilhado cadeias (Seção 5.2).
- Um **bloco de construção de matriz sistólica** para CNNs de baixa precisão (Seção 5.1) que utiliza cadeias de registros compartilhadas para dois propósitos : propagar dados para células sistólicas adjacentes e realizar multiplicação com pesos de potências de dois. Sistólica matrizes são usadas como um exemplo de matrizes de processador (nosso projeto de cadeia de registradores pode se estender a outros processos de processamento paralelo). arquiteturas de matriz).
- Uma **estrutura CNN simplificada** que alcança competitividade desempenho no ImageNet na configuração móvel usando apenas Convolução 1×1 sem conexões residuais (Seção 3.1).
- **Remodelação de entrada** que diminui a resolução espacial de a imagem de entrada enquanto aumenta o número de canais (Seção 3.3). Isso aumenta significativamente a matriz sistólica utilização para a primeira camada de convolução, que é uma parte significativa do tempo de execução total.

Aproveitar todos esses avanços em um único sistema é um desafio e uma das principais conquistas deste trabalho. Nós temos construiu um mecanismo de inferência CNN eficiente em um FPGA (Xilinx VC707 banca de avaliação) e validaram sua exatidão verificando a saída em relação à saída do nosso software. Todo o tempo e poder os resultados de consumo relatados neste documento são baseados no real medições obtidas a partir desta implementação de FPGA. Nosso FPGA O design é composto quase inteiramente por tabelas de consulta (LUTs). Nós usamos DSPs apenas para implementar a camada final totalmente conectada.

Acreditamos que nosso projeto fornece uma base útil para futuras implementações de ASIC. Nosso código de treinamento CNN (usando PyTorch [35]), python código que converte uma CNN esparsa e quantizada treinada em um representação compactada para FPGA e código Verilog para FPGA

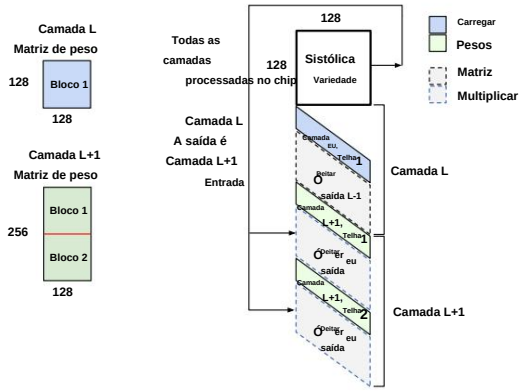



**Figura 2: Inferência CNN para duas camadas consecutivas (camada L e L + 1) usando uma única** matriz sistólica de 128×128 **semelhante a a implementação neste artigo. A matriz sistólica executa alternativamente instruções de peso de carga e multiplicação de matriz para todos os blocos em uma camada (seis instruções no total para este exemplo; consulte a Seção 4.2).**

implementação estão disponíveis em https://github.com/BradMcDanel/multiplicação-livre-dnn.

## 2 ANTECEDENTES E TRABALHOS RELACIONADOS

Nesta seção, descrevemos primeiro a inferência CNN usando matrizes sistólicas e resumir os recentes aceleradores CNN baseados em FPGA que compare com a Seção 6. Em seguida, revisamos os avanços na eficiência CNNs que são usadas como ponto de partida para nossa abordagem.

### 2.1 Matrizes Sistólicas

Uma matriz sistólica [27] é uma coleção de células sistólicas interconectadas. Geralmente, cada célula sistólica em uma matriz sistólica é codificada para realizar a mesma operação aritmética simples (por exemplo, uma operação MAC ). Além disso, a comunicação (isto é, fluxo de dados) na pressão sistólica array ocorre apenas entre células vizinhas. Essas propriedades fazem matrizes sistólicas atraentes para problemas computacionais que dependem fortemente em um único tipo de operação aritmética, como MAC em Inferência da CNN.

A Figura 2 mostra como uma matriz sistólica implementada em um FPGA realiza inferência CNN reutilizando a matriz nas camadas CNN. Observe que para a sincronização do array sistólico, os itens inseridos e a saída da matriz está devidamente distorcida, conforme mostrado na figura. Quando uma camada tem mais filtros do que a matriz sistólica pode suportar, particionamos a camada em blocos verticais através de filtros, conforme mostrado em à esquerda da figura e reutilize a matriz sistólica nesses blocos. Quando uma camada tem mais canais do que a matriz sistólica pode suportar, particionamos a camada em blocos horizontais entre canais (estes ladrilhos horizontais não são mostrados na figura).

### 2.2 Aceleradores FPGA para CNNs

Nos últimos anos, vários projetos de FPGA para inferência CNN foi proposto (geralmente visando redes proeminentes como LeNet-5 [29], AlexNet [26] e VGG-16 [39]) com os objetivos principais de baixa latência e alta eficiência energética. Uma estratégia comum

implementado por esses projetos é minimizar o grau de peso e movimentação de dados, especialmente da memória fora do chip, pois eles adicionam uma sobrecarga significativa em termos de latência e consumo de energia.

Uma abordagem para minimizar a movimentação de dados é a fusão de camadas, onde múltiplas camadas CNN são processadas ao mesmo tempo em pipeline para permitir o uso instantâneo de dados intermediários sem acesso à memória externa [20, 30, 46]. Outra abordagem, usada para filtros convolucionais 3×3 ou maiores, é determinar a ordem do cálculo de inferência que minimiza o número de somas parciais que devem ser armazenadas [33, 49]. Como nossa arquitetura CNN (Seção 3) usa apenas filtros 1×1 , a convolução é reduzida à multiplicação de matrizes , que pode ser implementada de forma eficiente usando matrizes sistólicas.

Além disso, diferentes estratégias de computação são frequentemente adotadas para a primeira camada [47], pois ela possui apenas três canais de entrada no caso de imagens RGB e a camada final totalmente conectada [36], onde há significativamente mais pesos do que dados. Neste trabalho, propomos usar o mesmo bloco de construção da matriz sistólica para implementações eficientes de todas as camadas de convolução em uma CNN.

## 2.3 Estruturas CNN eficientes

Desde que o VGG-16 [39] foi introduzido em 2014, tem havido uma tendência geral de projetar CNNs mais profundas através do uso de conexões residuais (ResNets[14]) e conexões concatenativas (DenseNet [19]), já que redes mais profundas tendem a alcançar maior precisão de classificação para conjuntos de dados de benchmark como ImageNet [6].
No entanto, conforme apontado na Tabela 2 do artigo original da ResNet [14], as conexões residuais parecem adicionar pouca melhoria na precisão da classificação de uma CNN mais rasa (18 camadas). Com base nessas observações, optamos por usar uma CNN mais rasa (19 camadas) sem quaisquer conexões residuais ou concatenativas, que descrevemos na Seção 3.1. Na nossa avaliação (Secção 6.5.3) mostramos que para esta CNN mais superficial, a exclusão de ligações adicionais tem um impacto mínimo na precisão da classificação, ao mesmo tempo que simplifica significativamente a nossa implementação de hardware e melhora a sua eficiência.

Além disso, várias alternativas à convolução padrão foram propostas recentemente para reduzir o custo de computação. A convolução separável em profundidade [3] reduz drasticamente os pesos numéricos e as operações, separando uma camada de convolução padrão em duas camadas menores: uma camada em profundidade que utiliza apenas pixels vizinhos dentro de cada canal de entrada e uma camada pontual que opera em todos os canais, mas não usa pixels vizinhos. pixels dentro de um canal (ou seja, ele usa apenas filtros 1×1 ). Wu et al. [45] mostraram que uma operação de mudança de canal pode ser usada para substituir a camada de profundidade sem um impacto significativo na precisão da classificação. Conforme descrito na Seção 3.1, nossa CNN proposta usa esta operação de mudança de canal imediatamente anterior a uma camada de convolução 1×1 . Observe que o paradigma de treinamento descrito na Seção 3 não é específico para convolução com deslocamento e também pode ser aplicado a redes mais convencionais (por exemplo, AlexNet, VGG e ResNet).

**2.4 Quantização de pesos e dados** Vários métodos foram propostos para quantizar os pesos CNN após o treinamento, usando representações de ponto fixo de 16 bits [12] e 8 bits [7] , sem impactar dramaticamente a precisão da classificação. Mais recentemente, métodos de quantização de baixa precisão (isto é, 1-4 bits), como métodos binários [4, 17] e quantização ternária [42, 50, 52] , têm também foram estudados, o que pode incorrer em alguma perda na precisão da classificação em comparação com abordagens de maior precisão. Geralmente, para esses métodos de baixa precisão, o treinamento ainda é realizado usando pesos de precisão total, mas o gráfico de treinamento é modificado para incluir operações de quantização que correspondam à aritmética de ponto fixo usada na inferência. Neste artigo, a quantização logarítmica [50] é adotada para pesos, com cada ponto de quantização sendo uma potência de dois.
Isso permite uma inferência significativamente mais eficiente, à medida que a multiplicação de ponto fixo é substituída por operações de deslocamento de bits correspondentes ao peso de potências de dois, conforme discutido na Seção 5.

Além da quantização de peso, também podemos quantizar a saída de dados ativados de cada camada CNN [1, 2, 37, 50, 51]. A quantização de dados reduz não apenas o custo das operações MAC, mas também o custo do acesso à memória para essas saídas intermediárias entre as camadas de uma CNN durante a inferência. No entanto, foi demonstrado que a quantização de ativação de baixa precisão (isto é, 1-4 bits) parece levar a uma degradação significativa na precisão da classificação em comparação com a quantização de peso [5, 31]. Devido a essas considerações, usamos quantização linear de 8 bits para dados neste artigo e nos concentramos em uma implementação eficiente de multiplicações de potências de dois pesos com dados de 8 bits.

Além disso, notamos que muitos dos métodos propostos para pesos e dados de baixa precisão omitem detalhes que são importantes para o desempenho eficiente do sistema de ponta a ponta. Primeiro, o trabalho nesta área muitas vezes trata a primeira camada de uma maneira especial, mantendo os pesos e os dados com precisão total para mitigar uma queda potencial na precisão da classificação [ 1, 5, 31]. Em segundo lugar, eles muitas vezes omitem explicitamente as considerações de quantização da normalização de lote e usam cálculo padrão de precisão total, conforme realizado durante o treinamento [1, 51]. Como a normalização em lote é essencial para a convergência de CNNs de baixa precisão, esta omissão dificulta a implementação eficiente de muitas abordagens de baixa precisão, pois são necessárias unidades de ponto flutuante para a normalização em lote. Neste trabalho, conforme discutido na Seção 3.2, lidamos com essas duas questões (1) quantizando os pesos e dados em todas as camadas (incluindo a primeira camada) sob um único esquema de quantização e (2) incluindo a quantização de normalização em lote no gráfico de treinamento (representado na Figura 6) para que adicione sobrecarga zero durante a inferência.

**2.5 Poda de Peso** É bem conhecido na literatura que a maioria dos pesos em uma CNN (até 90% para modelos grandes como o VGG-16) pode ser definida como zero (removida) sem ter um impacto significativo na precisão da classificação [ 13]. A rede podada resultante pode ter pesos distribuídos de forma esparsa com uma estrutura de dispersão irregular, o que geralmente é difícil de implementar de forma eficiente usando hardware convencional, como GPUs. Métodos subsequentes propuseram técnicas de poda estruturada que resultam em modelos com esparsidade estruturada [ 11, 15, 18, 32, 34, 44]. Embora esses métodos permitam implementações de CPU e GPU mais eficientes, eles parecem incapazes de atingir o mesmo nível de redução no tamanho do modelo que a poda não estruturada pode alcançar.

A combinação de colunas é um método de poda que permite camadas esparsas de CNN, mas garante que os pesos esparsos restantes possam ser compactados em um formato mais denso quando implantados em uma matriz sistólica [28].

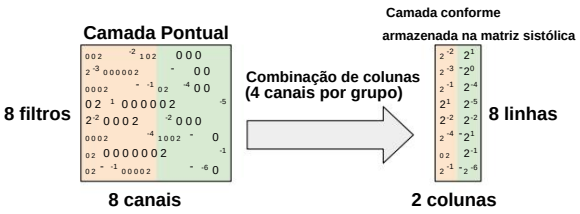



**Figura 3: Uma camada de convolução pontual (esquerda) com quatro canais por grupo resultante da poda para combinação de colunas [28]. Depois de combinar colunas na matriz de filtro (esquerda), cada grupo de quatro canais (mostrados em creme e verde) são reduzido em uma única coluna (direita). Observe que durante a combinação de colunas, para cada grupo, todas as entradas em cada linha são removido (podado), mas aquele de maior magnitude.**

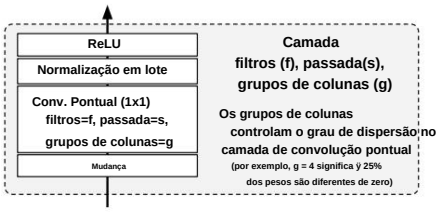



**Figura 4: Cada camada dos modelos CNN de avaliação neste o papel consiste em uma operação de deslocamento [45], convolução pontual (1x1), normalização de lote e ativação de ReLU. Uma camada é parametrizado com vários filtros (f), uma passada (s) e grupos de colunas (g) para combinação de colunas no empacotamento de um esparso camada convolucional [28].**

Em nosso pipeline de treinamento proposto, usamos combinação de colunas além de peso e quantização de dados, conforme discutido no item anterior. seção, a fim de obter uma inferência CNN esparsa e eficiente. Figura 3 mostra como uma camada de convolução pontual esparsa com pesos de potências de dois é convertida em um formato mais denso removendo tudo, exceto a maior entrada diferente de zero em cada linha nos canais combinados quando armazenado em uma matriz sistólica. Neste exemplo, a combinação de colunas reduz a largura da pequena camada por um fator de 4× de 8 para 2. Na Seção 5, descrevemos o projeto serial de bits para hardware eficiente implementação deste formato compactado mostrado no lado direito do Figura 3.

## 3 TREINAMENTO DE MODELO FULL-STACK

Nesta seção, primeiro fornecemos uma visão geral de nosso sistema simplificado Estrutura da CNN na Seção 3.1, direcionada para a implementação do FPGA relatada neste artigo. Em seguida, descrevemos os vários designs escolhas para melhorar a utilização e eficiência do sistema FPGA . Especificamente, empregamos um gráfico de treinamento com reconhecimento de quantização, incluindo normalização de lote quantizado (Seção 3.2) e uma entrada operação de remodelagem para melhorar a utilização de nossa matriz sistólica para a primeira camada de convolução (Seção 3.3). Os métodos propostos na Seção 3.2 e Seção 3.3 não são projetados especificamente para o rede de avaliação usada no artigo (descrita na Seção 3.1) e geralmente pode ser aplicado a qualquer estrutura CNN.

### 3.1 Arquitetura simplificada proposta para CNN

Nosso objetivo ao projetar uma arquitetura CNN é alcançar alta precisão da classificação usando uma estrutura simplificada em todas as camadas de convolução que pode ser mapeada eficientemente em uma taxa sistólica variedade. A Figura 4 mostra a estrutura de cada camada convolucional em nossa rede. Para obter desempenho semelhante ao padrão 3×3 convolução usando apenas convolução pontual (1×1) , cada camada começa com uma operação de mudança de canal conforme descrito em [45]. O a saída da operação de deslocamento é então aplicada a um ponto esparso camada de convolução, seguida de normalização em lote e retificação unidade linear (ReLU). Durante o treinamento, os pesos no sentido camada de convolução são podadas com combinação de colunas usando o parâmetro de grupos de colunas (g) como em [28]. Para a convolução anterior camadas em uma rede que têm menos pesos, um grupo de colunas tamanho de 2 é usado, o que reduz o número de pesos diferentes de zero em cerca de 50%. Para as últimas camadas CNN, que são maiores e possuem maior redundância, é utilizado um tamanho de grupo de 8 (uma redução de 87,5%). Cada camada é progressivamente podada ao longo do treinamento, como que após o treinamento eles alcançarão a dispersão alvo definida pelo grupos de colunas para a camada.

A Figura 5 mostra os modelos de avaliação para a estrutura CNN simplificada proposta para os conjuntos de dados CIFAR-10 [25] e ImageNet [6] . Conforme discutido na Seção 2.3, optamos por manter a rede relativamente raso (19 camadas) e sem quaisquer conexões residuais ou concatenadas. Na Seção 6, mostramos que esta simplificação estrutura pode alcançar a classificação competitiva Top-1 ImageNet precisão com baixa latência e alta eficiência energética. Avaliamos o ImageNet em três configurações de rede: ImageNet-Small/224, ImageNet-Small/56 e ImageNet-Large/56, onde 224 e 56 se referem para a largura e altura da imagem de entrada após o atraente estágio. Os modelos pequenos têm pesos de 1,5M e o modelo grande tem Pesos de 8,5M após o treino. Esse modelo de avaliação foi escolhido avaliar a importância do tamanho do modelo e do tamanho da entrada espacial na precisão da classificação, latência e rendimento. Além disso, como descrito na Seção 3.3, para as configurações com tamanho de entrada (56×56) , nós use uma operação de remodelagem para aumentar o número de canais de entrada de 3 (para imagens RGB) para 48 para aumento da matriz sistólica utilização.

### 3.2 Treinamento consciente de quantização

Para obter alta precisão de classificação usando potências de dois pesos, adicionamos operações de quantização ao treinamento CNN gráfico para corresponder aos pesos de ponto fixo e aos dados usados na inferência. A Figura 6 mostra os gráficos de treinamento e inferência para uma única camada na CNN mostrada na Figura 4. Conforme discutido na Seção 2.4, este abordagem de injetar quantização no gráfico de treinamento é conhecida na literatura e tem sido usado principalmente para treinar binários e redes ternárias [4, 51]. Em nosso gráfico de treinamento, usamos quantização logarítmica para os pesos, que quantiza um valor subjacente de precisão total. peso (mostrado em azul) elevado à potência de dois mais próxima. Durante o treinamento , a retropropagação usa gradientes de precisão total para atualizar o pesos de precisão total em cada camada como em [4].

Além disso, realizamos quantização nas operações de normalização em lote que seguem cada camada convolucional. Geralmente,

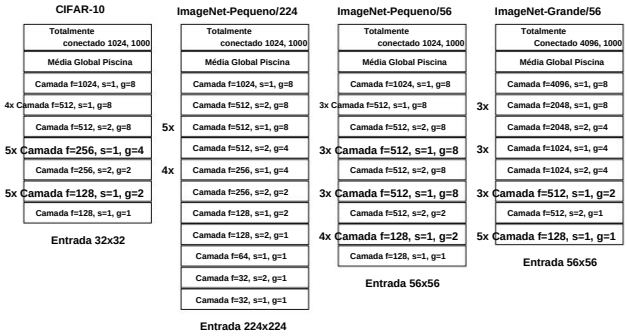



**Figura 5: Os modelos de avaliação para os conjuntos de dados CIFAR-10 e Ima-geNet. Cada rede consiste em 19 camadas de pesos treináveis, onde cada camada possui uma estrutura mostrada na Figura 4, com a primeira camada não incluindo um componente de deslocamento.**
**A redução da resolução é realizada usando convolução com passos indicados por camadas com passos de 2 (s = 2).**

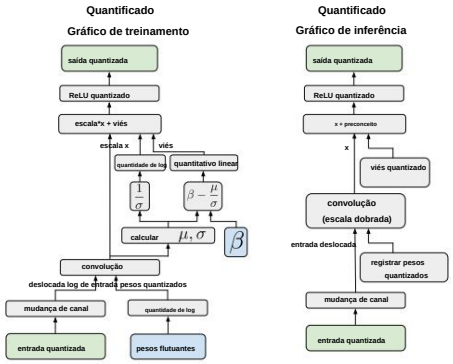



**Figura 6: O gráfico de treinamento quantizado (esquerda) realiza quantização linear para dados de entrada e quantização de potências de dois (quantização logarítmica) para parâmetros de normalização de peso e lote durante o treinamento. O gráfico de inferência (à direita) usa a versão quantizada dos pesos de precisão total aprendidos durante o treinamento e, portanto, não requer nenhuma operação de ponto flutuante.**

para pesos de maior precisão (por exemplo, pesos de 8 bits), esses parâmetros de normalização de lote podem ser dobrados diretamente nos pesos e termos de polarização da camada de convolução anterior após o treinamento, de modo que não tenham sobrecarga adicional [24]. No entanto, para pesos de menor precisão (como pesos binários ou potências de dois), este processo de dobramento introduz um erro de quantização significativo, levando a uma queda notável na precisão da classificação. Por esta razão, trabalhos anteriores que utilizam pesos de baixa precisão empregam normalização de lote de precisão total, incorrendo no custo de cálculo de precisão total correspondente. Para nossa arquitetura bit-serial proposta, essas operações de normalização em lote de precisão total introduziriam uma sobrecarga significativa e quebrariam nosso objetivo de inferência livre de multiplicação. Portanto, conforme mostrado na Figura 6, incluímos a quantização dos parâmetros de normalização do lote em nosso gráfico de treinamento. Aplicando quantização de log nos parâmetros da escala de normalização do lote permite que eles sejam dobrados nos pesos log quantizados sem introduzir qualquer erro de quantização.

A normalização em lote é definida como

$$xbn = \ddot{y}\left(\frac{x \ddot{y} \mu}{\ddot{y}}\right) + \ddot{y} \quad (1)$$

onde μ e ÿ são a média e o desvio padrão de cada minilote durante o treinamento e as estatísticas médias de execução durante a inferência . ÿ e ÿ são parâmetros que podem ser aprendidos que são introduzidos para melhorar o poder de representação da rede. Quando seguido por ReLU, como é o caso em nossa CNN, os efeitos do parâmetro de escala que pode ser aprendido ÿ podem ser capturados na camada de convolução seguinte e podem, portanto, ser omitidos definindo ÿ como 1 [10]. Em seguida, fatoramos μ, ÿ e ÿ em uma escala e termo de polarização como segue

$$xbn = \underbrace{\frac{1}{\ddot{y}}}_{\text{escala}} x + \underbrace{\ddot{y} \ddot{y} \frac{\mu}{\ddot{y}}}_{\text{viés}} \quad (2)$$

Depois de aplicar a normalização de lote quantizado à saída da camada de convolução anterior, uma função de ativação não linear (ReLU) é executada, que define todos os valores negativos como 0. Além disso , aplica quantização linear de 8 bits nos dados para que corresponde ao cálculo do ponto fixo na inferência. O gráfico de inferência da Figura 6 mostra como a computação é realizada no FPGA durante a inferência. O fator de escala de normalização de lote log quantizado é dobrado nos pesos log quantizados na camada de convolução anterior. Somente operações de mudança de canal e adição de ponto fixo são realizadas durante a inferência.

**3.3 Remodelagem de entrada para melhorar a utilização** Para CNNs treinadas no ImageNet, a primeira camada de convolução representa 10-15% do cálculo total de inferência realizado devido ao grande tamanho espacial da imagem de entrada (3×224×224). No entanto, conforme discutido recentemente por Xilinx [47], o cálculo nesta camada não é bem mapeado para uma matriz sistólica, porque a imagem de entrada possui apenas três canais de entrada, o que significa que a maior parte da largura de banda de entrada da matriz pode não ser utilizada. Para resolver esse desequilíbrio, a Xilinx propõe o uso de duas matrizes sistólicas, uma matriz sistólica designada especificamente para a primeira camada e a outra matriz sistólica usada para as camadas de convolução restantes na CNN.

Neste artigo, em vez de usar uma matriz sistólica diferente para a camada de entrada, remodelamos a imagem de entrada para a CNN para aumentar a utilização de nossa matriz sistólica única para a primeira camada convolucional. A Figura 7 mostra a operação de remodelagem de entrada para uma imagem RGB com 3 canais e 224×224 pixels por canal. Cada bloco 2×2 de pixels é dividido em quatro grupos (denotados 1, 2, 3 e 4) com 1 pixel por grupo. Os pixels do mesmo grupo em todos os blocos 2×2 são então colocados em um novo conjunto de canais RGB (4 grupos, cada um com canais RGB, levando a um total de 12 canais). Cada um desses canais possui 112×112 pixels, o que representa um quarto da imagem de entrada original. Na Seção 6, avaliamos as redes ImageNet-Small/56 e ImageNet-Large/56 com uma operação de remodelagem ainda mais agressiva , onde usamos 16 grupos para converter a imagem de entrada 3×224×224 em uma imagem 48×56×. 56 entrada para a CNN.

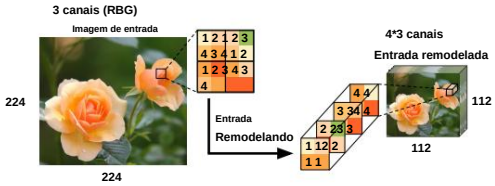



**Figura 7: Remodelação dos dados de entrada diminuindo o tamanho espacial e aumentando o número de canais para melhorar a utilização do array sistólico no processamento da primeira camada.**

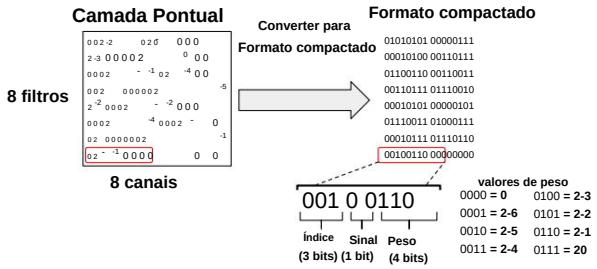



**Figura 8: Uma camada pontual esparsa com potências de dois pesos (esquerda) é convertida em uma representação compactada para armazenamento eficiente em um FPGA (direita), onde cada grupo de canais combinados (4 neste exemplo) produz 1 canal de 8 bits codificação por filtro (linha).**

#### 4 CONFIGURAÇÃO E INSTRUÇÕES

Nesta seção, mostramos como nossa CNN treinada descrita na Seção 3 é codificada para uso eficiente em um FPGA (Seção 4.1). Em seguida, explicamos como os pesos em cada camada são divididos em blocos que cabem em uma determinada matriz sistólica e as instruções correspondentes para cada bloco executado no FPGA (Seção 4.2).

**4.1 Codificando camadas CNN esparsas para FPGA** Após a conclusão do treinamento, cada camada de convolução terá atingido uma dispersão alvo definida pelo parâmetro do grupo de colunas para a camada, conforme descrito na Seção 3.1. O lado esquerdo da Figura 8 ilustra os pesos de uma camada de convolução pontual após o treinamento com 8 filtros, 8 canais e grupos de colunas de tamanho 4. Para esta camada, cada grupo de 4 canais será combinado em uma única coluna na matriz sistólica no FPGA, pois há no máximo uma entrada diferente de zero por filtro (linha) em cada grupo. Os pesos restantes diferentes de zero são potências de dois devido ao esquema de quantização de peso discutido na Seção 3.2.

Para ilustrar o procedimento de codificação, destacamos 4 pesos em vermelho na camada pontual mostrada na Figura 8 , que será convertida em uma representação de 8 bits no formato compactado. O segundo elemento neste grupo é o peso diferente de zero -2 3 bits na codificação armazena o ÿ1 . O primeiro índice do peso diferente de zero que é 1 (001) correspondente ao segundo elemento do grupo. Observe que para camadas maiores combinamos até 8 canais, exigindo um índice de 3 bits. Os 5 bits restantes são compostos de um bit de sinal e 4 bits para indicar o peso da potência de dois, que é 00110 para ÿ2 ÿ1 . Como

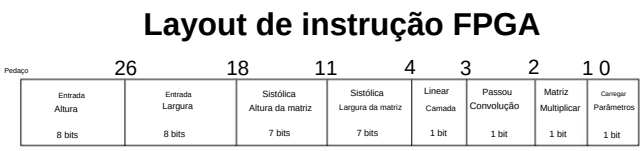



**Figura 9: O layout de instruções do FPGA para uma matriz sistólica** de 128×64 em um FPGA.

representado na Figura 8, os pesos representativos de potências de dois são 0 ordenados do menor para o maior (por exemplo, 2 é 0111), é 0001 e 2 com 0000 sendo usado para representar 0. Sistólica, para configurar cada ÿ6 São necessários 8 bits.

**4.2 Instruções para FPGA**

A inferência CNN é composta por uma série de multiplicações matriz-matriz, uma por camada de convolução, entre os dados que são inseridos em uma camada e os pesos aprendidos de uma camada. Ao usar uma única matriz sistólica para realizar as multiplicações de matrizes para todas as camadas, as instruções geradas realizarão um processo relativamente simples de carregar alternativamente pesos na matriz sistólica e realizar a multiplicação de matrizes entre os dados e os pesos carregados, em ordem sequencial do Camadas CNN. No entanto, quando uma matriz de peso é maior que o tamanho fixo do conjunto sistólico, ela deve ser particionada em blocos menores, onde cada bloco pode caber no conjunto sistólico. Em seguida, um par de instruções de carregamento de peso e multiplicação de matrizes são agendadas para cada bloco. Neste artigo, usamos a combinação de colunas para reduzir drasticamente o número de colunas para inferência (por exemplo, de 512 canais para 64 colunas na matriz sistólica por meio de combinação de 8 vias), diminuindo o número total de blocos.

A Figura 2 mostra como a inferência é realizada em duas camadas (camada L e camada L + 1) usando uma única matriz sistólica. Primeiro, uma instrução de pesos de carga é usada para carregar a matriz de peso de 128 filtros por 128 canais para a camada L. Em seguida, a multiplicação da matriz é realizada entre os pesos carregados e a saída da camada anterior (camada L - 1), passando os dados para o sistólico. variedade. Esta multiplicação da matriz gera a saída da camada L que será usada para a camada L + 1. Como a matriz de peso da camada L + 1 de 256×128 é maior que a matriz sistólica de 128×128, ela é particionada em dois blocos. O primeiro bloco da camada L + 1 é então carregado na matriz sistólica e multiplicado pela saída da camada L, o que gera metade da saída da camada L + 1. O segundo bloco da camada L + 1 é então processado da mesma maneira como a primeira peça. Um total de seis instruções são usadas na Seção 3.2 par de instruções de carga de peso e multiplicação de matriz para a camada L e dois pares de instruções para a camada L + 1.

A Figura 9 mostra o layout de instruções FPGA para a arquitetura da matriz sistólica descrita na Seção 5. Uma instrução de peso de carga é indicada quando o primeiro bit é definido, com os campos de largura e altura da matriz sistólica controlando o tamanho do bloco que está sendo carregado na matriz. Uma instrução de multiplicação de matriz é indicada quando o segundo bit é definido. A altura da matriz de dados a ser multiplicada pelos pesos carregados é definida pelos campos de largura e altura de entrada (por exemplo, 56×56 para a primeira camada).

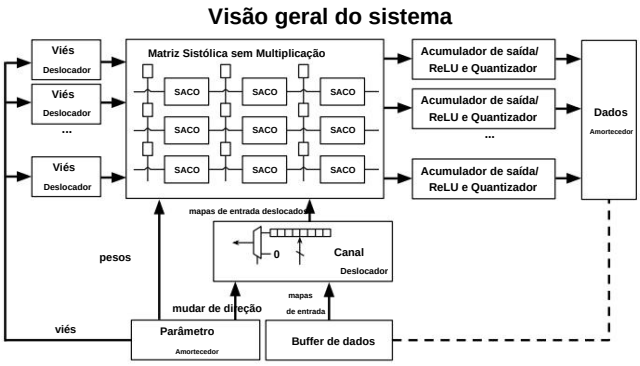



**Figura 10: Projeto do sistema implementado em um FPGA.**

## 5PROJETO DE FPGA

Nesta seção, fornecemos uma descrição detalhada de nosso projeto FPGA para inferência CNN esparsa com potências de dois pesos.

### 5.1 Descrição do Sistema

A Figura 10 mostra uma visão geral do sistema de inferência CNN implementado em um FPGA. O buffer de parâmetro armazena os pesos do filtro, polarizações e direções de mudança para a operação de mudança de canal [45]. Durante uma instrução de peso de carga, os pesos do filtro são carregados na matriz sistólica (Seção 5.2) e a polarização do filtro e as direções de mudança de canal são carregadas na polarização e nos deslocadores de canal, respectivamente. Durante uma instrução de multiplicação de matrizes, os dados de entrada são carregados do buffer de dados para os deslocadores de canal, que realizam as operações de deslocamento antes de enviar os dados para a matriz sistólica em série de bits [28]. Cada coluna na matriz sistólica recebe dados de entrada de vários canais de entrada para suportar a combinação de colunas mostrada na Figura 3. Cada célula Seletor-Acumulador (SAC) (Figura 11) dentro de uma coluna da matriz sistólica recebe a entrada múltipla. canais em diferentes potências de dois deslocamentos de deslocamento para selecionar o índice de canal e o índice de peso de potências de dois para a célula correspondente ao formato compactado na Figura 8.

A saída de cada linha da matriz sistólica é passada para o bloco ReLU & Quantization (Seção 5.4) antes de armazenar os resultados de volta no buffer de dados. Os dados de saída armazenados no buffer de dados da camada anterior são os dados de entrada da próxima camada. O acumulador de saída (Seção 5.5) é usado apenas na camada final (totalmente conectada) para reduzir o mapa de características de cada classe a um único número usado para previsão. Os parâmetros para o próximo bloco de peso são carregados na DRAM off-chip (não mostrada na Figura 10) no buffer de parâmetros à medida que a multiplicação da matriz é realizada na matriz sistólica para o bloco atual. Durante a inferência, todos os resultados intermediários são armazenados na RAM do chip no buffer de dados.

### 5.2 Seletor-Acumulador (SAC) para Projeto de Matriz Sistólica Livre de Multiplicação

Nesta seção, descrevemos nosso projeto Seletor-Acumulador (SAC) para uma matriz sistólica livre de multiplicação para multiplicação de matrizes esparsas . Escolhemos um design serial de bits para multiplexação eficiente de vários fluxos de dados em uma única coluna do array para suportar a combinação de colunas [28]. No layout do livre de multiplicação

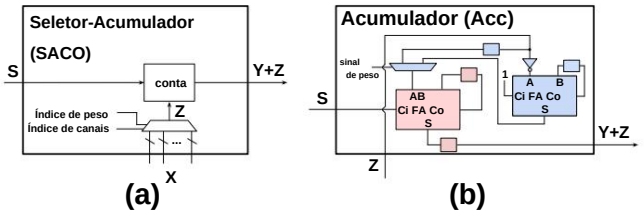



**Figura 11: Seletor-acumulador serial de bits (SAC).**

**Tabela 1: Comparação de recursos e potência do FPGA para um** array sistólico 64×64 **implementado com MAC e SAC.**

| | 64×64 MAC | 64×64 SAC | MAC/SAC |
|---|---|---|---|
| LUT | 212388 | 43776 | 4,85× |
| FF | 192293 | 54330 | 3,54× |
| Poder | 4,21W | 1,7W | 2,48× |

matriz sistólica (mostrada na Figura 10), cada coluna da matriz leva até oito canais de entrada (para suportar a combinação de colunas) na cadeia de registro da coluna em uma forma serial de bits. A cada ciclo , os dados de entrada são deslocados para o próximo registro na cadeia. Essa cadeia de registros atende ao propósito padrão em uma matriz sistólica de propagação de dados através das células da coluna. No entanto, ao usar pesos de potências de dois, pode servir a um propósito adicional de multiplicação de pesos de potências de dois, já que cada posição crescente na cadeia de registro corresponde aos dados de entrada sendo multiplicados por uma potência crescente de dois. Em nosso projeto, utilizamos esta observação da natureza de dupla finalidade da cadeia de registros ao usar potências de dois pesos para projetar uma célula sistólica mais eficiente.

A Figura 11a mostra as células do seletor-acumulador (SAC) que pegam os dados de entrada de vários pontos na cadeia de registro e selecionam o ponto correspondente ao peso de potências de dois armazenado na célula usando um multiplexador. Além disso, utiliza um índice de canal, também armazenado na célula, para determinar a posição do peso na matriz de filtro esparso original (veja a Figura 8 para detalhes sobre o esquema de indexação). O elemento selecionado é então passado para o acumulador serial de bits mostrado na Figura 11b. Os elementos lógicos azuis no acumulador negam o produto Y com base no sinal do peso das potências de dois e adicionam o resultado ao acumulador serial de bits ( somador completo rosa) antes de passar o resultado para o SAC à direita.

Comparado com um acumulador multiplicador (MAC) de 8 bits que requer 8 somadores completos de 1 bit para multiplicação, o SAC requer apenas um multiplexador para selecionar um deslocamento correspondente a um peso de potência de dois . Usando o conjunto de design Xilinx Vivado, observamos que, comparado a um MAC tradicional de 8 bits, o SAC reduz substancialmente os LUTs necessários (4,85×), FFs (3,54×) e o consumo de energia (2,48×), conforme mostrado na Tabela 1. Como mostramos discutido na Seção 6.2, isso reduz drasticamente o custo de hardware de cada célula sistólica e permite que uma matriz sistólica substancialmente maior seja implementada no FPGA do que com células MAC padrão de 8 bits.

A Figura 12 mostra como uma cadeia de registro é usada na geração de uma versão deslocada dos dados de entrada 10010 (vermelho) com a quantidade de deslocamento correspondente ao peso de potências de dois associado à célula ao longo das etapas de tempo (T = 0, 1, 2 , . Conforme ilustrado na Figura 12 (a), (b) e (c), suponha que o SAC exija uma versão alterada do original

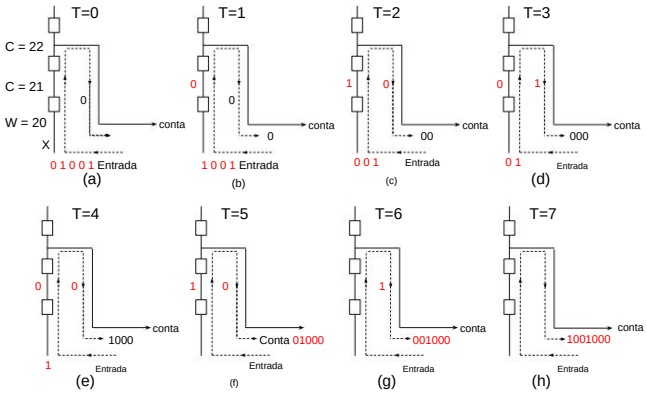



**Figura 12: Um exemplo de envio de uma versão alterada de entrada fluxo (correspondente a uma multiplicação com peso de potência de dois) para uma célula SAC em um formato serial de bits. Neste exemplo, o peso é 2 2.**

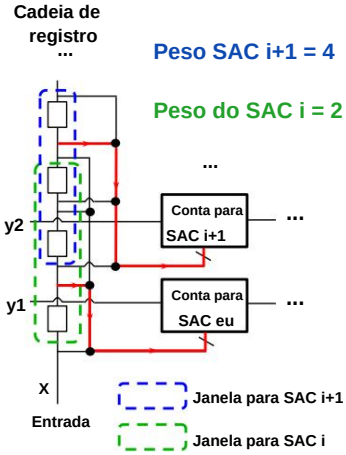



**Figura 13: Cadeia de registros com janela por célula para pesos de potências de dois para duas células adjacentes em uma coluna da matriz sistólica. As linhas vermelhas mostram a posição onde o deslocamento versões da entrada original são obtidas do registro corrente.**

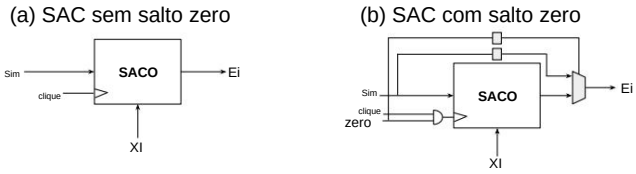



**Figura 14: SAC sem zero-skipping (a) e com zero-skipping (b).**

entrada com dois zeros pendentes no início (o peso do filtro é quatro). Então, o Acumulador (Acc) capturará o fluxo de dados de entrada no segundo registro na cadeia de registros, então os primeiros dois bits enviados para o Acc são zeros (preto). Após 4 ciclos adicionais, o Acc recebe um entrada de 1001000, que é quatro vezes a entrada original 10010.

A Figura 13 mostra como a cadeia de registradores pode ser compartilhada entre dois células SAC consecutivas em uma coluna da matriz sistólica. Suponha que cada de duas células SAC pode exigir qualquer uma das três versões deslocadas da entrada original (correspondendo a três possíveis potências de dois pesos). Então, isso leva ao uso de duas janelas com vão de três na cadeia de registros (mostrados em verde e azul na Figura 13). As linhas vermelhas nas figuras mostram as posições onde as células SAC pegue as versões deslocadas da entrada original do registro corrente. Assim, a cadeia de registradores é usada para dois propósitos: (1) deslocamentos os dados de entrada para cima para todas as células SAC na mesma coluna e (2) gera as versões deslocadas dos dados de entrada para o multiplicação de potências de dois.

## 5.3 SAC energeticamente eficiente com Zero-Skipping

Cada SAC pode ser desligado para economizar energia quando o peso ou a entrada para o SAC é zero, o que chamamos de salto zero. O estrutura de um SAC com e sem mecanismo zero-skipping são mostrados na Figura 14. Para o SAC com zero-skipping, o zero O sinal é definido quando a entrada ou o peso é 0 e é usado como o sinal de habilitação para o relógio fechado. Quando o sinal zero é definido devido à entrada atual ser 0, o acúmulo Yi ignora SAC e é encaminhado diretamente para o próximo SAC na linha em a matriz sistólica. Observe que devido ao ReLU, aproximadamente metade dos os elementos de dados são zero, o que significa que o SAC será desabilitado cerca de metade do tempo. Quando o peso do SAC é 0, então o SAC ficará desabilitado para toda a multiplicação da matriz. Em Na Seção 6.3, mostramos que esse mecanismo de salto zero reduz potência em cerca de 30%.

## 5.4 Projeto de ReLU e Bloco de Quantização

Conforme mencionado na Seção 2.4, usamos uma representação de ponto fixo de 8 bits para os dados de entrada de cada camada. Portanto, a quantização O processo deve converter a saída do acumulador de maior precisão (32 bits) da matriz sistólica de volta para um intervalo de 8 bits para ser usado como entrada para a próxima camada. Neste artigo, uma vez que a escala de ponto fixo fator é compartilhado por todas as camadas, esta etapa de quantização simplifica para extraindo um intervalo de 8 bits do acumulador de 32 bits. Esta etapa de quantização pode ser fundida com a função de ativação ReLU, por definir as saídas negativas do acumulador para 0.

A Figura 15 mostra a arquitetura do ReLU & Quantization bloquear. Uma matriz de registradores é usada para coletar a saída de 32 bits do a matriz sistólica. O resultado de 32 bits é deslocado pelo menor peso de potências de dois representável (por exemplo, $2^{6}$ conforme mostrado na Figura 8) e passado para o comparador. O comparador gera o indicador bit para o multiplexador, que corta o resultado entre (0, 255) antes armazenando-o de volta no buffer.

## 5.5 Projeto do Acumulador de Saída

Dados os canais de saída produzidos pelo convolucional final camada, o agrupamento médio é usado para reduzir os componentes espaciais de cada canal para um único valor médio. Para nossa matriz sistólica única implementação, dobramos esta operação de pooling nos pesos de camada totalmente conectada. Deixe $x^{k}$ k e $\bar{x} = \frac{\sum_{eu=1}^{R} x^{k}}{R}$ denotar o i-ésimo elemento do mapa de entrada k e a média do canal de entrada k,

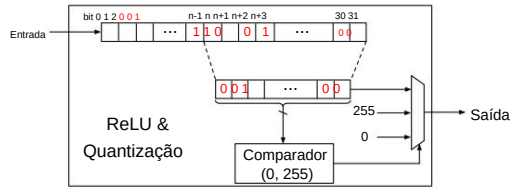
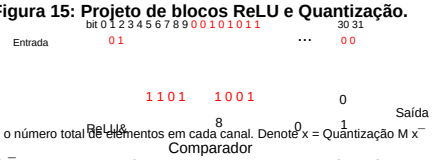



**Figura 15: Projeto de blocos ReLU e Quantização.**




onde R é o número total de elementos em cada canal. Denote x = Quantização M x

{x } como o vetor de médias dos canais, onde M é o número total de canais de entrada. Temos a seguinte derivação para a saída y da camada totalmente conectada:

$$y = L\,x = W\,\frac{\sum_{i=1}^{R} xi}{R} = \sum_{eu=1}^{R} \frac{W xi}{R} \qquad (3)$$

onde xi = {x } e W é a matriz de pesos do camada totalmente conectada. Da equação 3, notamos que xi pode ser calculado realizando a multiplicação da matriz entre e X = {x } com a matriz sistólica e y pode ser calculado por somando todos os WRxi .

O acumulador de saída é usado para calcular a soma de R xi .

O fluxo de saída de 32 bits de uma linha na matriz sistólica entra no acumulador de saída de forma serial de bits. Este fluxo é paralisado em uma matriz de registradores até que o bit final chegue. A saída de 32 bits resultante é adicionada ao próximo fluxo de saída de 32 bits. Usamos DSPs no FPGA para realizar essas adições de 32 bits.

## 6 AVALIAÇÃO

Nesta seção, primeiro reiteramos brevemente as principais contribuições de nossa abordagem proposta a partir das perspectivas tanto do treinamento quanto do hardware da CNN e vinculamos cada contribuição à seção de avaliação correspondente (Seção 6.1). Em seguida, avaliamos o desempenho de nossa implementação FPGA em relação aos aceleradores FPGA de última geração no conjunto de dados ImageNet (Seção 6.2). A seguir, medimos o impacto do salto zero na eficiência energética (Seção 6.3) para matrizes sistólicas de diferentes tamanhos e o impacto da normalização do lote dobrável (Seção 6.4). Finalmente, na Seção 6.5, analisamos o impacto de nossa estrutura simplificada de CNN e procedimento de treinamento apresentado na Seção 3 na precisão da classificação, incluindo remodelagem de entrada, usando potências de dois pesos, e a omissão de conexões residuais da estrutura CNN mencionada. na Seção 3.1.

Nós nos concentramos em duas métricas principais de desempenho: (1) latência da entrada da imagem até a saída da classificação e (2) eficiência energética, que é o número de imagens que o mecanismo de inferência pode processar por joule. Observe que o último também é o número de imagens/seg (ou seja, taxa de transferência ) por watt. Para aplicações de inferência de alto rendimento, podemos usar vários mecanismos de inferência em paralelo. Se cada um desses motores individuais oferecer baixa latência e inferência com alta eficiência energética, então o sistema agregado fornecerá inferências de alto rendimento por watt, ao mesmo tempo em que atenderá aos requisitos de baixa latência de inferência.

**6.1 Recapitulação da otimização full-stack** A otimização full-stack por meio de treinamento permitiu os seguintes avanços de design que levam à nossa implementação eficiente de FPGA apresentada na Seção 5.

- Usando potências de dois para pesos e parâmetros de escala de normalização de lote, descritos na Seção 2.4, para todas as camadas de convolução na CNN. Isso permite um design simplificado, onde uma única matriz sistólica esparsa e livre de multiplicação é usada para todas as camadas da CNN. Na Seção 6.5.2, discutimos o impacto do esquema de quantização proposto na precisão da classificação. • Salto zero dos dados quantizados (Seção 5.3). Na Seção 6.3, mostramos que o salto de zero reduz o consumo de energia durante a multiplicação de matrizes em aproximadamente 30%. • Empacotamento de CNNs esparsas usando combinação de colunas [28] para armazenamento eficiente e uso em FPGAs, que descrevemos na Seção 4.1. Nosso modelo de avaliação ImageNet-Small/56 tem apenas 1,5M potências de dois pesos, que é 40× menor que AlexNet e 92× menor que VGG-16 (os dois CNNs usados por outros projetos de FPGA). • Usando deslocamentos de canal [45] para substituir convoluções 3×3 por convoluções 1×1 . Tal como acontece com a combinação de colunas, isto reduz o número de parâmetros do modelo. Além disso, agiliza o design do sistema de matriz sistólica, pois 1×1 reduz-se a uma multiplicação de matriz menor, em oposição aos filtros convolucionais 3×3 .

- Remodelação de entrada (Seção 3.3) para aumentar a utilização da matriz sistólica bit-serial e reduzir drasticamente a latência da primeira camada de convolução. Na Seção 6.5.1, mostramos que a remodelagem da entrada alivia parte da perda de precisão ao usar um tamanho de entrada espacial menor de 48×56×56 em vez do convencional 3×224×224.

**6.2 Comparando com aceleradores FPGA anteriores** Comparamos nosso projeto FPGA de 170 MHz com vários aceleradores FPGA de última geração no conjunto de dados ImageNet em termos de precisão de classificação top-1, latência para uma única imagem de entrada e eficiência energética quando não há lote o processamento é executado (ou seja, tamanho de lote de 1). Ao escolher essas métricas, nos concentramos em cenários em tempo real onde as amostras de entrada devem ser processadas imediatamente para atender a uma restrição de tempo difícil. Nosso modelo de avaliação é a rede ImageNet-Small/56 mostrada na Figura 5 com entrada remodelada para 48×56×56. Nosso FPGA pode acomodar uma matriz sistólica com 128 linhas por 64 colunas. Cada uma das colunas pode abranger até 8 canais na matriz de peso de convolução, ou seja, quando o parâmetro do grupo de colunas é definido como 8, para um total de 512 canais.

A Tabela 2 fornece uma comparação de nossa implementação de FPGA com outros aceleradores CNN baseados em FPGA. Nosso design atinge uma latência por imagem de 2,28 ms, que está entre as mais baixas de todos os designs. Além disso, em comparação com alguns dos trabalhos mais recentes [41, 49], nosso projeto os supera em 5,64× e 3,26×, respectivamente, em termos de eficiência energética. Além disso, em comparação com uma implementação que atinge baixa latência comparável [30], nossa implementação tem eficiência energética 9,29x maior.

Nosso projeto alcança a mais alta eficiência energética entre todos esses projetos por vários motivos. Primeiro, usamos uma estrutura CNN altamente eficiente (Seção 3.1) com pesos de apenas 1,5M (comparados

**Tabela 2: Comparação com outros aceleradores CNN baseados em FPGA.**

| | [49] [36] | | [46] | [33] | [30] | [38] | [41] | Nosso |
|---|---|---|---|---|---|---|---|---|
| Chip FPGA Xilinx | VC706 | ZC706 | ZC706 | Arria-10 | VC709 | Virtex-7 | ZC706 | VC707 |
| FF | 51 mil (12%) | 127 mil (29%) | 96 mil (22%) | 262 mil (30%) | 348 mil (40%) | 51 mil (12%) | 201 mil (33%) | |
| LUT | 86k(39%) | 182k(83%) | 148k(68%) | 161k(38%) | 273k(63%) | 236k(55%) | 86k(39%) | 239k(78%) |
| DSP | 808(90%) | 780(89%) | 725(80%) | 1518(100%) | 2144(59%) | 3177(88%) | 808(90%) | 112(4%) |
| BRAM | 303(56%) | 486(86%) | 901(82%) | 1900(70%) | 1913(65%) | 1436(49%) | 303(56%) | 834(81%) |
| Precisão (Principal 1) | 53,30% | 64,64% | N / D | N / D | N / D | 55,70% | 52,60% | 50,84% |
| Frequência (MHz) | 200 | 150 | 100 | 150 | 150 | 100 | 200 | 170 |
| Latência (ms) | 5,88 | 224 | 17.3 | 47,97 | 2,56 | 11.7 | 5,84 | 2.28 |
| Eficiência (img./S/W) | 23,6 | 0,46 | 6.13 | 0,98 | 12,93 | 8.39 | 40,7 | 120,7 |

**Tabela 3: Comparação do consumo de energia do salto zero.**

| | Sem pular Com pular | |
|---|---|---|
| 32×64 | 1,0 W 0,7 W | |
| 64×64 | 1,7W | 1,3W |
| 128x64 | 3,0 W | 2,2 W |

a 60M para AlexNet e pesos de 136M para VGG-16 [33, 36]). Nosso modelo na Tabela 2 é significativamente menor e todos os pesos (incluindo pesos em camadas de normalização de lote) são quantizados. Nossa precisão é 50,84% (cerca de 2% pior que os projetos concorrentes mais próximos [41] em termos de eficiência energética). No entanto, a nossa implementação tem pelo menos eficiência energética pelo menos 3x maior. Em segundo lugar, nossa quantização por potências de dois proposta (Seção 2.4) permite o uso de uma matriz sistólica livre de multiplicação (Seção 5.1), onde cada célula contém apenas um seletor e dois somadores completos (veja a Figura 11) que são mais eficientes comparado com [33] e possui estrutura mais simples em comparação com [38]. Isso permite que uma grande matriz sistólica (128×64) caiba no FPGA, reduzindo assim o número de blocos necessários para realizar inferência para cada amostra. Além disso, usando a combinação de colunas, podemos empacotar camadas CNN esparsas para implementação eficiente de matriz sistólica com alta utilização de hardware [28]. Além disso, DSPs são usados no acumulador de saída (Seção 5.5) apenas para um único totalmente camada conectada e são desativados para o restante das camadas. Finalmente, o mecanismo de salto zero, que avaliamos com mais detalhes em Seção 6.3, economiza ainda mais energia ao desligar dinamicamente a pressão sistólica células quando os dados que entram em uma célula são zero.

### 6.3 Redução de potência por salto zero

Para avaliar a redução de potência devido ao salto de zero, medimos o consumo de energia do FPGA durante a matriz multiplicação em duas configurações. A configuração “Sem pular” usa entradas que são todas diferentes de zero, o que significa que cada célula será ativo durante a multiplicação de matrizes. A configuração “Com salto” usa entradas que são meio zero, a fim de aproximar a saída de ReLU, que define aproximadamente metade dos elementos como zero [9].

A Tabela 3 mostra a quantidade de consumo de energia para inferência para as configurações “Sem pular” e “Com pular” por três matrizes sistólicas de tamanhos crescentes. Para todos os três tamanhos de matriz sistólica, observamos que “With Skipping” reduz o consumo de energia da multiplicação de matrizes em cerca de 30%.

### 6.4 Redução de potência por lote dobrável Normalização

Avaliamos ainda mais a economia em hardware e energia dobrando parâmetros de normalização em lote em pesos de filtro. Para comparação, implementamos um bloco de normalização em lote no FPGA, que executa operações MAC de ponto fixo de 8 bits nas saídas de 128× 64 matriz sistólica. Em seguida, medimos o hardware adicional e energia consumida pelo bloco de normalização em lote. Apresentando o bloco adicional para normalização de lote não consome apenas LUTs e FFs adicionais (12653 e 9377 respectivamente), mas também aumenta o consumo de energia em 0,6W.

### 6.5 Impacto do treinamento full-stack na precisão

Avaliamos agora o impacto das modificações tanto na CNN estrutura e procedimento de treinamento conforme proposto na Seção 3 sobre precisão de classificação.

6.5.1 Impacto da remodelação de insumos. Para determinar a eficácia da operação de remodelagem de entrada descrita na Seção 3.3, comparamos modelos usando o mesmo tamanho de entrada espacial com e sem remodelagem (por exemplo, 3×56×56 versus 48×56×56) e modelos com tamanho de entrada espacial diferente (por exemplo, 3×224×224 versus 48×56×56). Além disso , treinamos um modelo ImageNet maior (ImageNet-Large/56) usando remodelagem de entrada para ver a melhor precisão que nossa proposta abordagem pode alcançar quando usada com um pequeno tamanho de entrada espacial.

A Tabela 4 mostra a precisão da classificação para os quatro avaliados configurações de rede. Primeiro, observamos que o ImageNet-Small/56 com remodelagem é capaz de alcançar precisão de classificação semelhante à o ImageNet-Small/224 sem remodelar, mesmo com 16× menos pixels em cada canal. Isso mostra que a remodelagem de entrada permite imagens de entrada com canais adicionais para negar parte da perda em precisão devido ao menor tamanho de entrada espacial. Além disso, para os dois modelos ImageNet-Small/56 (com e sem remodelagem), vemos que a remodelação de insumos proporciona uma melhoria substancial de cerca de 4% de precisão. Isto é especialmente interessante considerando estes duas redes têm estruturas idênticas, exceto pela camada inicial (48 canais com remodelação de entrada versus 3 canais sem remodelação ). Finalmente, o modelo ImageNet-Large/56 atinge um impressionante 67,57%, o que está apenas 2% atrás do MobileNet de precisão total [16] usando Entrada 224×224 . Isso mostra que a estrutura proposta da CNN e o método de quantização de potências de dois pode alcançar alta classificação precisão com entrada remodelada ao usar uma CNN maior.

**Tabela 4: Avaliação do impacto da remodelação dos insumos.**

| Modelo | Precisão de remodelagem de entrada (%) | |
|---|---|---|
| ImageNet-Pequeno/224 | Não | 52,32 |
| ImageNet-Pequeno/56 | Não | 46,92 |
| ImageNet-Pequeno/56 | Sim | 50,84 |
| ImageNet-Grande/56 | Sim | 67,57 |

**Tabela 5: Comparando precisão total e potências de dois pesos para os modelos CIFAR-10 e ImageNet-Small/56.**

| | CIFAR-10 | ImageNet-Pequeno/56 |
|---|---|---|
| Precisão total | 95,28 | 57.16 |
| Poderes de dois | 92,80 | 50,84 |

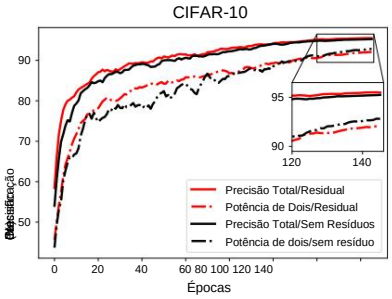



**Figura 16: Precisão de classificação em 150 épocas Modelos CIFAR-10 treinados com/sem conexões residuais e com/sem potências de duas quantizações.**

6.5.2 Impacto da quantificação de peso por potências de dois. Embora a quantização de pesos em potências de dois permita uma implementação extremamente eficiente , elas introduzem alguma perda na precisão da classificação. quando comparado com uma versão de precisão total da mesma rede . Além disso, se estes regimes forem avaliados apenas em termos de conjuntos de dados (como CIFAR-10), a redução na precisão pode ser subestimada durante a transição para conjuntos de dados mais difíceis (como ImageNet). A Tabela 5 mostra a precisão da classificação para o CIFAR-10 e Modelos ImageNet-Small/56 usando precisão total e potências de dois pesos. Vemos que embora a lacuna entre os modelos CIFAR-10 é de apenas cerca de 2,5%, a diferença para ImageNet está mais próxima de 6%. No entanto, como demonstramos na Seção 6.5.1, esta redução na classificação a precisão muitas vezes pode ser aliviada aumentando o tamanho do modelo.

6.5.3 Impacto da remoção de conexões residuais. A Figura 16 mostra o impacto das conexões residuais avaliando o CIFAR-10 estrutura de rede com e sem conexões residuais. Em ou- para garantir que não haja uma interação invisível entre o quantização de potências de dois e conexões residuais, comparamos o impacto das conexões residuais em redes com e sem quantização. Vemos que, independentemente da quantização, as redes treinados sem conexões residuais alcançam desempenho semelhante às redes treinadas com conexões residuais. Isto mostra que conexões residuais têm impacto menor na precisão da classificação para as redes de 19 camadas conforme mostrado por He et al. no original Artigo ResNet [14].

## 7. CONCLUSÃO

Neste artigo, propomos o uso de otimizações full-stack para inferência CNN precisa, de baixa latência e alta eficiência energética. Nós demonstrar que projetos que vão desde o treinamento do modelo CNN em um alto nível, até aqueles de estruturas de computação e implementação de FPGA em baixo nível podem ser otimizados simultaneamente para garantir eles se encaixam, alcançando assim um alto desempenho do sistema. Embora a otimização entre camadas (co-design) seja um conceito conhecido em literatura, o sistema relatado neste artigo é um dos mais realizações abrangentes baseadas na otimização full-stack para o projeto de implementações de aprendizagem profunda em um chip.

Descrevemos detalhes de implementação de várias técnicas de otimização, incluindo (1) mudanças de canal em vez de computacionalmente mais convoluções 3×3 caras, (2) empacotando CNNs esparsas de irregulares estrutura de dispersão para implementações eficientes em matrizes de processadores regulares, (3) quantização de ativações de dados para economia de energia com salto zero e armazenamento eficiente de dados intermediários entre camadas e (4) uso de potências de dois pesos e normalização de lote para computação eficiente.

Nosso design Seletor-Acumulador (SAC) resultante de full-stack otimização com potências de dois pesos representa uma solução extremamente maneira eficiente de implementar MAC selecionando a partir de um registrador de deslocamento em vez de realizar operações aritméticas. (Parece difícil para ter um design MAC mais eficiente, sem implementações analógicas !) Dado que MAC é a operação básica no produto escalar cálculo para combinar dados com filtros, acreditamos que nosso SAC o design é significativo.

## AGRADECIMENTOS

Este trabalho é apoiado em parte pelo Laboratório de Pesquisa da Força Aérea sob o contrato número FA8750-18-1-0112, um presente da MediaTek EUA e um Projeto de Desenvolvimento Conjunto com a TSMC.

## REFERÊNCIAS

[1] Zhaowei Cai, Xiaodong He, Jian Sun e Nuno Vasconcelos. 2017. Aprendizado profundo com baixa precisão por quantização gaussiana de meia onda. (2017). arXiv:1702.00953

[2] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan e Kailash Gopalakrishnan. 2018. PACT: Parametrizado Ativação de clipping para redes neurais quantizadas. (2018). arXiv:1805.06085

[3] François Chollet. 2016. Xception: Aprendizado profundo com separável em profundidade Convoluções. (2016). arXiv:1610.02357

[4] Matthieu Courbariaux, Yoshua Bengio e Jean-Pierre David. 2015. Binaryconnect : Treinamento de redes neurais profundas com pesos binários durante propagações. Em Avanços em sistemas de processamento de informação neural. 3123–3131.

[5] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv e Yoshua Bengio. 2016. Redes neurais binarizadas: treinando redes neurais profundas com pesos e ativações restritas a + 1 ou -1. pré-impressão arXiv arXiv:1602.02830 (2016).

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li e Li Fei-Fei. 2009. Imagnet : Um banco de dados de imagens hierárquicas em grande escala. Em Visão Computacional e Padrão Reconhecimento, 2009. CVPR 2009. Conferência IEEE sobre. IEEE, 248–255.

[7] Tim Dettmers. 2015. Aproximações de 8 bits para paralelismo em aprendizagem profunda. arXiv pré-impressão arXiv:1511.04561 (2015).

[8] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen e Olivier Temam. 2015. ShiDianNao: Mudança processamento de visão mais próximo do sensor. Em Arquitetura de Computadores ACM SIGARCH Notícias, vol. 43. ACM, 92–104.

[9] Xavier Glorot, Antoine Bordes e Yoshua Bengio. 2011. Retificador esparso profundo redes neurais. Nos Anais da Décima Quarta Conferência Internacional sobre inteligência artificial e estatísticas. 315–323.

[10] Google. [nd]. Interface funcional para a camada de normalização em lote. https://www.tensorflow.org/api_docs/python/tf/layers/batch_normalization. ([e]).

Acesso: 04/12/2018.

[11] Scott Gray, Alec Radford e Diederik Kingma. 2017. Kernels de GPU para pesos esparsos em blocos. https://s3-us-west-2.amazonaws.com/openai-assets/ blocosparse/blocksparsepaper.pdf. (2017). [On-line; acessado em 12 de janeiro de 2018].

[12] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan e Pritish Narayanan. 2015. Aprendizado profundo com precisão numérica limitada. Na Conferência Internacional sobre Aprendizado de Máquina. 1737-1746.

[13] Song Han, Huizi Mao e William J Dally. 2015. Compressão profunda: Compressão de redes neurais profundas com poda, quantização treinada e codificação huffman. Pré-impressão do arXiv arXiv:1510.00149 (2015).

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren e Jian Sun. 2016. Aprendizagem residual profunda para reconhecimento de imagens. Em Anais da conferência IEEE sobre visão computacional e reconhecimento de padrões. 770–778.

[15] Yihui He, Xiangyu Zhang e Jian Sun. 2017. Poda de canal para acelerar redes neurais muito profundas. Na Conferência Internacional sobre Visão Computacional (ICCV), Vol. 2.

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto e Hartwig Adam. 2017. Mobilenets: Redes neurais convolucionais eficientes para aplicações de visão móvel. pré-impressão arXiv arXiv :1704.04861 (2017).

[17] Qinghao Hu, Peisong Wang e Jian Cheng. 2018. Do hashing às CNNs: treinando redes BinaryWeight via hashing. Pré-impressão do arXiv arXiv:1802.02733 (2018).

[18] Gao Huang, Shichen Liu, Laurens van der Maaten e Kilian Q Weinberger. 2017. CondenseNet: um DenseNet eficiente usando convoluções de grupo aprendidas. Pré-impressão do arXiv arXiv :1711.09224 (2017).

[19] Gao Huang, Zhuang Liu, Laurens Van Der Maaten e Kilian Q Weinberger. 2017. Redes convolucionais densamente conectadas.. Em CVPR, Vol. 1.3.

[20] IEEE Press 2016. Aceleradores CNN de camada fundida. Imprensa IEEE.

[21] Sergey Ioffe e Christian Szegedy. 2015. Normalização em lote: Acelerando o treinamento profundo da rede, reduzindo a mudança interna de covariáveis. pré-impressão arXiv arXiv :1502.03167 (2015).

[22] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam e Dmitry Kalenichenko. [nd]. Quantização e treinamento de redes neurais para inferência eficiente apenas de aritmética inteira. ([n. d.]).

[23] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony , Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox e Doe Hyun Yoon. 2017. Análise de desempenho no datacenter de uma unidade de processamento tensor. Nos Anais do 44º Simpósio Internacional Anual de Arquitetura de Computadores (ISCA '17). ACM, Nova York, NY, EUA, 1–12. https://doi.org/10.1145/3079856.3080246

[24] Raghuraman Krishnamoorthi. 2018. Quantizando redes convolucionais profundas para inferência eficiente: um whitepaper. Pré-impressão do arXiv arXiv:1806.08342 (2018).

[25] Alex Krizhevsky, Vinod Nair e Geoffrey Hinton. 2014. O conjunto de dados CIFAR-10. (2014).

[26] Alex Krizhevsky, Ilya Sutskever e Geoffrey E Hinton. 2012. Classificação Imagenet com redes neurais convolucionais profundas. Em Avanços em sistemas de processamento de informação neural. 1097–1105.

[27] HT Kung. 1982. Por que arquiteturas sistólicas? Computador IEEE 15 (1982), 37–46. Problema 1.

[28] HT Kung, Bradley McDanel e Sai Qian Zhang. 2019. Empacotando redes neurais convolucionais esparsas para implementações eficientes de matriz sistólica: combinação de colunas sob otimização conjunta. 24ª Conferência Internacional ACM sobre Suporte de Arquitetura para Linguagens de Programação e Sistemas Operacionais (2019).

[29] Yann LeCun, Léon Bottou, Yoshua Bengio e Patrick Haffner. 1998. Aprendizagem baseada em gradiente aplicada ao reconhecimento de documentos. Processo. IEEE 86, 11 (1998), 2278–2324.

[30] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou e Lingli Wang. 2016. Um acelerador baseado em FPGA de alto desempenho para redes neurais convolucionais em grande escala. In Field Programmable Logic and Applications (FPL), 2016 26ª Conferência Internacional sobre. IEEE, 1–9.

[31] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu e Kwang-Ting Cheng. 2018. Rede bi-real: Melhorando o desempenho de cnns de 1 bit com capacidade de representação aprimorada e algoritmo de treinamento avançado. Pré-impressão do arXiv arXiv :1808.00278 (2018).

[32] Jian-Hao Luo, Jianxin Wu e Weiyao Lin. 2017. Thinet: Um método de remoção de nível de filtro para compactação profunda de redes neurais. pré-impressão arXiv arXiv:1707.06342 (2017).

[33] Yufei Ma, Yu Cao, Sarma Vrudhula e Jae-sun Seo. 2017. Otimizando operação de loop e fluxo de dados na aceleração fpga de redes neurais convolucionais profundas . Nos Anais do Simpósio Internacional ACM/SIGDA 2017 sobre Matrizes de Portas Programáveis em Campo. ACM, 45–54.

[34] Sharan Narang, Eric Undersander e Gregory F. Diamos. 2017. Redes Neurais Recorrentes Esparsas em Blocos. CoRR abs/1711.02782 (2017). arXiv:1711.02782 http://arxiv.org/abs/1711.02782 [35] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga e Adam Lerer. 2017. Diferenciação automática em PyTorch. (2017).

[36] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. 2016. Aprofundando a plataforma FPGA incorporada para rede neural convolucional. Nos Anais do Simpósio Internacional ACM/SIGDA 2016 sobre Matrizes de Portas Programáveis em Campo. ACM, 26–35.

[37] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon e Ali Farhadi. 2016. Xnor-net: Classificação Imagenet usando redes neurais convolucionais binárias. Na Conferência Europeia sobre Visão Computacional. Springer, 525–542.

[38] Yongming Shen, Michael Ferdman e Peter Milder. 2017. Maximizando a eficiência do acelerador CNN por meio do particionamento de recursos. Em Arquitetura de Computadores (ISCA), 2017 ACM/IEEE 44º Simpósio Internacional Anual em. IEEE, 535–547.

[39] Karen Simonyan e Andrew Zisserman. 2014. Redes convolucionais muito profundas para reconhecimento de imagens em grande escala. pré-impressão arXiv arXiv:1409.1556 (2014).

[40] Robert Tibshirani. 1996. Encolhimento de regressão e seleção via laço. Jornal da Royal Statistical Society. Série B (Metodológica) (1996), 267–288.

[41] Junsong Wang, Qiuwen Lou, Xiaofan Zhang, Chao Zhu, Yonghua Lin e Deming Chen. 2018. Fluxo de projeto de aceleração de rede neural híbrida de largura de bits extremamente baixa em FPGA incorporado. Pré-impressão do arXiv arXiv:1808.04311 (2018).

[42] Peisong Wang e Jian Cheng. 2017. Redes fatoradas de ponto fixo. Em Visão Computacional e Reconhecimento de Padrões (CVPR), 2017 IEEE Conference on. IEEE, 3966–3974.

[43] Shihao Wang, Dajiang Zhou, Xushen Han e Takeshi Yoshimura. 2017. Chain- NN: Uma arquitetura de cadeia 1D com eficiência energética para acelerar redes neurais convolucionais profundas. Em 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 1032–1037.

[44] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen e Hai Li. 2016. Aprendendo esparsidade estruturada em redes neurais profundas. Em Avanços em Sistemas de Processamento de Informação Neural. 2074–2082.

[45] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez e Kurt Keutzer. 2017. Mudança: Um FLOP Zero, Alternativa de Parâmetro Zero para Convoluções Espaciais. pré-impressão arXiv arXiv :1711.08141 (2017).

[46] Qingcheng Xiao, Yun Liang, Liqiang Lu, Shengen Yan e Yu-Wing Tai. 2017. Explorando algoritmos heterogêneos para acelerar redes neurais convolucionais profundas em FPGAs. Nos Anais da 54ª Conferência Anual de Automação de Design 2017. ACM, 62.

[47] Xilinx. 2018. Acelerando DNNs com placas aceleradoras Xilinx Alveo. Relatório técnico . Xilinx.

[48] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen e Yunji Chen. 2016. Cambricon-x: Um acelerador para redes neurais esparsas. No 49º Simpósio Internacional Anual IEEE/ACM sobre Microarquitetura. Imprensa IEEE, 20.

[49] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu e Deming Chen. 2018. DNNBuilder: uma ferramenta automatizada para construção de aceleradores de hardware DNN de alto desempenho para FPGAs. Nos Anais da Conferência Internacional sobre Design Auxiliado por Computador. ACM, 56.

[50] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu e Yurong Chen. 2017. Quantização incremental de redes: Rumo a redes sem perdas com pesos de baixa precisão. pré-impressão arXiv arXiv:1702.03044 (2017).

[51] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen e Yuheng Zou. 2016. Dorefa-net: Treinamento de redes neurais convolucionais de baixa largura de bits com gradientes de baixa largura de bits. Pré-impressão arXiv arXiv:1606.06160 (2016).

[52] Chenzhuo Zhu, Song Han, Huizi Mao e William J Dally. 2016. Ternário treinado quantização. Pré-impressão do arXiv arXiv:1612.01064 (2016).