ISEL

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

SERVIÇO DE DOCUMENTAÇÃO E PUBLICAÇÕES

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

Otimização de Redes Neuronais Convolucionais em FPGA utilizando Técnicas de Compressão

Tiago Alexandre Mateus Peres

(Licenciado em Engenharia Eletrónica e Telecomunicações e de Computadores)

Trabalho Final de Mestrado para obtenção do grau de Mestre em Engenharia de Eletrónica e Telecomunicações

Orientador:

Professor Doutor Mário Pereira Véstias

Júri:

Presidente: Professor Doutor António Couto Pinto

Vogal (Arguente): Professor Doutor Rui Policarpo Duarte

Vogal (Orientador): Professor Doutor Mário Pereira Véstias

Novembro 2018

Agradecimentos

Na realização da presente dissertação contei com o apoio de algumas pessoas que quero deixar expresso os meus agradecimentos.

Ao orientador da dissertação o Doutor Mário Pereira Véstias, agradeço pela sua total disponibilidade e pelo seu apoio que sempre demonstrou.

À minha família, namorada e a todos os amigos agradeço pela atenção e força que prestaram.

Resumo

Uma Rede Neuronal Convolucional (Convolutional Neural Network - CNN) é uma classe de redes profundas aplicada ao processamento e classificação de imagens. Uma CNN consiste em várias camadas de nós interligados entre si. A sua capacidade de classificação advém do treino das suas ligações ou pesos, obtidos por um processamento de adaptação ou aprendizagem através de um conjunto de padrões de treino. Após treinadas, conseguem classificar imagens através de um processo de inferência.

Existem atualmente vários modelos ou redes CNN de classificação de imagens, com probabilidades de erro bastante baixas, mas que são computacionalmente muito exigentes e requerem bastante memória para armazenar os pesos da rede. Como tal, são geralmente executadas em plataformas de elevado desempenho.

No entanto, a utilização deste tipo de algoritmos em sistemas embebidos tem despertado bastante interesse, pois permitirá a execução de algoritmos junto dos sistemas de aquisição de dados, permitindo decisões inteligentes junto dos sensores e evita a comunicação dos dados para um servidor central de execução de CNN.

Esta dissertação teve como objetivo a redução da complexidade das redes CNN, através de métodos de corte dos pesos e de compressão de dados, de modo a permitir a execução das CNN em FPGA de baixo custo com aplicação em sistemas embebidos sem comprometer a precisão da rede. Os métodos foram aplicados à arquitetura LiteCNN tendo-se conseguido melhorar os tempos de execução em cerca de 85% para redes com elevada quantidade de pesos nas camadas totalmente conectadas.

Palavras Chave: Rede Neuronal Convolucional; Compressão; Corte; Caffe; FPGA.

Abstract

A Convolutional Neural Network (CNN) is a class of deep networks applied to image processing and classification. A CNN consists of several layers of interconnected nodes. Their ability to classify comes from training their connections or weights, obtained by an adaptation or learning process through a set of training standards. After being trained, they can classify images through an inference process.

There are currently several CNN image sorting networks, with very low error probabilities, but which are computationally demanding and require a lot of memory to store network weights. As such, they are usually executed on high-performance platforms.

However, there has been an increased interest in the use of this type of algorithm in embedded systems, since it will allow the execution of algorithms near the data acquisition systems, allowing intelligent decisions near the sensors and avoiding the communication of data to a CNN execution central server.

The aim of this dissertation was to reduce the complexity of CNN networks, by means of weight-cutting and data-compression methods, in order to allow the execution of CNNs in low-cost FPGAs for embedded computing without compromising network accuracy. The methods were applied to the LiteCNN architecture and managed to improve execution times by about 85% for networks with higher number of weights on fully-connected layers.

Keywords: Convolutional Neural Network; Compression; Pruning; Caffe; FPGA.

ÍNDICE

Lista de Figuras

Lista de Tabelas

LISTA DE ACRÓNIMOS

CNN - Convolutional Neural Network

DNN - Deep Neural Network

FPGA - Field Programmable Gate Array

ILSVRC – ImageNet Large Scale Visual Recognition Challenge

RGB - Red Green Blue

FC - Fully-Connected

ReLU - Rectified Linear Unit

PReLU - Parametic ReLU

ELU - Exponential Linear Units

MAC - Multiply-and-Accumulate

OBD - Optimal Brain Damage

VLC - Variable Length Code

, OPS - Operations per Second

PE – Processing Element

Introdução

1

As redes neuronais profundas (*Deep Neural Networks* - DNN) são modelos para aprendizagem automática que após treinadas com um conjunto de dados conseguem classificar imagens novas não utilizadas na fase de treino.

Estas redes são frequentemente usadas em problemas que possam ser formulados em termos de classificação ou previsão, ou seja, podem ser usadas para análise estatística e modelagem de dados que necessitem de utilizar regressões não lineares. Muitas outras áreas podem beneficiar com a utilização destas redes, tais como:

- **Medicina:** as DNNs desempenham um papel cada vez mais importante para obter informações sobre doenças genéticas e para detetar outro tipo de doenças como cancro na pele, no cérebro e na mama;
- **Reconhecimento de voz:** com as DNNs houve um aumento significativo da precisão no reconhecimento de voz e na tradução em tempo real;
- **Reconhecimento de imagem**: a classificação de imagens, reconhecimento de ações, deteção e localização de objetos aumentaram de precisão significativamente com as DNNs;
- **Robótica**: aqui é utilizada principalmente na navegação visual do robô, para o controlo da estabilidade e na condução autónoma de veículos.

Uma Rede Neuronal Convolucional (*Convolutional Neural Network* - CNN) é uma classe de redes profundas aplicada ao processamento e classificação de imagens. Uma CNN consiste em vários tipos de camadas de nós interligadas entre si.

O funcionamento destas redes é inspirado nos neurónios dos seres humanos (ver Figura 1). O cérebro humano é constituído por aproximadamente 100 biliões de neurónios e cada neurónio comunica através de sinais elétricos, impulsos de curta duração, aplicados à membrana celular. Estes impulsos, que são transmitidos de neurónio para neurónio, são chamados de sinapses e estão localizadas nos ramos da célula denominados por dendrites. As dendrites são responsáveis na receção de impulsos de um ambiente para outro, ou seja, entre neurónios ou por exemplo entre neurónios e músculos. Cada neurónio recebe milhares de conexões de outros neurónios e por sua vez estão constantemente a receber sinais de cada conexão até eventualmente chegarem ao corpo da célula. No corpo celular os sinais são integrados ou somados e o sinal resultante irá dar a informação a outros neurónios através de uma fibra ramificada conhecida como axónio. Quando o sinal resultante exceder um de-

terminado limite (*threshold),* o neurónio ativa, ou caso contrário, é gerado um sinal de resposta. Há neurónios que têm capacidade para inibir, tendem a impedir as ativações, ou para estimular, tendem a promover a geração de impulsos.

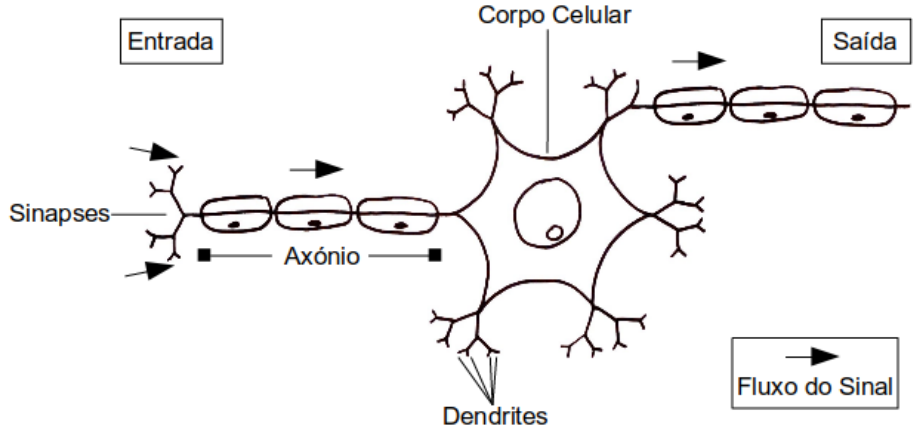

Figura 1 Neurónio humano simplificado.

As redes neuronais modelam este comportamento com um neurónio artificial, como se pode observar na Figura 2.

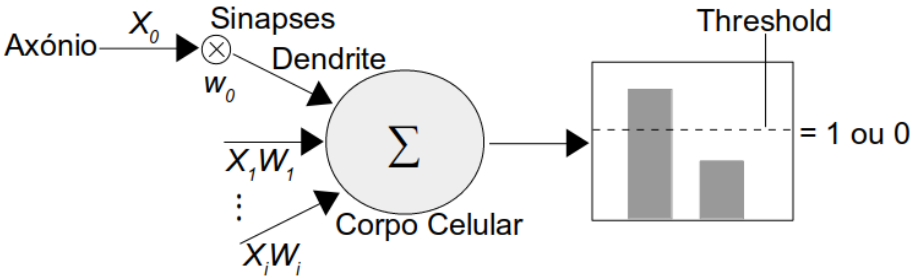

Figura 2 Neurónio artificial simplificado.

Um neurónio artificial ou nó recebe um padrão de sinais entrada (x_i) que são multiplicados por pesos (w_i). Em seguida, esses valores são somados entre si, é adicionado um fator de ajuste (b) e é gerada uma ativação (y_i). No caso mais simples, se a ativação exceder um *threshold,* o nó produz o valor lógico um, caso contrário é produzido o valor lógico 0. Em comparação com os termos do neurónio, as sinapses são normalmente chamadas de pesos e as ligações entre nós representam os axónios.

O somatório produzido no nó da Figura 2 está representado na seguinte equação:

$$y_i = \sum_i x_i w_i + b$$

A ativação de um nó, y_i, é igual ao somatório dos diferentes pesos, w_i, multiplicados pelo sinal de entrada, x_i, e na maioria das redes é adicionado um fator de ajuste, designado *bias,* b, que ajuda no treino das redes. O índice, i, corresponde ao número de pesos.

Para criar uma rede são utilizados vários neurónios artificiais todos ligados entre si e agrupados em diferentes camadas.

As primeiras redes neuronais tinham apenas três camadas. Recentemente, surgiram as redes neuronais profundas. Em relação às redes neuronais tradicionais, as DNN têm mais do que três camadas, ou seja, mais do que uma camada oculta. Uma das classes de DNN mais utilizadas é a das redes neuronais convolucionais (*Convolutional Neural Networks* - CNN) utilizadas para a classificação de padrões. São utilizadas essencialmente no processamento e análise de imagens digitais, como por exemplo no reconhecimento de caracteres ou de objetos.

As CNN têm uma capacidade de aprendizagem mais elevada e mais complexa do que as redes neuronais tradicionais. Por isso, são utilizadas principalmente para o processamento de dados visuais. Nestas aplicações, como as imagens são compostas por pixels, os sinais de entrada da camada de entrada vão ser os pixels da imagem. A presença de características, como linhas, curvas e formas dão a maior informação que as redes necessitam para compreenderem se a imagem é um objeto ou uma cena em particular.

Antes da classificação de padrões, as redes necessitam de ser treinadas, ou seja, é preciso determinar os valores dos pesos e do *bias*. Tanto no treino como na classificação das DNN, o sinal de entrada pode ser uma ou mais imagens e a saída é um vetor de resultados em que o maior resultado indica a classe a que a imagem pertence. Na fase de treino os valores dos pesos e do *bias* são ajustados conforme a precisão dos resultados, ou seja, o treino determina os pesos que maximizam os resultados com a classe correta e minimiza os resultados com a classe errada. Para aumentar o desempenho do treino é utilizado um processo chamado de *backpropagation,* que calcula como a perda é afetada por cada peso. A perda (*loss*) é a diferença entre o resultado correto e o resultado produzido pela DNN. Na Figura 3 está representado a classificação de uma imagem com o número zero.

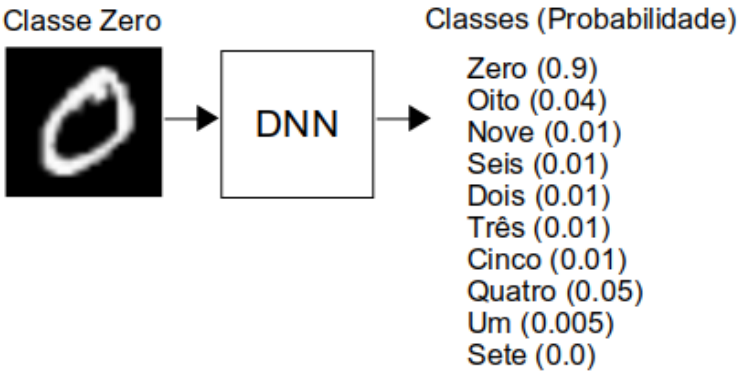

Figura 3 Exemplo da classificação do dígito zero manuscrito.

No exemplo apresentado, às dez possíveis classes de saída são associadas probabilidades de a imagem pertencer a cada uma das classes. A classe *zero* aparece com a maior probabilidade indicando que a imagem contém um zero.

1.1. Sistemas Embebidos Inteligentes

Atualmente, os sistemas embebidos inteligentes são muito utilizados, permitindo apoiar o utilizador na tomada de decisões. Estes sistemas estão presentes em todo o lado, como em casas inteligentes (televisores, frigorífico, iluminação, etc…), em veículos (controlo do ar condicionado, estacionamento e condução assistida, etc…), no mundo industrial (impressoras 3D, monitorização da temperatura, controlo de máquinas industriais, etc…) e em muitos outros locais. As CNN são vistas com grande potencial para o desenvolvimento de sistemas embebidos inteligentes. Como tal, é importante o desenvolvimento de arquiteturas embebidas que suportem a execução eficiente de CNN.

Os principais requisitos dos sistemas embebidos em geral são:

- O consumo reduzido;
- A capacidade de execução de múltiplas operações em tempo real;
- O serem estáveis e reutilizáveis com atualizações de software;
- O poderem funcionar sem a intervenção humana.

A execução de CNN em sistemas embebidos é bastante desafiadora uma vez que exige bastante capacidade computacional, memória e largura de banda de acesso à memória. Assim, são necessárias plataformas embebidas de computação que respondam a estes requisitos com elevada eficiência energética.

As FPGA (*Field Programmable Gate Array*), são vistas como uma potencial plataforma para computação embebida devido à sua elevada capacidade computacional com baixo consumo energético, quando comparado com processadores genéricos ou mesmo com um GPU embebido.

As gerações mais modernas possuem centenas de milhares de blocos lógicos configuráveis que permitem implementações de elevado desempenho e baixo consumo energético. A vantagem deste sistema, em relação a sistemas baseados em processadores ou GPUs, é a sua disponibilidade de blocos lógicos que podem ser programados e organizados de forma a serem altamente especializados para tarefas específicas, resultando no aumento da velocidade de processamento e na melhoria do consumo de energia. Contudo, a complexidade no projeto e desenvolvimento de sistemas nestes dispositivos é uma desvantagem perante os outros sistemas baseados em programação.

Quando uma CNN é desenvolvida para uma FPGA o maior problema é encontrar um mapeamento eficiente entre o modelo computacional da CNN e o modelo suportado pela FPGA. Os métodos de compressão e de redução de dados, que estudámos neste trabalho, permitem aumentar a relação desempenho/consumo das implementações de CNN em FPGA sem comprometer a precisão de classificação das redes.

Neste trabalho, é utilizada a FPGA ZYNQ XC7Z020 [1], que é uma FPGA de baixo custo e baixa densidade com aplicação em computação embebida. A ZYNQ 7Z020 é constituída por uma zona de hardware programável e por um processador dual-core ARM que permite o desenvolvimento de sistemas hardware/software de forma bastante eficiente.

1.2. Objetivos da Dissertação

O principal objetivo desta dissertação é o estudo e a aplicação de técnicas de otimização de redes CNN com vista a serem implementadas em hardware. As técnicas estudadas foram adaptadas para otimizar a sua implementação em hardware, viabilizando a implementação de redes CNN de grandes dimensões em dispositivos FPGA de baixa densidade. O estudo foi realizado sobre a arquitetura LiteCNN [2] procurando otimizar o desempenho da versão atual desta arquitetura. Os objetivos são assim:

- Estudo de técnicas de compressão e de redução do número de dados, genericamente designados neste trabalho de métodos de compressão, que permitem reduzir as necessidades computacionais e de memória das redes CNN;
- Adaptação das técnicas para serem implementadas em hardware;

- Proposta de modelos de desempenho e de área para as arquiteturas hardware propostas;
- Determinar a relação entre a precisão da rede com aplicação dos métodos de otimização e os recursos hardware ocupados.

1.3. Descrição da Dissertação

Esta dissertação encontra-se organizado em seis capítulos.

Capítulo 1 – No primeiro capítulo é realizado o enquadramento da tese nas Redes Neuronais Convolucionais e descrito os objetivos desta dissertação.

Capítulo 2 – No segundo capítulo é feita uma explicação sobre as Redes Neuronais Convolucionais, incluindo os diferentes tipos de camadas que a estrutura da CNN pode conter e como é realizado o treino destas redes. Tem incluído ainda o tipo de compressões utilizadas nas CNN.

Capítulo 3 – O terceiro capítulo descreve o estado da arte com a descrição de diversas CNN existentes e de trabalhos que implementaram CNN em FPGA com e sem compressão.

Capítulo 4 – O quarto capítulo descreve os diferentes algoritmos implementados nesta arquitetura para obter o melhor desempenho possível, a constituição da arquitetura implementada na FPGA e a forma como os algoritmos foram implementados nessa arquitetura, bem com os modelos de área e de desempenho da arquitetura.

Capítulo 5 – O quinto capítulo apresenta todos os resultados obtidos para as diversas técnicas de compressão aplicadas a várias redes e os resultados de implementação em hardware com a arquitetura LiteCNN.

Capítulo 6 – O último capítulo contém as conclusões deste projeto e propostas para trabalho futuro.

2.1. Arquitetura da Rede

Uma rede neuronal é constituída por vários neurónios artificiais todos ligados entre si e agrupados em diferentes camadas, designadas:

- Camada de entrada (*Input layer)*
- Camada oculta (*Hidden layer*)
- Camada de saída (*Ouput layer*)

Pode observar-se na Figura 4 um exemplo de uma rede neuronal, em que os nós estão representados por círculos e os pesos estão implícitos em todas as ligações entre nós.

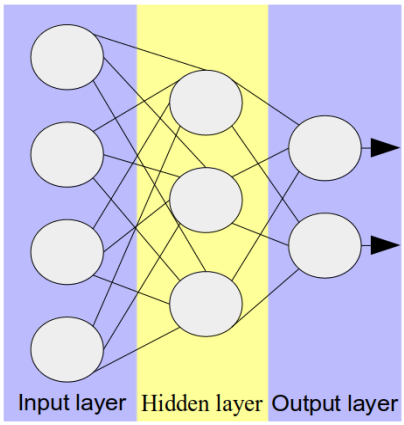

Figura 4 Topologia de uma rede neuronal.

Esta estrutura tem propagação de dados para a frente (*feedforward*) e é utilizada principalmente para classificar um padrão, que é utilizado como sinal de entrada. A camada de entrada recebe vários valores e propaga-os para os outros neurónios, que se encontram na camada oculta. Podem existir uma ou mais camadas ocultas, onde os pesos são somados e propagados até à camada de saída. A camada de saída pode ter um ou mais nós, conforme o número de classes a que o padrão pode pertencer, ou seja, quanto mais classes existirem mais nós da camada de saída irão existir. Por exemplo, uma rede para classificar números de zero a nove, irá conter dez classes, cada uma com o nome de cada número, assim sendo, a camada de saída irá conter dez nós, em que no fim irá produzir a percentagem a que poderá pertencer o padrão de entrada. Numa rede perfeita, ou seja, com cem porcento de precisão, caso o padrão de entrada seja o número zero, a rede irá produzir dez classes, em que o

número zero aparecerá com cem porcento de probabilidade e as restantes com zero porcento.

As DNN têm uma capacidade de aprendizagem mais elevada e mais complexa do que as redes neuronais de apenas três camadas, por isso, são utilizadas principalmente para o processamento de dados visuais. Nestas aplicações, como as imagens são compostas por pixels, os sinais de entrada da camada de entrada vão ser os pixels da imagem. A presença de características, como linhas, curvas e formas dão a maior informação que as redes necessitam para compreenderem se a imagem é um objeto ou uma cena em particular.

Alguns dos parâmetros mais importantes que são utilizados para o funcionamento das camadas de uma rede neuronal são:

FeedFoward: Uma rede neuronal *feedforward* típica tem uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída, ou seja, é uma rede que não tem ciclos. O mapa de características gerado em cada camada é a entrada da próxima camada.

Kernel: No contexto das CNN, os *kernels* correspondem aos filtros. São usados para detetar as especificações do padrão de entrada, dependendo dos pesos dos filtros são detetadas diferentes especificações.

Batch: As CNN podem processar os dados num conjunto de dados para aumentar o rendimento e diminuir o tempo de treino. Por exemplo, para imagens a cores, RGB três canais, com imagens 32×32, caso o *batch* seja igual a dez, significa que há uma concatenação de dez imagens, logo é processada uma matriz de $10 \times 3 \times 32 \times 32$. Durante o treino, é selecionado aleatoriamente um conjunto de dados que são alterados durante o treino de acordo com os parâmetros da rede. Caso o *batch* seja muito baixo poderá causar uma perda de precisão muito elevada, mas, caso seja muito grande, o processamento computacional pode ser bastante elevado e a máquina poderá ficar sem memória.

Depth: Corresponde ao número de filtros que são usados nas camadas das CNN. Quantos maior o número de filtros usados, mais características serão extraídas e melhor precisão a rede terá na classificação de imagens. Contudo, será necessário maior poder computacional e mais memória de armazenamento.

Stride: É o número de pixels que o filtro se desloca em cada passo ao percorrer o padrão de entrada. Por exemplo, se o *stride* for dois, o filtro sofre um deslocamento, para a direita, de dois pixels em vez de apenas um, como acontece nos casos em que não se aplica o *stride* (o *stride* é igual a um). Quanto maior o *stride*, menor serão os mapas de características produzidos. Na Figura 5 está ilustrado um exemplo com um *stride* de dois.

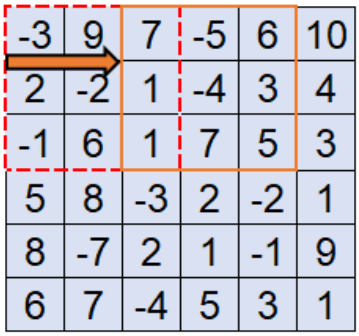

Figura 5 Exemplo da utilização de um *stride* igual a dois.

Zero-padding: É uma técnica que consiste em preencher com zeros as bordas da matriz. É utilizada no padrão de entrada para não se perder características que possam estar nas bordas da matriz. A Figura 6 representa a utilização do *zero-padding*, com um tamanho igual a um. Com esta técnica o tamanho do mapa de características gerado é maior e os pesos situados nas bordas não são descartados.

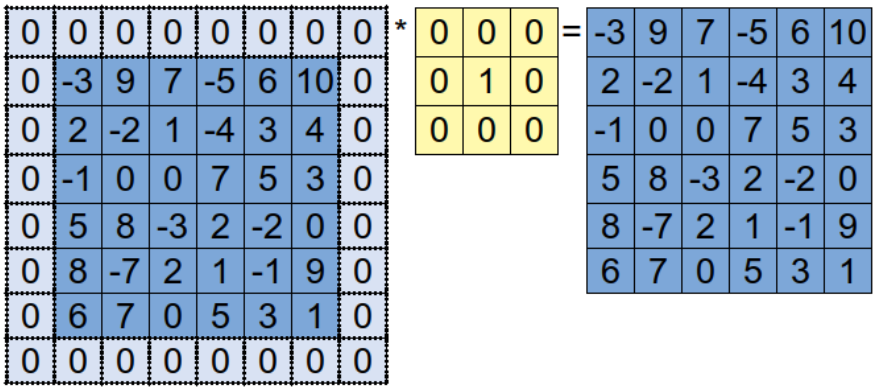

Figura 6 Exemplo da utilização *do zero-padding* de tamanho um.

Na prática, os valores dos filtros utilizados nestas camadas aprendem por conta própria durante o processo de treino e normalmente têm um tamanho de 3×3, 5×5, 7×7 ou 11×11. O tamanho do mapa de características depende do tamanho desses filtros, ou seja, quanto menor o filtro, maior será o mapa de características, com o mesmo *stride* e *zero-padding*.

As CNN são redes neuronais compostas por múltiplas camadas convolucionais, como se pode observar no exemplo da Figura 7.

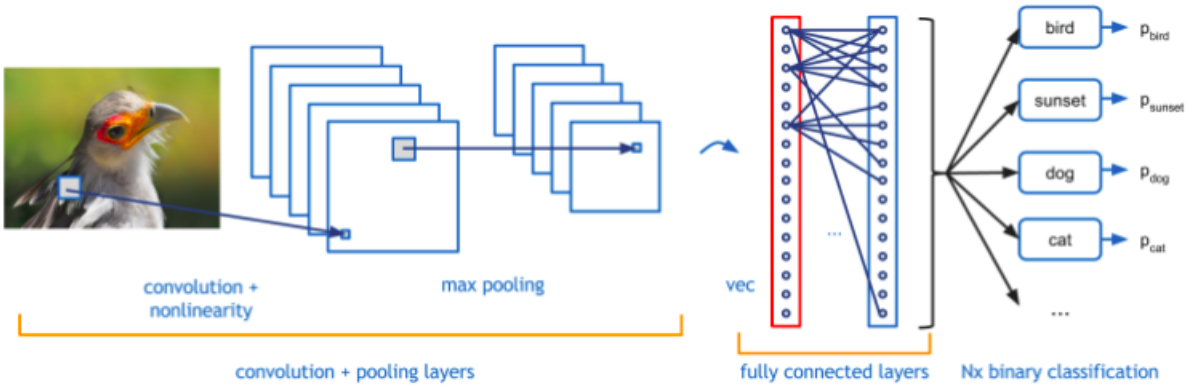

Figura 7 Exemplo de uma rede neuronal convolucional [3].

Nestas redes, cada camada produz um mapa de características (*feature map*) com a informação essencial do sinal de entrada, ou seja, no caso de uma imagem, são guardados os pixels que dão maior informação sobre a imagem a classificar, em que os padrões de entrada são estruturados como uma matriz 2-D e são chamados de canais. O mapa de características é também uma matriz 2-D que é gerado através da convolução entre um canal e um filtro deslizante 2-D, onde o filtro pode variar de camada para camada. Os resultados filtrados, que sofreram convolução, são adicionados num determinado ponto da rede e algumas redes utilizam ainda um vetor adicional, *bias*, que é adicionado às ativações.

As múltiplas camadas que são utilizadas nas CNN são do tipo:

- Camada convolucional (*convolutional layer*)
- Camada não linear (*non-linearity layer*)
- Camada de agrupamento (*pooling layer*)
- Camada totalmente conectada (*fully-connected layer - FC layer*)
- Camada de perda (*loss layer*)

Na **camada convolucional** é gerado um mapa de características através da convolução de um padrão de entrada ou canal e um filtro deslizante de tamanho $k \times k$. Como o filtro é deslizante, o cálculo da convolução inicia-se no canto superior esquerdo do padrão de entrada e percorre todo o canal com um deslocamento de N pixels (*stride*). Os filtros utilizados nesta camada permitem detetar características específicas como bordas, linhas, contornos, curvas, entre outros. Dependendo da rede, o número de padrões de entrada varia, assim como o *stride*.

A Figura 8 mostra o exemplo de uma convolução entre um padrão de entrada de $6 \times 6 \times 1$ pixels (a azul) e um filtro identidade $3 \times 3 \times 1$ (a amarelo), em que este filtro retorna o mesmo valor que o sinal de entrada. O tamanho da imagem ficou reduzido com dimensões de $4 \times 4 \times 1$ pixels. Para filtros maiores que 1×1, o mapa de características gerado é inferior às

dimensões do padrão de entrada, devido ao efeito das bordas. Normalmente, para reduzir esse efeito, o padrão de entrada é preenchido com zeros à sua volta com a técnica de *zero-padding*.

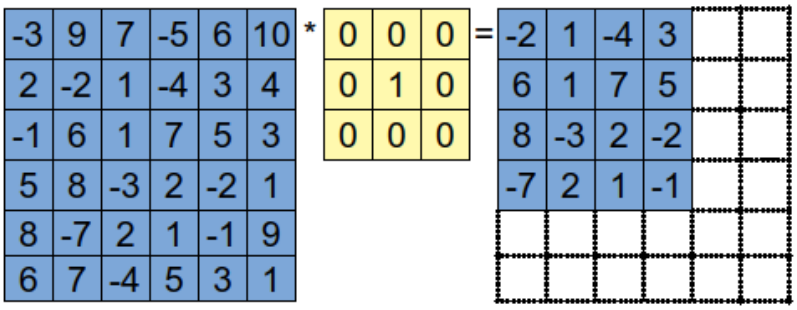

Figura 8 Exemplo da convolução entre um padrão de entrada e o filtro identidade.

A **camada não linear** aplica uma função de ativação não linear ao mapa de características produzido pela camada convolucional. As funções não lineares mais utilizadas são:

- ReLU *(Rectified Linear Unit)*

A função ReLU para valores menores que zero retorna zero e para valores maiores ou iguais a zero retorna o valor de entrada, (ver Figura 9).

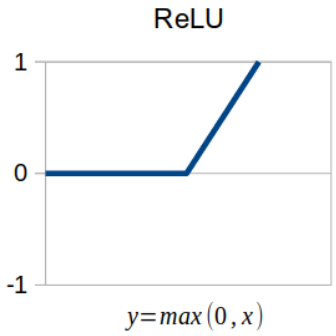

Figura 9 Função não linear (*rectified linear unit*).

- PReLU (*Parametic ReLU*)

Com a função PReLU, os valores negativos são multiplicados por uma constante α e os valores maiores ou iguais a zero mantêm-se iguais (ver Figura 10). A constante α tem normalmente valores com uma casa decimal, em que o valor típico é 0,1.

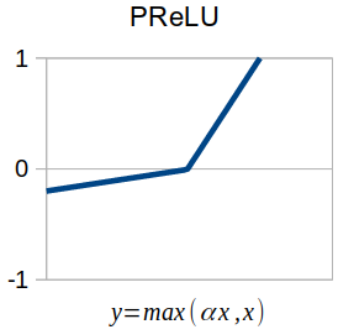

Figura 10 Função não linear (*parametic rectified linear unit* – PreLU).

- ELU *(Exponential Linear Units)*

A função ELU utiliza $\alpha(e^x - 1)$ para valores negativos de *x*. Caso contrário, para valores maiores ou iguais a zero é retornado o valor de entrada (ver Figura 11). Também nesta função, a constante α tem um valor típico igual a 0,1.

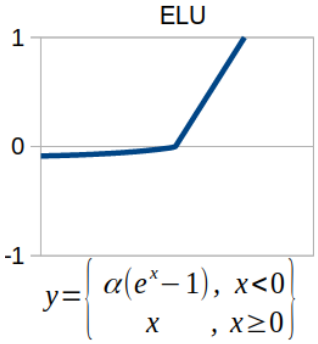

Figura 11 Função não linear (*exponential linear unit*).

- *Sigmoid*

A função *sigmoid* é uma função não-linear mais tradicional em relação às restantes. No entanto, ainda é bastante utilizada. É aplicada a função representada na Figura 12 a todos os valores de x. Esta função resulta em valores entre zero e um.

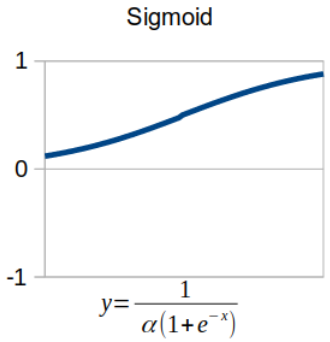

Figura 12 Função não linear *sigmoid*.

O tempo de treino é maior quanto maior for a complexidade das funções, apesar de aumentar a precisão, o que para grandes conjuntos de dados pode influenciar na escolha da função a utilizar. A rapidez de treino pode ser preferida em relação à precisão, pois para grandes CNN o tempo de treino é muito elevado. Por isso, podem ser escolhidas funções menos complexas. A função ReLU é a função menos complexa em relação às restantes funções descritas e, por isso, é normalmente a mais utilizada. A Figura 13 ilustra a aplicação da função ReLU no mapa de características gerado na Figura 8.

Figura 13 Exemplo da aplicação da ReLU no mapa de características.

Como se observa na Figura 13, os valores negativos no mapa de entrada passam a zero no mapa de saída, enquanto que os positivos se mantêm iguais.

A **camada de agrupamento** simplifica a computação das próximas camadas através da redução do tamanho do mapa de características. Essa redução ocorre utilizando um filtro deslizante que percorre todo o mapa de características, em que o maior valor (*max-pooling*) ou o valor médio (*avg-pooling*) dessa janela é guardado num novo mapa de características. Esta simplificação é feita porque não é necessário saber a posição exata de um objeto, mas sim o que o objeto representa. Por exemplo, se a imagem for um carro, não importa se as rodas estão ligeiramente deslocadas da posição original. Um exemplo desta utilização está representado na Figura 14, em que uma janela deslizante de 2×2 com um deslocamento de dois pixels (é a implementação mais comum) percorre todo o mapa de características da Figura 13. Esta camada é utilizada entre camadas convolucionais e entre camadas totalmente conectadas.

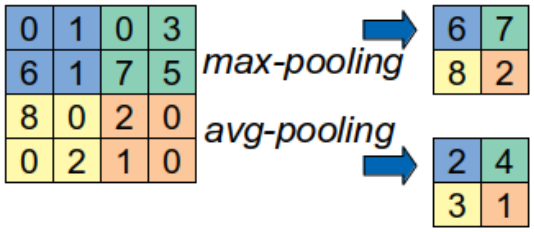

Figura 14 Exemplo da aplicação das funções *max-pooling* e *avg-pooling* no mapa de características.

A função *max-pooling* é dada por:

$$y = \max(x)$$

E a função *average-pooling* é dada por:

$$y = \frac{1}{A \times L} \sum_{i=1}^{A} \sum_{j=1}^{L} x[i][j]$$

A **camada totalmente conectada** (*Fully Connected* - FC) é uma camada em que todos os nós estão interligados entre si (ver exemplo na Figura 15).

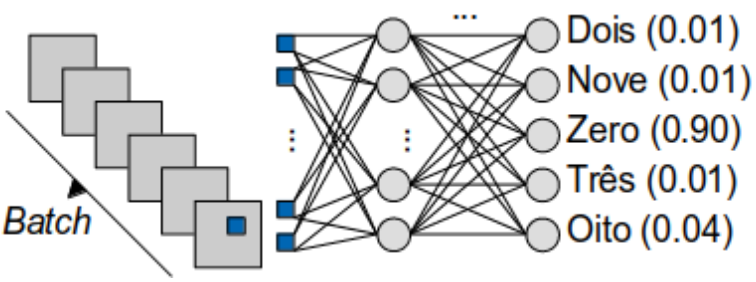

Figura 15 Exemplo da arquitetura da camada totalmente conectada.

A partir de uma camada totalmente conectada obtém-se um mapa de características com a extração das principais características do padrão de entrada. Caso não seja a última camada totalmente conectada, podem ser seguidas de uma camada não linear. Do ponto de vista matemático, as camadas FC calculam a soma de produtos entre os pesos e os padrões de entrada, dada pela seguinte equação:

$$y_i = \sum_i w_i \times x_i + b$$

Como anteriormente descrito, a equação é igual à equação utilizada nas DNN e o número de nós da última camada totalmente conectada é igual ao número de classes a serem reconhecidas.

Na fase de inferência, ou seja, na classificação, a camada FC é geralmente a última camada de uma CNN. Desta forma, é possível aplicar uma operação de normalização para fornecer a probabilidade da classificação prevista. A operação mais comum é a função *Softmax,* σ*,* que é dada pela seguinte função:

$$\sigma_i = \frac{y_i}{\sum_{n=1}^{N} e^{y_n}}$$

Esta operação calcula a distribuição de probabilidade para os N valores da saída, ou seja, calcula o valor da probabilidade a que pertence uma determinada classe, impondo assim a que os N valores de saída pertençam a um intervalo entre zero e um.

Por fim, caso seja a última camada totalmente conectada da CNN, esta está ligada à camada de perda, que se descreve de seguida.

Normalmente, uma rede CNN termina com uma **camada de perda**. Esta é utilizada durante o treino para impedir a existência de redundâncias nos pesos, o que evita a sobreposição em grandes CNN. A camada de perda controla o ajuste dos pesos da rede, sendo que, antes do treino, os pesos e o *bias* são iniciados com valores aleatórios. De seguida, durante o treino, esta camada verifica o valor dos pesos da camada totalmente conectada em relação aos valores reais, com o objetivo de minimizar a diferença entre os valores da classe resultante e os valores da classe correta. Durante o treino, a operação *softmax* é aplicada nesta camada, em vez de ser na camada totalmente conectada.

2.2. Métodos de Otimização

As redes neuronais convolucionais projetadas têm um grande número de pesos para conseguirem uma precisão elevada na classificação de imagens. Contudo, é necessário um poder computacional elevado e muita capacidade de memória, tanto na fase de treino como na fase de inferência. Na fase de treino, o consumo de energia é bastante elevado, pois é necessária uma grande utilização de características. Na fase de teste, há um elevado número de pesos treinados que causa um grande custo no armazenamento de memória e no seu processamento. Por exemplo, o modelo da rede AlexNet [13] ocupa mais de 200 MB em pesos. Estes problemas dificultam a implementação de redes neuronais em sistemas com pouca memória e baixa capacidade de processamento, como os sistemas embebidos.

Existe um conjunto de técnicas que permitem reduzir o consumo de energia e melhorar os tempos de inferência com a minimização da degradação da precisão da rede. As principais técnicas de otimização existentes estão agrupadas em duas categorias:

- Quantização linear
- Quantização não linear

A quantização linear reduz a precisão das operações e dos operandos através da redução do número de bits por peso e por ativação e a simplificação das operações com a utilização de vírgula fixa em vez de vírgula flutuante.

A quantização linear é a redução da precisão, convertendo, tipicamente, valores e operações de vírgula flutuante para vírgula fixa. Um número de 32 bits com vírgula flutuante está representado na Figura 16, em que a mantissa é representada por 23 bits e o expoente por 8 bits.

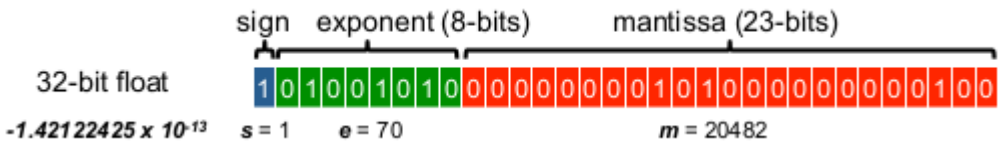

Figura 16 Exemplo da vírgula flutuante de 32 bits [4].

Ao reduzir a precisão, o tamanho dos operadores aritméticos é mais baixo, o que provoca uma redução no consumo de energia, no espaço de armazenamento necessário e na largura de banda de acesso à memória.

Em redes de grande dimensão existe, em geral, redundância nos pesos. Por exemplo, no cálculo das convoluções para as diferentes camadas muitas vezes resultam em mapas de características correlacionados entre si.

O objetivo dos métodos de quantização não linear é conseguir reduzir o número de pesos redundantes, com a consequente redução do número de operações, e/ou comprimir os pesos de modo a reduzir o tamanho do modelo, com a consequente redução das necessidades de memória e de requisitos de largura de banda de acesso à memória, sem comprometer a precisão. Existem várias técnicas para a redução do número de operações e do tamanho dos modelos. Algumas das técnicas mais utilizadas são o *pruning, a* codificação de Huffman e a divisão dos filtros, que descrevemos de seguida.

Normalmente, as redes têm parâmetros em excesso (pesos redundantes) que facilitam o treino. Esses pesos podem ser removidos, sendo este processo chamado de corte (*pruning*). O *pruning* é baseado numa técnica chamada *Optimal Brain Damage* (OBD) que foi desenvolvida em 1989 por Yann Le Cun et al [5]. A técnica considera uma equação para medir objetivamente a saliência de cada peso para encontrar os pesos com baixa saliência. Os pesos de baixa saliência são pesos que têm menor efeito na precisão no treino e quanto mais próximos de zero, mais baixa é a sua saliência. Em 2015 [4], obtiveram-se resultados sem perda de precisão, utilizando a energia gasta pelos pesos como métrica, ou seja, a energia gasta é associada diretamente aos pesos e os pesos são cortados com base na maior energia gasta. Contudo, os métodos de avaliação de energia usados para estimar a energia de uma CNN são muito pesados computacionalmente e difíceis de utilizar em redes de grande escala. Uma métrica de corte simples de usar é através da amplitude dos pesos, ou seja, os valores dos pesos abaixo de um dado limite são removidos e os restantes são aperfeiçoados para a rede aprender os pesos finais. Com uma grande percentagem de corte nos pesos a existência de pesos sozinhos aumenta, o que não é bom para a gestão de energia.

Com o *pruning* é possível uma redução da quantidade de pesos superior a 50% com uma redução na precisão muito pouco significativa. Muitas vezes, para se obter maior eficácia na

redução do tamanho das redes, o *pruning* é utilizado juntamente com outras técnicas de compressão.

A codificação de Huffman é um método de compressão sem perdas que usa as probabilidades de ocorrência dos símbolos a comprimir, determinando um código binário para cada símbolo. Os códigos são de comprimento variável (*variable-length code - vlc*).

Em 1952, o algoritmo foi desenvolvido por David A. Huffman e pode ser composto por cinco etapas:

1. Os símbolos são colocados por ordem decrescente de probabilidade.
2. São escolhidos dois símbolos de menor probabilidade e são agrupados num símbolo com probabilidade igual à soma destes dois.
3. É inserido um novo símbolo na lista ordenada.
4. Repete-se o segundo ponto até existir apenas um símbolo.
5. São atribuídos os códigos binários partindo do último símbolo criado.

Este algoritmo pode ser utilizado em CNN, principalmente depois de se aplicar *pruning*, pois, em muitos dos casos, os pesos sofrem *pruning* sem serem removidos, apenas colocados a zero. Por vezes, os pesos não podem ser removidos, porque é necessário saber a posição de todos os pesos. Assim, como a probabilidade de o peso ser zero é muito superior à probabilidade de o peso ser qualquer outro valor, pode ser aplicado um código binário com menos bits ao símbolo zero, o que irá provocar uma compressão mais eficiente principalmente para as grandes CNN.

Na Figura 17 está ilustrado um exemplo de uma aplicação da codificação de Huffman. A probabilidade de ocorrer o símbolo zero é muito superior às restantes. Assim sendo, o símbolo zero fica codificado com o bit um e, nos piores casos, para os símbolos com 0,1 de probabilidade, ficam codificados com um código de tamanho de seis bits.

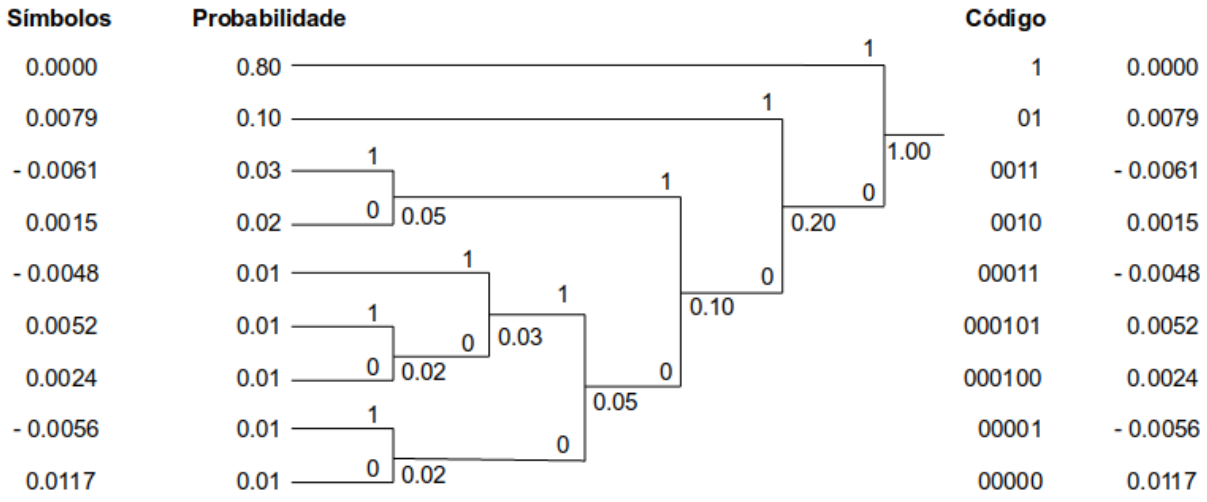

Figura 17 Exemplo da implementação da codificação de Huffman.

Uma outra técnica de redução do número de pesos é a de divisão de filtros**.** Esta técnica tem como objetivo reduzir o tamanho do mapa de características e diminuir o tempo de treino da rede. Para atingir este objetivo, o tamanho dos filtros das camadas convolucionais são decompostos em dois filtros mais pequenos. Há dois tipos de decomposição dos filtros:

- Decompor em dois filtros quadrados mais pequenos que o original.
- Decompor num filtro vertical, $k \times 1$, e num filtro horizontal, $1 \times k$.

Utilizando esta técnica, as redes ficam com o dobro das camadas convolucionais, pois, para dividir os filtros em dois é criada uma camada convolucional igual à camada a que o filtro pertence e é apenas alterado o tamanho dos filtros.

Na Figura 18 é dado um exemplo para um filtro 5×5, que é dividido em dois filtros 3×3 (em cima), ou dividido em dois filtros 5×1 e 1×5 (em baixo).

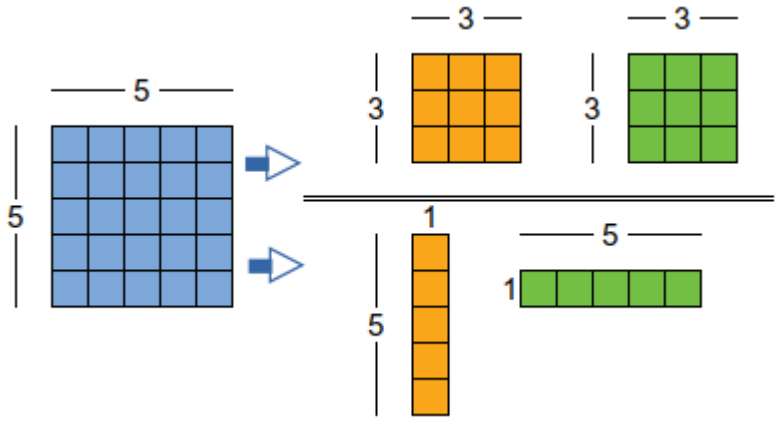

Figura 18 Exemplo da técnica de divisão de filtros.

Neste projeto são consideradas as técnicas de *pruning,* de codificação de Huffman e de divisão dos filtros que são otimizadas para serem implementadas em FPGA. A codificação de Huffman não foi implementada em FPGA.

Estado da Arte

3

Existem várias ferramentas para o treino e para a classificação de redes neuronais convolucionais, sendo as principais a Neural Network Toolbox para o MATLAB [6], o Theano [7], o TensorFlow [8] e o Caffe [9]. A maioria dos modelos de treino e de classificação de imagens usam a ImageNet [10]. A ImageNet é uma base de dados de imagens a cores de 256×256 pixels organizadas de acordo com a hierarquia do WordNet [11], em que cada nó da hierarquia é representado por centenas ou milhares de imagens. O WordNet é uma base de dados lexicais em inglês. Os substantivos, verbos, adjetivos e advérbios são agrupados por um conjunto de sinónimos cognitivos, cada um expressando um conceito distinto. Estas bases de dados são livres e publicamente disponíveis, sendo muito importantes para o treino e aprendizagem de algoritmos. A ImageNet tem mais de 22 mil categorias e cerca de 15 milhões de imagens catalogadas. As imagens são coletadas da *web* e rotuladas pela *Amazon Mechanical Turk* [12].

A *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) [10] é uma competição anual que avalia algoritmos de deteção de objetos e classificação de imagens em larga escala, onde é utilizada a base de dados da ImageNet. O concurso utiliza cerca de 1,2 milhões de imagens distribuídas por mil classes e a taxa de erro do *top-5* para fins de classificação. A imagem é considerada classificada corretamente se a classe a que a imagem pertencer estiver entre as cinco classes previstas. A taxa de erro *top-5* compara as cinco classes previstas com a imagem rotulada e a taxa de erro *top-1* compara a classe prevista com a imagem rotulada.

Muitos dos modelos de CNN mais conhecidos concorreram nesta competição como a AlexNet [13], a VGG-16 [14], a GoogleNet [15] e a ResNet [16]. Na Tabela 1 pode observar-se a comparação dos melhores modelos em termos de probabilidade de erro segundo a ImageNet, em que a melhor precisão é a da ResNet. Em termos de número de pesos e de MACs (*multiply-and-accumulates*), a VGG-16 é bastante superior às restantes.

	AlexNet	VGG-16	GoogleNet	ResNet-50
Top-1 erro [%]	37.5	24.4	-	20.74
Top-5 erro [%]	17.0	7.32	6.67	5.25
Total de Pesos	61 M	138 M	7 M	25.5 M
Total de MACs	724 M	15.5 G	1.43 G	3.9 G

Tabela 1 - Percentagem de erro *top-1* e *top-5* com base na ImageNet para diferentes modelos [4].

Outras redes bastante utilizadas são a LeNet [17] e a CIFAR-10 [18], mas que utilizam a sua própria base de dados. A LeNet utiliza a base de dados da MNIST que é uma base de dados de dígitos manuscritos de 28×28 pixels, com 60 mil imagens de treino e 10 mil imagens de teste. A CIFAR-10 é uma base de dados com imagens a cores de 32×32 pixels e com dez classes (avião, automóvel, pássaro, gato, veado, cão, sapo, cavalo, barco e camião). Tem 50 mil imagens para treino e 10 mil de teste. A Figura 19 ilustra as três bases de dados mais utilizadas pelas CNN.

Figura 19 Base de dados da MNIST (à esquerda), CIFAR-10 (ao centro) e da ImageNet (à direita) [4].

As CNN diferem entre si pelo número e tipo de camadas, pelo número de pesos e de como estes pesos são aplicados em cada camada. Para se perceber melhor as diferenças entre redes, vão ser descritas algumas das CNN mais conhecidas e utilizadas neste projeto. Para cada uma das camadas é apresentado o tamanho do padrão de entrada, o tamanho do filtro, o deslocamento do filtro, tamanho do *zero-padding* (*pad*), tamanho da saída, o número total de pesos e o número de operações MAC (*Multiply and Accumulate* - MAC). O número de pesos por camada é calculado através da seguinte equação:

$$Pesos = k_{size} \times n_{outputs}$$

E o número de MACs através de:

$$MACs = k_{size} \times n_{outputs} \times Input_{size}$$

- k_{size}, tamanho do filtro
- $n_{outputs}$, número de canais do filtro
- $Input_{size}$, tamanho do padrão de entrada

3.1. Redes CNN

A LeNet foi uma das primeiras abordagens de CNN, introduzidas em 1989 [17]. Esta rede foi desenvolvida para reconhecer dígitos manuscritos em imagens com tons de cinza de tamanho 28×28. Esta rede contém duas camadas convolucionais e duas camadas totalmente conectadas. Nas camadas convolucionais, os filtros têm tamanho de 5×5 com um *stride* igual a um e são seguidas por uma camada de agrupamento, que utilizam filtros 2×2 com *max-pooling* e um *stride* de dois. Na primeira camada totalmente conectada é aplicada a função ReLU, enquanto que na segunda é aplicada a função *softmax*. É uma rede composta por 60 mil pesos e 341 mil MAC. A Tabela 2 descreve a arquitetura LeNet.

Camada	Entrada	Filtro	*Stride*	*Pad*	Saída
Conv1	28 x 28 x 1	5 x 5	1	0	24 x 24 x 20
Max Pool1	24 x 24 x 20	2 x 2	2	0	12 x 12 x 20
Conv2	12 x 12 x 20	5 x 5	1	0	8 x 8 x 50
Max Pool2	8 x 8 x 50	2 x 2	2	0	4 x 4 x 50
FC1+ReLU	4 x 4 x 50	1 x 1	1	0	4 x 4 x 500
FC2+Softmax	4 x 4 x 500	1 x 1	1	0	4 x 4 x 10

Tabela 2 - Modelo da rede LeNet.

A LeNet é uma rede de pequena dimensão, comparada com as seguintes, mas suficiente para a classificação dos dígitos com precisões próximas dos 99%.

A CIFAR-10 é bastante utilizada, sobretudo no estudo e no teste de novos algoritmos, por usar imagens a cores de tamanho reduzido, 32×32 (ver Tabela 3), o que em termos computacionais é mais vantajoso.

Camada	Entrada	Filtro	*Stride*	*Pad*	Saída
Conv1	32 x 32 x 3	5 x 5	1	2	32 x 32 x 32
Max Pool1+ReLU	32 x 32 x 32	3 x 3	2	0	15 x 15 x 32
Conv2+ReLU	15 x 15 x 32	5 x 5	1	2	7 x 7 x 32
Avg Pool2	7 x 7 x 32	3 x 3	2	0	7 x 7 x 32
Conv3+ReLU	7 x 7 x 32	5 x 5	1	2	7 x 7 x 64
Avg Pool3	7 x 7 x 64	3 x 3	2	0	3 x 3 x 64
FC1+Softmax	3 x 3 x 64	1 x 1	1	0	3 x 3 x 10

Tabela 3 - Modelo da rede CIFAR-10.

A CIFAR-10 tem três camadas convolucionais e uma camada totalmente conectada. Todas as camadas convolucionais utilizam filtros 5×5 com um stride igual a um e um *zero-padding* igual a dois, sendo que, apenas nas duas últimas camadas é empregue a função ReLU. É ainda composta por uma max-pooling e duas avg-pooling com filtros de tamanho 3×3 e um stride de dois. A camada totalmente conectada sofre a operação softmax. A rede Cifar-10 consegue precisões de aproximadamente 72% na classificação de imagens.

A AlexNet ganhou em 2012 o concurso ILSVRC e teve um grande impacto no crescimento das CNN. A rede é composta por cinco camadas convolucionais e três camadas totalmente conectadas. Nas camadas convolucionais são aplicados filtros de 11×11, na primeira camada, 5×5, na segunda camada, e 3×3, nas últimas três camadas. Após cada camada convolucional e de cada camada totalmente conectada é aplicada a função ReLU. Existem ainda três camadas de agrupamento *max-pooling* com filtros de tamanho 3×3. A arquitetura da AlexNet está representada na Figura 20 e descrita na Tabela 4.

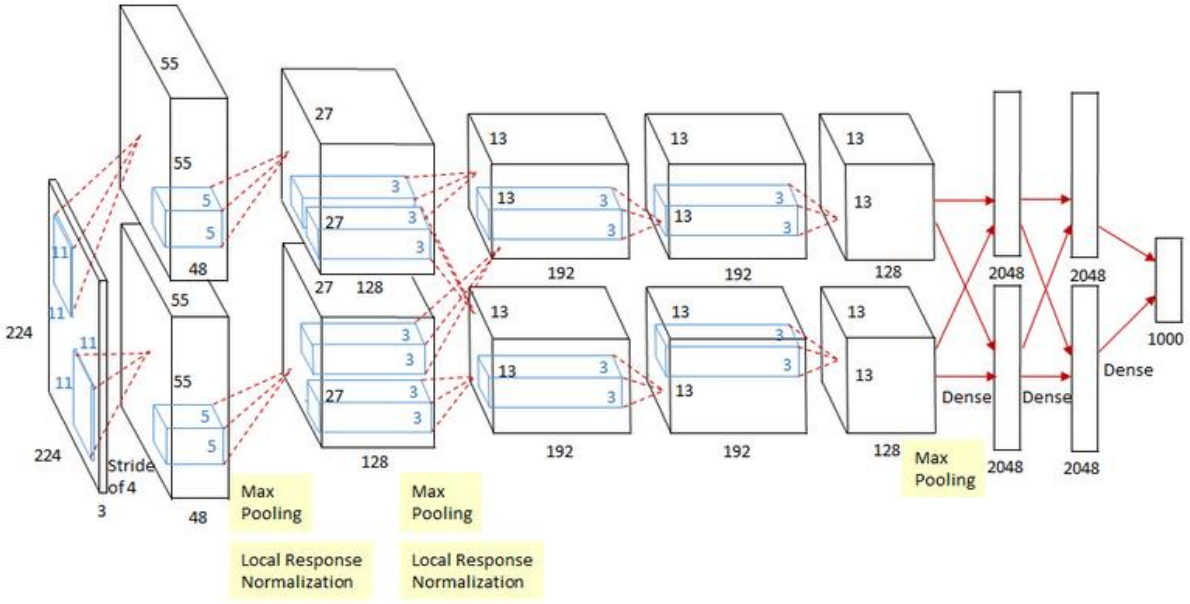

Figura 20 Arquitetura da rede AlexNet [13].

Em relação à LeNet e à CIFAR-10, o número de pesos e de operações é muito superior, com 61 milhões de pesos e 724 milhões de MAC [13] para processar imagens a cores de 227×227 da ImageNet.

Na competição a rede conseguiu uma taxa de erro *top-1* de 37,5% e uma taxa de erro *top-5* de 17,0%.

Camada	Entrada	Filtro	*Stride*	*Pad*	Saída
Conv1+ReLU	227 x 227 x 3	11 x 11	4	0	55 x 55 x 96

Max Pool1	55 x 55 x 96	3 x 3	2	0	27 x 27 x 96
Conv2+ReLU	55 x 55 x 96	5 x 5	1	2	27 x 27 x 256
Max Pool2	27 x 27 x 256	3 x 3	2	0	13 x 13 x 256
Conv3+ReLU	13 x 13 x 256	3 x 3	2	0	13 x 13 x 384
Conv4+ReLU	13 x 13 x 384	3 x 3	1	1	13 x 13 x 384
Conv5+ReLU	13 x 13 x 384	3 x 3	1	1	13 x 13 x 256
Max Pool3	13 x 13 x 256	3 x 3	2	0	6 x 6 x 256
FC1+ReLU	6 x 6 x 256	6 x 6	1	0	1 x 1 x 4096
FC2+ReLU	1 x 1 x 4096	1 x 1	1	0	1 x 1 x 4096
FC3+Softmax	1 x 1 x 4096	1 x 1	1	0	1 x 1 x 4096

Tabela 4 - Modelo da rede AlexNet.

A GoogleNet foi a vencedora do concurso ILSVRC em 2014 e é composta por vinte e duas camadas, mas com uma redução no número de pesos para 7 milhões. A GoogleNet é também mais complexa, pois para diminuir o número de pesos foi utilizada um tipo de arquitetura designada *inception*, representada na Figura 21. Esta arquitetura é aplicada nas camadas convolucionais para reduzir o tamanho dos filtros. As camadas convolucionais são divididas em três camadas que diferem no tamanho dos filtros (1×1, 3×3 e 5×5) e no número de canais. É utilizado paralelismo entre as três camadas convolucionais mais uma camada de agrupamento de 3×3, enquanto que anteriormente havia apenas uma conexão em série. Os filtros 1×1 servem para reduzir o número de canais para cada filtro, diminuindo assim o número de pesos. No fim, as quatro camadas são concatenadas.

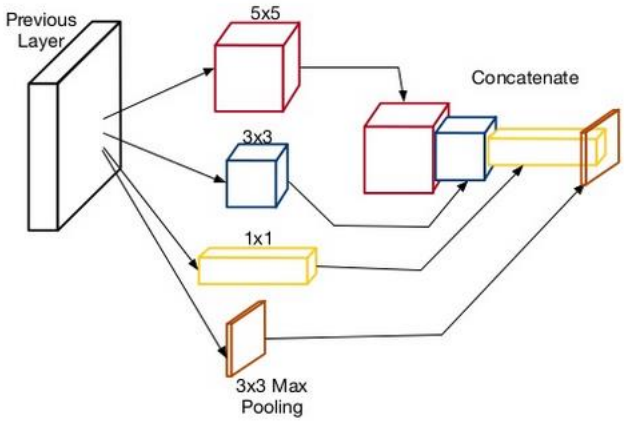

Figura 21 Modulo *inception* da GoogleNet [15].

As vinte e duas camadas são compostas por três camadas convolucionais (a *Conv2* tem duas camadas convolucionais), seguidas por nove camadas *inception* (cada uma com duas camadas convolucionais) e uma camada FC. São utilizados filtros de tamanho 7×7 na primeira camada convolucional com um *stride* de dois e *zero-padding* de três, e na camada de

agrupamento *Pool5*, com o *stride* igual a um, sem *zero-padding* e com a função *avg-pooling.* Os restantes filtros das camadas de agrupamento têm tamanho 3×3 com um *stride* de dois, sem *zero-padding* e é aplicada a função *max-pooling* (ver Tabela 5).

Camada	Entrada	Filtro	*Stride*	*Pad*	Saída
Conv1+ReLU	227 x 227 x 3	7 x 7	2	3	112 x 112 x 64
Max Pool1	112 x 112 x 64	3 x 3	2	0	56 x 56 x 64
Conv2+ReLU	56 x 56 x 64	3 x 3	1	1	56 x 56 x 192
Max Pool2	56 x 56 x 192	3 x 3	2	0	28 x 28 x 192
Inception3	28 x 28 x 192	-	-	-	28 x 28 x 480
Max Pool3	28 x 28 x 480	3 x 3	2	0	14 x 14 x 480
Inception4	14 x 14 x 480	-	-	-	14 x 14 x 832
Max Pool4	14 x 14 x 832	3 x 3	2	0	7 x 7 x 832
Inception5	7 x 7 x 832	-	-	-	7 x 7 x 1024
Avg Pool5	7 x 7 x 1024	7 x 7	1	0	1 x 1 x 1024
Loss	1 x 1 x 1024	-	-	-	1 x 1 x 1024
FC1+Softmax	1 x 1 x 1024	1 x 1	1	0	1 x 1 x 1000

Tabela 5 - Modelo da rede GoogleNet.

A Tabela 6 mostra o tamanho das entradas e das saídas das camadas *inception* da Tabela 5. A camada *inception3* é composta por duas camadas, a *inception4* por cinco camadas e a *inception5* por duas camadas. Num total de nove camadas *inception* para reduzir o número de pesos e conseguir ainda uma melhor precisão, em relação à AlexNet.

Camada	Entrada	Filtros			Saída
Inception3a	28 x 28 x 192	1 x 1	3 x 3	5 x 5	28 x 28 x 256
Inception3b	28 x 28 x 256	1 x 1	3 x 3	5 x 5	28 x 28 x 480
Inception4a	14 x 14 x 480	1 x 1	3 x 3	5 x 5	14 x 14 x 512
Inception4b	14 x 14 x 512	1 x 1	3 x 3	5 x 5	14 x 14 x 512
Inception4c	14 x 14 x 512	1 x 1	3 x 3	5 x 5	14 x 14 x 512
Inception4d	14 x 14 x 512	1 x 1	3 x 3	5 x 5	14 x 14 x 528
Inception4e	14 x 14 x 528	1 x 1	3 x 3	5 x 5	14 x 14 x 832
Inception5a	7 x 7 x 832	1 x 1	3 x 3	5 x 5	7 x 7 x 832
Inception5b	7 x 7 x 832	1 x 1	3 x 3	5 x 5	7 x 7 x 1024

Tabela 6 - Camadas *inception* do modelo GoogleNet.

São necessários um total de 7 milhões de pesos e 1,43 mil milhões de MACs para processar uma imagem a cores de 224×224. Na competição a GoogleNet conseguiu uma taxa de

erro *top-5* de 6,67%, melhor que a conseguida com a AlexNet, com menos pesos, mas com mais operações.

A rede VGG-16 ficou em segundo lugar no concurso ILSVRC de 2014. Tem dezasseis camadas, treze convolucionais e três totalmente conectadas. Para diminuir o número de pesos, os filtros 5×5 são divididos em múltiplos filtros 3×3. Todos os filtros das camadas convolucionais têm o mesmo tamanho de 3×3. Os filtros 1×1 é uma maneira de aumentar a não-linearidade sem afetar o número de canais de entrada e de saída, assim há uma não-linearidade adicional introduzida pela função de retificação.

Todas as camadas convolucionais têm um *stride* e um *zero-padding* igual a um, enquanto que todas as camadas de agrupamento têm um *stride* de dois e um *zero-padding* de um com *max-pooling*, como se pode observar na Tabela 7.

Camada	Entrada	Filtro	*Stride*	*Pad*	Saída
Conv1	224 x 224 x 3	3 x 3	1	1	224 x 224 x 64
Conv2	224 x 224 x 64	3 x 3	1	1	224 x 224 x 64
Max Pool1	224 x 224 x 64	2 x 2	2	1	113 x 113 x 64
Conv3	113 x 113 x 64	3 x 3	1	1	113 x 113 x 128
Conv4	113 x 113 x 128	3 x 3	1	1	113 x 113 x 128
Max Pool2	113 x 113 x 128	2 x 2	2	1	57 x 57 x 128
Conv5	57 x 57 x 128	3 x 3	1	1	57 x 57 x 256
Conv6	57 x 57 x 256	3 x 3	1	1	57 x 57 x 256
Conv7	57 x 57 x 256	1 x 1	1	1	57 x 57 x 256
Max Pool3	57 x 57 x 256	2 x 2	2	1	29 x 29 x 256
Conv8	29 x 29 x 256	3 x 3	1	1	29 x 29 x 512
Conv9	29 x 29 x 512	3 x 3	1	1	29 x 29 x 512
Conv10	29 x 29 x 512	1 x 1	1	1	29 x 29 x 512
Max Pool4	29 x 29 x 512	2 x 2	2	1	15 x 15 x 512
Conv11	15 x 15 x 512	3 x 3	1	1	15 x 15 x 512
Conv12	15 x 15 x 512	3 x 3	1	1	15 x 15 x 512
Conv13	15 x 15 x 512	1 x 1	1	1	15 x 15 x 512
Max Pool5	15 x 15 x 512	2 x 2	2	1	8 x 8 x 512
FC1	8 x 8 x 512	1 x 1	1	0	7 x 7 x 4096
FC2	7 x 7 x 4096	1 x 1	1	0	6 x 6 x 4096
FC3+SoftMax	6 x 6 x 4096	1 x 1	1	0	1 x 1 x 1000

Tabela 7 - Modelo da rede VGG-16.

No total são necessários 138 milhões de pesos e 15,5 mil milhões MAC para processar cada imagem a cores de 224×224. Na competição ILSVRC14 a VGG-16 conseguiu uma taxa de erro *top-1* de 7,32% e *top-5* de 24,4%.

A ResNet é uma rede que se baseia em módulos de microarquitetura chamados de arquiteturas de rede em rede (*Network-in-Network,* NIN). Foi introduzida pela primeira vez por Kaiming He et al. [16]. Esta arquitetura foi desenvolvida para ultrapassar um problema das redes profundas, em que o aumento da profundidade leva a uma degradação da precisão até saturar. A Figura 22 mostra um exemplo do erro de treino (à esquerda) e do erro de teste (à direita) para a CIFAR-10 com 20 (amarelo) e 56 (vermelho) camadas, que comprova quanto mais profunda é a rede maior é o erro.

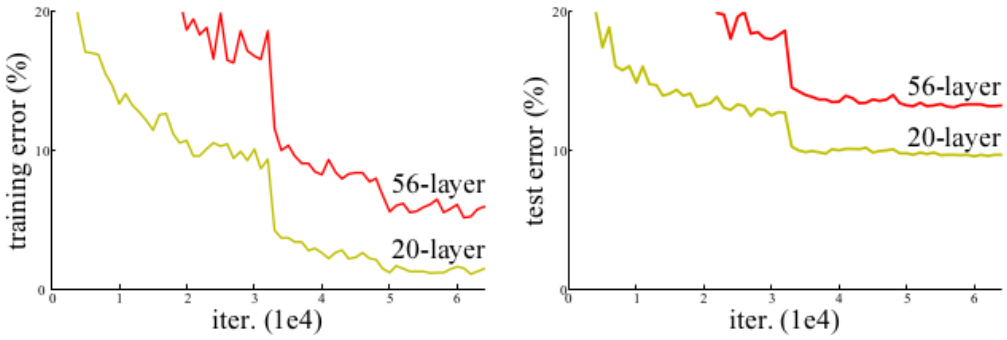

Figura 22 Erro de treino e de teste da CIFAR-10 para 20 e 56 camadas [16].

Para contornar este problema, foram criadas funções residuais para reformular as camadas. As entradas das camadas estão conectadas a mapas residuais que são mais fáceis de otimizar do que os mapas originais. Pode ser realizado através de *feedfoward* com atalhos nas conexões, como mostra a Figura 23, ou seja, os atalhos de conexões saltam uma ou mais camadas. A função $F(x) + x$ representa a função residual que é aplicada através de *feedfoward.*

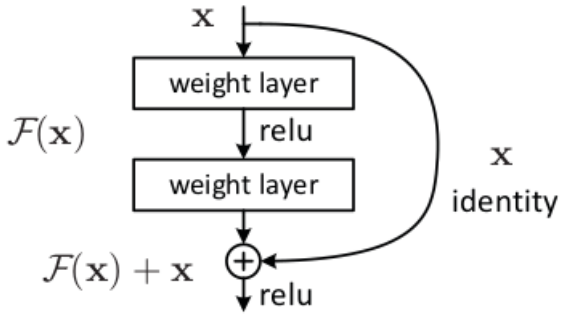

Figura 23 Aprendizagem residual [16].

As camadas convolucionais têm maioritariamente filtros de 3×3 com o mesmo número de canais, caso o mapa de características seja metade dos anteriores, o número de canais é o dobro para preservar a complexidade do tempo por camada.

A ResNet tem muitas versões, uma das mais utilizadas é a ResNet-50 com cinquenta camadas convolucionais. São necessários um total de 25,5 milhões de pesos e 3,9 mil milhões de MAC para processar uma imagem a cores de 224×224 (confirmar). A taxa de erro *top-1* é de 20,74% e *top-5* é de 6,67%.

Os modelos treinados de todas estas CNN estão disponíveis. Contudo as suas precisões são mais baixas em relação às precisões publicadas. Dependendo de como o modelo foi treinado, podem variar mais de 5% na precisão.

3.2. Aceleradores de CNN em Hardware

Os processadores genéricos não são eficientes para processar as CNNs e, por isso, foram propostos vários aceleradores de CNN. Comparando com os processadores, os aceleradores de CNN para a FPGA têm tido bons resultados devido ao seu elevado desempenho, baixo consumo e flexibilidade.

Em geral, as plataformas para computação embebida têm uma largura de banda de acesso a memória e uma capacidade de computação inferior, quando comparada com dispositivos ou plataformas de elevado desempenho. As redes descritas são bastante exigentes em termos computacionais e de memória. Por exemplo, a AlexNet tem cerca de 60 milhões de pesos, o que indica que é necessário carregar todos esses pesos para um sistema embebido, sendo necessário 240 MB de armazenamento de memória (considerando uma representação dos dados com quatro bytes).Como tal, para conseguir executar estas redes em plataformas embebidas com tempos de execução aceitáveis ou que cumpram requisitos de tempo-real, é essencial aplicar técnicas de redução destes requisitos. Neste sentido, várias técnicas de compressão de dados e de redução do número de dados têm sido propostas de modo a reduzir a largura de banda necessária na transferência de dados, bem como o número de operações aritméticas a executar.

No estudo de técnicas de aceleradores de CNNs para sistemas embebidos destacou-se um projeto com um modelo denominado de *efective roofline* [19]. Este modelo melhora o desempenho e reduz o consumo de energia com a restrição da largura de banda entre 100 a 200 MB/s. São aplicados diferentes algoritmos de compressão para melhorar a largura de banda necessária e o armazenamento de memória.

O modelo *roofline* permite determinar a configuração de uma arquitetura base para ser executada em FPGA (neste trabalho foi utilizada uma Xilinx Virtex7 485t), tendo em conta a largura de banda a utilizar e o número total de recursos computacionais necessários. A Figura 24 mostra o funcionamento deste modelo, onde o eixo dos XX representa a relação entre a

computação e a comunicação, *CTC Ratio*, ou seja, indica o número de operações por tráfego e o eixo dos YY representa o desempenho que um modelo pode atingir, AP (*attainable performace*), em GOPS.

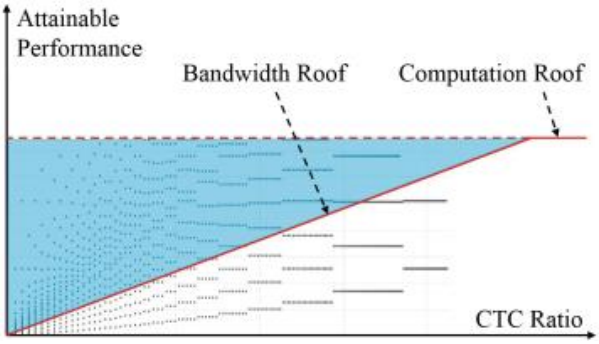

Figura 24 Análise de desempenho do modelo *roofline* [19].

O valor máximo (*computation roof*) define o limite de computação que é possível obter com os recursos computacionais considerados. A inclinação (*bandwidth roof*) define o limite de largura de banda, que é definido pela largura de banda máxima de acesso aos dados de um sistema embebido, BW_r.

Deste modo, tem-se:

$$CTC\ Ratio = \frac{Opereations}{Data}$$

$$AP = \frac{Opereations}{Cycles}$$

$$BW_r = \frac{Data}{Cycles} = \frac{AP}{CTC\ Ratio}$$

A nomenclatura utilizada é:

- *CTC Ratio, número de operações por tráfego*
- *Opereations,* número de operações
- *Data*, quantidade de dados externos
- *AP,* desempenho que um modelo pode atingir
- *Cycles*, número de ciclos
- BW_r, largura de banda necessária

O modo de compressão, neste projeto, não tem em conta o tempo e os recursos computacionais necessários para compactar os pesos das CNN que são compactados noutra plataforma. Assim, só é necessário enviar os pesos compactados e implementar um algoritmo de descompressão que não gera muito tempo nem recursos computacionais.

Os algoritmos de compressão utilizados são o LZ77, algoritmo baseado em dicionários, e a codificação de Huffman, algoritmo baseado em codificação de entropia. Muitos dos algoritmos de compressão são baseados nestes dois algoritmos.

Foi desenvolvido um acelerador para a CNN VGG-19 que tem dezasseis camadas convolucionais. A Tabela 8 mostra a diferença da taxa de compressão, *r*, a velocidade de descompressão, BW_d, e a percentagem dos recursos utilizados na descompressão, DSP, BRAM, LUT e Flip-flops (FF).

A taxa de compressão, *r,* é dado por:

$$r = \frac{Tamanho\ dos\ dados\ comprimidos}{Tamanho\ dos\ dados\ originais}$$

Algoritmo	r	BW_d [MB/s]	DSP [%]	BRAM [%]	LUT [%]	FF [%]
LZ77	0.48	114.7	0.00	0.97	4.52	0.82
Huffman	0.37	90.61	0.00	0.49	1.04	0.16

Tabela 8 - Comparação de uma unidade de descompressão.

Na Tabela 8 pode observar-se que a codificação de Huffman é melhor $1.30\times$ na taxa de compressão em relação ao LZ77. No geral, a utilização de recursos é também mais baixa com a codificação de Huffman, apesar de ser bastante baixa nas duas descompressões (ver Tabela 9).

Implementação	DSP [%]	BRAM [%]	LUT [%]	FF [%]
CNN	10.00	6.25	8.66	5.23
CNN -D(LZ)	27.14	17.48	85.36	32.78
CNN-D(HE)	40.00	30.10	89.92	18.39

Tabela 9 - Utilização global de recursos.

Foram criados três casos de estudo sem descompressão, CNN, com descompressão LZ77, CNN-D(LZ), e com descompressão de Huffman, CNN-D(HE). Todas as implementações são realizadas com a melhor configuração de hardware e com uma largura de banda de acesso aos dados de 181,20 MB/s.

Na Tabela 9 pode observar-se que a utilização de recursos aumentou, em comparação com a Tabela 8, pois é a utilização de recursos global para toda a rede. Deste modo, os casos com descompressão utilizam mais recursos, como era espectável, sendo, a descompactação de Huffman a que utiliza mais recursos.

A comparação de desempenho está representada na Tabela 10. Algumas das configurações das camadas convolucionais na VGG-19 são iguais, logo, os resultados obtidos são mostrados na mesma linha.

Número de Camadas	CNN		CNN – D(LZ)		CNN – D(HE)	
	Tempo (s)	GOPS	Tempo (s)	GOPS	Tempo (s)	GOPS
1	0.061	5.69	0.031	11.19	0.031	11.19
2	1.31	5.65	0.66	11.21	0.65	11.38
3	0.49	7.55	0.16	23.12	0.16	23.12
4	0.98	7.55	0.33	22.42	0.33	22.42
5	0.41	9.02	0.16	23.12	0.082	45.11
6, 7, 8	0.82	9.02	0.33	22.42	0.16	46.24
9	0.37	10.00	0.16	23.12	0.12	30.83
10, 11, 12	0.73	10.14	0.33	22.42	0.24	30.21
13, 14, 15,16	0.18	10.07	0.082	22.55	0.061	
Total GOPS	8.66		20.49		27.69	
Aceleração	1.00x		2.37x		3.20x	

Tabela 10 - Comparação de desempenho.

Como mostra a Tabela 10, o desempenho geral da CNN é de 8,66 GOPS, o que é muito mau em relação a outros projetos, no entanto, a largura de banda utilizada é de 181,20 MB/s. Comparado com a CNN, o CNN-D(LZ) alcança uma aceleração de $2{,}37\times$ no desempenho global, e a aceleração que o CNN-D(HE) alcança é de $3{,}20\times$. De entre os vários trabalhos sobre compressão e redução de redes CNN analisados, verificou-se que. existem várias métricas de corte que avaliam a importância dos pesos, como a média dos pesos, o desvio padrão, a amplitude dos pesos ou o custo de energia. O corte através da amplitude dos pesos é o critério mais simples e não necessita de um nível de computação elevado. Como tal, tem sido um dos mais utilizados.

Na implementação de técnicas de *pruning* de CNNs de Pavlo Molchanov, et al. [20] foi introduzida uma nova técnica baseada na série de Taylor. Para cada mapa de características é calculada a sua importância, ou seja, o custo que cada mapa de características produz. É aplicada aos mapas de características uma métrica baseada na serie de Taylor, selecionando os mapas de características com uma diferença de custo de aproximadamente zero.

A série de Taylor é dada por:

$$f(x) = \sum_{p=0}^{P} \frac{f^{(p)}(a)}{p!}(x-a)^p + R_p(x)$$

Considerando um conjunto de exemplos de treino $D = \{X, Y\}$, onde X representa o padrão de entrada e Y o padrão de saída.

$$X = \{x_0, x_1, \dots, x_N\}$$
$$Y = \{y_0, y_1, \dots, y_N\}$$

O custo, C, é calculado através da função probabilidade:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Uma função de custo é selecionada independentemente do corte e depende apenas dos pesos treinados da rede.

A diferença de custo entre o modelo cortado, $C(D|h_i = 0)$, e o modelo original, $C(D|h_i)$ é:

$$\Delta C(h_i) = C(D|h_i = 0) - C(D|h_i)$$

h_i representa o mapa de características.

Utilizando o polinómio de Taylor de primeiro grau, onde $f^{(p)}(a)$ é a derivada de f no ponto a, tem-se:

$$C(D|h_i = 0) = C(D|h_i) - \frac{\delta C}{\delta h_i} h_i + R_1(hi = 0)$$

$R_1(hi = 0)$ pode ser calculado através da fórmula de Lagrange:

$$R_1(h_i = 0) = \frac{\delta^2 C}{\delta(h_i^2 = \xi)} \frac{h_i^2}{2}$$

ξ é um número real entre zero e h_i.

Na Tabela 11 apresentam-se os resultados obtidos para diferentes redes, AlexNet, VGG-16 e R3DCNN [20], e para diferentes plataformas de hardware: CPU Intel Core i7-5930k, GPU GeForce GTX Titan X, GPU NVIDIA Jetson TX1 e GPU GeForce GT 730M, depois de treinadas. Todos os resultados foram realizados através do PyTorch com *cuDNN v.5.1.0*, exceto a rede R3DCNN que foi implementada em C++ com o *cuDNN v.4.0.4*. A base de dados é a ImageNet e a precisão é a classificação do *top-5*.

Hardware	Batch	Acc.	Tempo [ms]	Acc	Tempo (Aceleração)	Acc.	Tempo (Aceleração)

AlexNet 1.46 GFLOP				41% mapa carac. 0.4 GFLOP		19.5% mapa carac. 0.2 GFLOP	
Intel Core i7-5930k	16	80.1%	226.4		121.4 (1.9x)		87.0 (2.6x)
GeForce GTX TitanX	16		4.8	79.8%	2.4 (2.0x)	74.1%	1.9 (2.5x)
GeForce GTX TitanX	512		88.3	-0.3%	36.6 (2.4x)	-6.0%	36.6 (2.4x)
NVIDIA Jetson TX1	32		169.2		73.6 (2.3x)		73.6 (2.3x)
VGG-16 30.96 GFLOP				66% mapa carac. 11.5 GFLOP		52% mapa carac. *8.0 GFLOP*	
Intel Core i7-5930k	16	89.3%	2564.7	87.0%	1483.3 (1.7x)	84.5%	1218.4 (2.1x)
GeForce GTX TitanX	16		68.3	-2.3%	31.0 (2.2x)	-4.8%	20.2 (3.4x)
NVIDIA Jetson TX1	4		456.6		182.5 (2.5x)		138.2 (3.3x)
R3DCNN 37.8 GFLOP				25% mapa carac. *3 GFLOP*			
GeForce GT 730M	1	80.7%	438.0	78.2% -2.5%	85.0 (5.2x)		

Tabela 11 - *Pruning* através do critério de Taylor.

A AlexNet obteve uma compressão de aproximadamente 1,06 GFLOP (giga floating operations) com uma redução na precisão de $0{,}3\%$. Com um maior corte na rede, 80,5%, a precisão diminui 6% em relação à precisão original. Para a VGG-16 houve uma redução para 11,5 GFLOPs num total de 30,96 GFLOP da rede original, com uma redução na precisão de 2,3%. Com 8,0 GFLOPs obteve-se uma precisão de 84,5%, sendo a precisão original de 89,3%. Em relação à R3DCNN a redução foi de 75% e a precisão só diminuiu 2,5% em relação à original.

A plataforma *hardware* mais rápida é a GeForce GTX TITAN X. Em suma, esta técnica de corte obteve bons resultados comparando a precisão e a percentagem de corte da rede.

Outra técnica de *pruning* designada *Sparse Shrink* foi proposta por Xing Li e Changsong Liu [21], onde avalia a importância de cada canal. Os canais com pesos menos importantes, mais perto de zero, são considerados mais redundantes e são cortados para se obter uma rede mais pequena. O algoritmo foi utilizado na arquitetura da CIFAR-100 (semelhante à CIFAR-10, mas com mais classes) e obteve-se uma diminuição nos recursos computacionais com pouca diminuição da precisão.

A Tabela 12 compara a precisão em percentagem das camadas convolucionais para diferentes tamanhos de corte dos canais da CIFAR-100.

Canais Cortados	0	64	96	128	160	176
Conv1	68.08	67.80	67.86	67.86	67.36	67.38

Conv2	68.08	67.51	67.36	66.98	65.95	64.67
Conv3	68.08	67.68	67.00	66.07	65.09	61.17

Tabela 12 - Comparação da precisão em percentagem dos modelos cortados da CIFAR-100.

Como mostrado na Tabela 12, com uma diminuição de aproximadamente 1% na precisão, pode cortar-se até 176, 128 e 96 canais para as camadas convolucionais um, dois e três (destacado a laranja). Destacar que para um corte de 176 canais a camada *Conv1* tem apenas uma redução de 0,7% na precisão.

A Tabela 13 compara o número de pesos e de multiplicações entre o modelo original e o modelo reduzido da CIFAR-100.

Layer	Input Size	Número de Pesos			Número de Multiplicações		
		Modelo Base	Modelo Cortado	Redução (%)	Modelo Base	Modelo Cortado	Redução (%)
Conv1	32 x 32	193x3x5x5	16x3x5x5	91.67	$1.47x10^7$	$1.23x10^6$	91.67
Cccp1	32 x 32	160x192x1x1	160x16x1x1	91.67	$3.15x10^7$	$2.62x10^6$	91.67
Cccp2	32 x 32	96x160x1x1	96x160x1x1	0	$1.57x10^7$	$1.57x10^7$	0
Conv2	16 x 16	192x96x5x5	64x96x5x5	66.67	$1.18x10^8$	$3.93x10^7$	66.67
Cccp3	16 x 16	192x192x1x1	192x64x1x1	66.67	$9.44x10^6$	$3.15x10^6$	66.67
Cccp4	16 x 16	192x192x1x1	192x192x1x1	0	$9.44x10^6$	$9.44x10^6$	0
Conv3	8 x 8	192x192x3x3	96x192x3x3	50.00	$2.12x10^7$	$1.06x10^7$	50.00
Cccp5	8 x 8	192x192x1x1	192x96x1x1	50.00	$2.36x10^6$	$1.18x10^6$	50.00
Cccp6	8 x 8	100x192x1x1	100x192x1x1	0	$1.23x10^6$	$1.23x10^6$	0
Total	-	$9.83x10^5$	$4.24x10^5$	56.77	$3.23x10^8$	$8.45x10^7$	73.84

Tabela 13 - Comparação do número de pesos e multiplicações entre o modelo base e o modelo cortado.

O número de cortes de canais para as camadas *Conv1*, *Conv2* e *Conv3* foi de 176, 128 e 96, os melhores resultados da Tabela 12, e obteve-se uma precisão final de 65,63%. A Tabela 13 fornece uma comparação entre o modelo base e o modelo cortado em termos do número de pesos e do número de multiplicações. O número de pesos é reduzido a 56,77% e o número de multiplicações tem uma redução de 73,84%, com uma pequena redução de 2,55% na precisão.

Pode concluir-se que este algoritmo de corte é capaz de diminuir os recursos computacionais de um modelo bem treinado sem diminuir significativamente a precisão.

Uma outra implementação consistiu na utilização de três técnicas de corte e compressão aplicadas sequencialmente às CNN por Song Han, et al. [22]. Na Figura 25 pode observar-se um diagrama dos três estágios de compressão sequencial. No primeiro estágio é aplicado o

pruning com a magnitude dos pesos como critério, em que há uma redução de $10\times$ no número de pesos. De seguida, é aplicada a quantização, corresponde ao segundo estágio, que melhora a taxa de compressão entre $27\times$ a $31\times$. Por último é aplicado a codificação de Huffman que dá uma compressão entre $35\times$ a $49\times$, sendo este o terceiro estágio.

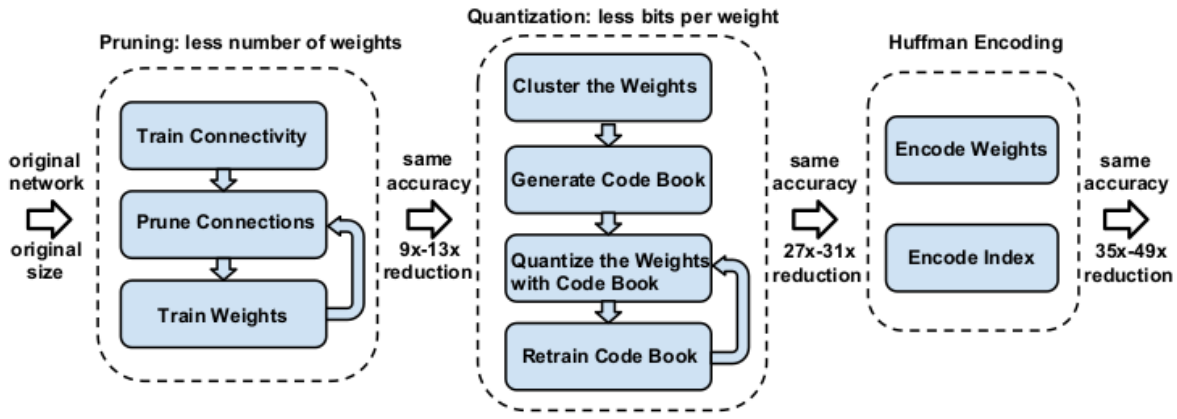

Figura 25 Três estágios de compressão, *pruning,* quantização e codificação de Huffman [22].

No primeiro estágio, o *pruning* é aplicado a uma rede treinada e é realizado através da verificação dos pesos, em que todos os pesos abaixo de um determinado *threshold* são removidos. Depois do *pruning* a rede é treinada novamente. Com o *pruning* os pesos ficam muito dispersos, assim para aumentar a compressão é armazenada a diferença de índice em vez do índice da posição de cada peso. A diferença entre índices é codificada em 8 bits para as camadas convolucionais e em 5 bits para as camadas FC. Para índices maiores que o limite, é feito um preenchimento com zero. A Figura 26 mostra um exemplo quando o índice é excedido com 8 bits. Para os modelos da AlexNet e da VGG-16 há uma redução no número de pesos de $9\times$ e de $13\times$.

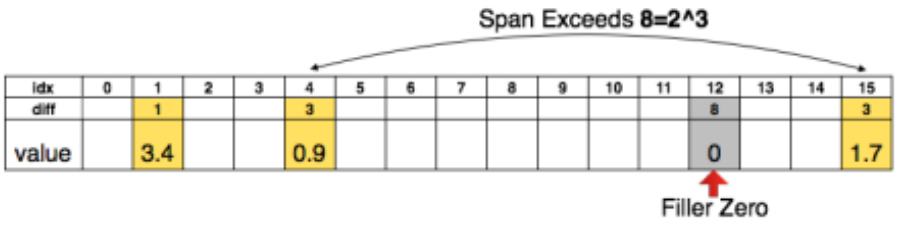

Figura 26 Representação da matriz esparsa com o índice relativo.

No segundo estágio, a quantização reduz o número de bits necessários para representar cada peso, por exemplo, reduz de 32 bits para 16 bits e no fim é aplicado *fine-tunning*.

Por fim, no terceiro estágio, é aplicada uma compressão sem perdas, a codificação de Huffman, que aumenta a compressão entre $27\times$ a $31\times$.

Na Figura 27 é comparado a precisão para diferentes taxas de compressão com o *pruning* e com a quantização juntos e individualmente.

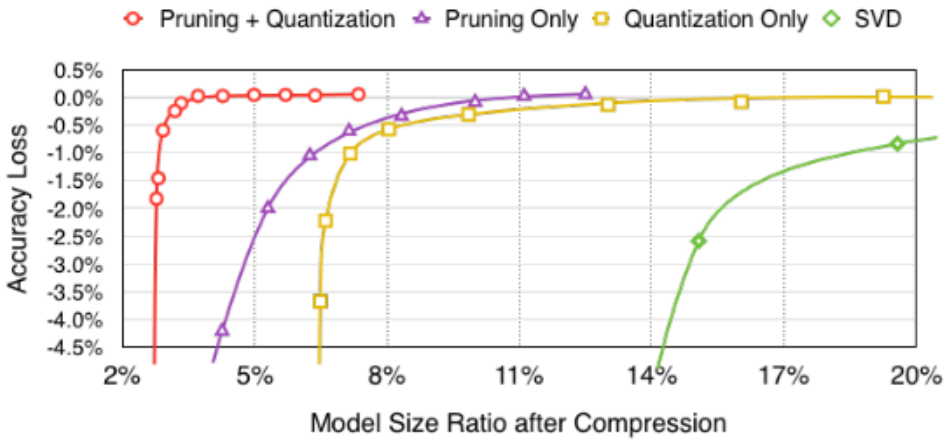

Figura 27 Precisão v.s. taxa de compressão para diferentes métodos de compressão [22].

Com a compressão aplicada individualmente, como mostrado nas linhas roxa e amarela, a precisão da rede cortada começa a cair significativamente quando comprimida abaixo dos 8% do seu tamanho original. A precisão com a quantização também começa a cair significativamente quando comprimida abaixo dos 8% do seu tamanho original. Quando aplicados juntamente, como mostrado na linha a vermelho, a rede pode ser comprimida até 3% do tamanho original sem perda de precisão. Pode concluir-se que o *pruning* e a quantização funcionam melhor quando combinadas.

Na Tabela 14 pode-se observar o erro *top-1* e *top-5*, o tamanho total dos pesos em *KBytes* e a taxa de compressão para as redes LeNet-300-100, LeNet-5, AlexNet e VGG-16 depois de aplicar os três estágios de compressão.

Rede	*Top-1* Erro	*Top-5* Erro	Pesos	Taxa Compressão
LeNet-300-100 Ref. LeNet-300-100 Comp.	1.64 % 1.58 %	- -	1070 KB 27 KB	40 x
LeNet-5 Ref. LeNet-5 Comp.	0.80 % 0.74 %	- -	1720 KB 44 KB	39 x
AlexNet Ref. AlexNet Comp.	42.78 % 42.78 %	19.73 % 19.70 %	240 MB 6.9 MB	35 x
VGG-16 Ref. VGG-16 Comp.	31.50 % 31.17 %	11.32 % 10.91 %	552 MB 11.3 MB	49 x

Tabela 14 - Compressão sequencial para diferentes redes.

Pode concluir-se que, através destes três métodos de compressão, a precisão das redes não varia e a taxa de compressão é bastante elevada. Para as redes da LeNet o erro *top-1* diminui e obteve-se uma taxa de compressão de aproximadamente $40\times$. Com a AlexNet o erro manteve-se, não houve perda na precisão, com uma taxa de compressão de $35\times$ e para a VGG-16 há uma compressão de $49\times$, mas com uma diminuição no erro de 0,33% e 0,41% para o *top-1* e *top-5*.

A técnica *Low-Rank Regularization - LRR*, foi proposto por Cheng Tai, et al. [23], é aplicada às camadas convolucionais. O método LRR altera, nas camadas convolucionais, o tamanho dos filtros para filtros unidimensionais, ou seja, para um filtro de tamanho $k \times k$ o algoritmo transforma em dois filtros de tamanho $k \times 1$ e $1 \times k$. A Figura 28 ilustra à esquerda a camada convolucional original e à direita a camada convolucional com a utilização desta técnica. Para alterar o tamanho do filtro é criada uma camada convolucional igual à original só com o valor do número de canais, K, e o tamanho do filtro diferentes.

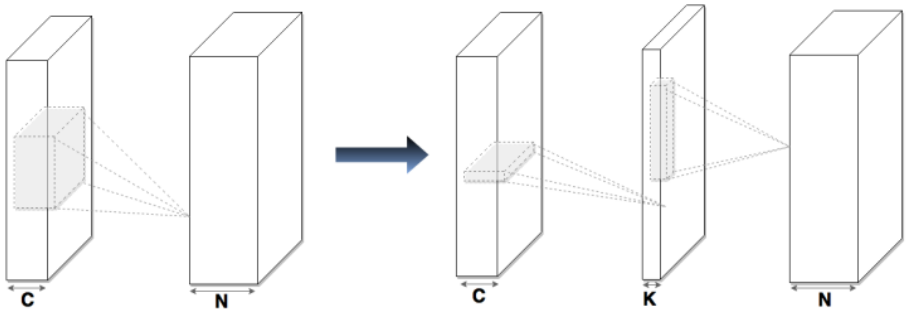

Figura 28 Parametrização proposta para Low-Rank Regularization [23].

Na tabela 15 está representado para diferentes valores de $\boldsymbol{K}$ e para as três camadas convolucionais do modelo CNN+Dropout, a precisão, a aceleração da camada, a aceleração teórica em toda a rede e a taxa de redução do número de pesos. Estes resultados foram obtidos através do GPU Nvidia Titan e o Torch7 com cuDNN. O modelo utilizado tem uma precisão de 87,71%.

Camada	K1	K2	K3	Precisão	Aceleração Camada	Aceleração Rede	Redução Pesos
1ª	4	64	256	+0.69%	1.20 x	2.91 x	3.5 x
	8	64	256	+0.85%	1.13 x	2.87 x	1.8 x
	12	64	256	+0.94	1.05 x	2.85 x	1.2 x
2ª	12	8	256	-0.02%	7.13 x	3.21 x	47.5 x
	12	16	256	+0.50%	6.76 x	3.21 x	23.8 x
	12	32	256	+0.89%	6.13 x	3.13 x	12.0 x
	12	64	256	+0.94%	3.72 x	2.86 x	6.0 x
	12	128	256	+1.32%	2.38 x	2.58 x	3.0 x

	12	256	256	+1.40%	1.25 x	1.92 x	1.5 x
3ª	12	64	8	-2.25%	6.98 x	3.11 x	52.5 x
	12	64	16	+0.21%	6.89 x	3.11 x	26.4 x
	12	64	32	+0.19%	5.82 x	3.10 x	13.3 x
	12	64	64	+0.19%	3.74 x	2.96 x	6.7 x
	12	64	128	+0.94%	2.38 x	2.86 x	3.3 x
	12	64	256	+1.75%	1.31 x	2.30 x	1.7 x

Tabela 15 - Low-Rank para diferentes valores de K.

A aplicação do LRR para todas as camadas convolucionais reduz o número de pesos sem degradar muito o desempenho. Por exemplo, para $k1 = 12$, $k2 = 16$ e $k3 = 32$, os pesos são reduzidos 91% e a precisão é de +0,25%.

Esta técnica foi aplica aos modelos da AlexNet, da VGG-16 e da GoogleNet, onde nas Tabela 16 estão representados os valores de K utilizados. No caso da VGG-16 há camadas com duas ou três camadas sub-convolucionais, por isso, têm dois ou três valores de K. Para o caso da GoogleNet cada camada *inception* tem três valores de K.

AlexNet		VGG-16	
Camada	K	Camada	K
Conv1	8	Conv1	5, 24
Conv2	40	Conv2	48, 48
Conv3	60	Conv3	64, 128, 160
Conv4	100	Conv4	192, 192, 256
Conv5	200	Conv5	320, 320, 320

GoogleNet			
Camada	K	Camada	K
Conv1	8	Inception4c	64, 64, 64
Conv2	48	Inception4d	64, 96, 96
Inception3a	32, 32, 48	Inception4e	64, 128, 160
Inception3b	32, 32, 48	Inception5a	128, 96, 128
Inception4a	32, 64, 80	Inception5b	128, 96, 128
Inception4b	64, 64, 80		

Tabela 16 - Modelos da AlexNet, da VGG-16 e da GoogleNet com Low-Rank.

Na Tabela 17 há a comparação entre os modelos originais e os modelos com *Low-Rank* com os valores de K da Tabela 16. Os resultados obtidos são da precisão *top-5*, da aceleração teórica, da aceleração prática e da taxa de redução do número de pesos.

Rede	*Top-5* Acc.	Aceleração Teórica	Aceleração Prática	Redução Pesos
AlexNet	80.03	1 x	1 x	1 x
AlexNet Low-Rank	79.66	5.27 x	1.82 x	5.00 x
VGG-16	90.60	1 x	1 x	1 x
VGG-16 Low-Rank	90.31	3.10 x	2.05 x	2.75 x
GoogleNet	92.21	1 x	1 x	1 x
GoogleNet Low-Rank	91.79	2.89 x	1.20 x	2.84 x

Tabela 17 - Comparação entre os modelos com Low-Rank e os modelos base.

Para os três modelos a precisão diminui muito pouco, a AlexNet conseguiu uma diminuição de pesos de $5\times$ e um aumento na aceleração de $1{,}82\times$. A VGG-16 teve uma redução nos pesos de $2{,}75\times$ e um aumento na aceleração de $2{,}05\times$, enquanto que a GoogleNet teve um aumento inferior em relação aos outros dois modelos, com um aumento de $1{,}20\times$ e uma redução de $2{,}84\times$ no número de pesos.

O método LRR obteve bons resultados para algumas das CNNs mais utilizadas, o que demonstra ser uma boa ferramenta para acelerar grandes CNNs.

Um trabalho em que é aplicado compressão e redução de dados a redes implementadas em FPGA é o trabalho *Accelerating CNN inference on FPGAs* [24]. Neste trabalho é aplicado o método de *Low-Rank* e de *pruning* a diferentes modelos de FPGAs. Para reduzir o número de pesos aplicou-se às camadas FC uma aproximação do método *Low-Rank* ao modelo VGG16, que tem uma precisão de 87,96% com a ImageNet.

O método de *pruning é* também aplicado ao modelo VGG16, mas com diferentes bases de dados, a ImageNet e a Cifar-10. Após a aplicação destes métodos em separado, os pesos ficam muito dispersos, o que pode ser aproveitado pela FPGA. Foi desenvolvida uma implementação onde as multiplicações pelos pesos com valor zero são ignoradas. A Tabela 18 mostra esta implementação para diferentes dispositivos.

	Base de Dados	Comp [GOP]	Param. [M]	Param. Removidos [%]	Nº bits	Acc [%]	Dispositivo	Freq. [MHz]	Through. [GOPs]	P [W]	LUT [K]	DSP	Mem. [MB]
Low-Rank	ImageNet	30,5	138,0	63,6	16 Fixed	87,96	Zynq 7Z045	150	137	9,6	183	780	17,5
Pruning	Cifar10	0,3	132,9	89,3	8 Fixed	91,53	Kintex 7k325T	100	8621	7,0	17	145	15,1
Pruning	ImageNet	1,4	61,0	85,0	32 Float	79,70	Stratix 10	500	12000	141,2	-	-	-

Tabela 18 – Aplicação de aceleradores de CNN em FPGA através do *pruning e Low-Rank.*

Pode verificar-se que através do *Low-Rank* o número de parâmetros removidos é mais reduzido do que através do *pruning*. Para a mesma base de dados, a precisão da rede irá ser menor quanto maior o número de corte de pesos apesar do número de bits ser superior, o que leva a concluir que os pesos não têm valores muito baixos.

O dispositivo com maior frequência tem um número de operações por segundo muito superior aos restantes, apesar de também ter aproximadamente metade do número de pesos.

É possível concluir que com esta implementação é possível reduzir o número de operações, contudo depende do tipo de dispositivo.

Com os resultados obtidos verifica-se que, em geral, é ineficiente implementar as técnicas de *pruning* em qualquer plataforma devido à irregularidade introduzida. Assim para evitar esta ineficiência, neste trabalho pretende-se implementar o método de *pruning* para blocos com um agrupamento de n pesos.

O número de trabalhos de compressão e de redução de dados aplicados a redes implementadas em FPGA é bastante reduzido, o que para efeitos comparativos entre dispositivos e técnicas de compressão torna difícil obter boas conclusões. Desta forma, neste trabalho pretende-se estudar como cada técnica aplicada, tanto em forma individual como em conjunto com outras técnicas, influencia a precisão da rede e o desempenho da arquitetura LiteCNN [2].

Arquitetura LiteCNN: Otimização com Métodos de Compressão
4

Neste capítulo, descrevemos a arquitetura LiteCNN que visa a implementação de CNN de grande dimensão em FPGA de baixo custo com aplicação em sistemas embebidos. A versão atual da arquitetura suporta implementações com dados representados a 8 bits em vírgula fixa dinâmica.

No trabalho desta tese, procurou-se otimizar a LiteCNN com a utilização de técnicas de redução de pesos e de compressão que permitam manter a representação de dados reduzida (evitando a representação em vírgula flutuante). Assim, neste capítulo, começamos com a descrição do ambiente e das metodologias utilizadas para estudar o impacto das técnicas na precisão das CNN.

De seguida, descrevemos a arquitetura LiteCNN, versão atual a 8 bits. Depois, adaptamos a arquitetura para dar suporte às técnicas de redução e de compressão estudadas e propostas. No âmbito deste estudo, com o objetivo de estabelecer relações entre a precisão da rede e o desempenho e a área do hardware, são propostos dois modelos da arquitetura: um de desempenho e outro de área.

4.1. Otimização da CNN com Redução e Compressão dos Operandos

Os algoritmos de redução e compressão desenvolvidos foram aplicados inicialmente na ferramenta Caffe. O Caffe é uma ferramenta de treino e de classificação de redes, desenvolvida pela Berkeley AI Research (BAIR) e por colaboradores da comunidade e é uma estrutura *open-source* com modelos e exemplos de trabalhos para CNN.

Esta escolha deve-se à fácil configuração da aplicação entre CPU e GPU e à grande comunidade e suporte para vários projetos de pesquisa académica e para aplicações industriais em larga escala na visão, na fala e em multimédia. Existem também várias ferramentas desenvolvidas para o Caffe, de forma a alterar as definições dos modelos e a otimizar as suas configurações. A sua utilização é possível em qualquer sistema operativo juntamente com uma linguagem de programação como Python, MATLAB ou C/C++.

A arquitetura e a configuração do treino/teste das redes são feitas em ficheiros *prototxt* que depois de treinadas criam um ficheiro *caffemodel* com os pesos treinados. O ficheiro

prototxt contém a arquitetura da rede, e nele é descrito todas as camadas da rede incluindo o nome e o tipo. Cada camada é constituída por parâmetros que a caracterizam, sendo que estes parâmetros variam conforme o tipo de camada. Os parâmetros da camada convolucional são:

- Obrigatórios:
  - *num_output*: número de filtros;
  - *kernel_size* (ou *kernel_h* e *kernel_w*): especifica a altura e o comprimento de cada filtro.
- Recomendados:
  - *weight_filler*: tipo de filtro (valor padrão é constante com o valor 0).
- Opcionais:
  - *bias_term*: especifica se é necessário aprender e aplicar o *bias* à saída do filtro;
  - *pad* (ou *pad_h* e *pad_w*): indica quantos bits a adicionar a cada lado do mapa de entrada (*zero_padding*). O valor padrão é zero;
  - *stride* (ou *stride_h* e *stride_w*): deslocamento dos filtros à entrada (o valor padrão é um);
  - *group* (g): se g é maior que um, a conectividade de cada filtro é um subconjunto da entrada. Os canais de entrada e saída são separados em grupos g.

Os parâmetros da camada de agrupamento são:

- Obrigatórios:
  - *kernel_size* (ou *kernel_h* e *kernel_w*).
- Opcionais:
  - *pool*: tipo de pooling, o valor padrão é a *max-pooling*;
  - *pad* (ou *pad_h* e *pad_w*);
  - *stride* (ou *stride_h* e *stride_w*).

Os parâmetros da camada totalmente conectada são:

- Obrigatórios:
  - *num_output*.: número de filtros
- Recomendados:
  - *weight_filler: tipo de filtro*.
- Opcionais:
  - *bias_filler*, tipo de filtro (valor padrão é constante com o valor 0);
  - *bias_term*: ver explicação acima.

Os modelos são definidos de baixo para cima, ou seja, as ligações das camadas são feitas através dos parâmetros *bottom* e *top,* que se referem à camada anterior e à camada atual (ver Figura 29).

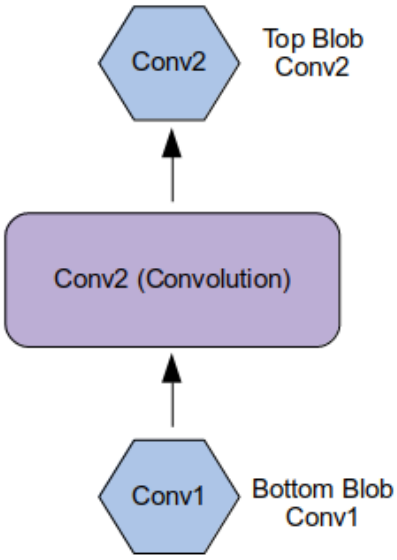

Figura 29 Exemplo da utilização do *blob* [8].

O *blob* é um *array* bidimensional que armazena os dados que são processados e transmitidos pelo Caffe e fornece a capacidade de sincronização entre CPU e GPU.

Neste projeto são consideradas as técnicas de *Low-Rank*, de *pruning*, de codificação de Huffman e de compressão de matrizes esparsas. A metodologia de implementação de cada uma dessas técnicas são descritas ao longo deste capítulo.

4.1.1. Método de *Low-Rank*

Através das ferramentas do Caffe foi implementado em Python ficheiros *prototxt* dos modelos de algumas redes com a aplicação do método *Low-Rank*. Os ficheiros criados são baseados nos ficheiros dos modelos das redes já existentes, mas com a alteração do número de camadas convolucionais e utilizando os parâmetros *kernel_h* e *kernel_w* para definir a altura e o comprimento do filtro ($k \times 1$ e $1 \times k$).

Um exemplo em Python da aplicação deste método está representado na Figura 30. São utilizados os parâmetros descritos anteriormente, retirados dos modelos já existente usados como teste para este trabalho. A *NetSpec* é uma classe utilizada para especificar explicitamente os nomes dos *blobs,* contendo todas as camadas necessárias para produzir um modelo de uma rede. É necessário importar as bibliotecas do Caffe, *layers* e *params*, que servem para definir o tipo de camada e os parâmetros de cada camada. O primeiro parâmetro de

cada camada é o nome da camada anterior. O *num_output* das camadas convolucionais verticais tem de ser o menor possível para diminuir ao máximo o tamanho das redes, o *num_output* das camadas convolucionais horizontais mantém-se igual ao original para não alterar a estrutura da rede.

```
import caffe
from caffe import layers as L, params as P

n = caffe.NetSpec()
n.conv_v = L.Convolution(n.data, kernel_h=5, kernel_w=1 num_output=1,
                         weight_filler=dict(type='xavier'))
n.conv_h = L.Convolution(n.data, kernel_h=1, kernel_w=5 num_output=20,
                         weight_filler=dict(type='xavier'))
n.pool = L.Pooling(n.conv_h, kernel_sizer=5, stride=2,
                   pool=P.Pooling.MAX)
n.fc = L.InnerProduct(n.pool, num_output=50, weight_filler=dict(type='xavier'))
n.relu = L.Relu(n.fc, in_place=True)
n.loss = L.SoftmaxWithLoss(n.relu, n.label)
with open('ficheiro.prototxt', 'w') as f :
```

Figura 30 Exemplo em Python para gerar um ficheiro *prototxt* de uma rede com Low-Rank.

4.1.2. Método de *Pruning*

Como anteriormente descrito, o *pruning* pode ter diferentes métricas e métodos para reduzir o tamanho das CNNs. Neste projeto a métrica utilizada é a saliência dos pesos das camadas totalmente conectadas. É verificada a sua magnitude e são colocados a zero uma percentagem dos pesos que têm a magnitude mais próxima de zero, seguindo o fluxo representado no diagrama da Figura 31.

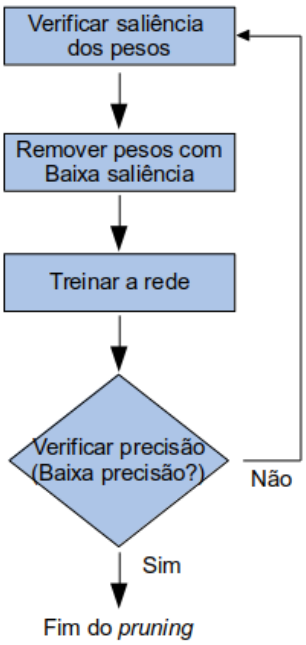

Figura 31 Diagrama da implementação do *pruning.*

O algoritmo de *pruning* foi implementado em Python (ver Anexo A - ALGORITMO *PRUNING*), onde verifica a magnitude dos pesos e coloca a zero uma percentagem de pesos mais próxima de zero, (ver pseudocódigo do algoritmo na Figura 32). O ficheiro do tipo *lmdb* é onde se encontra guardada a base de dados de imagens de treino.

Entrada: Caffemodel, prototxt (com modelo da rede), lmdb de teste (base de dados), camadas a cortar, percentagem de corte, tamanho do bloco.

Saída: Caffemodel

```
for idx = 0 to layer_lenght do
        block_data = layer[idx:idx+block_lenght]
        data[i] = meanFuction( abs(block_data) )
        idx = idx + block_lenght
end for
idx_sort = sortIndexFuction(data)
for idx = 0 to pruning_ratio do
        i = idx_sort[idx]
        i = i x block_lenght
        layer[i:i+block_lenght] = 0
end for
```

Figura 32 Pseudocódigo algoritmo de *pruning.*

Para otimizar a implementação em FPGA, propôs-se neste trabalho a aplicação de *pruning* a grupos de pesos. Esta otimização está relacionada com o facto de que as arquiteturas hardware dedicadas implementam paralelismo ao nível do produto interno com a leitura em paralelo de grupos de pesos (como veremos na secção seguinte, a LiteCNN também implementa paralelismo a este nível). Para tirar proveito deste paralelismo, propomo-nos fazer *pruning* em grupos de pesos. Assim, mantemos a granularidade de processamento do algoritmo similar à granularidade de computação de modo a manter a eficiência de cálculo da arquitetura.

Como a arquitetura LiteCNN original suporta o processamento de produtos internos com até 16 bytes em paralelo, foi implementado e testado *pruning* para blocos com um agrupamento de dois, quatro, oito e dezasseis pesos (tendo em conta a precisão da rede, outros tamanhos poderiam ter sido facilmente considerados).

A técnica consiste em cortar os pesos por blocos, ou seja, é calculada a média da magnitude de um bloco de N pesos e são ordenados por ordem crescente. A percentagem pretendida de blocos com a média mais pequena são colocados a zero. A Figura 33 mostra um exemplo para blocos de quatro, com cortes de 50 e de 75 %.

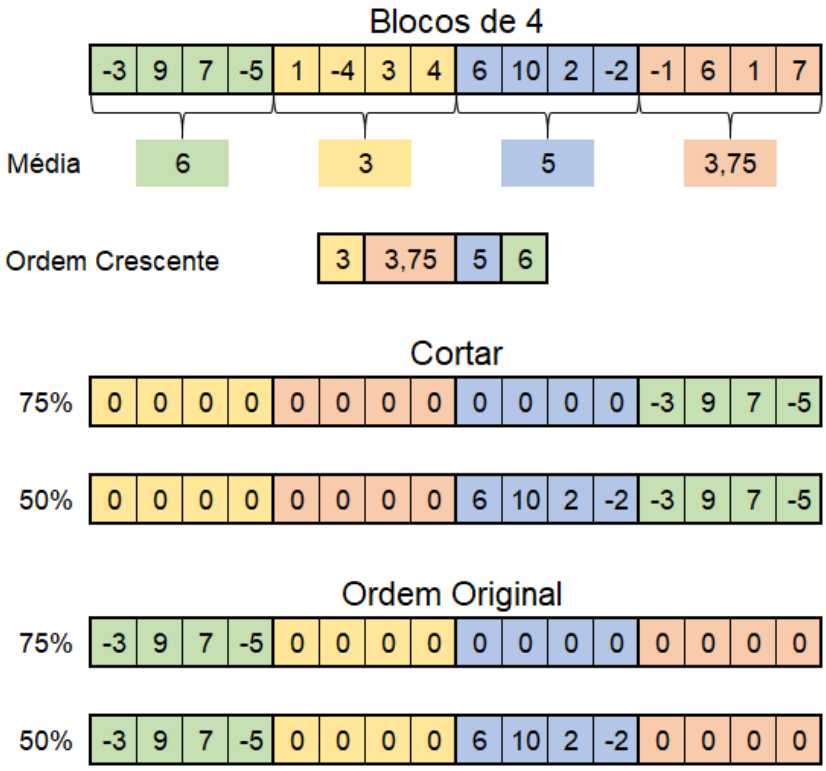

Figura 33 Exemplo da aplicação do *pruning* com blocos de 4.

Em FPGA, é necessário saber a posição dos pesos, assim o *pruning* não aplica qualquer compressão de dados, pois os pesos não podem ser removidos para não se perder a sua

posição. Contudo, como uma grande percentagem dos pesos ficam com o valor zero são utilizados algoritmos de compressão para comprimir as camadas totalmente conectadas.

Para auxiliar o *pruning* são aplicados dois algoritmos de compressão diferentes, a codificação de Huffman e a compressão baseada na compressão de matrizes esparsas. Uma matriz esparsa é uma matriz na qual a grande maioria dos seus elementos possui um valor padrão, neste caso o valor zero.

4.1.3. Método de Codificação de Huffman e de Compressão de Matrizes Esparsas

A codificação de Huffman, explicada anteriormente, foi desenvolvida em Python (ver Anexo B – ALGORITMO HUFFMAN) com base na Figura 17. O seu pseudocódigo está representado na Figura 34.

```
Entrada: symbols
Saída: code
while len < lenght(symbols) :
        sortProbability(symbols)
        s0 = remove_less_probability(symbols)
        s1 = remove_less_probability(symbols)
        s = new_symbol_node
        s.probability = s0.probability + s1.probability
        s.symbol = s0.symbol + s1.symbol
        insert(symbols, s)
        len = len + 1
end while
while s :
        s.code = run_all_tree(s)
end while
```

Figura 34 Pseudocódigo do algoritmo de Huffman.

O outro tipo de compressão, baseada na compressão de matrizes esparsas, foi desenvolvida em alternativa à codificação de Huffman.

Este tipo de codificação guarda só os valores diferentes do valor padrão e a sua posição na matriz. Uma possível codificação, caso a matriz seja bidimensional, é guardar numa lista

a posição das linhas dos pesos diferentes de zero que apontam para uma matriz com o seu valor e a posição da coluna. Na Figura 35 pode observar-se um exemplo através da codificação de uma matriz bidimensional.

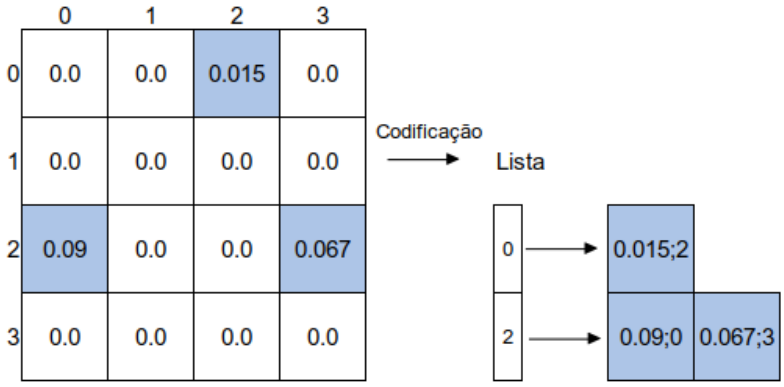

Figura 35 Exemplo da compressão de matrizes esparsas.

Neste trabalho, foi desenvolvida outra solução para tornar a descodificação em FPGA mais simples, ilustrado na Figura 36. A solução encontra foi guardar o número total de zeros entre valores diferentes de zero, em vez de guardar a sua posição, o código desenvolvido em Python encontra-se no Anexo C – ALGORITMO MATRIZ ESPARSA.

Para armazenar os pesos, são utilizados 16 bits para guardar o valor do peso (8 bits) e o número de zeros até ao próximo número (8 bits). O ficheiro *caffemodel* guarda o valor dos pesos com vírgula flutuante de 32 bits, assim, é necessário passar os pesos de 32 bits para 8 bits. Caso, o número de zeros até ao próximo valor ultrapasse os 8 bits (256 zeros) é guardado um novo valor com o número zero e o número total de zeros até o próximo valor.

Guardar o número total de zeros torna-se mais simples na parte da descodificação e torna-se menos dispendioso no armazenamento de memória, porque o contador do número de zeros ocupa apenas 8 bits. Por exemplo, no caso da AlexNet, uma das camadas totalmente conectada tem cerca de 16,7 milhões de pesos, sendo necessário 24 bits para representar cada a posição.

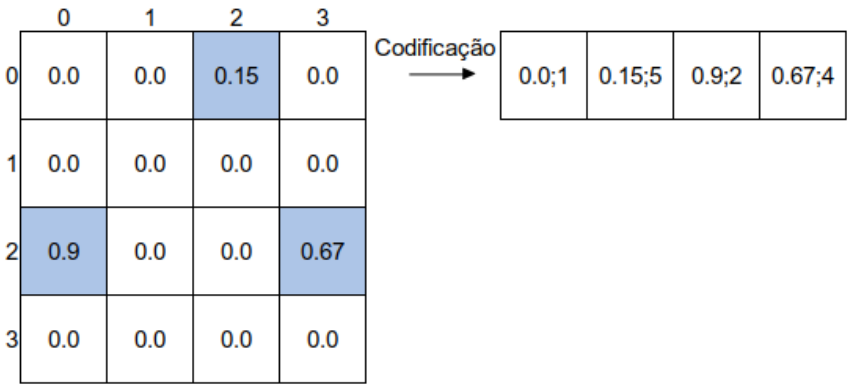

Figura 36 Exemplo da compressão baseada nas matrizes esparsas.

Por exemplo, se for utilizada a rede da AlexNet, tem uma camada totalmente conectada com 16,7 milhões de pesos, que ocupam cerca de 67 MBytes. Aplicando o primeiro algoritmo de compressão (Figura 35) seria necessário um corte na camada de pelo menos 60% para ocupar o mesmo espaço do que sem compressão (índice de 24 bits e pesos de 32 bits). Com os pesos a 8 bits seria necessário um corte de aproximadamente 40% para ocupar o mesmo espaço do que sem compressão. Para o algoritmo da Figura 36, como apenas são necessários 16 bits, com um *pruning* de 50%, ocupa aproximadamente 17 MBytes, caso o número máximo de zeros entre pesos seja inferior a 255.

Quando se considera *pruning* sobre grupos de bytes, apenas se utiliza um byte de contador de zeros por cada bloco. Assim, é armazenado um bloco de *m* pesos seguido de um valor com o número de blocos a zero até ao próximo bloco diferente de zero.

4.2. Arquitetura LiteCNN: Versão 8 bits

A arquitetura LiteCNN tem uma estrutura configurável que executa uma camada de cada vez. O núcleo principal da arquitetura calcula convoluções 3D entre as ativações e os pesos explorando paralelismo de saída (calcula vários mapas de saída em paralelo), paralelismo do mapa de saída (calcula várias ativações de um mapa de saída em paralelo) e paralelismo do *kernel* (com a paralelização do cálculo das convoluções).

Em termos estruturais, a arquitetura contém um núcleo de cálculo com vários PEs, um buffer de memória, responsável por armazenar a imagem inicial e os resultados intermédios, uma memória externa, e módulos responsáveis por enviar e receber ativações e pesos entre o *buffer* de memória e o núcleo de cálculo (ver Figura 37).

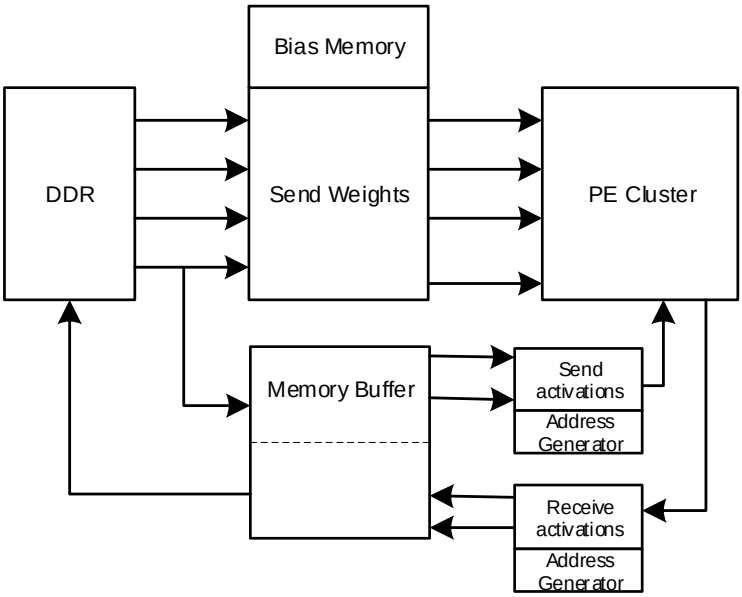

Figura 37 Arquitetura LiteCNN [2].

A execução de uma CNN na arquitetura processa-se do seguinte modo:

1. Configura-se a arquitetura para uma camada específica (convolucional ou totalmente ligada). Adicionalmente, configura-se a existência de *pooling*;
2. Envia-se a imagem a processar para o buffer de memória e os pesos dos filtros da memória externa para os PE (enviados pelo módulo *sendWeights*) juntamente com o *Bias* (armazenado no *Bias memory*) e os produtos redundantes do algoritmo da redução do número de multiplicações descrito em 4.1.1 que envolvem apenas os pesos (como é o caso de W_0W_1, W_2W_3, W_4W_5 e W_6W_7). Esta soma é armazenada numa memória local do sistema. Cada PE recebe um *kernel* e é responsável por calcular o mapa de saída associado a esse *kernel*;
3. De seguida, a imagem ou as ativações são enviadas em *broadcast* do *buffer* para todos os PEs (*sendActivations*). Enquanto é feita a sua leitura são calculados e armazenados os valores redundantes da aplicação do algoritmo da redução do número de multiplicações que envolvem apenas os valores do mapa de características (como é o caso de P_1P_0, P_3P_2, P_5P_4 e P_7P_6 no exemplo da Figura 37);
4. Após o cálculo dos PEs entre o filtro e a parte do mapa de características que o PE recebeu, os resultados são enviados de volta para o módulo recetor de ativações (*receiveActivations*) que subtrai o *Bias* e os valores redundantes das multiplicações e guarda o resultado no *buffer* de memória;

Quando a convolução entre os filtros enviados para os PEs e a imagem estiver concluída são carregados novos filtros nos PEs e processo repete-se até serem executados todos os filtros.

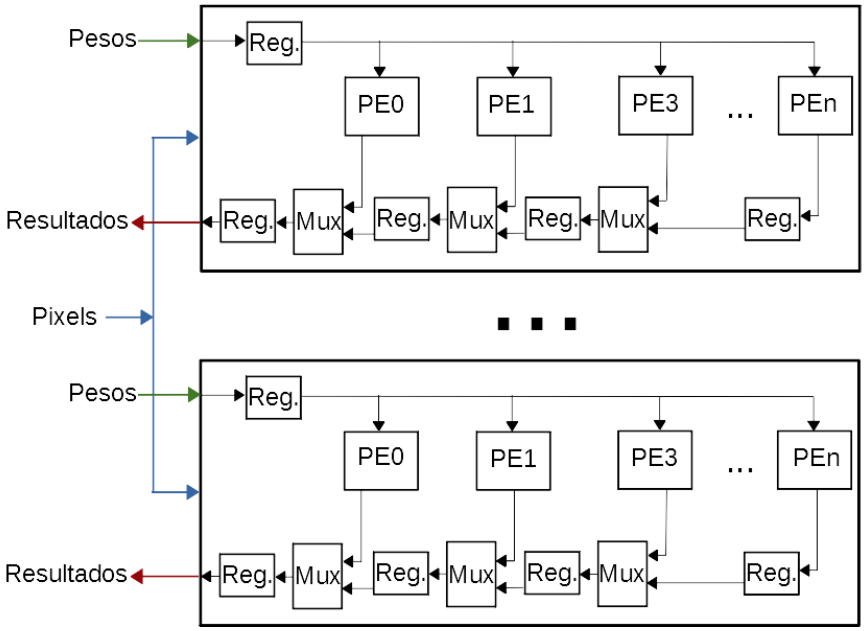

Figura 38 Arquitetura do módulo ClusterSet.

O módulo *PE cluster* corresponde ao grupo de elementos de processamento (PE) responsáveis por realizar o cálculo dos produtos escalares. Este módulo é constituído por vários grupos de elementos de processamento, e a sua arquitetura está ilustrada na Figura 38.

O PE é composto por uma memória local para armazenar os pesos e por um módulo *Core* (ver Figura 39). O módulo *Core* é constituído por um módulo *Cell*, que através de DSP e LUT realiza o cálculo de dois produtos escalares diferentes, que depois são acumulados (em separado).

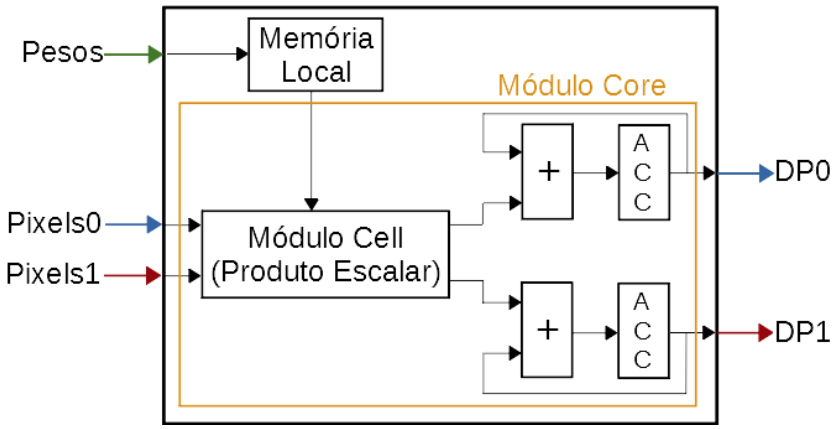

Figura 39 Arquitetura do PE.

Cada PE recebe e armazena na memória local os pesos de um *kernel* diferente. Cada PE pode aplicar os filtros a mais do que um bloco de ativações de entrada, ou seja, é possível

aplicar o mesmo *kernel* a diferentes mapas de características de entrada, produzindo assim diferentes ativações do mapa de saída. O número de ativações de saída calculadas em paralelo é configurável. No exemplo da Figura 39, este valor é de dois, pois recebe dois blocos de ativações em paralelo (*pixels0* e *pixels1*). Por fim, o PE lê múltiplos pesos e ativações em paralelo com um único acesso à memória (a versão atual lê palavras de 64 bits a que corresponde 8 filtros de 8 bits em paralelo) permitindo explorar o paralelismo no cálculo do produto interno.

O produto escalar das convoluções entre o mapa de características de entrada e os pesos dos filtros realizadas nas camadas convolucionais e nas camadas totalmente conectadas está otimizado com um método de redução do número de multiplicações. Na FPGA, o custo de processamento de uma multiplicação é superior ao custo de processamento de uma adição. A redução em metade dos multiplicadores utilizando esta técnica permite reduzir os recursos necessários para implementar um PE.

O método de redução de multiplicações reorganiza o produto interno, passando o produto interno entre os pesos W_0 e W_1 e os pixels P_0 e P_1 a ser dado por:

$$PE = (W_0 + P_1) \times (W_1 + P_0) - W_0 \times W_1 - P_1 \times P_0$$

Apesar de ter mais multiplicações do que um produto interno, o produto $W_0 \times W_1$ pode ser pré-calculado e o produto $P_1 \times P_0$ é calculado apenas uma vez para todos os filtros.

Uma outra otimização considerada na implementação da LiteCNN consiste na maximização da utilização dos DSP e das LUT da FPGA. Um DSP permite um baixo consumo de energia, alta velocidade de processamento, possui um tamanho pequeno e permite flexibilidade no design do sistema. O DSP suporta diversas funções como multiplicação, multiplicação e acumulação (MAC), adição de três entradas, funções lógicas bit a bit, entre outras. O DSP configurado para multiplicação de acumulação torna-se bastante atrativo na implementação de CNN em FPGA tendo em conta os cálculos que são efetuados na CNN através do produto escalar. O DSP configurado para multiplicação de acumulação recebe quatro valores na entrada, soma dois deles e multiplica esse resultado ao terceiro valor na entrada, por fim ao resultado da multiplicação é adicionado o quarto valor de entrada do DSP. Um exemplo como a utilização do DSP é realizado na arquitetura tendo em conta também o acelerador do produto escalar está ilustrado na Figura 40.

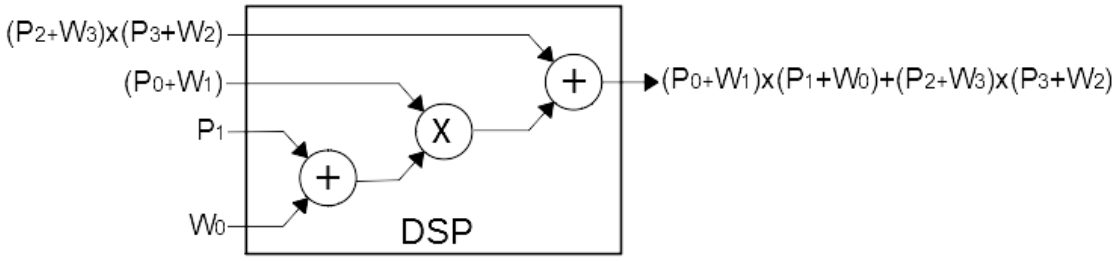

Figura 40 DSP configurado para multiplicação de acumulação.

Assim um DSP, caso tenha numa das entradas o resultado de uma multiplicação implementada com LUT, permite obter o resultado do produto escalar de 4 pesos diferentes com 4 pixels diferentes. Ao utilizar o máximo de DSP possíveis introduz na arquitetura paralelismo, pois é possível realizar vários produtos escalares ao mesmo tempo.

Utilizando esta técnica, cada PE implementa 8 multiplicações em paralelo para realizar o cálculo de dois produtos internos em paralelo com 8 parcelas cada um.

4.3. Arquitetura LiteCNN com Suporte a Redução de Dados

A arquitetura LiteCNN foi adaptada para dar suporte ao método de redução de dados através de *pruning* por blocos. A utilização do método *Low-Rank* não requer qualquer alteração da arquitetura pois trata-se apenas de adicionar convoluções de formatos diferentes, o que já é suportado pela arquitetura atual. A compressão de dados de Huffman não foi implementada nesta versão. A técnica de *pruning* é aplicada apenas às camadas totalmente conectadas.

A implementação do *pruning* foi realizada do seguinte modo:

- Os pesos reduzidos são lidos da memória e enviados para as memórias locais dos PE. Para além dos pesos, é armazenado o endereço do próximo bloco diferente de zero. Como, em geral, o número de *kernels* de uma camada é superior ao número de PE, os mapas de saída vão sendo calculados por fases, ou seja, é enviado um grupo de kernels, calculam-se os mapas de saída respetivos e depois envia-se o próximo grupo de *kernels*. Para melhorar o desempenho, os próximos pesos *kernels* são enviados enquanto são processados os atuais guardados nas memórias dos PE;
- As ativações da última camada guardadas no buffer de memória são enviadas para os PEs;
- Cada PE verifica através do endereço atual se corresponde a um bloco igual ou diferente de zero. Se for diferente de zero realiza o cálculo e avança para o próximo. Caso contrário, multiplica por 0 e mantém o endereço.

Neste método, não se reduz o número de cálculos, mas apenas a quantidade de pesos que é transferido da memória externa, pois apenas se transferem os pesos diferentes de zero. A implementação mantém-se eficiente, uma vez que o ponto crítico da execução das camadas totalmente conectadas está na transferência dos pesos e não no cálculo dos mesmos, como veremos pelos resultados. Além disso, é apenas calculado um produto interno em cada PE, pois nas camadas totalmente conectadas não é possível tirar proveito deste paralelismo (ao contrário do que se passa com as camadas de convolução em que a LiteCNN calcula dois em paralelo, podendo a arquitetura ser configurada para mais).

Uma possível otimização deste método seria não realizar o cálculo de multiplicação por zeros para reduzir o consumo de energia. No entanto, como neste trabalho não estamos a considerar esta métrica, esta otimização não foi implementada.

A principal alteração da LiteCNN foi no elemento de processamento, em que foi alterada a forma como é realizada a leitura da memória local e o envio dos pesos para o módulo Cell. Caso o endereço do mapa de entrada corresponda ao campo endereço do bloco de pesos, os são lidos da memória local, enviados para a célula de cálculo e o endereço da memória local é incrementado. Caso contrário, são enviados zeros para a célula e o endereço local mantém-se. Na Figura 41 estão ilustradas as ligações entre a memória e o *módulo Cell* do módulo PE da Figura 39.

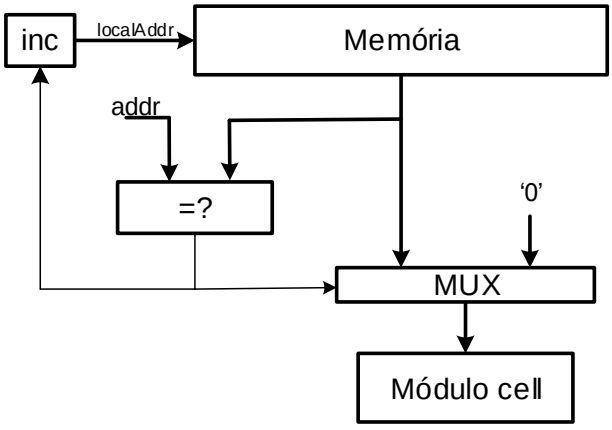

Figura 41 Alterações realizadas ao módulo PE para suportar o método de *pruning*.

Considerando estas alterações, obtivemos os resultados de área descritos na Tabela 19, referentes às células de cálculo dos produtos internos.

Número de bits (AtivaçãoxPeso)		16x16		8x8			4x4			
Tamanho de bloco		2	4	2	4	8	2	4	8	16
PE 0	LUTs	735	706	645	645	614	722	722	722	696
	DSP	7	7	4	4	4	4	4	4	4

	BRAM	2	2	1	1	1	1	1	1	1
PE 1	LUTs	1074	1046	756	756	696	804	804	804	772
	DSP	6	6	3	3	3	3	3	3	3
	BRAM	2	2	1	1	1	1	1	1	1
Nº de operações por PE (MACs)		16	16	16	16	16	32	32	32	32

Tabela 19 – Área dos PE da LiteCNN com suporte a redução de dados (*pruning*).

A Tabela 19 apresenta diferentes arquiteturas dos PE considerando 16, 8 e 4 bits para o tamanho dos pesos e ativações com diferentes tamanhos de blocos de *pruning*.

Na implementação 8x8 com tamanho de bloco de 2 e de 4, os recursos utilizados são os mesmos pois os blocos de pesos são guardados em memória acrescentados de zeros para perfazer o mesmo número de ativações (8) recebidas do mapa de entrada. O mesmo se passa na implementação 4x4 com blocos de tamanho 2, 4 e 8. Apesar de exigir mais memória local, o método simplifica bastante o armazenamento e a leitura alinhada dos blocos com dimensão inferior à dimensão dos blocos de ativações.

4.4. Modelo de Área e de Desempenho da LiteCNN

Para estimar a área da arquitetura em função da rede CNN foi desenvolvido um modelo de área genérico. Para este modelo é contabilizado o número de LUTs, de DSP e de BRAM para cada módulo da arquitetura LiteCNN com diferentes representações de dados para as ativações e para os pesos e diferentes tamanhos de blocos de *pruning*. O modelo de área genérico baseia-se nos valores obtidos após implementação das células (ver Tabela 19) e dos restantes blocos da arquitetura LiteCNN, representados na figura 37 (ver Tabela 20).

Número de bits (AtivaçãoxPeso)		16x16		8x8			4x4			
Tamanho de bloco		2	4	2	4	8	2	4	8	16
Memory Buffer	LUTs	882	882	775	775	775	774	774	774	774
	BRAM	72	72	70	70	70	70	70	70	70
Send Activations	LUTs	2784	2784	1133	1133	1133	860	860	860	860
Send Weights (x4)	LUTs	112	44	112	112	46	116	116	116	44
Bias Mem	LUTs	677	677	588	588	588	504	504	504	504
Receive Activations	LUTs	603	603	555	555	555	487	487	487	487
	BRAM	2	2	2	2	2	2	2	2	2
CnnControl	LUTs	28	28	55	55	55	52	52	52	52
PE Cluster (# PE)	PE 0	28	28	28	28	28	28	28	28	28
	PE 1	4	4	36	36	36	36	36	36	36

Total	LUTs	30298	29102	48830	48830	45538	52301	52301	52301	50133
	DSP	220	220	220	220	220	220	220	220	220
	BRAM	138	138	136	136	136	136	136	136	136
Nº total de PE		32	32	64	64	64	64	64	64	64
Nº de operações por PE (MACs)		16	16	16	16	16	32	32	32	32

Tabela 20 – Modelo de área da LiteCNN.

As BRAM referidas no modelo genérico correspondem a blocos de RAM de 36 Kbits. As leituras realizadas nos elementos de processamento da arquitetura que utiliza 16 bits para representar os pesos e as ativações são realizadas em grupos de 128 bits, enquanto que nas arquiteturas que utilizam 4 e 8 bits para representar os pesos e as ativações as leituras são feitas em grupos de 64 bits. Assim, tendo em conta o número de bits lidos por cada leitura dos PE o tamanho máximo do bloco para a arquitetura que utiliza 16 bits é 8, pois em 128 bits apenas se consegue ter 8 blocos de 16 bits. O tamanho máximo do bloco para a arquitetura que utiliza 8 bits é de 8, em 64 bits consegue-se obter no máximo 8 blocos de 8 bits, enquanto que na arquitetura que utiliza 4 bits o tamanho máximo do bloco é de 16, em 64 bits consegue-se obter 16 blocos de 4 bits.

Através do modelo de área verifica-se que quando o tamanho do bloco é máximo, o número de LUTs diminui. A utilização de DSP pelas arquiteturas é sempre máximo, o que permite obter paralelismo na arquitetura.

Para estimar o desempenho da arquitetura na execução de uma determinada CNN, é necessário saber o número de bytes que é necessário transferir entre a memória externa e a LiteCNN para cada uma das camadas e o número de ciclos necessários para realizar todos os cálculos de cada camada. Através destes dados é possível estimar o tempo de execução de uma inferência da CNN. Este valor depende das características da rede, nomeadamente, o número e o tipo de camadas, o tamanho e a quantidade de filtros, o tamanho da imagem de entrada e a redução de dados aplicada.

O número de ciclos das camadas convolucionais depende do tempo de transferência dos pesos de todos os filtros da camada convolucional e do tempo de processamento necessário para realizar todas as convoluções da camada convolucional. O tempo de transferência dos pesos depende do número de *kernels*, do tamanho dos *kernels* (convoluções 3D), do número de bits utilizados para representar os pesos, da largura de banda de acesso à memória externa, BW, onde estão armazenados os filtros e do tamanho dos blocos de *pruning*, *bSize*.

O número de bytes transferidos durante uma camada convolucional é dado por:

$$nº\ bytes = nº\ Kernels\ \times Tamanho\ do\ kernel\ \times \frac{nº\ bits}{8}$$

O número de bytes transferidos durante uma camada totalmente conectada é influenciado pela percentagem de corte utilizada pelo método de *pruning* e é dado por:

$$nº\ bytes = nº\ Kernels\ \times Tamanho\ do\ kernel\ \times \frac{nº\ bits}{8} \times \left(\frac{100 - \%\ corte}{100}\right) \times \frac{1 + bSize}{bSize}$$

Para além de percentagem de corte, o número de bytes a transferir depende ainda do tamanho do bloco de *pruning*, modelado pelo último fator de multiplicação da equação anterior. Este fator é maior do que um pois modela os bytes extra necessários para indicar o próximo bloco de dados diferente de zero.

O tempo que demora a transferir os bytes de cada camada está dependente da largura de banda utilizada na implementação. Assim, o tempo de transferência dos dados em cada camada é dado por:

$$Tempo\ de\ transferência = \frac{nº\ bytes}{Largura\ de\ Banda}$$

O número de ciclos que demora a processar uma camada convolucional está dependente do número de *kernels*, do tamanho da convolução (tamanho do *kernel*) e do número de convoluções, que consistem nas características da rede à qual o modelo de desempenho é aplicado, e do número de PE e do número de MAC realizado por PE, dado pelo modelo de área da arquitetura. Assim, o número de ciclos para processar uma camada convolucional é dado por:

$$Ciclos\ processamento\ = \frac{nº\ Kernels}{nº\ Cores} \times \frac{nº\ Convoluções \times Tamanho\ do\ kernel}{nº\ MACs}$$

O cálculo do número de ciclos necessários para processar uma camada totalmente conectada é semelhante ao do das camadas convolucionais. No entanto, o número de convoluções tem o valor de um nas camadas totalmente conectadas.

Enquanto que nas camadas convolucionais, o PE da LiteCNN calcula duas ativações do mapa de saída em paralelo, nas camadas totalmente conectadas tal não é possível, pois o *kernel* só é usado uma vez. Assim, o cálculo do número de ciclos das camadas totalmente conectadas é multiplicado pelo número de produtos internos paralelos suportado pelo PE,

nParallel, para corrigir o número total de MAC da arquitetura. Desta forma, o número de ciclos necessários para processar uma camada totalmente conectada é dado por:

$$\text{Ciclos de processamento} = \frac{nº\ Kernels}{nº\ Cores} \times \frac{tamanho\ do\ kernel}{nº\ MACs} \times nº\ Parallel$$

O tempo de atraso das camadas convolucionais corresponde à soma do tempo de transferência dos dados com o tempo de processamento da camada convolucional. O tempo de atraso das camadas totalmente conectadas corresponde ao maior tempo entre o tempo de transferência dos parâmetros das camadas totalmente conectadas e o tempo de processamento das camadas totalmente conectadas, uma vez que a transferência dos pesos ocorre em paralelo com o cálculo.

O desempenho depende do número de MAC realizados em cada camada, do número de ciclos necessários para processar a camada e da frequência utilizada na implementação. Para o desempenho ser dado em OPs (operações por segundo) é necessário multiplicar o número de MAC realizados por 2, pois um MAC corresponde a duas operações.

Assim, o desempenho é dado por:

$$Desempenho = \frac{nº\ MAC\ realizados\ \times 2}{nº\ ciclos} \times frequência$$

No próximo capítulo estão representados os modelos de desempenho aplicado à diferentes redes.

Resultados Obtidos

5

Neste projeto, utilizou-se o ambiente Caffe para estudar as quatro técnicas de compressão e redução de dados: *Low-Rank*, *pruning*, compressão com codificação de Huffman e compressão com formato de matrizes esparsas, para os seguintes modelos de CNN:

- LeNet
- CIFAR-10
- CIFAR-10 Quick (tem mais uma camada FC do que a CIFAR-10 e o modo de treino mais rápido)
- AlexNet (com a base de dados da CIFAR-10)

O estudo da AlexNet com a base de dados da CIFAR-10 deve-se à capacidade de processamento que é necessário para o treino e teste da rede com a base de dados da ImageNet. As imagens a cores de 227×227 levam à necessidade de ter mais espaço de memória, cerca de 236 Gbytes em imagens de treino, e mais capacidade de processamento, pois após a utilização da compressão é necessário verificar a sua precisão. Optou-se assim por utilizar a base de dados da CIFAR-10.

O *pruning* foi implementado para testar a precisão de cada camada totalmente conectada dos modelos anteriormente descritos, através da aplicação de cortes graduais de 10 % dos pesos até a um valor máximo de 95%. Desta forma, verificou-se a relação entre a precisão de cada camada para diferentes percentagens de corte. Por fim, aplicou-se cortes com diferentes percentagens para cada camada para maximizar essa relação.

A métrica aplicada em todos os testes é através da magnitude dos pesos, como anteriormente explicado. Contudo, pare se perceber melhor a influência dos pesos, aplicou-se o critério de corte através da sua amplitude. No geral, através desta métrica a precisão é bastante pior em relação à métrica da magnitude, concluindo-se que os pesos negativos com maior valor também são importantes na classificação. Os resultados foram obtidos para um teste com 10 mil imagens.

Em relação à compressão com codificação de Huffman e com representações de matrizes esparsas só são aplicadas juntamente com o *pruning.* Para maximizar a compressão dos algoritmos de codificação, os pesos foram quantizados de vírgula flutuante precisão simples (32 bits) para vírgula fixa dinâmica de 8 bits.

5.1. Resultados de *Pruning*

Foi aplicado a diferentes redes CNN o algoritmo de *pruning* desenvolvido, de forma a verificar o desempenho da rede depois de treinada para diferentes percentagens de pesos com o valor zero.

5.1.1. Rede Lenet

A LeNet é a rede que obtém melhores precisões, até 99,91%, em relação às restantes redes utilizadas neste projeto, pois trata-se de um problema mais de simples de classificação de caracteres. A rede é composta por duas camadas totalmente conectadas, a IP1 com 400 mil pesos e a IP2 com 5 mil pesos, ocupando um tamanho de 1,62 MBytes de um total de 1,7249 MBytes (ver Tabela 21).

Camada	Pesos	Precisão	Tamanho
IP1	400000	-	1.6 MB
IP2	5000	-	20 KB
Total	405000	-	1.62 MB
Toda da Rede	431235	99.01 %	1.725 MB

Tabela 21 – Rede da LeNet utilizada na prática.

A maioria dos pesos encontram-se na primeira camada totalmente conectada, o que torna a aplicação do *pruning* mais eficiente na redução do tamanho total da rede. Na Figura 42 está ilustrado a diferença entre o corte através da amplitude, a preto, e através da magnitude, a vermelho, para cada uma das camadas, ou seja, aplicando um corte a cada uma delas separadamente e aplicando um corte para ambas em simultâneo. A precisão em percentagem varia em função da percentagem de corte dos pesos que varia entre 10% e 95%.

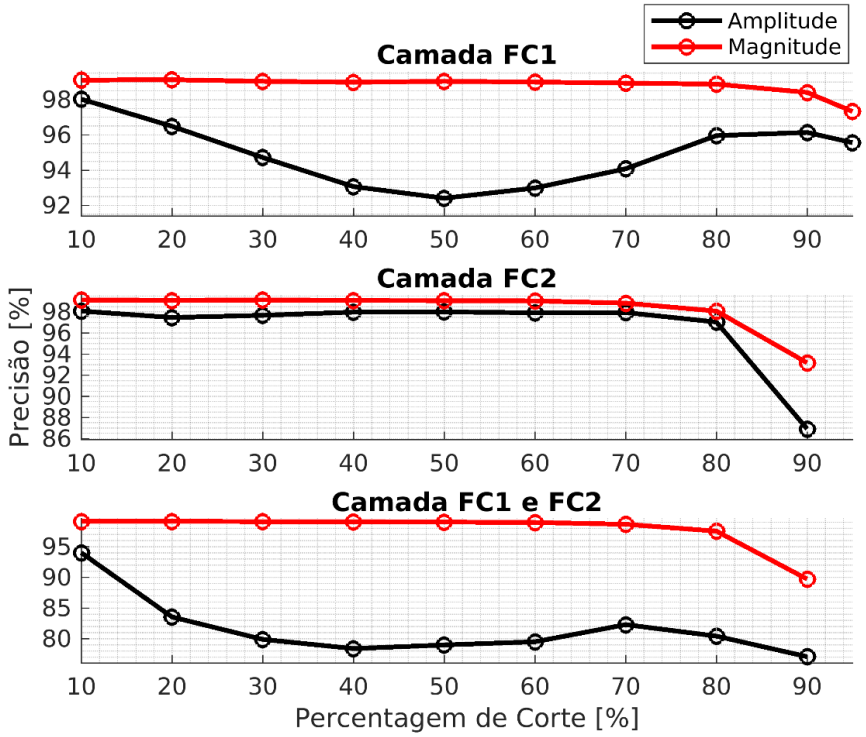

Figura 42 Amplitude v.s. magnitude na aplicação do *pruning* na LeNet.

Em relação à camada FC1, com um corte de 95% dos pesos através da amplitude tem uma precisão de aproximadamente 95% e através da magnitude tem uma precisão aproximadamente igual a 97%. Para a camada FC2 nota-se uma maior diferença de precisão a partir de um corte de 80%. Quando ambas as camadas são cortadas em simultâneo, a diferença de precisão entre métricas é bastante maior. A diferença de precisão entre métricas não é linear. Contudo, exceto para alguns valores, quanto maior o corte maior irá ser essa diferença.

Pode concluir-se que a técnica de corte com a métrica através da amplitude tem pior eficácia do que através da magnitude para a LeNet.

Na Figura 43 mostra a diferença de precisão entre camadas para o *pruning* com a magnitude como critério. A camada FC1 está representada a vermelho, a camada FC2 a azul e para as camadas em simultâneo está representado a preto.

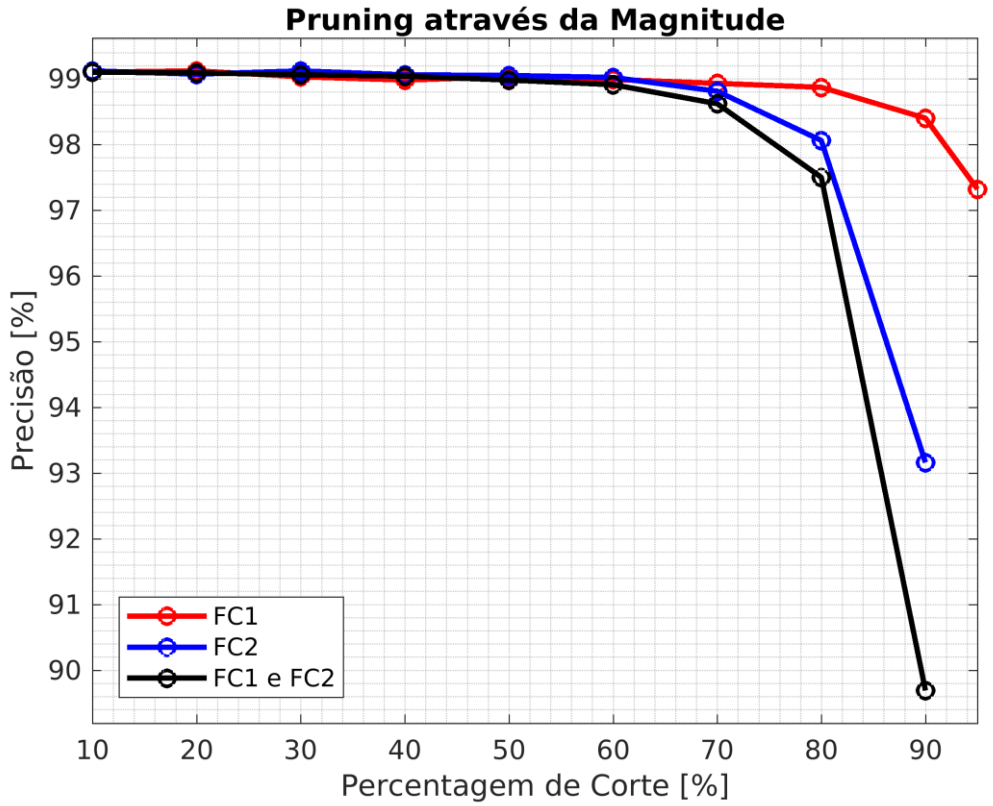

Figura 43 Diferença de precisão das camadas FC da LeNet.

Como mencionado anteriormente, as camadas mais profundas tendem a ter mais pesos, o que torna a rede com bastantes pesos redundantes, o que não fazem aumentar significativamente a precisão da rede. Para a LeNet, a camada FC1 tem um número mais elevado de pesos do que a camada FC2. Por isso, o corte para a camada FC1 tem melhor precisão quando a percentagem de corte é mais elevada. Por exemplo, com um corte de 90% na camada FC1 há uma precisão de 97,32%, enquanto que na camada FC2 há uma precisão de 93,16% e para ambas há uma precisão com 89,69%. O corte da camada FC2 influencia negativamente a precisão, pois com menos número de pesos existem menos pesos redundantes.

Na Figura 44 está representada a comparação da precisão em função do *pruning* para diferentes tamanhos de blocos. O bloco um, B1 a magenta, são os resultados obtidos através do corte sem agrupamento de pesos. O bloco com um agrupamento de dois pesos, B2, está representado a amarelo, o bloco com quatro, B4, está representado a verde, o bloco de oito, B8, a vermelho e o bloco de dezasseis, B16, a azul.

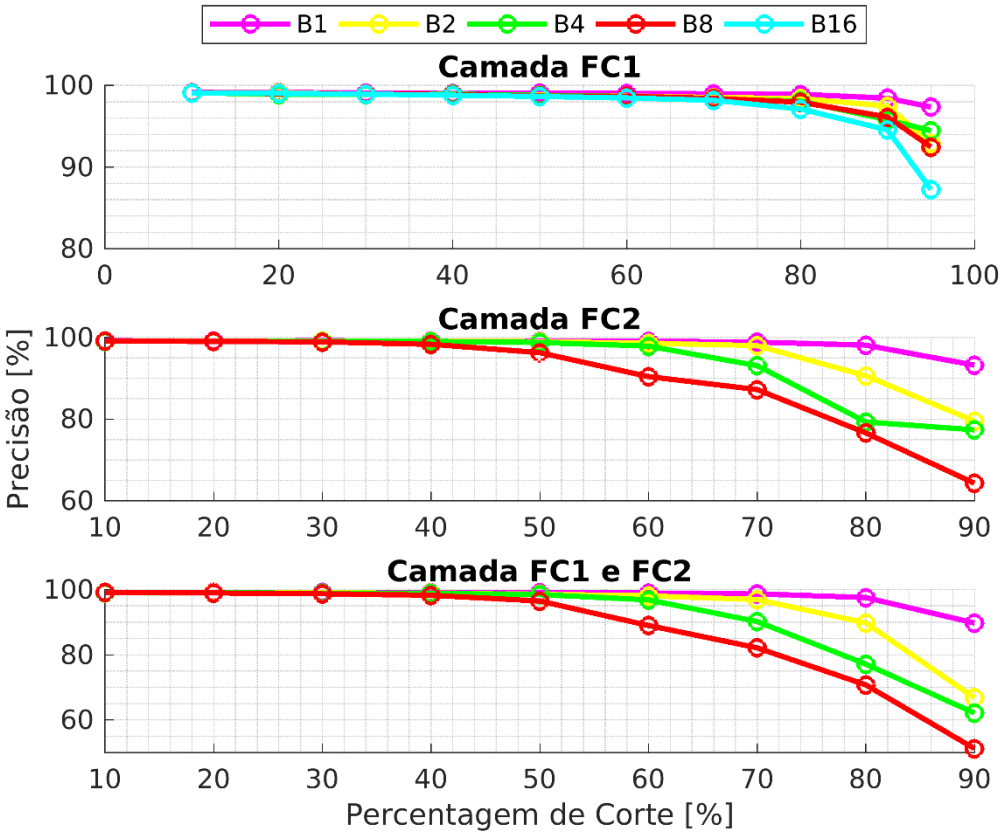

Figura 44 Diferença de precisão para diferentes tamanhos de blocos da LeNet.

Para a camada FC1 a precisão é muito semelhante para todos os blocos até aos 80% de corte da camada. Com 95% de corte, em FC1, a precisão do B16 é de 87,22%, uma redução de 8% na precisão.

O corte com o bloco de dezasseis não foi testado para a camada FC2, porque o número de pesos existente nesta camada não é múltiplo de 16, logo no final iria existir um bloco com menos pesos do que os 16. A solução poderia ser aumentar o número de pesos desta camada, contudo não é possível alterar o tamanho das camadas do ficheiro *caffemodel*. Como o objetivo é facilitar a leitura em *hardware*, ter um bloco diferente dos restantes torna-se pouco eficiente e em termos de precisão, a avaliar por os restantes resultados, o corte da camada FC2 com este bloco não iria ser tão eficiente. Para as três situações de corte pode verificar-se que com o aumento do tamanho dos blocos a precisão é mais baixa.

Na Figura 45 pode observar-se a variação da precisão em função da percentagem de corte para a camada FC1 e a diferença, para os diferentes blocos, entre vírgula flutuante 32 bits, a preto, e vírgula fixa 8 bits, a vermelho.

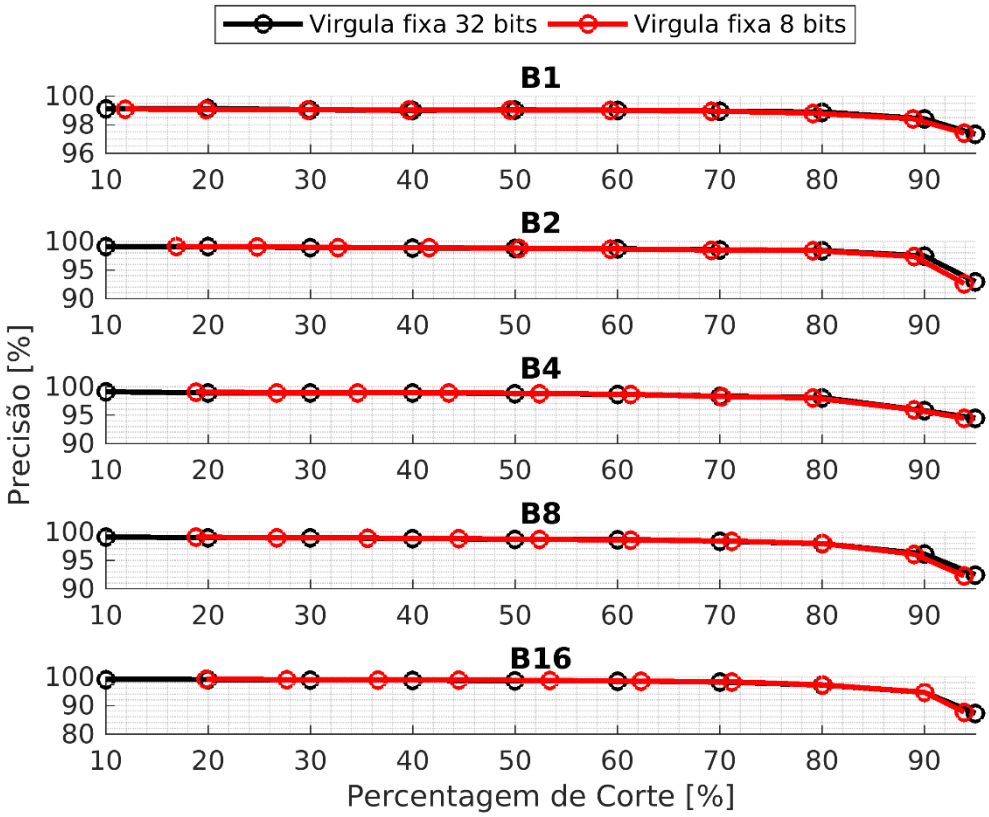

Figura 45 Vírgula flutuante 32 bits v.s. vírgula fixa 8 bits para diferentes blocos da FC1 da LeNet.

Ao aplicar menos número de bits para representar cada peso, muitos dos pesos ficam com o valor zero, pois, muitos desses valores são muito próximos de zero não sendo possível a sua representação com 8 bits. Assim, a percentagem de corte aumenta em comparação com a percentagem de corte sem quantização, como se pode verificar para todos os blocos, quando a percentagem de corte é mais baixa. Por exemplo, para o B16 aplicando um corte de 10% com quantização, há um corte de 20% na camada, ou seja, a quantização adiciona 10% de zeros. Quando a percentagem de corte é mais elevada esse fenómeno é praticamente inexistente.

Pode ainda verificar-se que a diferença de percentagem, com e sem quantização, é aproximadamente igual, sendo ligeiramente mais baixa quando aplicada a quantização.

A Tabela 22 representa a melhor relação entre precisão e percentagem de corte para cada um dos blocos sem e com quantização, no Anexo D – RESULTADOS LENET encontram-se resultados adicionais.

Tipo	IP1 [%]	IP2 [%]	# Corte	Corte [%]	Tamanho [KB]	Precisão [%]	# Erros
B1	95	50	382500	94.44	90	97.62	238
B2	90	50	362500	89.51	170	96.96	304

B4	90	10	360500	89.01	178	95.56	444
B8	90	0	360000	88.89	180	96.09	391
B16	90	0	360000	88.89	180	94.52	548
B1_Q	95	50	382500	94.44	22.5	97.66	234
B2_Q	90	51	362550	89.52	42.45	96.97	303
B4_Q	90	14	360700	89.06	44.3	95.58	442
B8_Q	90	6	360300	88.96	44.7	96.06	394
B16_Q	91	6	364300	89.95	40.7	94.55	545

Tabela 22 - Resultados com melhor relação entre precisão e percentagem de corte da LeNet.

A melhor solução para aplicar na FPGA é com quantização e um bloco de 4. Na FPGA a leitura de blocos de quatro pesos é possível implementar em todas as arquiteturas e a precisão entre blocos é muito semelhante. Com a quantização a redução do tamanho da camada é muito significativa. A diferença do tamanho das camadas FC aplicando *pruning* sem e com quantização é de aproximadamente $4{,}5\times$.

Através do modelo de desempenho proposto foi estimado, a partir das características da rede LeNet, o desempenho para diferentes configurações de redução considerando a LiteCNN configurada com 32 PEs para dados a 16×16 e 64 PEs para os restantes. O número de MAC por core é de 16, para dados a 16 e a 8 bits, e 32 para dados a 4 bits. A arquitetura funciona a 200 MHz (ver Tabela 23).

			Imagem	Conv1	Conv2	FC1	FC2	Total
Número de *Kernels*			0	20	50	500	10	-
Tamanho do *kernel* (2D)			0	5	5	4	1	-
Tamanho do *kernel* (3D)			-	75	25	800	500	-
Padding			-	0	0	0	0	-
Passo (*Stride)*			0	1	1	1	1	-
Largura do mapa de entrada			-	28	12	4	1	-
Comprimento do mapa de entrada			-	28	12	4	1	-
Profundidade do mapa de entrada			-	1	20	50	500	-
Largura do mapa de saída			28	24	8	1	1	-
Comprimento do mapa de saída			28	24	8	1	1	-
Profundidade do mapa de saída			1	20	50	500	10	-
Número de convoluções			-	576	64	1	1	642
Número de operações			-	864000	80000	400000	5000	1349000
16x16	Corte 10%	Bytes	1568	3000	2500	720000	9000	736068
		Tempo de transferência [ms]	0,0004	0,001	0,001	0,171	0,002	0,175
		Ciclos de processamento para Conv	-	1688	156	-	-	1844
		Ciclos de processamento para FC	-	-	-	1562	20	1582
		Atraso total [ms]	-	0,009	0,001	0,171	0,002	0,186
		Desempenho	-	2,05E+11	2,05E+11	1,02E+11	1,02E+11	2,05E+11

	Corte 90%	Bytes	1568	3000	2500	80000	1000	88068
		Tempo de transferência [ms]	0,0004	0,0007	0,0006	0,019	0,0002	0,021
		Ciclos para Conv	-	1688	156	-	-	1844
		Ciclos para FC	-	-	-	1562	20	1582
		Atraso total [ms]	-	0,009	0,001	0,019	0,0002	0,031
		Desempenho		2,05E+11	2,05E+11	1,02E+11	1,02E+11	2,05E+11
8x8	Corte 10%	Bytes	784	1500	1250	360000	4500	368034
		Tempo de transferência [ms]	0,0002	0,0004	0,0003	0,086	0,001	0,088
		Ciclos para Conv	-	844	78	-	-	922
		Ciclos para FC	-	-	-	781	10	791
		Atraso total [ms]	-	0,005	0,001	0,086	0,001	0,094
		Desempenho		4,09E+11	4,10E+11	2,05E+11	2,05E+11	4,10E+11
	Corte 90%	Bytes	784	1500	1250	40000	500	44034
		Tempo de transferência [ms]	0,0002	0,0004	0,0003	0,010	0,0001	0,010
		Ciclos para Conv	-	844	78	-	-	922
		Ciclos para FC	-	-	-	781	10	791
		Atraso total [ms]	-	0,005	0,001	0,010	0,0001	0,016
		Desempenho		4,095E+11	4,096E+11	2,048E+11	2,048E+11	4,10E+11
4x4	Corte 10%	Bytes	392	750	625	180000	2250	184017
		Tempo de transferência [ms]	0,0001	0,0002	0,0002	0,043	0,0005	0,044
		Ciclos para Conv	-	422	39	-	-	461
		Ciclos para FC	-	-	-	391	5	396
		Atraso total [ms]	-	0,002	0,0003	0,043	0,001	0,048
		Desempenho		8,19E+11	8,19E+11	4,10E+11	4,10E+11	8,19E+11
	Corte 90%	Bytes	392	750	625	20000	250	22017
		Tempo de transferência [ms]	0,0001	0,0002	0,0001	0,048	0,0001	0,005
		Ciclos para Conv	-	422	39	-	-	461
		Ciclos para FC	-	-	-	391	5	396
		Atraso total [ms]	-	0,002	0,0003	0,0048	0,0001	0,009
		Desempenho	-	8,19E+11	8,19E+11	4,10E+11	4,10E+11	8,19E+11

Tabela 23 - Desempenho estimado da LiteCNN para a rede LeNet.

Através do modelo de desempenho verifica-se que reduzir o número de bits para representar as ativações e os pesos permite aumentar o desempenho da arquitetura.

Com os resultados da Tabela 23 verifica-se também que a percentagem de corte aplicada influência apenas os tempos de transferência, visto que a arquitetura continua a realizar as multiplicações quando os pesos possuem o valor zero. Para a rede LeNet e utilizando 16 bits para representar as ativações e os pesos da rede, alterar a percentagem de corte de 10% para 90% leva a uma diminuição do atraso, em aproximadamente, 87%.

5.1.2. Rede CIFAR-10 Quick

A CIFAR-10 Quick é um modelo com um tempo de treino mais reduzido e com mais uma camada totalmente conectada, em relação à CIFAR-10. As duas camadas totalmente conectadas têm 66176 pesos, cerca de 265 KBytes, a camada FC1 tem 65536 pesos e a camada FC2 tem 640 pesos. A precisão desta rede é de apenas 54.02% e tem um tamanho total de 538,059 KBytes (ver Tabela 24).

Camada	Pesos	Precisão	Tamanho
IP1	65536	-	262.144 KB
IP2	640	-	2.560 KB
Total	66176	-	264.704 KB
Toda da Rede	145765	54.02 %	583.059 KB

Tabela 24 - Rede da CIFAR-10 Quick utilizada na prática.

A Figura 46 mostra a diferença de precisão entre o corte através da amplitude, a preto, e da magnitude, a vermelho, para cada camada e para as duas camadas em simultâneo.

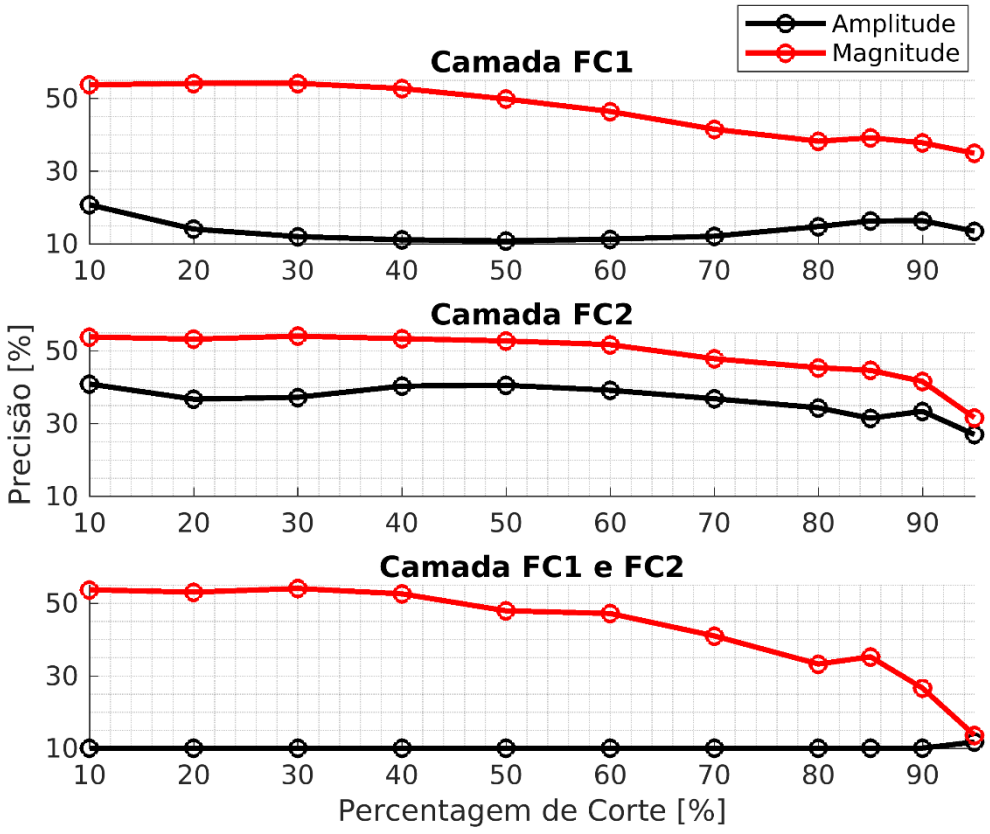

Figura 46 Amplitude v.s. Magnitude na aplicação do *pruning* na CIFAR-10 Quick.

Em relação à camada FC1 a diferença de precisão é bastante elevada, chegando a ser superior a 30% para algumas percentagens de corte, sendo melhor para o corte através da

magnitude. Para a camada FC2 a diferença é mais pequena, contudo, também é melhor para o corte através da magnitude. Com um corte de 95% nas duas camadas, a precisão aproximadamente igual para as duas métricas. Pode-se concluir, que para este modelo da CIFAR-10, a métrica da magnitude é melhor.

A Figura 47 representa a diferença de precisão para um corte através da magnitude da camada FC1, a vermelho, FC2, a azul, e para ambas, a preto.

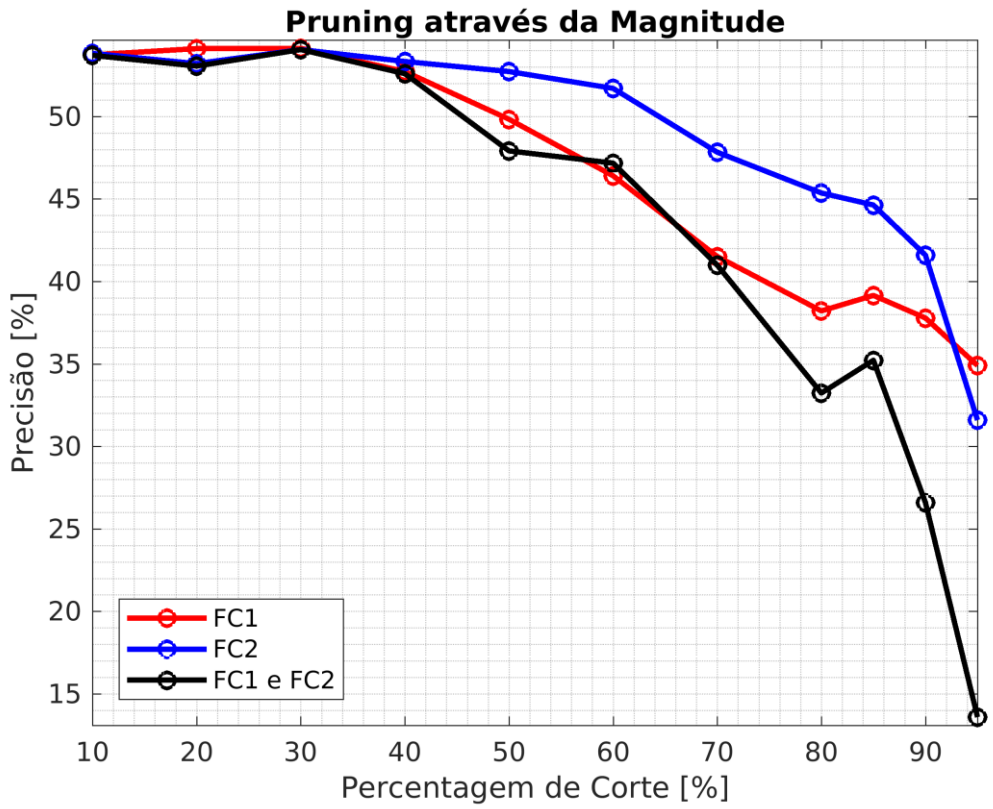

Figura 47 Diferença de precisão das camadas FC da CIFAR-10 Quick.

A camada FC1 tem cerca de 100x mais pesos do que a camada FC2. Contudo, só acima dos 92% de corte é que a precisão é melhor para a camada FC1. Teoricamente, como a camada FC1 tem mais pesos, deveria haver mais pesos redundantes nesta camada, a precisão deveria ser melhor para todas as percentagens de corte elevadas. Mesmo que no geral a camada FC2 tenha melhor precisão, o corte desta camada tem pouco impacto para a redução do tamanho da rede.

Na Figura 48 está ilustrado a diferença de precisão para o corte com diferentes tamanhos de blocos das duas camadas. O bloco um sem agrupamento de pesos, B1 a magenta, o bloco com um agrupamento de dois pesos, B2, está representado a amarelo, o bloco com quatro, B4, está representado a verde, o bloco de oito, B8, a vermelho e o bloco de dezasseis, B16, a azul.

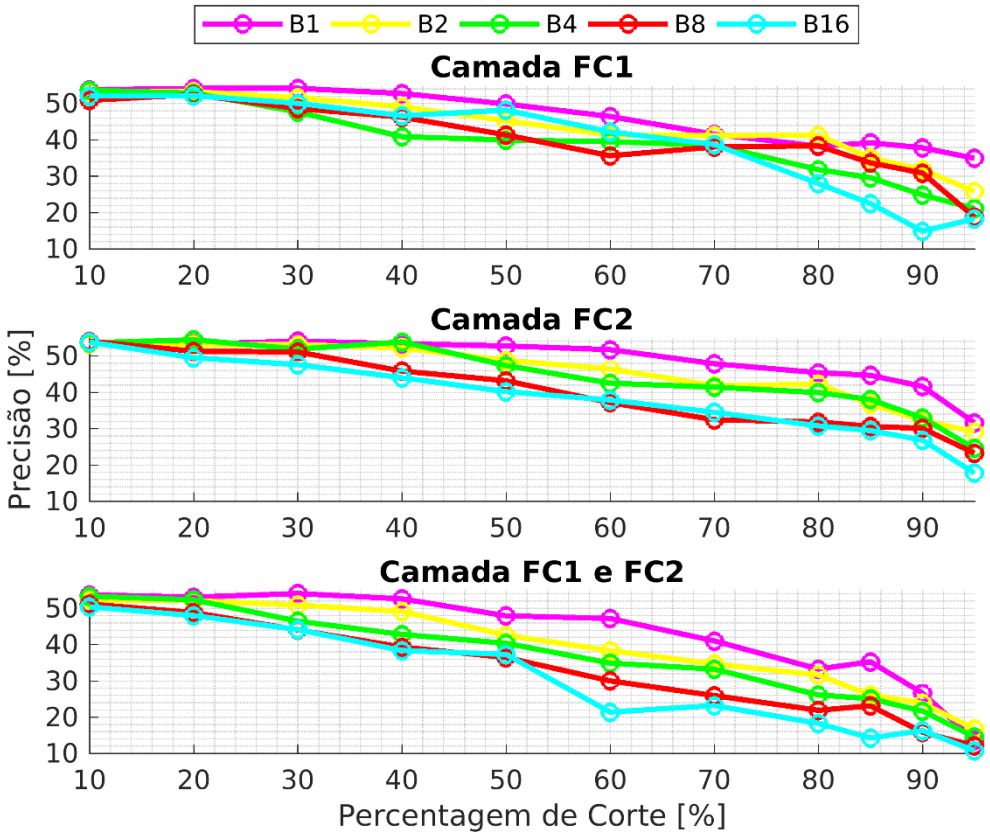

Figura 48 Diferença de precisão para diferentes tamanhos de blocos da CIFAR-10 Quick.

Teoricamente, quanto maior forem os blocos, menor irá ser a precisão, pode verificar-se este fenómeno para as duas camadas, especialmente quando a percentagem de corte é mais elevada. Ao contrário da LeNet, a CIFAR-10 Quick tem uma grande quebra na precisão à medida que a percentagem de corte aumenta, pode dever-se à precisão inicial ser bastante mais baixa. Para se conseguir uma precisão razoável, a percentagem de corte tem de ser inferior a 40%. Aplicando uma percentagem de corte apenas na camada FC1 (camada com mais pesos), com cerca de 50% de corte para os blocos B1, B2 e B4 consegue-se uma precisão razoável, apenas 5 % abaixo do conseguido sem cortes. A redução na rede é mais elevada aplicando apenas um corte de 50% na camada FC1 do que aplicando um corte de 40% nas duas camadas.

A Figura 49 mostra a diferença de precisão para a utilização da quantização, a vermelho está representa com vírgula flutuante de 32 bits e a preto com vírgula fixa de 8 bits, para a camada FC1 com diferentes tamanhos de blocos.

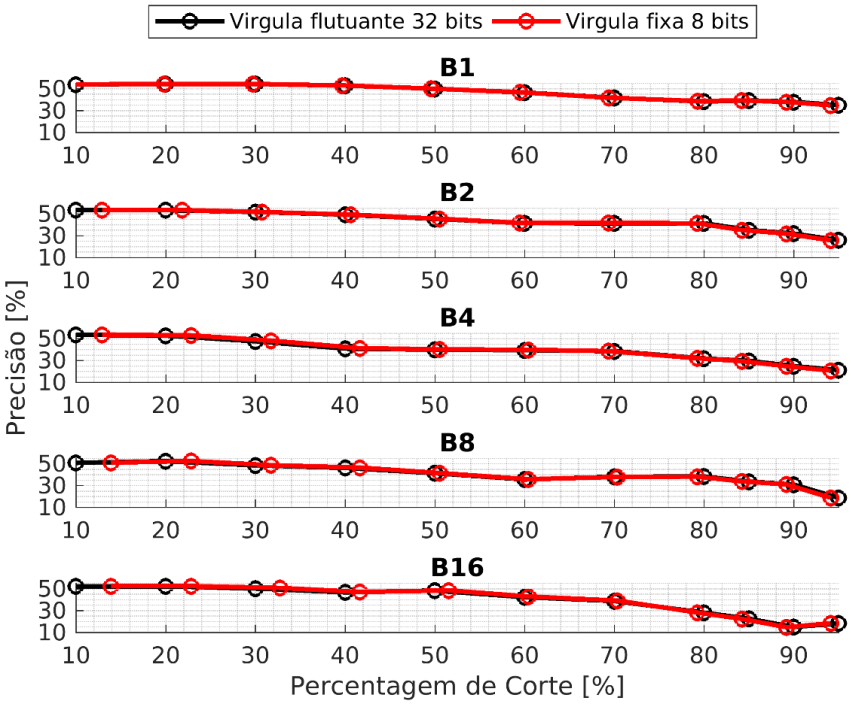

Figura 49 Vírgula flutuante 32 bits v.s. vírgula fixa 8 bits para diferentes blocos da FC1 da CIFAR-10 Quick.

Pode-se verificar que com a utilização da quantificação, a precisão não varia muito em relação à original (sem quantização), contudo, como anteriormente mencionado a redução da rede irá ser mais pequena em relação à LeNet, devido à reduzida precisão para grandes percentagens de corte.

A Tabela 25 representa a melhor relação entre a precisão e a percentagem de corte para cada um dos blocos sem e com quantização do modelo da CIFAR-10 Quick.

Tipo	IP1 [%]	IP2 [%]	# Corte	Corte [%]	Tamanho [KB]	Precisão [%]	# Erros
B1	40	40	26470	40	158.82	52.59	4741
B2	40	40	26470	40	158.82	49.13	5087
B4	20	20	13235	20	211.76	52.32	4768
B8	20	0	13107	19.8	212.28	52.4	4760
B16	20	0	13107	19.8	212.28	52.11	4789
B1_Q	40	40	26470	40	39.71	52.67	4733
B2_Q	41	40	27125	40.99	39.05	49.05	5095
B4_Q	23	4	15098	22.81	51.08	53.04	4696
B8_Q	23	4	15098	22.81	51.08	52.57	4743
B16_Q	33	4	21652	32.72	44.52	50.58	4942

Tabela 25 - Resultados com melhor relação entre precisão e percentagem de corte da CIFAR-10 Quick.

A partir de uma redução superior a 40% para ambas as camadas a precisão começa a decair bastante, cerca de menos 5% em relação à precisão original (ver ANEXO E – RESULTADOS CIFAR-10 QUICK). Para a utilização na FPGA a escolha recairia para o método de quantização e um bloco de 8 devido aos mesmos motivos anteriormente descritos. A redução do número de bits proporciona uma redução no tamanho das camadas de aproximadamente $4\times$.

O desempenho estimado calculado através do modelo de desempenho proposto e das características da rede CIFAR-10 Quick para diferentes configurações de redução está representado na Tabela 26. Considerou-se a mesma configuração da rede LiteCNN utilizada na execução da LeNet.

			Imagem	Conv1	Conv2	Conv3	FC1	FC2	Total
Número de *Kernels*			0	32	32	54	64	10	-
Tamanho dos *Kernels*			0	5	5	5	3,125	1	-
Conv_size			-	75	25	3200	527,34375	64	-
Padding			-	2	2	2	0	0	-
Passo (*Stride*)			0	1	1	1	1	1	-
Largura do mapa de entrada			-	32	15,5	7,25	3,125	1	-
Comprimento do mapa de entrada			-	32	15,5	7,25	3,125	1	-
Profundidade do mapa de entrada			-	3	32	32	54	64	-
Largura do mapa de saída			32	32	15,5	7,25	1	1	-
Comprimento do mapa de saída			32	32	15,5	7,25	1	1	-
Profundidade do mapa de saída			3	32	32	54	64	10	-
Número de convoluções			-	1024	240	53	1	1	1319
Número de operações			-	2457600	192200	9082800	33750	640	11766990
16x16	Corte 10%	Bytes	6144	4800	1600	345600	60750	1152	420046
		Tempo de transferência [ms]	0,001	0,001	0,0004	0,082	0,014	0,0003	0,1
		Ciclos de processamento para Conv	-	4800	375	17740	-	-	22916
		Ciclos de processamento para FC	-	-	-	-	132	3	134
		Atraso total [ms]	-	0,025	0,002	0,171	0,014	0,0003	0,219
		Desempenho	-	2,05E+11	2,05E+11	2,05E+11	1,02E+11	1,02E+11	2,05E+11
	Corte 90%	Bytes	6144	4800	1600	345600	6750	128	365022
		Tempo de transferência [s]	0,001	0,001	0,0004	0,082	0,002	0,00003	0,087
		Ciclos para Conv	-	4800	375	17740	-	-	22916
		Ciclos para FC	-	-	-	-	132	3	134
		Atraso total [ms]	-	0,025	0,002	0,171	0,002	0,00003	0,206
		Desempenho		2,05E+11	2,05E+11	2,05E+11	1,02E+11	1,02E+11	2,05E+11

8x8	Corte 10%	Bytes	3072	2400	800	172800	30375	576	210023
		Tempo de transferência [ms]	0,001	0,001	0,0002	0,041	0,007	0,0001	0,050
		Ciclos para Conv	-	2400	188	8870	-	-	11458
		Ciclos para FC	-	-	-	-	66	1	67
		Atraso total [ms]	-	0,013	0,001	0,085	0,007	0,0001	0,112
		Desempenho		4,10E+11	4,10E+11	4,10E+11	2,05E+11	2,05E+11	4,10E+11
	Corte 90%	Bytes	3072	2400	800	172800	3375	64	182511
		Tempo de transferência [ms]	0,001	0,001	0,0002	0,041	0,0008	0,00002	0,043
		Ciclos para Conv	-	2400	188	8870	-	-	11458
		Ciclos para FC	-	-	-	-	66	1	67
		Atraso total [ms]	-	0,013	0,001	0,085	0,001	0,00002	0,106
		Desempenho		4,096E+11	4,096E+11	4,096E+11	2,048E+11	2,048E+11	4,10E+11
4x4	Corte 10%	Bytes	1536	1200	400	86400	15188	288	105012
		Tempo de transferência [ms]	0,0004	0,0003	0,0001	0,020	0,036	0,0001	0,025
		Ciclos para Conv	-	1200	94	4435	-	-	5729
		Ciclos para FC	-	-	-	-	33	1	34
		Atraso total [ms]	-	0,0063	0,0006	0,043	0,0036	0,0001	0,059
		Desempenho		8,19E+11	8,19E+11	8,19E+11	4,10E+11	4,10E+11	8,19E+11
	Corte 90%	Bytes	1536	1200	400	86400	1688	32	91256
		Tempo de transferência [ms]	0,0004	0,0003	0,0001	0,020	0,0004	0,00008	0,022
		Ciclos para Conv	-	1200	94	4435	-	-	5729
		Ciclos para FC	-	-	-	-	33	1	34
		Atraso total [ms]	-	0,0063	0,0006	0,043	0,0004	0,00008	0,056
		Desempenho	-	8,19E+11	8,19E+11	8,19E+11	4,10E+11	4,10E+11	8,19E+11

Tabela 26 - Desempenho estimado da LiteCNN para a rede CIFAR-10 Quick.

Com o modelo de desempenho estimado para a rede Cifar-10 Quick, representado na Tabela 26, verifica-se que reduzir o número de bits para representar as ativações e os pesos da rede permite obter um melhor desempenho da arquitetura. Ao aumentar a percentagem de corte o número de bytes é reduzido, o que provoca uma diminuição do tempo de transferência e por consequente uma diminuição no atraso total da arquitetura. Para a rede Cifar10 Quick, utilizando 16 bits para representar as ativações e os pesos da rede, alterar a percentagem de corte de 10% para 50% permite reduzir o atraso total da arquitetura em aproximadamente, 7%.

Comparativamente com a rede LeNet a rede Cifar10 Quick possui menos pesos, o número de pesos é inferior em cerca de 66%, o que faz com que a percentagem de corte aplicada influencie mais a rede LeNet do que a rede Cifar10 Quick.

5.1.3. Rede CIFAR-10

A CIFAR-10 tem apenas uma camada totalmente conectada com aproximadamente 10 mil pesos. O tamanho da rede depois de treinada é de 359 KBytes, o que é bastante elevado em relação à camada FC1 que tem aproximadamente 41 KBytes, logo, a redução do tamanho da rede não vai ser muito elevado. As características desta rede estão representadas na Tabela 27, onde se verifica que tem uma precisão é de 71,88%.

Camada	Pesos	Precisão	Tamanho
IP1	10240	-	40.96 KB
Toda da Rede	89769	71.88	359.075 KB

Tabela 27 - Rede da CIFAR-10 utilizada na prática.

A Figura 50 compara a diferença da precisão em função da percentagem de corte através da amplitude, a preto, e através da magnitude, a vermelho.

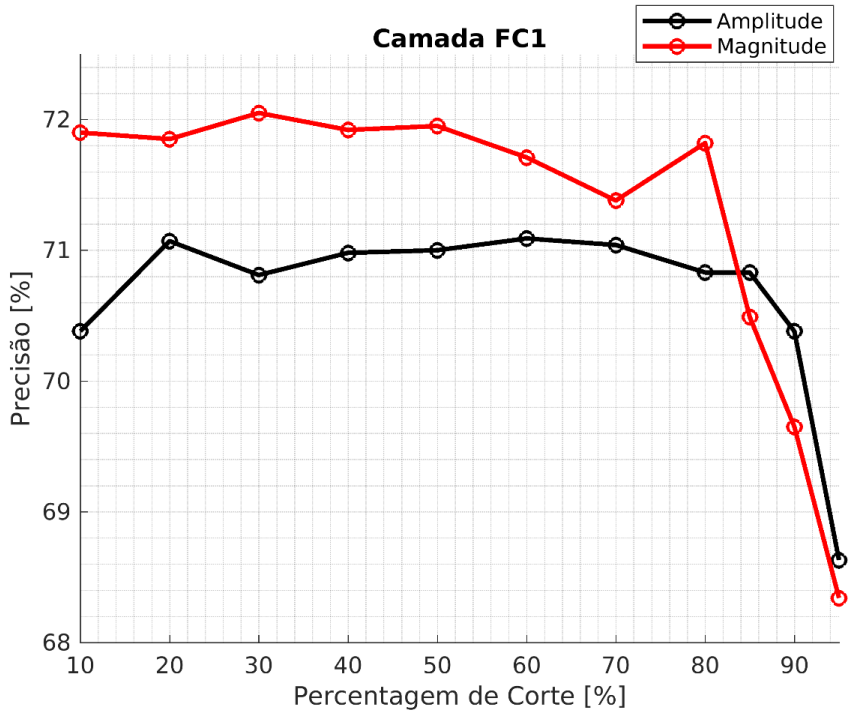

Figura 50 Amplitude v.s. Magnitude na aplicação do *pruning* na CIFAR-10.

A precisão com um corte através da magnitude é melhor em relação ao corte através da amplitude, até aproximadamente aos 84% de corte. Neste caso, para cortar um grande número de pesos é preferível utilizar a amplitude como métrica, contudo, os próximos resultados foram obtidos através da magnitude. A diferença da precisão original e da precisão com um corte de 95% através da magnitude é de 3,54%.

A Figura 51 representa a diferença da precisão para os diferentes tamanhos de blocos. O B1 está representado a magenta, o B2 a amarelo, o B4 a verde, o B8 a vermelho e o B16 a azul.

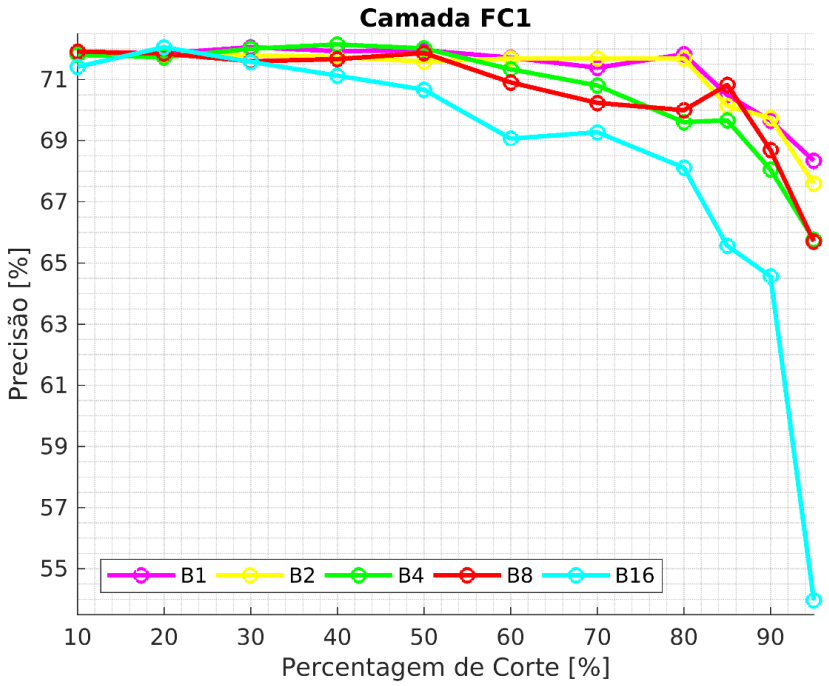

Figura 51 Diferença de precisão para diferentes tamanhos de blocos da CIFAR-10.

Pode verificar-se que para o bloco de 16 a diferença de precisão é muito elevada, sendo no pior caso igual a 53,96% com um corte de 95%. Em relação aos outros blocos a precisão não varia muito, contudo, entre os blocos B1, B2 e B4, B8 há uma diferença de aproximadamente 2% para 95% de corte.

A diferença da precisão com número de bits diferentes, com vírgula flutuante de 32 bits a preto e com vírgula fixa de 8 bits a vermelho, está ilustrado na Figura 52.

A quantização aplica um corte de 88%, ou seja, sem aplicar *pruning* há 88% de pesos a zero. O número de zeros aplicados pela quantização aumenta com o aumento do tamanho dos blocos. Para o B16 aplicando um corte de 95% após a quantização há um número de zeros total de 97,5%. Verifica-se ainda, para todos os blocos, que utilizando quantização a precisão não varia significativamente.

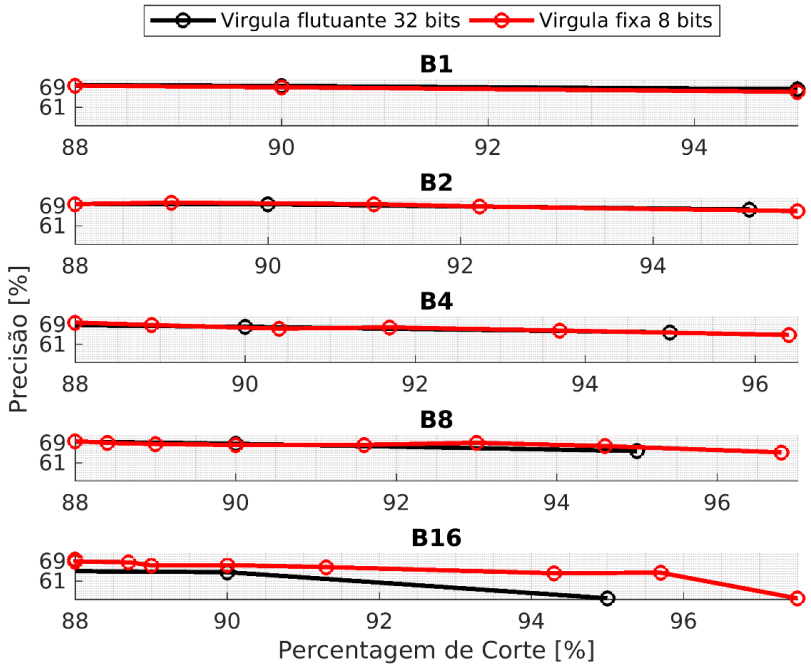

Figura 52 Vírgula flutuante 32 bits v.s. vírgula fixa 8 bits para diferentes blocos da FC1 da CIFAR-10.

A Tabela 28 representa a melhor relação entre a precisão e a percentagem de corte, do Anexo F – RESULTADOS CIFAR-10, para cada um dos blocos sem e com quantização do modelo da CIFAR-10.

Tipo	IP1 [%]	# Corte	Tamanho [KB]	Precisão [%]	# Erros
B1	95	9728	2.05	68.34	3166
B2	90	9216	4.1	69.73	3027
B4	90	9216	4.1	68.05	3195
B8	90	9216	4.1	68.69	3131
B16	80	8192	8.19	68.11	3189
B1_Q	90	9216	1.02	69.12	3088
B2_Q	92.2	9442	0.80	68.82	3118
B4_Q	91.7	9391	0.85	67.81	3219
B8_Q	94.6	9688	0.55	67.75	3225
B16_Q	90	9216	1.02	67.44	3256

Tabela 28 - Resultados com melhor relação entre precisão e percentagem de corte da CIFAR-10.

No melhor caso, com uma precisão de 67,75% obteve-se um tamanho para a camada totalmente conectada de aproximadamente 550 bytes. Este caso é com aplicação de quantização e um corte de 90% em blocos de oito pesos. O pior caso é com o bloco de dezasseis sem quantificação, apresentando um tamanho de 8,19 KBytes e uma precisão de 68,11%, que apesar de ser superior a outros casos, não justifica o tamanho da compressão aplicado.

Determinou-se o desempenho estimado a partir do modelo de desempenho proposto para diferentes configurações de redução e através das características da rede CIFAR-10 (ver Tabela 29).

			Ima-gem	Conv1	Conv2	Conv3	FC1	Total
Número de *Kernels*			0	32	32	64	10	-
Tamanho dos *Kernels*			0	5	5	5	3	-
Conv_size			-	75	25	3200	528	-
Padding			-	2	2	2	0	-
Passo (*Stride*)			0	1	1	1	1	-
Largura do mapa de entrada			-	32	16	7	3	-
Comprimento do mapa de entrada			-	32	16	7	3	-
Profundidade do mapa de entrada			-	3	32	32	54	-
Largura do mapa de saída			32	32	16	8	1	-
Comprimento do mapa de saída			32	32	16	8	1	-
Profundidade do mapa de saída			3	32	32	64	10	-
Número de convoluções			-	1024	240	52	1	1318
Número de operações			-	2457600	192200	10764800	5273	13419873
16x16	Corte 10%	Bytes	6144	4800	1600	409600	9493	431637
		Tempo de transfe-rência [ms]	0,001	0,001	0,000	0,098	0,002	0,103
		Ciclos de processa-mento para Conv	-	4800	375	21025	-	26200
		Ciclos de processa-mento para FC	-	-	-	-	21	21
		Atraso total [ms]	-	0,025	0,002	0,203	0,002	0,238
		Desempenho	-	2,05E+11	2,05E+11	2,05E+11	1,02E+11	2,05E+11
	Corte 90%	Bytes	6144	4800	1600	409600	1055	423199
		Tempo de transfe-rência [ms]	0,001	0,001	0,0004	0,098	0,0003	0,101
		Ciclos para Conv	-	4800	375	21025	-	26200
		Ciclos para FC	-	-	-	-	21	21
		Atraso total [ms]:	-	0,025	0,002	0,203	0,0003	0,236
		Desempenho		2,05E+11	2,05E+11	2,05E+11	1,02E+11	2,05E+11
8x8	Corte 10%	Bytes	3072	2400	800	204800	4747	215819
		Tempo de transfe-rência [ms]	0,001	0,001	0,0002	0,049	0,001	0,051
		Ciclos para Conv	-	2400	188	10512	-	13100
		Ciclos para FC	-	-	-	-	10	10
		Atraso total [ms]	-	0,013	0,001	0,101	0,001	0,122
		Desempenho		4,10E+11	4,10E+11	4,10E+11	2,05E+11	4,10E+11
	Corte	Bytes	3072	2400	800	204800	528	211600

	90%	Tempo de transfe-rência [ms]	0,001	0,001	0,0002	0,049	0,0001	0,050
		Ciclos para Conv	-	2400	188	10512	-	13100
		Ciclos para FC	-	-	-	-	10	10
		Atraso total [ms]	-	0,013	0,001	0,101	0,0001	0,121
		Desempenho		4,096E+11	4,096E+11	4,096E+11	2,048E+11	4,10E+11
4x4	Corte 10%	Bytes	1536	1200	400	102400	2374	107910
		Tempo de transfe-rência [ms]	0,0004	0,0003	0,0001	0,024	0,001	0,026
		Ciclos para Conv	-	1200	94	5256	-	6550
		Ciclos para FC	-	-	-	-	5	5
		Atraso total [ms]	-	0,006	0,001	0,051	0,001	0,064
		Desempenho		8,19E+11	8,19E+11	8,19E+11	4,10E+11	8,19E+11
	Corte 90%	Bytes	1536	1200	400	102400	264	105800
		Tempo de transfe-rência [ms]	0,0004	0,0003	0,0001	0,024	0,0001	0,025
		Ciclos para Conv	-	1200	94	5256	-	6550
		Ciclos para FC	-	-	-	-	5	5
		Atraso total [ms]	-	0,006	0,001	0,051	0,0001	0,063
		Desempenho	-	8,19E+11	8,19E+11	8,19E+11	4,10E+11	8,19E+11

Tabela 29 - Desempenho estimado da LiteCNN para a rede CIFAR-10.

Através dos resultados Tabela 29 verifica-se que, tal como acontece nas restantes redes, diminuir o número de bits para representar os pesos e as ativações da rede provoca um aumento no desempenho da arquitetura.

O aumento da percentagem de corte provoca uma diminuição no tempo de transferência e do atraso total. Para a rede Cifar-10, utilizando uma representação de 16 bits para os dados da rede, verifica-se que aumentar a percentagem de corte de 10% para 90% provoca uma diminuição no tempo de atraso de apenas 2%, isto porque a rede apenas possui uma camada totalmente conectada.

5.1.4. Rede AlexNet

A AlexNet foi treinada com a base de dados da CIFAR-10, por isso, teve de ser alterada em alguns parâmetros devido ao tamanho das imagens da CIFAR-10. Assim, esta rede depois de treinada fiou com menos pesos do que a rede da AlexNet original [13]. Este modelo tem três camadas convolucionais, a FC1 com aproximadamente 1,05 milhões de pesos, a FC2 com aproximadamente 16,7 milhões de pesos e a FC3 com aproximadamente 41 mil pesos, o que no total correspondem a 71,467 MBytes. A rede com todos os pesos tem 80,84 MBytes. A precisão com a utilização da base de dados da CIFAR-10 é só de 47,78%, o que

não favorece a aplicação dos métodos de compressão, obtendo-se resultados menos bons. A Tabela 30 mostra as especificações descritas.

Camada	Pesos	Precisão	Tamanho
FC1	1048576	-	4.194 MB
FC2	16777216	-	67.109 MB
FC3	40960		163.840 KB
Total	17866752	-	71.467 MB
Toda da Rede	20209388	47.78 %	80.838 MB

Tabela 30 - Rede da AlexNet utilizada na prática.

Aplicando quantização a este modelo a camada FC2 fica com os valores todos a zero, ou seja, os valores dos pesos são muito pequenos e não são possíveis representar com 8 bits. Pode dever-se também ao fator de treino, como o modelo não é otimizado para a base de dados da CIFAR-10 a sua precisão já é muito baixa, o que torna os valores dos pesos mais baixos. Assim, a precisão utilizando quantização para todos os blocos é bastante baixa sendo, para todos os blocos, abaixo dos 30% para diferentes percentagens de corte. Na Tabela 31 estão representados os resultados com melhor relação precisão e percentagem de corte.

Tipo	IP1 [%]	IP2 [%]	IP3 [%]	# Corte	Tamanho [MB]	Precisão [%]	# Erros
B1	80	90	10	15942452	7.697	46.07	5393
B2	80	90	10	15942452	7.697	44.73	5527
B4	50	90	10	15627879	8.96	44.03	5597
B8	0	90	0	15099494	11.069	43.86	5614
B16	50	90	10	15627879	8.96	40.7	5930
B1_Q	81	100	90	17663427	0.8133	26.3	7370
B2_Q	81	100	90	17663427	0.8133	19.06	8094
B4_Q	81	100	90	17663427	0.8133	26.03	7397
B8_Q	81	100	90	17663427	0.8133	17.84	8216
B16_Q	81	100	90	17663427	0.8133	19.34	8066

Tabela 31 - Resultados com melhor relação entre precisão e percentagem de corte da AlexNet.

Os restantes resultados obtidos da Tabela 31 estão ilustrados no

Anexo G – Resultados ALEXNET. A partir das características da rede AlexNet, determinou-se o desempenho estimado a partir do modelo de desempenho proposto para diferentes configurações de redução (ver Tabela 32).

			Imagem	Conv1	Conv2	Conv3	Conv4	Conv5	FC1	FC2	FC3	Total
Número de *Kernels*			0	96	256	384	384	256	4096	4096	10	-
Tamanho dos *Kernels*			0	11	5	3	3	3	1,03125	1	1	-
Conv_size			-	363	1200	1152	1728	1728	272	4096	4096	-
Padding			-	4	2	1	1	1	0	0	0	-
Passo (*Stride*)			0	4	1	1	1	1	1	1	1	-
Largura do mapa de entrada			-	32	4	2	2	2	1	1	1	-
Comprimento do mapa de entrada			-	32	4	2	2	2	1	1	1	-
Profundidade do mapa de entrada			-	3	96	256	384	384	256	4096	4096	-
Largura do mapa de saída			32	8	4	2	2	2	1	1	1	-
Comprimento do mapa de saída			32	8	4	2	2	2	1	1	1	-
Profundidade do mapa de saída			3	96	256	384	384	256	4096	4096	10	-
Número de convoluções			-	68	17	4	4	4	1	1	1	100
Número de operações			-	2371842	5227200	1881792	2822688	1881792	1115136	16777216	40960	32118626
16x16	Corte 10%	Bytes	6144	69696	614400	884736	1327104	884736	2007245	30198989	73728	36066778
		Tempo de transferência [ms]	0,001	0,017	0,146	0,211	0,316	0,211	0,478	7,190	0,018	8,587
		Ciclos de processamento para Conv	-	4633	10209	3675	5513	3675	-	-	-	27706
		Ciclos de processamento para FC	-	-	-	-	-	-	4356	65536	160	70052
		Atraso total [ms]	-	0,040	0,197	0,229	0,344	0,229	0,478	7,190	0,018	8,730
		Desempenho	-	2,05E+11	2,05E+11	2,05E+11	2,05E+11	2,05E+11	1,02E+11	1,02E+11	1,02E+11	2,05E+11
	Corte 90%	Bytes	6144	69696	614400	884736	1327104	884736	223028	3355444	8192	7373480
		Tempo de transferência [ms]	0,001	0,017	0,146	0,211	0,316	0,211	0,053	0,799	0,002	1,756
		Ciclos para Conv	-	4633	10209	3675	5513	3675	-	-	-	27706
		Ciclos para FC	-	-	-	-	-	-	4356	65536	160	70052
		Atraso total [ms]	-	0,040	0,197	0,229	0,344	0,229	0,053	0,799	0,002	1,899
		Desempenho		2,05E+11	2,05E+11	2,05E+11	2,05E+11	2,05E+11	1,02E+11	1,02E+11	1,02E+11	2,05E+11
8x8	Corte 10%	Bytes	3072	34848	307200	442368	663552	442368	1003623	15099495	36864	18033390
		Tempo de transferência [ms]	0,001	0,008	0,073	0,105	0,158	0,105	0,239	3,595	0,009	4,294
		Ciclos para Conv	-	2317	5104	1837	2756	1838	-	-	-	13854

		Ciclos para FC	-	-	-	-	-	-	2178	32768	80	35026
		Atraso total [ms]	-	0,020	0,099	0,115	0,172	0,115	0,239	3,595	0,009	4,368
		Desempenho		4,09E+11	4,10E+11	4,10E+11	4,10E+11	4,10E+11	2,05E+11	2,05E+11	2,05E+11	4,10E+11
	Corte 90%	Bytes	3072	34848	307200	442368	663552	442368	111514	1677722	4096	3686740
		Tempo de transferência [ms]	0,001	0,008	0,073	0,105	0,158	0,105	0,027	0,399	0,001	0,878
		Ciclos para Conv	-	2317	5105	1838	2756	1838	-	-	-	13854
		Ciclos para FC	-	-	-	-	-	-	2178	32768	80	35026
		Atraso total [ms]	-	0,020	0,099	0,115	0,172	0,115	0,027	0,399	0,001	0,952
		Desempenho		4,095E+11	4,096E+11	4,096E+11	4,096E+11	4,096E+11	2,048E+11	2,048E+11	2,048E+11	4,10E+11
4x4	Corte 10%	Bytes	1536	17424	153600	221184	331776	221184	501812	7549748	18432	9016696
		Tempo de transferência [ms]	0,000	0,004	0,037	0,053	0,079	0,053	0,119	1,798	0,004	2,147
		Ciclos para Conv	-	1159	2552	919	1378	919	-	-	-	6927
		Ciclos para FC	-	-	-	-	-	-	1089	16384	40	17513
		Atraso total [ms]	-	0,010	0,049	0,057	0,086	0,057	0,119	1,798	0,004	2,187
		Desempenho		8,19E+11	8,19E+11	8,19E+11	8,19E+11	8,19E+11	4,10E+11	4,10E+11	4,10E+11	8,19E+11
	Corte 90%	Bytes	1536	17424	153600	221184	331776	221184	55757	838861	2048	1843370
		Tempo de transferência [ms]	0,000	0,004	0,037	0,053	0,079	0,053	0,013	0,200	0,000	0,439
		Ciclos para Conv	-	1159	2552	919	1378	919	-	-	-	6927
		Ciclos para FC	-	-	-	-	-	-	1089	16384	40	17513
		Atraso total [ms]	-	0,010	0,049	0,057	0,086	0,057	0,013	0,200	0,000	0,479
		Desempenho	-	8,19E+11	8,19E+11	8,19E+11	8,19E+11	8,19E+11	4,10E+11	4,10E+11	4,10E+11	8,19E+11

Tabela 32 - Desempenho estimado da LiteCNN para a rede AlexNet.

Com o modelo de desempenho aplicado à rede AlexNet verifica-se que reduzir o número de bits para representar as ativações e os pesos para metade, faz reduzir para metade o número de bytes, o tempo de transferência e o tempo de atraso, o que permite aumentar o desempenho da arquitetura.

Em relação à percentagem de corte aplicado no método de *pruning* verifica-se através do modelo de desempenho que quanto maior a percentagem de corte menor é o número de bytes, o tempo de transferência e o tempo de atraso. Através do modelo de desempenho verifica-se que quando maior o número de bits para representar as ativações e os pesos maior a influência da percentagem de corte no atraso total da arquitetura, sendo que na arquitetura que utiliza 16 bits alterar a percentagem de corte de 10% para 90% permite reduzir o atraso total em, aproximadamente, 79%.

5.2. Resultados das Redes com Métodos de Compressão

Os próximos resultados foram obtidos com a aplicação da codificação de Huffman e da compressão de matrizes esparsas (ver Tabela 33) aos resultados com os pesos quantizados das Tabela 22, Tabela 25 e Tabela 28.

Rede	Tipo	Huffman	Matrizes Esparsas
LeNet	B1_Q	61.094 KB	25.313 KB
	B2_Q	71.601 KB	47.756 KB
	B4_Q	74.766 KB	49.838 KB
	B8_Q	76.128 KB	50.288 KB
	B16_Q	73.309 KB	45.788 KB
CIFAR-10 Quick	B1_Q	38.553 KB	44.669 KB
	B2_Q	40.162 KB	43.932 KB
	B4_Q	47.677 KB	57.463 KB
	B8_Q	48.290 KB	57.463 KB
	B16_Q	46.252 KB	50.09 KB
CIFAR-10	B1_Q	1.663 KB	1.152 KB
	B2_Q	1.596 KB	0.898 KB
	B4_Q	1.550 KB	0.955 KB
	B8_Q	1.576 KB	0.621 KB
	B16_Q	1.625 KB	1.152 KB

Tabela 33 – Codificação de Huffman v.s. compressão de matrizes esparsas.

O tamanho para cada uma das técnicas de compressão representa o tamanho das camadas totalmente conectadas comprimidas. A codificação de Huffman tem uma menor taxa de compressão em relação à compressão de matrizes esparsas para os modelos da LeNet e CIFAR-10. Em relação à CIFAR-10 Quick, a utilização da compressão de Huffman tem melhores taxas de compressão. No geral, a utilização da compressão de matrizes esparsas é melhor, por se obter melhores taxas de compressão e por a complexidade computacional do algoritmo ser também mais simples.

5.3. Resultados das Redes com o Método *Low-Rank*

A técnica de compressão *Low-Rank* obteve resultados abaixo dos publicados [23]. A Tabela 34 mostra para os quatro modelos de CNN a precisão obtida, o tempo de treino e o tamanho do ficheiro *caffemodel* para diferentes valores de *num_outputs* (K).

Rede	K	Precisão	Tempo	Tamanho
LeNet	-	99.01 %	2.12 min	1.725 MB
	1, 1	99.16 %	2.5 min	1.588 MB
	32, 32	99.92 %	3 min	1.643 MB
CIFAR-10	-	78.77 %	6.14 h	359 KB
	32, 32, 32	9.37 %	6.41 h	165 KB
	1, 64, 1	9.37 %	6.09 h	124 KB
	1, 1, 64	9.37 %	5.95 h	165 KB
	1, 32, 1	9.37 %	5.84 h	84 KB
	32, 1, 1	9.37 %	6.20 h	68 KB
CIFAR-10 Quick	-	52.02 %	3.38 min	583 KB
	32, 32, 64	10 %	5.48 min	546 KB
	64, 64, 64	10 %	8.36 min	687 KB
	1, 1, 64	10 %	3.06 min	409 KB
	1, 32, 1	10 %	4.61 min	406 KB
	32, 1, 1	10 %	4.08 min	311 KB
	1, 1, 1	10 %	2.90 min	289 KB
AlexNet	-	47.78 %	14.45 h	80.838 MB
	96, 256, 384, 384, 256	10 %	37.81 h	89.2 MB

Tabela 34 – Resultados para diferentes redes da aplicação da técnica L*ow-Rank*.

Apenas para a LeNet é que se obtiveram bons resultados com a aplicação da técnica, com uma precisão igual a 99,16% e uma redução para 1,588 MB do ficheiro *caffemodel*. A compressão não é muito elevada, pois as camadas FC têm muitos mais pesos do que as camadas convolucionais ao contrário dos modelos da CIFAR-10. Para estes modelos a compressão é maior, contudo a variação do *num_outputs* não influencia a precisão, sendo igual para todos os casos. Pode-se ainda verificar que com a diminuição do K a compressão é maior e o tempo de treino também.

Conclusões e Trabalho Futuro

6

As redes neuronais convolucionais são modelos com inteligência artificial, compostas por múltiplas camadas convolucionais, que quando treinadas com um conjunto de dados aprendem a classificar dados similares.

Muitos dos modelos CNN já existentes necessitam de uma grande capacidade computacional. Para reduzir esse impacto são aplicadas um conjunto de técnicas que permitem reduzir o consumo de energia e melhorar os tempos de inferência sem comprometer a precisão da rede, principalmente para sistemas embebidos.

O *pruning* é uma técnica de compressão muito eficaz, pois reduz muito o tamanho das redes com um impacto reduzido na precisão das redes. Com o auxílio das técnicas de codificação de dados a compressão é ainda mais eficaz. Com a compressão de matrizes esparsas obteve-se melhores do que a codificação de Huffman, por se obter melhores taxas de compressão e por a complexidade computacional do algoritmo ser também mais simples.

Verificou-se que com a utilização da métrica da amplitude que a precisão era pior comparativamente com a utilização da métrica da magnitude, concluindo-se que os pesos negativos com maior valor também são importantes na classificação.

As camadas com mais pesos tendem a ter mais pesos redundantes, o que não adicionam conhecimento à rede, logo a percentagem de corte aplicada a essas camadas pode ser maior, com minimização da degradação da precisão. O tamanho dos blocos no método de *pruning* proposto neste trabalho também influencia a precisão, pois quanto maior os blocos menor irá ser a precisão.

A arquitetura LiteCNN é um modelo para correr modelos de CNN em FPGAs de baixo custo. Através da implementação dos aceleradores de CNN para a FPGA obtiveram-se resultados de desempenho muito bons, tendo em conta que foi implementada numa FPGA de baixo custo, que permitem a execução das CNN em tempo-real.

A técnica de redução dos filtros *Low-Rank*, foi a que obteve resultados menos bons, tendo em conta os resultados publicados anteriormente. A redução dos pesos das camadas convolucionais com esta técnica é importante, principalmente em redes como a CIFAR-10 que tem mais pesos nessas camadas.

Como trabalho futuro, pretende-se estudar melhor a técnica de *Low-Rank*, pois permite reduzir a complexidade das camadas convolucionais.

Uma outra proposta de estudo é complementar as técnicas de quantificação não linear estudadas neste trabalho com técnicas de quantificação linear, com a redução do número de bits de representação dos dados, e estabelecer a correlação entre estes métodos.

Em relação à arquitetura LiteCNN pretende-se integrar uma das técnicas de compressão de dados para reduzir ainda mais a comunicação de pesos vindos da memória externa.

Referências

[1] Zynq-7000 All Programmable SoC Family Product Tables and Product Selection Guide, Xilinx.

[2] Mário Véstias, José Teixeira, Rui Duarte, Horácio Neto, “Lite-CNN: A High-Performance Architecture to Run CNNs in Low Density FPGAs”, FPL 2018.

[3] Axel Angel, “Towards Distorcion-Predictable Embedding of Neural Networks”, arXiv:1508.00102, June 2015.

[4] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey”, Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Nov. 2017.

[5] Yann Le Cun, John S. Denker, Sara A. Solla, “Optimal Brain Damage”, Advances in neural information processing systems 2, pp. 598-605, Jun. 1990.

[6] https://www.mathworks.com/products/neural-network.html

[7] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions", arXiv:1605.02688, May 2016.

[8] Martín Abadi, et al, "TensorFlow: Large-scale machine learning on heterogeneous systems", arXiv:1603.04467, May 2015.

[9] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In Proceedings of the 22nd ACM international conference on Multimedia, pp. 675-678, 2014.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", International Journal of Computer Vision, vol. 115, no. 3, pp. 211-252, Apr. 2015.

[11] George A. Miller, "WordNet: A Lexical Database for English", Communications of the ACM, vol. 38, no. 11, pp. 39-41, 1995.

[12] Amazon Mechanical Turk: API Reference, Amazon Web Services, 2018.

[13] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, "ImageNet classification with deep convolutional neural networks", NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems, vol. 1, pp. 1097-1105, Dec. 2012.

[14] Jianming Zhang, Shugao Ma, Mehrnoosh Sameki, Stan Sclaroff, Margrit Betke, Zhe Lin, Xiaohui Shen, Brian Price and Radomr Mech, “Salient Object Subitizing”, IEEE Conference on Computer Vision and Pattern Recognition, Oct. 2015.

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich, “Going deeper with convolutions”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Oct. 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, “Deep Residual Learning for Image Recognition”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Dec 2016.

[17] Yann LeCun, Lon Bottou, Yosgua Bengio and Pattrick Haffner, “Gradient-Bases Learning Applied to Document Recognition”, Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov 1998.

[18] Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical report, Cap. 3, University of Toronto, Apr. 2009.

[19] Guan Y., Xu N., Zhang C., Yuan Z., Cong J. “Using Data Compression for Optimizing FPGA-Based Convolutional Neural Network Accelerators” In: Dou Y., Lin H., Sun G., Wu J., Heras D., Bougé L. (eds) Advanced Parallel Processing Technologies. APPT 2017. Lecture Notes in Computer Science, vol 10561. Springer.

[20] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz, “Pruning Convolutional Neural Networks for arXiv:1611.06440, Jun. 2017.

[21] Xin Li, and Changsong Liu, “Prune the Convolutional Neural Networks with Sparse Shrink”, arXiv:1708.02439, Aug 2017.

[22] Song Han, Huizi Mao, William J. Dally, “Deep Compression: Compression Deep Neural Networks with Pruning Trained Quantization and Huffman Coding”, arXiv:1510.00149, Feb 2016.

[23] Cheng Tai, Tong Xiaom Yi Zhangm Xiaogang Wang, and Weinam E, “Convolutional Neural Networks with Low-Rank Regularization”, arXiv:1511.06067, Feb 2016.

[24] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, François Berry, "Accelerating CNN inference on FPGAs: A Survey", arXiv:1806.01683, May 2018.

Anexos

Anexo A - Algoritmo *Pruning*

```
def pruning_block(model_file, in_caffemodel, out_caffemodel, layers, ratio, array_len) :

    caffe.set_mode_cpu()
    net = caffe.Net(model_file, in_caffemodel, caffe.TEST)

    nnz_after = np.zeros(len(layers))
    nnz_after_prob =  np.zeros(len(layers))

    for l in range(0, len(layers) ) :

        name = layers[l]
        print(''); print('Layer: ' + name)

        temp = net.params[name][0].data
        temp2 = np.zeros(net.params[name][0].shape, np.float32)
        np.copyto(temp2, temp)

        nnz_before = np.sum(temp2 != 0)             # Total number of no zeros
        rt = ratio[l]*nnz_before                    # Total number of weights to cut
        rt = int(rt/array_len)
        total_size = temp2.shape[0]*temp2.shape[1]  # Size total of array
        temp2 = np.reshape(temp, total_size)        # reshape array 2D to 1D

        data = np.zeros(int(len(temp2)/array_len))

        if ratio[l] != 0.0 :
            i = 0
            for idx in range(0, len(temp2), array_len) :
                aux = temp2[idx:idx+array_len]
                data[i] = np.mean(abs(aux))
                i += 1

            idx_data = np.argsort(data)               # sort idx array
            for idx in range(0, rt) :

                i = idx_data[idx]
                i = i*array_len
                temp2[i:i+array_len] = 0

        #temp2 = np.round(temp2, 2)
        temp2 = np.reshape(temp2, temp.shape)
        np.copyto(net.params[name][0].data, temp2)
        nnz_after[l] = np.sum(temp != 0)
        nnz_after_prob[l] =  np.round((np.sum(temp != 0)/nnz_before), 2)

        print("# of non-zero (before): ", nnz_before)
        print("# of non-zero (after) : ", nnz_before-nnz_after[l],
              '(', 1-nnz_after_prob[l], ')')

    net.save(out_caffemodel)
```

Anexo B – Algoritmo Huffman

```
class stru:
    def __init__(self, symbol, noccu, bit):
        self.symbol = symbol
        self.noccu = noccu
        self.bit = bit

def ord_array(xa) :
    alterado=0
    for i in range(1, len(xa)) :
       if xa[i].noccu > xa[i-1].noccu :
            aux = xa[i]
            xa[i] = xa[i-1]
            xa[i-1] = aux
            alterado = 1
    if alterado == 1 :
        xa = ord_array(xa)
    return xa


def gera_huffman(xi, h) :
    array = []
    for i in range(0, len(xi)) :
        new_sym = '-'+str(xi[i])
        array.append(stru(new_sym, h[i], '-'))
    array_cod = gera_huffman_aux(copy.deepcopy(array))

    for i in range(0, len(array_cod)):
        array_cod[i].symbol=(array_cod[i].symbol+'-')

    for i in range(0, len(array)) :
        #array[i].bit = b''
        symb_find = (array[i].symbol+'-')

        for j in range(1, len(array_cod)):
            if symb_find in array_cod[j].symbol :
                array[i].bit = array[i].bit + array_cod[j].bit
        if array[i].symbol == '--' :
            array[i].symbol = '-'
        elif '-' in array[i].symbol :
            array[i].symbol = array[i].symbol.replace('-','')
            for j in range(0, len(xi)) :
                if array[i].symbol == str(xi[j]) :
                    array[i].symbol = xi[j]
    array=ord_array(array)
    return array
```

```
def gera_huffman_aux(x) :
    x = ord_array(x)
    alterado=0

    for i in range(len(x)-1, 0, -1) :
        if x[i].bit == '-' :
            x[i].bit = '0'
            x[i-1].bit = '1'
            new_sim = (x[i].symbol+x[i-1].symbol)
            new_noccu = (x[i].noccu+x[i-1].noccu)
            x.append(stru(new_sim,new_noccu,'-'))
            alterado = 1
            break
    if alterado == 1 :
        x = gera_huffman_aux(x)
    return x

def occurrences_struc(x) :
    array = [];
    for i in range(0, len(x)) :
        already_count = 0;
        for j in range(0, len(array)):
            if x[i] == array[j].symbol:
                already_count = 1;
        if already_count == 0:
            count = 0;
            for j in range(0, len(x)):
                if(x[i] == x[j]):
                    count = count+1
            array.append(stru(x[i],count,'-'))
    return array
```

```
def array_symbol(x):
    symbols = [];
    for i in range (0, len(x)) :
        symbols.append(x[i].symbol)
    return symbols

def array_occurences(x) :
    occurences = [];
    for i in range (0, len(x)):
        occurences.append(x[i].noccu)
    return occurences

def codifica(x,code):
    code_bin = ''
    for i in range(0, len(x)) :
        for j in range(0, len(code)) :
            if(x[i] == code[j].symbol) :
                code_bin = (code_bin+code[j].bit[1:])
    return code_bin

def entropia(array_final) :
    count_noccu = 0.0;
    for i in range(0, len(array_final)) :
        count_noccu = count_noccu+array_final[i].noccu
    entropia = 0.0
    for i in range(0, len(array_final)) :
        entropia = entropia-((array_final[i].noccu/count_noccu)*
                             *math.log(array_final[i].noccu/count_noccu,2))
    return entropia
```

Anexo C – Algoritmo Matriz Esparsa

```
def compressed_sparse(model_file, caffemodel, layers):

    net = caffe.Net(model_file, caffemodel,caffe.TEST)
    weights = []; size = []

    for i in range(0, len(layers) ) :
        name = layers[i]
        temp = net.params[name][0].data
        total_size = temp.shape[0]*temp.shape[1]
        temp = np.reshape(temp, total_size)

        aux = ''; cnt_size = 0; cnt_zeros = 0
        for idx in range(0, total_size) :
            if temp[idx] == 0 :
               cnt_zeros += 1
               if cnt_zeros == 256 or idx == total_size-1 :
                  aux = aux+str(temp[idx])+'('+str(cnt_zeros)+')'
                  cnt_size += 1
                  cnt_zeros = 0
            else :
                aux = aux+str(temp[idx])+'('+str(cnt_zeros)+')'
                cnt_size += 1
                cnt_zeros = 0

        print(''); print('Size of',name,':', cnt_size*2)
        size.append(cnt_size)
        weights.append(aux)

    return weights, size
```

Anexo D – Resultados LeNet

- *Pruning* sem quantização

					Sem Quantização											
					Amplitude		B1		B2		B4		B8		B16	
ip1	ip2	# Corte	% Corte	Size MB	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros
10	0	40000	9,88	1,46	98.02	198	99.1	90	99.08	92	99.06	94	99.06	94	99.04	96
20	0	80000	19,75	1,30	96.48	352	99.12	88	99.07	93	98.87	113	98.97	103	98.99	101
30	0	120000	29,63	1,14	94.71	524	99.03	97	98.89	111	98.87	113	98.93	107	98.89	111
40	0	160000	39,51	0,98	93.06	694	98.98	102	98.83	117	98.89	111	98.82	118	98.76	124
50	0	200000	49,38	0,82	92.4	760	99.03	97	98.76	124	98.74	126	98.66	134	98.61	139
60	0	240000	59,26	0,66	92.98	702	98.99	101	98.66	134	98.64	136	98.64	136	98.43	157
70	0	280000	69,14	0,50	94.07	593	98.93	107	98.48	152	98.28	172	98.36	164	98.14	186
80	0	320000	79,01	0,34	95.95	405	98.87	113	98.36	164	98.04	196	97.91	209	97.07	293
90	0	360000	88,89	0,18	96.13	387	98.4	160	97.43	257	95.75	425	96.09	391	94.52	548
95	0	380000	93,83	0,10	95.55	445	97.32	268	92.89	711	94.42	558	92.42	395	87.22	1278
0	10	500	0,12	1,62	98.06	194	99.12	88	99.05	95	99.05	95	99.08	92		
0	20	1000	0,25	1,62	97.45	255	99.07	93	99.04	96	98.99	101	98.98	102		
0	30	1500	0,37	1,61	97.66	234	99.12	88	99.08	92	98.93	107	98.86	114		
0	40	2000	0,49	1,61	97.96	204	99.06	94	99.03	97	98.96	104	98.27	173		
0	50	2500	0,62	1,61	97.98	202	99.05	95	98.9	110	98.7	130	96.23	377		
0	60	3000	0,74	1,61	97.9	210	99.02	98	98.54	146	97.83	217	90.37	963		
0	70	3500	0,86	1,61	97.92	208	98.81	119	97.94	206	93.05	695	87.19	1281		
0	80	4000	0,99	1,60	97.02	298	98.06	194	90.57	943	79.26	2074	76.57	2343		
0	90	4500	1,11	1,60	86.9	1310	93.16	684	79.48	2052	77.32	2268	64.25	3575		
10	10	40500	10,00	1,46	93.99	601	99.1	90	99.04	96	99.03	97	99.07	93		
20	20	81000	20,00	1,30	83.52	1648	99.09	91	98.99	101	98.91	109	98.91	109		
30	30	121500	30,00	1,13	79.87	2013	99.06	94	98.91	109	98.86	114	98.65	135		
40	40	162000	40,00	0,97	78.37	2163	99.04	96	98.85	115	98.79	121	98.2	180		
50	50	202500	50,00	0,81	78.96	2104	98.98	102	98.49	151	98.41	159	96.44	356		
60	60	243000	60,00	0,65	79.48	2052	98.91	109	98.02	198	96.78	322	88.94	1106		
70	70	283500	70,00	0,49	82.27	1773	98.62	138	96.87	313	90.15	985	82.08	1792		
80	80	324000	80,00	0,32	80.4	1960	97.5	250	89.74	1026	77.09	2291	70.71	2929		
90	90	364500	90,00	0,16	77.05	2295	89.69	1031	66.75	3325	62.06	3794	51.16	4884		
70	40	282000	69,63	0,49	81.04	1896	98.93	107	98.48	152	98.23	177	96.93	307		
70	50	282500	69,75	0,49	82.14	1786	98.93	107	98.19	181	97.76	224	94.26	574		
80	10	320500	79,14	0,34	83.21	1679	98.84	116	98.31	169	97.89	211	98.0	200		
80	20	321000	79,26	0,34	78.7	2130	98.92	108	98.35	165	97.97	203	97.59	241		
80	50	322500	79,63	0,33	84.84	1516	98.71	129	98.16	185	96.86	314	93.82	618		
85	20	341000	84,20	0,26	80.65	1935	98.66	134	97.99	201	97.32	268	97.08	292		
85	50	342500	84,57	0,25	86.04	1396	98.65	135	97.55	245	94.82	518	90.88	912		
90	10	360500	89,01	0,18	87.44	1256	97.32	268	97.39	261	95.56	444	95.91	409		
90	50	362500	89,51	0,17	88.66	1134	98.35	165	96.96	304	93.38	662	84.87	1513		
95	50	382500	94,44	0,09	88.41	1159	97.62	238	91.31	869	93.26	674	77.89	2211		

- *Pruning* com Quantização

Com Quantização													
B1							B2						
ip1	ip2	# Corte	% Corte	Size MB	% Acc	# Erros	ip1	ip2	# Corte	% Corte	Size MB	% Acc	# Erros
12	6	48300	11,926	0,36	99.08	92	17	6	68300	16,864	0,34	99.1	90
20	6	80300	19,827	0,32	99.05	95	25	6	100300	24,765	0,30	99.04	96
30	6	120300	29,704	0,28	99.03	97	33	6	132300	32,667	0,27	98.9	110
40	6	160300	39,58	0,24	99.01	99	42	6	168300	41,556	0,24	98.87	113
50	6	200300	49,457	0,20	99.01	99	51	6	204300	50,444	0,20	98.76	124
60	6	240300	59,333	0,16	99	100	60	6	240300	59,333	0,16	98.65	135
70	6	280300	69,21	0,12	98.94	106	70	6	280300	69,21	0,12	98.45	155
80	6	320300	79,086	0,08	98.79	121	80	6	320300	79,086	0,08	98.36	164
90	6	360300	88,963	0,04	98.4	160	90	6	360300	88,963	0,04	97.33	267
95	6	380300	93,901	0,02	97.4	260	95	6	380300	93,901	0,02	92.61	739
12	10	48500	11,975	0,36	99.09	91	12	13	48650	12,012	0,36	99.03	97
12	20	49000	12,099	0,36	99.06	94	12	23	49150	12,136	0,36	99	100
12	30	49500	12,222	0,36	99.1	90	12	32	49600	12,247	0,36	99.08	92
12	40	50000	12,346	0,36	99.05	95	12	41	50050	12,358	0,35	98.96	104
12	50	50500	12,469	0,35	99	100	12	51	50550	12,481	0,35	98.9	110
12	60	51000	12,593	0,35	98.97	103	12	61	51050	12,605	0,35	98.58	142
12	70	51500	12,716	0,35	98.78	122	12	70	51500	12,716	0,35	98	200
12	80	52000	12,84	0,35	97.87	213	12	80	52000	12,84	0,35	91.08	892
12	90	52500	12,963	0,35	93.35	665	12	90	52500	12,963	0,35	79.92	2008
12	10	48500	11,975	0,36	99.09	91	17	13	68650	16,951	0,34	99.03	97
20	20	81000	20	0,32	99.05	95	25	23	101150	24,975	0,30	98.98	102
30	30	121500	30	0,28	99.02	98	33	32	133600	32,988	0,27	98.94	106
40	40	162000	40	0,24	99.01	99	42	41	170050	41,988	0,23	98.84	116
50	50	202500	50	0,20	98.96	104	51	51	206550	51	0,20	98.57	143
60	60	243000	60	0,16	98.89	111	60	61	243050	60,012	0,16	97.99	201
70	70	283500	70	0,12	98.66	134	70	70	283500	70	0,12	96.78	322
80	80	324000	80	0,08	97.42	258	80	80	324000	80	0,08	89.89	1011
90	90	364500	90	0,04	89.64	1036	90	90	364500	90	0,04	66.75	3325
70	40	282000	69,63	0,12	98.92	108	70	41	282050	69,64	0,12	98.46	154
70	50	282500	69,75	0,12	98.94	106	70	51	282550	69,77	0,12	98.27	173
80	10	320500	79,14	0,08	98.82	118	80	13	320650	79,17	0,08	98.34	166
80	20	321000	79,26	0,08	98.91	109	80	23	321150	79,30	0,08	98.35	165
80	50	322500	79,63	0,08	98.73	127	80	51	322550	79,64	0,08	98.18	182
85	20	341000	84,20	0,06	98.66	134	85	23	341150	84,23	0,06	97.97	203
85	50	342500	84,57	0,06	98.64	136	85	51	342550	84,58	0,06	97.46	254
90	10	360500	89,01	0,04	98.41	159	90	13	360650	89,05	0,04	97.38	262
90	50	362500	89,51	0,04	98.38	162	90	51	362550	89,52	0,04	96.97	303
95	50	382500	94,44	0,02	97.66	234	95	51	382550	94,46	0,02	91.11	889

Com Quantização																						
B4							B8							Block 16								
ip1	ip2	# Corte	% Corte	Size MB	% Acc	# Erros	ip1	ip2	# Corte	% Corte	Size MB	% Acc	# Erros	ip1	ip2	# Corte	% Corte	Size MB	% Acc	# Erros		
19	6	76300	18,84	0,33	99.04	96	19	6	76300	18,84	0,33	99.06	94	20	6	80300	19,827	0,32	99.08	92		
27	6	108300	26,741	0,30	98.9	110	27	6	108300	26,741	0,30	98.94	106	28	6	112300	27,728	0,29	98.96	104		
35	6	140300	34,642	0,26	98.9	110	36	6	144300	35,63	0,26	98.9	110	37	6	148300	36,617	0,26	98.9	110		
44	6	176300	43,531	0,23	98.9	110	45	6	180300	44,519	0,22	98.79	121	45	6	180300	44,519	0,22	98.82	118		
53	6	212300	52,42	0,19	98.77	123	53	6	212300	52,42	0,19	98.66	134	54	6	216300	53,407	0,19	98.66	134		
62	6	248300	61,309	0,16	98.61	139	62	6	248300	61,309	0,16	98.54	146	63	6	252300	62,296	0,15	98.41	159		
71	6	284300	70,198	0,12	98.22	178	72	6	288300	71,185	0,12	98.36	164	72	6	288300	71,185	0,12	98.18	182		
80	6	320300	79,086	0,08	98	200	81	6	324300	80,074	0,08	97.91	209	81	6	324300	80,074	0,08	97.11	289		
90	6	360300	88,963	0,04	95.92	408	90	6	360300	88,963	0,04	96.06	394	91	6	364300	89,951	0,04	94.55	545		
95	6	380300	93,901	0,02	94.42	548	95	6	380300	93,901	0,02	92.25	775	95	6	380300	93,901	0,02	87.64	1236		
12	14	48700	12,025	0,36	99.03	97	12	14	48700	12,025	0,36	99.06	94									
12	23	49150	12,136	0,36	98.99	101	12	24	49200	12,148	0,36	98.95	105									
12	33	49650	12,259	0,36	98.98	102	12	33	49650	12,259	0,36	98.83	117									
12	42	50100	12,37	0,35	98.95	105	12	43	50150	12,383	0,35	98.32	168									
12	51	50550	12,481	0,35	98.74	126	12	52	50600	12,494	0,35	96.44	356									
12	61	51050	12,605	0,35	98.04	196	12	61	51050	12,605	0,35	90.56	944									
12	71	51550	12,728	0,35	93.56	644	12	71	51550	12,728	0,35	87.31	1269									
12	80	52000	12,84	0,35	79.29	2071	12	80	52000	12,84	0,35	76.57	2343									
12	90	52500	12,963	0,35	77.4	2260	12	90	52500	12,963	0,35	64.2	3580									
19	14	76700	18,938	0,33	99	100	19	14	76700	18,938	0,33	99.04	96									
27	23	109150	26,951	0,30	98.88	112	27	24	109200	26,963	0,30	98.83	117									
35	33	141650	34,975	0,26	98.86	114	36	33	145650	35,963	0,26	98.62	138									
44	42	178100	43,975	0,23	98.76	124	45	43	182150	44,975	0,22	98.2	180									
53	51	214550	52,975	0,19	98.48	152	53	52	214600	52,988	0,19	96.42	358									
62	61	251050	61,988	0,15	96.91	309	62	61	251050	61,988	0,15	89.14	1086									
71	71	287550	71	0,12	90.62	938	72	71	291550	71,988	0,11	82.09	1791									
80	80	324000	80	0,08	76.95	2305	81	80	328000	80,988	0,08	70.62	2938									
90	90	364500	90	0,04	62.06	3794	90	90	364500	90	0,04	50.94	4906									
71	42	286100	70,64	0,12	98.21	179	72	43	290150	71,64	0,11	96.77	323									
71	51	286550	70,75	0,12	97.67	233	72	52	290600	71,75	0,11	94.06	594									
80	14	320700	79,19	0,08	97.9	210	81	14	324700	80,17	0,08	97.97	203									
80	23	321150	79,30	0,08	97.92	208	81	24	325200	80,30	0,08	97.47	253									
80	51	322550	79,64	0,08	96.9	310	81	52	326600	80,64	0,08	93.86	614									
85	23	341150	84,23	0,06	97.33	267	85	24	341200	84,25	0,06	97.17	283									
85	51	342550	84,58	0,06	94.44	556	85	52	342600	84,59	0,06	91.13	887									
90	14	360700	89,06	0,04	95.58	442	90	14	360700	89,06	0,04	95.95	405									
90	51	362550	89,52	0,04	93.26	674	90	52	362600	89,53	0,04	84.79	1521									
95	51	382550	94,46	0,02	93.22	678	95	52	382600	94,47	0,02	78.2	2180									

Anexo E – Resultados CIFAR-10 Quick

- *Pruning* sem quantização

					Sem Quantização											
					Amplitude		B1		B2		B4		B8		B16	
ip1	ip2	# Corte	% Corte	Size KB	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros
10	0	6553,6	9,9033	238,49	20.73	7927	53.73	4627	53.27	4673	53.59	4641	50.83	4917	51.99	4801
20	0	13107	19,807	212,28	14.03	8597	54.11	4589	53.21	4679	52.83	4717	52.4	4760	52.11	4789
30	0	19661	29,71	186,06	11.97	8803	54.13	4587	51.56	4844	47.62	5238	48.57	5143	50.01	4999
40	0	26214	39,613	159,85	11.08	8892	52.72	4728	49.07	5093	40.85	5915	46.21	5379	46.6	5340
50	0	32768	49,516	133,63	10.75	8925	49.83	5017	45.24	5476	39.92	6008	41.33	5867	48.14	5186
60	0	39322	59,42	107,42	11.27	8873	46.4	5360	41.32	5868	39.45	6055	35.59	6441	42.15	5785
70	0	45875	69,323	81,20	12.07	8793	41.51	5849	41.15	5885	38.5	6150	38	6200	38.67	6133
80	0	52429	79,226	54,99	14.74	8526	38.21	6179	41.21	5879	31.77	6823	38.3	6170	27.95	7205
85	0	55706	84,178	41,88	16.27	8373	39.15	6085	35.03	6497	29.56	7044	33.64	6636	22.44	7756
90	0	58982	89,13	28,77	16.34	8366	37.78	6222	31.78	6822	24.82	7518	30.84	6916	14.86	8514
95	0	62259	94,081	15,67	13.45	8655	34.9	6510	25.75	7425	21.05	7895	18.69	8131	18.2	8180
0	10	64	0,0967	264,45	40.87	5913	53.82	4618	53.38	4662	53.59	4641	54	4600	53.75	4625
0	20	128	0,1934	264,19	36.7	6330	53.2	4680	53.08	4692	54.41	4559	51.25	4875	49.48	5052
0	30	192	0,2901	263,94	37.21	6279	54.05	4595	53.08	4692	51.85	4815	51.05	4895	47.57	5243
0	40	256	0,3868	263,68	40.3	5970	53.33	4667	52.18	4782	53.82	4618	45.8	5420	43.98	5602
0	50	320	0,4836	263,42	40.51	5949	52.72	4728	48.72	5128	47.35	5265	43.14	5686	40.15	5985
0	60	384	0,5803	263,17	39.15	6085	51.7	4830	46.38	5362	42.47	5753	37.12	6288	37.83	6217
0	70	448	0,677	262,91	36.81	6319	47.83	5217	41.77	5823	41.38	5862	32.41	6759	34.46	6554
0	80	512	0,7737	262,66	34.32	6568	45.36	5464	42.2	5780	39.9	6010	31.8	6820	30.72	6928
0	85	544	0,8221	262,53	31.45	6855	44.63	5537	36.74	6326	37.96	6204	30.57	6943	29.44	7056
0	90	576	0,8704	262,40	33.36	6664	41.59	5841	32.2	6780	32.95	6705	30.04	6996	26.8	7320
0	95	608	0,9188	262,27	26.97	7303	31.59	6841	29.05	7095	24.52	7548	23.21	7679	17.73	8227
10	10	6617,6	10	238,23	10.02	8998	53.7	4630	52.3	4770	53.3	4670	51.14	4886	50.37	4963
20	20	13235	20	211,76	10	9000	53.05	4695	52.1	4790	52.32	4768	48.81	5119	47.97	5203
30	30	19853	30	185,29	10	9000	54.05	4595	50.88	4912	46.42	5358	44.03	5597	44.12	5588
40	40	26470	40	158,82	10	9000	52.59	4741	49.13	5087	42.77	5723	39.29	6071	38.33	6167
50	50	33088	50	132,35	10	9000	47.91	5209	42.56	5744	40.33	5967	36.41	6359	37.27	6273
60	60	39706	60	105,88	10	9000	47.17	5283	38.32	6168	34.87	6513	29.97	7003	21.34	7866
70	70	46323	70	79,41	10	9000	40.99	5901	34.63	6537	33.19	6681	25.9	7410	23.15	7685
80	80	52941	80	52,94	10	9000	33.23	6677	31.56	6844	26.09	7391	21.84	7816	18.23	8177
85	85	56250	85	39,71	10.0	9000	35.22	6478	26.05	7395	25.04	7496	23.1	7690	14.20	8579
90	90	59558	90	26,47	10.03	8997	26.59	7341	23.8	7620	21.64	7836	15.8	8420	16.26	8374
95	95	62867	95	13,24	11.72	8828	13.59	8641	16.7	8330	14.47	8553	12.05	8795	10.7	8926

- *Pruning* com quantização

Com Quantização													
B1							B2						
ip1	ip2	# Corte	% Corte	Size KB	% Acc	# Erros	ip1	ip2	# Corte	% Corte	Size KB	% Acc	# Erros
10	4	6579,2	9,942	59,60	53.83	4617	13	4	8545,3	12,913	57,63	53.21	4679
20	4	13133	19,845	53,04	54.15	4585	22	4	14444	21,826	51,73	53.28	4672
30	4	19686	29,749	46,49	54.21	4579	31	4	20342	30,739	45,83	51.5	4850
40	4	26240	39,652	39,94	52.86	4714	41	4	26895	40,642	39,28	49.07	5093
50	4	32794	49,555	33,38	49.97	5003	51	4	33449	50,545	32,73	45.32	5468
60	4	39347	59,458	26,83	46.61	5339	60	4	39347	59,458	26,83	41.49	5851
70	4	45901	69,362	20,28	41.65	5835	70	4	45901	69,362	20,28	41.61	5839
80	4	52454	79,265	13,72	38.57	6143	80	4	52454	79,265	13,72	41.12	5888
85	4	55731	84,217	10,44	39.15	6085	85	4	55731	84,217	10,44	35.03	6497
90	4	59008	89,168	7,17	37.76	6224	90	4	59008	89,168	7,17	31.74	6826
95	4	62285	94,12	3,89	34.74	6526	95	4	62285	94,12	3,89	25.68	7432
5	10	3340,8	5,0484	62,84	53.9	4610	5	13	3360	5,0774	62,82	53.33	4667
5	20	3404,8	5,1451	62,77	53.42	4658	5	23	3424	5,1741	62,75	53.29	4671
5	30	3468,8	5,2418	62,71	54.32	4568	5	32	3481,6	5,2611	62,69	53.07	4693
5	40	3532,8	5,3385	62,64	53.29	4671	5	42	3545,6	5,3578	62,63	52.27	4773
5	50	3596,8	5,4352	62,58	52.66	4734	5	51	3603,2	5,4449	62,57	48.77	5123
5	60	3660,8	5,5319	62,52	51.58	4842	5	61	3667,2	5,5416	62,51	46.56	5344
5	70	3724,8	5,6286	62,45	47.74	5226	5	70	3724,8	5,6286	62,45	41.79	5821
5	80	3788,8	5,7253	62,39	45.2	5480	5	80	3788,8	5,7253	62,39	42.37	5763
5	90	3852,8	5,8221	62,32	41.82	5818	5	90	3852,8	5,8221	62,32	32.29	6771
10	10	6617,6	10	59,56	53.83	4617	13	10	8583,7	12,971	57,59	52.42	4758
20	20	13235	20	52,94	53.39	4661	22	20	14546	21,981	51,63	52.22	4778
30	30	19853	30	46,32	54.28	4572	31	30	20508	30,99	45,67	50.95	4905
40	40	26470	40	39,71	52.67	4733	41	40	27126	40,99	39,05	49.05	5095
50	50	33088	50	33,09	48.04	5196	51	50	33743	50,99	32,43	42.55	5745
60	60	39706	60	26,47	47.08	5292	60	60	39706	60	26,47	38.38	6162
70	70	46323	70	19,85	41.28	5872	70	70	46323	70	19,85	34.61	6539
80	80	52941	80	13,24	33.13	6687	80	80	52941	80	13,24	31.58	6842
90	90	59558	90	6,62	26.56	7344	90	90	59558	90	6,62	23.79	7621

Com Quantização																			
B4							B8							B16					
ip1	ip2	# Corte	% Corte	Size KB	% Acc	# Erros	ip1	ip2	# Corte	% Corte	Size KB	% Acc	# Erros	ip1	ip2	# Corte	% Corte	Size KB	% Acc	# Erros
13	4	8545,3	12,913	57,63	53.75	4625	14	4	9200,6	13,903	56,98	50.91	4909	14	4	9200,6	13,903	56,98	52.15	4785
23	4	15099	22,816	51,08	53.04	4696	23	4	15099	22,816	51,08	52.57	4743	23	4	15099	22,816	51,08	52.26	4774
32	4	20997	31,729	45,18	48.21	5179	32	4	20997	31,729	45,18	48.67	5133	33	4	21652	32,72	44,52	50.58	4942
42	4	27551	41,632	38,63	41.29	5871	42	4	27551	41,632	38,63	46.45	5355	42	4	27551	41,632	38,63	46.85	5315
51	4	33449	50,545	32,73	40.13	5987	51	4	33449	50,545	32,73	41.46	5854	52	4	34104	51,536	32,07	48.33	5167
61	4	40003	60,449	26,17	39.78	6022	61	4	40003	60,449	26,17	35.74	6426	61	4	40003	60,449	26,17	42.75	5725
70	4	45901	69,362	20,28	38.59	6141	71	4	46556	70,352	19,62	37.89	6211	71	4	46556	70,352	19,62	38.98	6102
80	4	52454	79,265	13,72	32	6800	80	4	52454	79,265	13,72	38.15	6185	80	4	52454	79,265	13,72	28.1	7190
85	4	55731	84,217	10,44	29.38	7062	85	4	55731	84,217	10,44	33.79	6621	85	4	55731	84,217	10,44	22.32	7768
90	4	59008	89,168	7,17	24.73	7527	90	4	59008	89,168	7,17	31.02	6898	90	4	59008	89,168	7,17	14.74	8526
95	4	62285	94,12	3,89	20.58	7942	95	4	62285	94,12	3,89	18.81	8119	95	4	62285	94,12	3,89	18.25	8175
5	13	3360	5,0774	62,82	53.86	4614	5	13	3360	5,0774	62,82	54.15	4585	5	13	3360	5,0774	62,82	53.83	4617
5	23	3424	5,1741	62,75	54.39	4561	5	23	3424	5,1741	62,75	51.42	4858	5	23	3424	5,1741	62,75	49.7	5030
5	32	3481,6	5,2611	62,69	51.95	4805	5	32	3481,6	5,2611	62,69	51.1	4890	5	32	3481,6	5,2611	62,69	47.54	5246
5	41	3539,2	5,3482	62,64	53.75	4625	5	42	3545,6	5,3578	62,63	45.85	5415	5	42	3545,6	5,3578	62,63	44.06	5594
5	51	3603,2	5,4449	62,57	47.61	5239	5	52	3609,6	5,4545	62,57	43.53	5647	5	52	3609,6	5,4545	62,57	40.34	5966
5	61	3667,2	5,5416	62,51	42.71	5729	5	61	3667,2	5,5416	62,51	37.53	6247	5	61	3667,2	5,5416	62,51	38.05	6195
5	71	3731,2	5,6383	62,44	41.32	5868	5	71	3731,2	5,6383	62,44	32.71	6729	5	71	3731,2	5,6383	62,44	34.69	6531
5	80	3788,8	5,7253	62,39	40.05	5995	5	79	3782,4	5,7157	62,39	32.02	6798	5	78	3776	5,706	62,40	30.86	6914
5	90	3852,8	5,8221	62,32	32.92	6708	5	90	3852,8	5,8221	62,32	30.42	6958	5	90	3852,8	5,8221	62,32	26.58	7342
13	13	8602,9	13	57,57	53.64	4636	14	13	9258,2	13,99	56,92	51.23	4877	14	13	9258,2	13,99	56,92	50.75	4925
23	23	15220	23	50,96	52.26	4774	23	23	15220	23	50,96	49.03	5097	23	23	15220	23	50,96	48.34	5166
32	32	21176	32	45,00	46.53	5347	32	32	21176	32	45,00	44.01	5599	33	32	21832	32,99	44,34	44.16	5584
42	41	27788	41,99	38,39	42.44	5756	42	42	27794	42	38,38	39.53	6047	42	42	27794	42	38,38	38.07	6193
51	51	33750	51	32,43	40.67	5933	51	52	33756	51,01	32,42	36.77	6323	52	52	34412	52	31,76	37.33	6267
61	61	40367	61	25,81	35.15	6485	61	61	40367	61	25,81	29.75	7025	61	61	40367	61	25,81	21.18	7882
70	71	46330	70,01	19,85	33.21	6679	71	71	46985	71	19,19	26.13	7387	71	71	46985	71	19,19	23.35	7665
80	80	52941	80	13,24	26.16	7384	80	79	52934	79,99	13,24	21.84	7816	80	78	52928	79,981	13,25	18.12	8188
90	90	59558	90	6,62	21.3	7870	90	90	59558	90	6,62	15.87	8413	90	90	59558	90	6,62	16.21	8379

Anexo F – Resultados CIFAR-10

- *Pruning* sem quantização

			Sem Quantização											
			Amplitude		B1		B2		B4		B8		B16	
ip1	# Corte	Size KB	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros
10	1024	36,86	70.38	2962	71.9	2810	71.9	2810	71.8	2820	71.91	2809	71.41	2859
20	2048	32,77	71.07	2893	71.85	2815	71.8	2820	71.72	2828	71.84	2816	72.05	2795
30	3072	28,67	70.81	2919	72.05	2795	71.79	2821	72	2800	71.59	2841	71.57	2843
40	4096	24,58	70.98	2902	71.92	2808	71.73	2827	72.14	2786	71.66	2834	71.12	2888
50	5120	20,48	71	2900	71.95	2805	71.58	2842	72.01	2799	71.86	2814	70.66	2934
60	6144	16,38	71.09	2891	71.71	2829	71.68	2832	71.33	2867	70.9	2910	69.06	3094
70	7168	12,29	71.04	2896	71.38	2862	71.68	2832	70.8	2920	70.23	2977	69.27	3073
80	8192	8,19	70.83	2917	71.82	2818	71.68	2832	69.6	3040	69.99	3001	68.11	3189
85	8704	6,14	70.83	2917	70.49	2951	70.13	2987	69.66	3034	70.82	2918	65.56	3444
90	9216	4,10	70.38	2962	69.65	3035	69.73	3027	68.05	3195	68.69	3131	64.56	3544
95	9728	2,05	68.63	3137	68.34	3166	67.59	3241	65.76	3424	65.7	3430	53.96	4604

- *Pruning* com quantização

Com Quantização									
B1					B2				
ip1	# Corte	Size KB	% Acc	# Erros	ip1	# Corte	Size KB	% Acc	# Erros
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	89	9113,6	1,13	70.41	2959
88	9011,2	1,23	69.78	3022	91,1	9328,6	0,91	69.76	3024
90	9216	1,02	69.12	3088	92,2	9441,3	0,80	68.82	3118
95	9728	0,51	67.25	3275	95,5	9779,2	0,46	66.97	3303

Com Quantização														
B4					B8					B16				
ip1	# Corte	Size KB	% Acc	# Erros	ip1	# Corte	Size KB	% Acc	# Erros	ip1	# Corte	Size KB	% Acc	# Erros
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022
88	9011,2	1,23	69.78	3022	88	9011,2	1,23	69.78	3022	88	9011,2	1,23	68.92	3108
88	9011,2	1,23	69.78	3022	88,4	9052,2	1,19	69.06	3094	88,7	9082,9	1,16	68.56	3144
88	9011,2	1,23	69.78	3022	89	9113,6	1,13	68.58	3142	89	9113,6	1,13	67.37	3263
88,9	9103,4	1,14	68.79	3121	90	9216	1,02	68.21	3179	90	9216	1,02	67.44	3256
90,4	9257	0,98	67.33	3267	91,6	9379,8	0,86	68.26	3174	91,3	9349,1	0,89	66.63	3337
91,7	9390,1	0,85	67.81	3219	93	9523,2	0,72	69.08	3092	94,3	9656,3	0,58	64.13	3587
93,7	9594,9	0,65	66.52	3348	94,6	9687	0,55	67.75	3225	95,7	9799,7	0,44	64.36	3564
96,4	9871,4	0,37	64.72	3528	96,8	9912,3	0,33	65.17	3483	97,5	9984	0,26	53.91	4609

Anexo G – Resultados AlexNet

- *Pruning* sem quantização

						Sem Quantização											
						Amplitude		B1		B2		B4		B8		B16	
fc6	fc7	fc8	# Corte	% Corte	Size MB	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros	% Acc	# Erros
10	0	0	104858	0,5869	71,048	45.78	5422	47.78	5222	47.77	5223	47.78	5222	47.79	5221	43.91	5609
20	0	0	209715	1,1738	70,628	45.92	5408	47.78	5222	47.83	5217	47.8	5220	47.83	5217	43.98	5602
30	0	0	314573	1,7607	70,209	46.15	5385	47.78	5222	47.85	5215	47.77	5223	47.8	5220	43.89	5611
40	0	0	419430	2,3475	69,789	46.48	5352	47.9	5210	47.84	5216	47.73	5227	47.68	5232	43.82	5618
50	0	0	524288	2,9344	69,37	46.68	5332	47.86	5214	47.79	5221	47.64	5236	47.48	5252	43.67	5633
60	0	0	629146	3,5213	68,95	46.71	5329	48.02	5198	47.81	5219	47.58	5242	47.19	5281	43.4	5660
70	0	0	734003	4,1082	68,531	46.67	5333	48.19	5181	47.73	5227	47.23	5277	46.88	5312	42.93	5707
80	0	0	838861	4,6951	68,112	46.49	5351	48.12	5188	47.54	5246	46.8	5320	45.7	5430	42.04	5796
90	0	0	943718	5,282	67,692	46.65	5335	48.2	5180	46.84	5316	44.66	5534	43.58	5642	39.57	6043
0	10	0	2E+06	9,3902	64,756	47.78	5222	47.78	5222	47.78	5222	47.77	5223	47.78	5222	43.93	5607
0	20	0	3E+06	18,78	58,045	47.28	5272	47.78	5222	47.77	5223	47.77	5223	47.77	5223	43.89	5611
0	30	0	5E+06	28,171	51,334	47	5300	47.77	5223	47.72	5228	47.73	5227	47.77	5223	43.85	5615
0	40	0	7E+06	37,561	44,623	46.76	5324	47.72	5228	47.66	5234	47.68	5232	47.7	5230	43.73	5627
0	50	0	8E+06	46,951	37,913	46.71	5329	47.64	5236	47.58	5242	47.54	5246	47.51	5249	43.59	5641
0	60	0	1E+07	56,341	31,202	46.72	5328	47.5	5250	47.38	5262	47.36	5264	47.22	5278	43.49	5651
0	70	0	1E+07	65,731	24,491	46.76	5324	47.27	5273	47.13	5287	46.94	5306	46.88	5312	43.33	5667
0	80	0	1E+07	75,122	17,78	46.6	5340	46.92	5308	46.62	5338	46.49	5351	46.1	5390	42.6	5740
0	90	0	2E+07	84,512	11,069	46.06	5394	46.19	5381	45.78	5422			43.86	5614	40.55	5945
0	0	10	4096	0,0229	71,451	46.33	5367	47.78	5222	47.87	5213	47.82	5218	47.7	5230	47.7	5230
0	0	20	8192	0,0459	71,434	46.03	5397	47.78	5222	47.74	5226	47.69	5231	47.42	5258	47.42	5258
0	0	30	12288	0,0688	71,418	45.77	5423	47.83	5217	47.65	5235	47.68	5232	47.19	5281	47.19	5281
0	0	40	16384	0,0917	71,401	45.54	5446	47.74	5226	47.62	5238	47.52	5248	47.19	5281	47.19	5281
0	0	50	20480	0,1146	71,385	45.5	5450	47.55	5245	47.62	5238	47.23	5277	46.99	5301	46.99	5301
0	0	60	24576	0,1376	71,369	45.29	5471	47.31	5269	47.43	5257	46.67	5333	46.75	5325	46.75	5325
0	0	70	28672	0,1605	71,352	44.73	5527	47.29	5271	47.09	5291	46.71	5329	45.86	5414	45.86	5414
0	0	80	32768	0,1834	71,336	43.61	5639	47.06	5294	47.01	5299	46.52	5348	45.21	5479	45.21	5479
0	0	90	36864	0,2063	71,32	41.29	5871	46.9	5310	46.13	5387	45.77	5423	43.92	5608	43.92	5608
10	10	10	2E+06	10	64,32	36.92	6308	47.77	5223			47.82	5218				
20	20	20	4E+06	20	57,174	30.83	6917	47.82	5218			47.58	5242				
30	30	30	5E+06	30	50,027	24.55	7545	47.87	5213			47.48	5252				
40	40	40	7E+06	40	42,88	20.52	7948	47.73	5227			47.27	5273				
50	50	50	9E+06	50	35,734	19.71	8029	47.66	5234			46.68	5332				
60	60	60	1E+07	60	28,587	19.24	8076	47.27	5273			45.5	5450				
70	70	70	1E+07	70	21,44	18.27	8173	47.31	5269			44.94	5506				
80	80	80	1E+07	80	14,293	17.16	8284	46.55	5345			44.1	5590				
90	90	90	2E+07	90	7,1467	16.02	8398	44.63	5537			38.81	6119				

- *Pruning* com quantização

Com Quantização															
B1								B2							
fc6	fc7	fc8	# Corte	% Corte	Size MB	% Acc	# Erros	fc6	fc7	fc8	# Corte	% Corte	Size MB	% Acc	# Erros
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.69	8731
81	100	21	2E+07	98,704	0,9264	12.68	8732	82	100	21	2E+07	98,762	0,8844	12.66	8734
81	100	21	2E+07	98,704	0,9264	12.68	8732	84	100	21	2E+07	98,88	0,8005	12.5	8750
81	100	21	2E+07	98,704	0,9264	12.68	8732	88	100	21	2E+07	99,115	0,6328	12.14	8786
90	100	21	2E+07	99,232	0,5489	11.94	8806	93	100	21	2E+07	99,408	0,423	11.37	8863
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	24	2E+07	98,711	0,9214	12.53	8747
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	30	2E+07	98,724	0,9116	11.79	8821
81	100	30	2E+07	98,724	0,9116	12.34	8766	81	100	37	2E+07	98,74	0,9001	11.88	8812
81	100	40	2E+07	98,747	0,8952	12.07	8793	81	100	45	2E+07	98,759	0,887	12.57	8743
81	100	50	2E+07	98,77	0,8788	11.67	8833	81	100	54	2E+07	98,779	0,8723	12.83	8717
81	100	60	2E+07	98,793	0,8625	11.6	8840	81	100	63	2E+07	98,8	0,8575	14.26	8574
81	100	70	2E+07	98,816	0,8461	12.5	8750	81	100	72	2E+07	98,821	0,8428	13.48	8652
81	100	80	2E+07	98,839	0,8297	14.46	8554	81	100	81	2E+07	98,841	0,828	16.05	8395
81	100	90	2E+07	98,862	0,8133	26.3	7370	81	100	90	2E+07	98,862	0,8133	19.06	8094

Com Quantização																							
B4								B8								B16							
fc6	fc7	fc8	# Corte	% Corte	Size MB	% Acc	# Erros	fc6	fc7	fc8	# Corte	% Corte	Size MB	% Acc	# Erros	fc6	fc7	fc8	# Corte	% Corte	Size MB	% Acc	# Erros
81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732	81	100	21	2E+07	98,704	0,9264	12.68	8732
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	82	100,00	21	2E+07	98,762	0,8844	12.69	8731	82	100,00	21	2E+07	98,762	0,8844	12.68	8732
82	100	21,00	2E+07	98,762	0,8844	12.67	8733	83	100,00	21	2E+07	98,821	0,8425	12.64	8736	84	100,00	21	2E+07	98,88	0,8005	12.64	8736
83	100	21,00	2E+07	98,821	0,8425	12.62	8738	84	100,00	21	2E+07	98,88	0,8005	12.5	8750	85	100,00	21	2E+07	98,939	0,7586	12.52	8748
85	100	21,00	2E+07	98,939	0,7586	12.46	8754	86	100,00	21	2E+07	98,997	0,7166	12.34	8766	87	100,00	21	2E+07	99,056	0,6747	12.31	8769
87	100	21,00	2E+07	99,056	0,6747	12.18	8782	89	100,00	21	2E+07	99,173	0,5908	12.1	8790	90	100,00	21	2E+07	99,232	0,5489	12.05	8795
90	100	21,00	2E+07	99,232	0,5489	11.73	8827	92	100,00	21	2E+07	99,349	0,465	11.65	8835	92	100,00	21	2E+07	99,349	0,465	11.66	8834
94	100	21,00	2E+07	99,467	0,3811	10.91	8909	95	100,00	21	2E+07	99,525	0,3391	10.95	8905	96	100,00	21	2E+07	99,584	0,2972	11.06	8894
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.54	8746
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.63	8737	81	100,00	21	2E+07	98,704	0,9264	12.26	8774
81	100	21,00	2E+07	98,704	0,9264	12.68	8732	81	100,00	21	2E+07	98,704	0,9264	12.2	8780	81	100,00	21	2E+07	98,704	0,9264	11.43	8857
81	100	26,00	2E+07	98,715	0,9182	12.57	8743	81	100,00	27	2E+07	98,718	0,9165	12.23	8777	81	100,00	28	2E+07	98,72	0,9149	12.73	8727
81	100	33,00	2E+07	98,731	0,9067	12.62	8738	81	100,00	34	2E+07	98,734	0,9051	12.2	8780	81	100,00	35	2E+07	98,736	0,9034	12.62	8738
81	100	40,00	2E+07	98,747	0,8952	13.37	8663	81	100,00	42	2E+07	98,752	0,8919	12.32	8768	81	100,00	43	2E+07	98,754	0,8903	12.21	8779
81	100	48,00	2E+07	98,766	0,8821	13.25	8675	81	100,00	49	2E+07	98,768	0,8805	13.68	8632	81	100,00	50	2E+07	98,77	0,8788	13.1	8690
81	100	56,00	2E+07	98,784	0,869	13.52	8648	81	100,00	57	2E+07	98,786	0,8674	14.42	8558	81	100,00	58	2E+07	98,789	0,8657	12.04	8796
81	100	64,00	2E+07	98,802	0,8559	12.49	8751	81	100,00	65	2E+07	98,805	0,8543	16.28	8372	81	100,00	66	2E+07	98,807	0,8526	11.82	8818
81	100	73,00	2E+07	98,823	0,8412	12.85	8715	81	100,00	74	2E+07	98,825	0,8395	14.51	8549	81	100,00	74	2E+07	98,825	0,8395	11.43	8857
81	100	81,00	2E+07	98,841	0,828	16.84	8316	81	100,00	82	2E+07	98,844	0,8264	13.99	8601	81	100,00	83	2E+07	98,846	0,8248	16.41	8359
81	100	91,00	2E+07	98,864	0,8117	26.03	7397	81	100,00	91	2E+07	98,864	0,8117	17.84	8216	81	100,00	91	2E+07	98,864	0,8117	19.34	8066