Pós-Graduação em Ciência da Computação

Aline Gondim Santos

Análise de métodos de otimização de parâmetros e tempo de inferência para modelos de aprendizagem profunda

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

Aline Gondim Santos

Análise de métodos de otimização de parâmetros e tempo de inferência para modelos de aprendizagem profunda

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Inteligência Computacional.

Orientador: Cleber Zanchettin

Recife

2019

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S237a Santos, Aline Gondim
Análise de métodos de otimização de parâmetros e tempo de inferência para modelos de aprendizagem profunda / Aline Gondim Santos. – 2019.
90 f.: il., fig., tab.

Orientador: Cleber Zanchettin.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2019.
Inclui referências.

1. Inteligência computacional. 2. Redes neurais convolucionais. 3. Redes generativas adversárias. I. Zanchettin, Cleber (orientador). II. Título.

006.31 CDD (23. ed.) UFPE - CCEN 2020-12

Aline Gondim Santos

"Análise de Métodos de Otimização de Parâmetros e Tempo de Inferência para Modelos de Aprendizagem Profunda"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 12 de dezembro de 2019.

BANCA EXAMINADORA

_____

Prof. Dr. Adriano Lorena Inácio de Oliveira
Centro de Informática/UFPE

_____

Prof. Dr. Bruno José Torres Fernandes
Escola Politécnica de Pernambuco / UPE

_____

Prof. Dr. Cleber Zanchettin
Centro de Informática/UFPE
(**Orientador**)

AGRADECIMENTOS

Eu gostaria de agradecer a meus pais, Wellington Fernandes Santos e Ana Bernadete Beltrão Gondim, por me darem todo o suporte, por me incentivarem a procurar pelo meu melhor e por serem os exemplos de pessoa que são.

Às minhas irmãs, Marina e Ana Celina, por sempre estarem presente, principalmente nos momentos de dificuldade.

Ao meu marido, Paulo, que está sempre ao meu lado como exemplo de pessoa e como fonte de suporte, carinho e amor.

Aos meu avós, tios, tias, primos e sobrinhos.

Ao meu orientador Cleber Zanchettin, que me deu uma oportunidade de carreira e sempre que solicitado forneceu suporte e direcionamento.

À todos os meus amigos.

Aos professores da banca Adriano Lorena e Bruno Fernandes.

À todos, meu muito obrigada.

Aline Gondim Santos

RESUMO

Nos últimos anos as Redes Neurais Profundas ou *Deep Neural Networks* (DNNs) se tornaram o estado da arte em diversos campos de pesquisa como Visão Computacional, Processamento de Linguagem Natural, Diagnóstico por Imagem, Sistemas de Recomendação, entre outros. O surpreendente é que elas chegaram nesse ponto poucos anos após uma Rede Neural Convolucional ou *Convolutional Neural Network* (CNN), a AlexNet, ter ganho notoriedade ao ser a vencedora da competição da ImageNet (ILSVRC) em 2012. Além do interesse proveniente dos resultados que a AlexNet apresentou na competição, o surgimento de bibliotecas de código aberto e a publicação de artigos em conjunto com seus códigos-fonte também impulsionaram o crescimento da área. Depois da AlexNet, surgiram diversas outras arquiteturas, dentre elas destacam-se, por exemplo, redes como a VGG, a GoogleLeNet, a ResNet e a Pix-2-Pix. Acompanhando o ritmo acelerado da comunidade acadêmica, logo pesquisadores passaram a desejar implantar aplicações baseadas em CNNs em ambientes reais. Muitas dessas aplicações precisam ser processadas em dispositivos com poucos recursos computacionais e, nesse ponto, os desenvolvedores se depararam com problemas relacionados às limitações de suas plataformas. Para atender a necessidade de redes mais eficientes, surgiram diversas técnicas de otimização de arquiteturas. Essas técnicas podem ser divididas entre as que são aplicadas durante ou após o treinamento dos modelos e aquelas que são aplicadas antes do seu treinamento. No primeiro dos grupos se consideram técnicas como a Poda e a Quantização e, no segundo grupo estão técnicas como a Convolução Separável em Profundidade ou *Dephtwise Separable Convolution* (DSC), a Mistura de Canais da ShuffleResNet, o Deslocamento de Canais da ShiftNet e as Contrações e Expansões da SqueezeNet. Esta dissertação propõe o estudo comparativo da utilização de diferentes técnicas de otimização nos modelos CNNs. Para tal, é proposta a implementação da DSC, e dos módulos de Mistura e de Deslocamento de Canais nas redes SqueezeNet, ResNet e Pix-2-Pix. Os experimentos são conduzidos nas bases de dados CIFAR 10 e CIFAR 100, nas duas primeiras redes e com a base *maps*$\longleftrightarrow$*satellite* na última rede. Os resultados obtidos formam um conjunto de referência que futuros desenvolvedores podem utilizar como guia na escolha entre as técnicas de otimização aqui investigadas.

Palavras-chaves: Redes Neurais Convolucionais (CNNs). Redes Generativas Adversárias (GANs). Técnicas de otimização de arquitetura. Convoluções separáveis em profundidade (DSCs). Mistura de Canais. Deslocamento de canais.

ABSTRACT

Over the past few years, Deep Neural Networks have become state of the art in different research fields such as Computer Vision, Natural Language Process, Image Diagnostics, and Recommendation Systems, among others. It is surprising, however, that they arrived at this point only a few years after a Convolutional Neural Network, the AlexNet, have gained notoriety by being the winner of the ImageNet Challenge (ILSVRC) in 2012. Beyond the interest from the AlexNet's achieved results in this competition, the emergence of open source libraries and the article's publication containing the source code algorithms also boosted this research area. After AlexNet, several other architectures emerged, we can highlight the VGG, GoogleLeNet, ResNet, and Pix-2-Pix models. Keeping up with the fast pace of the academic community, many companies soon started to develop CNNs based applications. Many of these applications need to run on devices with low computational resources. At this point, the developers encountered problems related to the limitations of their platforms. To adress the need for more efficient networks, several architecture optimization techniques have emerged. They may be divided into two main strategies: during or after training, where the network architecture is changed interactively and; before training, where the architecture definition is optimization oriented. In the first of these two groups, we have techniques like Prunning and Quantization. In the second group, we have methods like the DephtWise Separable Convolution (DSC), Channel Shuffle, Channel Shift, and the Contraction-Expansion sequence from SqueezeNet. This dissertation proposes a comparative study of the effects of different CNN's optimization techniques. The implementation of DSC, Channel Shuffle modules, and Channel Shift in the networks SqueezeNet, ResNet, and Pix-2-Pix is proposed. The experiments are conducted in the CIFAR 10 and CIFAR 100 databases for the first two networks and the maps↔satellite database for the last one. The presented results constitute a reference material that developers can use as a guide in choosing among the considered optimization techniques.

Keywords: Convolutional Neural Networks (CNNs). Generative Adversarial Networks (GANs). Architecture Optimization Techniques. DepthWise Separable Convolution (DSCs). Channel Shuffle. Channel Shift.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

float32	ponto flutuante de 32 bits
int8	inteiro de 8 bits
CNN	Rede Neural Convolucional ou *Convolutional Neural Network*
DNN	Rede Neural Profunda ou *Deep Neural Network*
DSC	Convolução Separável em Profundidade ou *Dephtwise Separable Convolution*
FLOPs	Operações de Ponto Flutuante
FPGA	vetor de portas lógicas programáveis em campo ou *Field Programmable Gate Array*
GAN	Rede Generativa Adversária ou *Generative Adversarial Network*
GPU	Unidade de Processamento Gráfico ou *Graphics Processing Unit*
IA	Inteligência Artificial
MLP	Perceptron de Multi Camadas ou *Multi Layer Perceptron*
ReLU	Unidade Neuronal Retificada ou *Rectified Linear Unit*
RNA	Rede Neural Artificial
SGD	Gradiente Descendente Estocástico

SUMÁRIO

1 INTRODUÇÃO

1.1 REVISÃO HISTÓRICA E TRABALHOS RELACIONADOS

O início das pesquisas na área de Redes Neurais Artificiais (RNAs) remonta à década de 1940. Em 1943, o psiquiatra e neurocientista Warren McCulloch e o matemático Walter Pitts publicaram o seu modelo artificial inspirado no funcionamento de um neurônio biológico (MCCULLOCH; PITTS, 1943). Pouco depois, em 1949, Donald Hebb postulou a Teoria da plasticidade sináptica, também chamada de Teoria de Hebb (HEBB, 1949). Em sua teoria, ele descreve o processo de aprendizagem como o fortalecimento das sinapses[1] entres células que disparam juntas. A aprendizagem hebbiana foi fundamental para a elaboração das primeiras regras de treinamento e é, ainda hoje, a essência do aprendizado das RNAs.

A partir desses trabalhos, uma nova área de pesquisa foi desenvolvida pela comunidade acadêmica. Neste novo campo de estudo seria possível realizar diferentes tarefas, como operações matemáticas e reconhecimento de padrões, através do aprendizado dos parâmetros de um modelo artificial de um neurônio biológico. Logo, nas décadas seguintes, surgiram variadas propostas e modificações tanto do modelo de funcionamento quanto das regras de aprendizado destes modelos baseados em dados.

Dentre as diversas publicações realizadas, vale três destaques. O primeiro destaque é para o surgimento do modelo e da regra de atualização dos pesos do *perceptron* de Rosenblat[2] (ROSENBLATT, 1958); em seguida, a concepção das redes **ADALINE** e **MADALINE**[3] por Bernard Widrow e Marcian Hoff em (WIDROW; HOFF, 1960); e, por último, destaca-se a proposta de David Rumelhart, Geoffrey Hinton e Ronald Williams de um algoritmo de treinamento por retro-propagação dos erros da rede, ou *backpropagation*[4]. A RNA proposta por eles, hoje é conhecida por Perceptron de Multi Camadas ou *Multi Layer Perceptron* (MLP).

Em (RUMELHART et al., 1988) os autores provaram matematicamente que uma MLP era capaz de implementar qualquer função contínua que ligasse os dados de entrada às saídas, desde que houvesse um número suficiente de neurônios na rede. No entanto, a MLP tem dois pontos negativos. O primeiro deles se apresenta na quantidade de dados de treinamento necessário para o ajuste do modelo. Essas redes, quando definidas com uma

[1] Sinapse é o meio através do qual duas células neuronais realizam a troca de informações por estímulos bio-químicos.

[2] Rosenblat foi o primeiro a implementar uma RNA, treinando-a para solucionar um problema com duas classes. Ele realizou esse experimento em um computador IBM 704 no Aeronautical Laboratory da universidade de Cornell.

[3] A **MADALINE** foi a primeira RNA a ser largamente utilizada em aplicações reais.

[4] Cronologicamente, o termo MLP foi sugerido por Werbos (WERBOS, 1981), embora a ideia tenha sido proposta anteriormente por Linnainmaa (LINNAINMAA, 1970). Em (RUMELHART et al., 1988) os pesquisadores apresentaram uma nova versão prática do MLP.

ou duas camadas escondidas, ao tentarem modelar um problema de alta complexidade, necessitam de um número elevado de neurônios e, quanto mais parâmetros a rede possui em uma camada, um número muito maior de exemplos de treinamento são necessários para que esta possa generalizar bem o problema. Uma forma de contornar esse ponto negativo seria adicionar mais camadas escondidas à MLP. Com mais camadas escondidas, a MLP é capaz de representar funções complexas com menos neurônios mas, com elas vem o segundo problema. Apesar do conhecido sucesso obtido pela **LeNet** de Yan LeCun(LECUN et al., 1989), empiricamente mais camadas não costumavam trazer mais poder de generalização e, por vezes, apresentavam resultados piores do que modelos com menos camadas escondidas (TESAURO, 1992). Isso pode ser explicado pelo problema do desaparecimento do gradiente, ou o *vanish gradient problem*. Ele surge quando uma rede possui muitas camadas e, durante seu treinamento, na etapa de retro-propagação do erro, o erro não alcança as camadas iniciais do modelo.

O primeiro método a contornar o *vanish gradient problem* foi publicado em 2006 por Salakhutdinov e Hinton. Salakhutdinov utilizou um treinamento por auto codificadores sucessivos (HINTON; SALAKHUTDINOV, 2006), ou *stack de auto encoders*. Já em 2010, com a adoção de uma Unidade Neuronal Retificada ou *Rectified Linear Unit* (ReLU)[5,6], o *vanish gradient problem* foi contornado por outra estratégia (NAIR; HINTON, 2010). A ReLU, juntamente com a técnica de *dropout* (HINTON et al., 2012) e do uso da Unidade de Processamento Gráfico ou *Graphics Processing Unit* (GPU)[7], permitiu em 2012 que Alex Krizhevsky treinasse e publicasse a **AlexNet** (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), rede ganhadora da competição da ImageNet[8,9] (ILSVRC) (DENG et al., 2009).

Nos anos seguintes à publicação da AlexNet, as redes profundas foram unânimes na competição ILSVRC. Os pesquisadores, em busca de maiores acurácias, saíram, em 2012, de 8 camadas e 16,4% de taxa de erro na AlexNet para a ResNet-152 com 152 camadas e 3,57% de erro em 2015 (HE et al., 2015; HE et al., 2016a) sendo que Russanovsk *et al.* reportou que o nível de erro humano nesse mesmo desafio era de 5,1% (RUSSAKOVSKY et al., 2015). Nos dois últimos anos da ILSVRC, encerrada em 2017, os ganhadores foram a GoogleLeNetV4 (SZEGEDY et al., 2017) e a SE-Net(HU; SHEN; SUN, 2018) com 3,08% e 2,25% de erro, respectivamente. Essa rápida evolução beneficiou-se do fato que, em conjunto com as publicações, muitos autores tornaram seus códigos públicos para que outros interessados pudessem evoluir suas pesquisas.

[5] A ReLU é um modelo neuronal caracterizado por uma função de ativação $f(x) = 0$ *para* $x < 0$ e $f(x) = x$ *para* $x \geq 0$.

[6] Uma função de ativação é uma regra que define se um neurônio irá transmitir informação para a célula seguinte.

[7] Mais informações sobre as GPUs em: <https://pt.wikipedia.org/wiki/Unidade_de_processamento_gráfico>.

[8] A ImageNet Large Scale Visual Recognition Challenge (ILSVRC) foi uma competição de algoritmos no problema de classificação de imagens sobre um banco de dados conhecido como ImageNet

[9] Mais informações sobre a competição e o banco de dados podem ser encontrados em <http://www.image-net.org/>.

Considerando os resultados expostos anteriormente, a tendência tem sido construir redes cada vez mais profundas e complexas, a fim de obter melhores resultados. Entretanto, tais avanços não necessariamente tem feito com que as redes sejam mais eficientes com relação ao número de parâmetros, ao custo computacional e ao tempo de inferência.

Exemplificando, dado que duas redes possuem a mesma taxa de acerto, quanto mais parâmetros uma rede possui, maior o espaço de armazenamento ocupado por ela e também o número de Operações de Ponto Flutuante (FLOPs) realizadas em sua inferência. Ou seja, devido a essas características, a rede menor será mais eficiente com relação ao espaço de armazenamento e ao número de parâmetros e de FLOPs. Considere as redes AlexNet e SqueezeNet, enquanto que primeira possui 60 milhões de parâmetros e ocupa 240MB, a última tem 1,25 milhão de parâmetros e ocupa 4,8MB e ambas atingem 80,3% de taxa de acerto top-5[10] na base de dados ImageNet (IANDOLA et al., 2016a).

Para analisar a eficiência de uma rede em relação ao tempo de inferência, é necessário considerar o quanto ela é paralelizável e pode se beneficiar do *hardware* em que está sendo executada. Em (BIANCO et al., 2018) são reportadas as análises sobre os tempos de inferência da AlexNet e da SqueezeNet-v1.0 em duas plataformas diferentes, uma composta por uma GPU de ponta e outra composta por uma placa de processamento cujos recursos computacionais são mais restritos. Uma rede mais paralelizável tenderá a ter uma discrepância maior entre os tempos de inferência nas duas plataformas. Já uma rede menos paralelizável, a variação entre as aferições é menor, indicando uma menor dependência quanto à plataforma na qual a rede será executada. Bianco *et al.* reportam $1,28ms$ e $28,88ms$ como os tempos de inferência médios de um único exemplo de teste para a AlexNet, já para a SqueezeNet esses valores são de $1,53ms$ e $17,00ms$. Por esses valores é possível deduzir que a segunda rede é mais eficiente com relação a sua latência quando considerados os ambientes restritivos. Entre as redes usadas como exemplo e tendo um ambiente com poucos recursos, um modelo como o da SqueezeNet seria preferível.

Apesar dos avanços não necessariamente terem trazido mais eficiência, o fato dos pesquisadores abrirem seus códigos juntamente com a publicação de seus artigos e a elevada capacidade de aprendizado apresentada ao longo dos últimos anos pelos modelos baseados em redes neurais profundas, cada vez mais, não somente pesquisadores e empresas, mas também entusiastas e hobistas são atraídos para a área. Em comum, eles almejam a criação de funcionalidades que muitas vezes precisam ser implantadas em dispositivos restritivos já que, para diversas aplicações, utilizar a inteligência computacional diretamente nos dispositivos é essencial. Isso quer dizer que, em vez de enviar os dados para um centro de processamento (*server-side*), a análise deve ser feita no dispositivo cliente (*client-side*). As vantagens buscadas por realizar o processamento mais perto do usuário e da geração dos dados são relacionadas ao tempo de resposta e à segurança das informações.

[10] Na taxa de acerto top-5 é considerado acerto caso a resposta correta esteja entre as cinco respostas mais prováveis retornadas pela rede.

Na contra-mão da tendência de construir redes cada vez mais profundas e complexas, surgiu, portanto, a necessidade de que elas fossem mais eficientes. Com isso, vários pesquisadores e empresas se debruçaram sobre a otimização das arquiteturas de redes neurais.

Os estudos para otimização de arquiteturas profundas podem ser divididos em duas vertentes, nas quais a primeira foca em comprimir uma rede pré-treinada e a segunda na definição de arquiteturas mais eficientes durante seu treinamento. O primeiro desses grupos trabalha em modificar uma rede que já tenha sido treinada e os processos utilizados podem ser: a poda de conexões (HAN; MAO; DALLY, 2015; HAN et al., 2015), ou de canais (WEN et al., 2016), que reduz as conexões redundantes enquanto mantém a acurácia do modelo; Quantização (RASTEGARI et al., 2016; SOUDRY; HUBARA; MEIR, 2014; WU et al., 2016; ZHOU et al., 2017; ZHOU et al., 2016) e fatoração (JADERBERG; VEDALDI; ZISSERMAN, 2014; JIN; DUNDAR; CULURCIELLO, 2014; LEBEDEV et al., 2014), que reduzem a representação das informações no modelo e portanto acelera os cálculos para a inferência; e a destilação (HINTON; VINYALS; DEAN, 2015), que transfere o conhecimento de uma rede complexa para outra rede menor e com performance similar. Já o segundo dos grupos trabalha na definição das arquiteturas das redes profundas a fim de que elas sejam concebidas de forma otimizada. Algumas das estratégia utilizadas para concepção de arquiteturas mais eficientes são: a utilização de camadas convolucionais ou de *pooling* no lugar das camadas totalmente conectadas; a escolha do tamanho dos filtros das camadas convolucionais; definição de blocos de rede com organização fixa, como o *inception* (SZEGEDY et al., 2016), o *fire* (IANDOLA et al., 2016a), o módulo de Mistura de Canais (ZHANG et al., 2018) e o de Deslocamento de Canais (WU et al., 2018); e a organização das conexões entre as camadas, ou os blocos, desde o início até o final de uma rede (HE et al., 2016a).

Independente do público e de qual a plataforma de implantação, uma documentação capaz de guiar o desenvolvedor é essencial. Quando nos voltamos para a vasta variedade de otimizadores de desempenho disponíveis, no entanto, é possível que aqueles que tem apenas o intuito de utilizar as CNNs como ferramentas de trabalho tenham que escolher e usar alguma das técnicas de otimização sem um conhecimento adequado sobre seus efeitos.

Nesse contexto, a presente dissertação busca estudar comparativamente os efeitos causados numa rede neural profunda considerando a implementação de diferentes propostas para otimização de sua arquitetura. Os impactos buscados estão na redução do número de parâmetros e no tempo de inferência ao passo que não se deseja que a acurácia sofra perdas maiores do que 5%[11]. A intenção é prover um estudo crítico sobre os efeitos das estratégias adotadas de forma a listar as vantagens e desvantagens de cada uma delas.

[11] Este valor foi arbitrado por considerarmos que perdas maiores poderiam impactar na aplicabilidade prática dos modelos.

Além disso, com os resultados apresentados, esta dissertação pode fornecer um guia para a definição de arquiteturas que sejam mais eficientes em relação a um, ou mais, dos aspectos aqui estudados.

Como base deste estudo foram escolhidas duas redes amplamente utilizadas na literatura: a ResNet e a SqueezeNet, e, como propostas a serem comparadas, estão as convoluções separáveis em profundidade da MobileNet (HOWARD et al., 2017), o módulo de Mistura de Canais da ShuffleNet (ZHANG et al., 2018) e o módulo de Deslocamento de Canais da ShiftNet (WU et al., 2018). Os estudos serão conduzidos nas bases de dados CIFAR 10 e CIFAR 100 (KRIZHEVSKY; HINTON, 2009), as quais são bases de dados amplamente referenciadas na área. Além disso, dada a atenção recebida pelas Redes Generativas Adversárias ou *Generative Adversarial Networks* GANs por parte da comunidade da área, também se escolheu a Pix-2-Pix (ISOLA et al., 2017) para avaliar os efeitos das otimizações. Os experimentos foram conduzidos sobre a base *maps↔satellite* e a avaliação das GANs é realizada através de um estudo perceptual das imagens obtidas. A coleta de respostas para o estudo perceptual foi feito pelo do site <http://www.pesquisadequalidadegan.com.br/>[12].

1.2 OBJETIVOS

Esta dissertação objetiva estudar comparativamente os efeitos causados em uma rede neural profunda por diferentes métodos de otimização de sua arquitetura. Os impactos buscados consideram a redução do número de parâmetros do modelo e o tempo de inferência, ao passo que não se deseja que a taxa de acerto seja consideravelmente impactada.

Para atingir este objetivo consideramos técnicas de otimização na estrutura das redes SqueezeNet, ResNet e Pix-2-Pix a fim de obter redes variantes mais eficientes e com pouca perda em suas taxas de acerto em relação à rede base.

Além disso, almejamos prover um estudo crítico sobre os efeitos das estratégias de otimização adotadas a fim de guiar presentes e futuros desenvolvedores na escolha entre as técnicas de otimização de arquiteturas aqui investigadas.

1.3 PRINCIPAIS CONTRIBUIÇÕES

Esta seção reúne as principais contribuições da autora para a área de otimização de redes neurais convolucionais durante seus estudos:

1. Publicação do artigo SANTOS, A. G.; SOUZA, C. O. de; ZANCHETTIN, C.; MACEDO, D.; OLIVEIRA,A. L.; LUDERMIR, T. Reducing squeezenet storage size with depthwise separable convolutions. In: IEEE.2018 International Joint Conference on Neural Networks(IJCNN). [S.l.], 2018. p. 1–6;

[12] O site foi construído neste trabalho como forma de avaliar os resultados dos experimentos realizados nesta dissertação.

2. Implementação de técnicas de otimização na Pix-2-Pix e nas adaptações da SqueezeNet e da ResNet aos bancos de dados CIFAR; e

3. Criação de uma documentação comparativa entre as técnicas de otimização abordadas a fim de facilitar o desenvolvimento de trabalhos futuros.

1.4 ORGANIZAÇÃO DO DOCUMENTO

O restante deste trabalho é apresentado da seguinte forma:

- O capítulo 2 apresenta uma visão geral sobre o desenvolvimento e difusão das CNNs. Ele se detém sobre as redes ResNet, SqueezeNet e Pix-2-Pix, as quais são descritas detalhadamente;

- O capítulo 3 apresenta uma visão geral sobre os métodos de otimização de arquitetura, os separando em duas vertentes (pré e pós-otimizações). Em seguida as técnicas de convolução separável em profundidade, mistura de canais e deslocamento de canais são apresentadas em detalhes;

- O capítulo 4 apresenta a metodologia experimental proposta para as análises. Ele descreve a metodologia, os bancos de dados que são utilizados, as implementações das técnicas de otimização detalhadas no capítulo 3, as métricas de avaliação e os protocolos de treinamento e de teste;

- O capítulo 5 reúne os resultados obtidos, considerando as análises quantitativas e qualitativas realizadas; e

- O capítulo 6 conclui esta dissertação e discute possíveis trabalhos futuros.

2 REDES NEURAIS PROFUNDAS CONVOLUCIONAIS

Este capítulo tem como objetivo contextualizar o leitor acerca do desenvolvimento e difusão das pesquisas com redes neurais convolucionais ou *Convolutional Neural Networks* (CNNs). Dentre as informações a serem destacadas no capítulo estão as CNNs que serão utilizadas no desenvolvimento do trabalho.

2.1 DESENVOLVIMENTO DAS CNNS

Apesar das redes neurais profundas terem seus conceitos propostos nos anos 60, e em 1975 Fukushima (FUKUSHIMA, 1975) ter apresentado o que pode ser considerada a primeira CNN, este modelo foi evoluído em 1989, em uma arquitetura conhecida como LeNet (SZE et al., 2017). Porém, apenas com a AlexNet que este modelo foi consolidado e sua importância começou a crescer. Hoje as CNNs são a base para variadas aplicações de Inteligência Artificial (IA) e isso se deve a sua habilidade em aprender características de alto nível diretamente dos dados de entrada. Essa habilidade foi consolidada pela AlexNet em 2012 ao ganhar a competição ImageNet (ILSVRC) sem a necessidade de utilizar engenharia de características.

Até então, os modelos de aprendizagem de máquina existentes eram dependentes de profissionais especializados para o desenvolvimento de cálculos de características dos dados em estudo. As características extraídas desses cálculos eram então usadas como entrada para um algoritmo à escolha do desenvolvedor (ex: vizinhos mais próximos ou máquina de vetores de suporte). O problema em depender da extração de características é que muitas delas eram complexas, difíceis de serem criadas, dependiam de um conhecimento profundo sobre os dados e, uma vez desenvolvidas, poderiam ser patenteadas, o que dificultaria o seu uso por parte da comunidade acadêmica. A AlexNet, por sua vez, era um modelo que, por si só, pode aprender a extrair as características necessárias e, a partir delas, realizar a classificação/regressão/agrupamento das informações de entrada. Ou seja, essa rede compreende, em uma única estrutura, o extrator de características e o modelo baseado em aprendizagem e o seu treinamento é feito em um único processo.

Além dessa vantagem frente aos modelos de AM tradicionais, outros fatores foram importantes para o aumento da quantidade de pesquisas com as CNNs. Acredita-se que o sucesso e popularização veio através da confluência de três fatores:

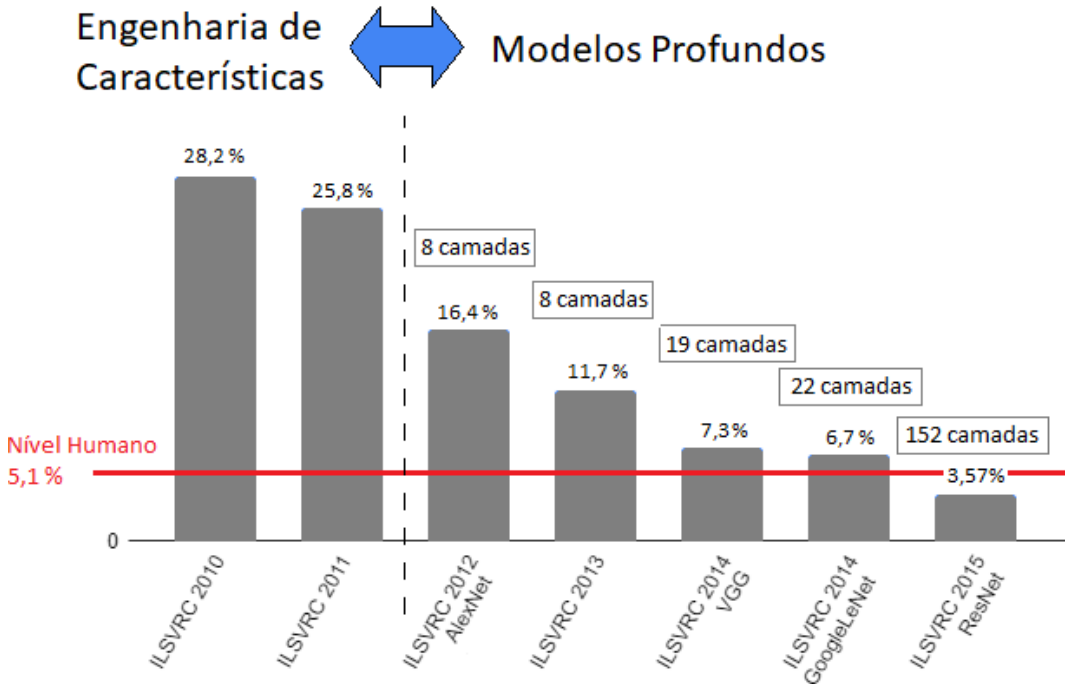
1. O aumento da **quantidade de dados** disponíveis para treinar modelos;
2. A disponibilidade de **capacidade computacional** em novos *hardwares*;
3. O surgimento de diversas ferramentas (abertas em sua maioria) para a implementação dos modelos.

Mais detalhadamente, como as CNNs possuem um quantidade elevada de parâmetros a serem treinados, a quantidade de exemplos de treinamento que são necessários para proporcionar um aprendizado adequado também é superior a modelos que possuem menos parâmetros. Com o surgimento de bancos de dados com uma quantidade considerável de exemplos que poderiam ser utilizados pelas CNN, seu aprendizado passou a ser plausível. Como dito, essas redes possuem mais parâmetros treináveis do que modelos tradicionais e todos eles têm que ser atualizados iterativamente no decorrer do treinamento do modelo. Essa não é uma tarefa impossível para uma CPU tradicional, mas a quantidade enorme de operações necessárias tomaria um tempo tão grande que tornaria proibitiva a tentativa em computadores com poucos núcleos de processamento. O contorno a este problema veio por avanços em *hardware*. Em vez de utilizar a CPU do computador, o treinamento poderia ser realizado com o uso de GPUs que fornecem uma quantidade elevada de núcleos de processamento e memória dedicada. Por fim, o último fator que alavancou as CNNs está no surgimento de diversas bibliotecas e *frameworks* (ex: Caffe, PyTorch e TensorFlow) para a implementação, treinamento e uso de tais algoritmos.

Os modelos mais conhecidos de CNNs são as redes que participaram da competição da ImageNet nos anos que sucederam a AlexNet. Na figura 1 pode-se ver que enquanto os modelos de aprendizagem de máquina que não eram profundos (anos de 2010 e 2011) atingiram até $25,8\%$ de acurácia, a partir de 2012, com as CNNs, essa taxa teve decréscimos significativos. A evolução foi tão rápida que, em 2015, apenas três anos depois, as CNNs superaram a capacidade humana de $\sim 5,1\%$ na tarefa de classificação de objetos da ImageNet (RUSSAKOVSKY et al., 2015) com a ResNet atingindo uma taxa de erro de $3,57\%$. Outras redes que merecem destaque são a VGG (SIMONYAN; ZISSERMAN, 2014) e a GoogleLeNet, ganhadora de 2014 e também chamada de InceptionV1, (SZEGEDY et al., 2015). Ainda da figura 1 é importante acrescentar que, acompanhando a assertividade, no decorrer dos anos, as redes cresceram em número de camadas. Enquanto que em 2012, a vencedora era composta por oito camada convolucionais, em 2015 esse número havia saltado para 152. Esse crescimento demonstra uma tendência entre os pesquisadores de aumentar a rede em busca de melhores resultados.

Desde o início da popularização das CNNs com a AlexNet, muitas aplicações foram propostas tais como: classificação e segmentação de imagem, localização e detecção de objetos, transferência de estilo, reconhecimento de fala, geração de áudio, processamento de linguagem natural, detecção de câncer em imagens médicas, planejamento de ações para robôs, estratégias de direção para carros autônomos, entre outras. Hoje, diversas dessas aplicações estão presentes em nosso cotidiano. Por exemplo: os usuários estão usando de forma transparente modelos capazes de "ler" quando recebem um carta ou quando abrem uma conta no banco e mandam fotos de documentos contendo suas informações, pois os sistemas utilizados tanto no Correios como em diversos bancos são capazes de, a partir de uma imagem, ler as informações que são necessárias para prosseguirem com os

Figura 1 – Resultados da ImageNet entre 2010 e 2015



Fonte: A autora (2019).

processos nos quais estão alocados. Há também sistemas de reconhecimento de face que servem como instrumentos de segurança, como o faceID da Apple[1] e como o utilizado no carnaval de Salvador em 2019. Esse último utiliza imagens de câmeras de segurança detectando faces e verificando a semelhança delas com as faces contidas num banco de dados com informações de criminosos foragidos (OH, 2019). Além disso, aplicativos recentes que permitem que uma pessoa seja "transformada" de homem para mulher e vice-versa ou que envelhecem e que rejuvenescem estão munidos de modelos profundos generativos treinados a partir de Redes Generativas Adversárias ou *Generative Adversarial Networks* (GPUs). Esses modelos são capazes de gerar uma imagem modificando características de outra de acordo com o que foi treinada para realizar. Alguns exemplos dessas aplicações podem ser encontradas em (BROWNLEE, 2019). Inclusive, as GANs são hoje um tema de pesquisa em evidência, tendo Yann LeCun[2] dito "Redes generativas adversárias é a ideia mais interessante dos últimos 10 anos no campo de aprendizagem de máquina"[3].

As plataformas nas quais as aplicações com CNNs estão sendo desenvolvidas e disponibilizadas varia. O treinamento de uma rede normalmente requer uma grande massa de dados de exemplo, juntamente com um grande poder de paralelização de operações tanto no cálculo das ativações quanto na atualização dos pesos e, portanto, normalmente é feito em equipamentos de ponta como GPUs e TPUs, quer eles sejam locais, quer estejam alo-

[1] <https://support.apple.com/pt-br/HT208108>.
[2] Criador da LeNet e hoje Diretor de pesquisa em IA no Facebook - <http://yann.lecun.com/>.
[3] Em tradução livre de "*Generative Adversarial Networks is the most interesting idea in the last 10 years in Machine Learning.*"

cados na nuvem. Já a utilização das mesmas, além de poder ser realizada em máquinas com as mesmas características das máquinas de treinamento, também exigem adaptações para trabalharem em dispositivos como os celulares, *drones*, robôs e também diretamente em sensores. Entretanto, a tendência geral de se desenvolver redes maiores para melhorar o desempenho não necessariamente tem feito com que elas sejam mais eficientes com respeito ao tempo de inferência e ao seu tamanho de armazenamento.

Os desafios de utilizar os modelos de CNNs em plataformas como as já citadas se concentram em três principais fatores limitantes: o tamanho do modelo e a quantidade de memória necessária para seu armazenamento; o uso de bateria; e disponibilidade de unidades de processamento de dados mais simples. Para contornar essas dificuldades é clara a necessidade de modelos cujo espaço de armazenamento esteja de acordo com o espaço disponível no dispositivo alvo; cuja complexidade computacional seja reduzida a fim da inferência demandar menos recursos e menos energia; e cujas operações sejam menos dependentes de paralelização do processamento como o que é realizado nas GPUs.

Com o objetivo de facilitar ou até mesmo permitir o uso de CNNs em dispositivos nos quais as redes tradicionais, e mais robustas, não se mostram uma escolha possível, surgiram duas vertentes de pesquisa. Ambas se debruçam em otimizar as redes neurais de forma que o algoritmo de inferência tenha uma maior eficiência em relação ao tempo de processamento e ao espaço de armazenamento. Entretanto, enquanto a primeira vertente deu origem a metodologias de otimização que são aplicáveis às redes cujo treinamento já tenha sido realizado ou esteja em andamento, a segunda segue a ideia de que, dado um nível desejado de assertividade, é possível definir redes mais eficientes já na construção de sua arquitetura.

Em seguida serão detalhadas as redes que serão utilizadas no desenvolvimento deste trabalho. Mais considerações sobre as estratégias de otimização de CNNs serão apresentadas no capítulo 3.

2.2 MODELO RESNET

A publicação da pesquisa que deu origem à Resnet (HE et al., 2016a) foi feita em 2015, apenas três anos após a apresentação da rede AlexNet e representou uma redução de $\sim 78\%$ da taxa de erro em relação a sua predecessora, a GoogleLeNet (SZEGEDY et al., 2015). Além disso esta rede foi a primeira a obter um nível de assertividade maior do que a capacidade humana no mesmo problema (RUSSAKOVSKY et al., 2015).

2.2.1 Motivação para o modelo

Evidências anteriores à ResNet sugerem a importância do número de camadas de uma rede neural no desempenho do modelo (SIMONYAN; ZISSERMAN, 2014; SZEGEDY et al., 2015). Entretanto, apesar de avanços terem sido obtidos ao se adicionar camadas, a partir

de certo ponto era possível observar que o modelo sofria saturação e, adicionando-se ainda mais camadas, a assertividade do modelo caía (HE et al., 2016a).

O causador da degradação da assertividade de redes com um número elevado de camadas não era mais o desaparecimento do gradiente, visto que esse problema, no advento da publicação da ResNet, já havia sido endereçado por diferentes métodos de inicialização de parâmetros e funções de ativação (HE et al., 2015; HINTON et al., 2012; HE et al., 2016a) e da normalização de camadas intermediárias (normalização por *batch* ou *batch normalization*) (IOFFE; SZEGEDY, 2015). A degradação do desempenho das rede, no entanto, não era causada pelo *overfitting* do modelo, e ocorria ainda durante o processo de treinamento. Essa observação indica que modelos diferentes possuem dificuldades diferentes para serem treinados.

Focando resolver o problema, causador da degradação da assertividade das CNNs, He *et al.* argumentam que aumentar uma rede não deveria piorar os resultados já que poderíamos adicionar camadas cujo mapeamento fosse uma função identidade. Assim, quando propagados por essas camadas, os erros não sofreriam perdas e a rede se comportaria da mesma forma como se essas camadas não existissem. Ou seja, isso indica que as redes com mais camadas não produziriam erros maiores do que suas versões reduzidas.

He *et al.* se basearam na suposição que seria mais fácil as camadas de uma CNN aprenderem um mapeamento residual dos dados do que deixá-las aprenderem o mapeamento direto dos mesmos. Formulando matematicamente, sendo $H(x)$ a função de mapeamento do conjunto de algumas camadas convolucionais e x o exemplo de entrada da primeira dessas convoluções, ao se procurar por uma função que se aproxime do residual $R(x) := H(x) - x$, o mapeamento original se transforma em $H(x) := R(x) + x$. No processo de atualização dos pesos, quando os erros que passam pelo caminho residual ($R(x)$) se perdem (ou desaparecem), eles ainda tem o caminho identidade (x) a seguir. Com essa hipótese e no caso de o mapeamento identidade ser a solução ótima de uma camada, seria mais fácil levar o residual a zero do que treinar um mapeamento identidade entre camadas com funções de ativação não-lineares.

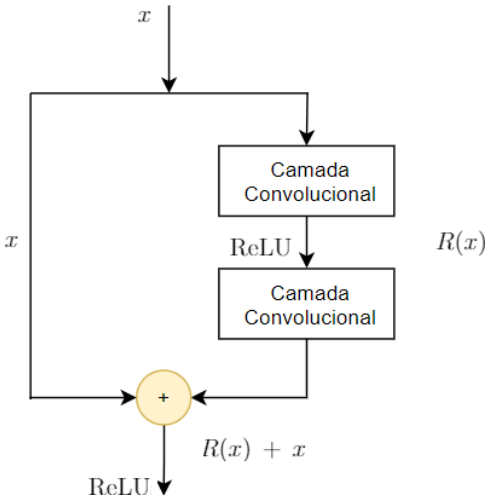
2.2.2 Arquitetura

A arquitetura ResNet é definida em termos de uma macro e de uma micro-arquitetura. Enquanto que a micro-arquitetura é a definição detalhada de uma camada individual ou um módulo, a macro-arquitetura define a organização sistemática de uma rede neural, com suas múltiplas camadas e/ou blocos, como um todo coeso.

O módulo que define a micro-arquitetura da ResNet, chamados de módulos (ou blocos) residuais, traduzem a formulação matemática descrita na seção anterior. Cada bloco pode ser visto como uma rede direta (*feedforward*) com uma conexão de atalho entre a entrada e a saída de acordo com o diagrama representado pela figura 2.

A saída de cada bloco residual se constitui na adição dos mapas de ativação advindos de dois caminhos. O primeiro desses caminhos é o identidade e o segundo é composto por duas camadas convolucionais, cada uma contendo uma operação convolucional e uma operação de normalização dos dados.

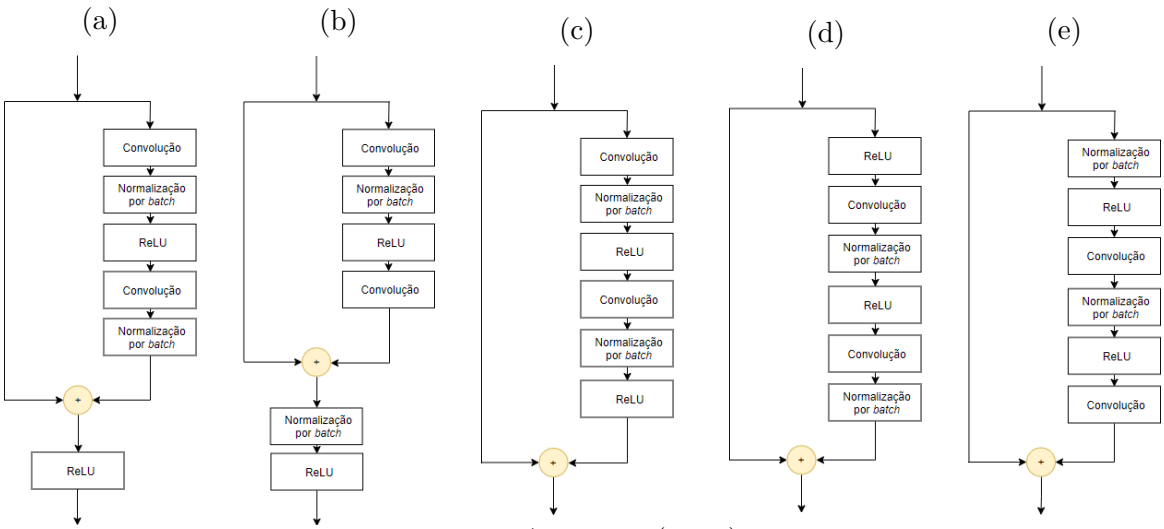
Figura 2 – Bloco residual original da ResNet



Fonte: A autora (2019).

Pouco tempo após a primeira publicação sobre a ResNet, os mesmos autores também publicaram redefinições do bloco básico, ilustrado na figura 2, apenas alterando a ordem dos componentes da camada convolucional (HE et al., 2016b). Essas variações estão ilustradas na figura 3.

Figura 3 – Diagramas detalhados dos blocos residuais (a) original; (b) com normalização após a adição; (c) com ReLU antes da adição; (d) de pré-ativação somente com a ReLU e; (e) de pré-ativação completa



Fonte: A autora (2019).

Além da pré-definição dos módulos que a constituem, a ResNet apresenta um padrão

na forma de interligá-los. A sua macro-arquitetura, portanto, pode ser descrita em três partes:

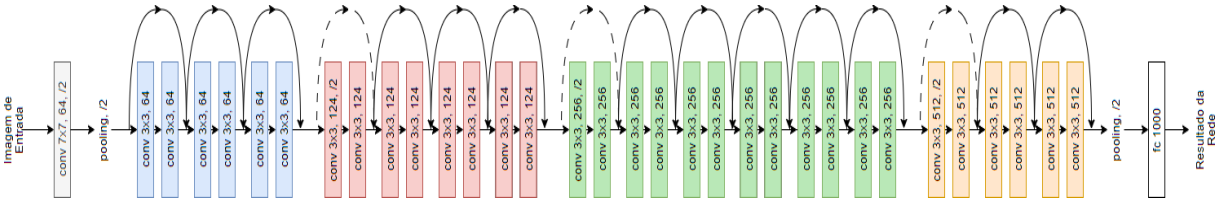
1. uma camada convolucional inicial;
2. quatro conjuntos de blocos residuais;
3. uma camada totalmente conectada ao final;

A primeira parte é detalhada como uma convolução com 64 filtros de núcleo 7×7 e que reduz à metade as dimensões dos dados de entrada. À essa convolução se segue uma camada de *pooling* de núcleo 2×2. A última parte é uma camada totalmente conectada que leva seu mapa de ativação de entrada à dimensão esperada de saída.

Já na segunda parte tem-se quatro conjuntos de módulos residuais. O número desses módulos em cada conjunto é um hiper-parâmetro que define a quantidade de camadas final da rede. Dentro de um mesmo conjunto, todas as convoluções de todos os blocos possuem o mesmo núcleo 3×3 e o mesmo número de filtros. Essa quantidade é de 64 filtros no conjunto inicial e, a cada conjunto seguinte, dobra de valor em relação ao anterior. Assim, os três conjuntos seguintes são formados por convoluções com 124, 256 e 512 filtros, nessa ordem. Excetuando-se o primeiro conjunto, as primeiras convoluções de cada conjunto reduzem pela metade as dimensões de largura e altura e dobram o número de canais dos mapas de características que recebem como entrada. Devido às modificações nas dimensões dos dados, a conexão de atalho que liga mapas de característica de dimensões diferentes realiza uma convolução com núcleo 1×1 e passo 2 a fim de fornecer à operação de adição dados de formato adequado ao mapa de características de saída. Por fim, o último conjunto é seguindo por uma camada de *pooling* com núcleo 2×2.

Na figura 4 está ilustrada a ResNet34, assim chamada por possuir 34 camadas. Ela é composta por quatro conjuntos de blocos residuais compostos por 3, 4, 6 e 3 blocos. Ainda na figura 4, cada conjunto é representado por uma cor diferente, as conexões de atalho que ligam dados com dimensões diferentes estão representadas por linhas tracejadas e "/2" indica que a largura e a altura do mapa de características de entrada é reduzido pela metade ao passar pela convolução.

Figura 4 – Representação da arquitetura da ResNet, tomando como exemplo a ResNet-34.



Fonte: A autora (2019).

2.3 MODELO SQUEEZENET

Publicada em 2016, a proposta da SqueezeNet é ser uma rede com arquitetura enxuta, ao passo que sua acurácia seja comparável a de outra arquitetura já bem conhecida, a AlexNet (IANDOLA et al., 2016a). De fato, Iandola *et al.* alcançam a taxa de acerto da AlexNet com 50x menos parâmetros e ocupando apenas 4,8MB de espaço de armazenamento contra 240MB da primeira rede.

2.3.1 Motivação para o modelo

A concepção da SqueezeNet se deu pela busca de redes mais eficientes com respeito ao número de parâmetros e ao espaço de armazenamento. Iandola *et al.*, sabendo que há uma infinidade de arquiteturas possíveis que podem alcançar desempenhos equivalentes e que, dentre elas, as redes maiores apresentariam também uma maior redundância, buscavam uma arquitetura que fosse mais eficiente. Essa busca foi motivada pelo fato que redes com um menor número de parâmetros e que ocupassem menor espaço em disco teriam vantagens na ocasião de serem utilizadas em aplicações reais. Entre essas vantagens estão:

- **Maior eficiência em treinamentos distribuídos** - Para o treinamento distribuído de CNNs, a carga de comunicação entre os servidores é diretamente proporcional ao número de parâmetros no modelo (IANDOLA et al., 2016b), entretanto, o transporte de dados entre os servidores é limitado pela rede de comunicação. Dessa forma, modelos menores teriam um treinamento mais veloz pois requerem menos comunicação entre os servidores.

- **Menor carga de comunicação ao transferir modelos** - A distribuição e atualização de aplicativos nos celulares ou em carros como os da Tesla são operações frequentes. Entretanto, quando é necessário que uma CNN seja baixada ou atualizada, redes como uma AlexNet requerem a transferência de consideráveis 240MB de dados. Quando esse *download* é esporádico, pode não perturbar o cliente, entretanto, se a frequência aumenta ou o cliente está numa rede com baixa largura de banda, é provável que o processo seja tido como desgastante. Nesse contexto, redes menores e que requerem menor comunicação facilitariam as atualizações frequentes.

- **A possibilidade de realizar a inferência em FPGAs e outros dispositivos embarcados** - Há o interesse de desenvolvedores em utilizar CNNs em produtos que não contam com todo o aparato de um computador. Esses produtos normalmente se baseiam em placas eletrônicas que possuem vetor de portas lógicas programáveis em campo ou *Field Programmable Gate Array* (FPGA) ou um circuito integrado de aplicação específica (ASIC do inglês *application-specific integrated circuit*). Entretanto, uma FPGA, por exemplo, tem normalmente menos de 101MB de memória em seu *chip*, tamanho que requereria o acesso à memória externa para a leitura

dos dados de uma rede como a AlexNet. Uma rede pequena o suficiente para que pudessem ser inseridas diretamente nesses dispositivos permitiriam aplicações mais velozes, visto que estas não estariam limitadas pela velocidade de comunicação com a memória externa (QIU et al., 2016). Além disso, redes menores poderia gerar ASICs com *chips* menores.

Ao focarem numa arquitetura eficiente, os pesquisadores da SqueezeNet se diferenciaram do que estava sendo investigado à época por outros pesquisadores como Han (HAN; MAO; DALLY, 2015; HAN et al., 2015). Han, assim como outros pesquisadores, abordavam aos problemas de implantação de redes neurais profundas se baseando em, a partir de uma rede já treinada, aplicar técnicas de otimização, como a poda e a quantização dos pesos do modelo. Em contrapartida, Iandola *et al.* definiram a arquitetura da SqueezeNet se baseando em heurísticas com o potencial de otimizar a passagem de informações de uma camada para a outra. Dessa forma, e sem a utilização de procedimentos além do treinamento, a SqueezeNet apresentou uma assertividade superior a AlexNet ao passo em que possuía um tamanho de armazenamento em disco 50x menor.

A SqueezeNet, assim como a ResNet, também foi definida em termos de uma macro e de uma micro-arquitetura que seguiam as três estratégias listadas abaixo.

1. Substituir filtros de dimensão 3×3 por filtros de dimensão 1×1 nas camadas convolucionais;

2. Diminuir o número de canais de entrada nas camadas convolucionais cujos filtros tivessem dimensão 3×3 e;

3. Realizar *pooling* tardiamente na arquitetura.

Os dois primeiros itens dizem respeito a reduzir o número de parâmetros da camada. Para demonstrar isso, considere o cálculo do número de parâmetros, P, dado pela equação 2.1, na qual C_{in} representa o número de canais de entrada, K é a dimensão do filtro e C_{out} é o número de filtros da camada. A primeira observação a ser feita é que, caso K seja reduzido de 3 para 1, o número de parâmetro será reduzido por um fator de $9\times$. A segunda observação é que, ao se necessitar de um filtro com $K > 1$, pode-se reduzir o número de canais de entrada para que o número de parâmetros da camada sofra uma redução na mesma proporção. Por exemplo, se temos o número de parâmetros de uma camada dado por P, reduzindo o número de filtros de entrada para um valor $C'_{in} = \frac{C_{in}}{2}$, o novo número de parâmetros obtido seria $P' = K^2 * C'_{in} * C_{out}$ de onde $P' = \frac{P}{2}$.

$$P = C_{in} * K * K * C_{out} \tag{2.1}$$

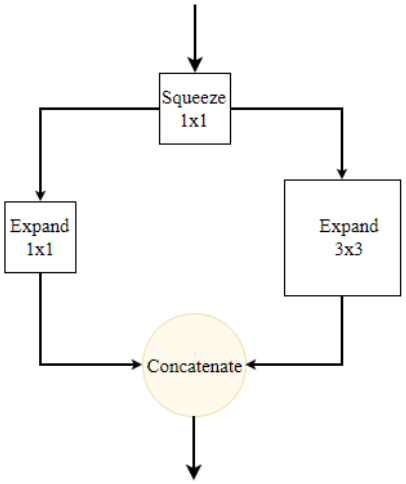
Já o terceiro e último item é uma estratégia para melhorar a assertividade do modelo. Normalmente a amostragem dos dados é feita quando as convoluções são realizadas com

um passo maior do que 1 ou quando adiciona-se uma operação de *pooling*. Caso as convoluções iniciais de uma rede tenham um passo alto, as camada subsequentes receberão um mapa de características reduzido. Caso contrário, se as convoluções com passos diferentes de 1 estejam concentradas mais ao final da rede, os mapas de características que fluem através das camadas serão maiores. A consequência direta observada ao se realizar *pooling* tardiamente é que os mapas de ativação são propagados com mais informações até camadas mais profundas. Sun aplicou essa ideia em diferentes arquiteturas de CNN e, em cada caso, o resultado obtido foi uma melhora no desempenho (HE; SUN, 2015).

2.3.2 Arquitetura

A figura 5 ilustra a estrutura do módulo que define a micro-arquitetura da SqueezeNet, o módulo *fire*. Esse bloco é composto por duas camadas, uma de contração e outra de expansão. A etapa de contração é assim denominada pois tem o papel de reduzir o número de canais do mapa de ativação que será a entrada da etapa de expansão e que, por sua vez, realiza o caminho inverso e aumenta o número de canais do mapa de ativação que será propagado para a camada ou módulo seguinte da rede.

Figura 5 – Diagrama do módulo *fire*



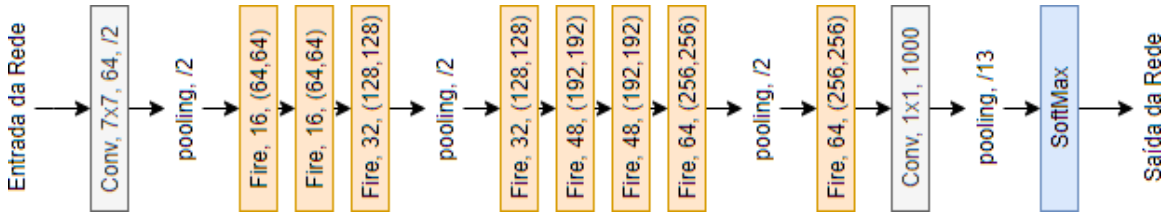
Fonte: A autora (2019).

A etapa de contração é composta por uma operação convolucional cujo núcleo tem dimensão 1×1 e que fornece uma mapa de características em sua saída com menos canais do que o que recebeu por entrada. Já a etapa de expansão se divide em dois fluxos: o primeiro passa por uma camada de convolução com filtros 1×1 e o segundo fluxo tem núcleo de tamanho 3×3. Ambas as operações realizam a expansão do número de canais de acordo com o valor que o projetista queira definir. Em seguida, a etapa de expansão realiza a concatenação das saídas de seus dois fluxos.

A composição da macro-arquitetura da SqueezeNet segue a ilustração da figura 6. Nessa imagem observa-se que a rede possui uma convolução de entrada, seguida de di-

versos módulos *fire*. O último módulo tem sua saída elevada a mil canais através de uma convolução unitária. Por fim, uma operação de *pooling*, seguida pela ativação *SoftMax*, fornece a saída da rede. Tomando-se como exemplo o primeiro bloco *fire* da imagem, lê-se seus parâmetros como a etapa de contração fornecendo 16 canais de saída e a etapa de expansão fornecendo 64 canais de saída em cada uma de suas ramificações. A combinação das ramificações formam a saída do módulo com 128 canais ao todo.

Figura 6 – Representação da arquitetura da SqueezeNet



Fonte: A autora (2019).

Pela composição da SqueezeNet, é possível observar que: as camadas de *pooling* são poucas e a amostragem mais representativa é realizada ao final da rede, mantendo a propagação de informações da imagem de entrada por mais camadas; há mais convoluções cujos núcleos tenham dimensão unitária do que 3×3 e; por último, o módulo *fire* reduz o número de canais do mapa de ativação que é repassado às camadas convolucionais cujo núcleo não é unitário. Essas três observações se correspondem às diretrizes que foram adotadas pelos autores.

2.4 MODELO PIX-2-PIX

A proposta inicial das CNNs era funcionar como um algoritmo de aprendizagem de máquina fim-a-fim. Ou seja, que fosse capaz de realizar a extração de características dos dados e então classificar a entrada. Dessa forma, elas seriam um modelo de extração de características e classificação de dados.

A complexidade de representação dos dados que as CNNs conseguem aprender é maior quanto mais profunda é a camada que fornece essa representação. As camadas iniciais de uma rede aprendem características simples dos dados, como bordas, texturas, linhas ou círculos em imagens, e as camadas mais profundas aprendem representações como um rosto ou um estojo (objetos complexos). Ou seja, as CNNs são capazes de aprender representações probabilísticas dos dados que são cada vez mais complexas de acordo com a quantidade de camadas da rede.

Tal qual redes como a AlexNet, ResNet e SqueezeNet, os trabalhos mais proeminentes envolvendo as CNNs são de modelos discriminativos. Aqueles que a partir de dados, como imagens naturais e sinais de áudio, aprendem representações complexas dos mesmos e indicam a que classe estes pertencem. Não obstante, hoje as aplicações mais populares das CNNs não são os classificadores, mas sim os modelos generativos.

As primeiras tentativas de trabalho com modelos generativos não obtiveram sucesso. Esses trabalhos buscavam uma representação paramétrica da distribuição de probabilidade dos dados em estudo. Essas representações eram obtidas através da estimação da máxima verossimilhança. As primeiras tentativas se depararam com muita dificuldade em aproximar as variadas distribuições de probabilidade que surgiam da estimação de máxima verossimilhança. A fim de contornar essas dificuldades, Ian Goodfellow *et al.* propuseram uma nova metodologia de treinamento (GOODFELLOW et al., 2014).

O processo proposto por Goodfellow *et al.* se baseava em treinar duas redes, uma generativa e outra discriminativa, simultaneamente. Uma das redes, a generativa, tentaria criar imagens que fossem indistinguíveis de imagens reais enquanto que a outra rede, a discriminativa, teria o papel de indicar se uma imagem é verdadeira ou falsa.

O processo de treinamento das duas redes seria análogo à corrida que existe entre os falsificadores de quadro e aqueles que tentam impedir que as obras falsas sejam tidas como verdadeiras. Nessa corrida, enquanto os criminosos estão sempre melhorando suas técnicas de falsificação para que seus quadros sejam cada vez mais fiéis aos originais, os policiais devem conseguir detectar as obras falsificadas. Essa competição leva ambos os lados a melhorarem suas habilidades.

Publicada em 2017, o modelo Pix-2-Pix investiga GANs como uma solução geral para o problema de transformação de imagens em imagens (ISOLA et al., 2017).

2.4.1 Motivação para o modelo

Diversos problemas no processamento de imagem podem ser modelados como a transformação de uma imagem de entrada em uma imagem correspondente de saída e, tradicionalmente, diferentes tarefas são resolvidas por algoritmos específicos. Não obstante, em última análise, transformações realizadas numa imagem se resumem à obtenção de pixeis a partir de pixeis da mesma forma como a tradução de um texto de uma língua para outra transforma palavras em palavras. Ou seja, uma imagem de uma cena pode ser traduzida em um mapa de gradientes, em um mapa de calor, ou em outra representação da cena da mesma forma que se pode traduzir textos entre duas línguas como do português para o inglês.

Atualmente, as CNNs se tornaram o principal recurso na solução de uma grande variedade de problemas de processamento de imagens. Entretanto, apesar do processo de treinamento dessas redes ser automático, ainda é necessário que os projetistas se empenhem no desenvolvimento de funções de perda[4] visto que essas funções são essenciais na aferição da qualidade dos resultados obtidos. Em outras palavras, apesar do treinamento ser automático, ainda é necessário realizar estudos sobre qual é a função de perda adequada para se minimizar o erro do modelo durante seu treinamento.

[4] Funções que permitem avaliar a qualidade das soluções ou do conjunto de pesos do modelo durante seu treinamento.

A complexidade existente na busca pela função de perda específica pode ser contornada se, em vez de definir funções de perda específicas a cada problema, fosse possível defini-las de forma genérica e utilizá-las para diversas tarefas. Dessa forma, o objetivo de Isola *et al.* era a elaboração de uma solução para a tradução de imagens.

Nesse contexto, as GANs fazem exatamente o que se é desejado. Por definição, a rede discriminativa de uma GAN aprende uma função de perda que tenta definir se uma imagem é real ou falsa enquanto que, ao mesmo tempo, um modelo generativo tenta minimizar essa perda. Com essa formação, as GANs podem se adaptar aos dados em estudo e podem ser aplicadas a uma diversidade de tarefas que, de outra forma, necessitariam de diversas funções de perda específicas.

2.4.2 Arquitetura

As funções de perda utilizadas no desenvolvimento de diversos modelos de processamento de imagem são normalmente formuladas como uma função para classificação (ou regressão) pixel a pixel (ISOLA et al., 2017). Entretanto, tais funções tratam os pixeis de uma imagem como dados sem ligações estruturais com seus vizinhos. Com o objetivo de levar em consideração a importância da vizinhança do pixel no cálculo de sua perda, a Pix-2-Pix foi concebida como uma GAN condicional, ou *c*GAN.

As *c*GANs são um gênero específico de GANs, assim chamadas pois o valor da perda de um pixel é condicionada aos valores dos pixeis no entorno desse pixel de saída. Isola *et al.* não foram os primeiros a utilizarem *c*GANs (ISOLA et al., 2017). Entretanto há duas diferenças essenciais entre as publicações anteriores e a Pix-2-Pix:

- Cada um desses trabalhos foi idealizado para a solução de uma tarefa específica, enquanto que a Pix-2-Pix é idealizada para ser um modelo de propósito geral;
- As arquiteturas das redes generativas e discriminativas são diferentes.

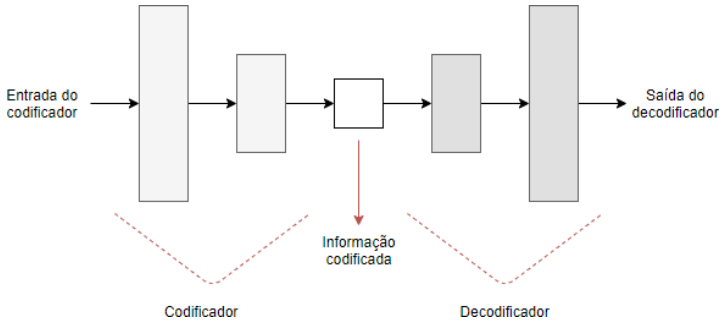
Em relação ao segundo item, diferentemente de trabalhos anteriores, a rede generativa da Pix-2-Pix é baseada na arquitetura da UNet (RONNEBERGER; FISCHER; BROX, 2015) enquanto que para a rede discriminativa os autores utilizam uma arquitetura similar à proposta pela "Patch-GAN" (LI; WAND, 2016). Ambas as arquiteturas são detalhadas a seguir.

2.4.2.1 Rede Generativa

Diversos trabalhos sobre transformação de imagem em imagem utilizavam modelos com a estrutura codificador-decodificador, ilustrada na figura 7. Numa rede com essa estrutura, a cada camada a entrada é representada por um número menor de dados até uma camada de estrangulamento a partir da qual o processo é revertido e os dados,

progressivamente, voltam às dimensões originais de forma que as informações sobre a entrada devem ser recuperadas com o mínimo de perda.

Figura 7 – Diagrama genérico da estrutura de um Codificador-Decodificador

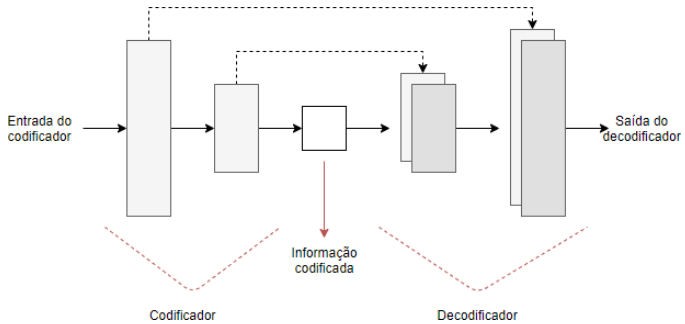


Fonte: A autora (2019).

No caso de trabalhos de transformação de imagens, há uma grande quantidade de informações compartilhadas entre a entrada e a saída do processo. Por exemplo, quando uma foto de satélite é transformada num mapa, a estrutura das ruas são mantidas. A UNet (RONNEBERGER; FISCHER; BROX, 2015) tem como contribuição a introdução de conexões entre as etapas de codificação e decodificação. Essa última, ao receber informações da primeira, assegura que as características aprendidas na contração dos dados, em vez de possivelmente se perderem durante o estrangulamento da rede, serão utilizadas na reconstrução da imagem.

Assim sendo, para facilitar a passagem de informações e fazer com que a estrutura da imagem fosse mantida na reconstrução, Isola *et al.* utilizam uma rede com a estrutura da UNet como rede generativa. A figura 8 e a figura 9 ilustram um codificador-decodificar com conexões de atalho e a rede UNet da Pix-2-Pix, respectivamente.

Figura 8 – Diagrama genérico da estrutura de um Codificador-Decodificador com conexões de atalho

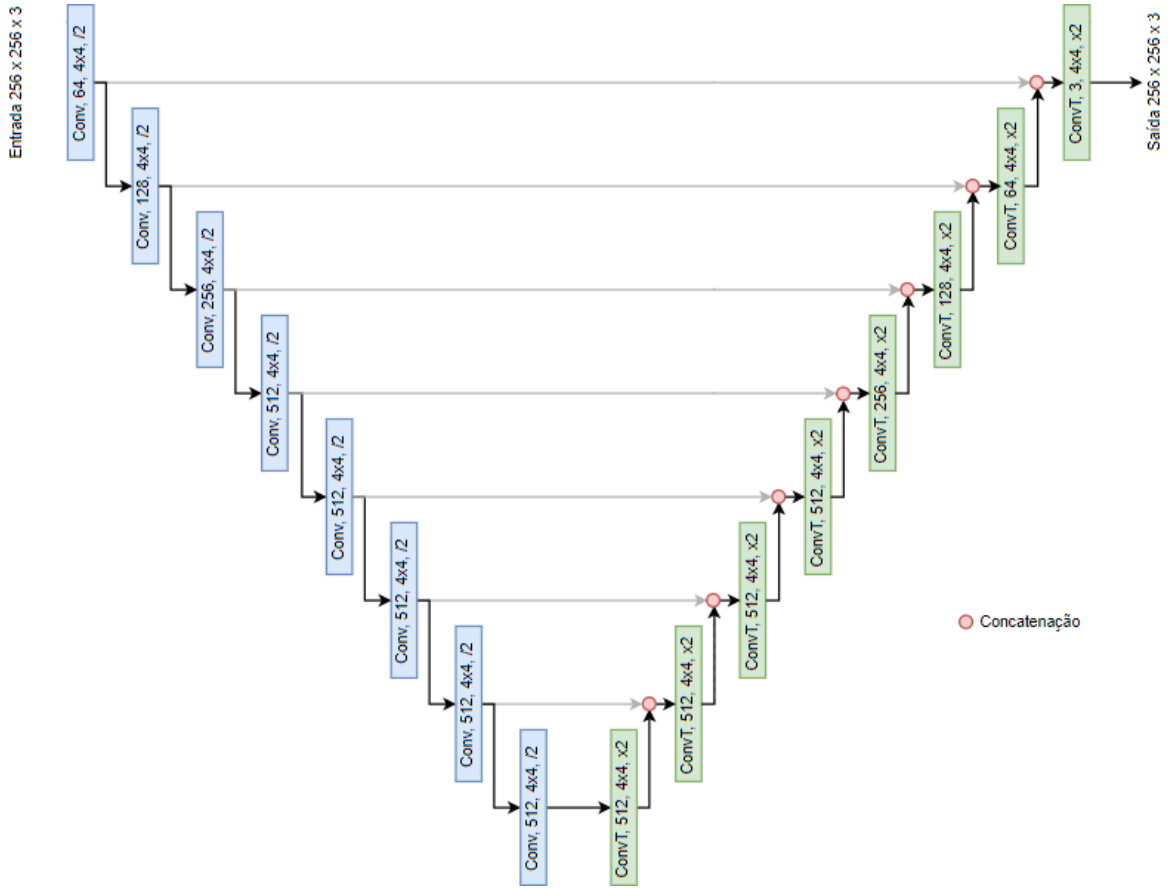


Fonte: A autora (2019).

2.4.2.2 Rede Discriminativa

Resultados apresentados por Isola *et al.* demonstram, assim como em (LARSEN et al., 2015), que o uso das perdas $L1$ e $L2$ resultam em imagens borradas. Em outras palavras,

Figura 9 – Representação da arquitetura UNet da Pix-2-Pix.



Fonte: A autora (2019).

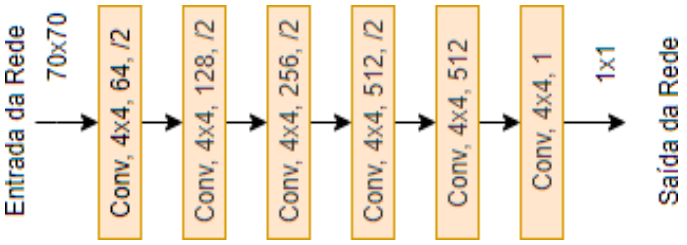
essas perdas podem fornecer bons resultados ao modelarem baixas frequências enquanto que não são efetivas para as altas frequências. Ao se buscar a reconstrução de imagens, no entanto, é necessário que ambos os espectros sejam mantidos afim de que imagens mais nítidas sejam geradas.

Em vez de criar uma nova função de perda que fosse capaz de simultaneamente modular altas e baixas frequências, Isola *et al.* propuseram utilizar o bom desempenho da perda $L1$ na modelagem das baixas frequências enquanto que a modelagem das altas seria feita restringindo-se a entrada da rede discriminativa a pequenas partes da imagem total. Dessa forma, a rede discriminativa proposta, a PatchGAN, tenta definir se cada *patch*[5] de uma imagem é real ou falso. Para a definição sobre a imagem total, o discriminador é aplicado de forma convolucional e a média das respostas é tomada como a resposta do discriminador.

A arquitetura da PatchGAN é composta por cinco camadas convolucionais. Ela recebe uma parte de dimensões 70×70 da imagem completa e resulta se aquela pequena parte é de uma imagem verdadeira ou falsa. A figura 10 ilustra a arquitetura da rede discriminativa da Pix-2-Pix.

[5] No presente contexto, um *patch* é um pedaço da imagem que se deseja classificar como real ou sintética.

Figura 10 – Representação da arquitetura da PatchGAN



Fonte: A autora (2019).

3 OTIMIZAÇÃO DE ARQUITETURAS DE REDES NEURAIS CONVOLUCIONAIS

Este capítulo tem como objetivo contextualizar o leitor sobre do desenvolvimento e difusão das pesquisas na área de otimização das arquiteturas das CNNs. Dentre as informações a serem destacadas no capítulo podem ser pontuados os métodos de otimização que serão utilizados no desenvolvimento do trabalho.

No presente trabalho vale ressaltar o sentido do termo "otimização". Quando se fala em otimização de redes neurais é comum pensar nos otimizadores de treinamento, no entanto, no presente trabalho, o termo "otimização"é associado à otimização de arquiteturas.

3.1 MÉTODOS DE OTIMIZAÇÃO DE ARQUITETURAS

No início da popularização das CNNs, não era nítida ou enfatizada a necessidade de se construir arquiteturas que pudessem ser utilizadas em dispositivos com restrição de processamento como os celulares, por exemplo. As atenções da comunidade de pesquisa se voltavam para contornar problemas no treinamento das CNNs. Dentre as dificuldades existentes para um treinamento bem sucedido de uma rede profunda, podemos citar três principais: a necessidade de possuir uma base de dados com quantidade de exemplos suficiente para o número de parâmetros, ou de pesos da rede, que deveriam ser treinados (BENGIO et al., 2007; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SZE et al., 2017); o elevado custo computacional para realização da retro-propagação do erro, o qual requeria computadores com grande poder computacional, a fim de que o treinamento pudesse ser realizado em um tempo adequado (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SZE et al., 2017) e; por fim, ainda existia o problema do desaparecimento do gradiente, impossibilitando o treinamento de redes muito profundas (GLOROT; BENGIO, 2010; HE et al., 2016a; IOFFE; SZEGEDY, 2015).

Com o desenvolvimento de técnicas que contornaram esses obstáculos (DENG et al., 2009; KRIZHEVSKY; HINTON, 2009; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; NAIR; HINTON, 2010; IOFFE; SZEGEDY, 2015), as CNNs se tornaram o principal modelo na abordagem de diversos problemas de visão computacional, processamento de linguagem natural e outros domínios (LECUN; BENGIO; HINTON, 2015; SZE et al., 2017). Com sua popularização e o desenvolvimento de inúmeras funcionalidades (LECUN; BENGIO; HINTON, 2015), o próximo passo naturalmente seria implantação de CNNs como ferramentas em projetos que envolvessem o reconhecimento de fala[1] (HINTON et al., 2012), diagnósticos na área da saúde (WANG et al., 2016; FAUST et al., 2018; ESTEVA et al., 2017) e na detecção de fraudes em sistemas bancários (ROY et al., 2018) e em redes de comunicação (THING, 2017).

[1] presente na Siri da Apple e na Cortana da Microsoft.

Entretanto, em aplicações práticas, muitos dispositivos de *hardware* se mostraram inadequados para utilizar esse tipo de modelo. Enquanto que em laboratórios de pesquisa estão disponíveis computadores com grande poder computacional, equipados com GPUs, com alta capacidade de armazenamento de dados e que cuja alimentação energética não é restringida por uma bateria, fora desse ambiente estas máximas não são verdadeiras e é importante observar quaisquer limitações que se façam presentes nos dispositivos. Em diversos sistemas embarcados a memória disponível para o funcionamento do sistema é pequena e não é possível aumentá-la; também os processadores são normalmente menos potentes; é fácil encontrar dispositivos cujo pleno funcionamento deve ser suportado apenas por baterias e; por último, há aplicações com requisitos de tempo de inferência que devem ser obedecidos, seja pela segurança, seja pela qualidade da experiência do usuário ou por outros requisitos do processo. Dessa forma, pode-se definir um grupo requisitos com os principais aspectos para otimização desses modelos:

1. Restringir número de parâmetros e a complexidade computacional;
2. Reduzir o espaço de armazenamento dos modelos;
3. Restringir a energia necessária para realizar uma inferência do modelo; e
4. Restringir o tempo de inferência;

Com foco nesses aspectos, há duas vertentes de estudos sobre a otimização da arquitetura de CNNs. A primeira das vertentes se preocupa com técnicas que são aplicadas durante ou após o treinamento do modelo e serão aqui referenciadas como pós-otimizações. Já a segunda vertente, cujas técnicas são analogamente chamadas de pré-otimizações, se preocupa em conceber arquiteturas que, com menos recursos, obtenham resultados competitivos em comparação àqueles alcançados por redes mais robustas.

3.1.1 Técnicas de pós-otimização

As técnicas de pós-otimização são aplicadas a uma rede após a definição inicial de sua arquitetura. Dado que um fluxo tradicional de pesquisa pode ser simplificado em três fases: inicialmente se define um problema, depois se analisa diferentes soluções e, por fim, busca-se otimizações da solução proposta. Quando são objetos de estudo, uma CNN e sua arquitetura são definidas na segunda das três fases anteriormente listadas. Assim sendo, quando se discorre sobre pós-otimizações, a etapa à qual a aplicação das técnicas está atrelado é a última dessas mesmas três fases. Como tal, as técnicas são aplicadas após o processo de treinamento ou durante uma etapa de ajuste fino do modelo, podendo ou não alterar a arquitetura inicialmente definida pelo projetista.

Como metodologias de pós-otimização, podemos listar: Poda (HAN et al., 2015; HAN; MAO; DALLY, 2015); Quantização (HUBARA et al., 2016; ZHOU et al., 2017; WU et al., 2016;

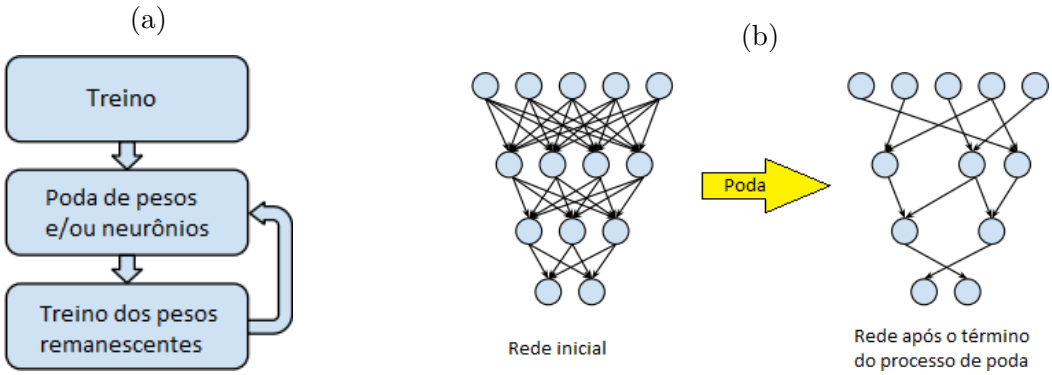
HAN; MAO; DALLY, 2015; GONG et al., 2014); Fatoração (JADERBERG; VEDALDI; ZISSERMAN, 2014; JIN; DUNDAR; CULURCIELLO, 2014); e métodos que juntam dois ou mais das técnicas anteriores (HAN; MAO; DALLY, 2015).

A poda de conexões ou nodos não é uma ideia recente no contexto das redes neurais artificiais. Ela remonta à 1990, quando LeCun publicou (LECUN; DENKER; SOLLA, 1990). As RNAs treinadas nesse período poderiam possuir uma grande quantidade de parâmetros, muitos dos quais redundantes e que poderiam ser eliminados da arquitetura da rede sem prejuízo para seu desempenho. Essa característica se manteve quando as redes neurais evoluíram para arquiteturas profundas. Apesar dos dados de entrada tratados terem se tornado mais complexos, o crescimento considerável no número de parâmetros manteve a redundância nas informações armazenadas pelos pesos das redes. Assim sendo, uma CNN também pode passar pelo processo de poda a fim de eliminar conexões desnecessárias.

A lógica do processo de poda segue a ideia de que se pudéssemos agrupar os neurônios de uma rede de acordo com a sua contribuição e então eliminar os menos importantes, poderíamos obter uma rede mais enxuta, mais veloz e sem perda de desempenho. A maneira com que se pode ordenar a importância dos neurônios pode ser de acordo com: o valor da função de perda, por exemplo $L1$ ou $L2$; o valor da ativação média; ou outro meio criativo como o número de vezes que, em dado conjunto de imagens, o valor de saída de um neurônio não foi anulado. A remoção dos pesos não importantes, naturalmente poderá ser responsável por uma queda na acurácia do modelo. Entretanto, caso a poda seja feita de uma maneira adequada à natureza dos pesos e também do problema atacado pela rede, essa queda pode ser pequena, e eventualmente desprezível. Além disso, para ajudar na recuperação da taxa de acerto, geralmente é realizado um curto treinamento de ajuste dos pesos remanescentes. Assim sendo, na prática, a poda é uma técnica iterativa na qual o processo que realiza a remoção dos pesos e um novo treinamento se repete até o ponto no qual o projetista não consegue mais recuperar as perdas decorrentes da última iteração de poda. Na figura 11 estão representados o fluxo iterativo do processo de poda assim como o efeito causado pelo processo nas conexões e nodos da rede.

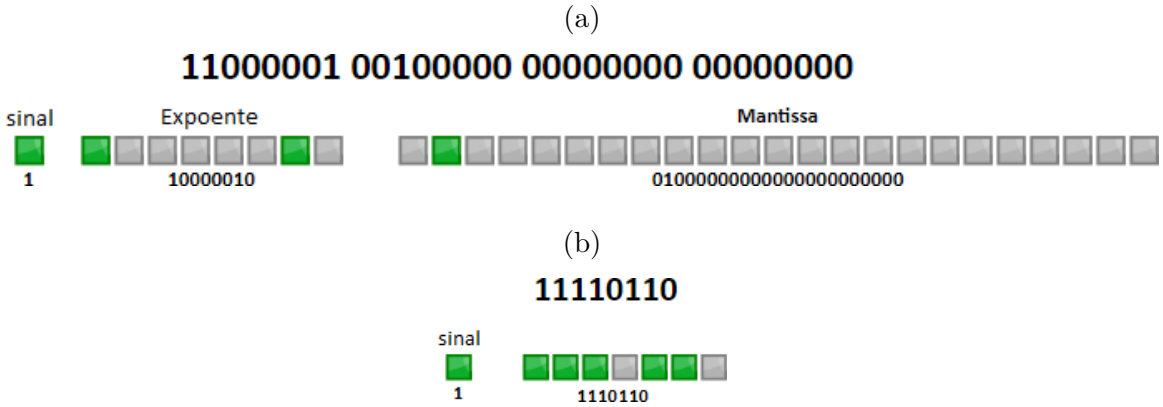
Diferentemente da poda, que tenta diminuir a quantidade de parâmetros de uma rede, a quantização é um processo que modifica a representação dos valores dos parâmetros ou pesos do modelo. Uma rede é normalmente constituída por pesos e mapas de ativação cujos valores, dependendo da arquitetura de hardware, normalmente são expressos no formato de *float32*. Quando falamos em quantizar uma rede, o que o processo de otimização fará é mudar esse formato de dado para outro que necessite de uma quantidade menor de bits para representar os valores dos pesos e dos mapas de ativação da arquitetura em questão. A representação escolhida para os dados depende do dispositivo no qual a rede será implantada. Por exemplo, um computador de uso pessoal trabalha normalmente com representação *float32*, entretanto, como ela não é diretamente suportada em FPGAs, o uso de dados no formato *int8* seria mais adequado. Na figura 12 estão representados a

Figura 11 – (a) Representação do fluxo iterativo do processo de poda;
(b) Representação dos nodos e conexões de uma rede antes e após o término do processo de poda.



Fonte: Imagens retiradas e adaptadas de (HAN et al., 2015).

Figura 12 – Representação do número -10 nos formatos (a) *float32*; e (b) *int8*



Fonte: A autora (2019).

formação do número -10 nos formatos *float32* e *int8*.

Por fim, a técnica de fatoração é uma maneira de alterar a representação das matrizes de pesos de uma CNN. Essa técnica é um artifício matemático que decompõe uma matriz, M, em outras matrizes, suponha A e B, de menores dimensões. Por exemplo, uma camada de uma rede com saída N atribuída a um dado X pela fórmula $N = MX$, em sua versão fatorada deve fornecer uma saída $N' \approx N$ tal que $N' = ABX$. Numericamente, uma matriz de pesos $M_{3\times3}$ pode ser fatorada em duas matrizes $A_{3\times1}$ e $B_{1\times3}$ de acordo com os cálculos das equações abaixo.

$$N = MX = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$N' = ABX = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

A inferência de redes profundas requer a realização de milhares de operações de adição e multiplicação. Quando se utiliza técnicas de otimização os efeitos alcançados geralmente apresentam vantagens como: menor espaço de armazenamento; um menor custo computacional; menos acesso à memória; menor latência da rede; e menor gasto energético.

Quanto menos parâmetros ou bits são necessários para representar a rede, menos espaço ela ocupa na memória do dispositivo. Isso leva a uma menor movimentação de dados da memória para o processamento (e vice-versa), reduzindo o consumo de energia e o tempo de inferência da arquitetura.

3.1.2 Técnicas de pré-otimização

As técnicas de pré-otimização podem ser utilizadas desde a concepção da arquitetura de uma rede neural artificial. Tomando como base as três fases de pesquisa descritas na sessão anterior, estas técnicas de otimização já são consideradas na proposta e análise das diferentes soluções, podendo ainda serem seguidas de outros processos de melhorias.

Existem redes que por definição são mais eficientes do que outras. Como exemplo temos a GoogleLeNet (SZEGEDY et al., 2015), ou InceptionV1; a SqueezeNet (IANDOLA et al., 2016a), já detalhada na sessão 2.3; a MobileNet (HOWARD et al., 2017); a ShuffleNet (ZHANG et al., 2018); a ShiftNet (WU et al., 2018); a ResNet com módulos de estrangulamento, ou *bottleneck units*, (HE et al., 2015; HE et al., 2016a; HE et al., 2016b); entre outras. Isso se deve ao fato que, na elaboração de suas arquiteturas, foram aplicadas diretrizes e técnicas como:

1. Contração e expansão dos mapa de características;
2. Fatoração das convoluções;
3. Convoluções separáveis em profundidade;
4. Mistura de canais; e
5. Deslocamento de canais;

O item 1 é um módulo que reduz o número de canais dos dados de entrada de uma convolução espacial. Em outras palavras, é um bloco que em vez de passar os dados de entrada diretamente a uma convolução espacial, realiza uma operação que reduz o número de canais da entrada e então realiza a convolução espacial. Uma vez que o custo computacional e a quantidade de parâmetros estão concentrados nas convoluções, esse

artifício permite a construção de camadas mais eficientes. A ideia de contração e expansão é utilizada no módulo *fire* da SqueezeNet(HU; SHEN; SUN, 2018) e nos módulos de estrangulamento, ou *bottleneck*, da ResNet (HE et al., 2016a) e da Inception (SZEGEDY et al., 2016).

O item 2 foi introduzido pela GoogleLeNet (SZEGEDY et al., 2015) e posteriormente utilizado em outras redes como a SqueezeNet (IANDOLA et al., 2016a). A ideia de fatorar as convoluções, diferentemente do que é feito nas pós-otimizações, é dividir o fluxo de um módulo para que ele seja processado por filtros com núcleos, ou *kernels*, de diferentes tamanhos. Ou seja, se inicialmente há uma convolução cujo filtro tem tamanho $K = 7$, podemos dividi-lá em operações distintas com *kernels* de tamanhos $K_1 = 5$, $K_2 = 3$ e $K_3 = 1$. A vantagem em se dividir o fluxo de dados para agregá-los em seguida é que cada convolução irá extrair diferentes características que, ao serem agregadas, são mais representativas do que utilizar um único filtro de maior tamanho. Em termos de número de parâmetros, P, uma convolução com $P = K * K * M * N$ e $K = 7$, ao ser separada em duas convoluções cada uma com $K_1 = 1$ e $K_2 = 3$, corresponde ao novo número de parâmetros $P_{fatorizado} = M * N * (K_1^2 + K_2^2)$. A diferença entre as duas quantidades é $\Delta P = M * N * 39$.

Por serem utilizados no desenvolvimento do presente trabalho, os demais itens da lista, são descritos de forma detalhada nas seções seguintes.

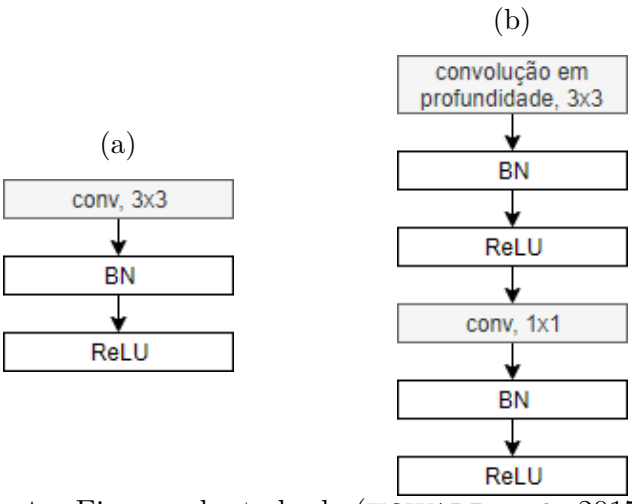
3.2 CONVOLUÇÕES SEPARÁVEIS EM PROFUNDIDADE

As DSCs foram inicialmente publicadas em (SIFRE; MALLAT, 2014) e posteriormente utilizadas no desenvolvimento do módulo *Inception* (IOFFE; SZEGEDY, 2015) antes de serem adotadas pela MobileNet (HOWARD et al., 2017), que leva essa variação da camada convolucional como a sua principal estrutura e característica. Diferente do que é feito por uma camada convolucional padrão que realiza a combinação dos dados de entrada para fornecer os dados de saída e os filtra em um único passo, a DSC realiza essa operação em dois passos. O primeiro deles é uma convolução em profundidade, na qual cada canal de entrada passa separadamente por um filtro, e o segundo passo é uma convolução padrão cuja dimensão do *kernel* do filtro é $K = 1$.

Pode-se comparar uma convolução padrão e uma DSC na figura 13. Os filtros de uma convolução padrão, de uma convolução em profundidade e de uma convolução pontual estão ilustrados na figura 14.

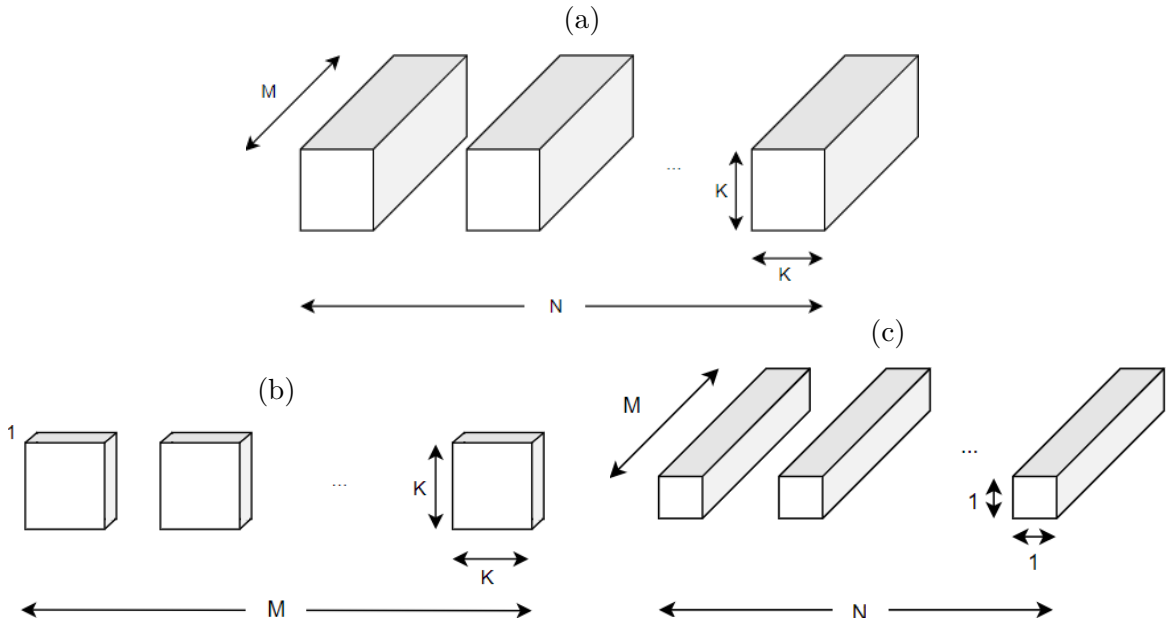
Formulando matematicamente, uma convolução padrão tem como entrada uma imagem, ou um mapa de características, de tamanho $F * F * M$ e como saída um mapa de características de tamanho $G * G * N$ no qual F e G representam tanto a largura quanto a altura dos dados de entrada e de saída, respectivamente, enquanto que M e N indicam o número de canais de entrada e de saída. Já os filtros que compõem essa camada são definidos em torno de seu tamanho. A largura e a altura do núcleo de um filtro são dados

Figura 13 – Diagrama de uma camada convolucional (a) padrão e (b) separável em profundidade.



Fonte: Figura adaptada de (HOWARD et al., 2017).

Figura 14 – Representação dos filtros de (a) uma convolução padrão; (b) uma convolução em profundidade; e (c) de uma convolução pontual.



Fonte: Figura adaptada de (HOWARD et al., 2017).

por K enquanto que a profundidade de cada um deve ter o mesmo valor do número de canais dos dados de entrada, M. Por fim, o número de filtros existentes na camada define o número de canais dos dados de saída. Em outras palavras, se os dados de saída tem N canais, isso se deve ao fato que há N filtros convolucionais na camada em estudo.

Como já mencionado anteriormente, as DSCs são, na verdade dois blocos convolucionais. A primeira delas é uma convolução em profundidade na qual é aplicado um único filtro a cada canal de entrada. A segunda delas é uma convolução padrão cujo *kernel* dos filtros tem tamanho $K = 1$ e que pode ser chamada de convolução pontual. As convolu-

ções em profundidade são extremamente eficientes quando comparadas com a convolução padrão, entretanto, como ela não realiza a combinação das características provenientes dos canais de entrada, é necessário uma camada adicional (no caso a convolução pontual) para realizar a combinação das características através dos canais.

Os cálculos sobre o número de parâmetros e de FLOPs demonstrados a seguir exemplificam os ganhos proporcionados pela substituição de uma convolução padrão por uma DSC.

As equações abaixo sugerem a redução do custo computacional da operação, ou o número FLOPs entre uma convolução padrão e uma DSC que pode ser medido pela razão entre o custo individual de cada uma dessas operações. Sendo $F = G$, o custo computacional de uma convolução padrão dado pela equação 3.1, o custo de uma convolução separável em profundidade dado pela equação 3.2, a redução alcançada pela substituição direta das duas operações é dada pela equação 3.3.

$$FLOP = K * K * M * N * F * F \tag{3.1}$$

$$FLOP_{DSC} = K * K * M * F * F + M * N * F * F \tag{3.2}$$

$$\frac{FLOP_{DSC}}{FLOP} = \frac{K * K * M * F * F + M * N * F * F}{K * K * M * N * F * F}$$

$$\frac{FLOP_{DSC}}{FLOP} = \frac{1}{N} + \frac{1}{K^2} \tag{3.3}$$

Similarmente, o cálculo que indica a redução do número de parâmetros, P, entre as duas variações de camadas convolucionais pode ser medido pela razão entre o número de parâmetros de cada uma das convoluções. Sendo o número de parâmetros de uma convolução padrão dado pela equação 3.4 e os parâmetros de uma convolução separável em profundidade dado pela equação 3.5, a redução alcançada pela substituição direta das duas camadas é dada pela equação 3.6.

$$P = K * K * M * N \tag{3.4}$$

$$P_{DSC} = K * K * M + M * N \tag{3.5}$$

$$\frac{P_{DSC}}{P} = \frac{K * K * M + M * N}{K * K * M * N}$$

$$\frac{P_{DSC}}{P} = \frac{1}{N} + \frac{1}{K^2} \tag{3.6}$$

É possível notar a relação direta do número de parâmetros com a quantidade de FLOPs no cálculo de uma convolução. Como exemplo, se temos uma camada com núcleo de tamanho $K = 3$ e canais de saída $N = 128$, a sua versão otimizada terá um tamanho menor de cerca de $8,41\times$, seja considerando operações ou parâmetros.

3.3 MISTURA DE CANAIS

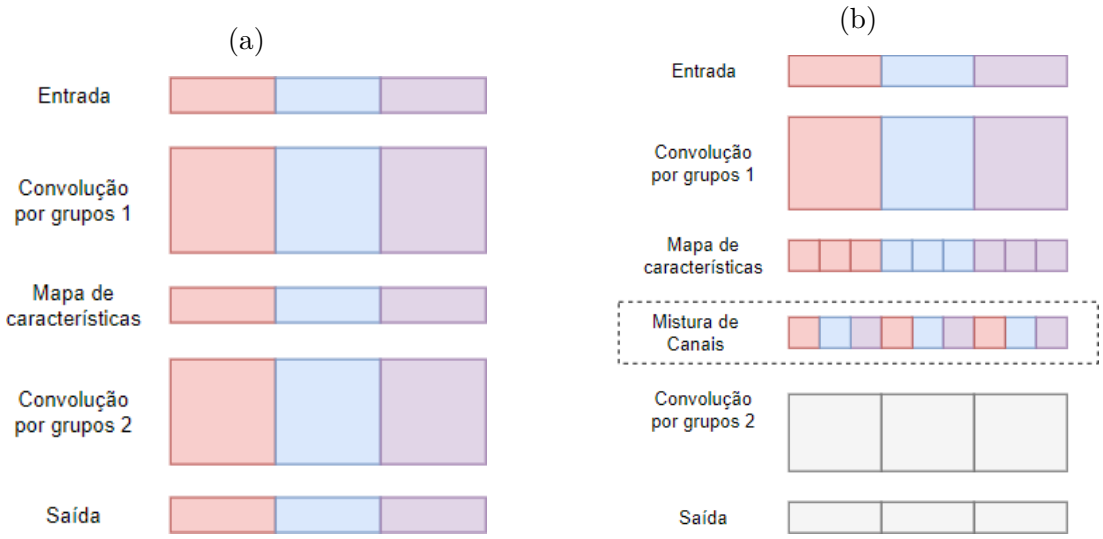
Da mesma maneira que as redes ResNet e SqueezeNet que são construídas a partir da repetição de módulos, outras arquiteturas do estado-da-arte também possuem blocos próprios. A Xception (CHOLLET, 2017) e a ResNeXt (XIE et al., 2017), por exemplo, se baseiam em DSCs e em convoluções por grupo para atingirem uma boa relação entre a capacidade de representação das informações e o custo computacional. Não obstante, essas redes não consideram a quantidade de convoluções pontuais que as constituem e que, apesar de econômicas, representam uma parte considerável das FLOPs de suas micro-arquiteturas. Na ResNeXt, por exemplo, para cada unidade residual, 93,4% das operações se devem à convolução pontual (ZHANG et al., 2018). Em redes pequenas, o alto custo computacional associado às convoluções pontuais induz ao uso de menos canais para atingir a complexidade desejada. A redução do número de canais na representação das informações que fluem pela rede, por sua vez, pode influenciar negativamente a taxa de acerto do modelo.

Zhang *et al.* argumentam que uma solução direta para a redução do custo computacional associado às convoluções pontuais é utilizar o agrupamento de canais. Na figura 15a, na qual está representa uma rede com duas camadas convolucionais por grupos, cada agrupamento de canais é destacado por uma cor diferente. Na imagem, uma entrada tem seus canais divididos em três grupos que passam por filtros distintos e geram um mapa de características no qual as informações de um grupo não são compartilhadas com os demais. É importante observar que ao se utilizar o agrupamento, há um bloqueio no compartilhamento de informações entre canais que pode enfraquecer as representações obtidas e, consequentemente, ter efeitos negativos sobre a acurácia da rede profunda.

Para permitir a troca de informações entre os diferentes grupos, importante para a formação de mapas de características representativos, Zhang *et al.* propõe uma operação de mistura de canais. Na figura 15b, acrescenta-se a mistura de canais à estrutura da rede da figura 15a. O efeito causado pela mistura é redistribuir a ordem de todos os canais, ignorando os agrupamentos, de forma que a informação antes presentes apenas em um dos agrupamentos agora pode ser aproveitada na codificação das informações de outros grupos.

Uma importante característica da operação de mistura de canais, além de permitir o fluxo de informações entre os agrupamentos de canais, é que ela é diferenciável. Ou seja, ela pode ser acrescentada à arquitetura de uma rede profunda e não será incompatível com o treinamento por *backpropagation*.

Figura 15 – Diagrama de uma rede com (a) duas camadas convolucionais por grupos; e (b) duas camadas convolucionais por grupos intercaladas por uma operação de mistura de canais.



Fonte: Imagens retiradas e adaptadas de (ZHANG et al., 2018).

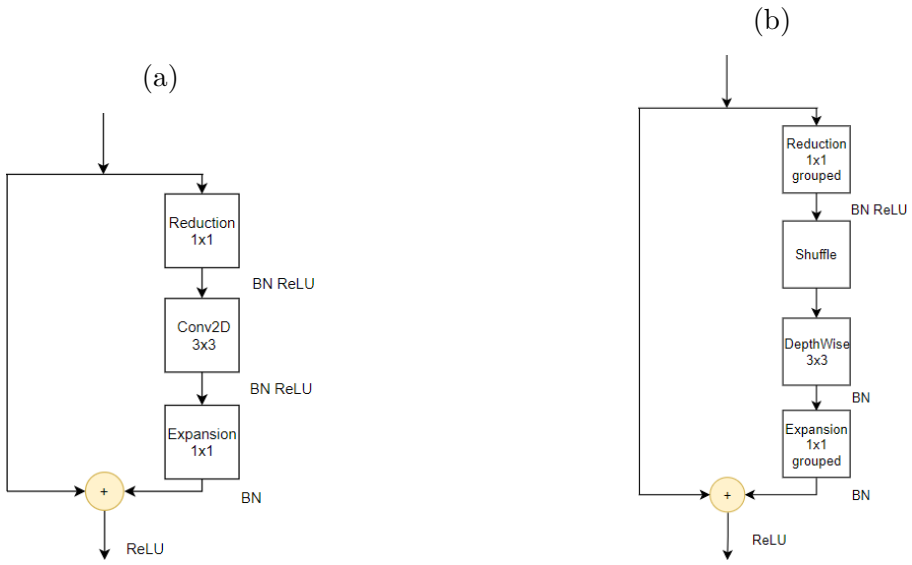
3.3.1 Módulo de mistura de canais

Da mesma maneira que a ResNet define seus módulos construtores em (HE et al., 2016a; HE et al., 2016b), em (ZHANG et al., 2018) se define a **unidade de mistura de canais**, ou *Shuffle Unit*. Essa unidade se baseia no módulo *bottleneck* da ResNet (HE et al., 2016a), ilustrada pela figura 16a. Nesse bloco, o braço que modula o resíduo dos dados é composta por duas convoluções pontuais, das quais a primeira reduz o número de canais do mapa de características da entrada enquanto que a segunda restaura essa quantidade. Entre as duas convoluções pontuais, há uma convolução padrão. A *Shuffle Unit*, ilustrada pela figura 16b é obtida ao substituir as convoluções pontuais por convoluções pontuais agrupadas, a convolução padrão por uma convolução em profundidade e ao adicionar, antes da convolução em profundidade, a operação de mistura de canais. O uso da *batch normalization* e da ReLU é feito assim como em (HE et al., 2016a), exceto por não utilizar a ReLU após as convoluções em profundidade como o sugerido em (CHOLLET, 2017).

As análises sobre o número de parâmetros e de FLOPs apresentados a seguir exemplificam os ganhos proporcionados pela substituição de um módulo de *bottleneck* da ResNet por uma *Shuffle Unit*.

Estas análises sugerem que a redução do custo computacional, ou número de FLOPs entre um módulo de *bottleneck* e uma *Shuffle Unit* pode ser medido pela razão entre o custo individual de cada um desses blocos. Sendo $F = G$, $M = N$, m o número de canais intermediários, g o número de grupos, o custo computacional de uma unidade de *bottleneck* dado pela equação 3.7, o custo de uma *Shuffle Unit* dado pela equação 3.8, a

Figura 16 – Diagramas (a) do Módulo de *bottleneck* da ResNet; e (b) do Módulo de Mistura de Canais.



Fonte: A autora (2019).

redução alcançada pela substituição direta dos dois módulos é dada pela equação 3.9

$$FLOP_{bottleneck} = F^2 * (2Mm + K^2m^2) \tag{3.7}$$

$$FLOP_{Shuffle} = F^2 * (\frac{2Mm}{g} + K^2m) \tag{3.8}$$

$$\frac{FLOP_{Shuffle}}{FLOP_{bottleneck}} = \frac{F^2 * (\frac{2Mm}{g} + K^2m)}{F^2 * (2Mm + K^2m^2)}$$

$$\frac{FLOP_{Shuffle}}{FLOP_{bottleneck}} = \frac{2M + gK^2}{2gM + gK^2m} \tag{3.9}$$

Similarmente, o cálculo que indica a redução do número de parâmetros, P, entre as duas variações de módulos convolucionais residuais pode ser medido pela razão entre o número de parâmetros de cada um dos módulos. Sendo o número de operações de uma unidade de *bottleneck* dado pela equação 3.10 e os parâmetros de uma *Shuffle Unit* dado pela equação 3.11, a redução alcançada pela substituição direta dos dois módulos é dada pela equação 3.12.

$$P_{bottleneck} = 2Mm + K^2m^2 \tag{3.10}$$

$$P_{Shuffle} = \frac{2Mm}{g} + K^2m \tag{3.11}$$

$$\frac{P_{Shuffle}}{P_{bottleneck}} = \frac{\frac{2Mm}{g} + K^2m}{2Mm + K^2m^2}$$

$$\frac{P_{Shuffle}}{P_{bottleneck}} = \frac{FLOP_{Shuffle}}{FLOP_{bottleneck}} = \frac{2M + gK^2}{2gM + gK^2m} \tag{3.12}$$

Através das equações 3.9 e 3.12, é possível notar a relação direta do número de exemplos com a quantidade de FLOPs para a realização do cálculo da saída de um bloco residual. Como exemplo, se temos uma camada com *kernel* de tamanho $K = 3$, $M = 128$ canais de entrada, $m = 64$ canais de *bottleneck* e $g = 2$ grupos, a sua versão otimizada terá cerca de $6,08\times$ menos tanto de operações quanto de parâmetros. Caso $g = 3$, a redução seria de $9,11\times$.

Suponha uma quantidade definida de complexidade computacional para uma dada camada, uma rede que utiliza as unidades de mistura de canais pode processar um mapa de características com mais canais do que a unidade de *bottleneck* anteriormente definida. Essa capacidade é especialmente importante visto que redes pequenas tendem a representar os dados de uma maneira menos efetiva por possuir um número insuficiente de canais.

3.4 DESLOCAMENTO DE CANAIS

Até o momento, todas as camadas, módulos e redes profundas mencionada usam convoluções espaciais com *kernels* de tamanho 3 (ou maior) para aprenderem informações espaciais relacionadas a uma imagem de entrada. Essas convoluções espaciais possuem alto custo computacional e, mesmo variações otimizadas podem apresentar desvantagens. Na DSC, por exemplo, a razão R_{DSC} entre o número de FLOPs e a quantidade de acessos à memória necessários para a computação da camada é baixa, o que leva à sub-utilização do *hardware* (WU et al., 2018; ZHANG et al., 2018; CHOLLET, 2017).

Considerando uma camada com N filtros de *kernel* K e cujos dados de entrada e de saída possuem respectivamente as dimensões $F\times F\times M$ e $F\times F\times N$, define-se a razão entre a quantidade de FLOPs e a quantidade de operações de acesso à memória de uma convolução. A razão associada a uma convolução padrão é descrita na equação 3.13. Da mesma forma, define-se $R_{DephtWise}$ para uma convolução em profundidade na equação 3.14.

$$R = \frac{MNF^2K^2}{F^2(M + N) + K^2MN} \tag{3.13}$$

$$R_{DephtWise} = \frac{MF^2K^2}{2F^2M + K^2M} \tag{3.14}$$

O processo de leitura ou escrita de um dado na memória é mais custoso com relação ao tempo e à energia gastos do que a realização de uma FLOP. Diante disso, quanto maior o valor da razão entre o número de FLOPs e de parâmetros, maior será o tempo dedicado no processamento da rede profunda com relação às operações de acesso à memória.

Numericamente, se tomarmos uma camada com $M = 3$, $N = 16$, $F = 32$ e $K = 3$, as razões para as convoluções padrão e em profundidadesão $R = 22,24$ e $R_{DephtWise} = 4,48$, respectivamente. Como uma convolução em profundidade apresenta uma razão bastante inferior à de uma convolução padrão, ela pode ser incapaz de fornecer a eficiência computacional esperada em dispositivos nos quais a largura de banda da memória seja pequena.

Para reduzir o impacto dos acessos à memória, Wu *et al.* propõem uma nova operação: o **deslocamento de canais**. A operação proposta, diferentemente de uma convolução espacial, não requer nenhum parâmetro e nenhuma FLOP. A sua essência é ser uma série de ajustes dos canais em direções pré-definidas.

De forma mais específica, os filtros dessa operação são definidos pela equação 3.15, com $\mathscr{K} \in \mathrm{R}^{K \times K \times M}$, com K sendo a dimensão do *kernel* do filtro, M sendo o número de canais da entrada, e onde i_m e j_m são índices que definem o deslocamento de cada canal m. Ou seja, o resultado da passagem do dados de um canal pelo filtro, $\mathscr{K}$ será a função identidade aplicada a uma posição específica e definida pelo deslocamento daquele canal. Para uma operação de deslocamento com um filtro $\mathscr{K}$ de tamanho K, existem K^2 possibilidade de deslocamentos.

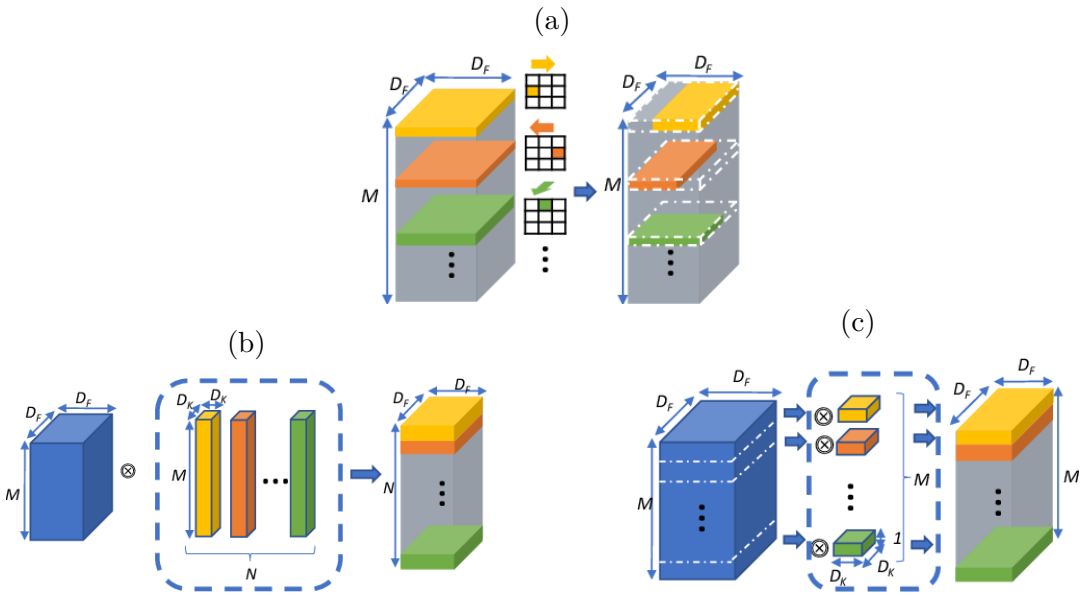
$$\mathscr{K}_{i,j,m} = \begin{cases} 1 \text{ se } i = i_m \text{ e } j = j_m \\ 0, \text{ caso contrário,} \end{cases} \tag{3.15}$$

Na figura 17, retirada de (WU et al., 2018), está ilustrada a operação de deslocamento de canais em contraste com a convolução padrão e com a convolução em profundidade. Na imagem 17a, as células brancas correspondem a zero e as células coloridas a um.

3.4.1 Módulo de deslocamento de canais

A partir da operação de deslocamento de canais, da mesma maneira que a ResNet e a ShuffleNet definem seus módulos construtores, Wu *et al.* apresenta um bloco de composição para redes profundas chamado de módulo de mistura de canais ou módulo *Shift*. Sua proposta,ao substituir as convoluções em profundidade por operações de deslocamento de canais, reduz a quantidade de FLOPs e de acesso à memória necessários ao treinamento e à inferência de uma CNNs. Tal qual em (ZHANG et al., 2018), o bloco *Shift* também se baseia no módulo *bottleneck* da ResNet (HE et al., 2016a). Nesse bloco, a entrada passa por uma convolução unitária, seguida de uma operação de deslocamento de canais para redistribuir as informações espaciais e por último as informações entre os canais são redistribuídas através de outra convolução pontual. Ambas as convoluções são precedidas

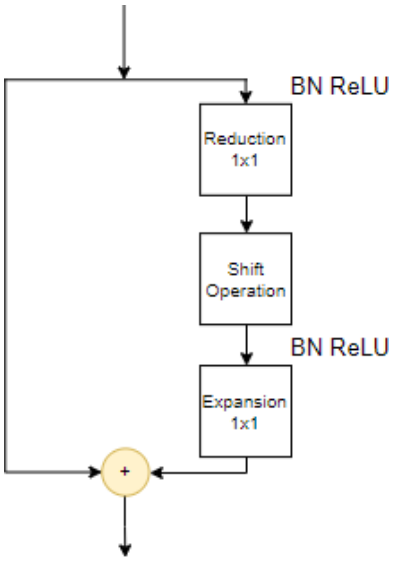
Figura 17 – Diagramas de (a) operação de deslocamento de Canais com $K = 3$; (b) uma convolução padrão; e (c) uma convolução em profundidade.



Fonte: Imagens retiradas de (WU et al., 2018).

de *batch normalization* e de uma ReLU. O módulo *Shift* definido é representado na figura 18.

Figura 18 – Diagrama do módulo *Shift*.



Fonte: A autora (2019).

Quando há a redução do mapa de características através do módulo *Shift*, o *pooling* é realizado pela segunda convolução a fim de que as informações espaciais sejam trocadas antes da redução dos dados. Similarmente ao módulo de *bottleneck* da ResNet, é possível controlar o quanto de expansão é oferecida aos dados intermediários do módulo. Entretanto, enquanto que nos módulos de *bottleneck* há convoluções espaciais padrão com campo de receptivo $K = 3$ de alto custo computacional e que forçam uma redução na

quantidade de canais intermediários, no módulo *Shift* não há esse alto custo. Como o tamanho do *kernel* de deslocamentos definido não afeta a quantidade de parâmetros e nem a de FLOPs da camada, há mais liberdade em definir o número de canais intermediários, m, do bloco e quanto maior o valor de m, mais informação pode ser analisada pela rede.

Os cálculos sobre o número de parâmetros, de FLOPs demonstrados a seguir exemplificam os ganhos proporcionados pela substituição de um módulo *bottleneck* da ResNet por um bloco *Shift*. Além disso, as razões entre o número de FLOPs e de acesso à memória de um bloco de *botteneck*, de uma DSC, de uma *Shuffle Unit* e do módulo *Shift* são apresentados.

O cálculo que indica a redução do número de parâmetros (ou do custo computacional) entre módulos *bottleneck* e *Shift* é medido pela razão entre o número de parâmetros (ou de FLOPs) de cada um desses blocos. Sendo $F = G$, $M = N$, m o número de canais intermediários, o número de parâmetros de uma unidade de *bottleneck* é dado pela equação 3.10 e os parâmetros de uma unidade *Shit* são dados pela equação 3.16, a redução alcançada pela substituição direta dos dois módulos é dada pela equação 3.17.

$$P_{Shift} = 2Mm \tag{3.16}$$

$$\frac{P_{Shift}}{P_{bottleneck}} = \frac{2Mm}{2Mm + K^2m^2}$$

$$\frac{P_{Shift}}{P_{bottleneck}} = \frac{FLOP_{Shift}}{FLOP_{bottleneck}} = \frac{2M}{2M + K^2m} \tag{3.17}$$

Como exemplo, se temos uma camada com *kernel* de tamanho $K = 3$, $M = 128$ canais de entrada e $m = 64$ canais de *bottleneck*, o bloco de otimização terá cerca de $3,25\times$ menos operações e parâmetros.

$$R_{DSC} = \frac{F^2(K^2 + M)}{4F^2 + K^2 + M} \tag{3.18}$$

$$R_{bottleneck} = \frac{mF^2(mK^2 + 2M)}{2F^2(2m + M) + K^2m^2 + 2Mm} \tag{3.19}$$

$$R_{Shuffle} = \frac{mF^2(gK^2 + 2M)}{2gF^2(2m + M) + gK^2m + 2Mm} \tag{3.20}$$

$$R_{Shift} = \frac{MmF^2}{F^2(M + m) + Mm} \tag{3.21}$$

Acrescenta-se às especificações para o exemplo numérico o tamanho da imagem de entrada e saída $F = 32$ e o número de grupo $g = 2$. Os cálculos dos valores das razões feitos

pelas equações 3.18, 3.19, 3.20 e 3.21 resultam em 1,04, 94,41, 16,84 e 40,96, respectivamente. Valores mais altos de razão indicam que mais tempo é passado no processamento da camada do que no acesso à memória.

4 ADAPTAÇÃO E IMPLEMENTAÇÃO DAS OTIMIZAÇÕES

Apesar de existirem diversos trabalhos na literatura sobre pré-otimizações de RNAs, a métrica de avaliação comum aos estudos é a taxa de acerto da rede otimizada em relação a sua versão original. Ou seja, diferentes autores treinam seus modelos e analisam o resultado que a otimização obteve nessas redes. Assim sendo, o desempenho da técnica de otimização é ofuscado por melhorias trazidas pela arquitetura e a comparação entre as vantagens e desvantagens da aplicação de diferentes técnicas de pré-otimização, portanto, não é direta. Para obter a comparação de desempenho entre as técnicas é necessário aplicá-las a uma única rede base e avaliar os resultados obtidos.

Neste capítulo, as bases de dados utilizadas, as implementações das redes base, as técnicas de otimização, as metodologias de treinamento e as métricas de avaliação serão detalhados. O objetivo é traçar um estudo que reúna os resultados das otimizações e aponte as vantagens e desvantagens de cada uma delas sobre à eficiência em relação ao número de parâmetros e ao tempo de processamento.

4.1 ANÁLISE PROPOSTA

Com a evolução e resultados empíricos obtidos nas pesquisas com RNAs, é de se esperar que as redes profundas estejam presentes em aplicações como celulares, carros, prédios, sensores, dispositivos médicos, entre outros produtos. Além disso, a elevada capacidade de aprendizado apresentada ao longo dos últimos anos pelas redes neurais profundas atrai cada vez mais não somente pesquisadores, mas também empresas, entusiastas e hobistas.

A utilização de modelos baseados em RNAs passa por duas fases: o treinamento do modelo e a sua inferência usando o modelo treinado. No primeiro caso, o modelo escolhido é treinado sobre amostras do problema a ser tratado, de forma a modelar suas características. Na segunda fase, a inferência, realizada sobre o modelo para se obter sua opinião sobre um exemplo no contexto do problema aprendido, pode ser realizada através de um servidor (*off site*) ou então pode ocorrer no próprio dispositivo (*in loco*). Na versão *off site*, é necessário que os dispositivos que estão utilizando o modelo estejam conectados à uma rede de comunicação para o envio dos dados a serem processados e para o recebimento das respostas da rede treinada. No conceito *in loco*, em vez de mandar os dados para um centro de processamento, a análise é realizada no dispositivo que gerou as informações a serem avaliadas.

Há vantagens e desvantagens em cada uma das alternativas, seja *in loco* ou *off site*. A necessidade de envio e recebimento de informações cria na aplicação a dependência da existência de conexão com a rede de telecomunicação, seja ela pública ou privada, a qual pode ou não estar disponível à todos os usuários. Por outro lado, enviar as informações

para serem processadas em outro local, faz com que a aplicação não se preocupe com as especificações de *hardware* do dispositivo utilizado, como é o caso das aplicações em que é possível realizar seus testes próximos dos usuários. Entretanto, realizar a inferência *in loco*, pode trazer vantagens relacionadas ao tempo de resposta, à privacidade e à segurança dos dados.

Enquanto muitas aplicações são adequadas ao processamento *off site*, para outras a inferência *in loco* é bastante desejável. Por exemplo, se temos um processo que depende de informações extraídas de imagens capturadas por múltiplas câmeras[1], ou então aplicações em carros autônomos, e em navegação de *drones* ou robôs, a inferência *in loco* é mais adequada uma vez que reduz o custo de envio das informações à um servidor e os riscos relacionados à segurança e ao atraso das respostas, os quais são muito altos para serem considerados.

No entanto, a realização da inferência de redes profundas *in loco* é um desafio, uma vez que os dispositivos nos quais elas estão alocadas muitas vezes possuem restrição de consumo de energia, de memória e de capacidade computacional. As pesquisas, desenvolvimento e o treinamento de diferentes arquiteturas de redes profundas costumam ocorrer em ambientes com grande quantidade de recursos computacionais, entretanto redes que se comportam bem nesses ambientes, podem ser inadequadas à ambientes restritivos. Dessa forma, para encontrar arquiteturas mais eficientes com respeito ao tempo de inferência, ao número de parâmetros e ao espaço de armazenamento, surgiram diferentes técnicas de otimização.

Quando se analisa a vasta variedade de otimizadores de arquitetura disponível na literatura, é possível que os projetistas que tem apenas o intuito de utilizar redes profundas como ferramentas de trabalho tenham que escolher e utilizar alguma das técnicas de otimização sem um conhecimento profundo sobre seus efeitos ou a teoria associada a esta técnica. Dessa forma, uma documentação que permita guiar o desenvolvedor é essencial. Tomando o Tensorflow[2] como exemplo, quando verificamos a documentação dessa biblioteca, a possibilidade óbvia de melhoria de desempenho está ligada ao uso do TensorflowLite e à quantização do modelo. No entanto, quando os resultados alcançados por esses métodos não são suficientes, não há outros indicativos de possíveis passos a serem seguidos. A falta desta informação pode levar ao abandono de muitos projetos que, de outra maneira, obteriam êxito em aplicações práticas.

Desta forma, objetivamos reunir e comparar os efeitos de algumas das técnicas de pré-otimização da literatura, as quais foram apresentadas no capítulo 3. A intenção é prover um estudo crítico sobre os efeitos das estratégias de otimização de arquiteturas

[1] Exemplos seriam a medição do tempo de espera de clientes em uma loja ou então o reconhecimento de pessoas em sistemas de segurança.

[2] O Tensorflow é uma biblioteca de código aperto para aprendizagem de máquina largamente utilizada para o desenvolvimento de redes profundas. Mais sobre a biblioteca pode ser encontrado em <https://www.tensorflow.org/>.

profundas, de forma a listar as vantagens e desvantagens de cada uma delas. Além disso, com os resultados apresentados, esta dissertação pode fornecer um guia na definição de arquiteturas que sejam mais eficientes em relação a um, ou mais, dos aspectos aqui estudados. Essas técnicas serão aplicadas a duas diferentes redes de classificação e em uma rede generativa. As redes são a SqueezeNet (IANDOLA et al., 2016a), a ResNet (HE et al., 2015) e a Pix-2-Pix (ISOLA et al., 2017) afim de analisar a eficiência das otimizações com respeito ao número de parâmetros e ao tempo de processamento de um exemplo.

Como precursor deste trabalho, o artigo (SANTOS et al., 2018) investiga os efeitos sobre a redução do número de parâmetros e a sua relação com o tempo de processamento de um exemplo de teste ao ajustar o módulo *fire* para que ele fosse formado por DSCs. A rede resultante é comparada a diferentes redes neurais como a AlexNet, a MobileNet, a VGG19 e a SqueezeNet original sobre as bases de dados CIFAR 10 e CIFAR 100. A SqueezeNet-DSC mostrou uma redução de 37% no seu número de parâmetros enquanto que, para as bases de dados citadas, sua acurácia ficou apenas 1,07% e 3,06% abaixo da acurácia obtida pela SqueezeNet original.

4.2 BASES DE DADOS UTILIZADAS

Neste trabalho serão utilizadas três bases de dados. Os dois primeiros conjuntos de dados correspondem a problemas de classificação, os quais são o CIFAR 10 e o CIFAR 100 (KRIZHEVSKY; HINTON, 2009). O terceiro é o *maps↔satellite*, um dos vários banco de dados de transformação de imagem em imagem estudados em Phillip Isola *et al.* (ISOLA et al., 2017).

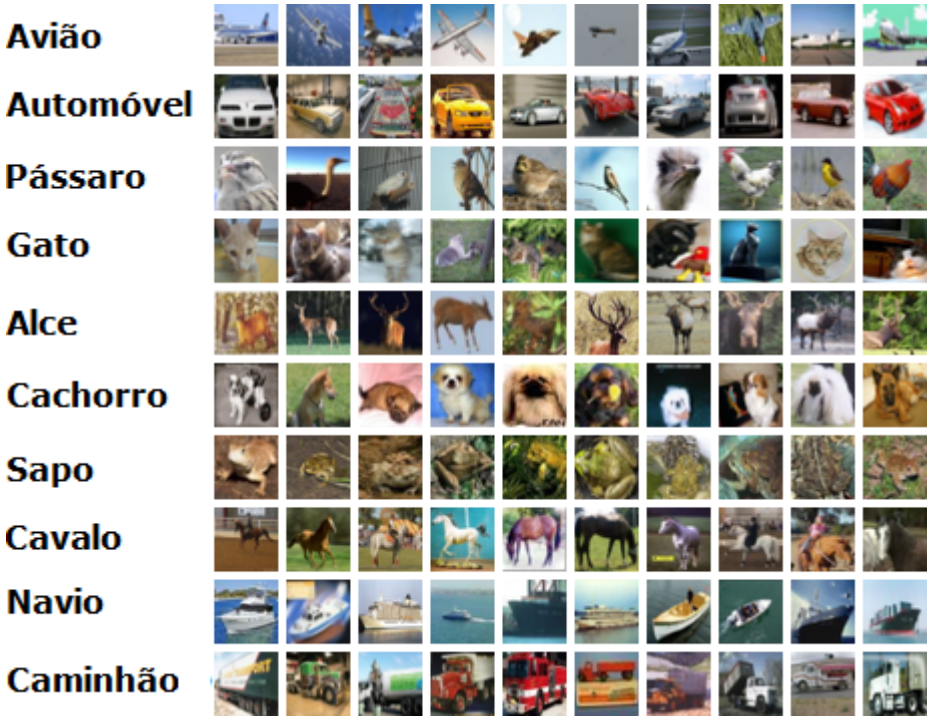
A base de dados ImageNet (DENG et al., 2009) não foi utilizada para o desenvolvimento deste trabalho pois seu tamanho e complexidade impõe a necessidade de um longo tempo de treinamento nas redes profundas. Diante do tempo disponível para realização deste trabalho e da proposta de treinar e comparar os efeitos alcançados por diferentes técnicas de pré-otimização aplicadas a uma variedade de arquiteturas, o uso dessa base de dados se tornou inviável.

4.2.1 Base de dados CIFAR

A base de dados CIFAR foi organizada por Alex Krizhevsky, Vinod Nair e Geoffrey Hinton em 2009 com o intuito de pesquisar sobre modelos generativos e classificadores compostos por redes profundas (KRIZHEVSKY; HINTON, 2009). As imagens que compõem esta base são um subconjunto rotulado de um outro banco de dados composto por 80 milhões de imagens, o *Tiny Image* descrito em detalhes em (TORRALBA; FERGUS; FREEMAN, 2008).

Existem duas variações da base de dados CIFAR, a versão com 10 classes (conhecida como CIFAR 10), e a versão com 100 classes (CIFAR 100). O CIFAR 10 é composto por

Figura 19 – Exemplos de imagens da base de dados CIFAR10.



Fonte: Image retirada de <https://www.cs.toronto.edu/~kriz/cifar.html>.

60.000 imagens coloridas (RGB) com 32x32 pixels de largura e altura, os quais são agrupadas em dez diferentes classes, cada uma com o mesmo número de imagens (6.000 imagens por classe). As imagens do CIFAR 10 são agrupadas em seis conjuntos de dados, sendo cinco de treinamento e um de teste. O conjunto de teste contém mil imagens selecionadas aleatoriamente de cada classe, totalizando dez mil imagens. As 50.000 imagens restantes são organizadas e distribuídas de forma aleatória nos cinco conjuntos de treinamento de forma que um deles pode possuir mais imagens de uma classe e menos de outra. Na figura 19 estão exemplos de imagens do CIFAR 10.

A base de dados CIFAR 100 é composta e dividida em conjuntos de treinamento e de teste da mesma forma que a CIFAR 10. A diferença entre os dois está no número de classes e no número de imagens por classe. Enquanto que o CIFAR 10 possui 10 classes com 6.000 imagens cada, o CIFAR 100 possui 100 possíveis classes com 600 imagens cada.

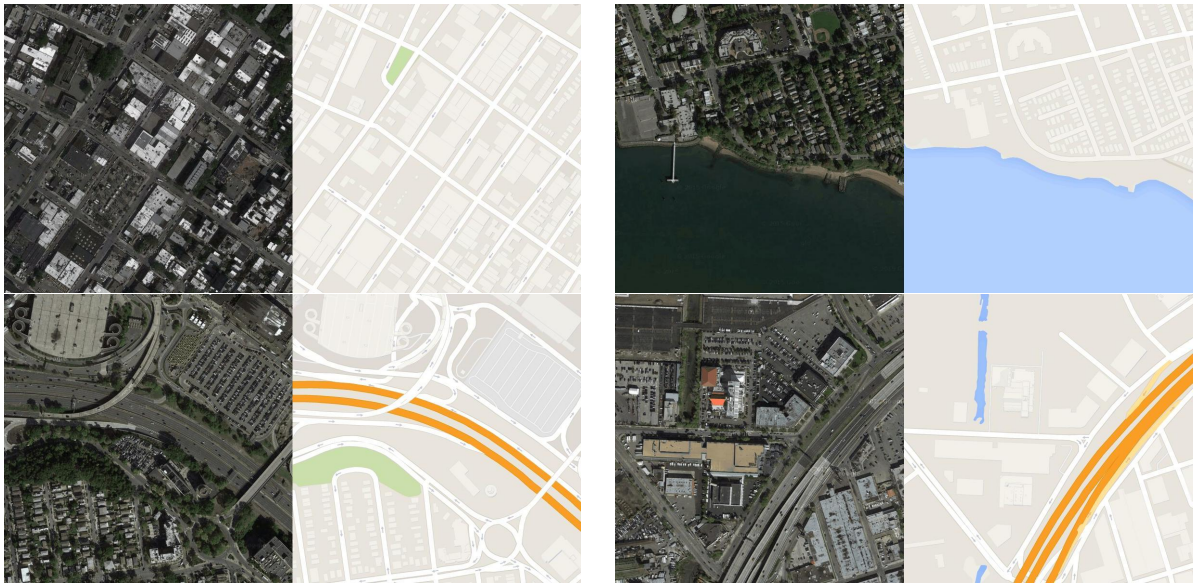
Para ambos os bancos de dados a anotação (rótulo) de uma imagem é definida por um número inteiro. Para o CIFAR 10, suas dez classes são representadas por números de 0 a 9 e de forma análoga, para o CIFAR 100, suas cem classes são representadas por números de 0 a 99.

4.2.2 Base de dados *Maps* ↔ *Satellite*

A base de dados *maps*↔*satellite* foi construída para permitir a conversão de imagens de satélite para mapas, e vice versa. Ela é uma dentre as opções disponibilizadas para

a avaliação das redes Pix-2-Pix em (ISOLA et al., 2017) e seus exemplos foram obtidos a partir de imagens da cidade de Nova Iorque e seus arredores usando o Google Mapas[3].

Figura 20 – Exemplos de imagens da base de dados *Maps↔Satellite*.



Fonte: A autora (2019).

A base é composta por 1.096 imagens de treinamento e 1.098 imagens de teste. Para definir os conjuntos de treinamento e de teste, o conjunto todo foi dividido de acordo com a latitude média, tomando cuidado para que nenhum pixel de treinamento fosse reproduzido no conjunto de teste. Cada imagem da base corresponde a concatenação de uma imagem de satélite e a sua corresponde imagem de mapa, ambas com dimensões 600×600.

Na figura 19 são apresentados exemplos de imagens da base *Maps↔Satellite*.

O *Maps↔Satellite* foi escolhido para o treinamento das redes generativas visto que neste trabalho será realizado o procedimento de avaliação perceptual para esses modelos e que Isola *et al.* avaliam suas arquiteturas através do mesmo método com essa base de dados.

4.3 METODOLOGIA PROPOSTA

No presente trabalho são avaliadas duas arquiteturas de classificação (ResNet e SqueezeNet) e suas respectivas variações que utilizam a DSC, a mistura (*Shuffle*) e o deslocamento (*Shift*) de canais. Além delas, uma arquitetura generativa, a Pix-2-Pix e suas variações também são avaliadas. Todos os modelos são implementados utilizando a biblioteca PyTorch[4]

[3] https://www.google.com/maps

[4] PyTorch é uma biblioteca de aprendizagem de máquina de código aberto. Mais informações disponíveis em <https://pytorch.org/>.

As métricas de avaliação e os parâmetros de treinamento utilizados são descritos a seguir.

4.3.1 Métricas de avaliação

A avaliação das redes e das técnicas de otimização é realizada utilizando: da taxa de acerto ou acurácia obtida pelo modelo, do número de parâmetros da rede, e o tempo de inferência de um único exemplo de teste em uma arquitetura do tipo GPU e em uma arquitetura do tipo CPU. Para a rede generativa, são utilizados a acurácia pixel-a-pixel e os resultados de um estudo perceptual das imagens geradas. Para cada rede treinada, um arquivo .t7 que compreende o modelo com a melhor acurácia sobre o conjunto de validação é salvo.

O espaço de armazenamento de cada rede é tido como o tamanho do arquivo .t7 gerado durante o seu treinamento e que contém as informações sobre sua arquitetura do modelo e sobre seus pesos. O número de parâmetros da rede é definido pela arquitetura implementada, ou seja, podem ser calculados a partir das equações apresentadas no decorrer do capítulo 3. A acurácia é medida a partir de exemplos de teste que não são utilizados para a atualização de pesos durante a fase de treinamento. Esses exemplos fazem parte do conjunto de teste de cada base de dados. Por fim, o tempo de inferência é estimado pela média dos tempos de processamento de cada exemplo do conjunto utilizado para medir a acurácia. A rede é então avaliada considerando os tempos de inferência de cada exemplo de teste. A média e o desvio padrão dos tempos de processamento serão utilizados na avaliação e comparação das arquiteturas.

Para a avaliação das redes GANs é utilizada a métrica de acurácia por pixel. Além dela, uma vez que avaliar a qualidade de imagens geradas sinteticamente é um problema em aberto (ISOLA et al., 2017) e que o principal objetivo da rede Pix-2-Pix é criar imagens que sejam reais o suficiente para um observador humano, realizamos também um estudo perceptual nas imagens de teste para avaliação dos modelos otimizados de forma similar ao realizado em (ISOLA et al., 2017). Esse estudo contou com voluntários e foi feito através de um site[5]. No site o usuário se cadastra e, ao iniciar o processo de análise, será apresentado a 50 imagens (25 reais e 25 sintéticas). Cada imagem fica disponível por 1 segundo, após esse tempo a imagem é removida e o processo aguarda que o usuário responda se esta é uma imagem real ou sintética. Nas 10 primeiras tentativas o sistema informa ao usuário se ele acertou ou errou e nas quarenta seguintes não há esse retorno.

Uma vez que cada rede avaliada fornece 10 imagens de treinamento ao usuário, no caso dele realizar a votação para mais de uma rede, ele terá passado por mais treinamento e as respostas obtidas podem diferir das que o mesmo usuário teria dado se essa segunda avaliação na verdade fosse a sua primeira avaliação. Como um determinado voluntário

[5] Disponível em <http://pesquisadequalidadegan.com.br/>.

poderia decidir responder diversas vezes enquanto outros apenas o fariam uma vez, então é determinado que cada usuário pode realizar apenas um teste.

Todos os modelos foram treinados e testados em uma placa de vídeo NVIDIA GTX TITAN e em seguida testados novamente através de um processador Intel® Core™i7-7700K. Ambas as medições foram feitas a fim de comparar o comportamento das redes quando rodadas através de processadores com mais ou menos poder computacional (no caso uma GPU e uma CPU).

4.3.2 Hiper-parâmetros Utilizados

Os hiper-parâmetros de uma CNN são variáveis definidas antes do início do treinamento e que determinam a estrutura (ex: o número de filtros de uma camada) e a maneira com a qual os pesos de uma rede são treinados, como o a regra de atualização dos pesos e a taxa de aprendizagem. Na presente sub-seção define-se o mecanismo de busca dos seguintes hiper-parâmetros de treinamento: a taxa de aprendizado[6] e seu decaimento[7], a paciência do treino[8], o tamanho dos *batches*[9], o número máximo de épocas[10] e o otimizador de treinamento[11].

Para cada rede treinada há um conjunto de hiper-parâmetros que proporcionam um melhor desempenho. Para encontrar esse conjunto, o processo de treinamento deve ser realizado variando os possíveis valores de cada hiper-parâmetro e escolhendo aquele que obteve a melhor taxa de acerto. As metodologias para encontrar o melhor conjunto de hiper-parâmetros podem ser por busca manual, aleatória ou por grade, entre outros (CLAESEN; MOOR, 2015). Na busca manual os valores são escolhidos pelo desenvolvedor e um a um os resultados obtidos são registrados para comparação; na busca aleatória esses valores são definidos aleatoriamente e; na busca por grade é definido um intervalo de busca e um passo e os valores compreendidos entre o limite mínimo e máximo do intervalo e um passo distantes entre si são avaliados. Um exemplo de valores a serem utilizados por uma busca por grade com intervalo entre 10 e 50 e com passo de 10 é $\mathbf{C} = \{10, 20, 30, 40, 50\}$.

Neste trabalho e para o estudos das redes de classificação, o método manual de busca dos hiper-parâmetros foi utilizado. Os espaços de busca da taxa de aprendizado e do

[6] A taxa de aprendizado controla o quanto os pesos de uma rede são alterados em resposta a um sinal de erro retro-propagado.

[7] O decaimento da taxa de aprendizado é o quanto essa taxa será reduzida por gatilhos pré-estabelecidos no algoritmo de treinamento do modelo.

[8] A paciência da rede é o número de épocas que o treinamento aguarda antes de realizar o decaimento da taxa de aprendizado. A contagem do número de épocas da paciência se inicia quando o a função de perda não apresenta melhorias significativas no treinamento.

[9] *Batches* são sub-conjuntos disjuntos e de tamanho fixo dos exemplos de treinamento. O número de *batches* em um conjunto de treino corresponde ao quociente da divisão do número de exemplos total pelo tamanho de um *batch*.

[10] No processo de treinamento os exemplos de treinamento são apresentados, *batch* a *batch* à rede. Quando todos os exemplos de treinamento foram vistos, completa-se um ciclo, ou uma época de treinamento.

[11] O otimizado de treinamento é a regra matemática que rege a atualização dos pesos da rede após a retro-propagação dos erros.

tamanho do *batch* foram definidos de acordo com a lista apresentada a seguir. Define-se também o número máximo de épocas de treinamento como 200 e o Gradiente Descendente Estocástico (SGD) (RUMELHART et al., 1988) como o otimizador de treinamento. Por fim, dada uma taxa de treinamento inicial, quando o valor da perda não sofre redução maior do que 0,01 em relação à época anterior por dez épocas sucessivas, aplica-se o decaimento com o valor da taxa de aprendizado sendo reduzido por um fator de 10. Quando o valor da função de perda se estabiliza pela quarta vez, o processo de treinamento é interrompido. Esse término antes do número máximo de épocas definido originalmente é chamado de término antecipado do treino.

- Taxa de aprendizado: $\mathscr{LR} = \{0,1; 0,01; 0,001\}$;
- Tamanho do *batch*: $\mathscr{B} = \{128; 256\}$;

Para toda as redes de classificação treinadas, os resultados listados no capítulo 5 são referentes ao conjunto de hiper-parâmetros que possibilitou a melhor acurácia.

Para as redes generativas a configuração padrão do repositório oficial[12] foi utilizada nos treinamentos.

4.3.3 Redução e Expansão da ResNet

Como avaliação complementar, da mesma forma como Wu *et al.* realizam em (WU et al., 2018), as ResNets base sofrem compressão do número de parâmetros e suas versões otimizadas são expandidas. Em outras palavras, as ResNets base (ResNet20, 56 e 110) são reduzidas até atingirem o patamar do número de parâmetros de suas versões com os módulos DSC, *Shuffle* e *Shift*, enquanto que as últimas são expandias até atingirem o tamanho das suas redes bases. As redes de tamanhos similares são comparadas entre si.

A compressão é realizada de duas maneiras distintas. A primeira delas utiliza o módulo de *bottleneck* associado à redução do número de canais intermediários no módulo. A segunda maneira apenas reduz o número de canais no decorrer da rede. No caminho contrário, a expansão é realizada sobre as redes otimizadas aumentando-se o número de canais intermediários dos módulos até que as redes atinjam o valor mais próximo possível do número de parâmetros das redes base. As taxas de compressão e expansão são as mesmas para todas as camadas numa rede.

4.4 ADAPTAÇÕES NAS REDES E PROPOSTAS DE OTIMIZAÇÃO

As redes base de classificação utilizadas neste trabalho (a SqueezeNet e a ResNet) foram originalmente propostas para a base de dados ImageNet (DENG et al., 2009), um banco complexo e que contém mais de 14 milhões de imagens dividias em mil classes.

[12] Disponível em <https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix>.

Sabendo que o uso da ImageNet acarretaria num elevado tempo de treinamento, para minimizar esse problema foram utilizados a CIFAR 10 e 100 em seu lugar. No entanto, enquanto as imagens da ImageNet possuem dimensões 256×256 pixels de largura e altura, a CIFAR possui dimensões de apenas 32×32 pixels. Além disso, as 1000 classes do primeiro banco de dados contrastam com as 10 ou 100 classes da CIFAR. Essas diferenças entre os bancos de dados levam à necessidade de adaptação nas redes. Já para as redes generativas, não é necessária a adaptação a outro banco de dados uma vez que a Pix-2-Pix base realiza experimento no conjunto *maps*↔*satellite*.

Além disso, como os módulos descritos em detalhes no capítulo 3 foram concebidos e inicialmente avaliados em redes residuais, para que sua eficiência seja verificada como um módulo genérico, é necessário que eles sejam introduzidos em redes de diferentes macro-estruturas. No presente trabalho, as redes SqueezeNet, ResNet20, 56 e 110 e a rede generativa da Pix-2-Pix serão utilizadas como redes base para verificação da eficiência dos módulos de otimização DSC, *Shuffle* e *Shift*.

As adaptações das três redes para atenderem às técnicas de otimizações apresentadas são descritas a seguir.

4.4.1 Arquitetura SqueezeNet

A primeira das redes a ser modificada já tem seu próprio módulo de otimização, o módulo *fire*. A fim de não interferir na estrutura que é a base da SqueezeNet, as outras técnicas foram introduzidas dentro do bloco de contração e expansão já existente e as modificações realizadas estão descritas a seguir.

4.4.1.1 Adaptação à base de dados

As adaptações aplicadas à SqueezeNet compreendem:

- adição do *batch normalization* que não está presente no artigo original;
- remoção do *dropout* na última camada convolucional;
- a redução do tamanho do filtro na primeira camada de 7×7 para 3×3;
- redução do passo da rede de 2 para 1;
- redução da taxa de redução do mapa de características de 3×3 para 2×2;

Essas adaptações foram sugeridas pelo repositório pytorch-cifar[13] e seguidas para o desenvolvimento do presente trabalho.

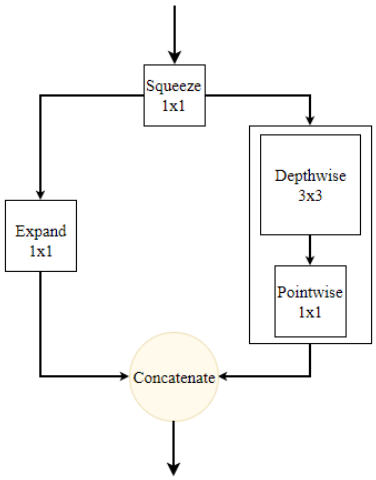
[13] Disponível em <https://github.com/kuangliu/pytorch-cifar>.

4.4.1.2 Adaptação às convoluções separáveis em profundidade

Para introduzir as DSCs na SqueezeNet, propõe-se modificar as convoluções com *kernel* 3×3 internas do módulo *fire*, transformando-as em convoluções separáveis em profundidade. À SqueezeNet formada por módulos *fire*-DSC, dá-se o nome de SqueezeNet-DSC.

A DSC fatora uma convolução padrão em uma convolução em profundidade e uma convolução pontual. Essa fatoração, como já demonstrado no capítulo 3, reduz drasticamente o número de parâmetros na camada alterada. O módulo *fire* alterado pode ser visto na figura 21.

Figura 21 – Diagrama do módulo *fire* com DSC.



Fonte: A autora (2019).

Sabe-se que o número de parâmetros P de uma convolução padrão é dada pela equação 3.4 enquanto que para uma DSC essa quantidade é dada pela equação 3.5. Dessa forma, se consideramos m o número de canais de saída da etapa de contração do módulo *fire* e N o número de canais de saída da camada de expansão, o número de parâmetros de um módulo *fire* pode ser escrito como:

$$P_{FIRE} = Mm + m\frac{N}{2} + D_K D_K m\frac{N}{2} \tag{4.1}$$

A equação anterior leva à equação 4.2 como o número de parâmetros para o módulo *fire*-DSC.

$$P_{FIRE-DSC} = Mm + mN + D_K D_K m \tag{4.2}$$

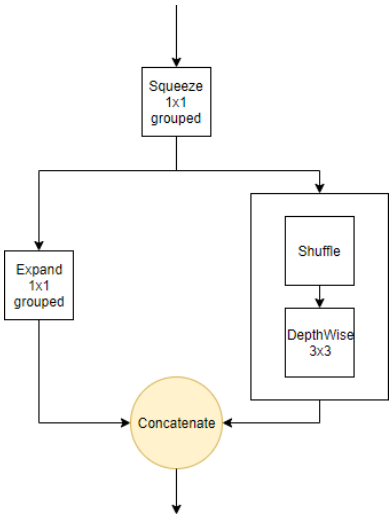
Para uma visualização matemática, se temos um número de canais de saída $N = 128$, o número de canais de entrada $M = 96$, o número de canais intermediários $m = 16$ e o tamanho do *kernel* do filtro $D_K = 3$. Em um módulo padrão teríamos 11.776 parâmetros, contrastando com 3.728 parâmetros do módulo modificado. Em outras palavras,

temos 68,34% de redução no número de parâmetros do módulo *fire* ao inserir a DSC na convolução 3×3 da etapa de expansão.

4.4.1.3 Adaptação ao módulo de mistura de canais

O módulo *fire*, ao se adaptar ao módulo *Shuffle*, realiza mudanças em todas as suas convoluções. Na etapa de contração a adaptação é realizada pela substituição da convolução unitária por uma convolução unitária por grupos. Na etapa de expansão, a o fluxo que passa pela convolução pontual recebe a mesma modificação já realizada na etapa de contração e o outro fluxo recebe uma operação de mistura de canais seguida por uma convolução 3×3 em profundidade em substituição à operação existente. À SqueezeNet formada por módulos *fire-Shuffle*, dá-se o nome de SqueezeNet-*Shuffle*. O diagrama que ilustra a introdução do módulo de mistura de canais no bloco *fire* pode ser visto na figura 22.

Figura 22 – Diagrama do módulo *fire-Shuffle*.



Fonte: A autora (2019).

O cálculo do número de parâmetros do módulo *fire-Shuffle* é dado pela equação 4.3.

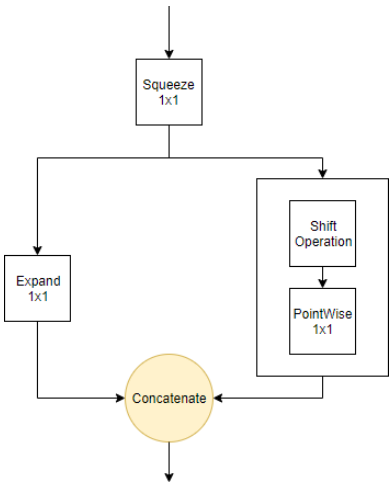
$$P_{FIRE-SHUFFLE} = \frac{Mm}{g} + \frac{mN}{2g} + mD_K D_K \tag{4.3}$$

Para efeito matemático, toma-se o módulo de exemplo descrito na sub-seção anterior e define-se a quantidade de grupos como $g = 2$. Enquanto que num módulo padrão teríamos 11.776 parâmetros, o módulo modificado nos permite reduzir esse valor a 1.424 parâmetros. Em outras palavras, temos 87,91% de redução no número de parâmetros de um módulo ao substituir o módulo padrão por sua versão *fire-Shuffle*.

4.4.1.4 Adaptação ao módulo de deslocamento de canais

Para introduzir o módulo *Shift* na SqueezeNet, propõe-se modificar as convoluções com *kernel* não unitário da etapa da de expansão do módulo *fire*, transformando-as em uma operação de deslocamento de canais seguida com outra convolução pontual. À SqueezeNet formada por módulos *fire-Shift*, dá-se o nome de SqueezeNet-Shift. O diagrama que ilustra a introdução do módulo de deslocamento de canais no módulo *fire* pode ser visto na figura 23.

Figura 23 – Diagrama do módulo *fire-Shift*.



Fonte: A autora (2019).

O cálculo do número de parâmetros do módulo *fire-Shift* é dado pela equação 4.4.

$$P_{FIRE-SHIFT} = Mm + mN \tag{4.4}$$

Para efeito matemático, toma-se o módulo de exemplo descrito na sub-seção 4.4.1.2. Enquanto que em um módulo padrão teríamos 11.776 parâmetros, o módulo modificado nos permite reduzir esse valor para 3.584 parâmetros. Em outras palavras, temos 69,56% de redução no número de parâmetros de um módulo ao substituir o módulo padrão por sua versão *fire-Shift*.

4.4.2 Arquitetura ResNet

As modificações realizadas na macro-arquitetura e no bloco residual da ResNet para se adaptarem à base de dados CIFAR e às técnicas de otimização em estudo são descritas a seguir, com excessão dos módulos *Shuffle* e *Shift*. Como os blocos residuais de construção da ResNet, foram utilizados em (ZHANG et al., 2018) e em (WU et al., 2018) na concepção dos módulos *Shift* e *Shuffle*, suas implementações foram mantidas tal qual descritas no capítulo 3.

4.4.2.1 Adaptação à base de dados

A ResNet original tem uma macro-arquitetura definida por uma camada de entrada seguida de *pooling*; um número definido de módulos residuais distribuídos entre quatro blocos; e uma camada totalmente conectada responsável pelo resultado da rede. Essa configuração foi construída tendo em foco a base de dados ImageNet, entretanto pelas diferenças existentes entre essa base de dados e o CIFAR, mudanças são necessárias.

A adaptação da macro-arquitetura da ResNet é feita em três passos:

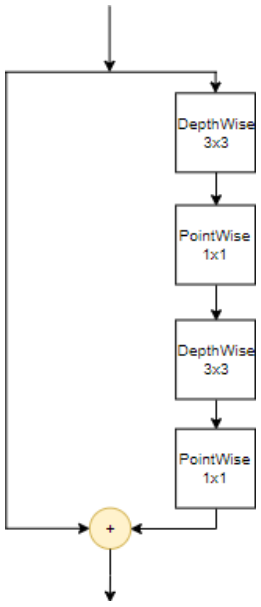
- o primeiro *pooling* da rede é retirado;
- a dimensão do *kernel* da primeira convolução é reduzido de 7 para 3; e
- em vez de quatro blocos de módulos residuais, tem-se apenas três;

Essas adaptações foram sugeridas pelo repositório shiftresnet-cifar[14] e seguidas para o desenvolvimento do presente trabalho.

4.4.2.2 Adaptação às convoluções separáveis em profundidade

Para introduzir as convoluções separáveis em profundidade na ResNet, propõe-se substituir as convoluções padrão do módulo original por DSCs. À ResNet formada pelos módulos modificados, dá-se o nome de ResNet-DSC. O módulo alterado pode ser visto no diagrama da figura 24.

Figura 24 – Diagrama do módulo DSC da ResNet.



Fonte: A autora (2019).

Com o número de canais de entrada, M, sendo o mesmo valor do número de canais de saída, N, o cálculo do número de parâmetros de um bloco residual padrão é dado pela

[14] Disponível em <https://github.com/alvinwan/shiftresnet-cifar>.

equação 4.5 e do número de parâmetros de um bloco residual com DSCs é dado pela equação 4.6. Por completude, define-se também o cálculo do número de parâmetros dos blocos residuais de *bottleneck*, *Shuffle* e *Shift* nas equações 4.7, 4.8, 4.9, respectivamente.

$$P_{Residual} = 2D_k^2 mM \tag{4.5}$$

$$P_{Residual-DSC} = D_k^2(M + m) + 2mM \tag{4.6}$$

$$P_{botteneck} = m * (2M + mD_K^2) \tag{4.7}$$

$$P_{Shuffle} = \frac{Mm}{g} + D_K^2 m + \frac{mM}{2 * g} \tag{4.8}$$

$$P_{Shift} = 2mM \tag{4.9}$$

Para uma visualização matemática, se temos um número de canais de entrada e de saída $M = N = 32$, o número de canais intermediários $m = 16$, o tamanho do núcleo do filtro $D_K = 3$ e o número de grupos $g = 2$ quando as convoluções por grupo são aplicadas. Em um bloco residual padrão teríamos 9.216 parâmetros, contrastando com 1.456, 3.328, 528, 1.024 parâmetros dos blocos modificados, respectivamente. Em outras palavras, temos 84,20%, 63,89%, 94,27% e 88,89% de redução no número de parâmetros dos respectivos blocos em relação ao modelo original.

Os diagramas dos demais módulos residuais estão representados nas figuras 2, 3, 16 e 18.

4.4.3 Arquitetura Pix-2-Pix

A rede generativa da Pix-2-Pix, na forma de uma UNet, é formada por uma etapa de codificação e outra de decodificação. Propõe-se, como modificação da arquitetura da rede generativa, a substituição das convoluções da etapa de codificação por DSC e por módulos de otimização *Shuffle* e *Shift* sem as conexões de atalho.

O número de parâmetros de uma convolução padrão e de uma DSC é dado pelas equações 3.4 e 3.5. Já o número de parâmetros dos módulos *Shuffle* e *Shift* sem as conexões de atalho são iguais às equações 4.8 e 4.9, uma vez que a retirada das conexões de atalho não modificam o número de parâmetros do bloco.

Para uma visualização matemática, se temos um número de canais de entrada e de saída $M = N = 32$, o número de canais intermediários $m = 16$, o tamanho do núcleo do filtro $D_K = 4$ e o número de grupos $g = 8$ quando as convoluções por grupo são aplicadas. Em uma convolução padrão de codificação teríamos 16.384 parâmetros, contrastando com

1.536, 352, 1.024 parâmetros dos blocos modificados, respectivamente. Em outras palavras, temos 90,62%, 97,85% e 93,75% de redução no número de parâmetros na substituição de uma convolução padrão por, respectivamente, uma DSC e módulos *Shuffle* e *Shift* sem conexões de atalho.

5 EXPERIMENTOS E RESULTADOS

Neste capítulo são apresentados os resultados obtidos pelo experimento descrito no capítulo 4. Os resultados de classificação são organizados de acordo com três cenários: a aplicação direta das técnicas de otimização nas redes base; a expansão das redes ResNet otimizadas até o patamar de número de parâmetros das redes base; e a contração das ResNets base até o patamar de número de parâmetros das redes otimizadas. Os resultados sobre os modelos generativos são apresentados em seguida.

Tomando as redes SqueezeNet, ResNet 20, ResNet 56 e ResNet 110, bem como as suas variantes que compreendem as técnicas de DSC, *Shuffle* e *Shift*, a eficiência da rede é apresentada pela razão entre a taxa de acerto e o número de parâmetros do modelo e, em seguida, pelo seu tempo de inferência.

Uma vez que a implementação da operação de deslocamento de canais[1] é feita para que ela funcione apenas em GPU, as medições em CPU realizadas durante o experimento consideraram que, quando presentes nas redes otimizadas, essas operações continuariam sendo realizadas em GPU. Para que a distribuição do processamento não deixe em desvantagens as redes que não possuem a operação de *Shift*, em cada arquitetura foi escolhida uma convolução para ser processada em GPU.

Na SqueezeNet e em suas otimizações, com exceção da otimização por deslocamento de canais, a convolução da etapa de contração do módulo *fire* é enviada para processamento na GPU. Essa escolha é feita considerando que o processamento mais eficiente dessa convolução traria menos benefícios no tempo de inferência da rede do que a escolha de outra operação do módulo *fire*.

Para as variações do modelo ResNet, a escolha do envio de convoluções de baixo custo computacional para uma GPU, em resultados preliminares, apresentou resultados bastante inferiores ao desempenho temporal das redes com operações de deslocamento de canais. Dessa forma, as medições em CPU que serão descritas nas seções seguintes foram realizadas com o envio de convoluções de alto custo computacional para a GPU. Como será demonstrado mais adiante no capítulo, mesmo essa escolha não foi suficiente para que as técnicas DSC e *Shuffle* superassem, em tempo de processamento, as redes ResNet-*Shift*.

Os resultados obtidos pela rede Pix-2-Pix, diferente das redes anteriores que se baseiam na acurácia na classificação das classes, são medidos de acordo com a acurácia pixel-a-pixel e com o impacto perceptual das imagens geradas. Para a base de dados *maps*↔*satellite*, respostas de voluntários são utilizadas para avaliar o impacto perceptual da aplicação de técnicas de otimização sobre o modelo de rede profunda original. Para a obtenção desta segunda métrica, cada rede generativa foi avaliada por aproximadamente 10 voluntários.

[1] Disponível em <https://github.com/peterhj/shiftnet_cuda_v2>.

5.1 EXPERIMENTOS COM AS DIFERENTES OTIMIZAÇÕES

Nesta seção são apresentados os resultados obtidos sobre o primeiro cenário de avaliação: a aplicação direta das otimizações por DSC e pelos módulos *Shift* e *Shuffle*.

5.1.1 Arquitetura SqueezeNet

Tabela 1 – Resultados do modelo SqueezeNet e suas otimizações na base CIFAR 10.

	Taxa acerto Top 1 (%)	Número de Parâmetros	E_P ($\times 10^{-3}$)	Tempo GPU (ms$\pm\Delta$)	Tempo CPU (ms$\pm\Delta$)
SqueezeNet	**90,11**	746638	0,12	**2,21 ± 0,25**	6,61 ± 0,12
SqueezeNet-DSC	87,50	248670	0,35	2,40 ± 0,21	9,55 ± 0,15
SqueezeNet-Shift	87,97	246926	0,36	2,66 ± 0,22	**5,36 ± 0,13**
SqueezeNet-Shuffle (g = 2)	87,83	**114574**	**0,77**	2,58 ± 0,31	10,23 ± 0,11

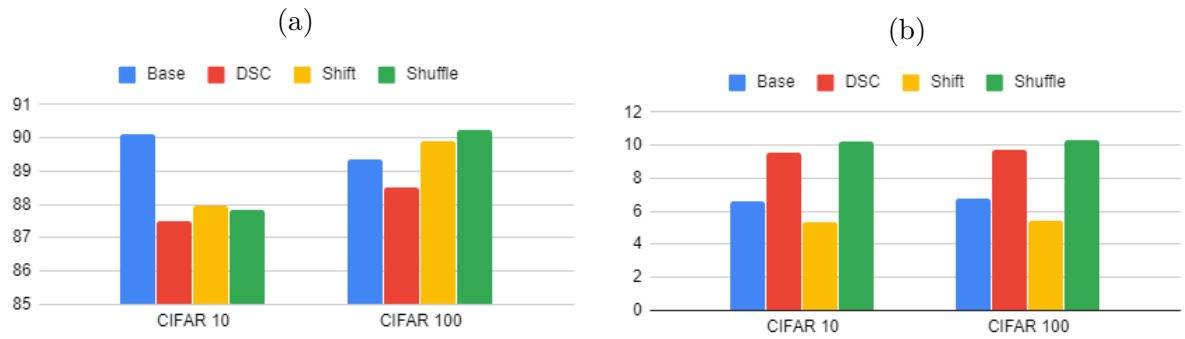
Fonte: A autora (2019).

Tabela 2 – Resultados do modelo SqueezeNet e suas otimizações na base CIFAR 100.

	Taxa acerto Top 5 (%)	Número de Parâmetros	E_P ($\times 10^{-3}$)	Tempo GPU (ms$\pm\Delta$)	Tempo CPU (ms$\pm\Delta$)
SqueezeNet	89,36	792988	0,12	**2,20 ± 0,17**	6,78 ± 0,13
SqueezeNet-DSC	88,51	295020	0,30	2,34 ± 0,26	9,69 ± 0,16
SqueezeNet-Shift	89,92	293276	0,31	2,66 ± 0,23	**5,44 ± 0,15**
SqueezeNet-Shuffle (g = 2)	**90,23**	**160924**	**0,56**	2,68 ± 0,27	10,27 ± 0,14

Fonte: A autora (2019).

Figura 25 – Gráficos dos resultados das redes Squeeze sobre a (a) acurácia; e (b) o tempo de inferência em CPU.



Fonte: A autora (2019).

Nas tabelas 1 e 2 e nos gráficos da figura 25 estão listados os resultados obtidos pelo modelo SqueezeNet e suas variações: a SqueezeNet-DSC, a SqueezeNet-*Shift* e a SqueezeNet-*Shuffle*.

Considerando a base de dados CIFAR 10, a rede base obteve a melhor taxa de acerto, no valor de 90,11%. Apesar desse resultado, as taxas de acerto das redes otimizadas não ficaram mais do que 2,61% abaixo deste valor. No entanto, quando se analisa a eficiência das redes com relação ao número de parâmetros, E_P, dada pela razão entre a taxa de acerto e o número de parâmetros, o pior desempenho foi da SqueezeNet original. A rede mais eficiente, por um fator de cerca de 6,42×, foi a SqueezeNet-*Shuffle*.

Analogamente, na base de dados CIFAR 100 a melhor taxa de acerto foi obtida pelo modelo SqueezeNet-*Shuffle*, um valor de 90,23%, e as demais redes não ficaram mais do que 1,72% abaixo deste valor. Da mesma forma, a SqueezeNet-*Shuffle* foi a arquitetura mais eficiente com relação a taxa de acerto, no entanto, o ganho frente a rede menos eficiente reduz-se para 4,67×.

Considerando o tempo de inferência em GPU, os resultados obtidos para as bases CIFAR foram bastante próximos. Utilizando os intervalos de confiança com nível de 95% a partir das médias e dos desvios padrão apresentados na tabela 1 e na tabela 2, tanto para a base CIFAR 10 quanto para a base CIFAR 100, a rede mais rápida é a SqueezeNet. Com o mesmo procedimento, para a inferência em CPU, a menor latência foi obtida pela SqueezeNet-*Shift*.

5.1.2 Arquitetura ResNet

Nas tabelas 3 e 4 e nos gráficos da figura 26 estão listados resultados obtidos nos experimentos com os modelos ResNets 20, 56 e 110 e para as suas otimizações: a ResNet-DSC, a ResNet-*Shift* e a ResNet-*Shuffle*.

Tabela 3 – Resultados do modelo ResNet e suas variações na base CIFAR 10.

	Taxa acerto Top 1 (%)	Número de Parâmetros	E_P ($\times 10^{-3}$)	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20	**86,47**	272762	0,32	**1,71 ± 0,08**	2,94 ± 0,10
ResNet20-DSC	80,37	40810	1,97	2,21 ± 0,07	5,28 ± 0,06
ResNet20-Shift	80,81	35194	2,29	1,92 ± 0,06	**2,07 ± 0,11**
ResNet20-Shuffle (g = 2)	81,94	**24682**	**3,29**	2,55 ± 0,07	2,61 ± 0,09
ResNet56	**88,79**	856058	0,10	**4,45 ± 0,06**	8,01 ± 0,10
ResNet56-DSC	82,51	120106	0.68	5,83 ± 0,14	17,32 ± 0,20
ResNet56-Shift	84,05	102394	0,82	4,95 ± 0,04	**5,19 ± 0,11**
ResNet56-Shuffle (g = 2)	83,36	**67786**	**1,23**	7,00 ± 0,08	7,06 ± 0,12
ResNet110	**88,99**	1731002	0,05	**7,32 ± 0,16**	16,05 ± 1,02
ResNet110-DSC	83,34	239050	0,35	11,50 ± 0,41	36,83 ± 1,22
ResNet110-Shift	85,02	203194	0,41	9,56 ± 0,20	**9,90 ± 0,08**
ResNet110-Shuffle (g = 2)	83,19	**132442**	**0,63**	13,62 ± 0,20	13,72 ± 0,16

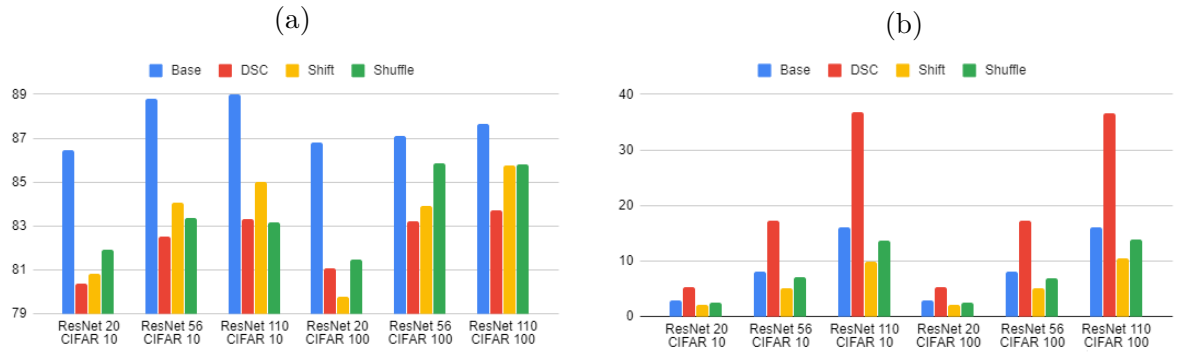
Fonte: A autora (2019).

Tabela 4 – Resultados do modelo ResNet e suas variações na base CIFAR 100.

	Taxa acerto Top 1 (%)	Número de Parâmetros	E_P ($\times 10^{-3}$)	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20	**86,78**	278612	0,31	**1,68 ± 0,04**	2,88 ± 0,08
ResNet20-DSC	81,08	46660	1,74	2,18 ± 0,07	5,29 ± 0,11
ResNet20-Shift	79,80	41044	1,94	2,00 ± 0,07	**2,08 ± 0,08**
ResNet20-Shuffle (g = 2)	81,48	**30532**	**2,67**	2,58 ± 0,08	2,58 ± 0,07
ResNet56	**87,10**	861908	0,10	**4,46 ± 0,12**	8,04 ± 0,09
ResNet56-DSC	83,20	125956	0,66	5,84 ± 0,09	17,36 ± 0,17
ResNet56-Shift	83,90	108244	0,78	4,94 ± 0,10	**5,19 ± 0,08**
ResNet56-Shuffle (g = 2)	85,85	**73636**	**1,16**	6,89 ± 0,14	7,00 ± 0,09
ResNet110	**87,65**	1736852	0,05	**7,41 ± 0,12**	16,09 ± 0,21
ResNet110-DSC	83,70	244900	0,34	11,53 ± 0,34	36,65 ± 0,75
ResNet110-Shift	85,76	209044	0,41	9,88 ± 0,22	**10,48 ± 0,28**
ResNet110-Shuffle (g = 2)	85,79	**138292**	**0,62**	13,77 ± 0,26	13,90 ± 0,14

Fonte: A autora (2019).

Figura 26 – Gráficos dos resultados das redes ResNet sobre (a) a acurácia; e (b) o tempo de inferência em CPU.



Fonte: A autora (2019).

Para a base de dados CIFAR 10, as melhores taxas de acerto foram 86,47%, 88,79% e 88,99% considerando as redes ResNet 20, 56 e 110, respectivamente. As taxas de acerto das redes otimizadas ficaram entre 3,97 e 6,28% abaixo destes valores. Ao analisar-se a eficiência das redes com relação ao número de parâmetros, E_P, os piores desempenhos foram obtidos pelas redes ResNets originais. As redes mais eficientes foram as ResNets-*Shuffle*.

Analogamente, para a base de dados CIFAR 100, as melhores taxas de acerto obtidas foram 86,78%, 87,10% e 87,65% nas ResNets 20, 56 e 110, respectivamente, e as outras arquiteturas não ficaram mais do que 6,8%, 3,9% ou 3,97% abaixo dessas acurácias. Novamente as redes mais eficientes com relação a taxa de acerto foram arquiteturas do tipo ResNet-*Shuffle*.

Considerando os tempos de inferência em GPU, os resultados obtidos para ambas as bases CIFAR foram bastante próximos. Utilizando os intervalos de confiança com nível de 95% a partir das médias e dos desvios padrão apresentados na tabela 3 e na tabela 4, tanto para a base CIFAR 10 quanto para a base CIFAR 100, as redes mais rápidas são as ResNet originais. Com o mesmo procedimento, para a inferência na CPU, as redes mais rápidas obtidas são as ResNet-*Shift*.

5.1.3 Considerações

Os resultados anteriores demonstram que uma rede, independente de seu tamanho, ao ser otimizada geralmente apresenta queda na acurácia. Os parâmetros de uma rede são os responsáveis pela representação dos dados em estudo. Em princípio, quanto mais parâmetros uma rede possui, mais fielmente ela pode se adequar à distribuição das informações que são apresentadas durante seu treinamento. Ao aplicar técnicas de otimização de redes neurais profundas, o primeiro efeito a ser observado é uma queda na acurácia uma vez que a rede otimizada possui menos parâmetros.

O número de parâmetros, no entanto, não é o único fator a influenciar a acurácia da rede. A distribuição e organização deles, ou seja, a definição da arquitetura da rede profunda, também tem influência em sua performance. Prova disso é que as redes mais compactas, obtidas nas otimizações por *Shuffle*, apresentaram acurácias maiores com relação a outras redes com mais parâmetros do que ela. Por exemplo, a SqueezeNet-*Shuffle* foi a rede que apresentou os melhores resultados na base de dados CIFAR 100 mesmo sendo a menor das redes Squeeze nessa base de dados. Outros exemplo são as redes *Shift*, as quais possuem menos parâmetros do que as redes DSC e, no entanto, na maioria dos casos ainda fornecem melhores resultados.

A quantidade de parâmetros, as operações realizadas e a plataforma na qual a rede está sendo testada influenciam os tempos de inferência. Ao se tomar as redes base ResNet20, ResNet56 e ResNet110 os módulos que as compões são os mesmos, apenas divergindo em quantidade de uma rede para a outra, e os tempos de inferência crescem diretamente quanto maior for a rede. Além disso, há operações que são mais ou menos paralelizáveis e operações que realizam mais ou menos FLOPs em relação à quantidade de acessos à memória. As operações mais paralelizáveis são mais velozes em GPU e as operações que demandam mais acessos à memória por FLOP tem seu tempo de processamento dominado pelas operações de leitura e escrita. As convoluções padrão com *kernels* maiores do que 1, presentes nas redes base, são operações bastante paralelizáveis e que se beneficiam do processamento em GPU, isso é verificado pelos resultados de latência dessas redes. As convoluções em profundidade são operações cuja latência é dominada pelo tempo despendido nos acessos à memória, sendo mais lentas e, por estarem presentes nas redes DSC e *Shuffle*, podem justificar o processamento mais lento delas. Essas operações, no entanto, também se beneficiam da paralelização do processamento. Observando as diferenças de

tempo de processamento nas tabelas 1 e 2, tanto o maior tempo de processamento que essas convoluções trazem às redes quanto o benefício que as GPUs fornecem são demonstrados pelas redes DSC e *Shuffle*. Nas tabelas que apresentam os resultados das ResNets os tempos de inferência também apontam o quanto as convoluções em profundidade se beneficiam das GPUs. Nessas tabelas, os tempos de processamento em CPU foram medidos levando em consideração o processamento distribuído, no qual uma camada de alto custo computacional dos módulos residuais é enviada para processamento em GPU, mas, enquanto que as redes DSC possuem duas convoluções em profundidade em seu módulo residual e apenas uma delas é enviada à GPU, as redes *Shuffle* possuem e enviam apenas uma camada. Dessa forma, o aumento de tempo de processamento devido à mudança de plataforma de teste é maior nas redes DSC.

Ademais, ao observar os tempos de inferência das tabelas 1 e 2, as redes *Shift* são as que menos apresentam diferença de latência entre as plataformas. Isso pode ser explicado pela ausência de operações de elevado custo computacional nessas redes. Sem convoluções pesadas computacionalmente e paralelizáveis que possam se beneficiar do processamento em GPU, o tempo de processamento das redes *Shift* tem pouca variação entre as plataformas de teste.

Assim sendo, não basta para o desenvolvedor escolher qualquer técnica de otimização. Diferentes otimizações alteram a arquitetura das redes de maneiras distintas e essas modificações podem ser melhores ou piores com relação aos efeitos sobre as acurácias e as latências das redes.

5.2 EXPANSÃO DAS RESNETS OTIMIZADAS

Supondo que uma rede base pudesse ser substituída por outras redes com o mesmo número de parâmetros mas formadas pelos módulos de otimização em análise, essas novas redes teriam vantagens em relação à acurácia e ao tempo de processamento? Essa é a pergunta que se deseja responder com os resultados apresentados nesta seção. Para tal, as redes ResNet otimizadas são expandidas até atingirem o número de parâmetros das redes base.

Nas tabelas 5 e 6 e nos gráficos da figura 27 estão listados os resultados obtidos pelos modelos ResNets-DSC, *Shift* e *Shuffle* expandidos. Nas tabelas, os valores nos nomes de cada rede corresponde às taxas de redução que, por estarem no intervalo entre 0 e 1, efetivamente definem suas taxas de expansão.

Considerando a base de dados CIFAR 10, as melhores taxas de acerto foram obtidas pela estratégia *Shuffle*, 89,02% e 89,78%, nas ResNets 20 e 56. Na ResNet 110, a melhor acurácia foi de 88,81% pela estratégia *Shift*, tendo a variação *Shuffle* ficado menos de 1% abaixo desta. De forma geral, as redes otimizadas não diferiram mais do que 2,44% em acurácia.

Analogamente, para a base CIFAR 100 as melhores taxas de acerto obtidas foram pela estratégia *Shuffle*, 89,63% e 89,97%, nas ResNets 20 e 56. Para a ResNet 110, a melhor acurácia foi de 90,23% pela estratégia *Shift*, tendo a variação *Shuffle* ficado menos de 2% abaixo deste valor. De forma geral, as redes otimizadas não diferiram mais do que 3,51% em acurácia.

Considerando os tempos de inferência em GPU das redes otimizadas, os resultados obtidos para ambas as bases CIFAR foram bastante próximos. Utilizando os intervalos de confiança com nível de 95% a partir das médias e dos desvios padrão apresentados na tabela 5 e na tabela 6, tanto para a base CIFAR 10 quanto para a base CIFAR 100, as redes mais rápidas são as ResNets-*Shift*. Com o mesmo procedimento, considerando inferência na CPU, as redes mais rápidas também são as ResNets-*Shift*.

5.2.1 Considerações

Considerando que em aplicações práticas uma diferença de até 5% na taxa de acerto é aceitável, as redes otimizadas são equivalentes às redes base[2]. Ao comparar as acurácias das redes expandidas com as ResNets base, para as duas bases de dados, os resultados obtidos ficaram todos dentro do intervalo de 86,12% a 90,23%, um intervalo de apenas

[2] Este foi um parâmetro empírico definido nesta dissertação com base em trabalhos de outros pesquisadores da área.

Tabela 5 – Resultados dos modelos ResNets otimizados e expandidos na base base CIFAR 10.

	Taxa acerto Top 1 (%)	Número de Parâmetros	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20	86,47	271762	**1,71 ± 0,08**	**2,94 ± 0,10**
ResNet20-DSC-0.36	86,58	270684	2,15 ± 0,05	12,94 ± 0,21
ResNet20-Shift-0.35	87,55	270101	1,96 ± 0,07	3,37 ± 0,10
ResNet20-Shuffle-0.27	**89,02**	**262960**	2,56 ± 0,07	4,88 ± 0,04
ResNet56	88,79	856058	**4,45 ± 0,06**	**8,01 ± 0,10**
ResNet56-DSC-0.35	87,78	**849176**	5,55 ± 0,10	45,03 ± 1,24
ResNet56-Shift-0.34	88,72	852966	5,00 ± 0,11	8,55 ± 0,09
ResNet56-Shuffle-0.25	**89,78**	864522	6,90 ± 0,13	13,50 ± 0,11
ResNet110	88,99	1731002	**7,32 ± 0,16**	**16,05 ± 1,02**
ResNet110-DSC-0.35	87,14	**1693880**	10,96 ± 0,16	94,39 ± 2,03
ResNet110-Shift-0.34	**88,81**	1699812	9,81 ± 0,13	16,24 ± 0,18
ResNet110-Shuffle-0.25	88,18	1699146	13,03 ± 0,22	25,91 ± 0,21

Fonte: A autora (2019).

Tabela 6 – Resultados dos modelos ResNets otimizados e expandidos na base CIFAR 100.

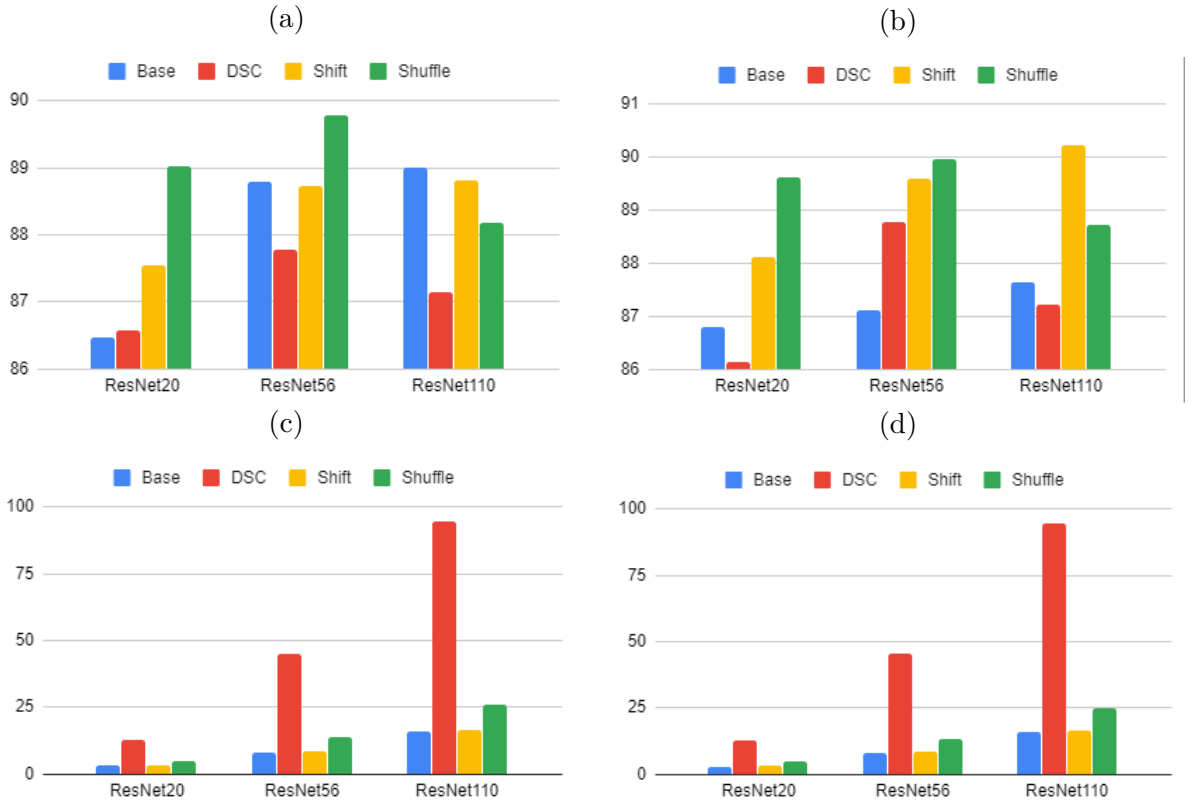
	Taxa acerto Top 5 (%)	Número de Parâmetros	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20	86,78	278612	**1,68 ± 0,04**	**2,88 ± 0,08**
ResNet20-DSC-0.37	86,12	272677	2,13 ± 0,09	12,58 ± 0,21
ResNet20-Shift-0.36	88,12	**271215**	1,94 ± 0,06	3,30 ± 0,12
ResNet20-Shuffle-0.27	**89,63**	284290	2,53 ± 0,05	4,74 ± 0,09
ResNet56	87,10	861908	**4,46 ± 0,12**	**8,04 ± 0,09**
ResNet56-DSC-0.35	88,77	**865646**	5,64 ± 0,10	45,29 ± 0,81
ResNet56-Shift-0.34	89,60	869976	5,11 ± 0,09	8,59 ± 0,10
ResNet56-Shuffle-0.25	**89,97**	887652	6,77 ± 0,12	13,05 ± 0,17
ResNet110	87,65	1736852	**7,41 ± 0,12**	**16,09 ± 0,21**
ResNet110-DSC-0.35	87,21	**1710350**	10,81 ± 0,19	94,71 ± 1,89
ResNet110-Shift-0.34	**90,23**	1716822	9,85 ± 0,25	16,19 ± 0,13
ResNet110-Shuffle-0.25	88,72	1722276	13,18 ± 0,38	24,93 ± 0,26

Fonte: A autora (2019).

4,11%. A rede base que teve a maior variação com relação a suas otimizações expandidas, a rede ResNet56 na base CIFAR 100, apresentou uma diferença máxima de 2,87%.

Com relação ao tempo de inferência, no entanto, todos os resultados obtidos foram favoráveis às redes originais. No processamento em GPU as redes otimizadas já eram mais lentas do que as redes base antes de serem expandidas. Com o aumento do número de canais ao longo da rede esse fato não sofreu alteração. As GPUs no entanto demonstram seu poder de paralelização ao fornecerem tempo de inferência das redes expandidas que não diferem mais de 0,64ms das redes otimizadas. O processamento em CPU no entanto, salvo no caso das redes *Shift*, é bastante afetado, uma vez que com mais canais há mais dados a serem calculados nas convoluções. As redes com módulos *Shift* são as menos afetadas pois esses módulos são compostos apenas por convoluções unitárias e que dependem menos da paralelização do processamento.

Figura 27 – Gráficos dos resultados das redes ResNet otimizadas e expandidas sobre (a) e (b) a acurácia na CIFAR 10 e na CIFAR 100, repectivamente; e (c) e (d) o tempo de inferência em CPU para as bases CIFAR 10 e CIFAR 100, respectivamente.



Fonte: A autora (2019).

5.3 REDUÇÃO DAS ARQUITETURAS RESNETS

Considerando que as ResNets base tivessem seu número de parâmetros reduzido ao patamar das redes otimizadas através da redução no número de canais ou do uso de seu módulo de *bottleneck*, as redes otimizadas teriam vantagens em relação à acurácia e ao tempo de processamento? Essa é a pergunta que se deseja responder com os resultados apresentados nesta seção.

Nas tabelas 7 e 8, 9 e 10 e 11 e 12 estão listados os resultados obtidos para as ResNets20, ResNets56 e ResNets110 reduzidas ao nível de número de parâmetros das ResNets-DSC, ResNets-*Shift* e ResNets-*Shuffle*, respectivamente. Os valores nos nomes de cada rede correspondem às taxas de redução utilizadas. Além disso, quando o nome da rede é formado pelo termo "block", significa que a rede é constituída por módulos *bottleneck*, caso contrário, a redução é feita pela diminuição do número de canais ao longo da arquitetura.

Em todos os testes desta seção, utilizando os intervalos de confiança com nível de 95% a partir das médias e dos desvios padrão apresentados nas tabelas 7, 8, 9, 10 11 e 12, tanto para a base CIFAR 10 quanto para a base CIFAR 100, as redes mais rápidas são as ResNets reduzidas pelo número de canais.

5.3.1 Redução ao nível das Redes DSC

Nas tabelas 7 e 8, para as duas bases de dados, as melhores taxas de acerto obtidas foram das redes ResNet-*bottleneck*. Apesar desse resultado, as taxas de acerto das ResNets-DSC não ficaram mais do que 2,45% abaixo deste valor.

Os piores tempos de inferência obtidos foram decorrentes das redes otimizadas pela DSC. Esses resultados, juntamente com os das tabelas 1, 2, 3, 4, 5 e 6 evidenciam que a substituição de convoluções padrão por DSC, apesar de reduzir o número de parâmetros de uma camada, eleva consistentemente o tempo de processamento.

Dessa forma, conclui-se que as redes DSC são equivalentes às redes *bottleneck* com relação à acurácia, porém entre as redes consideradas as DSC são mais lentas.

5.3.2 Redução ao nível das Redes *Shift*

Nas tabelas 9 e 10, para as duas bases de dados, as melhores taxas de acerto obtidas se dividem entre as redes ResNet-*Shift* e as ResNets-*bottleneck*.

Não obstante as menores latências não serem das ResNets-*Shift*, seus tempos de processamento foram melhores do que os das ResNets-*bottleneck*. Esses resultados, juntamente com os das tabelas 1, 2, 3, 4, 5 e 6 evidenciam que a substituição de convoluções padrão por operações de deslocamento de canais seguidas por convoluções pontuais, além de reduzir o número de parâmetros de uma camada, não afetam de maneira considerável os tempos de processamento.

Dessa forma, conclui-se que as redes *Shift* são equivalentes às redes *bottleneck* com relação à acurácia e superiores a elas com relação aos tempos de processamento.

Tabela 7 – Resultados dos modelos ResNets reduzidas ao tamanho da ResNet-DSC na base CIFAR 10.

	Taxa de acerto Top 1 (%)	Número de Parâmetros	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20-DSC	80,37	**40810**	2,21 ± 0,07	5,28 ± 0,06
ResNet20-2.5	79,20	41341	**1,53 ± 0,05**	**1,83 ± 0,08**
ResNet20-2.4_block	**82,30**	41169	2,12 ± 0,10	2,13 ± 0,06
ResNet56-DSC	82,51	120106	5,83 ± 0,14	17,32 ± 0,20
ResNet56-2.6	82,68	121648	**3,81 ± 0,08**	**4,58 ± 0,10**
ResNet56-2.4_block	**84,96**	**117279**	5,60 ± 0,09	5,73 ± 0,11
ResNet110-DSC	83,34	**239050**	11,50 ± 0,41	36,83 ± 1,22
ResNet110-2.6	83,51	245632	**7,35 ± 0,17**	**9,06 ± 0,14**
ResNet110-2.3_block	**84,96**	242374	10,85 ± 0,19	11,08 ± 0,11

Fonte: A autora (2019).

Tabela 8 – Resultados dos modelos ResNets reduzidas ao tamanho da ResNet-DSC na base CIFAR 100.

	Taxa de acerto Top 5 (%)	Número de Parâmetros	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20-DSC	81,08	**46660**	2,18 ± 0,07	5,29 ± 0,11
ResNet20-2.4	76,59	47669	**1,54 ± 0,04**	**1,76 ± 0,09**
ResNet20-2.4_block	**81,09**	47019	2,10 ± 0,06	2,20 ± 0,09
ResNet56-DSC	83,20	125956	5,84 ± 0,09	17,36 ± 0,17
ResNet56-2.6	80,69	123898	**3,92 ± 0,07**	**4,63 ± 0,08**
ResNet56-2.4_block	**84,14**	**123129**	5,61 ± 0,12	5,75 ± 0,12
ResNet110-DSC	83,70	**244900**	11,53 ± 0,34	36,65 ± 0,75
ResNet110-2.6	82,09	247882	**7,37 ± 0,26**	**8,91 ± 0,10**
ResNet110-2.3_block	**85,87**	248224	10,61 ± 0,32	11,02 ± 0,14

Fonte: A autora (2019).

Tabela 9 – Resultados dos modelos ResNets reduzidos ao tamanho da ResNet-*Shift* na base CIFAR 10.

	Taxa de acerto Top 1 (%)	Número de Parâmetros	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20-Shift	**80,81**	35194	1,92 ± 0,06	2,07 ± 0,11
ResNet20-2.7	76,86	34636	**1,52 ± 0,06**	**1,71 ± 0,10**
ResNet20-2.7_block	79,91	**34031**	2,10 ± 0,05	2,13 ± 0,10
ResNet56-Shift	84,05	102394	4,95 ± 0,04	5,19 ± 0,11
ResNet56-2.8	80,20	**101429**	**3,82 ± 0,06**	**4,50 ± 0,10**
ResNet56-2.6_block	**84,49**	104246	5,53 ± 0,07	5,66 ± 0,08
ResNet110-Shift	85,02	**203194**	9,56 ± 0,20	9,90 ± 0,08
ResNet110-2.8	81,91	204857	**7,25 ± 0,18**	**8,81 ± 0,10**
ResNet110-2.6_block	**85,19**	205298	10,82 ± 0,18	11,05 ± 0,13

Fonte: A autora (2019).

5.3.3 Redução ao nível das Redes *Shuffle*

Nas tabelas 11 e 12, para as duas bases de dados, as melhores taxas de acerto obtidas foram das redes ResNet-*Shuffle*. Esse resultado reitera a eficiência das redes *Shuffle* em arquiteturas com um número bastante reduzido de parâmetros. Não obstante, os piores tempos de inferência obtidos foram decorrentes das redes otimizadas pelo módulo *Shuffle*. Esses resultados, juntamente com os das tabelas 1, 2, 3, 4, 5 e 6 evidenciam, da mesma

Tabela 10 – Resultados dos modelos ResNets reduzidos ao tamanho da ResNet-*Shift* na base CIFAR 100.

	Taxa de acerto Top 5 (%)	Número de Parâmetros	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20-Shift	79,80	**41044**	2,00 ± 0,07	2,08 ± 0,08
ResNet20-2.6	75,82	41242	**1,52 ± 0,03**	**1,69 ± 0,04**
ResNet20-2.6_block	**81,07**	42728	2,11 ± 0,08	2,23 ± 0,07
ResNet56-Shift	83,90	**108244**	4,94 ± 0,10	5,19 ± 0,08
ResNet56-2.7	78,42	110632	**3,77 ± 0,09**	**4,56 ± 0,11**
ResNet56-2.6_block	**84,49**	110096	5,46 ± 0,06	5,73 ± 0,14
ResNet110-Shift	**85,76**	209044	9,88 ± 0,22	10,48 ± 0,28
ResNet110-2.8	80,31	**206927**	**7,27 ± 0,24**	**8,77 ± 0,08**
ResNet110-2.6_block	84,54	211148	10,77 ± 0,18	11,05 ± 0,10

Fonte: A autora (2019).

maneira que para a otimização por DSC, que as convoluções em profundidade, apesar de reduzir o número de parâmetros de uma camada, elevam o tempo de processamento.

Dessa forma, frente às outras formas de redução da ResNet, conclui-se que as redes *Shuffle* têm vantagens com relação a acurácia mas não em relação aos tempos de inferência.

Tabela 11 – Resultados dos modelos ResNets reduzidas ao tamanho da ResNet-*Shuffle* na base CIFAR 10.

	Taxa de acerto Top 1 (%)	Número de Parâmetros	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20-Shuffle	**81,94**	24682	2,55 ± 0,07	2,61 ± 0,09
ResNet20-3.25	74,02	**23604**	**1,54 ± 0,06**	**1,67 ± 0,10**
ResNet20-3.4_block	79,14	24869	2,05 ± 0,06	2,08 ± 0,05
ResNet56-Shuffle	**83,36**	67786	7,00 ± 0,08	7,06 ± 0,12
ResNet56-3.5	77,59	68037	**3,86 ± 0,08**	**4,52 ± 0,07**
ResNet56-3.5_block	82,49	**67739**	5,49 ± 0,09	5,52 ± 0,12
ResNet110-Shuffle	**83,19**	132442	13,62 ± 0,20	13,72 ± 0,16
ResNet110-3.5	79,96	137355	**7,49 ± 0,17**	**8,56 ± 0,12**
ResNet110-3.5_block	82,75	**132044**	10,87 ± 0,20	10,75 ± 0,08

Fonte: A autora (2019).

Tabela 12 – Resultados dos modelos ResNets reduzidos ao tamanho da ResNet-*Shuffle* na base CIFAR 100.

	Taxa de acerto Top 5 (%)	Número de Parâmetros	Tempo GPU (ms±Δ)	Tempo CPU (ms±Δ)
ResNet20-Shuffle	**81,48**	**30532**	2,58 ± 0,08	2,58 ± 0,07
ResNet20-3	71,44	31169	**1,53 ± 0,05**	**1,71 ± 0,10**
ResNet20-3.5_block	77,88	30719	2,06 ± 0,05	2,10 ± 0,08
ResNet56-Shuffle	**87,10**	73636	6,59 ± 0,14	7,00 ± 0,09
ResNet56-3.3	75,68	75636	**3,94 ± 0,10**	**4,45 ± 0,10**
ResNet56-3.5_block	83,28	**73589**	5,45 ± 0,09	5,53 ± 0,10
ResNet110-Shuffle	**85,79**	138292	13,77 ± 0,26	13,90 ± 0,14
ResNet110-3.5	77,61	139065	**7,40 ± 0,19**	**8,52 ± 0,08**
ResNet110-3.5_block	83.35	**137894**	10,68 ± 0,37	10,74 ± 0,16

Fonte: A autora (2019).

5.4 ARQUITETURA PIX-2-PIX

As redes GANs foram treinadas para gerarem imagens de mapas a partir de imagens de satélites. Os resultados obtidos no cálculo das taxas de acerto pixel-a-pixel médias com seus intervalos de confiança e seus resultados perceptuais são listados na tabela 13. O intervalo de confiança considerado é baseado em um nível de certeza de 95%.

Tabela 13 – Resultados dos modelos GANs.

	Acurácia pixel-a-pixel (%)	Taxa de erro perceptual	Número de Parâmetros ($\times 10^6$)
Pix-2-Pix	62,18 ± 0,58	0,20	54,41
Pix-2-Pix-DSC	57,35 ± 0,54	0,15	36,14
Pix-2-Pix-Shift	60,40 ± 0,61	0,20	36,06
Pix-2-Pix-Shuffle (g = 8)	51,59 ± 0,62	0,10	35,60

Fonte: A autora (2019).

As imagens foram avaliadas de forma empírica. Um site web foi construído, e voluntários puderam avaliar as imagens considerando sua qualidade. O site para a realização do estudo perceptual foi distribuído para voluntários. Dessa forma, o número de voluntários que responderam à pesquisa foi de 45 voluntários.

No estudo perceptual foram selecionadas imagens geradas a partir do conjunto de teste da base de dados *maps↔satellite*. Para cada rede, 25 imagens reais e 25 imagens sintéticas foram selecionadas de forma aleatória. O ordenamento das imagens na sequência

Figura 28 – Imagens geradas pelas redes (a) Pix-2-Pix; (b) Pix-2-Pix-DSC; (c)Pix-2-Pix-*Shift*; e (d) Pix-2-Pix-*Shuffle*

Fonte: A autora (2019).

de testes é feito de forma que das 10 primeiras imagens apresentadas 5 sejam sintéticas. Da mesma forma das 40 imagens seguintes, 20 serão sintéticas e as outras 20 serão reais. O ordenamento da apresentação das imagens ao usuário entre reais ou sintéticas é aleatório.

Para o cálculo da taxa de erro perceptual são utilizadas as respostas obtidas sobre as imagens sintéticas. Das 25 imagens sintéticas apresentadas ao usuário, 5 fazem parte das imagens de treinamento do voluntário e seus resultados são descartados. A taxa de erro perceptual de uma rede é calculado como a média dos erros cometidos em cada imagem e, por sua vez, a percepção de uma imagem como real ou sintética é medida pela maioria dos votos dados pelos voluntários. Cada rede foi testada por ~ 10 voluntários. Pela tabela 13, as redes que melhor foram avaliadas são a Pix-2-Pix e a Pix-2-Pix-*Shift*.

Também considerando a acurácia pixel-a-pixel, essas duas redes foram as mais bem avaliadas. Pela sobreposição dos intervalos de confiança da taxa de acerto pixel-a-pixel das redes Pix-2-Pix e Pix-2-Pix-*Shift*, suas acurácias são consideradas equivalentes.

A otimização pelo módulo *Shuffle* gerou a rede com menor número de parâmetros, no entanto, as perdas na taxa de acerto pixel-a-pixel e no erro perceptual sugerem que essa otimização não seja adequada para a rede generativa. Já as otimização por DSC e pelo módulo *Shift* tiveram menos perdas ao passo que reduziram o número de parâmetros da rede generativa para aproximadamente 66,35% do tamanho original.

A acurácia pixel-a-pixel alcançada pela rede otimizada por deslocamento de canais pode ser devido ao fato que essa operação realiza o deslocamento de informações estruturais que podem favorecer a geração da imagem de saída. Uma vez que ao longo da rede generativa da Pix-2-Pix há a passagem de informações estruturais da imagem entre as etapas de codificação e decodificação, a otimização por deslocamento de canais abrange a área considerada na determinação dos pixels de saída.

Por outro lado a redução de parâmetros alcançada pelas redes Pix-2-Pix-*Shuffle* e Pix-2-Pix-DSC podem ter comprometido a codificação das informações da imagem de entrada e assim a determinação da imagem de saída é comprometida.

Na figura 28 são apresentadas imagens sintéticas geradas pelas redes Pix-2-Pix e suas variações DSC, *Shift* e *Shuffle*.

Pela quantidade de pessoas alcançadas pelo estudo perceptual conclusões acerca dos resultados obtidos não são estatisticamente significativos e necessitam de maiores testes. No entanto, a criação do site com a implementação do fluxo de avaliação perceptual da forma como definido facilita a realização da avaliação de trabalhos futuros acerca de redes GANs ou outras redes que realizem a geração de imagens.

5.5 CONSIDERAÇÕES E DIRETRIZES AO DESENVOLVEDOR

Nas seções anteriores as técnicas de otimização foram analisadas a fim de definir vantagens e desvantagens atreladas a cada uma delas. Dos resultados expostos pode-se concluir para as redes de classificação:

1. A mistura de canais é a técnica que oferece a maior eficiência, considerando a acurácia e o número de parâmetros, mas não influenciam positivamente na latência dos modelos;

2. Dentre as técnicas de otimização analisadas, o deslocamento de canais é a que oferece redes com menor latência; e

3. As redes DSCs, apesar de reduzirem o número de parâmetros e melhorarem a eficiência, não influenciam positivamente na latência da rede;

Os itens acima são inferidos dos resultados obtidos nas redes de classificação otimizadas. A partir deles e das considerações feitas no decorrer deste capítulo, definem-se diretrizes a serem utilizadas por um desenvolvedor que deseja aplicar uma das técnicas de otimização dentre a estratégia DSCs e os módulos *Shift* e *Shuffle*.

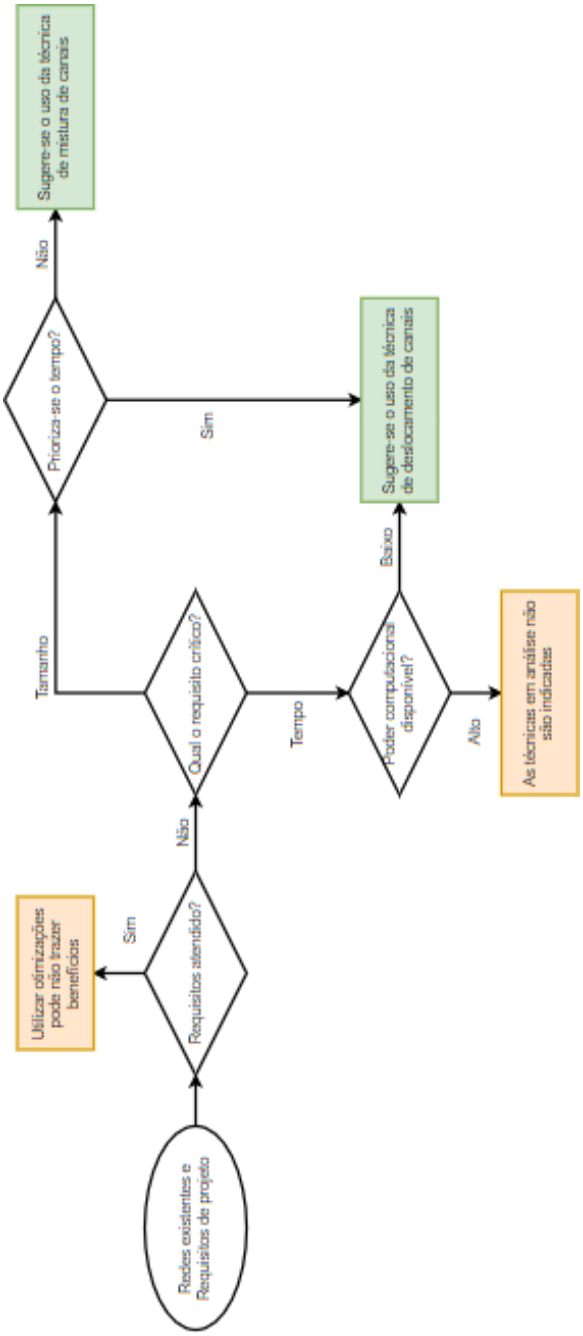
A primeira consideração que o desenvolvedor precisa fazer é se ele realmente necessita realizar a otimização das redes ao seu dispor. Para tal ele precisa saber se a rede que ele tem atende aos requisitos de projeto na plataforma de implantação. No caso da rede atender aos requisitos, alterações que sejam realizadas focando tão somente uma menor latência ou menor espaço de armazenamento podem não trazer benefícios significativos ao processo. No caso dos requisitos de projeto não serem atendidos, qual deles é mais importante de ser endereçado?

No caso de se precisar de uma rede mais veloz, mas esta rede já é utilizada em ambientes de alto poder de processamento, as técnicas em análise não são indicadas. Em lugar delas, o desenvolvedor deve procurar outros meios de modificar sua rede para atender os requisitos de seu projeto.

No caso da rede necessitar ser mais veloz e ser utilizada em ambiente de baixo poder de processamento, dentre as técnicas analisadas, se sugere o uso do módulo *Shift*. Contudo, a rede otimizada deve passar pelo processo de treinamento, teste e validação dos requisitos do processo. Além disso, ainda não está disponível nos frameworks normalmente utilizados a implementação da operação de deslocamentos de canais para CPUs, logo se o desenvolvedor necessitar utilizá-la ele deve ter em mente que ou seu ambiente de *deploy* deve ter uma GPU, mesmo que não seja com grande poder de processamento, ou será necessário implementar a operação de deslocamento de canais.

No caso da rede precisar ter seu tamanho reduzido, sugere-se três estratégias. Visto que todas as técnicas reduzem o tamanho da rede, inicialmente o desenvolvedor pode avaliar o uso do módulo *Shift*, seguida pela DSC e, por último o módulo *Shufle*. Essa sequência é assim definida pois, a primeira técnica reduz o tamanho da rede e influencia positivamente no tempo de inferência, as técnicas seguintes reduzem o tamanho da rede, mas ao custo de influenciarem negativamente sua latência. Logo, caso a rede também necessite ter sua latência controlada, apenas os módulos *Shift* são indicados.

Figura 29 – Fluxograma de diretrizes ao desenvolvedor.



Fonte: A autora (2019).

As diretrizes acima são formuladas com vistas para as redes de classificação e podem ser resumidas pelo fluxograma da figura 29. Em relação às GANs, como os resultados obtidos são poucos, não correspondem diretamente aos resultados das redes de classificação e o comportamento delas ainda pode ser de difícil controle, mais estudos são necessário antes da formulação de diretrizes de otimização direcionadas a estas arquiteturas. No entanto, caso queira, o desenvolvedor pode utilizar as mesmas diretrizes sugeridas para a classificação que benefícios podem ser obtidos, porém ainda não existe um consenso.

6 CONCLUSÃO

Hoje as redes neurais artificiais, em especial as Redes Neurais Convolucionais (CNNs), são utilizadas na solução de problemas de classificação e segmentação de imagens, localização e e detecção de objetos, transferência de estilo, processamento de linguagem natural, entre outros com resultados empíricos muito significativos. A utilização dessas redes pode ocorrer tanto em ambientes que possuem elevado poder computacional quanto em ambientes com poucos recursos. Em plataformas com pouco recursos, no entanto, o processo de inferência nas redes profundas é um desafio e, com o objetivo de facilitar o uso de CNNs nessas plataformas surgiram duas vertentes de pesquisa. Delas derivam técnicas que podem ser utilizadas antes da definição da arquitetura da rede e outras podem ser utilizadas no decorrer ou após o final do treinamento do modelo. Porém, durante o processo de revisão da literatura, não encontramos um estudo que compare diversas técnicas de otimização e que possa guiar um desenvolvedor/projetista na escolha nas melhores técnicas de otimização que possa atender as necessidades de projeto.

Esta dissertação realizou um estudo comparativo dos efeitos da utilização de diferentes técnicas de otimização nos modelos do tipo CNNs. Os efeitos analisados se referem ao número de parâmetros e ao tempo de inferência, ao passo que não se deseja que a assertividade seja consideravelmente impactada pelo uso da otimização da arquitetura ou processo de treinamento. Desta forma, foi realizada a implementação da Convolução Separável em Profundidade, e dos módulos de Mistura e de Deslocamento de Canais nas redes SqueezeNet, ResNet e Pix-2-Pix. Além disso, também é proposta a adaptação das redes SqueezeNet e ResNet para os tratamento dos dados das bases CIFAR 10 e CIFAR 100. Os resultados obtidos formam um conjunto de referência que futuros desenvolvedores podem usar como guia na escolha entre as técnicas de otimização aqui apresentadas. Como principais observações sobre as técnicas de otimização estudadas tem-se: a mistura de canais é a técnica que oferece a maior eficiência, considerando a acurácia e o número de parâmetros, mas não influenciam positivamente na latência dos modelos; dentre as técnicas de otimização analisadas, o deslocamento de canais é a que oferece redes com menor latência; e as redes DSCs, apesar de reduzirem o número de parâmetros e melhorarem a eficiência, não influenciam positivamente na latência da rede.

6.1 TRABALHOS FUTUROS

A partir do escopo desta dissertação, alguns trabalhos futuros são propostos: a implementação da técnica de deslocamento de canais para CPU; a investigação do efeito de técnicas de otimização em redes generativas adversárias através de outras redes e outras bases de dados; a análise do desempenho das técnicas investigadas quando testadas em

dispositivos móveis como celulares, por exemplo.

A implementação da operação de deslocamentos de canais para CPU é sugerida a fim de que essa técnica seja utilizada independentemente do *hardware* de implantação utilizado. Com essa implementação, a técnica de otimização por deslocamento de canais poderia ser testada e comparada a outras técnica sem o uso do processamento distribuído entre CPU e GPUs.

A investigação de técnicas de otimização em redes GANs utilizando outras arquiteturas e outras bases de dados fornecerão mais informações acerca dos efeitos que as técnicas analisadas geram sobre as redes base. Visto que as pesquisas sobre rede generativas estão em evidência, permitir que essas redes sejam criadas de maneira mais compacta favorecerá a disseminação de aplicações pautadas nesse conceito.

Os testes realizados neste trabalho consideraram o processamento de redes em GPU e em CPU para a comparação de técnicas de otimização. Além dessas plataformas, realizar os testes em dispositivos móveis permitiria uma análise mais completa e fiel dos efeitos causados pela utilização de diferentes técnicas de otimização. Uma vez que os dispositivos móveis hoje fazem parte do nosso cotidiano e que já há muitos aplicativos utilizando redes profundas em seus processos, a busca de arquiteturas mais eficientes nesses ambientes é uma preocupação real de desenvolvedores e a criação de uma documentação que compare diversas otimizações pelos seus efeitos poderia agilizar a pesquisa e o desenvolvimento de novos produtos bem como a atualização de antigos.

REFERÊNCIAS

BENGIO, Y.; LAMBLIN, P.; POPOVICI, D.; LAROCHELLE, H. Greedy layer-wise training of deep networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2007. p. 153–160.

BIANCO, S.; CADENE, R.; CELONA, L.; NAPOLETANO, P. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, IEEE, v. 6, p. 64270–64277, 2018.

BROWNLEE, J. *18 Impressive Applications of Generative Adversarial Networks (GANs)*. 2019. Accesso em: 19 de agosto de 2019. Disponível em: <https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/>.

CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 1251–1258.

CLAESEN, M.; MOOR, B. D. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: IEEE. *2009 IEEE conference on computer vision and pattern recognition*. [S.l.], 2009. p. 248–255.

ESTEVA, A.; KUPREL, B.; NOVOA, R. A.; KO, J.; SWETTER, S. M.; BLAU, H. M.; THRUN, S. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, Nature Publishing Group, v. 542, n. 7639, p. 115, 2017.

FAUST, O.; HAGIWARA, Y.; HONG, T. J.; LIH, O. S.; ACHARYA, U. R. Deep learning for healthcare applications based on physiological signals: A review. *Computer methods and programs in biomedicine*, Elsevier, v. 161, p. 1–13, 2018.

FUKUSHIMA, K. Neural network model for a mechanism of pattern recognition unaffected by shift in position — neocognitron —. *Trans. IECE (in Japanese)*, v. 10, n. J62-A, p. 658–665, 1975.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. [S.l.: s.n.], 2010. p. 249–256.

GONG, Y.; LIU, L.; YANG, M.; BOURDEV, L. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

GOODFELLOW, I.; POUGET-ABADIE, J.; MIRZA, M.; XU, B.; WARDE-FARLEY, D.; OZAIR, S.; COURVILLE, A.; BENGIO, Y. Generative adversarial nets. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2014. p. 2672–2680.

HAN, S.; MAO, H.; DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

HAN, S.; POOL, J.; TRAN, J.; DALLY, W. Learning both weights and connections for efficient neural network. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2015. p. 1135–1143.

HE, K.; SUN, J. Convolutional neural networks at constrained time cost. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 5353–5360.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1026–1034.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Identity mappings in deep residual networks. In: SPRINGER. *European conference on computer vision*. [S.l.], 2016. p. 630–645.

HEBB, D. O. *The organization of behavior*. [S.l.]: Wiley New York, 1949. v. 65.

HINTON, G.; DENG, L.; YU, D.; DAHL, G.; MOHAMED, A.-r.; JAITLY, N.; SENIOR, A.; VANHOUCKE, V.; NGUYEN, P.; KINGSBURY, B. et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, v. 29, 2012.

HINTON, G.; VINYALS, O.; DEAN, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *science*, American Association for the Advancement of Science, v. 313, n. 5786, p. 504–507, 2006.

HINTON, G. E.; SRIVASTAVA, N.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M.; ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

HU, J.; SHEN, L.; SUN, G. Squeeze-and-excitation networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2018. p. 7132–7141.

HUBARA, I.; COURBARIAUX, M.; SOUDRY, D.; EL-YANIV, R.; BENGIO, Y. Binarized neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2016. p. 4107–4115.

IANDOLA, F. N.; HAN, S.; MOSKEWICZ, M. W.; ASHRAF, K.; DALLY, W. J.; KEUTZER, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

IANDOLA, F. N.; MOSKEWICZ, M. W.; ASHRAF, K.; KEUTZER, K. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2016. p. 2592–2600.

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

ISOLA, P.; ZHU, J.-Y.; ZHOU, T.; EFROS, A. A. Image-to-image translation with conditional adversarial networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 1125–1134.

JADERBERG, M.; VEDALDI, A.; ZISSERMAN, A. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

JIN, J.; DUNDAR, A.; CULURCIELLO, E. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014.

KRIZHEVSKY, A.; HINTON, G. *Learning multiple layers of features from tiny images*. [S.l.], 2009.

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

LARSEN, A. B. L.; SØNDERBY, S. K.; LAROCHELLE, H.; WINTHER, O. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

LEBEDEV, V.; GANIN, Y.; RAKHUBA, M.; OSELEDETS, I.; LEMPITSKY, V. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989.

LECUN, Y.; DENKER, J. S.; SOLLA, S. A. Optimal brain damage. In: *Advances in neural information processing systems*. [S.l.: s.n.], 1990. p. 598–605.

LI, C.; WAND, M. Precomputed real-time texture synthesis with markovian generative adversarial networks. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2016. p. 702–716.

LINNAINMAA, S. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. Dissertação (Mestrado) — Univ. Helsinki, 1970.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. [S.l.: s.n.], 2010. p. 807–814.

OH, S. H. *Reconhecimento facial ajuda a prender criminoso no Carnaval de Salvador*. 2019. Accesso em: 17 de agosto de 2019. Disponível em: <https://canaltech.com.br/seguranca/reconhecimento-facial-ajuda-a-prender-criminoso-no-carnaval-de-salvador-134189/>.

QIU, J.; WANG, J.; YAO, S.; GUO, K.; LI, B.; ZHOU, E.; YU, J.; TANG, T.; XU, N.; SONG, S. et al. Going deeper with embedded fpga platform for convolutional neural network. In: ACM. *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. [S.l.], 2016. p. 26–35.

RASTEGARI, M.; ORDONEZ, V.; REDMON, J.; FARHADI, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2016. p. 525–542.

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: SPRINGER. *International Conference on Medical image computing and computer-assisted intervention*. [S.l.], 2015. p. 234–241.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

ROY, A.; SUN, J.; MAHONEY, R.; ALONZI, L.; ADAMS, S.; BELING, P. Deep learning detecting fraud in credit card transactions. In: IEEE. *2018 Systems and Information Engineering Design Symposium (SIEDS)*. [S.l.], 2018. p. 129–134.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. et al. Learning representations by back-propagating errors. *Cognitive modeling*, v. 5, n. 3, p. 1, 1988.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M. et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, Springer, v. 115, n. 3, p. 211–252, 2015.

SANTOS, A. G.; SOUZA, C. O. de; ZANCHETTIN, C.; MACEDO, D.; OLIVEIRA, A. L.; LUDERMIR, T. Reducing squeezenet storage size with depthwise separable convolutions. In: IEEE. *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2018. p. 1–6.

SIFRE, L.; MALLAT, S. Rigid-motion scattering for image classification. *Ph. D. dissertation*, Citeseer, 2014.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

SOUDRY, D.; HUBARA, I.; MEIR, R. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2014. p. 963–971.

SZE, V.; CHEN, Y.-H.; YANG, T.-J.; EMER, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, Ieee, v. 105, n. 12, p. 2295–2329, 2017.

SZEGEDY, C.; IOFFE, S.; VANHOUCKE, V.; ALEMI, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In: *Thirty-First AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2017.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 1–9.

SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J.; WOJNA, Z. Rethinking the inception architecture for computer vision. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 2818–2826.

TESAURO, G. Practical issues in temporal difference learning. In: *Advances in neural information processing systems*. [S.l.: s.n.], 1992. p. 259–266.

THING, V. L. Ieee 802.11 network anomaly detection and attack classification: A deep learning approach. In: IEEE. *2017 IEEE Wireless Communications and Networking Conference (WCNC)*. [S.l.], 2017. p. 1–6.

TORRALBA, A.; FERGUS, R.; FREEMAN, W. T. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 30, n. 11, p. 1958–1970, 2008.

WANG, D.; KHOSLA, A.; GARGEYA, R.; IRSHAD, H.; BECK, A. H. Deep learning for identifying metastatic breast cancer. *arXiv preprint arXiv:1606.05718*, 2016.

WEN, W.; WU, C.; WANG, Y.; CHEN, Y.; LI, H. Learning structured sparsity in deep neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2016. p. 2074–2082.

WERBOS, P. J. Applications of advances in nonlinear sensitivity analysis. In: *Proceedings of the 10th IFIP Conference*. [S.l.: s.n.], 1981. p. 762–770.

WIDROW, B.; HOFF, M. E. *Adaptive switching circuits*. [S.l.], 1960.

WU, B.; WAN, A.; YUE, X.; JIN, P.; ZHAO, S.; GOLMANT, N.; GHOLAMINEJAD, A.; GONZALEZ, J.; KEUTZER, K. Shift: A zero flop, zero parameter alternative to spatial convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2018. p. 9127–9135.

WU, J.; LENG, C.; WANG, Y.; HU, Q.; CHENG, J. Quantized convolutional neural networks for mobile devices. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2016. p. 4820–4828.

XIE, S.; GIRSHICK, R.; DOLLÁR, P.; TU, Z.; HE, K. Aggregated residual transformations for deep neural networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 1492–1500.

ZHANG, X.; ZHOU, X.; LIN, M.; SUN, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2018. p. 6848–6856.

ZHOU, A.; YAO, A.; GUO, Y.; XU, L.; CHEN, Y. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

ZHOU, S.; WU, Y.; NI, Z.; ZHOU, X.; WEN, H.; ZOU, Y. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.