Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Engenharia Elétrica

João Guilherme Reiser de Melo

Análise da Aplicação de Redes Neurais Convolucionais em Sistemas de Visão Computacional Embarcados

Florianópolis

2018

João Guilherme Reiser de Melo

Análise da Aplicação de Redes Neurais Convolucionais em Sistemas de Visão Computacional Embarcados

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Héctor Pettenghi Roldán

Florianópolis

2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Melo, João Guilherme Reiser de
Análise da Aplicação de Redes Neurais Convolucionais em Sistemas de Visão Computacional Embarcados / João Guilherme Reiser de Melo ; orientador, Héctor Pettenghi Róldan, 2018.
133 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós Graduação em Engenharia Elétrica, Florianópolis, 2018.

Inclui referências.

1. Engenharia Elétrica. 2. Deep Learning. 3. Sistemas Embarcados. 4. Visão Computacional. I. Róldan, Héctor Pettenghi. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

João Guilherme Reiser de Melo

Análise da Aplicação de Redes Convolucionais em Sistemas de Visão Computacional Embarcados

Esta Dissertação foi julgada adequada para obtenção do Título de “Mestre em Engenharia Elétrica” e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica.

Florianópolis, 6 de dezembro de 2018.

_____

Prof. Bartolomeu Ferreira Uchôa Filho, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

_____

Prof. Héctor Pettenghi Roldán, Dr.
Orientador

Banca Examinadora:

_____

Prof. Mauro Roisenberg, Dr.
Universidade Federal de Santa Catarina

_____

Prof. Richard Demo Souza, Dr.
Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus pais, Ana Cláudia e José Antônio e, a minha namorada, Adriana.

Agradecimentos

Aos meus pais, pelo apoio e por sempre terem me incentivado a leitura e aos estudos.

A minha namorada, pelo apoio e compreensão durante a realização dessa etapa.

Ao professor Héctor, pelas orientações e por aceitar participar dessa empreitada. Ao professor Djones, pelo convite para fazer parte do projeto Intel e consequentemente por me abrir as portas da inteligência artificial.

Aos colegas de mestrado, que tornaram esse processo menos doloroso. Aos meus professores, pelos ensinamentos e por compartilharem um pouco dos seus conhecimentos e, aos funcionários do programa de pós-graduação, sempre muito atenciosos e prestativos.

Gostaria de agradecer ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela concessão da bolsa de estudos e a Intel Corporation, pela bolsa e pela possibilidade fazer parte de um projeto em parceria.

"Todas as maiores invenções tecnológicas criadas pelo homem - o avião, o automóvel, o computador - dizem pouco sobre sua inteligência, mas falam bastante sobre sua preguiça."

(Mark Kennedy)

Resumo

A inteligência artificial vêm sendo amplamente difundida e utilizada para auxiliar ou permitir a solução de certos problemas. Uma área que recebe atenção especial é a de visão computacional, com avanços significativos a cada ano. Entretanto, aplicações embarcadas, visando *Edge Computing*, IoT, ou similares, que utilizem conceitos de *Deep Learning*, ainda são limitadas em virtude dos modelos exigirem elevado poder de processamento, alcançando resultados pouco expressivos em aplicações de tempo real. O trabalho toma como objetivo estudar a aplicação de redes neurais convolucionais em aplicações embarcadas, tendo como plataforma o dispositivo Intel Movidius Neural Compute Stick. Tomando como base o *dataset Plant Village*, composto por folhas de plantas saudáveis ou doentes, é realizada a análise das bibliografias pertinentes e as principais redes de *Deep Learning* são identificadas. A partir disso, o trabalho se inicia analisando os parâmetros pertinentes resultantes dessas redes, propõem a substituição de modelos de classificação por redes de detecção de características, apresenta um estudo de influência das camadas que compõem o modelo nos resultados de tempo de treinamento, tempo de inferência, consumo energético e acurácia, além de apresentar um método de inferência em dois estágios. Os resultados mostram que a utilização de redes para detecção possibilitam uma significativa redução no tempo de inferência e consumo energético, frente as redes normalmente empregadas. Já a inferência em dois estágios pode gerar ganhos que superam os 90%, dependendo da taxa de detecção positiva.

Palavras-chave: *Deep Learning.* Sistemas Embarcados. Visão Computacional.

Abstract

Artificial intelligence has been widely disseminated and used to help or enable the solution of certain problems. One area that receives special attention is the computer vision, with significant advances every year. However, embedded applications, aimed at Edge Computing, IoT, or similar, using concepts of Deep Learning, are still limited because the models require high processing power, reaching poor results in real-time applications. The work aims to study the application of convolutional neural networks in embedded applications, having as platform the device Intel Movidius Neural Compute Stick. Based on the dataset Plant Village, composed of healthy or diseased plant leaves, the relevant bibliographies are analyzed and the main networks of Deep Learning are identified. From this, the work begins by analyzing the pertinent parameters resulting from these networks, proposes the substitution of classification models by networks of detection of characteristics, presents a study of the influence of the layers that make up the model in the results of training time, time of inference, energy consumption and accuracy, as well as presenting a two-stage inference method. The results show that the use of networks for detection allows a significant reduction in the time of inference and energy consumption, compared to the networks normally employed. Two-stage inference can generate gains in excess of 90 %, depending on the positive detection rate.

Keywords: Deep Learning. Embedded Systems. Computer Vision.

Lista de Figuras

Lista de Tabelas

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ARM	Advanced RISC Machine
ASIC	Application Specific Integrated Circuit
CNN	Convolutional Neural Network
CPU	Central Process Unit
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
GAN	Generative Adversary Networks
GOPS	Giga Operations Per Second
GPU	Graphics Processing Unit
GUI	Graphics User Interface
IA	Inteligência Artificial
IoT	Internet of Things
NCS	Neural Compute Stick
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SDK	Software Development Kit
SIFT	Scale-Invariant Feature Transform
SoC	System-on-a-Chip
SVM	Support Vector Machine
VLIW	Very Long Instruction Word
VPU	Visual Processing Unit

Sumário

1 INTRODUÇÃO

Nos últimos 15 anos, os avanços alcançados nas áreas de elétrica, eletrônica, computação, possibilitaram a criação de uma gigantesca quantidade de dados a cada instante. Até 2003, somente 5 exabytes de dados haviam sido gerados em todo o mundo. Em 2013, essa mesma quantidade era atingida a cada 2 dias, como mostrado por Sagiroglu e Sinanc (2013). Atualmente, com o desenvolvimento de produtos e de pesquisas focados na Internet das Coisas (IoT) e em Indústria 4.0, o volume de dados que se espera para o futuro é ainda maior.

Em virtude dessa explosão de dados, fez-se necessária a criação de áreas do conhecimento que pudessem utilizá-los, como *Big Data* e *Data Science*, pois, como explicado por Gupta (2015), gerar material digital e não analisá-lo da forma devida, não traz beneficio algum. Em virtude desse trabalho mais refinado com os dados coletados, algumas áreas do conhecimento que estavam sem avanços significativos puderam voltar a apresentar resultados expressivos, como ocorreu com diversas técnicas de inteligência artificial.

A inteligência artificial (IA) faz cada vez mais parte do dia a dia das pessoas, de forma muitas vezes invisivel. Sua principal aplicação se dá no auxilio a tomadas de decisão, desde a compra ou não de uma ação, o fechamento ou não de um negócio, até sugestões de qual filme ou video assistir, música ouvir, produto comprar ou reportagem acessar. Pela enorme diversidade de problemas, existe uma infinidade de soluções na área de inteligência artificial. Quando se fala de aplicações de visão computacional, a área de *Deep Learning* atualmente é o estado da arte, com as redes neurais convolucionais (CNN).

Inicialmente, em virtude da sua grande exigência computacional, tais aplicações eram exclusivas de computadores pessoais ou grandes servidores. Aplicativos de celular ainda hoje se utilizam de soluções que exigem comunicação com servidores (*Cloud Computing*) para a execução de algum algoritmo de classificação; porém, cada vez mais, as aplicações off-line estão sendo difundidas. Tais aplicações recebem o nome de *Edge Computing*, permitindo com que as tomadas de decisão ocorram próxima da fonte geradora de dados, o que reduz a latência do sistema; aumenta a confiabilidade, a segurança e a privacidade dos dados; além de, em alguns casos, reduzir o consumo energético. Isso só foi possivel devido aos avanços tecnológicos dos processadores compactos e com o surgimento de

soluções de processamento dedicadas. Outra vantagem bastante interessante é a possibilidade de operar em regiões remotas, sem acesso à internet.

As pesquisas na área de *Deep Learning* praticamente se limitam ao desenvolvimento de técnicas que aumentem cada vez mais a acurácia do sistema e diminuam o tempo de treinamento, não levando tanto em consideração outros fatores, como tamanho da rede, tempo de inferência ou consumo energético. Em uma aplicação embarcada, além de uma boa precisão, faz-se necessária uma resposta bastante rápida, a fim de que um sistema subsequente tome uma decisão dentro de um tempo aceitável. Alguns estudos já demonstraram a eficiência de redes neurais convolucionais compactas, como mostrado no trabalho de HasanPour et al. (2016), redes essas que alcançaram resultados próximas ou melhores que outras bem mais complexas.

Boa parte das aplicações de visão computacional embarcadas baseadas em *Deep Learning* se utilizam de modelos tradicionais pré-treinados, agilizando o processo de desenvolvimento da solução, porém, não considerando os demais fatores que afetam o desempenho da aplicação. Pensando nisso, esse trabalho foi realizado a fim de propor métodos para a elaboração de soluções de visão embarcadas dedicadas, possibilitando atender requisitos de tempo, consumo e/ou acurácia.

1.1 OBJETIVOS

Nessa seção serão apresentados os objetivos do trabalho aqui exposto. Primeiramente será descrito o objetivo geral e, a seguir, os objetivos especificos e contribuições esperadas para essa dissertação.

1.1.1 Objetivo Geral

O trabalho aqui apresentado tem por objetivo o estudo da aplicação de redes convolucionais em sistemas de visão embarcados, analisando as atuais soluções e propondo metodologias para maior eficiência.

1.1.2 Objetivos Específicos

Para se alcançar o objetivo geral proposto, tem-se como objetivos específicos os seguintes tópicos:

- Realizar análise bibliográfica e do estado da arte para o caso em questão.
- Definir métricas para a coleta dos dados de tempo de inferência e consumo energético para as redes convolucionais.
- Selecionar, avaliar e realizar o pré-processamento do banco de dados.
- Definir metodologia para o desenvolvimento de solução de aprendizado de máquina utilizando modelos de duas saídas complementares para a detecção de padrão em uma imagem.
- Identificar a influência dos diversos parâmetros de uma rede convolucional no seu tempo de inferência, consumo energético, acurácia e tempo de treinamento.
- Definir metodologia de dois estágios de inferência.

1.1.3 Contribuições

- Análise das camadas CNN em um sistema embarcado no seu desempenho temporal, consumo energético, acurácia e tempo de treinamento.
- Descrição de metodologia de saidas complementares para geração de modelos convolucionais.
- Descrição de metodologia de duplo estágio de inferência utilizando modelo complementar e modelo tradicional.

1.2 ORGANIZAÇÃO DO TRABALHO

Além dessa seção, que apresenta a introdução em conjunto com os objetivos que se espera alcançar, o presente trabalho é dividido em mais cinco capítulos, sendo eles: Fundamentação Teórica, em que será abordada a base teórica do estudo, englobando conceitos de visão

computacional, inteligência artificial, sistemas embarcados, além de plataformas e componentes utilizados; Trabalhos Relacionados, em que serão apresentados estudos já realizados a respeito do tema abordado; Metodologia, em que serão explanados os prodecimentos e os métodos aplicados na elaboração do trabalho; Resultados, capitulo que apresentará os resultados obtidos no estudo aqui apresentado; e Conclusão, em que serão apresentadas as conclusões do trabalho, bem como possibilidades de novas pesquisas.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo serão apresentadas as bases teóricas pertinentes ao trabalho aqui exposto. A revisão é dividida em 5 seções, nas quais serão abordados os temas: Visão Computacional, Inteligência Artificial, Sistemas Embarcados, Agricultura, Plataformas e Componentes.

2.1 VISÃO COMPUTACIONAL

Segundo Bradski e Kaehler (2008), visão computacional é a transformação dos dados de uma imagem ou video em uma decisão ou em uma nova forma de representação. Para os seres humanos, que são criaturas altamente visuais, pode parecer algo bastante simples fazer o reconhecimento de uma cena ou identificar um objeto em uma imagem, porém, para um computador, que trabalha somente com uma grade de números, é uma tarefa bastante desafiadora.

Segundo Prince (2012), apesar do trabalho de várias décadas de diversas mentes criativas, o ser humano ainda está longe de desenvolver um sistema visão de propósito geral. Até o momento, somente sistemas para aplicações bastante especificas foram desenvolvidos com sucesso. Bradski e Kaehler (2008) complementam que os sistemas de visão ainda são bastante ingênuos.

A área de visão computacional sofreu um grande avanço nos últimos anos em virtude da utilização de técnicas de inteligência artificial nos processos de extração de características, detecção, identificação e classificação de imagens, entre outros.

2.2 INTELIGÊNCIA ARTIFICIAL

A inteligência artificial é uma subárea da ciência da computação, cujos Algoritmos, com certo nível de capacidade intelectual, são criados a fim de imitar o comportamento humano em atividades especificas. Segundo Copeland (2018), inteligência artificial é a capacidade de um computador realizar tarefas comumente associadas a seres inteligentes. O termo é normalmente atribuido a sistemas com a habilidade de raciocinar, descobrir significados, generalizar e aprender com experiências passadas. O autor ainda continua definindo a inteligência artificial como sendo um ramo da

ciência da computação que lida com a simulação de inteligência em computadores, atribuindo a eles a capacidade de imitar o comportamento humano. Para Kaplan (2016), existem muitas definições propostas, e o ponto em que elas se alinham é no conceito de criação de uma máquina ou de um programa de computador capaz de um comportamento considerado inteligente, quando apresentado para outras pessoas.

Dentro da inteligência artificial existem duas perspectivas, a IA fraca e a IA forte. Na IA fraca, as aplicações se limitam a imitar o comportamento humano em certas atividades de modo bastante restrito. Toda a inteligência é resultado de insumos fornecidos por humanos. A IA forte se baseia na criação de computadores que possuem consciência própria, sendo eles capazes de tomarem decisões por conta própria, aprender por conta própria e ter noção de suas ações. Para Kaplan (2016), a IA forte postula que as máquinas terão mentes, enquanto que na IA fraca elas simplesmente simulam a inteligência real. Até o momento, todas as aplicações de inteligência artificial desenvolvidas se enquadram como IA fraca.

A inteligência artificial pode ser subdividida dependendo do tipo de aplicação que se deseja implementar. No caso desse trabalho a área de interesse é o aprendizado de máquina (*Machine Learning*), mais especificamente a área de aprendizado profundo (*Deep Learning*), como mostrado na Figura 1.

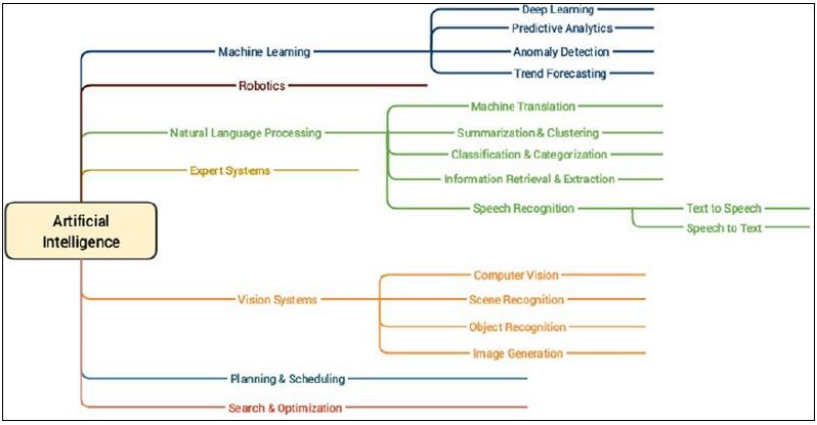

Figura 1: Subdivisões da Inteligência Artificial (Fonte: Sarkar, Bali e Sharma (2018)).

2.2.1 Aprendizado de Máquina

O aprendizado de máquina é a área cujos computadores são treinados para encontrar ou identificar padrões dentro de certo conjunto de dados. Segundo Russell e Norvig (2016), pode-se dizer que uma máquina está aprendendo quando seu desempenho em realizar uma dada tarefa estiver melhorando em virtude da observações dos resultados anteriores. Para Mitchell (1997) o aprendizado de máquina é qualquer programa de computador que melhora o seu desempenho em alguma tarefa por meio de experiência. Experiência pode ser entendida como o erro na saída do processo, ou seja, a distância entre a saída desejada e a saída alcançada.

Dentre os diversos algoritmos de aprendizado de máquina, um dos que mais se destaca são as redes neurais artificiais. Surgidas a partir de uma motivação biológica, elas se sobressaem em função da sua capacidade de aprendizado, generalização e expansibilidade. Esse tipo de rede tem como peça fundamental o neurônio artificial, apresentado juntamente com um esboço de neurônio biológico, como pode ser visualizado na Figura 2. Um neurônio biológico dispõe de diversos terminais receptores, os dendritos, que se conectam por meio de sinapses com os neurônios de níveis anteriores. A partir de certo nível de estímulo a célula irá gerar um sinal de saída resultante por meio dos terminais dos axônios, servindo como entrada para um neurônio de nível superior.

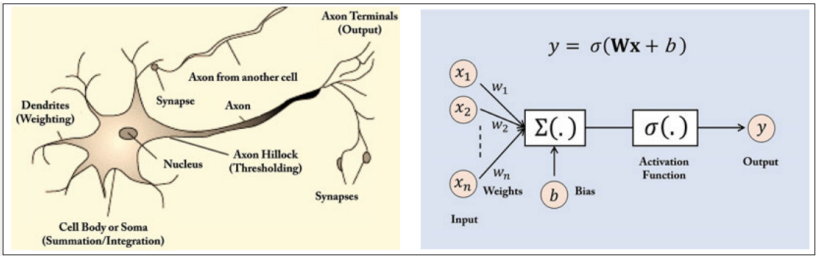

Figura 2: Neurônio Biológico e Neurônio Artificial (Fonte: Khan et al. (2018)).

Analisando novamente a Figura 2, é possível notar que o neurônio artificial é constituido basicamente pelo somatório das entradas X multiplicadas pelos respectivos pesos W. Além disso, adiciona-se um termo b (*bias*), que possibilita uma melhor ajuste do

modelo ao conceito que se deseja aprender. Por fim, o resultado dessa soma é aplicado a uma função de ativação, que introduz uma não linearidade ao sistema, permitindo o aprendizado de abstrações mais profundas. A partir da junção de diversos desses neurônios em uma rede, é possível aprender conceitos bastante complexos.

Existem três grandes classes no aprendizado de máquina, variando no modo como os modelos são treinados, sendo elas: supervisionado, não supervisionado e por reforço.

No aprendizado supervisionado, o modelo é treinado utilizando exemplos previamente classificados do que se deseja aprender, ou seja, o dado de treinamento é composto pelo seu valor de entrada e por um rótulo de saída. Para Zaccone, Karim e Menshawy (2017), nesse caso a questão crucial é o problema de generalização, em que após o treinamento o modelo deve funcionar corretamente para todas as entradas aplicadas a ele.

No treinamento não supervisionado uma série de entradas é aplicada ao modelo, mas desta vez sem nenhum tipo de classificação ou rótulo, um modo de funcionamento mais próximo ao presente no cérebro humano. Segundo Zaccone, Karim e Menshawy (2017), esse tipo de treinamento é particularmente usado em problemas de clusterização[1], em que, dado uma série de elementos, se deseja conhecer a relação entre eles.

No caso do aprendizado por reforço o modelo é treinado por meio de sua interação direta. Após a realização de um movimento ou tarefa, o modelo recebe um *feedback* a respeito da eficácia da sua ação e tem os seus parâmetros ajustados para melhorar em uma próxima execução. Várias aplicações bastante conhecidas de inteligência artificial utilizam essa abordagem, como é o caso de robôs que jogam xadrez, GO[2] ou jogos eletrônicos, uma diversidade de veiculos autônomos, entre outros.

A área de interesse desse trabalho são as aplicações de aprendizado supervisionado, mais especificamente, as de aprendizado profundo (*Deep Learning*).

2.2.2 Deep Learning

Nos modelos tradicionais de aprendizado de máquina aplicados à classificação, somente uma pequena parte do sistema passa pelo

[1] Classificação não supervisionada de dados relacionados ou com características semelhantes.

[2] Jogo de tabuleiro de origem chinesa.

treinamento propriamente dito. Tomando como exemplo a detecção de um objeto em uma imagem, todas as características que definem tal objetivo devem ser extraídas por meio de técnicas e ferramentas computacionais, para então, serem enviadas a um classificador, que será treinado para realizar a detecção. Esse tipo de abordagem demanda bastante tempo e esforço e normalmente não alcança resultados extremamente expressivos.

Para Goodfellow et al. (2016), pode ser muito difícil realizar a extração e a representação de abstrações de tão alto nível dos dados, sendo que vários desses fatores só podem ser identificados por meio de um conhecimento quase humano das informações de entrada.

Para melhorar essa situação, foram desenvolvidas algumas técnicas que, além de habilitar o classificador, realizam o treinamento de um sistema de extração de características. Essa abordagem foi denominada aprendizado profundo, ou *Deep Learning*. A Figura 3 apresenta o comparativo de uma aplicação de *Machine Learning* de uma aplicação de *Deep Learning*.

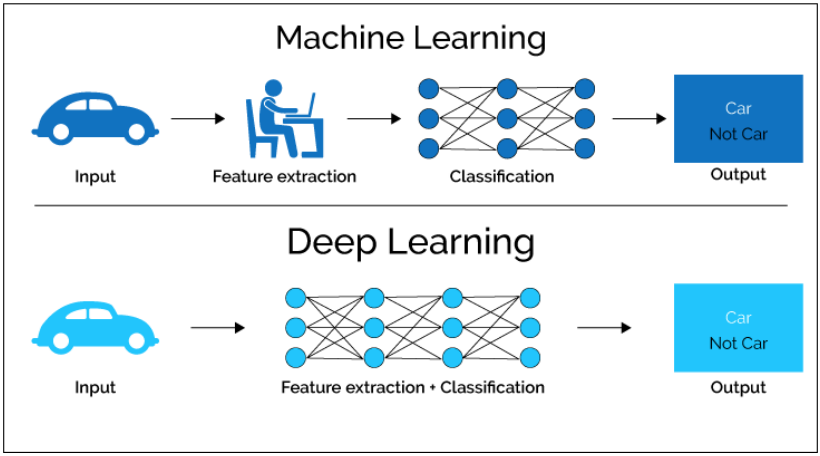

Figura 3: Comparativo do desenvolvimento de soluções de *Machine Learning* e *Deep Learning* (Fonte: Sharma (2018)).

Para Zaccone, Karim e Menshawy (2017), *Deep Learning* é uma área de pesquisa de aprendizado de máquina que é caracterizada pelo esforço em criar um modelo de aprendizado em vários níveis, cujos níveis mais profundos recebem como entrada a saída das camadas prévias, abstraindo cada vez mais a representação.

Goodfellow et al. (2016) dizem que o *Deep Learning* soluciona o

problema de representação do conhecimento, representado-o na forma de modelos mais simples, para, na união deles, gerar conceitos complexos.

Dentro da área de Deep Learning existem diversos tipos de redes disponíveis, dependendo do tipo de treinamento utilizado e da aplicação visada. Entre as mais conhecidas: Redes Neurais Convolucionais, Redes Neurais Recorrentes, Redes Neurais Profundas, Máquinas de Boltzmann Restritas, Redes de Crenças Profundas, Autoencoders, entre outras. No contexto desse trabalho, as redes de interesse são as convolucionais, por serem destinadas a aplicações de visão computacional e também pela compatibilidade com o dispositivo Intel Movidius NCS.

2.2.3 Redes Neurais Convolucionais

Redes Neurais Convolucionais (CNN) são um tipo singular de redes especializadas na análise e na classificação de imagens. Apesar de já existirem há vários anos, ganharam forte destaque a partir de 2012, quando a rede AlexNet, descrita por Krizhevsky, Sutskever e Hinton (2012), foi a grande campeã da ILSVRC-2012, competição de reconhecimento visual. Até então a competição fora vencida por métodos tradicionais de visão computacional. Desde então, somente modelos baseados em redes convolucionais têm vencido a competição. As soluções apresentadas a cada ano no desafio têm melhorado o seu desempenho de maneira bastante significativa, como pode ser visto na Figura 4.

Vale destacar que a, partir de 2015, os resultados obtidos pelas redes vencedoras nessa competição já se mostravam com acurácia superior à humana, cuja taxa de erro se encontra na casa dos 5%.

Pattanayak (2017) diz que a grande vantagem das redes neurais convolucionais é a sua conectividade esparsa que resulta em compartilhamento de pesos, algo que reduz significativamente o número de parâmetros treináveis. O mesmo filtro aprende a detectar certo contorno em qualquer porção da imagem.

Como pode ser visto na Figura 5, uma rede convolucional é composta por um série de diferentes camadas interligadas. A imagem em questão exibe a AlexNet, rede campeã do desafio ILSVRC-2012. A primeira camada representa a entrada do modelo, em que são esperadas imagens no formato 224x224. Os primeiros níveis da rede realizam a extração de caracteristicas das imagens, ao mesmo tempo

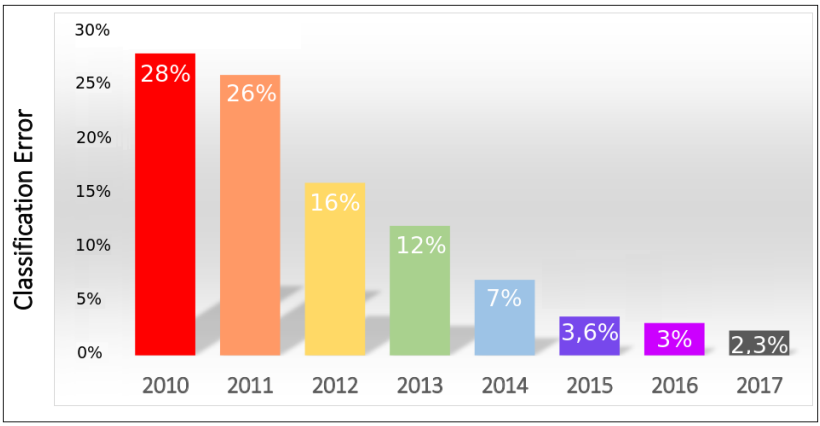

Figura 4: Declínio do Erro de Classificação na competição ILSVRC desde 2010 (Fonte: Park et al. (2017)).

em que reduzem o seu tamanho e complexidade e adicionam certa invariabilidade ao sistema. Nos três níveis finais, é realizada a classificação da imagem inserida, tomando como base os atributos detectados.

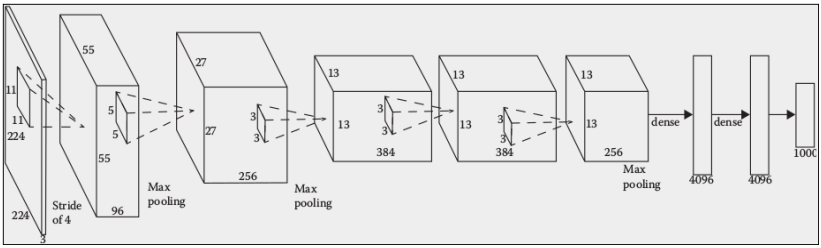

Figura 5: Topologia da Rede Convolucional AlexNet (Fonte: Krizhevsky, Sutskever e Hinton (2012)).

A seguir serão detalhadas algumas das principais camadas que compõem uma rede convolucional.

2.2.3.1 Camada Convolucional

A camada convolucional, principal camada em uma rede CNN, pode ser entendida como uma série de filtros, também chamados de

kernels, os quais possuem a finalidade de extrair e detectar certos padrões em uma imagem, partindo de um nível de abstração extremamente baixo (linhas verticais ou horizontais, bordas, cores), até um nível de abstração superior, presente nas camadas finais. Como dito por Khan et al. (2018), os filtros são inicializados de forma aleatória e, durante o treinamento aprendem a identificar as características presentes nas imagens a eles apresentadas. A equação 2.1 descreve o comportamento da camada.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n), \qquad (2.1)$$

onde S é o resultado da equação, I e K são, respectivamente, a imagem e o filtro, m e n são as dimensões dos filtros e i e j são as coordenadas das posições na imagem.

Buduma e Locascio (2017) descrevem o funcionamento de uma camada convolucional como, a partir de um detector de caracteristicas (filtro), iniciando do topo da imagem, o filtro é deslocado e a cada passo é verificada a existência de uma correspondência entre ele e a figura. Havendo uma relação, a saída do filtro ficará em destaque. O resultado de todo esse procedimento é um mapa de caracteristicas. A Figura 6 mostra um exemplo utilizando dois filtros por meio de uma imagem e os seus mapas de caracteristicas resultantes.

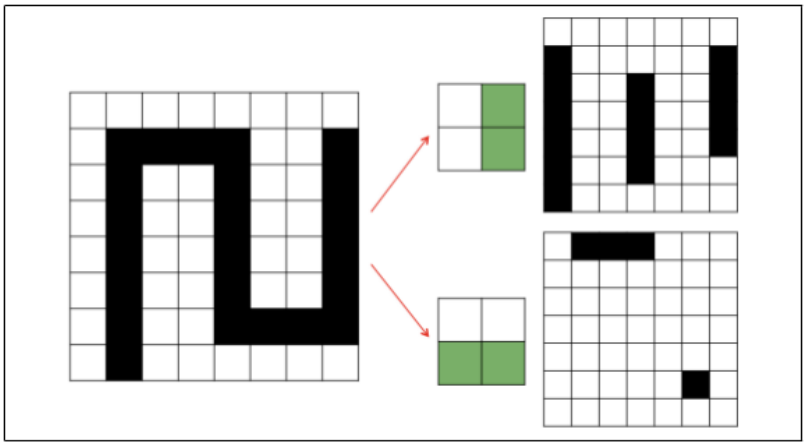

Figura 6: Exemplificação do Funcionamento de uma Camada Convolucional (Fonte: Buduma e Locascio (2017)).

É importante, durante a utilização de uma camada convolucional ficar atento aos hiperparâmetros a ela atribuídos. A escolha correta deles pode definir o sucesso ou não da aplicação. Alguns dos parâmetros mais importantes das redes convolucionais são: quantidade de filtros na camada, tamanho dos filtros na camada, número de pixels de deslocamento (*stride*) e *padding*[3], visto que após a convolução a imagem têm a sua dimensão reduzida. É importante salientar que um *stride* maior que 1 leva à redução da dimensão da imagem analisada.

2.2.3.2 Camada de *Pooling*

Após a aplicação de um ou mais níveis convolucionais é comum a utilização de uma camada de *pooling*. Pattanayak (2017) explica a camada de *pooling* como a operação que resume uma região da imagem, sendo essa região normalmente de tamanho 2x2. Esse resumo ocorre na forma de *max-pooling*, cujo o píxel de maior intensidade é tomado como o representativo do local, e de *average-pooling*, em que é feita a média entre os valores da região.

Para Khan et al. (2018), a operação de *pooling* consegue efetivamente reduzir a complexidade do mapa de características, sendo muito útil para obter uma representação compacta, invariante a variações moderadas nas escalas, posição e translação de objetos na imagem. Essa camada dispõe de dois parâmetros configuráveis, sendo eles o tamanho da região analisada, que é sempre quadrado; e o deslocamento (*stride*), que é a distância entre uma ponto de *pooling* e outro dentro da imagem. Por meia da Figura 7, mostra-se a utilização de uma camada de *pooling* com tamanho 2x2 e *stride* de 1.

2.2.3.3 Camada Densa

As camadas densas, também conhecidas como camadas totalmente conectadas, são a representação de uma rede neural tradicional dentro do modelo convolucional. Normalmente sua função é realizar classificação e atuar como saída do modelo, porém alguns exemplares já utilizaram esse tipo de camada em níveis intermediários com sucesso.

[3]Técnica que aplica zeros às bordas de uma imagem após a convolução para que ela, a imagem, mantenha o seu tamanho original.

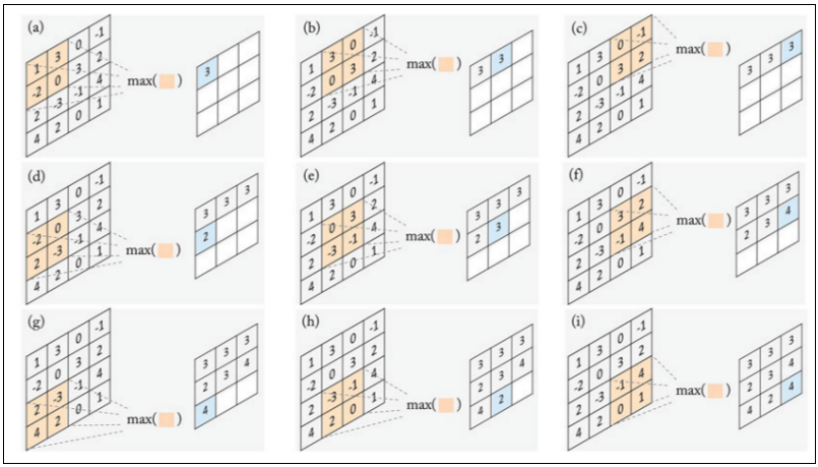

Figura 7: Funcionamento de uma camada de *Pooling* (Fonte: Khan et al. (2018)).

Como explica Khan et al. (2018), sua operação pode ser representada como uma multiplicação matricial, seguida da adição de um termo constante (*bias*) e da aplicação de um função não linear *f*, como descrita na equação 2.2:

$$y = f(W^T x + b) \tag{2.2}$$

onde x e y são respectivamente a entrada e a saída da camada, W^T representa a matrix de pesos transposta, e b o vetor de *bias*.

De forma prática, cada neurônio de uma camada está ligado a todos os neurônios da camada anterior e da camada posterior, como apresentado na Figura 8.

2.2.3.4 Função de Ativação

Nas redes neurais em geral, é comum, após a utilização de uma camada com pesos (convolucional, densa), a utilização de uma função de ativação, ou seja, uma função matemática não linear que é aplicada ao resultado dos níveis de cada rede.

Para Khan et al. (2018), a aplicação de uma função não linear após a camada de pesos, é muito importante, visto que isso permite que uma rede neural aprenda abstrações não lineares. Na ausência

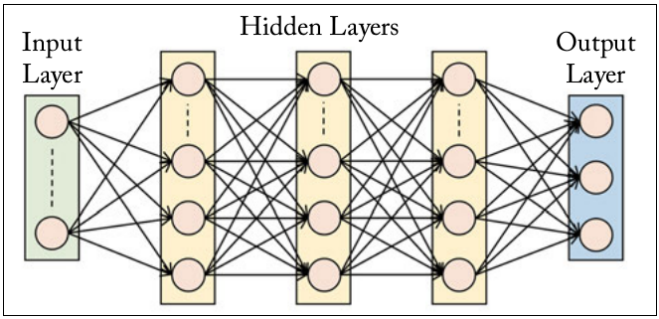

Figura 8: Funcionamento de uma camada Densa (Fonte: Khan et al. (2018)).

de não linearidades, a rede se torna um simples mapeamento linear entre entrada e saída. Buduma e Locascio (2017) complementam que no intuito de aprender relações complexas, é necessária a utilização de alguma não linearidade.

Existem várias opções de funções de ativação disponiveis para a utilização nas redes neurais, como pode ser visto na Figura 9.

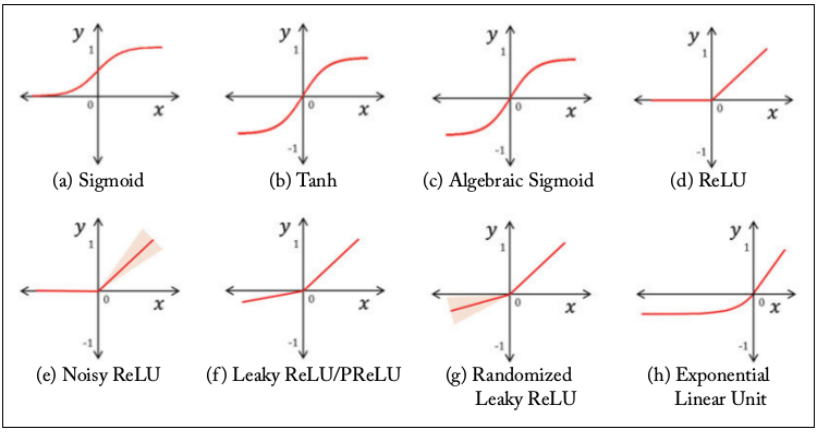

Figura 9: Principais Funções de Ativação (Fonte: Khan et al. (2018)).

Atualmente, a função que possui melhor aceitação e traz resultados mais expressivos é a *ReLU*. Ela possui uma rápida computação e sua utilização atenua o problema de desaparecimento do gradiente em redes extensas, que ocorre principalmente na

utilização das funções *Sigmoid* e *Tanh*. A função mapeia a saída para 0 se o valor da entrada for negativo, ou no caso do valor ser positivo, envia-o diretamente à saída. Matematicamente a *ReLU* é representada pela equação 2.3:

$$f_{relu}(x) = max(0, x) \tag{2.3}$$

2.2.3.5 *Dropout*

As redes neurais dispõem de algumas ferramentas, denominadas métodos de regularização, que visam otimizar os resultados da etapa de treinamento, seja melhorando a acurácia, evitando *overfitting*[4] ou acelerando o processo de treino.

Quando se utilizam camadas densas na rede, um dos métodos mais comuns de evitar o *overfitting* é por meio da utilização do *Dropout*. Esse método pode ser aplicado em camadas convolucionais, mas não é algo comum e seus ganhos não são significativos. Como Zaccone, Karim e Menshawy (2017) explicam, o*Dropout* é posicionado após os níveis que possuem uma grande quantidade de neurônios treináveis, como é o caso das camadas totalmente conectadas.

O funcionamento da técnica de *Dropout* se dá efetuando o desligamento randômico de alguns neurônios, normalmente 50%, da camada afetada. Pattanayak (2017) explica que uma quantidade especificada de neurônios é aleatoriamente desconectada durante o processo de treinamento, para que os neurônios remanescentes possam aprender características importantes por si só, sem contar com a cooperação dos demais. Pattanayak (2017) ainda diz que uma cooperação excessiva entre os neurônios faz com eles se tornem dependentes entre si, o que leva a dificuldades em aprender características distintas. Um alto nível de cooperação leva ao *overfitting*, já que o modelo se comporta bem durante o treinamento, mas falha ao realizar predições em dados desconhecidos.

Por meio da Figura 10, pode-se visualizar o comportamento do *Dropout* durante o treinamento de uma rede. Vale ressaltar que a técnica só deve ser aplicada durante a etapa de treino. Durante o processo de inferência, para que se alcance a melhor acurácia, todos os neurônios devem ser utilizados.

[4]Situação cujo o modelo apresenta elevada acurácia nos dados de treino e baixa acurácia nos de teste. Diz-se que o modelo decorou os dados de treinamento.

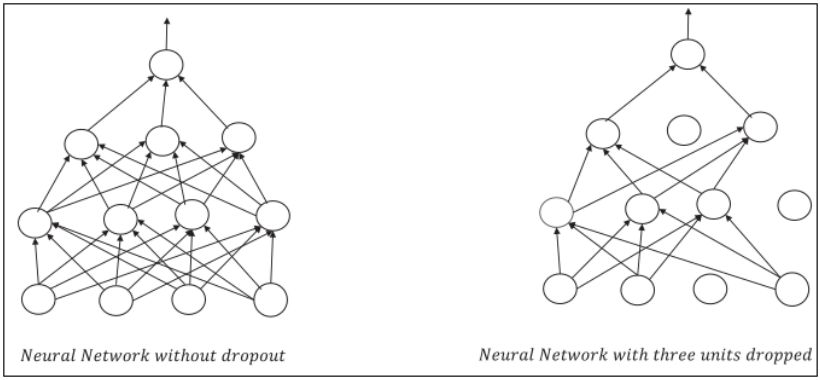

Figura 10: Exemplificação do Funcionamento de uma camada de *Dropout* (Fonte: Pattanayak (2017)).

2.2.3.6 *Batch Normalization*

Outra técnica bastante utilizada, que visa melhorar a acurácia, reduzir o tempo de treinamento e evitar o *overfitting*, é o *Batch Normalization*. Para Ioffe e Szegedy (2015), treinar uma rede neural profunda é complicado em função das mudanças na distribuição das entradas em cada camada em virtude dos parâmetros da camada anterior, fenômeno esse conhecido como deslocamento de covariância interna. Isso acaba tornando o treinamento mais lento, prejudica a convergência do modelo e exige um maior cuidado na inicialização de certos parâmetros.

Diferente de algumas técnicas de regularização que são aplicadas ao *dataset* completo, o *Batch Normalization* é utilizado em pequenas parcelas, permitindo com que ele faça parte da modelo construido. Por meio da utilização do valor da média e da variância dessa parcela de dados, faz-se possível realizar a regularização da rede. Além disso, é admissivel nesse processo a aplicação de deslocamento e/ou escalonamento no resultado gerado. A equação 2.4 mostra como o *Batch Normalization* é aplicado para cada elemento de um conjunto:

$$x = \frac{\gamma * (xi - \mu)}{\sigma} + \beta, \tag{2.4}$$

onde x é o valor resultante da normalização, γ é o valor de escalonamento, xi é o valor de entrada, μ é o valor calculado da média

do conjunto, σ é o valor calculado da variância do conjunto em função do elemento xi, e β é o valor do deslocamento.

2.3 SISTEMAS EMBARCADOS

A princípio, as aplicações de aprendizado de máquina se limitavam a sistemas centralizados de alto desempenho, como servidores ou *datacenters*, cujos modelos, que são computacionalmente exigentes, podem executar com eficiência. Com a evolução dos processadores compactos, as aplicações de *machine learning* puderam ser embarcadas rodando com resultados satisfatórios. Essa seção tratará dos sistemas embarcados e de sua aplicação para processamento descentralizado, o *Edge Computing*.

Sistemas embarcados, como explicado por Edwards et al. (1997), é um sistema computacional que executa função especifica, dispondo de componentes programáveis, cujo comportamento pode ser configurado e alterado quando necessário. Suas aplicações podem incluir controle de veículos, produtos eletrônicos em geral, sistemas de comunicação, sensoriamento remoto, eletrodomésticos, entre outros.

Atualmente, grande parte dos sistemas embarcados desenvolvidos contam com microcontroladores como núcleo computacional, porém não só eles estão disponíveis. Outras opções admissíveis às aplicações embarcadas podem ser: ASICs (application-specific integrated circuits), FPGAs (field programmable gate arrays), DSPs (digital signal processors), cuja opção vai depender da necessidade da aplicação. Segundo Edwards et al. (1997), uma das partes mais significativas no desenvolvimento de um sistema embarcado é decidir a arquitetura de *software* e *hardware* do sistema, e quais partes devem ser implementadas em *software* ou por um *hardware* mais especializado.

Sistemas embarcados podem ser do tipo genérico, por meio de plataformas de desenvolvimento, ou dedicados, em que para uma determinada aplicação, um sistema é desenvolvido. Sistemas dedicados tendem a ser bem mais otimizados, tanto em tempo de processamento, recursos computacionais e energéticos. Por outro lado, sistemas genéricos permitem um tempo de desenvolvimento bem mais curto.

Atualmente uma das aplicações mais notáveis de sistemas embarcados se encontra na aquisição e no processamento de dados em nós, conhecida como *Edge Computing*, aplicação essa que exige um

alto poder computacional proveniente principalmente de sistemas dedicados.

2.3.1 *Edge Computing*

Nos últimos anos, o mundo do processamento de dados foi dominado pela computação em nuvem (*Cloud Computing*), em que os dados coletados por um dispositivo são encaminhados para um servidor de alto desempenho e lá processados. Segundo Satyanarayanan (2017), a computação em nuvem apresenta dois grandes benefícios. Primeiro, a centralização reduz o custo de administração e operação do sistema; e, segundo, as empresas podem evitar o gasto da criação de um *datacenter* próprio utilizando os recursos computacionais de um provedor. Para o autor, o *Cloud Computing* deve permanecer permanentemente como uma opção no cenário computacional, mas não a única.

A *Edge Computing*, que segundo Adegbija et al. (2018) é a execução de processamentos, interpretações e uso de dados nos nós das redes, tem se mostrado uma alternativa viável aos processamentos em nuvem que costumam apresentar uma latência superior. Ainda segundo o autor, executando o processamento no nó minimiza a transmissão de dados, melhorando a latência, a largura de banda e o consumo energético. A ideia é que somente o resultado do dado processado seja transmitido, quando houver necessidade, para um *datacenter* ou unidade destino.

Uma das áreas que mais se beneficia com os avanços da *Edge Computing* é a Internet das Coisas (IoT). Segundo Shi e Dustdar (2016), certas aplicações de IoT exigem um tempo de resposta extremamente pequeno, algumas possuindo informações privativas e outros produzindo quantidades gigantescas de dados e, por isso, a computação em nuvem não pode atender a essas necessidades. Por outro lado, a *Edge Computing* pode e ainda irá possibilitar a criação de muitas outras aplicações.

Alguns trabalhos já desenvolvidos mostram os beneficios alcançados pela utilização da *Edge Computing*. Yi et al. (2015) mostram em seu trabalho, uma melhora na latência de um sistema de reconhecimento facial de 900 para 169 ms entre um sistema em nuvem para um de processamento em nó. Já Ha et al. (2014) apresentam um sistema que obteve redução na latência de até 200 ms, além de alcançar economia no consumo energético de até 40%.

2.4 PLATAFORMAS E COMPONENTES

Nessa seção, serão apresentadas as plataformas computacionais e os componentes utilizados durante o desenvolvimento do trabalho. O objetivo é fornecer uma visão geral sobre as partes.

2.4.1 Linguagem de Programação Python

Python é uma linguagem de programação interpretativa, orientada a objetos, de propósito geral e alto nível de abstração. Foi desenvolvida por Guido van Rossum no final da década de 80 e desde então se tornou uma das linguagens mais populares em virtude da sua sintaxe clara, simples e de rápido desenvolvimento. É uma das linguagens de programação com maior comunidade de desenvolvedores, algo que permite uma rápida e constante evolução da linguagem e das ferramentas disponíveis.

A enorme quantidade de ferramentas desenvolvidas para Python torna a linguagem amplamente usada e difundida. Entre suas aplicações mais comuns encontram-se: aplicações matemáticas, análise de dados, aplicações Web em conjunto com outras linguagens, aplicações de Desktop GUI, educação, aplicações em conjunto com banco de dados, mineração de dados, robótica, visão computacional, elaboração de scripts, inteligência artificial e aprendizado de máquina, o foco do trabalho aqui apresentado. Entre as ferramentas disponíveis em Python para aprendizado de máquina, pode-se destacar a biblioteca TensorFlow. Já para visão computacional, a biblioteca mais notável é a OpenCV.

2.4.1.1 Biblioteca de Aprendizado de Máquina TensorFlow

TensorFlow é uma biblioteca de *machine learning* de código Aberto, desenvolvida pela Google, destinada à pesquisa e à produção. A biblioteca oferece interfaces de programação de aplicações (API) para diversas finalidades (Desktop, Mobile, Web, Cloud). Diferente do *framework* Caffe, que é destinada para aplicações de visão, o Tensorflow é capaz de executar boa parte dos algoritmos de aprendizado de máquina disponiveis, sendo que, para alguns, a biblioteca dispõe de ferramentas específicas, que tornam o desenvolvimento mais rápido e simples.

Apesar de ser utilizado majoritariamente em Python, o TensorFlow é implementado em C++, fato esse que lhe confere ótimo desempenho e tempo de execução. Além disso, outras linguagens de programação possuem suporte à biblioteca, porém com recursos reduzidos, tomando como exemplo: C, Java, C++, C#, R, entre outras.

Um código escrito utilizando a biblioteca recebe o nome de Graph, e é composto por operações e tensores. Segundo Abadi et al. (2016), no TensorFlow cada operação é um nó no Graph e representa uma unidade de computação local, ou seja, realiza uma ou um conjunto de operações matemáticas, enquanto que os tensores são os dados que fluem pela malha.

A biblioteca dispõe de uma ferramenta de *debug*, otimização e visualização chamada TensorBoard. Com ela é possível monitorar qualquer peso ou parâmetro presente no modelo desenvolvido, a fim de melhorar a compreensão do resultado obtido. O TensorBoard permite a visualização do Graph gerado, além de apresentar em gráficos as grandezas armazenadas durante o treinamento, como a progressão dos pesos em uma determinada camada, ou a variação do erro ou da acurácia do modelo durante o treinanento.

2.4.1.2 Biblioteca de Visão Computacional OpenCV

O OpenCV (*Open Source Computer Vision Library*) é uma ferramenta originalmente desenvolvida pela Intel para fins acadêmicos ou comerciais de uso livre destinada à visão computacional. Para Pulli et al. (2012), a biblioteca tem como objetivo fornecer ferramentas para a resolução de problemas de visão computacional, contendo um misto de funções de baixo nível e algoritmos de alto nível, como para detecção facial, detecção de pessoas, objetos, entre outros.

Neste trabalho ela foi usada para realizar o pré-processamento de imagens, porém, sua aplicabilidade não se restringe a isso, podendo ser utilizada para: sistemas de reconhecimento facial, reconhecimento de gestos, detecção de objetos, realidade aumentada, rastreamento de movimento, além de dispor de algumas ferramentas de aprendizado de máquina. Possui interface para Python, C++ e Java e compatibilidade com diversos sistemas operacionais, além de poder ser embarcado em aplicações mobile.

2.4.2 Plataforma de Hardware Dedicado Intel Movidius NCS

Movidius é uma empresa do grupo Intel destinada ao desenvolvimento de soluções embarcadas de consumo energético ultrabaixo para a implementação de modelos de *Deep Learning* e Visão Computacional. A empresa é a fabricante do Intel Movidius Neural Compute Stick (NCS), um dispositivo destinado à inferência de redes convolucionais que, em conjunto com o Intel Movidius Neural Compute SDK (*Software Development Kit*), compõe a plataforma de desenvolvimento Movidius. Esse sistema permite o aprendizado e o teste de soluções em visão computacional, possibilitando uma rápida prototipagem, validação e desenvolvimento de aplicações, sem a necessidade de conexão com servidores ou serviços de nuvem.

O Movidius SDK é um conjunto de ferramentas computacionais destinado à conversão e à otimização das redes convolucionais descritas, utilizando os frameworks TensorFlow ou Caffe em um modelo compatível com o NCS. Durante o processo de otimização e conversão, um relatório de desempenho é apresentado, sendo possível extrair algumas informações camada a camada do modelo, como pode ser verificado na Tabela 1:

Tabela 1: Dados Fornecidos por Camada pelo Movidius SDK.

Complexidade da camada	**Largura de Banda**	**Tempo**
MFLOPS	MB/s	ms

A Figura 11 mostra uma parte da imagem descritiva gerada durante a compilação da rede pelo Movidius SDK. Por meio dela é possível verificar que na rede em questão espera-se como entrada uma imagem com formato (448,448,3) pixels. Além disso, é possível constatar que, durante a compilação, a rede foi otimizada para utilizar 12 núcleos de processamento, o tempo para realizar a inferência do modelo é de aproximadamente 115 ms e a largura de banda é em média 892,24 MB/sec. O gráfico disponibiliza as informações camada a camada do modelo, o que possibilita a verificação dos pontos que comprometem o desempenho.

A peça fundamental no Intel Movidius NCS é o chip Intel Movidius Myriad 2 VPU (*Visual Processing Unit*). Ele é um VPU SoC (*System-on-a-Chip*) dedicado para aplicações de visão computacional que visam ao baixíssimo consumo energético.

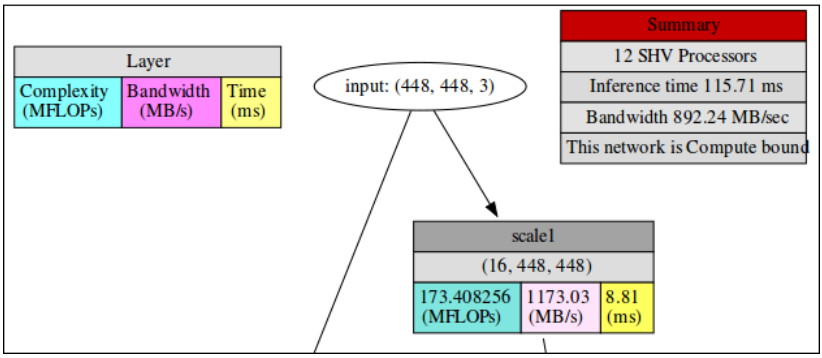

Figura 11: Gráfico gerado durante a compilação utilizando o Movidius SDK.

O chip Myriad 2 dispõe de 12 núcleos VLIW (*Very Long Instruction Word*) programáveis, denominados SHAVE, 20 aceleradores dedicados de visão e 2 CPUs. Segundo Barry et al. (2015), o SHAVE é um processador de fluxo híbrido que combina as melhores características das GPUs, DSPs e RISC, podendo realizar operações aritméticas de inteiros com 8, 16 e 32 bits ou de ponto flutuante com 16 e 32 bits, além de suporte em hardware para estruturas de dados esparsas. Ainda segundo o autor, o chip alcançou eficiência de 1188 GOPS[5]/W em operações com inteiros de 16 bits.

O Myriad 2 possui arquitetura de 28 nm e opera a 600 MHz. O design altamente paralelizado torna ele bastante adequado para aplicações com redes neurais profundas e demais sistemas de visão. Por meio da Figura 12, pode verificar a estrutura do chip.

2.4.3 Servidor com Processador Intel Xeon Phi

O servidor disponível para a realização dos treinamentos dos modelos desse trabalho possui um processador Intel Xeon Phi 7250, que conta com 68 núcleos de 1,4 GHz e um total de 34 MB de L2 Cache.

O Intel Xeon Phi é uma família de processadores desenvolvida pela empresa Intel que oferece grande paralelismo e vetorização, sendo indicado para aplicações de alto desempenho. Os dispositivos dessa

[5] *Giga Operations Per Second* (Giga Operações por Segundo)

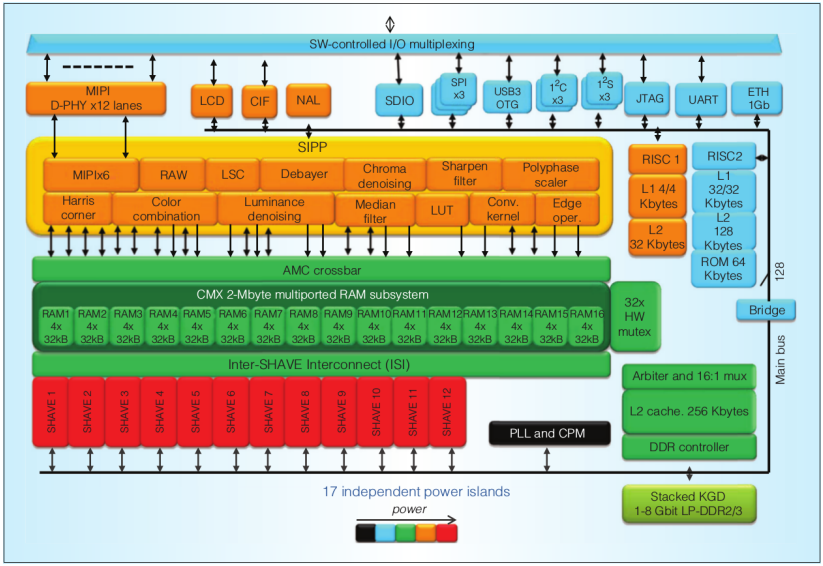

Figura 12: Estrutura Interno do chip Myriad 2 (Fonte: Barry et al. (2015)).

linha prometem maior poder computacional por unidade de energia consumida. Os modelos da atual geração são fabricados com tecnologia de 14 nm, possuem de 64 a 72 núcleos, de 32 a 36 MB de memória L2 Cache e trabalham com frequência base de 1,3 GHz a 1,5 GHz.

2.4.4 Plataforma de Desenvolvimento Raspberry Pi

Segundo Brock, Bruce e Cameron (2013), o Raspberry Pi é um computador de tamanho reduzido, desenvolvido pelo Laboratório de Computação da Universidade de Cambridge em 2012. Possui baixo custo, é capaz de executar o sistema operacional Linux e dispõe de entradas e saídas de uso geral, para conexão com sensores, atuadores e afins.

A placa é composta por um poderoso processador ARM, processador gráfico 3D capaz de gerar vídeos em alta definição, memória RAM, cartão SD, além de conexões externas que variam dependendo o modelo. Atualmente se encontra na terceira geração de placas, sendo que a partir da segunda geração a placa já dispunha de

processadores com 4 núcleos. Os modelos disponíveis do Raspberry Pi podem ser vistos na Tabela 2.

Tabela 2: Modelos da Plataforma Raspberry Pi.

Modelo	**CPU**	**SDRAM**	**Frequência**	**Potência**
RPi Zero	ARM1176JZF-S	512 MB	1 GHz	1.75W
RPi Zero W	ARM1176JZF-S	512 MB	1 GHz	1.75W
RPi 1 Model A+	ARM1176JZF-S	512 MB	700 MHz	-
RPi 1 Model B+	ARM1176JZF-S	512 MB	700 MHz	1.75W
RPi 2 Model B	Cortex-A7	1 GB	900 MHz	4.1W
RPi 3 Model B	Cortex-A53	1 GB	1.2 GHz	6.7W
RPi 3 Model B	Cortex-A53	1 GB	1.4 GHz	5.66W

Nesse trabalho o modelo utilizado é o Raspberry Pi 3 Model B, que foi empregado para a realização de alguns testes de tempo de inferência das redes e também como placa auxiliar na medição de consumo energético em conjunto com um sensor de potência INA219.

2.4.5 Sensor de Potência INA219

O INA219 é um sensor *shunt* que opera por meio de interface i2c, capaz de medir corrente, potência e tensões. Segundo Texas (2015), o sensor provê leituras digitais de corrente, tensão e potência necessárias para sistemas de controle e tomada de decisão com precisão. Registradores programáveis permitem configurações flexiveis de resolução e modos de operação. O sensor permite a leitura da tensão diferencial sobre o resistor *shunt* e a leitura da tensão de alimentação do sistema.

A principal vantagem do INA219 frente a outros sensores de potência é a sua utilização no lado superior da alimentação. Isso permite que o equipamento, cujas grandezas serão medidas, permaneça conectado diretamente à referência, além de possibilitar a medição da tensão de alimentação do sistema.

3 TRABALHOS RELACIONADOS

Nesse capítulo, são apresentados os trabalhos relacionados ao tema aqui abordado, descrevendo o que já foi desenvolvido e proposto, bem como seus métodos e resultados.

3.0.1 TRABALHOS CORRELATOS

O *Edge Computing* é uma área que está bastante em alta, principalmente em virtude de temas como IoT e Indústria 4.0. Até pouco tempo, o *Cloud Computing* era a referência em processamento de dados e tomada de decisões, porém, com a crescente expansão da quantidade de dados gerados a cada instante torna-se inviável a sua utilização exclusiva. Segundo Networking (2018), estima-se que em 2021 a quantidade de dados gerados por ano seja de 847 ZB, enquanto que a capacidade instalada dos data centers será de 20,6 ZB ao ano. Se for do interesse processar essa quantidade de dados excedente, outros meios precisarão ser utilizados, além dos data centers e, para isso, os dispositivos de *Edge Computing* podem auxiliar.

Em Shi et al. (2016), os autores abordam o tema *Edge Computing*, apresentando a sua definição, as vantagens da sua aplicação frente às estratégias atuais, além de exporem alguns estudos de caso, desafios e oportunidades futuras. No artigo dos autores, buscou-se mostrar que é mais eficiente realizar o processamento dos dados próximo ao local onde ele foi produzido, uma vez que o transporte das informações já está se tornando o gargalo nas aplicações de *Cloud Computing*. Algumas vantagens apresentadas pelos autores são com relação aos requisitos de latência das aplicações, limitações no tempo de vida de baterias, custo do transporte de dados, segurança e privacidade, sendo todos esses pontos beneficiados pelas aplicações do *Edge Computing*.

Em Satyanarayanan (2017), o autor cita a tendência, desde os anos 60, entre a alternância dos paradigmas centralizadores e descentralizadores. Há até pouco tempo, a computação em nuvem (processamento centralizado) era o grande trunfo do processamento de dados. Porém, em virtude da grande quantidade de dados sendo e a ser gerada, já se vê uma mudança acontecendo, com a análise das informações sendo feita cada vez mais próxima da sua origem e de forma descentralizada.

Alguns trabalhos já foram realizados a respeito da detecção de doenças em plantações, principalmente visando à melhora de condições de plantio em regiões mais precárias de tecnologia agricola, como Índia e alguns países do sudeste asiático. A maioria dos trabalhos relacionados utiliza o *dataset Plant Village*[1], o mesmo utilizado nesse trabalho. Esse *dataset* consiste em imagens de diversos tipos de plantas, e para cada uma delas, diversas variedades de doenças. Além disso, plantas saudáveis também são apresentadas como referência.

Em Mohanty, Hughes e Salathé (2016), os autores, por meio do *dataset Plant Village* realizaram a análise de dois modelos de rede conhecidos, *AlexNet* e *GoogLeNet*, em duas abordagens diferentes. A primeira, por meio de *Transfer Learning*, cujas redes, previamente treinadas com o *dataset Imagenet*, tiveram suas camadas de classificação retreinadas para o *dataset* desejado. A segunda abordagem consistia no treinamento completo, *from Scratch*, das mesmas redes. Os melhores resultados foram alcançados com a rede *GoogLeNet* em *Transfer Learning*, com acurácia de 99,34%.

Em Pooja, Das e Kanchana (2017) os autores realizaram um estudo, utilizando *dataset* próprio bastante limitado, somente 348 amostras, com 5 variedades de doenças. Nesse trabalho, os autores não aplicaram técnicas de *Deep Learning*, foram utilizadas técnicas de segmentação e extração de características e, por fim, um classificador *Support Vector Machine* (SVM), o que alcançou uma acurácia de 92,4%. Essa abordagem permite a utilização de menos dados para treinamento do modelo.

Em Hlaing e Zaw (2017), é apresentado um modelo para classificação de doenças em folhas de tomate. As imagens utilizadas são provenientes do *dataset Plant Village*. Como no trabalho anterior, os autores optaram pela extração manual das caracteristicas das imagens. Foram utilizadas técnicas para remoção do *background*, detecção de regiões saudáveis e doentes, além da adoção do algoritmo SIFT para a extração de características de textura. Como classificador foi utilizado o *Support Vector Machine*. Sua acurácia alcançada foi de 84,7%. Essa abordagem apresenta um tempo de treinamento bastante rápido, não superando 1 minuto. Esse trabalho visa aplicações off-line embarcadas em *smartphones*.

Em Gandhi et al. (2018), os autores realizaram sua pesquisa baseados no trabalho de Mohanty, Hughes e Salathé (2016). Do mesmo modo, foi utilizado o *dataset Plant Village* para a classificação

[1] Disponível em: https://github.com/spMohanty/PlantVillage-Dataset

de todas as classes disponíveis no *dataset*. Nesse trabalho os autores utilizaram as redes *Inception v3* e *MobileNet* para os estudos. Os autores utilizaram somente a estratégia de *Transfer Learning*, porém, além disso, foi utilizada uma rede GAN para realizar *data augmentation*. A rede *Inception v3* alcançou uma acurácia de 88,6%, enquanto a *MobileNet* teve 92%.

Em Ferentinos (2018), o autor utiliza novamente o *dataset Plant Village* para os seus estudos. Nesse trabalho é realizado o treinamento de 5 modelos de redes convolucionais, sendo elas: *AlexNet*, *AlexNetOWTBn*, *GoogLeNet*, *Overfeat* e *VGG*. Com essas redes, buscaram-se a detecção e a identificação da doença presente em cada folha analisada. A rede *VGG* obteve o melhor resultado de acurácia, com 99.53%, rodando por 48 épocas, com tempo médio de 7294 s/época.

De modo semelhante, em Tlhobogang e Wannous (2018), os autores apresentam a proposta de realizar o *Transfer Learning* das redes *GoogLeNet* e *Inception v3* para o *dataset Plant Village*. Os autores propõem o desenvolvimento de uma aplicação *Android* que realiza a comunicação com um servidor que realizaria a inferência da imagem enviada a ele. Além disso, um banco de dados MySQL seria utilizado para a verificação dos resultados. Diferentes das tendências atuais, esse trabalho apresenta uma solução on-line de classificação. Como se trata de uma proposta, não foram divulgados resultados no artigo.

A tabela 3 apresenta a compilação dos trabalhos relacionados selecionados.

Como alternativas mais leves às redes profundas existentes, foram desenvolvidas a rede *SqueezeNet*, apresentada em Iandola et al. (2016); e as redes *MobileNet*, por Howard et al. (2017). Esses modelos apresentam estruturas especiais que otimizam o processo de extração de caracteristicas, alcançando acurácias elevadas com quantidades menores de parâmetros. De modo semelhante os trabalhos citados anteriormente, as redes *SqueezeNet* e *MobileNet* são originalmente treinadas com o *dataset ImageNet*, o que permite a aplicação de *Transfer Learning*. Outra opção para a utilização delas é realizando o treinamento integral (*From Scratch*).

Os idealizadores da rede tomaram como objetivo desenvolver um modelo que apresentasse uma quantidade bastante reduzida de parâmetros, ao mesmo tempo que apresentasse um valor de acurácia expressivo. Para isso, foram adotadas 3 estratégias: ampla utilização de filtros 1x1 em substituição aos filtros 3x3, gerando uma redução de

Tabela 3: Tabela de Trabalhos Relacionados.

Autor	***Dataset***	**Tipo de Rede**	**Acurácia**	**Treinamento**
MOHANTY	Plant Village	Deep Learning	99,34%	*Transfer Learning* + *From Scratch*
POOJA W	Próprio	Support Vector Machine	92,4%	*From Scratch*
HLAING	Plant Village (parcial)	Support Vector Machine	84,7%	*From Scratch*
GANDHI	Plant Village	Deep Learning	92%	*Transfer Learning*
FERENTINOS	Plant Village	Deep Learning	99.53%	-
TLHOBOGANG	Plant Village	Deep Learning	-	*Transfer Learning*

9x no número de parâmetros; diminuição no número de canais na entrada dos filtros 3x3; maior espaçamento entre as camadas de pooling, estratégia que segundo os autores tende a aumentar a acurácia do modelo.

Seguindo as estratégias apresentadas, a rede *SqueezeNet* utiliza em sua arquitetura uma organização de filtros convolucionais denominada *Fire Module*, apresentada na Figura 13.

Na primeira camada do módulo, uma série de filtros convolucionais com dimensão unitária realiza a compressão (*Squeeze*) dos canais dos dados de entrada, reduzindo a quantidade de canais de N para 1. Na sequência são utilizados novamente filtros 1x1 em conjunto com filtros 3x3, ocorrendo ao final a concatenação do resultado de ambos. O *stride* do módulo apresentado é unitário.

Como resultado da implementação descrita, a rede *SqueezeNet* alcançou uma acurácia semelhante à da rede *AlexNet*, utilizando 50x menos parâmetros.

As redes *MobileNet* são um conjunto de modelos que visam à aplicação de visão em sistemas móveis e embarcados. Segundo Howard et al. (2017), em muitas aplicações do mundo real, as tarefas de reconhecimento de imagens exigem baixa latência e que sejam executadas em plataformas computacionais limitadas. Para alcançar

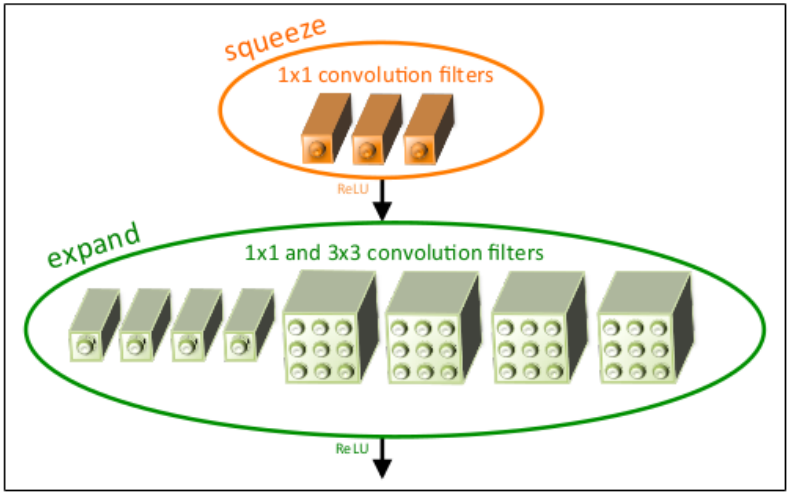

Figura 13: *Fire Module*: arquitetura de organização de filtros convolutionais da rede SqueezeNet (Fonte: Iandola et al. (2016)).

esse objetivo, os autores empregaram estruturas diferenciadas de redes convolucionais, como é o caso das camadas *Depthwise* e *Pointwise*. A Figura 14 mostra um comparativo entre uma camada convolucional regular e as camadas *Depthwise* e *Pointwise*. A junção desses duas estruturas possibilita uma menor quantidade de parâmetros e maior agilidade na execução do modelo.

A rede *MobileNet* apresenta uma estrutura diferente de camadas convolucionais com relação às demais redes, como mostrado na Figura 15, que apresenta o esquemático de uma rede padrão à esquerda; e o esquemático adotado na *MobileNet* à direita. Toma-se como estrutura padrão convolucional o conjunto: camada convolucional 3x3; *Batch Normalization*; ativação *ReLU*. No modelo proposto pelos autores utiliza-se: camada convolucional 3x3 *Depthwise*, camada em que a convolução é aplicada separadamente para cada canal; *Batch Normalization*; ativação *ReLU*; camada convolucional 1x1 (*Pointwise*); *Batch Normalization*; ativação *ReLU*. A utilização das convoluções *Depthwise* e *Pointwise* reduz consideravelmente a quantidade de operações realizadas, o que leva a uma melhora bastante expressiva no tempo de execução. Por outro lado, existe uma leve redução na acurácia do modelo.

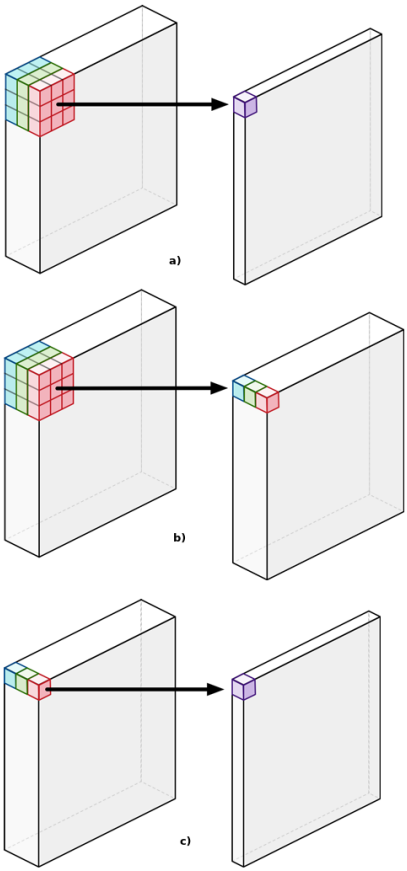

Figura 14: Comparativo de estruturas convolucionais, sendo: a) modelo regular, b) *Depthwise* e c) *Pointwise* (Fonte: Hollemans (2017)).

As redes *MobileNet* dispõem de variantes dependendo da necessidade da aplicação. Essas alternativas incluem variações na complexidade do modelo, cujos modelos mais complexos apresentam uma acurácia superior, e variações no tamanho do dado de entrada.

A Tabela 4 exibe um comparativo entre as redes apresentadas. Como pode ser notado, a rede *MobileNet* de média complexidade (0.5) e de resolução de 160 pixels apresenta acurácia comparável às redes *SqueezeNet* e *AlexNet*. Já a rede *MobileNet*, de maior complexidade (1), e de 224 pixels apresenta resultado consideravelmente superior, porém

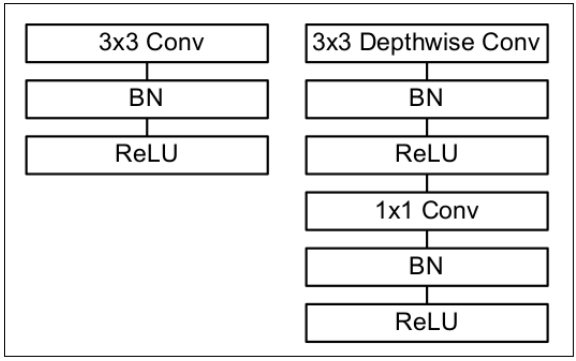

Figura 15: Comparativo entre modelo padrão de rede convolucional e modelo implementado na *MobileNet* (Fonte: Howard et al. (2017)).

ao custo de uma quantidade bem maior de operações de multiplicação e soma.

Tabela 4: Tabela Comparativa entre as redes AlexNet, SqueezeNet e MobileNet.

Modelo	***Acurácia***	**Milhões Mult-Soma**	**Milhões Parâmetros**
0.5 MobileNet-160	60.2%	76	1.32
SqueezeNet	57.5%	1700	1.25
AlexNet	57.2%	720	60
1.0 MobileNet-224	83.3%	569	3.3

Fonte: Howard et al. (2017)

Existe na literatura uma série de trabalhos dedicados a utilização de modelos de aprendizado profundo em sistemas embarcados. As referências apresentadas a seguir seguem essa linha de pensamento.

Em Li, Xiao e Liang (2017) é apresentado um trabalhos focado em redes neurais convolucionais, onde dois pontos foram analisados: o design dos modelos e a aceleração das plataformas embarcadas. Para isso foi apresentada uma técnica de *pruning* diferenciada, além de um

sistema de seleção de algoritmo de multiplicação matricial. O *pruning* é uma técnica que remove os nós com pesos menos significativos em um modelo, reduzindo assim o seu tamanho e complexidade. Normalmente após esse processo é realizado um novo treinamento para o ajuste dos pesos restantes. Os resultados mostraram que houve uma melhora nos resultados comparado a aplicação do *pruning* diretamente, com redução menos significativa nos valores de acurácia. Além disso, houve um redução no tamanho dos modelos e uma melhora no desempenho computacional.

Já em Changpinyo, Sandler e Zhmoginov (2017) foi proposta a redução na quantidade de filtros empregados nos modelos em conjunto com a utilização de conexões esparsas entre as camadas. Para otimizar os resultados, o treinamento foi realizado de forma incremental, aumentando a quantidade de conexões no decorrer do processo. Com isso observou-se uma redução significativa no tamanho dos modelos.

No trabalho Jaiswal e Gajjar (2017), os autores empregam um método de transferncia de conhecimento, também chamada de destilação de conhecimento, utilizando um conjunto de redes professor-aluno, método onde uma rede menor e mais simples (aluno) é treinada através de outra rede mais complexa (professor), copiando o seu comportamento. Nesse trabalho a rede aluno apresentou taxa de absorção de conhecimento de 95

Em Shafiee et al. (2017) foi tomada como base a rede SqueezeNet para a realização dos experimentos. Os autores propõem a redução na quantidade de classes aprendidas pelo modelo em conjunto com uma estratégia de síntese evolucionária que otimiza o aprendizado. Com isso, alcançou-se redução de até 5x no tamanho dos modelos originais com acurácia de até 81% para o sistema com 10 classes do *dataset* ImageNet.

Já em Polino, Pascanu e Alistarh (2018) foram apresentadas duas propostas de técnicas de destilação de conhecimento, a destilação quantificada e a destilação diferenciável, a primeira atuando diretamente no treinamento do modelo aluno, enquanto que a segunda busca otimizar a quantização do modelo aluno para melhor ajuste ao modelo professor. As técnicas apresentaram redução significativa no tempo de execução e no tamanho das redes.

Finalmente, o trabalho Zhu e Gupta (2017) apresenta um estudo a respeito da utilização do *pruning* para a redução dos modelos. É empregado um método gradual de aumento do *pruning* durante o treinamento. Além disso, foi realizado um comparativo

entre modelos campactos-densos e modelos grandes-esparsos, onde as redes que empregaram o *pruning* obtiveram melhores resultados em tamanho e acurácia.

A tabela 5 resume as técnicas empregadas no trabalhos relacionados anteriormente descritos.

Tabela 5: Tabela de Técnicas Empregadas nos Trabalhos Relacionados.

Autor Principal	**Técnicas**
Li (2017)	*pruning*
Changpinyo (2017)	*pruning* / redução de filtros
Jaiswal (2017)	destilação
Shafiee (2017)	redução de classes / síntese evolucionária
Polino (2018)	destilação
Zhu (2018)	*pruning*

Após a definição do problema e implementação da rede que o soluciona, é necessária a utilização de uma plataforma embarcada para a execução do modelo. A escolha do dispositivo irá afetar diretamente o tempo de inferência e o consumo energético da solução. Por outro lado, a acurácia não sofre influência da plataforma. Algumas opções que podem ser utilizadas nessa etapa são: processadores ARM em geral, Raspberry Pi, Beaglebone, Intel Joule, Intel Movidius NCS, SoC Myriad, entre outros.

Em Pena et al. (2017), é feito o comparativo de desempenho dos dispositivos Raspberry Pi 3, Intel Joule 570X e um NCS genérico baseado no SoC Myriad. No trabalho, algumas redes são executadas utilizando os *frameworks Caffe* e *TensorFlow* e os dados de consumo energético e tempo de inferência são coletados. O trabalho mostrou que a utilização de um dispositivo dedicado, como o SoC Myriad, proporcionou tempos de inferência até 4x menor, com um consumo energético semelhante. O Intel Joule apresentou resultados de tempo ainda melhores, porém, com consumo energético bastante superior aos demais.

4 METODOLOGIA

Na maioria das aplicações de identificação e classificação de imagens são utilizadas redes CNN padrões, normalmente pré-treinadas em grandes *datasets*, como *ImageNet*. Quando é empregado o *Transfer Learning*, as camadas de extração de características são preservadas e as camadas de classificação são retreinadas para se adaptar ao *dataset* e à aplicação desejada. A utilização de *Transfer Learning* agiliza e simplifica bastante o processo de desenvolvimento de uma solução, porém, em muitos casos, a utilização de uma rede tão robusta acaba sendo desnecessária, pois um modelo mais simples, mais rápido e que exija menos recursos computacionais poderia ser empregado, principalmente quando se fala em aplicações off-line, o foco do trabalho aqui exposto.

Como mostrado, alguns trabalhos empregam técnicas; como *pruning*, destilação, redução de filtros ou classes; para a diminuição do tamanho e das demandas computacionais dos modelos. Ainda assim, essas técnicas continuam empregando as redes CNN padrões, porém, em um formato mais compacto.

Outro ponto a ser analisado é com relação ao modo como os problemas são tratados e solucionados. Em muitas aplicações que possuem como objetivo um tratamento off-line, uma abordagem mais econômica, principalmente no que se refere à estrutura e à saída das redes, poderia ser aplicada, permitindo o emprego de redes mais simples, rápidas e menos exigentes.

Uma proposta que será apresentada é com relação à utilização de modelos com somente duas saídas com comportamento complementar, modelos esses com a capacidade de detectar ou não algum detalhe específico nos dados. Nesse tipo de abordagem não há a necessidade de realizar uma extração tão abrangente dos atributos das imagens, o que permite a utilização de redes mais simples. Alguns exemplos de aplicações nesse sentido são: presença ou não de doença em folhas, presença ou não de pessoas em uma imagem, presença ou não de obstáculos no caminho de um veículo autônomo, presença ou não de uma doença em uma imagem, entre outras. Dependendo do tipo de aplicação desejado, não existe a necessidade de maiores detalhes para o correto funcionamento da solução.

Outra proposta que será apresentada é a de uma metodologia a ser empregada em aplicações embarcadas, em que exista a necessidade de detecção e classificação de características em uma imagem, visando

um menor tempo total de inferência e um menor consumo energético. Nessa metodologia são utilizadas duas redes independentes, sendo uma delas com duas saídas complementares (menos complexa, mais rápida e de menor consumo energético) e a outra com rede um modelo padrão de múltiplas saídas. A partir do momento que certa característica é detectada em uma imagem, como uma doença em uma folha, a segunda rede é acionada e se realiza a classificação.

O trabalho aqui exposto toma como base o dataset *Plant Village*[1], apresentado inicialmente em Hughes, Salathé et al. (2015), um banco de dados de uso livre, mantido pela Universidade Estadual da Pensilvânia em parceria com uma comunidade global de especialistas em agricultura. O *dataset* é composto por imagens de diversas espécies de plantas, sendo elas de plantas saudáveis ou com algum tipo de doença ou infestação. A figura 16 apresenta alguns exemplares disponíveis no banco de dados.

Figura 16: Exemplo de Imagens Disponíveis no Banco de Dados.

Nesse capítulo, serão detalhados os processos e as etapas realizadas para a elaboração do trabalho aqui exposto.

[1]Disponível em: https://github.com/spMohanty/PlantVillage-Dataset

4.1 ANÁLISE DOS MODELOS APLICADOS AOS TRABALHOS RELACIONADOS

Por meio da análise dos trabalhos relacionados apresentados, fica clara uma tendência entre os autores para a utilização de redes convolucionais pré-desenvolvidas e muitas vezes, pré-treinadas para a rápida solução dos problemas. Porém, para as aplicações mostradas, o emprego de redes originalmente desenvolvidas para a classificação de uma grande quantidade de classes aparenta ser um superdimensionamento da solução.

Para partir de resultados concretos e tomar como base dados para comparações futuras, foram selecionadas diversas redes convolucionais pré-desenvolvidas, modelos esses selecionados a partir dos trabalhos relacionados, para a realização de testes e análise dos seus tempos de inferência e consumo energético.

As redes selecionadas para a realização dos testes são as seguintes: Inception_V1, Inception_V3, Inception_V4, MobileNet, SqueezeNet, AlexNet, GoogLeNet e TinyYolo. Com exceção dessa última, todos os demais modelos são treinados a partir do dataset ImageNet. A escolha dessas redes se deu em virtude de algumas delas serem utilizadas nos trabalhos relacionados e de outras estarem disponíveis no SDK usado.

Dentre as redes apresentadas, os modelos Inception_V1, Inception_V4 e MobileNet são utilizados para a realização de testes de tempo utilizando a plataforma Raspberry Pi, um servidor com o processador Intel Xeon Phi, um notebook contendo a CPU Intel Core i7-6500U de 2.50GHz, além do Movidius NCS. Esses experimentos visam verificar o desempenho da plataforma Movidius frente a outros dispositivos.

As demais redes terão o seu tempo de execução mensurado exclusivamento no Movidius NCS. Com relação aos testes de consumo energético, eles serão realizados para todos os modelos somente na plataforma Movidius.

4.1.1 Medição do Tempo de Inferência

Os métodos para a medição de tempo de inferência utilizando as diferentes plataformas são diferentes e descritos a seguir.

O Movidius NCS, por meio do seu kit de desenvolvimento de software, disponibiliza ao usuário um relatório descritivo da execução

da rede, discriminando os tempos de execução de cada camada do modelo. Por meio dessa ferramenta, os dados são coletados. O tempo de inferência é verificado através do valor médio de 20 execuções do modelo.

Para a medição do tempo utilizando o Raspberry Pi e ambas as CPUs, foi empregada a biblioteca *Time* disponível na linguagem *Python*. É feita a aferição da diferença de tempo entre o início e o fim da execução do modelo. De modo semelhante, o valor médio de 20 execuções é assumido como o valor do tempo de inferência do modelo.

4.1.2 Medição do Consumo Energético

Para a medição do consumo energético do Movidius NCS executando os modelos selecionados são utilizados o Raspberry Pi 3 em conjunto com o sensor de potência INA219, um computador e o próprio dispositivo. O Raspberry Pi tem a função de controlar o sensor de potência, coletar os dados e armazená-los. O computador, por meio de um script escrito em *Python*, comanda o Movidius NCS. O cabo que faz a conexão entre os dois é seccionado, abrindo espaço para o sensor. A Figura 17 mostra o esquemático utilizado e descrito anteriormente.

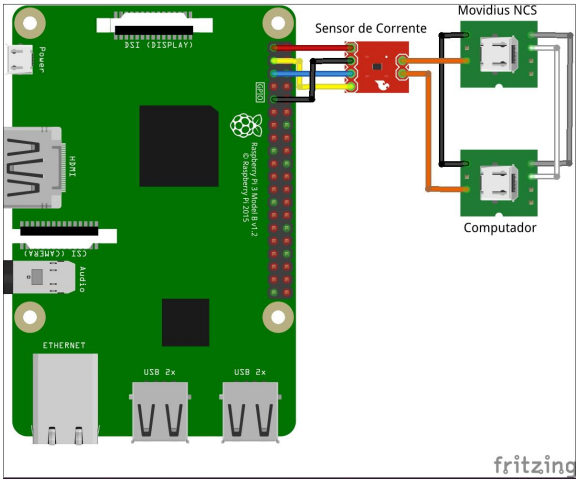

Figura 17: Esquemática de Ligação para Medição de Consumo Energético.

O sensor INA219 opera por meio de protocolo de comunicação i2c. A placa Raspberry Pi é utilizada como mestre na rede entre os dois. A frequência de clock configurada é de 1 MHz. O sensor, que utiliza um resistor *shunt* de 0.1 ohms, foi configurado para operar com correntes de até 1A e tensões de no máximo 16 volts. O conversor analógico-digital integrado foi programado para trabalhar com 9 bits de resolução, a fim de permitir uma maior taxa de aquisição dos dados.

Utilizando o sistema mostrado, cada modelo é inicializado e 10 inferências são realizadas em sequência. Os dados de corrente são coletados e armazenados dentro da placa Raspberry Pi. O gráfico resultante de um processo de medição de corrente, nesse caso para a rede GoogLeNet, é apresentado na Figura 18.

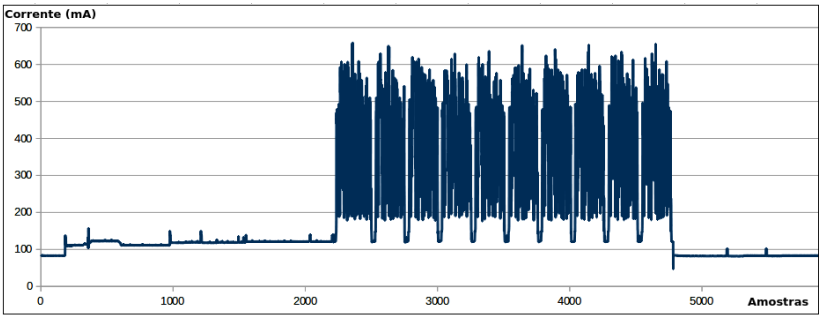

Figura 18: Gráfico de Medição de Corrente para a Rede GoogLeNet.

Inicialmente o dispositivo Movidius NCS encontra-se em estado de espera, resultando em um consumo inferior a 100 mA. Após o início da execução do *script* e da comunicação entre o computador e o dispositivo, a corrente apresenta um leve aumento. Nessa etapa anterior aos processos de inferência o modelo é transferido e inicializado no Movidius. Finalmente se inicia o procedimento de execução do modelo, que consiste no envio do dado que se deseja inferir, na inferência em si e no envio por parte do dispositivo do resultado para o computador. Como já mencionado, essa ação é repetida por 10 vezes, como pode ser verificado no gráfico. Ao final do processo, o Movidius retorna ao estado de espera.

Para a atribuição do valor médio consumido pelo modelo, optou-se utilizar exclusivamente os dados que representam o momento em que a inferência é realizada, sendo assim, eliminando do cálculo os valores em que o dispositivo se encontra em estado de espera ou mesmo durante a etapa de transferência do modelo. A partir das informações restantes

é calculada a média dos valores e é obtido o valor de máxima corrente.

Para a obtenção do valor de consumo energético de cada modelo é necessário utilizar a equação 4.1,

$$E = P * \Delta t \tag{4.1}$$

onde E é a energia consumida, medida em Joule (J); P é a potência, medida em Watts (W); e Δt é o tempo, medido em segundos (s).

Adaptando a equação para a necessidade do trabalho, cuja potência é regida pela equação 4.2,

$$P = V * I \tag{4.2}$$

sendo V a tensão elétrica, medida em Volts (V), e I a corrente elétrica, medida em Ampere (A).

Substituindo a equação 4.2 na equação 4.1, adotando como valor de tensão elétrica 5 Volts e realizando as adaptações de notação científica, tem-se como resultante a equação 4.3,

$$E = (\frac{5 * I}{1000}) * (\frac{\Delta t}{1000}) * 1000 \tag{4.3}$$

onde Δt é o tempo de inferência do modelo em questão. Para melhor apreciação dos resultados, o consumo energético é apresentado em milijoule (mJ).

4.2 PREPARAÇÃO DOS DADOS PARA TREINAMENTO E TESTE

O banco de dados utilizado para a elaboração desse trabalho é o *Plant Village*. Após a sua seleção e aquisição foi realizada a análise inicial dos dados contidos nele. Esse processo foi dividido em duas partes, uma em que somente os dados correspondentes às folhas de tomate, qualidade mais representativa do *dataset*, foram trabalhados; e uma segundo em que todas as espécies foram utilizadas.

Como o objetivo do trabalho é a discriminação entre espécimes saudáveis e não, foram analisadas inicialmente as quantidades de dados dessas duas classes. Observou-se uma discrepância bastante acentuada entre os volumes de informações, algo em torno de 10 vezes mais dados para a classe não saudável. A utilização de dados nessa situação torna os modelos treinados tendenciosos, o que resulta em respostas ruins no momento da utilização. Com isso, fez-se necessária a utilização de técnicas de *Data Augmentation*, procedimento esse que consiste na

geração de mais dados a partir dos existentes.

A biblioteca openCV, especializada em visão computacional, dispõe de diversas ferramentas que permitem a realização de ajustes e tratamento de imagens. Em virtude disso, ela foi utilizada para a geração das imagens complementares. A Figura 19 demonstra o resultado do processo de *Data Augmentation*. A imagem do canto superior esquerdo é a original. A partir dela as demais foram geradas, utilizando funções de rotação, espelhamento, corte e redimensionamento, além de dois níveis de ofuscamento.

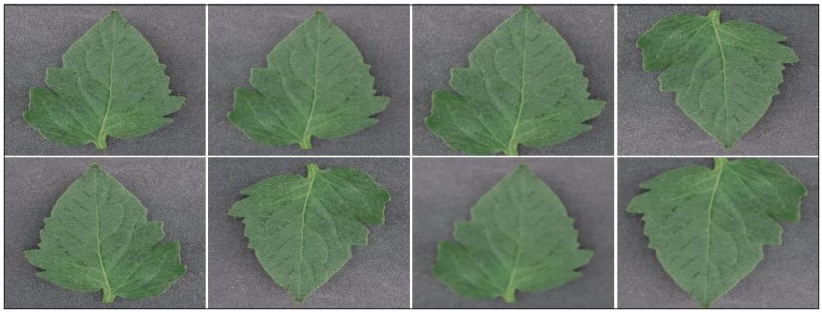

Figura 19: Resultado da Aplicação das Técnicas de *Data Augmentation*.

Todas as imagens presentes no banco de dados são apresentadas com o mesmo alinhamento espacial. Como as pertencentes à classe saudável passariam pelo procedimento de *Data Augmentation*, elas acabariam perdendo essa orientação. Para evitar que a rede aprendesse a detectar o alinhamento da imagem, em vez das possiveis doenças, optou-se por também aplicar as técnicas à parcela não saudável dos dados. Para que o desbalanceamento na quantidade de imagens não se mantivesse, somente um dos exemplares gerados durante o processo é, de forma randômica, aproveitado.

Ao final dessa etapa, para que ambas as parcelas do *dataset* dispusessem de quantidades idênticas de dados, algumas imagens foram manualmente e randomicamente deletadas. Com os dados equalizados, foi feita a divisão entre a parcela de treinamento e a parcela de teste. Optou-se pela proporção 80/20, valor esse comumente utilizado na literatura.

Nessa dissertação, a biblioteca *TensorFlow* está sendo empregada para a implementação e para o treinamento dos modelos. Além de dispor de funções que agilizam e simplificam a elaboração

das redes, a biblioteca ainda possui ferramentas que otimizam a gestão dos dados, uma etapa bastante crítica do processo de treinamento. Por isso, foi empregada a interface de programação de aplicação (API) *Dataset*, disponibilizada pelo *TensorFlow*. Essa interface dispõe de diversos métodos de entradas para os dados, porém um dos mais otimizados é por meio de arquivos *.tfrecords*, arquivos esses exclusivos da biblioteca. Fazendo o uso desse formato, os dados de treinamento são carregados de forma mais rápida e parcial para a memória, não sobrecarregando o sistema. Para isso, 4 arquivos foram gerados, sendo eles: treinamento e teste para folhas de tomate, treinamento e teste para o *dataset* completo.

4.3 METODOLOGIA DE SAÍDAS COMPLEMENTARES

Os trabalhos relacionados apresentados mostram que existe uma tendência na utilização de modelos convolucionais prontos, modelos esses que, em geral, são destinados a aplicações complexas e por isso são bastante volumosos. Visando a aplicações embarcadas, utilizar uma rede desse tipo em uma plataforma com poder computacional limitado, resulta em valores altos de consumo energético e tempo de inferência. Sistemas embarcados normalmente demandam respostas rápidas, para a tomada de ações ágeis. Pensando nisso, a utilização de modelos muito complexos não é uma boa medida.

Uma rede convolucional aplicada ao *dataset ImageNet*, que dispõe de 1000 classes, exige uma complexidade elevada para essa distinção. Para que seja possível reduzir o tamanho e a complexidade dos modelos, faz-se necessária a diminuição do número de saídas. Para isso é oportuno reavaliar o modo como os problemas são tratados. A solução aqui proposta visa à substituição da identificação de uma grande variedade de classes pela detecção de uma característica específica nas imagens. Uma vez que o objetivo se torna esse, as saídas se limitam à presença ou não do que se deseja detectar. Exemplificando a aplicação desse método: detectar a presença de uma anomalia em uma imagem, em vez de classificar qual é a anomalia; detectar a presença de um obstáculo, em vez de classificar qual seria o obstáculo.

A partir do momento em que se limita drasticamente o número de saídas, a necessidade de extrações de características extremamente complexas não se faz presente. Uma rede bem menos complexa é aplicável à solução, rede essa que gera ganhos em tempo de inferência,

consumo energético e até mesmo em tempo de treino.

Partindo para essa abordagem, é necessária inicialmente uma adaptação no banco de dados utilizado para o treinamento da rede, como apresentado na seção 4.2. Além disso, a elaboração de modelos mais compactos se faz possível e crucial, a fim de alcançar os ganhos propostos. Para permitir o desenvolvimento de redes nesse formato será realizada a análise das camadas de um modelo convolucional para entender a influência desses parâmetros nos resultados alcançados pelo modelo.

4.4 ANÁLISE E TESTE DA INFLUÊNCIA DAS DIFERENTES CAMADAS EM UM MODELO CONVOLUCIONAL

Para permitir o desenvolvimento de redes convolucionais complementares e apresentar um esclarecimento a respeito do modo que as camadas presentes em um modelo influenciam nos valores de acurácia, tempo de inferência, consumo energético e tempo de treinamento, serão realizadas análises em diversos formatos de redes. As camadas que serão avaliadas serão as convolucionais, de *batch normalization* e densa com *dropout*, em diversas variações.

Para padronizar os resultados obtidos, alguns parâmetros presentes nas redes e no processo de treinamento foram predeterminados e fixados. A Tabela 6 apresenta cada parâmetro e o valor definido para ele.

Os modelos criados para a realização dos testes seguem três padrões: convolucional, convolucional com *batch normalization* e convolucional com camada densa. As redes são construidas contendo de 1 a 4 camadas convolucionais, seguidas de camadas de *pooling*, sendo as convolucionais compostas por 4, 8, 16, 32, 64 ou 128 filtros. Já as camadas densas serão de 0 a 2, compreendendo 8, 16, 32, 256 ou 512 nós. A Figura 20 mostra o esquemático das redes implementadas. Os testes de tempo de inferência e consumo energético serão conduzidos de modo semelhante ao explicado na seção 4.1.

Para a obtenção dos valores de acurácia e tempo de treinamento, um servidor contendo o processador Intel Xeon Phi é utilizado. Após cada época de treinamento é realizada a verificação da acurácia do modelo e o valor obtido é armazenado em uma lista. Ao final das 50 épocas, o maior valor contido na lista é atribuido como acurácia da rede em questão. O tempo de treinamento é medido por meio da identificação do período entre o ínicio e o fim da execução de cada

Tabela 6: Parâmetros definidos para padronização dos resultados dos testes.

Parâmetro	**Valor**
Tamanho das imagens	256 x 256 x 3
Semente de inicialização de valores do TensorFlow	100
Função Custo	Cross-Entropy
Otimizador	Adam
Taxa de Aprendizagem	0.0005
Tamanho do batch	32
Número de épocas de treinamento	50
Tamanho dos filtros convolucionais	3x3
Stride dos filtros convolucionais	2
Tamanho do *Pooling*	2x2
Stride do *Pooling*	2
Função de Ativação	ReLU
Valor de *Dropout*	0.5
Função de Saída	Softmax

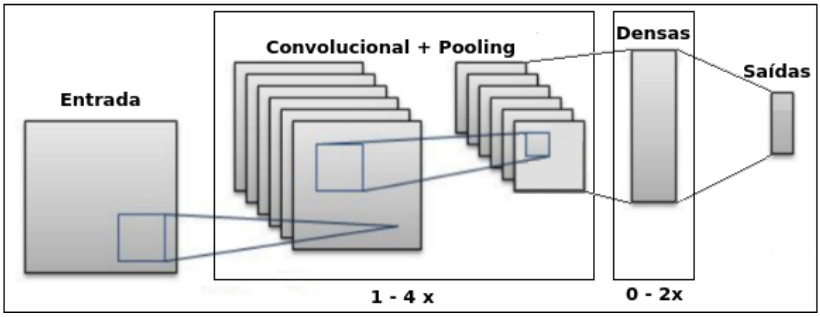

Figura 20: Esquemático dos modelos implementados para testes.

época, sendo esse valor inserido em uma lista. O valor médio de todos os termos dessa lista é atribuido ao tempo de treinamento, sendo esse

medido em segundos por época.

A sequência de testes realizados é descrita nas subseções seguintes, separadas em virtude da estrutura das redes.

4.4.1 Modelos Convolucionais

Na classe dos modelos "puramente"convolucionais, as saídas ainda são representadas por camadas densas. Será analisada a influência nas grandezas já mencionadas na utilização de 1, 2 ou 3 camadas contendo 4, 8, 16, 32, 64 ou 128 filtros cada. Para não tornar a quantidade de testes *ad infinitum*, limitou-se a utilização em uma mesma rede a camadas com número de filtros semelhantes. A Tabela 7 exemplifica parcialmente a estrutura dos testes realizados.

Tabela 7: Estrutura dos Testes Realizados com Modelos Convolucionais.

Camada Conv1	**Camada Conv2**	**Camada Conv3**	**Saída *Softmax***
4	0	0	2
4	4	0	2
4	4	4	2
8	0	0	2
8	8	0	2
8	8	8	2

4.4.2 Modelos Convolucionais com *Batch Normalization*

Na classe de modelos convolucionais com *batch normalization* o padrão dos testes seguirá o mesmo princípio que os modelos sem normalização. O *batch normalization* é uma estratégia de aplicação bastante difundida e por isso o intuito dessa análise é entender a influência de sua utilização na acurácia, tempo de inferência, consumo energético e tempo de treinamento das redes. Com isso, tem-se como objetivo realizar uma comparação direta entre os modelos

convolucionais com e sem o emprego da técnica.

4.4.3 Modelos Convolucionais com Camada Densa

A análise dos modelos convolucionais com camada densa é dividida em quatro blocos de testes.

No primeiro conjunto serão testados modelos com uma camada densa com 16 nós, em conjunto com 1 a 4 camadas convolucionais de 16 filtros. O intuito desses testes é entender a influência da adição da camada densa nas redes e, para tanto, realizar um comparativo com os modelos puramente convolucionais.

O segundo bloco será composto por redes contendo duas camadas convolucionais de 16 filtros cada, seguidas de uma camada densa contendo 8, 16, 32, 256 ou 512 nós. Nesses testes espera-se entender como a adição de nós na camada influencia nas grandezas analisadas.

O terceiro conjunto de testes engloba redes contendo uma camada densa de 16 nós. Essa camada é precedida por uma ou duas camadas convolucionais de 4, 8, 16, 32, 64 ou 128 filtros. Por meio desses testes, espera-se verificar a influência da adição de uma camada convolucional extra ao modelo, além de realizar comparativos a respeito da adição da camada densa.

No último bloco, serão avaliadas redes que possuem duas camadas densas, ambas com 16 nós. Na parcela convolucional serão empregadas duas camadas contendo 4, 8, 16, 32, 64 ou 128 filtros. Com esses testes, espera-se verificar se a adição de mais um camada densa afeta positivamente o desempenho dos modelos analisados.

4.5 DETALHAMENTO DOS RESULTADOS OBTIDOS

A partir dos resultados dos testes descritos na seção 4.4, uma série de análises gráficas é realizada. O intuito dessas análises é verificar como se dá a influência das diferentes configurações de camadas no desempenho dos modelos. A partir da apresentação desses dados, espera-se permitir a simples implementação de modelos convolucionais, como descritos na seção 4.3, visando à detecção de objetos ou anomalias, atendendo a critérios de acurácia, consumo energético e tempo de inferência.

Os resultados a serem verificados com relação à acurácia dos

modelos depende diretamente da complexidade do *dataset* utilizado, por isso os valores que serão apresentados podem indicar a tendência de um resultado, mas não o seu valor definitivo. Além disso, ajustes nos hiperparâmetros de treinamento podem resultar em valores mais ou menos expressivos. A mesma coisa acontece com relação ao tempo de treinamento, que possui forte influência do tamanho das imagens empregadas e da dimensão do banco de dados. Por outro lado, os dados de tempo de inferência e consumo energético dependem exclusivamente do modelo e do hardware empregado. Por isso, na utilização do chip Myriad 2, os resultados serão semelhantes aos a serem apresentados.

4.6 METODOLOGIA DE INFERÊNCIA EM DOIS ESTÁGIOS

A seção 4.3 descreve a proposta de implementação de redes utilizando somente duas saídas, com o intuito de detectar certo objeto ou anomalia. Quando se faz obrigatoriamente necessária a discriminação de diversas classes, uma rede com mais saídas deve ser utilizada.

Antes de realizar a inferência de um modelo, não há a certeza da presença de uma anomalia na imagem que será classificada. A execução em um sistema embarcado de um modelo complexo, que exige um tempo relativamente alto de execução, além de consumir uma energia limitada, para que o resultado seja a não detecção de um anomalia ou de um objeto, é um desperdicio de recursos. Com o intuito de reduzir esse desperdício, a metodologia de inferência em dois estágios foi elaborada.

A metodologia de dois estágios é composta por dois modelos convolucionais, sendo o primeiro um exemplar de saídas complementares, que realiza a detecção de certo objeto ou anomalia; e o segundo, um modelo tradicional, responsável pela classificação da imagem. A Figura 21 apresenta um esquemático que apresenta a estrutura do sistema.

Inicialmente, uma imagem é recebida para que se faça a detecção/classificação de um objeto ou anomalia. A imagem é repassada para a rede complementar que realizará a sua inferência. As saídas desse modelo irão indicar a detecção ou não do que se procura. Caso nada seja encontrado, o sistema volta ao seu estado inicial e recebe/espera uma nova imagem. Por outro lado, se algo for detectado pela primeira rede, a segunda irá receber o conteúdo e efetuará uma nova inferência, apresentando em suas saídas o resultado

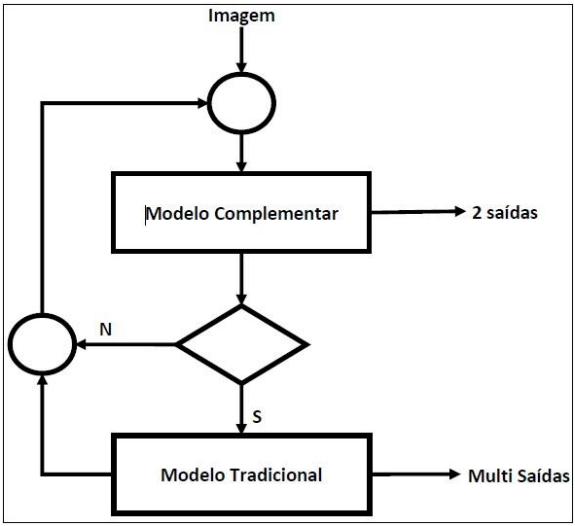

Figura 21: Esquemático da Implementação de Metodologia de Dois Estágios.

da classificação realizada. Ao final, o sistema retorna ao início. Uma vantagem clara nessa abordagem é que caso a primeira rede não realize uma detecção, a segunda não será requisitada, gerando uma economia de recursos que de outro modo seriam desperdiçados.

4.7 ANÁLISE DA APLICAÇÃO DA METODOLOGIA DE INFERÊNCIA EM DOIS ESTÁGIOS

A aplicação da metodologia de inferência em dois estágios, como descrito na seção 4.6, visa a uma redução no consumo energético do sistema aplicado, além da redução do tempo de inferência global. É notável que a execução pontual sucessiva de ambas as redes resulta em um tempo e consumo superiores ao da rede classificadora exclusivamente, porém, à medida que o sistema tem sua execução prolongada e certa quantidade de detecções negativas estão presentes, a vantagem na sua utilização se torna evidente. Essa estratégia apresenta um melhor desempenho quanto maior for o número de detecções negativas da aplicação. Por outro lado, caso o número de detecções positivas seja alto, o sistema perde eficiência,

podendo até se tornar desfavorável.

A equação 4.4, que leva em consideração os tempos de inferência da rede detectora (t1) e da rede classificadora (t2), permite a verificação do ponto percentual de detecções positivas máximo (Pmax) que torna a metodologia apresentada oportuna. A mesma fórmula pode ser aplicada na constatação a respeito do consumo energético, substituindo os tempos pelos consumos de cada rede.

$$Pmax < \frac{t2 * 100 - t1 * 100}{t2} \tag{4.4}$$

A partir da equação descrita e da verificação dos tempos de inferência das redes classificadoras analisadas e dos modelos complementares testados, serão apresentados resultados numéricos perante a utilização da metodologia proposta.

Já para calcular o valor do ganho percentual alcançado pelo conjunto utilizado em função da taxa de positivos e dos tempos ou consumos de cada rede empregada, a equação 5.3 pode ser empregada.

$$Ganho = \frac{100 * (t2 - (t1 + t2 * P)}{t2}, \tag{4.5}$$

onde P é o percentual inferências positivas.

5 RESULTADOS

O presente capitulo apresenta os resultados alcançados no desenvolvimento do trabalho aqui exposto. Essa apresentação será dividida em três seções, iniciando pela exposição dos resultados dos testes de tempo e consumo energético das redes genéricas selecionadas, como detalhado na seção 4.1. Na sequência, serão apresentados, por meio de gráficos e tabelas, complementados por explanações pertinentes, os resultados dos testes de influência das camadas convolucionais na acurácia, tempo de inferência, consumo e tempo de treinamento dos modelos utilizando saídas complementares, testes esses explicados na seção 4.4. Por último, serão apresentados os resultados da aplicação da metodologia de dois níveis de inferência, assim como exposto na seção 4.7.

5.1 TEMPO DE INFERÊNCIA E CONSUMO ENERGÉTICO DE REDES TRADICIONAIS

Inicialmente serão apresentados os dados referentes aos testes envolvendo as plataformas Raspberry Pi, Movidius NCS, Intel Core I7 e Intel Xeon Phi executando os modelos Inception_V1, Inception_v4 e MobileNet.

Ao realizar a inferência dos modelos, verificou-se uma discrepância de tempo entre a primeira execução das redes, sua inicialização, e as demais. Tal situação só não está presente no Movidius NCS, por isso, na Figura 22 ele foi suprimido.

Desconsiderando a inicialização das redes, o tempo de inferência para cada modelo foi calculado e pode ser verificado na Figura 23. As plataformas que apresentam maior discrepância entre os tempos de inicialização e de inferência são o processador Intel Xeon Phi e o Raspberry Pi, tendo respectivamente como piores resultados variações de 27 vezes e 8 vezes. Sendo o Raspberry Pi a plataforma menos robusta do experimento, era esperado o seu desempenho inferior. No caso do Intel Xeon Phi, em virtude da grande quantidade de núcleos de processamento presentes no dispositivo, o processo de alocação de tarefas pode demandar mais tempo.

Com relação aos tempos de inferência alcançados por plataforma é possível notar um desempenho bastante positivo do Movidius NCS, alcançando valores até melhores que o de um

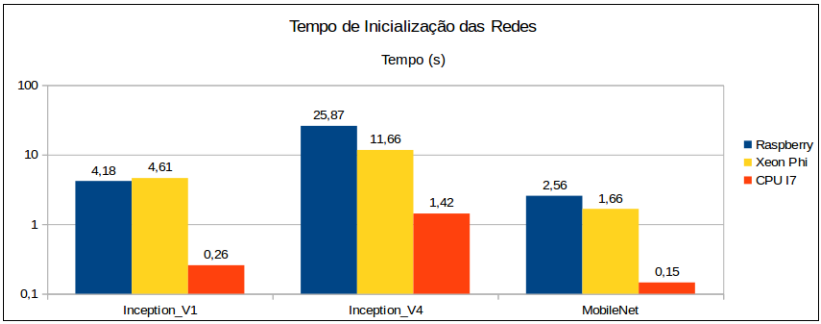

Figura 22: Tempo de Inicialização dos Modelos em Plataformas Diversas.

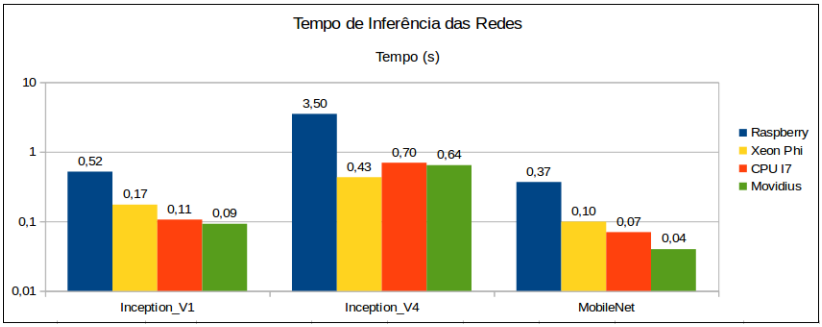

Figura 23: Tempo de Inferência dos Modelos em Plataformas Diversas.

processador Core I7, dispositivo esse com *clock* e potência bastante superior. Mais uma vez o Raspberry Pi mostrou resultados bem limitados. Por outro lado o Intel Xeon Phi teve seu resultado mais expressivo executando a rede Inception_V4, o que pode ser explicado pela sua alta capacidade de paralelização trabalhando em conjunto com um modelo que se beneficia de tal característica. As outras redes, por não apresentarem um perfil tão paralelizável, foram limitadas pelo baixo *clock* da CPU.

Dando sequência, utilizando as demais redes em conjunto com o Movidius, foram realizados os testes, conforme descritos na seção 4.1. A Tabela 8 apresenta os resultados dos modelos selecionados executados no dispositivo Movidius NCS. Foram destacados em verde os melhores resultados das medições; enquanto que, em vermelho, são mostrados os piores. Os valores apresentados são medidos respectivamente em

milisegundos, miliamperes, miliamperes e milijoules.

Tabela 8: Resultados de Tempo, Corrente e Consumo para os Modelos Analisados.

Modelos	**Tempo de Inferência (ms)**	**Corrente Média de Inferência (mA)**	**Corrente de Pico de Inferência (mA)**	**Consumo Energético (mJ)**
Inception_V1	92.1	358.7	636	165.19
Inception_V3	328.76	377.24	639	620.11
Inception_V4	**641.95**	402.48	641	**1291.86**
MobileNet	**39.81**	**406.99**	**753**	**81.01**
AlexNet	91.41	**270.33**	**621**	123.55
GoogLeNet	96.92	366.16	643	177.43
SqueezeNet	49.01	375.37	656	91.98
TinyYolo	114.34	346.7	665	198.2

As redes da classe *Inception*, bem como a rede *GoogLeNet*, por possuírem arquiteturas semelhantes, apresentaram tempos diretamente proporcionais aos seus tamanhos. A rede *AlexNet*, menor, porém menos otimizada que as demais, obteve tempos de inferência próximos aos da *GoogLeNet* e *Inception_ V1*. Por outro lado, as redes *MobileNet* e *SqueezeNet*, em virtude de suas arquiteturas diferenciadas, apresentaram os menores tempos de inferência, sendo a primeira 16 vezes mais rápida do que a rede *Inception_ V4*.

Pela análise da corrente média é possível constatar que, com exceção das redes *Inception_ V4, MobileNet* e *AlexNet*, as redes apresentam uma demanda média relativamente próxima. Isso pode ser explicado pelo nível de complexidade por nível dos modelos serem parecidos. Por outro lado, a rede *Inception_ V4*, mesmo empregando topologia comparável às redes anteriores, apresenta uma densidade bem superior, exigindo assim mais corrente para a sua execução. No caso da *MobileNet*, sua arquitetura diferenciada, como apresentado no capítulo 3.0.1, explica o fato do valor medido ser superior. A *AlexNet*, por ser o modelo menos complexo, acabou por apresentar o menor valor média das redes analisadas.

Com relação à corrente de pico, a rede *AlexNet*, pela sua complexidade inferior, apresenta o menor valor. Um detalhe que deve ser levado em conta é com relação à rede *MobileNet*. Em virtude de sua topologia diferenciada, como apresentado na subseção 3.0.1, a rede apresenta corrente média de inferência e de pico superior às demais. A Figura 24 mostra o comportamento da corrente durante a execução do modelo e pode ser constatado que durante vários momentos o valor se aproximou ou superou os 700 miliamperes. Na imagem apresentada, estão presentes duas inferências da rede *MobileNet*.

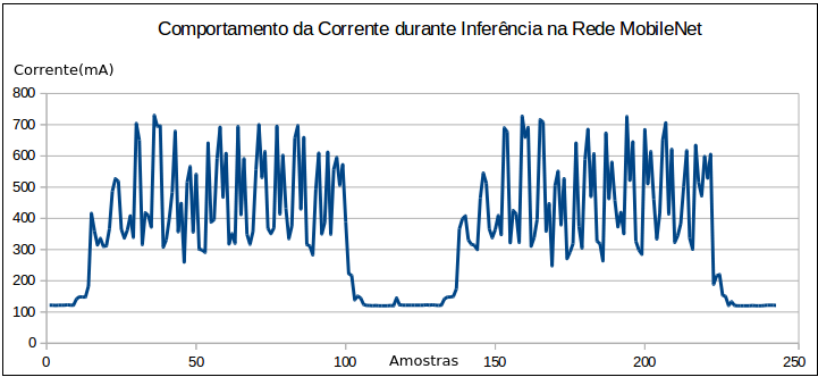

Figura 24: Comportamento da Corrente durante Inferência na Rede MobileNet.

O consumo energético da *MobileNet* é o menor entre os modelos analisados, em virtude do seu tempo de inferência bastante reduzido, porém, para a confecção de modelos cujo consumo seja prioridade, o emprego desse tipo de arquitetura deve ser feito com cuidado.

5.2 INFLUÊNCIA DAS CAMADAS NOS MODELOS CONVOLUCIONAIS

Nessa seção serão apresentados os resultados obtidos com os testes descritos na seção 4.4 e, de modo semelhante, a descrição dos dados se dará seguindo a divisão de modelos. Nessa seção, para ilustrar os resultados, serão utilizados gráficos em diferentes formatos.

5.2.1 Resultado dos Modelos Convolucionais

Inicialmente, os modelos puramente convolucionais foram testados. Foram criadas redes contendo de 1 a 3 camadas com 4 a 128 filtros por camada. A Figura 25 mostra os dados referentes aos modelos testados. O apêndice A apresenta os valores para os modelos testados.

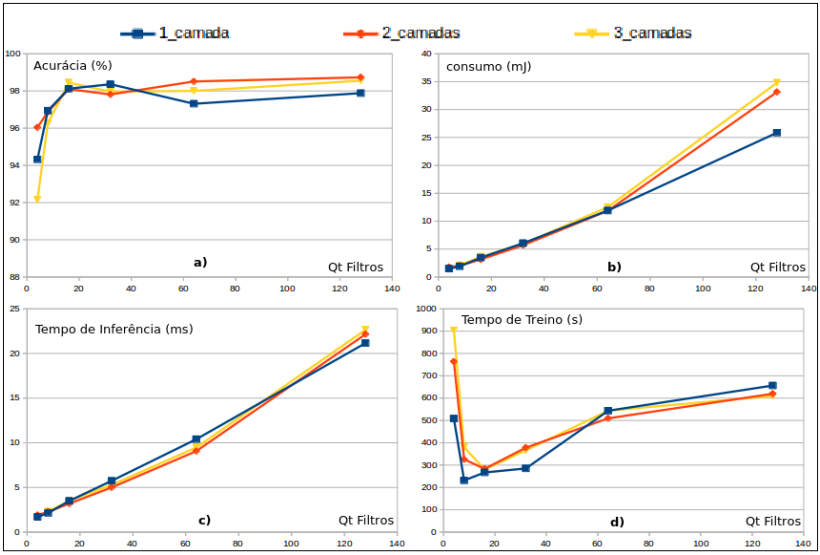

Figura 25: Comparativo dos Resultados dos Modelos Convolucionais de 1, 2 e 3 Camadas com Quantidade de Filtros por Camada Variável, com a) acurácia, b) consumo, c) tempo de inferência e d) tempo de treino.

Nos testes realizados, observa-se uma maior acurácia nas redes com duas camadas utilizando 64 ou 128 filtros. As redes que utilizaram camadas de 4 filtros, como esperado, obtiveram resultados bem abaixo das demais. O mesmo acontecendo com as de 8 filtros. Por outro lado, utilizando 16 filtros observa-se um resultado expressivo, alcançando acurácias, em alguns casos, superiores aos modelos com quantidade superior. Isso pode ser explicado por uma maior capacidade de generalizar as características presentes nessa aplicação específica.

Como comentado, os modelos de duas camadas, em alguns

casos, apresentaram resultados levemente superiores aos de três camadas. Uma explicação para essa situação se deve ao fato da utilização de três camadas de *pooling* por essas redes. Esse procedimento, além de reduzir a complexidade, o volume dos dados e garantir uma maior generalização, pode por outro lado resultar numa perda de informações, o que comprometeria a etapa de classificação.

Analisando o tempo de treinamento das redes, tempo esse medido em segundos por época, pode-se observar uma discrepância bastante acentuada nos valores das redes com camadas de 4 filtros, em especial com a utilização de mais de uma camada. Esse fenômeno foi observado quando executados os modelos no servidor com o Intel Xeon Phi, padrão utilizado para esse teste. A execução desses mesmos modelos em outras plataformas não apresentou esse problema. Observa-se então uma dificuldade por parte do equipamento em lidar com redes que utilizem camadas convolucionais com baixo número de filtros, visto que as redes com 2 e 3 camadas utilizando 8 filtros também apresentaram resultados discrepantes.

A partir de 16 filtros por camada, o tempo de treinamento apresenta um crescimento constante até a quantidade de 64 filtros. Para os 128 filtros observa-se uma atenuação na taxa de crescimento do tempo de treino, o que pode ser explicado pela capacidade de paralelismo do Intel Xeon Phi, que como mostrado na seção 5.1, entrega resultados mais satisfatórios, quando submetido a modelos com alto grau de paralelismo.

A Figura 25 b) apresenta os resultados para o consumo energético dos modelos convolucionais testados. A adição de uma camada aos modelos não ocasiona um aumento significativo nos consumos energéticos, em especial nas redes de 2 e 3 camadas, que resultaram em valores bastante próximos. A excessão ocorre nos modelos de 2 e 3 camadas contendo 128 filtros, que apresentou aumento considerável de consumo. Nesse caso, o aumento da complexidade das camadas convolucionais superou a redução de complexidade na execução da camada de saída. Entretanto, nota-se que o maior responsável pelo aumento de consumo energético é a adição de filtros as camadas.

Um detalhe a ser comentado é com relação à rede de 3 camadas e 4 filtros, que ocasionou um problema durante a execução no dispositivo Movidius NCS. Em virtude disso seu valor no gráfico se encontra zerado.

Com relação ao tempo de inferência dos modelos, a Figura 25 c) apresenta os resultados. Nesse caso, adição de camadas não parece ter uma influência significativa no aumento da latência do modelo. Verifica-

se que a variação na quantidade de filtros resulta em uma mudança praticamente linear nos valores de tempo.

Novamente, a rede de três camadas e quatro filtros teve o seu valor zerado em função do problema durante a execução.

Tendo como base os dados apresentados, foram gerados gráficos em rede a fim de apresentar uma visão mais generalista das grandezas analisadas. Para tanto, fizeram-se necessários o ajuste e o deslocamento dos valores. A Tabela 9 mostra as operações realizadas com cada grandeza.

Tabela 9: Operações Realizadas com as Grandezas.

Grandeza	**Fator Divisivo**	**Deslocamento**
Acurácia	10	-9
Tempo de Treino	200	+1
Consumo Energético	20	+1
Tempo de Inferência	15	+2

Com base nos valores apresentados na Tabela 9, os gráficos das figuras 26, 27 e 28 foram gerados mostrando o resultado da aplicação das operações nas grandezas analisadas para os modelos com uma camada convolucional.

Na Figura 26 é possível verificar que as redes de 4, 64 e 128 filtros se destacam pelo seu tempo de treinamento superior. A rede de 128 filtros ainda apresenta valores altos de consumo e tempo de inferência. Por outro lado, o modelo de 4 filtros exibe uma acurácia bem inferior aos demais. Nota-se um maior equilibrio entre as grandezas analisadas nas redes 8, 16 e 32.

Na Figura 27, que mostra os resultados para o modelo de duas camadas, fica evidente uma discrepância no tempo de treinamento para o modelo de 4 filtros, sendo esse bem superior aos demais. Além disso, a acurácia desse modelo se mostrou bem inferior. A rede de 128 filtros exibe valores superiores de consumo energético e tempo de inferência. Os modelos de 8, 16, 32 e 64 filtros apresentam resultados equilibrados, com uma leve vantagem para a rede 16.

Já no caso do modelo de três camadas, com os resultados apresentados na Figura 28, nota-se mais um vez o problema com o treinamento do modelo de 4 filtros, cujo valor é substancialmente superior aos demais. Outro problema desse modelo é com relação à

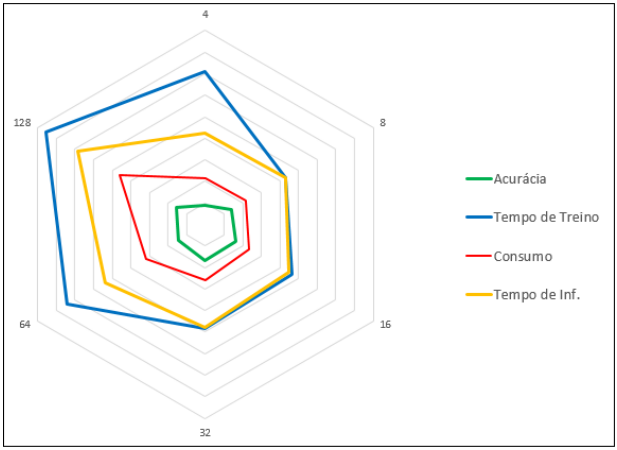

Figura 26: Análise das Grandezas Resultantes para Modelos Convolucionais de uma Camada.

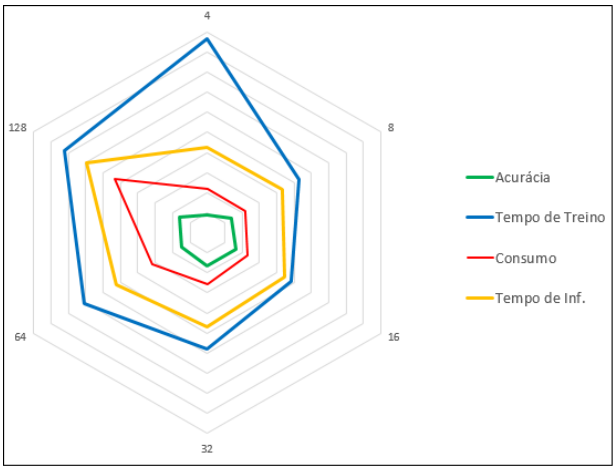

Figura 27: Análise das Grandezas Resultantes para Modelos Convolucionais de duas Camadas.

acurácia, bem abaixo, se comparado as outras redes. Mais uma vez o modelo de 128 filtros se destaca pelo alto consumo energético e elevado tempo de inferência. A rede 8, para essa situação, apresenta acurácia modesta e tempo de treinamento considerável. O modelo de

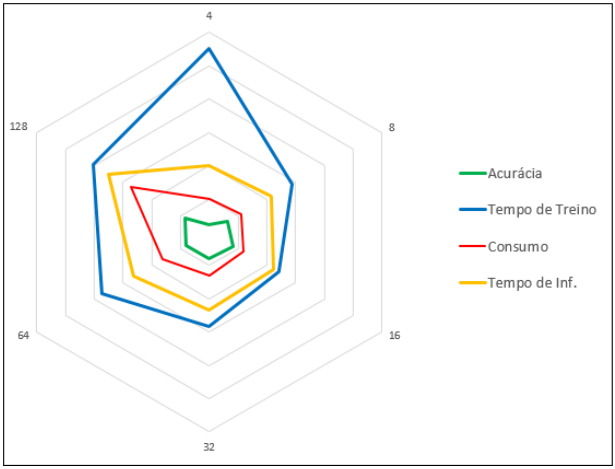

Figura 28: Análise das Grandezas Resultantes para Modelos Convolucionais de três Camadas.

16 filtros contém os resultados mais balanceados.

5.2.2 Resultados dos Modelos Convolucionais com *Batch Normalization*

Após os testes com a redes puramente convolucionais, foram realizadas as análises com os modelos que dispõem de *Batch Normalization*. A Figura 29 apresenta os resultados dos modelos. O apêndice B apresenta os valores para os modelos testados.

Com o uso da normalização os modelos de três camadas com 64 e 128 filtros obtiveram os melhores resultados de acurácia. Com exceção da rede com 32 filtros e três camadas, observa-se um resultado mais homogêneo quando comparado aos modelos sem normalização.

Dando sequência, a Figura 29 d) mostra dados para o tempo de treinamento dos modelos com normalização. Pode-se observar mais uma vez a discrepância bastante acentuada dos tempos das redes com 4 filtros. Além disso, novamente as redes de duas e três camadas com 8 filtros também demonstram um desempenho ruim de tempo de treinamento, quando comparados a redes com mais filtros.

Com relação aos demais resultados, pode-se notar um comportamento um pouco mais linear na evolução dos tempos em

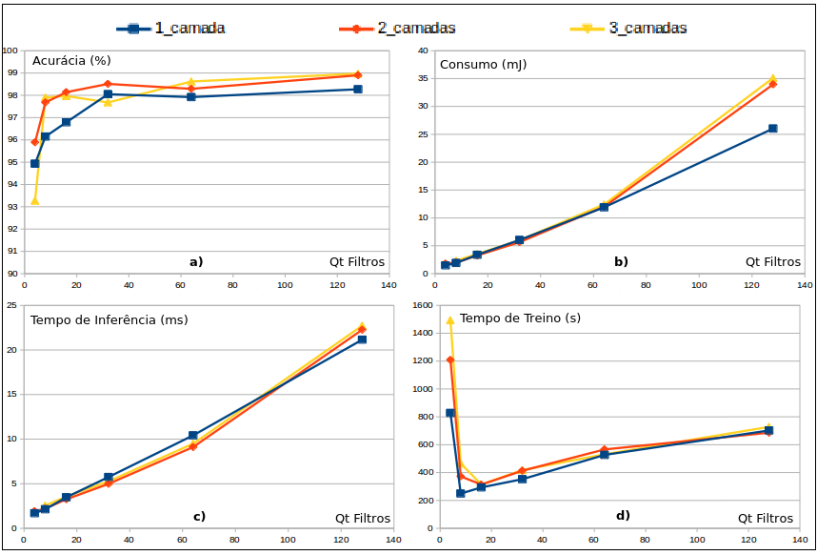

Figura 29: Comparativo dos Resultados dos Modelos com Normalização de 1, 2 e 3 Camadas com Quantidade de Filtros por Camada Variável, com a) acurácia, b) consumo, c) tempo de inferência e d) tempo de treino.

função do número de filtros.

A Figura 29 b) apresenta os valores dos consumos energéticos dos modelos. Novamente os resultados não apresentam grandes variações com relação à adição de camadas, além de mostrarem um comportamente praticamente linear com relação ao aumento da quantidade de filtros. Os únicos modelos que fogem a regra são os de 2 e 3 camadas com 128 filtros, que apresentaram um aumento no consumo em virtude do aumento na complexidade das camadas convolucionais das redes.

A respeito do tempo de inferência das redes, o gráfico da Figura 29 c) mostra os resultados obtidos, sendo que, mais um vez, a variação em função da adição de camadas é muito pequena.

O problema com relação ao modelo de três camadas e 4 filtros também se encontra presente nesse caso. Por isso o seu valor foi zerado para os gráficos de consumo e tempo de inferência.

Os três gráficos seguintes, referentes às figuras 30, 31 e 32, mostram a análise dos modelos convolucionais com *Batch*

Normalization englobando as quatro grandezas estudadas separados pela quantidade de camadas convolucionais. Foram utilizados os valores especificados na Tabela 9 para o ajuste dos números.

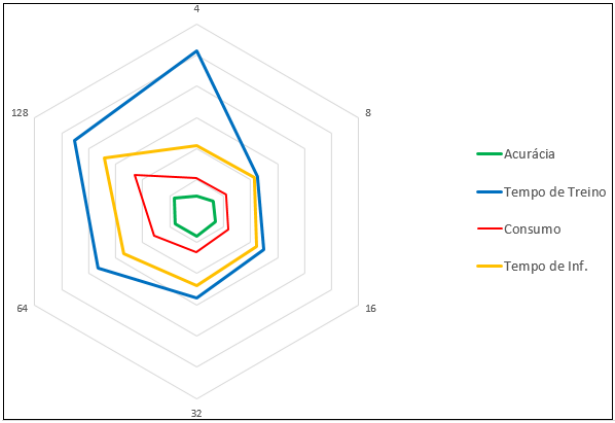

Figura 30: Análise das Grandezas Resultantes para Modelos com Normalização de uma Camada.

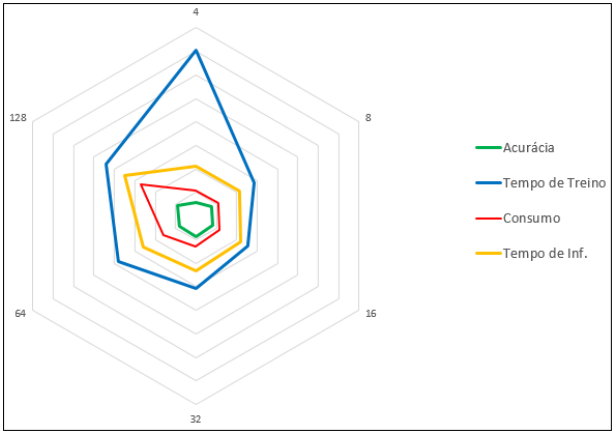

Figura 31: Análise das Grandezas Resultantes para Modelos com Normalização de duas Camadas.

A Figura 30 apresenta os resultados para o modelo de uma camada. Verifica-se um valor bastante discrepante de tempo de

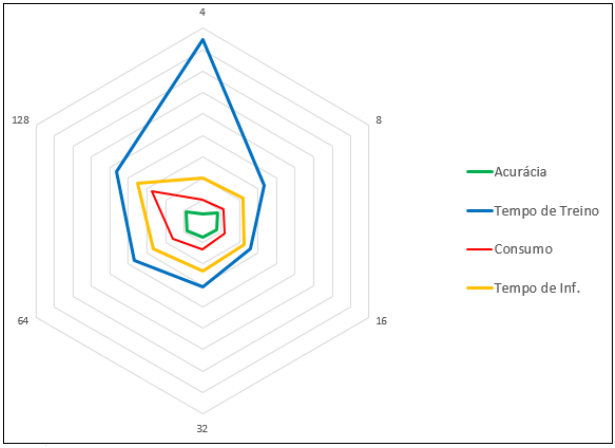

Figura 32: Análise das Grandezas Resultantes para Modelos com Normalização de três Camadas.

treinamento para a rede de 4 filtros. Com relação à acurácia, os modelos de 4, 8 e 16 filtros apresentam valores reduzidos. A rede 128 resultou em valores elevados para consumo energético, tempo de inferência e tempo de treino. Verifica-se que o modelo que se mostrou mais equilibrado é o de 32 filtros.

Na Figura 31 são apresentados os resultados para o modelo de duas camadas. Novamente a rede de 4 filtros apresentou valor bastante elevado para tempo de treinamento e valor reduzido em acurácia. O modelo de 16 filtros dispõe dos valores mais equilibrados. As redes com maior número de filtros geram resultados de tempo de treino, consumo e tempo de inferência proporcionalmente maior.

Finalmente, a Figura 32 mostra os dados para três camadas convolucionais com *Batch Normalization*. Novamente, para o modelo de 4 filtros obteve-se um elevado valor para o tempo de treino e um valor reduzido de acurácia. Nos demais modelos os resultados se assemelham bastante aos alcançados pelas redes de duas camadas, tendo uma maior equilibrio no modelo de 16 filtros e um aumento gradual nas redes de maior valor.

5.2.3 Comparativo de Modelos Com e Sem *Batch Normalization*

Tendo analisado os resultados das redes puramente convolucionais e convolucionais com *Batch Normalization*, cabe a realização de um comparativo entre o desempenho de ambas frente às grandezas analisadas.

A Figura 33 mostra o resultado da comparação entre redes de 3 camadas convolucionais com e sem a presença do *Batch Normalization*. Com exceção das redes contendo 16 e 32 filtros, os modelos contendo a normalização apresentaram uma leve melhora na acurácia, inferior a 1%.

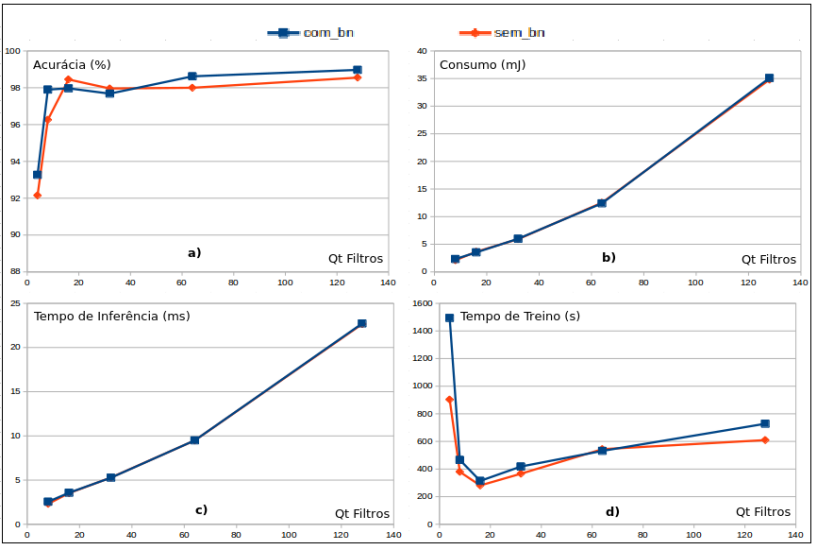

Figura 33: Comparativo dos Resultados entre os Modelos Convolucionais com e sem Normalização, com a) acurácia, b) consumo, c) tempo de inferência e d) tempo de treino.

Verificando a influência da utilização da normalização no tempo de treinamento, apresentada na Figura 33 d), pode-se ver uma piora bastante considerável no caso da utilização de 4 filtros convolucionais. Nos demais casos, com exceção da utilização de 64 filtros, os tempos com a normalização se mostraram um pouco superiores ao sem utilizar a técnica. Lembrando que os valores apresentados fazem referência à

execução de uma única época de treinamento e que um treinamento completo pode exigir dezenas ou centenas de épocas.

Com relação aos dados de consumo e tempo de inferência, mostrados nas figuras 33 b) e 33 c), pode-se notar que não houve variações aparentes nos resultados. Ambos os testes foram executados utilizando o dispositivo Intel Movidius NCS, um equipamento dedicado e otimizado para esse tipo de aplicação. Em virtude disso, fica claro que o dispositivo, que utiliza o chip Myriad 2, dispõe de soluções que executam modelos contendo *Batch Normalization* sem nenhum tipo de ônus aos resultados de consumo e tempo de inferência.

5.2.4 Resultados dos Modelos Convolucionais com Camada Densa

Após os testes apresentados anteriormente, foi dada a sequência ao trabalho, analisando o desempenho de redes utilizando camadas densas agregadas a elas. O apêndice C apresenta os valores para os modelos testados.

5.2.4.1 Camadas Convolucionais de 16 filtros e Camada Densa Variável

A primeira situação analisada é o de redes de duas camadas convolucionais de 16 filtros cada e uma camada densa com quantidade de nós variável (8, 16, 32, 256, 512). O objetivo desse teste é verificar, a partir de um modelo convolucional fixo, como ocorre a influência da adição de uma camada densa com quantidades diferentes de nós.

A Figura 34 a) apresenta o resultado da acurácia dos modelos testados. É interessante notar o resultado positivo da utilização de 16 nós na camada densa, melhor do que os modelos com 32 e 256 nós. Isso mostra que esse número gerou em uma boa capacidade de generalização no modelo. A rede com 512 alcançou o melhor resultado, generalizando de maneira mais ampla que os demais.

A Figura 34 d) mostra o tempo de treinamento das redes testadas. A variação entre os tempos medidos não apresentou grande variação, sendo a diferença máxima de menos de 6 segundos. Como era de se esperar, o modelo com 512 nós apresentou o maior tempo de treinamento. Por outro lado, a rede de 16 nós superou o modelo de 8, sendo a com o menor tempo por época.

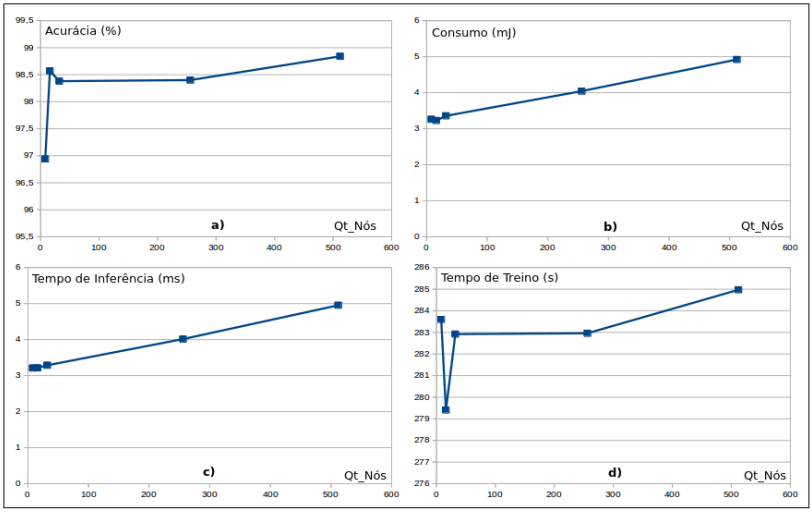

Figura 34: Valores Referentes aos Modelos com duas Camadas Convolucionais de 16 filtros e uma Camada Densa Variável, com a) acurácia, b) consumo, c) tempo de inferência e d) tempo de treino.

Na Figura 34 b) pode-se verificar o consumo energético dos modelos em questão. Verifica-se um crescimento linear no consumo em virtude do aumento da quantidade de nós nos modelos. Numericamente a variação entre as redes de 8 e 512 nós é de pouco mais de 1 mJ por inferência, porém, esse valor representa uma aumento de pouco mais de 55% no consumo.

Com relação ao tempo de inferência, apresentado na Figura 34 c), pode-se notar novamente uma tendência linear no aumento do consumo em virtude do aumento da quantidade de nós. Nesse caso, entre as redes de 8 e 512 nós, a variação foi de 54%.

A rede contendo 512 nós obteve um bom resultado no quesito acurácia, porém ao custo de maior tempo de inferência e consumo energético. Optou-se por explorar um pouco mais essa situação, analisando com maior detalhe a utilização de camadas densas mais robustas. Durante a execução dos testes, verificou-se um problema com a rede contendo uma camada convolucional de 128 filtros e uma camada densa de 512 nós. Em virtude da gigantesca quantidade de conexões entre os estágios, não foi possível executar o modelo por falta de recursos computacionais. Por isso, foram adotadas as redes contendo 256 nós para o prosseguimento das atividades.

5.2.4.2 Modelos de 1 a 4 Camadas Convolucionais Variáveis e Camada Densa de 256 nós

Foram testados modelos contendo de uma a quatro camadas convolucionais de quantidades variáveis e uma camada densa de 256 nós. Foi realizada a extração dos dados de consumo energético e tempo de inferência das redes, cujos dados são mostrados respectivamente nas figuras 35 e 36. O objetivo desse teste é verificar os resultados para diferentes quantidades de estágios convolucionais em modelos contendo uma camada densa robusta.

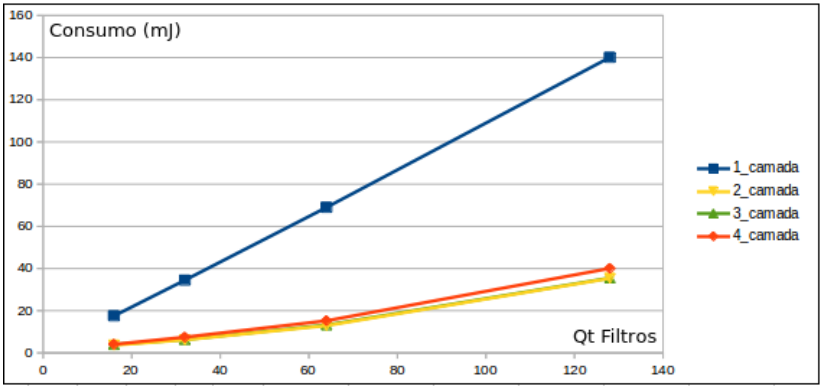

Figura 35: Consumo Energético de Modelo com Camadas Convolucionais Variáveis e uma Camada Densa de 256 nós.

Nota-se claramente um resultado bastante prejudicado para o modelo contendo uma única camada convolucional. Com a utilização de um camada densa numerosa, a quantidade de conexões e pesos existentes entre esse estágio e o convolucional é muito grande, exigindo um esforço computacional bastante superior. Isso pode ser verificado na tabela 10, onde o tempo de execução de cada camada é apresentado separadamente. A rede em questão é a que contém 128 filtros.

Nota-se claramente que o problema do alto tempo alcançado nesse modelo se encontra na camada densa. À medida que mais camadas convolucionais foram agregadas, mais operações de *pooling* foram executas, reduzindo a complexidade dos dados e, consequentemente, a quantidade de cálculos. Com a adição de um único conjunto convolucional (camada convolucional + *pooling*)

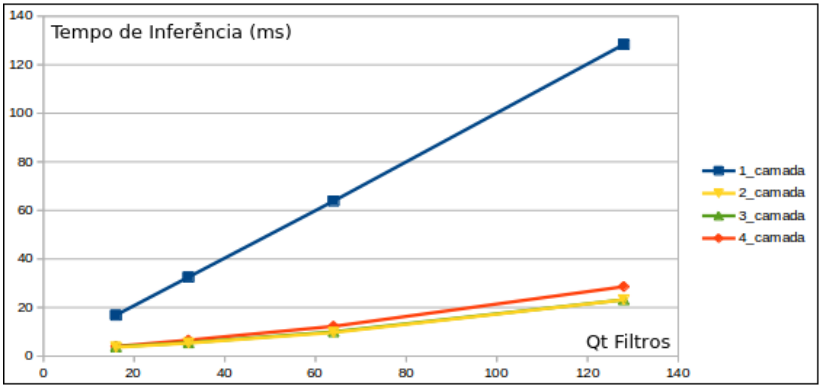

Figura 36: Tempo de Inferência de Modelo com Camadas Convolucionais Variáveis e uma Camada Densa de 256 nós.

Tabela 10: Discriminação dos Tempos (ms) de Execução por Camada.

Conv1	**Pool1**	**Dense1**	**Bias**	**Softmax**	**Total**
8,78	4,16	115,39	0,033	0,042	128,41

nota-se uma redução bastante significativa no tempo total do modelo. Os resultados camada a camada para essa rede são apresentados na Tabela 11, novamente considerando o modelo de 128 filtros.

Tabela 11: Discriminação dos Tempos (ms) de Execução por Camada em Modelo de Duas Camadas Convolucionais.

Conv1	**Pool1**	**Conv2**	**Pool2**	**Dense1**	**Bias**	**Softmax**	**Total**
8,81	4,15	8,281	0,299	6,967	0,033	0,047	28,587

Com a adição de um estágio convolucional nota-se redução de mais de 18 vezes no tempo de execução da camada densa, além de diminuição de quase 4,5 vezes no tempo total.

Voltando as figuras 35 e 36, verifica-se que o ponto de menor esforço computacional encontra-se na utilização de três níveis. A partir da utilização da quarta camada, a quantidade de cálculos dos estágios convolucionais já se torna predominante, enquanto que a redução no

tempo de execução da camada densa não apresenta uma redução tão acentuada quanto nos casos anteriores.

5.2.4.3 Camadas Convolucionais Variáveis e Camada Densa

Dando continuidade ao estudo, em virtude do desempenho positivo apresentado pelas redes contendo 16 nós na camada densa, os demais testes realizados utilizarão esse valor como quantidade padrão, procurando compreender a influência das demais variáveis nos resultados.

Uma nova classe de redes foi elaborada, considerando modelos de uma ou duas camadas convolucionais de quantidade variável de filtros em conjunto com uma camada densa contendo 16 nós. Os resultados de acurácia são mostrados na Figura 37 a). O objetivo desse teste é verificar a influência da quantidades de filtros nos estágios convolucionais atuando em conjunto com a camada densa.

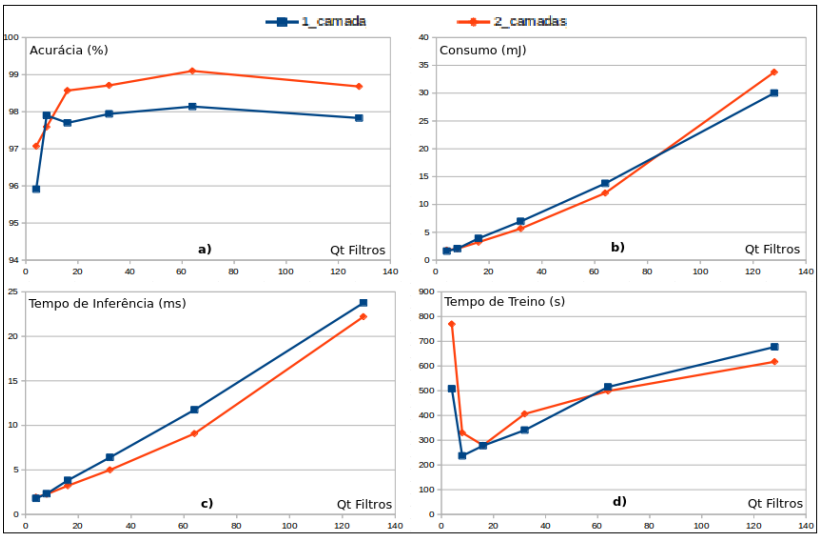

Figura 37: Valores Referentes aos Modelos Convolucionais com 1 ou 2 Camadas de Filtros Variáveis e 1 Camada Densa de 16 nós, com a) acurácia, b) consumo, c) tempo de inferência e d) tempo de treino.

A rede contendo 64 filtros obteve o maior valor de acurácia entre todos os testes realizados, com 99,1%. Nota-se mais um vez nesse caso

a dificuldade do modelo de 128 filtros em manter a acurácia elevada.

A Figura 37 d) apresenta os dados de tempo de treinamento para os modelos em questão. Novamente se observa um tempo bastante superior para as redes com 4 filtros por camada, sendo que o modelo de duas camadas resulta em um valor de mais de 150%, comparado ao de uma camada.

Quando se adiciona a camada densa, a partir de 64 filtros, os tempos de treinamento para os modelos de uma camada se tornam superiores aos de duas camadas. Nos modelos de uma única camada, a operação de *pooling* é aplicada uma única vez, por isso o volume dos dados não foi reduzido significativamente. Quando esse volume de dados é enviado a uma camada densa, o número de conexões, pesos e operações realizadas é bastante expressivo.

A Figura 37 b) mostra os dados referentes ao consumo energético das redes testadas. Pode-se observar um consumo levemente superior quando se empregam duas camadas convolucionais, porém, novamente, o maior influenciador no consumo dos modelos é a quantidade de filtros.

Tendo em vista o tempo de inferência, pode-se observar no gráfico presente na Figura 37 c) que o modelo de uma camada possui maior latência do que o modelo de duas camadas. Essa situação segue a mesma linha da análise apresentada no tempo de treinamento. A presença de uma única operação de *pooling* resulta em um maior volume de dados e, consequentemente, uma necessidade de tempo maior para a realização das operações da camada densa.

Realizada a análise individual, foi feita a verificação dos resultados das grandezas em função do número de camadas utilizadas. Novamente a Tabela 9 foi empregada para o ajuste dos valores aos gráficos. A figuras 38, 39 e 40 apresentam respectivamente os valores para os modelos de uma, duas camadas convolucionais e uma camada densa e duas camadas convolucionais e duas camadas densas.

Nota-se na Figura 38 uma tendência de tempo de treinamento, consumo energético e tempo de inferência bem superior por parte do modelo contendo 128 filtros. Por outro lado, o modelo com 4 filtros resultou em uma acurácia reduzida. Os modelos de 16 e 32 filtros se mostram mais equilibrados que os demais.

Os modelos de duas camadas, cujas grandezas são mostradas na Figura 39, resultam em uma piora significativa no tempo de treinamento para a rede de 4 filtros, além de uma acurácia um pouco abaixo das demais. O modelo de 128 filtros novamente apresenta valores superiores de consumo e tempo de inferência, enquanto que a rede com 16 filtros, mesmo exibindo acurácia um pouco abaixo dos

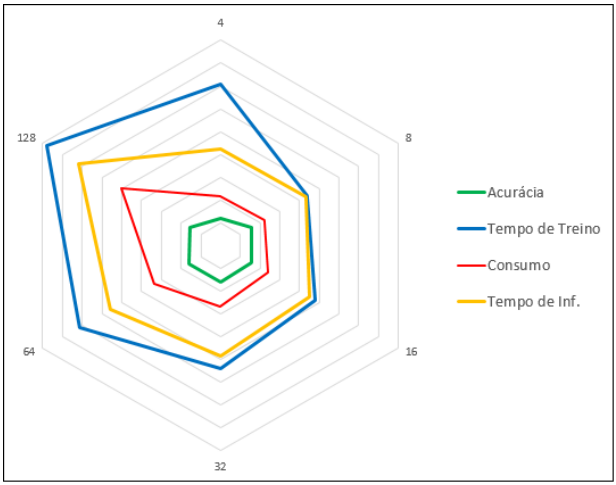

Figura 38: Análise das Grandezas Resultantes para Modelos com uma Camada Convolucional e uma Camada Densa.

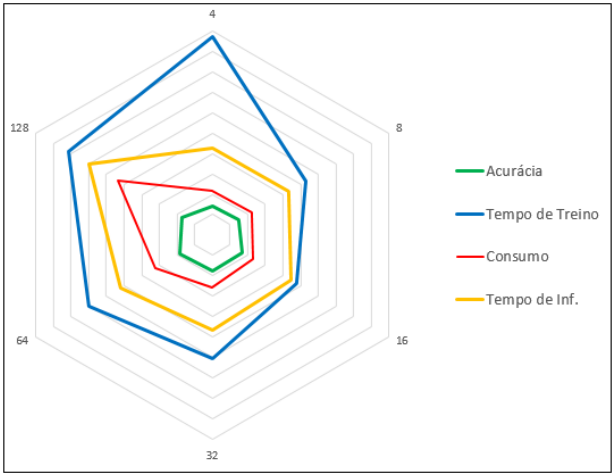

Figura 39: Análise das Grandezas Resultantes para Modelos com duas Camadas Convolucionais e uma Camada Densa.

melhores resultados, possui um maior equilibrio entre as quatro grandezas.

Na Figura 40, em que são apresentados os resultados para os

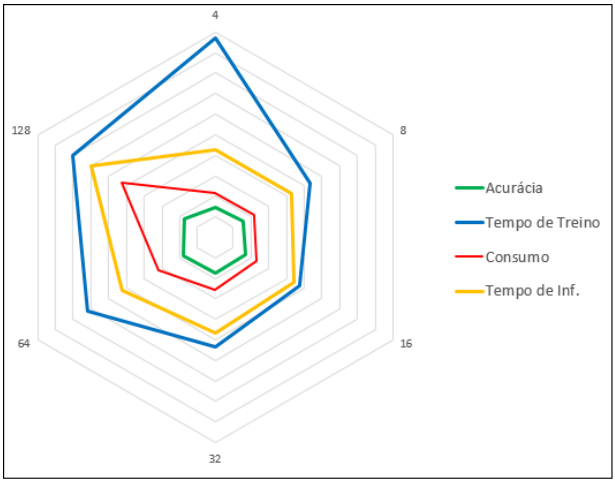

Figura 40: Análise das Grandezas Resultantes para Modelos com duas Camadas Convolucionais e duas Camadas Densas.

modelos com duas camadas densas, é possível notar um valor bastante alto para o tempo de treinamento do modelo de 4 filtros. Além dele, as redes de 64 e 128 filtros também apresentaram tempos superiores. O modelo de 128 filtros exibe um aumento considerável dos valores de consumo e tempo de inferência, quando comparado aos demais. Por outro lado, as redes de 16 e 32 filtros dispõem dos resultados mais equilibrados entre os testados.

5.2.4.4 Modelos de 1 a 4 Camadas Convolucionais e Camada Densa

Dando sequência aos testes e às variações adotadas, foram realizados ensaios utilizando-se redes com 16 filtros convolucionais com número de camadas variáveis, além de uma camada densa de 16 nós. Com esse teste esperava-se identificar a influência da adição de estágios convolucionais em um modelo com a presença de uma camada densa. A Figura 41 a) mostra os resultados obtidos para os testes de acurácia. O objetivo desse teste é verificar os resultados para diferentes quantidades de estágios convolucionais em modelos contendo uma camada densa de poucos nós.

É possível verificar que a utilização de uma única camada não proporciona resultados satisfatórios, visto que a extração de

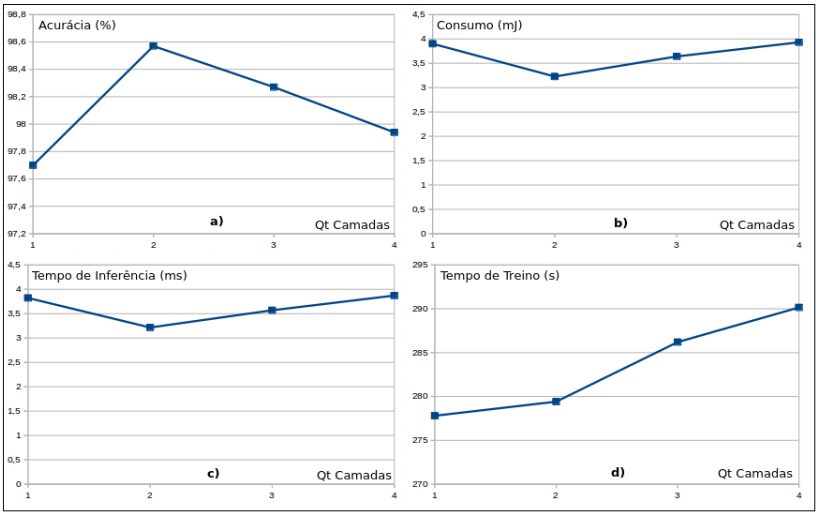

Figura 41: Valores Referentes aos Modelos com Quantidades de Camadas Convolucionais de 16 filtros Variável e uma Camada Densa de 16 nós, com a) acurácia, b) consumo, c) tempo de inferência e d) tempo de treino.

características, em virtude da pouca variedade de filtros, se torna comprometida. Por outro lado, no caso dos modelos de 3 e 4 camadas ocorre um declínio na acurácia em virtude do já comentado problema de perda de informações, ocasionada pela sucessiva aplicação da operação de *pooling*. Nesse cenário, percebe-se com clareza a vantagem da utilização do modelo de duas camadas.

A Figura 41 d) traz os resultados dos tempos de treinamento dos modelos. Nessa situação as redes de uma e duas camadas apresentaram resultados relativamente próximos, enquanto que os outros dois modelos trazem valores um pouco acima. Mais uma vez fica bastante evidente a vantagem do emprego da rede com somente duas camadas.

Com relação ao consumo energético dos modelos, apresentado na Figura 41 b), mostra-se mais uma vez vantajoso o emprego da rede de duas camadas. O modelo em questão resultou em um ponto de mínimo, quando se verifica a quantidade de operações realizadas. Enquanto que na rede de um estágio convolucional, a camada densa apresenta uma enorme quantidade de conexões o que resulta em uma enorme quantidade de cálculos; nas redes de 3 e 4 camadas os níveis

convolucionais representam o maior volume de operações de modelo.

A rede de duas camadas, por outro lado, nessa aplicação específica, representa um maior equilíbrio entre o volume de cálculos das camadas convolucionais e da camada densa.

Por último, a respeito do tempo de inferência, a Figura 41 c) apresenta um comportamento semelhante ao da análise de consumo. Em função disso, as explicações já apresentadas servem para esse caso também.

5.2.4.5 Modelos com Duas Camadas Densas

Uma nova rodada de testes foi realizada, verificando a influência da utilização de 1 ou 2 camadas densas em um modelo convolucional. Nas redes comumente disponibilizadas, é bastante ampla a utilização de mais de uma camada densa no estágio de classificação do modelo. Foram testados modelos contendo duas camadas com quantidades variáveis de filtros convolucionais em conjunto com camadas densas de 1 ou 2 estágios de 16 nós cada. A Figura 42 a) apresenta o comparativo de acurácia para as duas situações. O objetivo desse teste é verificar se existe vantagem em utilizar mais de uma camada densa nos modelos quando se visa tempo e consumo.

O gráfico mostra um resultado bastante próximo para os casos analisados. Os modelos de duas camadas tiveram acurácia levemente superior em quatro casos, enquanto que o de uma camada foi superior em dois. Aparentemente existe uma leve vantagem na utilização de duas camadas densas no estágio de classificação das redes convolucionais.

Com relação ao tempo de treinamento, a Figura 42 d) exibe os resultados. É possível notar que ambas as redes apresentam um comportamente bastante próximo, sendo mais favorável para uma ou outra dependendo a quantidade de filtros.

Analisando os resultados de consumo energético e tempo de inferência, disponíveis nas figuras 42 b) e 42 c), respectivamente, nota-se um comportamento praticamente idêntico para ambos os modelos, o que se faz concluir que a adição de uma camada densa não gera impactos no desempenho do modelo. Vale lembrar que esses resultados foram obtidos por meio da execução dos modelos no dispositivo Intel Movidius NCS, um equipamento otimizado para esse tipo de aplicação. Em outro tipo de hardware, a adição de camadas

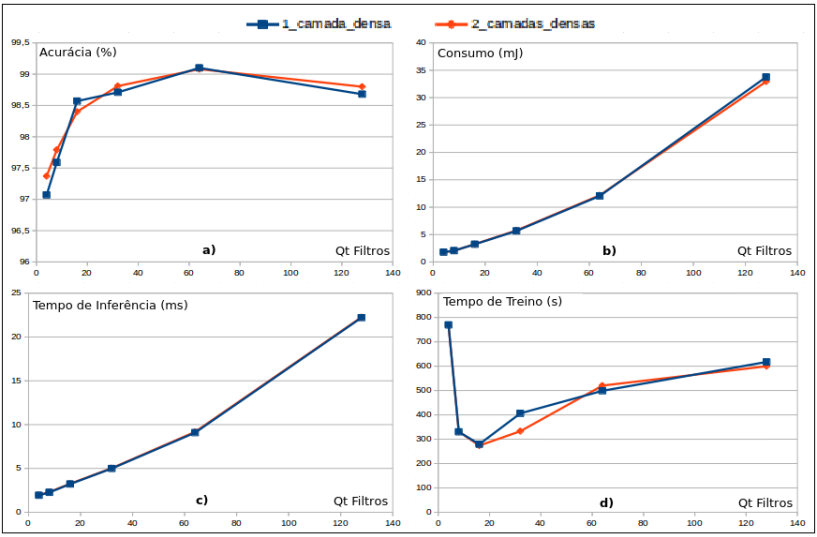

Figura 42: Comparativo de Valores entre Modelos com duas Camadas de Filtros Variáveis de uma ou duas Camadas Densas de 16 nós, com a) acurácia, b) consumo, c) tempo de inferência e d) tempo de treino.

densas pode resultar em alterações consideráveis nos resultados.

5.2.5 Comparativo de Modelos Com e Sem Camada Densa

Após a verificação da influência das camadas densas em diferentes cenários, foi realizado o comparativo direto entre a utilização ou não das camadas densas nos modelos. As análises feitas englobam os modelos de um e dois estágios convolucionais com quantidade variável de filtros por camada.

A Figura 43 a) exibe o comparativo de acurácia para os modelos de uma camada. É possível notar que com a adição da camada densa ao modelo reduz-se a variação nos resultados de acurácia a partir da utilização de 8 filtros na camada convolucional. Por outro lado, sem essa adição, os resultados mostram certa falta de linearidade com o aumento da quantidade de filtros.

Nesse caso, analisando exclusivamente os valores de acurácia, não fica clara a vantagem ou não na utilização de camadas densas no modelo.

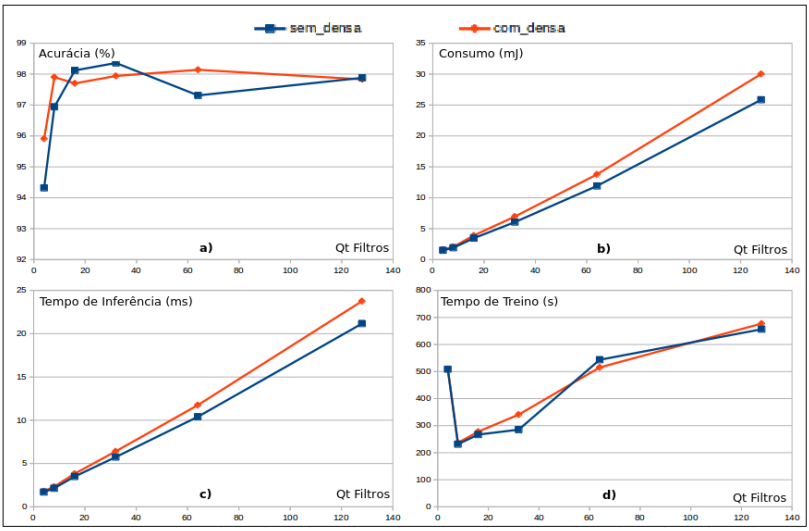

Figura 43: Comparativo de Valores entre Modelos com e sem Camada Densa - 1 Camada Convolucional, com a) acurácia, b) consumo, c) tempo de inferência e d) tempo de treino.

A Figura 43 d) mostram os comparativos de tempo de treinamento. De modo semelhante ao que ocorreu com a acurácia, a não utilização da camada densa gerou resultados não lineares, enquanto que na adição dela os dados possuem comportamento linear em função do aumento de filtros. Em virtude dos resultados oscilantes, a utilização ou não do estágio denso fica a critério da quantidade de filtros empregados.

O consumo energético e o tempo de inferência das redes de uma camada, mostrados nas figuras 43 b) e 43 c), indicam um crescimento praticamente linear nos valores em função da variação da quantidade de filtros com uma taxa maior de aumento para as redes com camada densa. No caso do emprego de camadas densas mais volumosas espera-se uma taxa ainda maior, proporcional à quantidade de nós.

No caso da utilização de duas camadas convolucionais, com relação a acurácia dos modelos, presentes na Figura 44 a), é possível notar certa vantagem na utilização de modelos contendo a camada densa. Os ganhos chegam a ser superiores a 1%, o que para certas aplicações pode ser algo considerável.

Analisando o tempo de treinamento, diferente dos modelos de

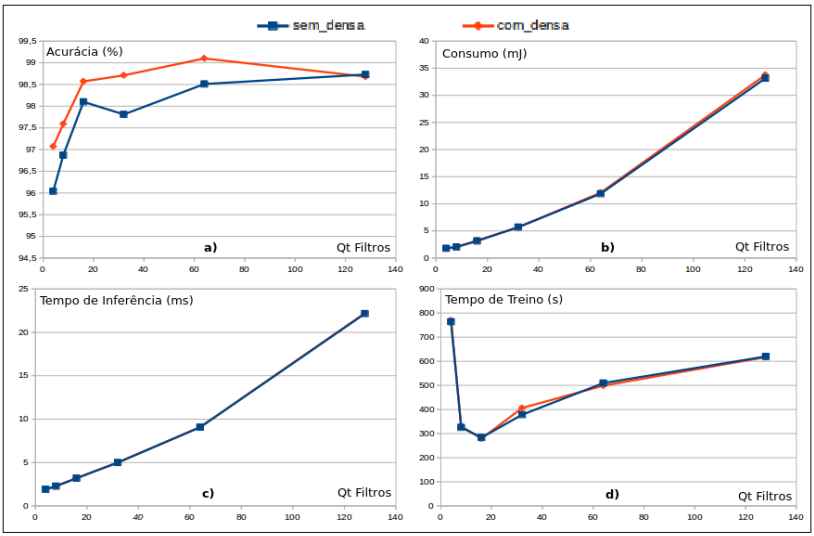

Figura 44: Comparativo de Valores entre Modelos com e sem Camada Densa - 2 Camadas Convolucionais, com a) acurácia, b) consumo, c) tempo de inferência e d) tempo de treino.

uma camada, nesse caso os resultados não apresentam oscilação significativa. Além disso, pode-se notar que a adição da camada densa não gerou variação notável nos valores. Deve-se levar em consideração que foi utilizada camada densa de 16 nós. No emprego de quantidades maiores de nós, variações significativas nos tempos podem ocorrer.

Com relação ao consumo, apresentado na Figura 44 b), nota-se uma equiparação nos resultados, em que se pode concluir que a camada densa de 16 nós não resultou em aumento de consumo das redes.

No caso do tempo de inferência, cujos dados são exibidos na Figura 44 c), verifica-se uma taxa de crescimento idêntica para ambos os modelos. No caso da necessidade de se utilizar mais nós na camada densa, esse valor deve apresentar alteração proporcional.

5.3 ANÁLISE DA METODOLOGIA DE DOIS NÍVEIS

Após o estudo da influência das camadas de um modelo convolucional nas quatro grandezas analisadas, passou-se para a análise da aplicação de dois níveis de inferência em um sistema que

exija saídas de múltiplas classes. Para isso foram selecionadas as redes Inception_V3 e GoogLeNet, redes utilizadas em diversas aplicações, inclusive nos trabalhos relacionados. Para trabalhar em conjunto com essas redes, foram selecionados quatro modelos complementares, baseados nos resultados dos testes apresentados na seção 5.2, que foram posteriormente treinados utilizando o *dataset Plant Village* completo, a fim de verificar a acurácia alcançada por esses modelos, uma vez que na seção 5.2 o banco de dados foi empregado parcialmente. A Tabela 12 mostra as redes utilizadas, bem como seus valores de acurácia[1], tempo de inferência e consumo energético. Os melhores e piores valores para cada tipo de rede foram destacados, respectivamente, em verde e vermelho.

Tabela 12: Modelos Utilizados nos Testes de Dois Níveis.

Modelo	**Tipo**	**Acurácia**	**Tempo de Inferência (ms)**	**Consumo (mJ)**
Inception_V3	Multiclasse	**88,6%**	**328,7**	**620,1**
GoogLeNet	Multiclasse	**99,34%**	**96,9**	**177,4**
Proposta1	Complementar	**94,48%**	**5,04**	**5,66**
Proposta2	Complementar	96,04%	9,17	12,17
Proposta3	Complementar	**97,68%**	9,59	12,79
Proposta4	Complementar	96,31%	**22,9**	**35,28**

As redes complementares treinadas possuem as seguintes topologias: Proposta1 contém duas camadas convolucionais de 32 filtros e duas camadas densas de 16 nós; Proposta2 contém duas camadas convolucionais de 64 filtros e duas camadas densas de 16 nós; Proposta3 contém três camadas convolucionais de 64 filtros e duas camadas densas de 16 nós; Proposta4 contém três camadas convolucionais de 128 filtros e duas camadas densas de 16 nós. Todas as quatro redes utilizam o *Batch Normalization*.

A rede Proposta2 foi selecionada em virtude de ter obtido o maior valor de acurácia entre os modelos testados na seção anterior. Os demais exemplares são variações deste, a fim de identificar o que melhor

[1]Valores de acurácia dos modelos Inception_V3 e GoogLeNet provenientes, respectivamente, de Mohanty, Hughes e Salathé (2016) e Gandhi et al. (2018).

se adequa ao novo banco de dados. Verifica-se mais uma vez que a utilização de camadas com 64 filtros obteve resultados mais expressivos do que os com 128. Por outro lado, anteriormente se havia verificado um melhor resultado nas redes de somente duas camadas, porém, para o *dataset* expandido, o modelo de três camadas convolucionais apresentou acurácia superior. Como já foi visto, a inserção de uma camada extra ao modelo não agrega significativamente em consumo energético e tempo de inferência.

As redes complementares, propostas na seção 5.2, mostram um tempo de inferência e um consumo energético bastante inferior aos dos modelos tradicionalmente empregados, nesse caso com redução de tempo de 4 a 65 vezes, e de consumo de 5 a 190 vezes. Com relação à acurácia, os quatro modelos testados alcançaram resultados superiores ao da rede Inception_V3. Por outro lado, obtiveram valores inferiores ao da rede GoogLeNet. Vale ressaltar que os quatro modelos testados não passaram por nenhum processo de refinamento dos resultados, podendo, caso isso seja realizado, alcançar acurácias maiores.

Utilizando um modelo complementar em conjunto com um modelo tradicional multiclasse, obtém-se o sistema de dois níves de inferência. Com base nas redes selecionados, para realizar a verificação da eficácia da aplicação dessa metodologia foram calculados os tempos totais de inferência para os diversos conjuntos em duas situações: sem a utilização da metodologia em dois níveis, em que somente o modelo tradicional é empregado (Tempo 100%); com a utilização da metodologia, em uma situação de 50% de positivos (Tempo 50%). A Tabela 13 mostra os resultados obtidos considerando a execução de 100 inferências. Os valores destacados em verde e vermelho indicam, respectivamente, o melhor e o pior tempo total.

Nota-se, principalmente para os conjuntos contendo a rede Inception_V3, uma redução bastante significativa nos valores. Porém, mesmo apresentado reduções bastante expressivas, os conjuntos que utilizam esse modelo, apresentam valores de tempo bem maiores do que os que empregam a rede GoogLeNet. Por isso, se o tempo for uma restrição presente no sistema, a rede Inception_V3 deve ser desconsiderada.

Na análise seguinte, foi utilizada a equação 4.4, apresentada na seção 4.7, bem como a equação para o cálculo dos ganhos. O objetivo é identificar o valor máximo de inferências positivos, que torna a metodologia vantajosa em tempo e consumo para determinada aplicação. As tabelas 14 e 15 apresentam os resultados para tempo de inferência e consumo energético respectivamente. Os resultados mais

Tabela 13: Tempos Totais de Inferência para 100 eventos.

Modelo Complementar	**Modelo Tradicional**	**Tempo 100% (ms)**	**Tempo 50% (ms)**
Proposta1	Inception_V3	32870	16939
Proposta2	Inception_V3	32870	17352
Proposta3	Inception_V3	32870	17394
Proposta4	Inception_V3	32870	**18725**
Proposta1	GoogLeNet	9690	**5349**
Proposta2	GoogLeNet	9690	5762
Proposta3	GoogLeNet	9690	5804
Proposta4	GoogLeNet	9690	7135

e menos expressivos são destacados, respectivamente, em verde e vermelho.

São apresentados nas tabelas o conjunto de modelos utilizados, bem como o valor de máximo positivo (Pmax) do conjunto, valor esse que determina o máximo de inferências que culminem em resultados de detecção positivo, exigindo com que o modelo tradicional seja requisitado, que torna a aplicação de dois níveis vantajosa. Percentuais de positivos acima do valor de máximo acabam ocasionando maior consumo e tempo total de inferência. Além disso, é apresentado em Positivo 50% o ganho nos valores totais, quando o conjunto é utilizado em uma aplicação cuja metade das inferências resulta na utilização do modelo tradicional. Para isso, foi utilizado na equação o valor do termo P como 0,5.

Nota-se que, para a maioria dos conjuntos o valor de máximo positivo ultrapassou os 90%, garantindo assim uma ampla faixa de trabalho para o sistema. Além de obter os piores resultados dessa amostra, o conjunto Proposta4 e GoogLeNet, especificamente a rede Proposta4, não apresenta resultados satisfatórios em tempo, acurácia e consumo, comparado com os demais modelos disponíveis. Portanto, essa combinação não deve ser desconsiderada para um aplicação real. Logo, para uma aplicação embarcada real os conjuntos mais adequados seriam os que empregam os modelos Proposta1, Proposta2, Proposta3

em conjunto com a rede GoogLeNet. Entre esses três, a escolha se daria em virtude da restrição mais importante, tempo e consumo ou acurácia.

Tabela 14: Resultados de Tempo de Inferência para a Inferência em Dois Níveis.

Modelo Complementar	**Modelo Tradicional**	**Máximo Positivo**	**Positivo 50%**
Proposta1	Inception_V3	**98,47%**	**48,47%**
Proposta2	Inception_V3	97,21%	47,21%
Proposta3	Inception_V3	97,08%	47,08%
Proposta4	Inception_V3	93,03%	43,03%
Proposta1	GoogLeNet	94,8%	44,8%
Proposta2	GoogLeNet	90,54%	40,54%
Proposta3	GoogLeNet	90,1%	40,1%
Proposta4	GoogLeNet	**76,37%**	**26,37%**

Com relação aos valores de consumo dos conjuntos analisados, a análise realizada na tabela anterior se mantém. Vale notar que os valores de Máximo Positivo e de Positivo 50%, quando analisado o consumo energético foram levemente superiores aos apresentados na análise pelo tempo. Isso indica que o emprego desse método de inferência resulta em uma faixa de trabalho maior e de ganhos superiores no caso do consumo energético do sistema. Visto isso, é mais indicado utilizar os dados de tempo para a verificação da eficácia do método, uma vez que os máximos e os ganhos são menos amplos.

Utilizando o conjunto Proposta3 e GoogLeNet foi gerado o gráfico da Figura 45, em que é possível verificar o ganho do método de inferência em dois níveis em função do percentual de inferências positivas. Foram plotados ambos os dados de tempo e consumo.

Nota-se que, com valores próximos a 0% de positivos o conjunto supera os 90% de ganho para ambas as grandezas analisadas. A medida que o valor de positivos cresce, existe uma declinação linear no resultado dos ganhos, e como já comentado anteriormente, com o consumo apresentando um valor levemente superior. No momento que a taxa de positivos extrapola o máximo, a

Tabela 15: Resultados de Consumo Energético para a Inferência em Dois Níveis.

Modelo Complementar	**Modelo Tradicional**	**Máximo Positivo**	**Positivo 50%**
Proposta1	Inception_V3	**99,08%**	**49,08%**
Proposta2	Inception_V3	98,04%	48,04%
Proposta3	Inception_V3	97,94%	47,94%
Proposta4	Inception_V3	94,31%	44,31%
Proposta1	GoogLeNet	96,8%	46,8%
Proposta2	GoogLeNet	93,14%	43,14%
Proposta3	GoogLeNet	92,8%	42,79%
Proposta4	GoogLeNet	**80,11%**	**30,11%**

utilização dessa estratégia deixa de ser benéfica, resultando nesse caso em perdas de 9,8% e 7% para tempo e consumo respectivamente.

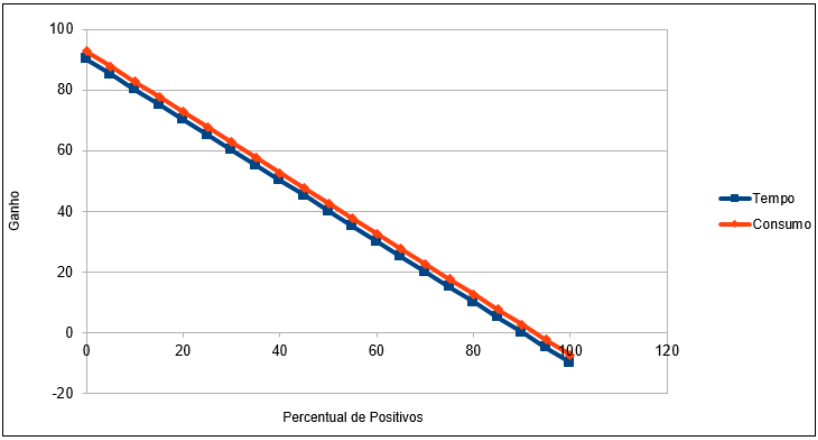

Figura 45: Análise dos Ganhos de Tempo e Consumo em Função do Índice de Positivos na Inferência.

6 CONCLUSÃO

O trabalho aqui apresentado abordou a utilização de soluções de visão computacional baseadas em *Deep Learning* aplicadas a Sistemas Embarcados. As pesquisas na área de redes convolucionais são focadas na melhora dos índices de acurácia, sem grandes preocupações com relação aos demais fatores de desempenho. Em virtude, disso buscou-se analisar e compreender a influência das camadas que compõem um modelo convolucional a fim de permitir a construção de uma rede mais otimizada para cada aplicação, utilizando como recurso computacional um dispositivo dedicado para a execução de modelos convolucionais. Além disso, foram propostas duas metodologias que visam, por meio de uma nova abordagem na resolução dos problemas, reduzir o tempo de inferência e o consumo energético dos modelos. A primeira metodologia proposta visou à substituição de modelos classificadores por modelos complementares detectores de características. Com o estudo da influência das camadas, buscou-se simplificar o processo de elaboração de modelos nesse formato. A segunda metodologia apresentou a utilização de duas redes, uma complementar e outra tradicional, que quando atuando em conjunto, podem reduzir o consumo energético e o tempo de inferência total do sistema desenvolvido.

Inicialmente, foram apresentados os resultados de tempo da execução de diversos modelos em diferentes plataformas computacionais. Verificou-se um resultado bastante expressivo para a utilização do chip Myriad 2, dispositivo esse dedicado a aplicações embarcadas de baixo consumo energético. Esse chip é uma excelente alternativa para a utilização em conjunto com plataformas como Raspberry Pi e similares, ou até mesmo na elaboração de sistemas altamente dedicados, no desenvolvimento de soluções de visão computacional.

Após essa etapa, foram apresentados os resultados para a influência das camadas nos modelos com relação à acurácia, ao tempo de treino, ao consumo e ao tempo de inferência, resultados esses que fazem parte da primeira metodologia proposta, na qual, por meio dessas análises, se pretendia simplificar e buscar um maior entendimento no processo de desenvolvimento do modelo.

Primeiramente foi possível perceber que a utilização de camadas convolucionais com quantidades muito baixas de filtros (4 e 8), apesar do baixo consumo e tempo de inferência, não obteve

resultados significativos em acurácia e, em geral, apresentou valores bastante altos para o tempo de treinamento. Com isso, não é aconselhável o emprego dessas quantidades de filtros. No outro extremo, as redes contendo 128 *kernels* não obtiveram os melhores valores de acurácia, além de apresentarem, no geral, os valores mais altos em consumo e tempo de inferência. Vale salientar que, para cada *dataset*, o valor de acurácia pode sofrer alterações, portanto é necessário realizar alguns treinamentos para verificar o comportamento. Para a aplicação mostrada nesse trabalho não se recomenda o uso dessa configuração.

Com relação ao número de camadas presente nos modelos, deve-se evitar os extremos. O emprego de uma única camada, no geral, leva a resultados não muito expressivos de acurácia. Foi mostrado que a adição de mais camadas convolucionais não gera um impacto muito significativo nas demais grandezas analisadas, uma vez que ao realizar a adição de níveis convolucionais extras, existe uma redução significativa no tempo de execução da camada densa ou de saída. Nessa situação o que mais influencia nesses valores é a quantidade de filtros em cada camada. Com relação à utilização de modelos com 4 estágios, foi verificado que, em virtude das sucessivas operações de *pooling*, houve a perda de muitas informações, e com isso, uma degradação dos resultados. Portanto, para a topologia utilizada, a melhor estratégia é o uso de duas ou três camadas convolucionais.

Outra técnica testada, o *Batch Normalization*, apresentou uma leve melhora na acurácia dos modelos sem comprometer os valores de consumo e tempo de inferência. Para o tempo de treino, houve uma piora nos resultados obtidos. Portante, é válida a utilização da normalização nas redes desenvolvidas. Vale ressaltar que os resultados são referentes ao chip Myriad 2; para outros dispositivos, os valores podem sofrer alterações.

Analisando a utilização das camadas densas no modelo, ela deve ser empregada com cuidado, principalmente para não comprometer os resultados de tempo de inferência e consumo. As camadas densas podem ser aplicadas a modelos contendo duas ou mais camadas convolucionais, a fim de reduzir o número de conexões entre os níveis. Os testes realizados mostraram que a quantidade de 16 nós obteve resultados bastante interessantes. Utilizando 512 nós alcançou-se acurácia superior, porém ao custo de tempo e consumo. Nesse caso deve-se ponderar e priorizar os requisitos pretendidos. O emprego de uma segunda camada densa ao modelo pode ser

considerado, exigindo uma análise dos resultados de acurácia da aplicação desejada.

Com relação à utilização da metodologia de dois níveis de inferência foi mostrado que, dependendo do percentual de detecções positivo, o sistema pode alcançar ganhos que superam os 90%. A seleção de uma boa rede complementar, que apresente acurácia elevada, em conjunto com baixo tempo de inferência e consumo, pode otimizar ainda mais os resultados. A utilização de redes tradicionais multiclasse pequenas também deve ser considerada para reduzir ainda mais os valores.

Esse trabalho buscou apresentar um novo caminho para o desenvolvimento de aplicações com o intuito de serem empregadas em sistemas embarcados, por isso, uma comparação direta com as técnicas; *pruning*, destilação; empregadas atualmente nessa área não se faz possível. Os trabalhos relacionados apresentados no capítulo 3 podem ser utilizados como base para avanços nos métodos propostos nesse trabalho a fim de otimizar ainda mais os resultados obtidos.

Para trabalhos futuros nesse tema sugere-se:

- Exploração de outros *datasets* para verificação da consistência dos dados de acurácia.
- Estudo utilizando novas topologias de redes: utilizar duas ou mais camadas convolucionais antes do *pooling* e utilizar a arquitetura das redes SqueezeNet e MobileNet.
- Realização de estudo semelhante utilizando diferentes plataformas computacionais embarcadas.
- Implementação prática das técnicas apresentadas.
- Aplicação aos modelos de técnicas de *pruning* e/ou destilação.

REFERÊNCIAS

ABADI, M. et al. Tensorflow: a system for large-scale machine learning. Em: **OSDI**. [S.l.: s.n.], 2016. v. 16, p. 265–283.

ADEGBIJA, T. et al. Microprocessor optimizations for the internet of things: a survey. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 37, n. 1, p. 7–20, 2018.

BARRY, B. et al. Always-on vision processing unit for mobile applications. **IEEE Micro**, IEEE, v. 35, n. 2, p. 56–66, 2015.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV: Computer vision with the OpenCV library**. [S.l.]: "O'Reilly Media, Inc.", 2008.

BROCK, J. D.; BRUCE, R. F.; CAMERON, M. E. Changing the world with a raspberry pi. **Journal of Computing Sciences in Colleges**, Consortium for Computing Sciences in Colleges, v. 29, n. 2, p. 151–153, 2013.

BUDUMA, N.; LOCASCIO, N. **Fundamentals of deep learning: Designing next-generation machine intelligence algorithms**. [S.l.]: "O'Reilly Media, Inc.", 2017.

CHANGPINYO, S.; SANDLER, M.; ZHMOGINOV, A. The power of sparsity in convolutional neural networks. **arXiv preprint arXiv:1702.06257**, 2017.

COPELAND, B. J. **Artificial intelligence**. Encyclopedia Britannica, inc., Aug 2018. Disponível em: <https://www.britannica.com/technology/artificial-intelligence>.

EDWARDS, S. et al. Design of embedded systems: Formal models, validation, and synthesis. **Proceedings of the IEEE**, IEEE, v. 85, n. 3, p. 366–390, 1997.

FERENTINOS, K. P. Deep learning models for plant disease detection and diagnosis. **Computers and Electronics in Agriculture**, Elsevier, v. 145, p. 311–318, 2018.

GANDHI, R. et al. Plant disease detection using cnns and gans as an augmentative approach. Em: IEEE. **Innovative Research and Development (ICIRD), 2018 IEEE International Conference on**. [S.l.], 2018. p. 1–5.

GOODFELLOW, I. et al. **Deep learning**. [S.l.]: MIT press Cambridge, 2016.

GUPTA, V. **Without Good Analysis, Big Data Is Just a Big Trash Dump**. Entrepreneur, May 2015. Disponível em: <https://www.entrepreneur.com/article/246470>.

HA, K. et al. Towards wearable cognitive assistance. Em: ACM. **Proceedings of the 12th annual international conference on Mobile systems, applications, and services**. [S.l.], 2014. p. 68–81.

HASANPOUR, S. H. et al. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. **arXiv preprint arXiv:1608.06037**, 2016.

HLAING, C. S.; ZAW, S. M. M. Model-based statistical features for mobile phone image of tomato plant disease classification. Em: IEEE. **Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2017 18th International Conference on**. [S.l.], 2017. p. 223–229.

HOLLEMANS, M. **Google's MobileNets on the iPhone**. machinethink.net, Jun 2017. Disponível em: <http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/>.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, 2017.

HUGHES, D.; SALATHÉ, M. et al. An open access repository of images on plant health to enable the development of mobile disease diagnostics. **arXiv preprint arXiv:1511.08060**, 2015.

IANDOLA, F. N. et al. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. **arXiv preprint arXiv:1602.07360**, 2016.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. **arXiv preprint arXiv:1502.03167**, 2015.

JAISWAL, B.; GAJJAR, N. Deep neural network compression via knowledge distillation for embedded applications. Em: IEEE. **Engineering (NUiCONE), 2017 Nirma University International Conference on**. [S.l.], 2017. p. 1–4.

KAPLAN, J. **Artificial Intelligence: What everyone needs to know**. [S.l.]: Oxford University Press, 2016.

KHAN, S. et al. A guide to convolutional neural networks for computer vision. **Synthesis Lectures on Computer Vision**, Morgan & Claypool Publishers, v. 8, n. 1, p. 1–207, 2018.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. Em: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. p. 1097–1105.

LI, Q.; XIAO, Q.; LIANG, Y. Enabling high performance deep learning networks on embedded systems. Em: IEEE. **Industrial Electronics Society, IECON 2017-43rd Annual Conference of the IEEE**. [S.l.], 2017. p. 8405–8410.

MITCHELL, T. M. **Machine learning**. [S.l.]: McGrawHill, 1997.

MOHANTY, S. P.; HUGHES, D. P.; SALATHÉ, M. Using deep learning for image-based plant disease detection. **Frontiers in plant science**, Frontiers, v. 7, p. 1419, 2016.

NETWORKING, C. V. Cisco global cloud index: Forecast and methodology, 2016-2021. white paper. **Cisco Public, San Jose**, 2018. Disponível em: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html>.

PARK, E. et al. **Large Scale Visual Recognition Challenge 2017 (ILSVRC2017)**. 2017. Disponível em: <http://image-net.org/challenges/LSVRC/2017/>.

PATTANAYAK, S. **Pro Deep Learning with TensorFlow**. [S.l.]: Apress, 2017.

PENA, D. et al. Benchmarking of cnns for low-cost, low-power robotics applications. Em: **RSS 2017 Workshop: New Frontier for Deep Learning in Robotics**. [S.l.: s.n.], 2017.

POLINO, A.; PASCANU, R.; ALISTARH, D. Model compression via distillation and quantization. **arXiv preprint arXiv:1802.05668**, 2018.

POOJA, V.; DAS, R.; KANCHANA, V. Identification of plant leaf diseases using image processing techniques. Em: IEEE. **Technological Innovations in ICT for Agriculture and Rural Development (TIAR), 2017 IEEE**. [S.l.], 2017. p. 130–133.

PRINCE, S. J. **Computer vision: models, learning, and inference**. [S.l.]: Cambridge University Press, 2012.

PULLI, K. et al. Real-time computer vision with opencv. **Communications of the ACM**, ACM, v. 55, n. 6, p. 61–69, 2012.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. [S.l.]: Malaysia; Pearson Education Limited,, 2016.

SAGIROGLU, S.; SINANC, D. Big data: A review. Em: IEEE. **Collaboration Technologies and Systems (CTS), 2013 International Conference on**. [S.l.], 2013. p. 42–47.

SARKAR, D.; BALI, R.; SHARMA, T. **Practical Machine Learning with Python: A Problem-Solver's Guide to Building Real-World Intelligent Systems**. [S.l.]: Apress, 2018.

SATYANARAYANAN, M. The emergence of edge computing. **Computer**, IEEE, v. 50, n. 1, p. 30–39, 2017.

SHAFIEE, M. J. et al. Squishednets: Squishing squeezenet further for edge device scenarios via deep evolutionary synthesis. **arXiv preprint arXiv:1711.07459**, 2017.

SHARMA, J. **Insights of The Machine Learning and The Deep Learning**. Thinkwik Blogs, Jul 2018. Disponível em: <http://blog.thinkwik.com/insights-of-the-machine-learning-and-the-deep-learning/>.

SHI, W. et al. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 5, p. 637–646, 2016.

SHI, W.; DUSTDAR, S. The promise of edge computing. **Computer**, IEEE, v. 49, n. 5, p. 78–81, 2016.

TEXAS, I. **INA219 Zero-Drift, Bidirectional Current/Power Monitor With I2C Interface**. Texas Instruments, Dallas, 2015. Disponível em: <www.ti.com/lit/ds/symlink/ina219.pdf>.

TLHOBOGANG, B.; WANNOUS, M. Design of plant disease detection system: A transfer learning approach work in progress. Em: IEEE. **2018 IEEE International Conference on Applied System Invention (ICASI)**. [S.l.], 2018. p. 158–161.

YI, S. et al. Fog computing: Platform and applications. Em: IEEE. **2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)**. [S.l.], 2015. p. 73–78.

ZACCONE, G.; KARIM, M. R.; MENSHAWY, A. **Deep Learning with TensorFlow**. [S.l.]: Packt Publishing Ltd, 2017.

ZHU, M.; GUPTA, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. **arXiv preprint arXiv:1710.01878**, 2017.

Apêndice A – Valores de Redes Puramente Convolucionais

C1	C2	C3	T.I.(ms)	C.(mJ)	Acc.(%)	T.T.(ms)
4	0	0	1,69	1,5	94,32	508,66
4	4	0	1,91	1,783	96,04	763,83
4	4	4	-	-	92,15	902,73
8	0	0	2,138	1,91	96,94	231,46
8	8	0	2,276	2,04	96,87	325,72
8	8	8	2,31	2,1	96,26	379,46
16	0	0	3,49	3,46	98,12	266,97
16	16	0	3,19	3,15	98,1	283,53
16	16	16	3,525	3,65	98,45	280,94
32	0	0	5,74	6,06	98,36	285,23
32	32	0	5,01	5,7	97,81	378,25
32	32	32	5,29	5,93	97,96	366,13
64	0	0	10,4	11,91	97,31	543,47
64	64	0	9,067	11,86	98,51	509,49
64	64	64	9,481	12,52	98	543,87
128	0	0	21,163	25,855	97,88	656,41
128	128	0	22,159	33,155	98,73	619,79
128	128	128	22,656	34,8	98,55	609,51

Apêndice B – Valores de Redes Convolucionais com *Batch Normalization*

C1	C2	C3	T.I.(ms)	C.(mJ)	Acc.(%)	T.T.(ms)
4	0	0	1,69	1,49	94,93	828
4	4	0	1,92	1,797	95,89	1208
4	4	4	-	-	93,27	1494
8	0	0	2,133	1,92	96,15	250
8	8	0	2,21	1,994	97,68	373
8	8	8	2,552	2,32	97,9	466
16	0	0	3,465	3,387	96,79	294
16	16	0	3,19	3,25	98,14	314
16	16	16	3,47	3,53	97,97	314
32	0	0	5,75	6,04	98,05	353
32	32	0	4,982	5,652	98,51	413
32	32	32	5,29	6	97,68	418
64	0	0	10,42	11,9	97,92	527
64	64	0	9,11	12,01	98,29	566
64	64	64	9,49	12,41	98,62	531
128	0	0	21,154	26,03	98,27	702
128	128	0	22,27	33,96	98,9	686
128	128	128	22,73	35,1	98,97	728

Apêndice C – Valores de Redes com Camadas Densas

C1	C2	C3	C4	D1	T.I.(ms)	C.(mJ)	Acc.(%)	T.T.(ms)
4	0	0	0	16	1,8	1,62	95,91	508,03
4	4	0	0	16	1,934	1,788	97,07	769,38
4	4	4	0	16	-	-	96,26	909,71
4	4	4	4	16	-	-	95,1	1056,36
8	0	0	0	16	2,326	2,064	97,9	236,57
8	8	0	0	16	2,27	2,06	97,59	330,27
8	8	8	0	16	2,6	2,41	-	-
8	8	8	8	16	2,88	2,55	-	-
16	0	0	0	16	3,82	3,9	97,7	277,8
16	16	0	0	16	3,215	3,23	98,57	279,41
16	16	16	0	16	3,57	3,64	98,27	286,2
16	16	16	16	16	3,87	3,932	97,94	290,15
32	0	0	0	16	6,4	6,96	97,94	340,77
32	32	0	0	16	5	5,67	98,71	406,16
32	32	32	0	16	5,3	6,015	-	-
32	32	32	32	16	5,57	6,33	-	-
64	0	0	0	16	11,75	13,78	98,14	515,25
64	64	0	0	16	9,08	12,05	99,1	498,84
64	64	64	0	16	9,56	12,73	-	-
64	64	64	64	16	10,06	13,26	-	-
128	0	0	0	16	23,76	30,02	97,83	677,26
128	128	0	0	16	22,2	33,77	98,68	617,4
128	128	128	0	16	22,82	34,5	-	-
128	128	128	128	16	23,53	35,46	-	-

C1	C2	C3	C4	D1	T.I.(ms)	C.(mJ)	Acc.(%)	T.T.(ms)
16	0	0	0	256	16,88	17,51	-	-
16	16	0	0	256	4,015	4,044	98,4	282,96
16	16	16	0	256	3,641	3,64	-	-
16	16	16	16	256	3,853	3,922	-	-
32	0	0	0	256	32,54	34,46	-	-
32	32	0	0	256	6,575	7,45	-	-
32	32	32	0	256	5,34	6,15	-	-
32	32	32	32	256	5,52	6,26	-	-
64	0	0	0	256	63,88	68,99	-	-
64	64	0	0	256	12,3	15,24	-	-
64	64	64	0	256	9,7	12,9	-	-
64	64	64	64	256	10,02	13,2	-	-
128	0	0	0	256	128,4	140,08	-	-
128	128	0	0	256	28,6	40,06	-	-
128	128	128	0	256	23,17	35,33	-	-
128	128	128	128	256	23,2	35,5	-	-
16	0	0	0	512	31,5	32,56	-	-
16	16	0	0	512	4,95	4,92		
16	16	16	0	512	3,71	3,73	-	-
16	16	16	16	512	3,87	4,04	-	-
32	0	0	0	512	61,75	64,78	-	-
32	32	0	0	512	8,36	9,12	-	-
32	32	32	0	512	5,46	6,3	-	-
32	32	32	32	512	5,63	6,4	-	-
64	0	0	0	512	122,21	126,4	-	-
64	64	0	0	512	15,86	18,83	-	-
64	64	64	0	512	9,98	13,1	-	-
64	64	64	64	512	10,1	13,33	-	-
128	0	0	0	512	-	-	-	-
128	128	0	0	512	35,8	47,0	-	-
128	128	128	0	512	23,6	36,02	-	-
128	128	128	128	512	23,58	36,72	-	-

C1	C2	D1	D2	T.I.(ms)	C.(mJ)	Acc.(%)	T.T.(ms)
4	4	16	16	1,965	1,81	97,37	768,36
8	8	16	16	2,31	2,05	97,79	331,6
16	16	16	16	3,25	3,26	98,4	273,82
32	32	16	16	5,04	5,76	98,81	332,84
64	64	16	16	9,17	12,17	99,08	520,22
128	128	16	16	22,26	32,94	98,8	599,71
4	4	32	32	1,99	1,782	98,14	765,77
8	8	32	32	2,34	2,056	98,25	333,34
16	16	32	32	3,3	3,31	-	-
32	32	32	32	5,13	5,77	-	-
16	16	256	256	4,125	4,233	-	-
32	32	256	256	6,66	7,38	-	-
64	64	256	256	12,45	15,34	-	-
128	128	256	256	28,78	40,6	-	-
16	16	512	512	5,233	5,29	-	-
32	32	512	512	8,67	9,75	-	-
64	64	512	512	16,01	19,22	-	-
128	128	512	512	36,23	48,66	-	-