Pós-Graduação em Ciência da Computação

CAMILA OLIVEIRA DE SOUZA

Usando Convolução Separável em Profundidade na Otimização da Arquitetura SqueezeNet

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

Recife
2020

CAMILA OLIVEIRA DE SOUZA

Usando Convolução Separável em Profundidade na Otimização da Arquitetura SqueezeNet

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.

Área de Concentração: Inteligência Computacional

Orientador: Cleber Zanchettin

Recife
2020

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

S729u Souza, Camila Oliveira de
Usando convolução separável em profundidade na otimização da arquitetura squeezenet / Camila Oliveira de Souza. – 2020.
74 f.: il., fig., tab.

Orientador: Cleber Zanchettin.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências.

1. Inteligência computacional. 2. Aprendizagem de máquina. I. Zanchettin, Cleber (orientador). II. Título.

006.31 CDD (23. ed.) UFPE - CCEN 2020 - 192

Camila Oliveira de Souza

“Usando Convolução Separável em Profundidade na Otimização da Arquitetura SqueezeNet”

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: 14/02/2020.

BANCA EXAMINADORA

_____

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

_____

Prof. Dr. Byron Leite Dantas Bezerra
Escola Politécnica de Pernambuco / UPE

_____

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

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela bençãos concedidas.

Aos meus pais, pelo suporte e pelo amor.

Aos meus amigos, pelo incentivo constante.

Ao meu orientador, Cleber Zanchettin, por estar sempre disponível e não me deixar desanimar.

A David Macêdo, por tirar minhas inúmeras dúvidas.

A todos que me apoiaram e auxiliaram de alguma forma durante essa jornada.

Muito obrigada.

RESUMO

Grandes avanços vêm sendo realizados em modelos baseados em Redes Neurais Convolucionais. Considerando problemas de processamento de imagens, esses modelos já possuem desempenho que em alguns casos superam o humano. Apesar dos excelentes resultados neste aspecto, modelos com taxas de acurácia extremamente altas são, em geral, muito grandes, chegando a possuir centenas de milhões de parâmetros, o que os torna inviáveis para diversas aplicações do mundo real, onde o poder computacional disponível normalmente é bastante limitado. Neste contexto, investigamos a possível redução do número de parâmetros da rede SqueezeNet, a qual se propõe a ser uma arquitetura com tamanho reduzido e boa taxa de acerto, a partir da substituição de suas convoluções tradicionais por Convoluções Separáveis em Profundidade (DSC), assim como o impacto desta substituição em outras métricas de análise do modelo. As métricas analisadas são a acurácia, o número de parâmetros, o tamanho de armazenamento e o tempo de inferência de um único exemplo de teste. A rede resultante, denominada SqueezeNet-DSC é então aplicada ao problema de classificação de imagens, e sua performance comparada com outras redes que são referência na área, como a MobileNet, a AlexNet e a VGG19. A SqueezeNet-DSC apresentou uma redução considerável no espaço de armazenamento, chegando a 37% do espaço de armazenamento da SqueezeNet original, com uma perda de acurácia de 1,07% na base CIFAR-10 e de 3,06% na base de dados CIFAR-100.

Palavras-chaves: Aprendizagem de Máquina. Redes Neurais Convolucionais. Convoluções Separáveis em Profundidade. SqueezeNet.

ABSTRACT

Great advances have been made in models based on Convolutional Neural Networks. Considering image processing problems, these models already outperform humans in some cases. Despite the excellent results in this area, models with extremely high accuracy rates are, in general, very large, reaching hundreds of millions of parameters, which makes them unfeasible for several real-world applications, where the computational power available is usually quite limited. In this context, we investigated the possible reduction in the number of parameters of the SqueezeNet network, which aims to be an architecture with reduced size and good accuracy, caused by the replacement of its traditional convolutions by Depthwise Separable Convolutions (DSC), as well as the impact of this replacement on other model analysis metrics. The metrics analyzed are the accuracy, the number of parameters, the storage size and the inference time of a single test example. The resulting network, called SqueezeNet-DSC, is then applied to the image classification problem, and its performance is compared to other networks that are a reference in the area, such as MobileNet, AlexNet and VGG19. SqueezeNet-DSC showed a considerable reduction in storage space, reaching 37% of the storage space of the original SqueezeNet, with a loss of accuracy of 1.07% on the CIFAR-10 base and 3.06% on CIFAR-100 database.

Keywords: Machine Learning. Convolutional Neural Networks. Depthwise Separable Convolutions. SqueezeNet.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

α	Alpha
λ	Lambda
$\in$	Pertence
θ	Teta
σ	Sigma
μ	Mu
ν	Nu

.

SUMÁRIO

1 INTRODUÇÃO

Algoritmos de Aprendizagem de Maquina (AM) estão inseridos no dia-a-dia de uma parcela significativa da população mundial. Sites de pesquisa, como o Google ou o Bing, funcionam tão bem porque utilizam algoritmos que aprendem a ranquear as páginas disponíveis na internet e utilizam isso nas buscas por termos de interesse do usuário. Também graças à utilização de algoritmos desse tipo, um servidor de e-mail pode poupar os seus clientes de visualizar dezenas de mensagens de spam todos os dias, o Facebook é capaz de reconhecer os seus usuários em fotos e a Netflix a recomendar filmes e séries baseados em interesses de seus assinantes.

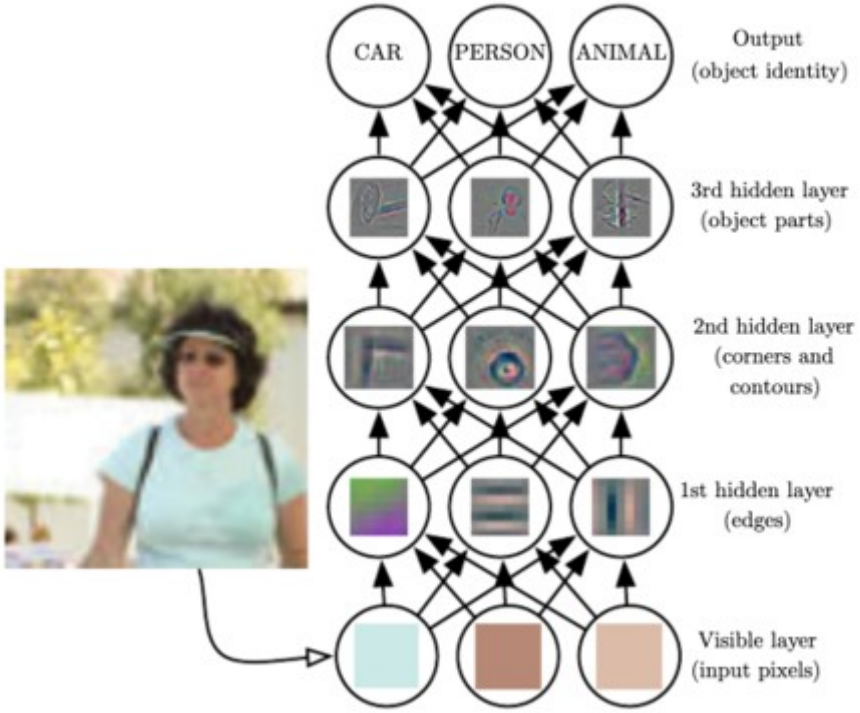
Um problema que surge na utilização desses algoritmos é que a sua performance depende fortemente da representação dos dados fornecidos a eles. Para entender esse problema vamos considerar o trabalho de Mor-Yosef et. al(MOR-YOSEF et al., 1990), que propõe um programa de computador para recomendar ou não um parto cesariana utilizando um algoritmo simples de AM chamado regressão logística. Para fazer essa recomendação, o médico fornece como entrada uma série de informações relevantes da paciente como, por exemplo, a presença ou ausência de cicatriz uterina. Cada uma dessas informações relevantes é denominada de característica (no inglês *feature*) e o trabalho do programa é entender a correlação dessas características com a sua recomendação final.

Muitos problemas do mundo real podem ser solucionados por AM se projetarmos o conjunto adequado de características. Porém, a escolha desse conjunto muitas vezes é tão complicada quanto o problema em si. Nesses casos, uma solução é usar AM também para essa seleção das informações de entrada. Essa abordagem é chamada de *representation learning* e frequentemente traz resultados mais interessantes do que aqueles atingidos com representações escolhidas manualmente pelo projetista do modelo (MIKOLOV et al., 2013) (HINTON; OSINDERO; TEH, 2006).

No entanto, a Aprendizagem de Máquina profunda, ou *Deep Learning* (DL), dá um passo adiante ao permitir que as características de um problema sejam expressas em termos de outras características ainda mais simples. A figura 1 apresenta como um sistema de DL pode representar objetos em uma imagem, combinando atributos simples até formar atributos mais complexos(GOODFELLOW; BENGIO; COURVILLE, 2016). Já a imagem 2 procura facilitar a compreensão das diferenças entre aprendizagem de máquina clássica, *representation learning* e *deep learning*.

As redes neurais são algoritmos extremamente populares no contexto de aprendizagem de máquina. Utilizando redes neurais com diversas camadas de neurônios, a aprendizagem de máquina profunda vem sendo aplicada em diversas áreas com resultados impressionantes. Sistemas que auxiliam na predição de terremotos(ADELI; PANAKKAT, 2009), na detecção de tumores (DAHAB et al., 2012) e na tradução de textos (SWE; TIN, 2005) são

Figura 1 – Ilustração de como a aprendizagem profunda trata um conjunto de pixels de entrada e constrói seus conceitos



Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

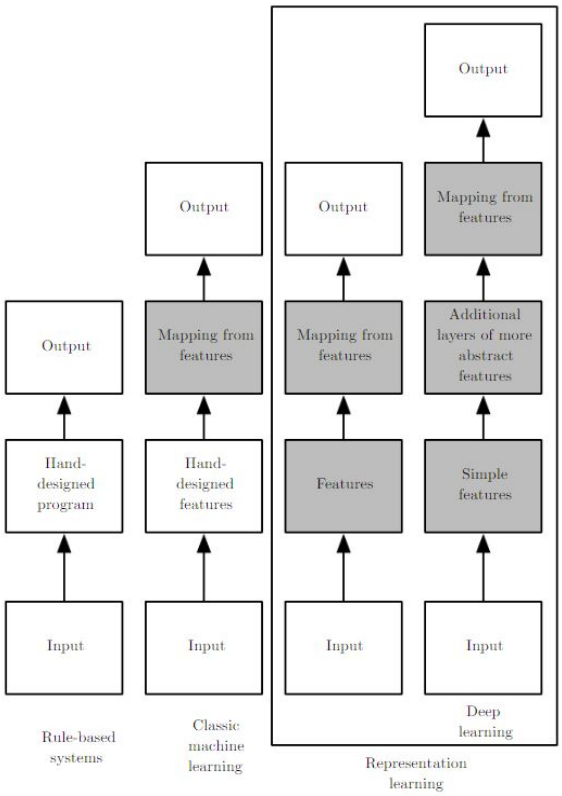
apenas alguns exemplos dessas aplicações.

A área de Visão Computacional, que visa extrair e processar informações a partir de imagens ou vídeos, foi uma das maiores beneficiadas com a evolução das técnicas de DL. As redes neurais convolucionais (CNNs) (LECUN et al., 1989), por exemplo, estão entre os algoritmos de aprendizagem profunda mais eficientes na solução de diversos problemas como classificação de imagens, detecção de objetos ou reconstrução de imagens (XIE et al., 2019) e (LIU et al., 2019).

CNNs são redes neurais profundas que possuem uma arquitetura especial, adaptada para lidar com imagens. A nomenclatura indica que essas redes utilizam a operação matemática da convolução no lugar da multiplicação de matrizes tradicional. De forma simplificada, a camada de convolução gera o que chamamos de um mapa de ativação, a partir da aplicação de um filtro sobre a sua entrada, que pode ser uma imagem ou um outro mapa de ativação de uma camada de convolução anterior. Ao longo dos anos, diversas formas de implementar a convolução foram propostas, com o objetivo de tornar essa operação mais eficiente. Entre elas, podemos citar a convolução separável em profundidade(CHOLLET, 2017), a convolução de grupo (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) e a refatorização diagonal(QIN et al., 2018).

Quando aplicadas para classificação de imagens, problema que será abordado nessa

Figura 2 – Diferenças entre os tipos de aprendizagem de máquina



Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

dissertação, as CNNs possuem taxas de acerto que em muito superam aquela dos seres humanos. Enquanto o erro top-5 para classificação de imagens de um ser humano na base de dados Imagenet é de 5.1% (DENG et al., 2009), a rede EfficientNet (TAN; LE, 2019), treinada com o método *Noisy Student*, relatou um erro top-5 de 1.8% em 2019, em uma competição conhecida como ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Enquanto o erro top-1 é uma métrica que indica a porcentagem de vezes em que a classe sugerida pelo classificador não era, de fato, a classe correta, o erro top-5 analisa as 5 classes consideradas pelo classificador como aquelas de maior probabilidade, indicando a porcentagem de vezes em que a classe correta não estava nesse grupo.

O ImageNet é um dataset utilizado como referência na literatura para comparação dos resultados dos novos modelos de redes neurais convolucionais apresentados. Esta base de dados conta com mais de um milhão de imagens de treinamento, distribuídas entre 1000 classes e foi utilizada entre 2010 e 2017 na *ImageNet Large Scale Visual Recognition Challenge*, ILSVRC, competição anual que costumava avaliar algoritmos de detecção de objetos e classificação de imagens em larga escala. Outras bases de dados como o CIFAR-10 e o CIFAR-100 (KRIZHEVSKY; NAIR; HINTON, 2009) também são utilizados como referências na literatura e serão apresentados posteriormente durante este trabalho.

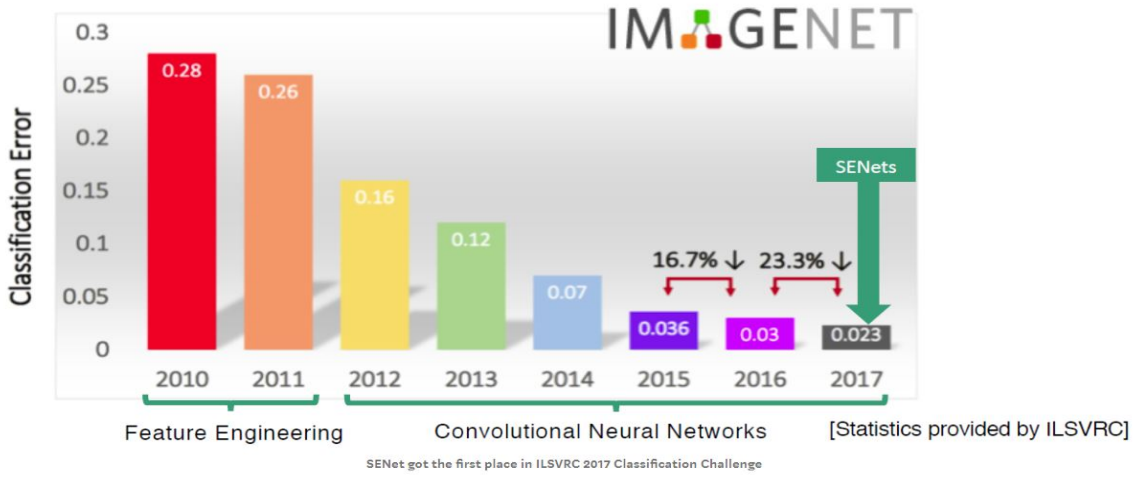
Em 2012, pela primeira vez, uma CNN foi utilizada no ILSVRC, ganhando a compe-

tição com um erro top-5 de 15.4%, em relação à 26.2% do segundo colocado. A AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) possuía 60 milhões de parâmetros e consistia de 5 camadas de convolução e 3 camadas completamente conectadas. Algumas das contribuições relatadas pelos autores da AlexNet foram a utilização de uma implementação da camada de convolução eficiente para GPU e de um método de regularização denominado "*dropout*".

O resultado surpreendente deste modelo marcou o início da utilização massiva das CNNs em tarefas de visão computacional. Os dados do ILSVRC demonstram o crescente desempenho das redes convolucionais na tarefa de classificação de imagens ao longo dos últimos anos. A imagem 3 mostra a taxa de erro top-5 da equipe vencedora desde 2010 até o final da competição em 2017. Esses resultados são fruto da intensa pesquisa científica que vem sendo realizada na área de redes neurais convolucionais, grande parte dela objetivando aumentar a acurácia das redes em bases de dados de visão computacional.

Para atingir tal objetivo, a tendência é utilizar redes cada vez mais profundas e complexas, o que resulta em modelos com um número exorbitante de parâmetros. A EfficientNet, citada anteriormente e que até o dia da publicação deste trabalho detinha a liderança no ImageNet, conta com 480 milhões de parâmetros. O atual segundo lugar pertence à FixResNeXt-101 (TOUVRON et al., 2019), com um erro top-5 de 2% e um total de 829 milhões de parâmetros.

Figura 3 – Taxa de erro Top-5 das equipes vencedoras do ILSVRC desde 2010



Fonte: Retirado da internet [1]

Vemos que apesar da intensa pesquisa realizada na área estar trazendo avanços excelentes em relação às taxas de acurácia das CNNs, esses avanços não são necessariamente traduzidos em impactos positivos em outras métricas das redes, como por exemplo, o tamanho de armazenamento e o tempo de inferência. Para muitas aplicações no mundo real,

[1] Disponível em: https://towardsdatascience.com/review-senet-squeeze-and-excitation-network-winner-of-ilsvrc-2017-image-classification-a887b98b2883. Acesso em 4 fev. 2020

no entanto, o poder computacional disponível é bastante restrito e redes com uma grande quantidade de parâmetros possuem aplicabilidade limitada. Por esse motivo, pesquisas que focam na otimização da arquitetura das redes, visando reduzir métricas como número de parâmetros, espaço de armazenamento e tempo de inferência, vêm ganhando cada vez mais relevância.

Entre as estratégias utilizadas na literatura para a otimização de arquiteturas das CNNs, podemos citar aquelas que tentam comprimir redes pré-treinadas, como pruning (HINTON; VINYALS; DEAN, 2015), quantization (CHOUKROUN; KRAVCHIK; KISILEV, 2019) e distillation(HEGDE et al., 2019); e aquelas que tentam treinar redes já inicialmente pequenas.

Esse último é o caso da SqueezeNet(IANDOLA et al., 2016), cujo objetivo é a redução do número de parâmetros através da introdução de *Fire Modules*, novos blocos que podem ser utilizados na construção de arquiteturas de CNNs e utilizam uma estratégia de compressão e expansão dos mapas de ativação em camadas de convolução.

Outros exemplos do segundo tipo de estratégia são os trabalhos que buscam conseguir redes pequenas mexendo na implementação da própria camada de convolução. Entre eles podemos citar, por exemplo, a MobileNet(HOWARD et al., 2017), que utiliza convoluções separáveis em profundidade (DSC), a ShuffleNet(ZHANG et al., 2018), que utiliza convoluções de grupo seguidas de operações de *channel shuffle*.

1.1 OBJETIVOS

O problema investigado nesse trabalho é a possível redução do número de parâmetros de uma rede neural convolucional a partir da substituição de suas convoluções por Convoluções Separáveis em Profundidade (DSC) e o impacto desta substituição em outros parâmetros importantes do modelo. A rede utilizada como base para a nossa pesquisa foi a SqueezeNet(IANDOLA et al., 2016), que procura reduzir o seu tamanho pela introdução dos denominados *Fire Modules*, um novo componente básico que pode ser utilizado para a construção de arquiteturas de redes convolucionais. Essa rede foi escolhida por já ser uma rede pequena, projetada para ambientes com restrições de poder computacional, e com excelente relação acurácia x número de parâmetros. Além disso, ela utiliza em seus *Fire Modules* filtros de convolução tradicionais, sem nenhuma alteração.

A nossa proposta é uma nova implementação da SqueezeNet que modifica os Fire Modules de forma a substituir os seus filtros de convolução 3x3 por filtros DSC. A esta nova implementação da SqueezeNet foi dado o nome de SqueezeNet-DSC.

Para melhor compreender os efeitos da DSC na rede SqueezeNet, nós consideramos quatro aspectos da sua aplicação.

- Acurácia
- Número de parâmetros

- Tamanho de armazenamento
- Tempo de inferência de um único exemplo de teste

A rede proposta é então aplicada ao problema de classificação de imagens, mais especificamente nas bases de dados CIFAR-10 e CIFAR-100, onde a sua performance é comparada a outras redes que são referência na área, como a própria SqueezeNet, a MobileNet, a AlexNet e o modelo da *Visual Geometry Group* com 19 camadas (VGG19).

1.2 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado da seguinte maneira:

- **Capítulo 2 - Fundamentação Teórica e Revisão da Literatura**: Conceitos relacionados ao problema de classificação de imagens, às redes neurais convolucionais e à convolução separável em profundidade são apresentados neste capítulo. O capítulo apresenta ainda uma revisão da literatura, focando em trabalhos que investigam CNNs projetadas para ambientes com limitações de memória e poder computacional (ambiente mobile).
- **Capítulo 3 - Sistema Proposto**: A SqueezeNet original é apresentada com maiores detalhes no capítulo 3, bem como a nossa proposta, a SqueezeNet-DSC.
- **Capítulo 4 - Experimentos**: Aqui são apresentadas as bases de dados CIFAR-10 e CIFAR-100, além dos experimentos realizados nelas a fim de avaliar a rede proposta.
- **Capítulo 5 - Resultados e Discussão**: Aqui são apresentados os resultados obtidos durante o desenvolvimento deste trabalho.
- **Capítulo 6 - Conclusão**: Dificuldades, conclusões gerais, contribuições e possíveis trabalhos futuros são discutidos nesta seção.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Este capítulo apresenta a base teórica dos temas abordados ao longo desta dissertação. A Seção 2.1. descreve o problema de classificação de imagens e a sua relevância na comunidade científica. Na Seção 2.2 são apresentados conceitos básicos sobre redes neurais, como o seu processo de treinamento, suas principais características, e os algoritmos e técnicas utilizados para obtenção de bons resultados. A Seção 2.3 introduz as Redes Neurais Convolucionais, mostrando detalhes de suas diferentes camadas e os benefícios de sua utilização na tarefa de classificação de imagens. A seção 2.4 traz o conceito de otimização de arquiteturas CNN, bem como algumas das principais técnicas apresentadas na literatura.

2.1 CLASSIFICAÇÃO DE IMAGENS

Após o surgimento da inteligência artificial, não demorou muito para que a nova ciência obtivesse resultados impressionantes atacando problemas considerados intelectualmente desafiadores para os seres humanos. Um bom exemplo é o sistema Deep Blue da IBM que conseguiu, em 1997, derrotar o campeão mundial de xadrez Garry Kasparov (CAMPBELL; JR; HSU, 2002). Apesar de ser um trabalho difícil para um ser humano, o jogo de xadrez é uma tarefa simples para um computador, por ser completamente descrita por um conjunto de regras formais que podem ser previamente fornecidas pelo programador.

Atualmente, um dos desafios da IA reside em tarefas que são fáceis de serem resolvidas pelos seres humanos, porém difíceis de serem descritas formalmente. Tarefas que nós resolvemos de forma automática, como reconhecer palavras faladas ou rostos em imagens. Esses problemas demandam uma grande quantidade de conhecimento sobre o mundo, que precisa ser absorvida pelos algoritmos para que estes se comportem de forma eficiente. Esse conhecimento, porém, é, em sua maior parte, subjetivo e intuitivo e, portanto, difícil de ser traduzido em regras formais. A tarefa de classificação de imagens é um bom exemplo dos problemas a que nos referimos.

Segundo (SOLOMON; BRECKON, 2000), a imagem pode ser definida como uma informação visual bidimensional que pode ser armazenada e exibida. Contudo, em um contexto mais amplo, imagem pode ser compreendida como qualquer coisa que pode ser vista e interpretada a fim de obter alguma informação.

Para os seres humanos, o processo de compreensão de imagens começa antes até do próprio parto, onde só se consegue diferenciar luz do escuro, e continua até por volta dos 2 anos, onde objetos complexos já podem ser discernidos. Isso ocorre porque o processo de compreensão do que é visto não consegue absorver todo o tipo de informação de uma vez. Primeiramente, imagens em preto e branco se tornam imagens coloridas, em seguida,

traços borrados começam a ficar mais definidos, começa-se a discernir rostos e a noção de profundidade é desenvolvida. Até então a imagem continua sem significado semântico, na qual nenhuma informação no sentido computacional pode ser extraída. A imagem é só uma imagem. É só por volta dos 8 a 10 meses que ganhamos a capacidade de explorar o mundo ao nosso redor e atribuímos sentido ao que vemos.

Diferentemente de nós, computadores só enxergam uma matriz bidimensional de pixels, na qual cada pixel guarda uma informação sobre características como cor, luminosidade e saturação, dependendo do tipo de representação utilizado. Consequentemente, a imagem não traz consigo nenhuma informação que esteja intrinsecamente ligada aos objetos que a compõem, tornando a tarefa de diferenciar e classificar objetos um trabalho difícil para um computador. A figura 4 amplia uma pequena região em uma imagem, mostrando a diferença entre o que é visto por um ser humano e o que é visto pelo computador.

Figura 4 – Representação de uma imagem RGB

Fonte: (SOLOMON; BRECKON, 2000)

No modelo computacional comum, o computador simplesmente executa uma determinada tarefa que lhe é dada seguindo uma série bem definida de instruções, ou seja, é necessário uma definição formal do problema. Consequentemente, é necessário definir quais seriam os passos na identificação de um objeto específico; agora, imagine que tenha que se definir regras para identificação de diversos objetos, parece bem difícil, não? Como descrever uma bola de baseball ou um óculos escuros ou um palito de fósforos; definir regras para identificação de objetos explicitamente por um programador pode se tornar uma tarefa extremamente dispendiosa. Por isso, torna-se necessário que os sistemas de IA tenham a habilidade de adquirir o próprio conhecimento, extraindo padrões de dados brutos. Este recurso é conhecido como Aprendizagem de Máquina (AM)(GOODFELLOW; BENGIO; COURVILLE, 2016).

A aprendizagem de máquina oferece ferramentas bastante úteis para a tarefa de classificação de imagens, como as Redes Neurais Convolucionais (CNN) e o seu treinamento a partir da aprendizagem supervisionada. No contexto da aprendizagem supervisionada,

diversas imagens de treinamento são fornecidas como entradas a uma CNN, juntamente com a informação sobre suas respectivas saídas ou classes. A CNN, que nada mais é do que um sistema de IA, tenta então extrair características importantes das classes a partir das imagens fornecidas como entrada e então generalizar esse conhecimento para classificar corretamente novas imagens nunca antes vistas. Os conceitos de Redes Neurais e Redes Neurais Convolucionais, além dos algoritmos envolvidos no seu treinamento serão apresentados nas seções seguintes.

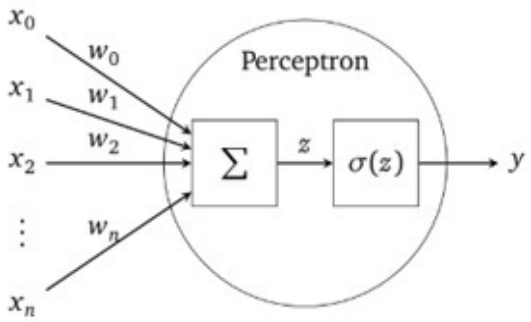
2.2 REDES NEURAIS ARTIFICIAIS

2.2.1 Conceito

As Redes Neurais Artificiais são alguns dos algoritmos mais populares dentro da área de Aprendizagem de Máquina. A sua inspiração vem da neurociência, em uma tentativa de imitar o neurônio biológico. O primeiro neurônio artificial, conhecido como neurônio MP, foi sugerido por McCulloch e Pitts em 1943(MACCULLOCH; PITTS, 1943), e representa um elemento com múltiplas entradas e apenas uma saída. Cada entrada pode ter o valor de 0 ou de 1 e ser excitatória ou inibitória. Entradas excitatórias são multiplicadas por 1, enquanto entradas inibitórias são multiplicadas por -1. Os valores dessas multiplicações são então somados e o resultado é comparado a um limiar. Caso essa soma ultrapasse o limiar pré-estabelecido, a saída do neurônio assume o valor 1. Caso contrário, o valor assumido é 0.

Inspirado pelo trabalho de McCulloch e Pitts, Rosenblatt propõe, em 1958, o modelo Perceptron (ROSENBLATT, 1958). Rosenblatt multiplica as entradas do Perceptron por pesos, que podem assumir valor diferentes de apenas -1 e 1, como era o caso do neurônio MP. A figura 5 mostra um modelo clássico do Perceptron, onde z recebe o valor da combinação linear das entradas com os seus respectivos pesos.

Figura 5 – Modelo do Perceptron



Fonte: (MÜNKER, 2016)

$$z = \sum_{i=0}^{N} x_i \cdot w_i \tag{2.1}$$

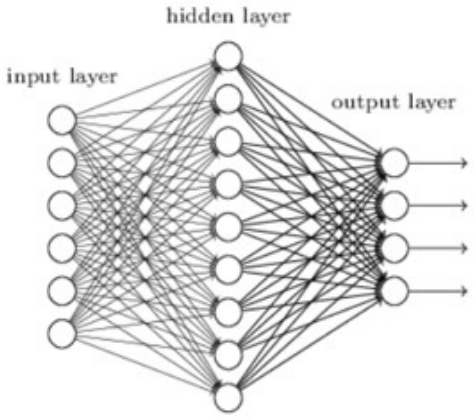
Este valor é posteriormente inserido em uma função de ativação $\sigma(z)$ que atribui 1 à saída do neurônio, caso o valor de z seja maior do que um limiar, ou *threshold*.

$$\sigma(z) = \begin{cases} 0, & \text{se } z < threshold \\ 1, & \text{se } z \geq threshold \end{cases} \tag{2.2}$$

Na equação 2.2, se passarmos o *threshold* para o lado oposto da inequação, temos o que chamamos de viés, ou *bias*.

À estrutura formada por vários neurônios interconectados e organizados em camadas dá-se o nome de Redes Neurais Artificiais (ANN). Caso os neurônios utilizados forem do tipo perceptron, é comum utilizar-se a nomenclatura *Multi-Layer Perceptrons*(MLP). Outros modelos de neurônio foram desenvolvidos posteriormente e podem ser utilizados nas redes neurais. Um exemplo é o neurônio sigmóide, que utiliza a função sigmóide como função de ativação, fazendo com que a sua saída possa receber não apenas 0 e 1, como também números reais nesse intervalo. Mais detalhes sobre diferentes funções de ativação podem vistos na seção 2.2.3. A imagem 6 oferece uma representação clássica de uma ANN.

Figura 6 – Camadas de uma rede neural



Fonte: Retirado da internet[1]

As camadas entre a camada de entrada e a camada de saída geralmente recebem o nome de camadas escondidas, ou *hidden layers*. Nesse exemplo, vemos que os neurônios das camadas estão completamente conectados. Esse tipo de camada é chamado de camada completamente conectada, ou *FC layer (Fully Connected layer)*. Esses não são os únicos tipos de camadas existentes e durante este capítulo comentaremos sobre outras camadas como as de *Softmax*, de convolução e de *pooling*. Esses diferentes modelos de camadas podem ser combinados para formar diferentes arquiteturas de redes neurais.

[1] Disponível em: https://www.researchgate.net/figure/Model-of-convolutional-neural-network-based-classification-Artificial-neural-network$_{f}ig3_{3}$22339077.$Acessoem$: $4fev$.2020

2.2.2 O treinamento do modelo

Nesta seção, abordaremos o treinamento das redes neurais no contexto da aprendizagem de máquina supervisionada, formato em que as redes tentam aprender uma função que mapeia uma entrada para uma saída baseado em pares de exemplo entrada-saída (RUSSELL; NORVIG, 2016).

De forma geral, uma rede neural é um modelo que tenta aproximar uma função $y = f(x)$, através de uma outra função $\hat{y} = f(x, \theta)$, que depende não apenas da sua entrada, mas do valor dos seus parâmetros Θ. Através do que chamamos de algoritmos de aprendizagem, as redes neurais podem realizar ajustes automáticos aos seus parâmetros, ou seja, seus pesos e vieses, de forma que $\hat{y} \tilde{=} y$. A esse processo, é dado o nome de treinamento. O que queremos então é diminuir a diferença entre as saídas do modelo e as saídas desejadas, que pode ser calculada por uma função de custo, também chamada de função de perda. Para diferentes problemas, diferentes funções de custo podem ser utilizadas. Um exemplo simples é o erro quadrático médio (MSE), que retorna a média da diferença entre y e ŷ, elevada ao quadrado. Na equação 2.3, N simboliza o número de exemplos do conjunto de treinamento.

$$J(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \tag{2.3}$$

Para encontrar os valores dos parâmetros que minimizam a função de custo, um dos métodos mais utilizados é o gradiente descendente e suas variações. A fim de realizar ajustes nos valores dos parâmetros, o gradiente descendente precisa calcular o gradiente da função de custo em relação a Θ e, para tal tarefa, o algoritmo de *backpropagation* é utilizado.

Mais detalhes sobre funções de custo, o método do gradiente descendente e o algoritmo de *backpropagation* serão apresentados a seguir.

2.2.2.1 Função de Perda

Dado que o objetivo de um algoritmo de aprendizagem é definir um conjunto de parâmetros tal que a saída da rede neural em treinamento aproxime a função $y = f(x)$, a perda nos traz uma referência do quão perto estamos desse propósito.

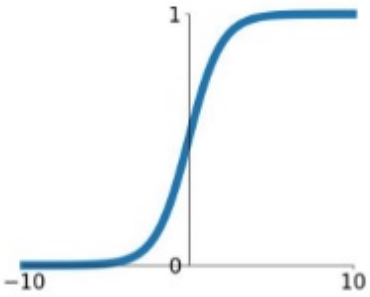
A escolha da melhor função de perda depende do problema atacado. O erro MSE, apresentado anteriormente, é uma função muito utilizada em problemas de regressão. Porém, quando utilizada em conjunto com a função de ativação sigmóide, apresenta o problema de *learning slowdown*. Esse problema surge porque o algoritmo de *backpropagation* utiliza o gradiente da função de perda para ajustar os parâmetros da rede. O gradiente da MSE,

por sua vez, depende da derivada da função sigmóide, $\sigma'(z)$.

$$\begin{aligned} J &= \frac{(y-\hat{y})^2}{2}; \hat{y} = \theta(z); z = wx + b \\ \frac{\partial J}{\partial w} &= \frac{\partial J}{\partial z}\frac{(y-\hat{y})^2}{2}\frac{\partial z}{\partial w} \\ \frac{\partial J}{\partial w} &= (\hat{y} - y)\sigma'(z)x \end{aligned} \tag{2.4}$$

Se lembrarmos do formato da função sigmóide, vemos que quando a saída do neurônio está próxima de zero ou de 1, a derivada se torna muito pequena, o que causa uma diminuição do ritmo do aprendizado.

Figura 7 – Função Sigmóide



Fonte: Retirado da internet[2]

Em problemas de classificação, a função de perda de entropia cruzada (NIELSEN, 2015) é uma das mais utilizadas. Ela resolve o problema de *learning slowdown*, além de ser uma escolha natural quando utilizamos uma camada *softmax* (GOODMAN, 2001) no final da rede neural. Isso porque a saída de uma camada *softmax* é um vetor com o tamanho equivalente ao número de classes do problema i, no qual cada posição representa a probabilidade de uma classe específica. O conceito de *softmax* será explicado em maiores detalhes na seção 2.2.3.3. A entropia cruzada é dada por:

$$H(y, \hat{y}) = -\sum_{i=1}^{C} y_i \cdot \ln \hat{y}_i \tag{2.5}$$

Na equação 2.5, tanto y quanto $\hat{y}$ são distribuições de probabilidade, com y_i sendo o valor desejado para a classe i e $\hat{y}_i$ a probabilidade computada pela rede neural para a mesma classe. O vetor y possui apenas uma posição com valor 1, a classe real, enquanto

[2] Disponivel em: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044. Acesso em 4 fev. 2020

todas as outras posições são 0. A função de perda, $J(\theta)$ é então computada a partir da média de todas as entropias dos exemplos de treinamento:

$$J(\theta) = \frac{1}{N} \sum_{n=1}^{N} H(y_n, \hat{y_n}) \tag{2.6}$$

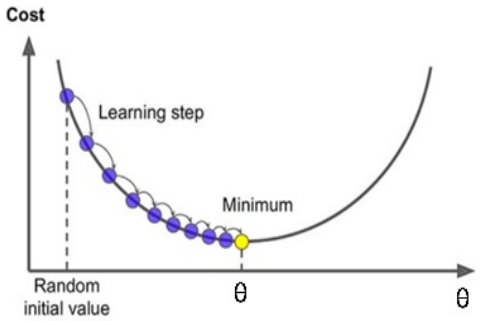
na qual N é o número de exemplos de treinamento.

Não é intuitivo perceber porque a entropia cruzada pode ser utilizada como função de perda. Duas características simples, porém, podem nos auxiliar no entendimento. A primeira é que $H(y, \hat{y})$ é sempre positiva. Isso acontece porque $\hat{y}_i$ é sempre um número entre 0 e 1, fazendo com que o seu logaritmo seja sempre negativo. A segunda é que a entropia cruzada tem valor próximo de 0 quando a saída da rede tem valor próximo ao desejado. Essas são duas características desejadas em uma função de perda. Além disso, se calcularmos o gradiente da entropia cruzada, vemos que não temos mais o termo proporcional à derivada da função sigmóide. O que resolve o problema do *learning slowdown*(NIELSEN, 2015).

2.2.2.2 Gradiente Descendente

O gradiente descendente(CAUCHY, 1847) (GD), é um algoritmo de otimização que tem como objetivo encontrar o valor mínimo de uma função de uma forma iterativa. A cada iteração, dá-se um passo na direção do negativo do gradiente. No contexto do treinamento das redes neurais, utiliza-se o gradiente descendente para atualizar os parâmetros do modelo, de forma a minimizar a função de perda.

Figura 8 – Gradiente Descendente



Fonte: Adaptado da internet[3]

O tamanho do passo que vai ser dado na direção negativa do gradiente é controlado por um parâmetro chamado de taxa de aprendizagem, ou *learning rate* (lr). A taxa de aprendizagem é um hiperparâmetro do modelo e deve ser escolhido com cautela. Uma taxa

[3] Disponível em: https://towardsdatascience.com/an-introduction-to-logistic-regression-8136ad65da2e. Acesso em: 4 fev. 2020

grande demais acelera o aprendizado, mas traz o risco de ultrapassar o mínimo global. Uma taxa muito pequena diminui esse risco, porém torna o aprendizado muito lento.

A equação 2.7 resume o algoritmo do gradiente descendente:

$$\theta = \theta - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta} \tag{2.7}$$

onde θ são os parâmetros do modelo, J é a função de custo escolhida e α é a taxa de aprendizagem.

Uma variação do GD é o Gradiente Descendente Estocástico (SGD), com *mini-batches*. Para cada passo, ou iteração, essa variação utiliza um subconjunto do conjunto de treinamento, cujo tamanho também é um hiperparâmetro a ser escolhido. Durante o restante do trabalho, nos referimos ao tamanho das *mini-batches* simplesmente como tamanho do *batch*, ou lote. O GD tradicional, também referido na literatura como *Vanilla Gradient Descent*, pode ser muito custoso computacionalmente, além de ter um longo tempo de atualização, já que, para cada iteração, utiliza todo o conjunto de treinamento, que pode ter milhares de exemplos. Aqui, é interessante definirmos dois termos utilizados no treinamento de redes neurais. Uma **iteração** representa uma atualização do gradiente descendente, ou *learning step*. Já uma **época** é uma passagem ao longo de todo o conjunto de treinamento. Normalmente, uma época consiste de várias iterações.

Uma outra alternativa bastante utilizada é o SGD com momento. Inicialmente proposto por Polyak em 1964 (POLYAK, 1964), essa variação do GD aplica uma média móvel exponencial no gradiente, fazendo com que ele passe a levar em consideração os seus valores passados. Essa técnica costuma acelerar o treinamento em casos onde os gradientes são pequenos e ruidosos. A atualização dos parâmetros no SGD com momento é realizada da seguinte forma:

$$\begin{aligned} v &= \mu\nu - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta}, \\ \theta &= \theta + \nu. \end{aligned} \tag{2.8}$$

no qual $\mu \in [0; 1]$ é um hiperparâmetro que descreve a taxa de decaimento do momento e v é a velocidade.

Para explicar esse algoritmo, a maior parte dos livros recorre à analogia de um objeto rolando montanha abaixo. A função de perda pode ser vista como a altura do objeto. Inicializar o objeto randomicamente significa colocá-lo em algum lugar da montanha, com velocidade zero. A força sentida pelo objeto é exatamente o negativo do gradiente da função de perda. Nessa visão do problema, o gradiente tem impacto direto apenas na velocidade, que por sua vez impacta na posição do objeto. O momento, apesar da nomenclatura, é mais parecido com um coeficiente de fricção, possibilitando a parada do objeto ao atingir o final da montanha.

A escolha da taxa de aprendizagem é uma parte crítica do treinamento de uma rede neural, já que pode influenciar profundamente o resultado do treinamento e é, muitas vezes, feita de forma empírica. Os algoritmos apresentados até então usam uma taxa de aprendizagem global e fixa para todos os parâmetros da rede. Os algoritmos de gradiente descendente adaptativos, porém, trazem a ideia de uma taxa de aprendizagem para cada parâmetro, de acordo com as suas necessidades individuais. A taxa global ainda existe, mas ela é adaptada para cada parâmetro, de acordo com os valores passados dos seus gradientes. A cada iteração do gradiente, as *learning rates* também são alteradas. Os algoritmos adaptativos mais famosos são o Adagrad, o RMSProp e o Adam.

O Adagrad (DUCHI; HAZAN; SINGER, 2011) guarda, para cada parâmetro, a soma do quadrado dos gradientes e depois usa esse valor para normalizar a taxa de aprendizagem individual. Efetivamente, isso faz com que os parâmetros com valores altos do gradiente tenham a sua taxa de aprendizagem reduzida, enquanto parâmetros com atualizações menores, ou menos frequentes, tenham taxas mais altas. Segundo Duchi (DUCHI; HAZAN; SINGER, 2011), essa adaptação permite encontrar "agulhas nos palheiros", ou seja, características muito previsíveis, mas que são raramente vistas.

Por utilizar a soma do quadrado dos gradientes, um termo que é sempre crescente, para controlar as atualizações dos parâmetros, o Adagrad pode levar a reduções excessivas nas taxas de aprendizagem individuais dos parâmetros, prejudicando ou até mesmo interrompendo o aprendizado. O RMSProp (TIELEMAN; HINTON, 2012) resolve esse problema ao trocar a soma dos gradientes por uma média móvel ponderada. Com isso, gradientes antigos têm menor influência no termo acumulativo. Temos, portanto, a introdução de um novo hiperparâmetro que controla a taxa com a qual os gradientes passados deixam de exercer influência na média móvel.

O Adam(KINGMA; BA, 2014), por sua vez, introduz uma melhoria no RSMProp, ao utilizar o momento do SGD na sua regra de atualização da taxa de aprendizagem. Ele também conta com um termo de correção para balancear o viés introduzido pelos termos do SGD, que são inicializados em zero.

2.2.2.3 Backpropagation

Pouco após o surgimento dos primeiros neurônios artificiais, Minsky e Papert(MINSKY; PAPERT, 1969) provaram que os neurônios utilizados na época, que eram modelos lineares e, portanto, bastante limitados, não conseguiam aprender a função XOR, onde $f([0,1],w) = 1$ e $f([1,0],w) = 1$ mas $f([1,1],w) = 0$ e $f([0,0],w) = 0$. Essa constatação levou a uma queda na popularidade das redes neurais, que só voltou a crescer após o advento do algoritmo de *backpropagation*(LINNAINMAA, 1970) e de seu reconhecimento como uma ferramenta para o treino de redes neurais(RUMELHART; HINTON; WILLIAMS, 1986; CUN; FOGELMAN-SOULIÉ, 1987).

Como visto anteriormente, o gradiente descendente utiliza o gradiente da função de

perda para atualizar os parâmetros da rede. O algoritmo de *backpropagation* traz um método eficiente para calcular esse gradiente, possibilitando o treinamento de redes neurais com várias camadas. De acordo com Cybenko, uma rede neural com pelo menos uma camada escondida pode não só aprender a função XOR, como, em teoria, aproximar qualquer função contínua(CYBENKO, 1989).

Este algoritmo possui duas fases bem definidas: a propagação e a retropropagação. Na primeira delas, o *forward pass*, as entradas são passadas através da rede e o erro entre a saída real e a saída esperada é calculado. Durante a segunda fase, ou *backward pass*, esse erro é retropropagado através da rede, para que os parâmetros possam ser ajustados. Cada camada em uma rede neural pode ser vista como como uma função $f^{(\mathrm{i})}(x, \Theta^{(\mathrm{i})}) = y$. Podemos enxergar então uma rede neural como uma composição de funções $f(x, \Theta) = f^{(\mathrm{n})}(...f^{(2)}(f^{(1)}(x, \Theta^{(1)}), \Theta^{(2)}), \Theta^{(\mathrm{n})})$. De forma simplificada, o backpropagation utiliza esse conceito e calcula o gradiente relativo aos parâmetros de cada camada através da regra da cadeia. Para mais detalhes da matemática por trás do backpropagation, recomendamos o segundo capítulo do livro *"Neural Networks and Deep Learning"*(NIELSEN, 2015).

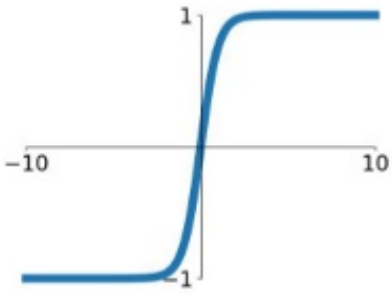
2.2.3 Funções de Ativação

A parte de um neurônio que dá a ele o poder de resolver problemas não-lineares é a sua função de ativação. Uma das características necessárias para uma função de ativação é que ela deve ser continuamente diferenciável, para que o algoritmo de *backpropagation* possa ser utilizado com sucesso. Na seção 2.2.2.1, mencionamos brevemente a função de ativação sigmóide e o seu problema de *learning slowdown*, causado pelo fato de sua derivada tender a zero nas proximidades dos valores de saturação. Nesta seção, apresentaremos outras funções de ativação bastante utilizadas nas redes neurais.

2.2.3.1 Tangente Hiperbólica

Uma variação simples da função sigmóide é a função tangente hiperbólica. As duas possuem a mesma forma, porém, enquanto a saída da sigmóide varia entre 0 e 1, a saída da tangente hiperbólica varia entre -1 e 1. O formato da função tangente hiperbólica pode ser visto na figura 9. Por ter o mesmo formato da sigmóide, ela apresenta o mesmo problema de *learning slowdown*.

Figura 9 – Função tangente hiperbólica



Fonte: Retirado da internet[4]

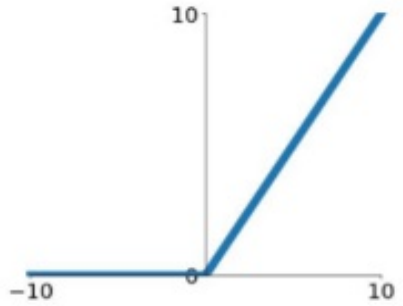
2.2.3.2 ReLU

Muitos trabalhos na área de reconhecimento de imagens têm mostrado benefícios em utilizar a função de ativação ReLU na maioria das camadas de uma rede neural (JARRETT et al., 2009). A sigla ReLU significa *Rectified Linear Unit* (NAIR; HINTON, 2010) e sua saída para uma entrada x, vetor de pesos w, e viés b é dada por

$$y = max(0, w \cdot x + b) \tag{2.9}$$

e sua representação gráfica pode ser vista na figura 10. Uma propriedade que torna a ReLU popular é a rapidez do cálculo da sua derivada, que é simplesmente 0 para entradas menores do que 0, e 1 para entradas maiores do que 1. Uma outra vantagem da utilização da ReLU é que, para entradas positivas, ela não apresenta o problema de *learning slowdown* apresentado pela sigmóide e pela tangente hiperbólica.

Figura 10 – Função gráfica que representa o comportamento do ReLU



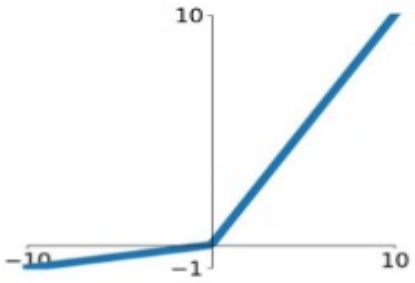
Fonte: Retirado da internet[5]

[4] Disponivel em: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044. Acesso em 4 fev. 2020

[5] Disponivel em: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044. Acesso em 4 fev. 2020

Porém, para o caso de entradas negativas, o aprendizado é completamente interrompido. Isso pode levar a um problema de *dying units*, ou seja, algumas unidades que não conseguem ser ativadas por nenhuma entrada do conjunto de treinamento. A variação *Leaky ReLU* (MAAS; HANNUN; NG, 2013), introduzida inicialmente em modelos acústicos, resolve esses problemas modificando o comportamento da unidade para entradas negativas. Essa modificação é feita inserindo um componente linear com inclinação entre 0 e 1, de forma que o gradiente da função de ativação nunca é zero.

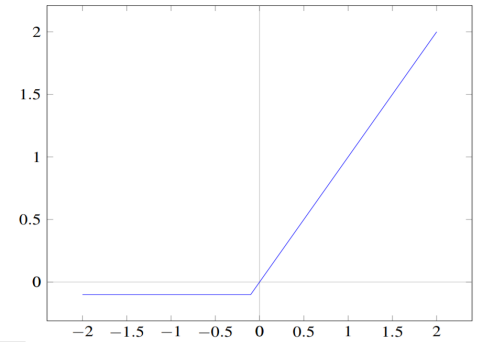
Figura 11 – Função gráfica que representa o comportamento do Leaky ReLU



Fonte: Retirado da internet[6]

A variação *Displaced ReLU*, proposta por (MACEDO, 2017), apresenta melhorias em relação ao tempo de treinamento das redes ao estender a função identidade da ReLU para o terceiro quadrante, com o objetivo de melhorar a integração entre a função de ativação e a normalização em *batch*.

Figura 12 – Função gráfica que representa o comportamento da Displaced ReLU



Fonte: (MACEDO, 2017)

A decisão de qual função de ativação utilizar nas unidades de uma rede neural não é trivial, sendo em grande parte baseada em experimentos conduzidos com as redes. Em um artigo publicado em 2015(XU et al., 2015), realizam avaliações empíricas de funções de

[6] Disponivel em: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044. Acesso em 4 fev. 2020

ativações retificadas nos resultados de redes neurais convolucionais. O artigo compara o desempenho não só da ReLU tradicional e da *Leaky ReLU*, como de outras variações como a PReLU (HE et al., 2015) e a RReLU que foi introduzida na *Kaggle NDSB Competition*.

2.2.3.3 Softmax

Geralmente utilizada na camada de saída de uma rede neural, a *Softmax* é utilizada para representar uma distribuição de probabilidade entre K classes, sendo a ativação de cada neurônio interpretada como a probabilidade de que aquela seja a classe correta. Isso significa que as ativações da camada de saída estão sempre em um intervalo entre 0 e 1, além de garantir que a soma delas seja igual a 1. A representação matemática da *Softmax* pode ser vista na equação 2.10. O efeito do uso da função exponencial é uma ênfase no maior valor de z, e uma atenuação dos demais valores.

$$\sigma(z) = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}}, paraj = 1, ..., k. \tag{2.10}$$

2.2.4 Generalização

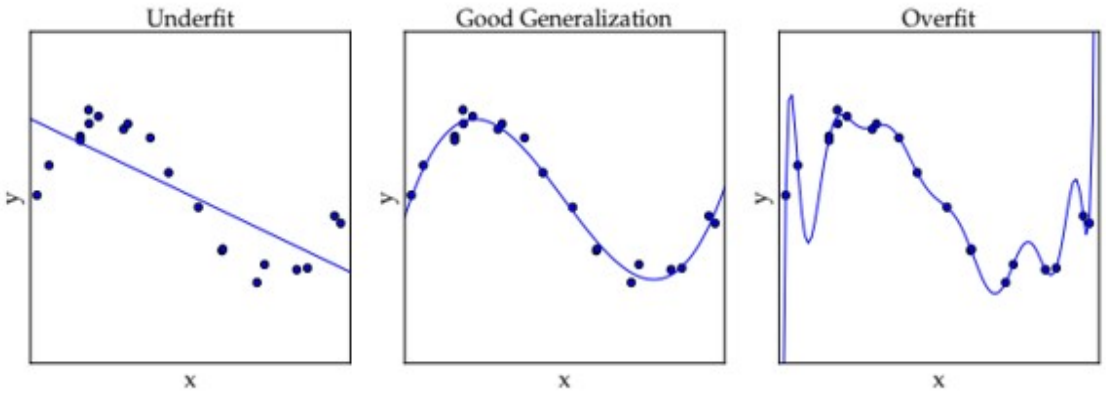
Um bom modelo de *machine learning* precisa apresentar uma boa performance em dados nunca antes vistos. À essa habilidade é dado o nome de generalização. De forma geral, um modelo não é treinado com todos os dados disponíveis, sendo uma boa prática dividir os dados em dois conjuntos. O primeiro deles, o conjunto de treinamento, é utilizado para fornecer os pares de exemplos que a rede utiliza para aprender os seus parâmetros. Já o segundo, o conjunto de teste, apresenta à rede novos dados e avalia o seu desempenho, medindo justamente o erro de generalização.

Quando o modelo não apresenta uma boa performance quanto à essa métrica, podemos suspeitar de duas situações. Se o modelo apresenta um erro alto tanto no conjunto de treinamento quanto no conjunto de teste, nos encontramos em uma situação de subajuste, ou *underfitting*. Uma das razões que pode causar essa condição é a rede não ter capacidade suficiente para aprender a função subjacente. Por outro lado, quando o modelo apresenta um erro baixo durante o treino, que aumenta durante a fase de teste, temos uma situação de sobre-ajuste, ou *overfitting*. Nesse caso, o modelo pode ter uma capacidade excessiva para a função subjacente em questão. A capacidade de uma rede diz respeito a sua quantidade de nós e ao seu número de camadas. Ao aumentarmos a capacidade de uma rede, o espaço das funções que ela consegue representar cresce, já que mais neurônios podem colaborar para expressar diferentes funções.

A figura 13 apresenta três diagramas, cada um deles exibindo a mesma amostragem ruidosa de uma função seno. O primeiro diagrama representa um modelo com *underfitting*, o diagrama do meio representa um modelo que aprendeu bem a função subjacente e o último, um modelo com *overfitting*. Notamos que no caso de *overfitting*, a linha azul

passa perfeitamente em todos os pontos do conjunto de treinamento, porém se comporta erraticamente no intervalo entre eles, o que leva o modelo a ter baixa performance no conjunto de teste.

Figura 13 – Modelo com *underfitting*, *good generalization*, *overfitting*



Fonte: Adaptado da internet[7]

A simples redução da capacidade do modelo não é um bom método de controle do *overfitting*. Ao invés disso, um método bastante utilizado em redes neurais é a regularização, apresentada na seção 2.2.5.

2.2.5 Regularização

Diferentes estratégias de regularização foram desenvolvidas a fim de tentar controlar o *overfitting* dos dados, diminuindo assim o erro de generalização de uma rede neural. Descrevemos nesta seção as estratégias que foram utilizadas durante este trabalho.

2.2.5.1 Regularização L2

A regularização $L2$, também chamada de *weight decay*, é, talvez, a forma de regularização mais comum e consiste em modificar a função de perda adicionando a ela um termo dependente da raiz quadrada da soma dos pesos. Esse termo, denominado termo de regularização, é dado por

$$\tilde{J}(\Theta) = J(\Theta) + \lambda \frac{1}{2} ||w||^2 \tag{2.11}$$

na qual lambda é um hiperparâmetro que representa a força da regularização. Dessa forma, quando tentamos minimizar a função de perda, os pesos com valores muito altos são penalizados. A rede tenta então encontrar um compromisso entre ter um baixo erro de treinamento e manter os pesos perto de zero, limitando assim a complexidade do modelo.

[7] Disponível em: https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html. Acesso em: 4 fev. 2020

Neste compromisso, o valor de lambda vai definir qual lado é mais valorizado. Se lambda for grande, a maior importância será dada a manter pesos pequenos. Caso contrário, a maior importância é dada a minimizar a função de custo original.

2.2.5.2 Parada antecipada

Para aplicar a estratégia de parada antecipada, ou *early stopping*, é necessário a utilização de um conjunto de validação. Esse é um conjunto diferente do conjunto de treino e do conjunto de teste, utilizado para validar o modelo após cada época de treinamento. Normalmente, o erro no conjunto de validação diminui com um tempo, até chegar em um momento em que ele começa a aumentar. Nesse ponto, o modelo está começando a apresentar *overfitting* e aprender características específicas do conjunto de treinamento. A estratégia de *early stopping* consiste em parar o treinamento nesse ponto. Para garantir que estamos no ponto certo e que o erro de validação está realmente caindo, geralmente se espera um número x de épocas em que não há uma melhoria do erro de validação. O treinamento é então parado e o modelo com melhor erro é retornado.

2.2.5.3 *Dataset augmentation*

A melhor forma de diminuir o erro de generalização de um modelo é treiná-lo com um número maior de exemplos de treinamento. Como nem sempre é possível conseguir novos dados para o treinamento, uma das opções é criar mais exemplos de treinamento a partir daqueles já existentes. É possível gerar novos pares de treinamento(x, y) simplesmente aplicando transformações nas entradas x de um classificador.

Para a tarefa de classificação de imagens, essa técnica é particularmente interessante, já que as imagens são um tipo de entrada que incluem uma enorme variedade de fatores de variação, muitos dos quais podem ser facilmente simulados. Realizar operações de rotação, flip ou redimensionamento em algumas imagens do conjunto de treinamento, gerando assim novos pares se mostrou bastante eficaz em alguns trabalhos(SHORTEN; KHOSHGOFTAAR, 2019).

2.2.5.4 *Dropout*

A técnica de *dropout* não foi utilizada nos experimentos realizados durante este trabalho, porém ela é mencionada durante o capítulo de experimentos. Dessa forma, fornecemos uma breve explicação sobre a técnica. O *dropout* foi idealizado por (HINTON et al., 2012) e consiste em eliminar aleatoriamente alguns neurônios da rede durante o processo de treinamento. Podemos pensar que, a cada iteração, onde cada neurônio tem 50% de chance de ser eliminado, estamos treinando uma rede neural diferente. Essa técnica busca evitar a co-adaptação dos neurônios, já que eles não podem contar com a certeza de que os seus

vizinhos estarão ativos, forçando-os assim a se tornarem mais robustos. A fase de teste não utiliza o *dropout*, tendo todos os pesos da rede original disponíveis.

O *dropout* foi utilizado como técnica de regularização na famosa rede AlexNet em 2012, apresentando excelentes resultados para a época. Uma desvantagem apresentada por esse método é o aumento do tempo de treinamento. Segundo Krizhevsky(KRIZHEVSKY; SUTSKEVER; HINTON, 2012), o *dropout* praticamente dobra o número de iterações necessárias para o treinamento convergir.

2.2.5.5 *Batch normalizataion*

Durante o treinamento de redes neurais profundas, existe uma mudança na distribuição das entradas de cada camada à medida que os parâmetros das camadas anteriores são atualizados. Segundo Ioffe Szegedy(IOFFE; SZEGEDY, 2015), esse fenômeno, denominado *Internal Covariate Shift*, dificulta o treinamento das redes, trazendo a necessidade da utilização de menores taxas de aprendizagem e uma cautelosa iniciação dos parâmetros. Para resolver esse problema, Ioffe Szegedy propõem a estratégia de *Batch Normalization*, que consiste em incluir a normalização dos dados na arquitetura do modelo, executando-a em cada um dos mini-batches de treinamento e permitindo assim o uso de taxas de aprendizagem mais altas, o que resulta em uma aceleração do treinamento do modelo. Ainda segundo Ioffe Szegedy, a técnica de *batch normalization* atua como uma técnica de regularização e, em alguns casos, elimina a necessidade da utilização do *dropout*.

2.3 REDES NEURAIS CONVOLUCIONAIS

Uma boa forma de diminuir o erro de generalização em tarefas complexas é utilizar algum conhecimento prévio na hora de compor a arquitetura de uma rede neural. Isso possibilita a redução do número de parâmetros livres sem comprometer a capacidade da rede(LECUN et al., 1989). Redes Neurais Convolucionais, ou CNNs, aplicam esse princípio, sendo um tipo de rede neural especializado no processamento de dados que possuem uma topologia no formato de grade (GOODFELLOW; BENGIO; COURVILLE, 2016). Durante este trabalho, focaremos em CNNs aplicadas a imagens, grades 3D de pixels. Dados temporais, porém, que podem ser vistos como uma grade 1D colhendo amostras em intervalos regulares, são outro exemplo de aplicação onde as CNNs possuem bons resultados(SELVIN et al., 2017). O nome dessas redes vem da operação matemática de convolução. Uma rede neural convolucional é simplesmente uma rede neural que utiliza a operação de convolução no lugar da multiplicação de matrizes tradicional (GOODFELLOW; BENGIO; COURVILLE, 2016).

De forma geral, a arquitetura das CNNs consiste basicamente em camadas de convolução, que extraem características úteis da entrada e produzem os chamados *feature maps*, camadas de *pooling*, cujo objetivo é diminuir o tamanho dos *feature maps*, e cama-

das completamente conectadas, geralmente utilizadas no final da rede para representar as pontuações das classes. Nesta seção, apresentaremos esses novos conceitos e traremos alguns exemplos de arquiteturas de CNNs populares na literatura. Os conceitos teóricos trazidos aqui foram extraídos do (GOODFELLOW; BENGIO; COURVILLE, 2016) e das notas de aula do curso CS231n da Universidade de Stanford[8].

2.3.1 Camada de convolução

Como mencionado anteriormente, as CNNs são tipos especiais de redes neurais, projetadas a partir da suposição de que as suas entradas serão imagens, o que permite com que algumas mudanças sejam feitas em sua arquitetura.

As imagens de entrada de uma CNN geralmente possuem três canais, correspondentes à intensidade das cores vermelha, verde e azul de cada pixel. As imagens presentes nas bases CIFAR, por exemplo, possuem dimensão 32 x 32 x 3. Um só neurônio completamente conectado que estivesse na primeira camada de uma rede convolucional teria 32x32x3 = 3072 pesos. Para evitar esse número extravagante de pesos e, consequentemente, um possível *overfitting* do modelo, as redes convolucionais utilizam o conceito de conectividade local. Dentro deste conceito, cada neurônio se conecta apenas a uma pequena região do volume de entrada. Essa região é chamada de campo receptivo, filtro ou kernel e a sua extensão espacial (altura x largura) é um hiperparâmetro da rede. Apesar de limitada espacialmente, a interação dos neurônios de um filtro se estende por toda a profundidade do volume de entrada. A quantidade de filtros em cada camada também é um hiperparâmetro da rede.

A figura 14 mostra um volume de entrada, em vermelho, e o volume de neurônios de uma camada de convolução, em azul, dando ênfase ao campo receptivo de um neurônio. Vemos que o volume de cor azul possui 5 neurônios, representando 5 filtros, cada um interagindo com a mesma região da entrada.

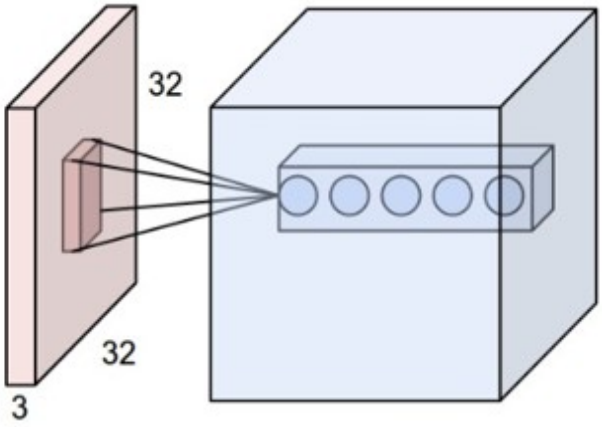
Durante o *forward pass*, cada um dos filtros em uma determinada camada será convoluído com o volume de entrada da camada em que ele se encontra, produzindo um mapa de ativação 2D. Como temos vários filtros, o conjunto dos mapas de ativação será o volume de saída daquela camada.

Intuitivamente, a operação de convolução é feita "deslizando" cada um dos filtros pelo volume de entrada e computando o produto escalar entre eles. A imagem 15 mostra um kernel sendo deslizado ao longo de uma imagem de entrada.

Dois hiper-parâmetros importantes para a camada de convolução são o *stride* e o *zero padding*. Quando um filtro se move pelo volume de entrada, ele se desloca de uma quantidade de pixels indicada pelo *stride*. Se o *stride* for igual a 1, o filtro se move apenas 1

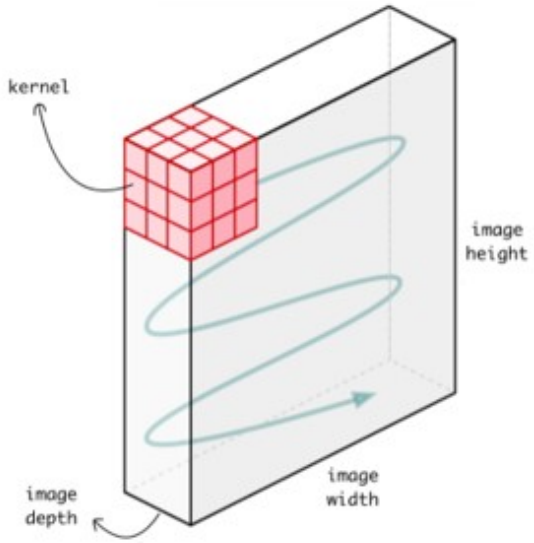
[8] Disponível em: http://cs231n.github.io/convolutional-networks. Acesso em: 4 fev. 2020
[9] Disponível em: http://cs231n.github.io/convolutional-networks/pool. Acesso em: 4 fev. 2020
[10] Disponível em: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610. Acesso em: 4 fev. 2020

Figura 14 – Ilustração de um neurônio com ênfase no campo receptivo de sua entrada



Fonte: Adaptado da internet. [9]

Figura 15 – Ilustração de um kernel e o processo de convolução em uma imagem de entrada



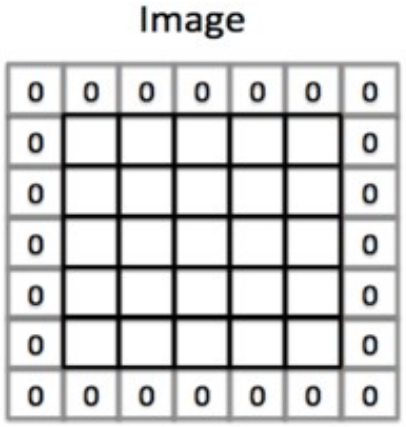
Fonte: Retirado da Internet [10]

pixel de cada vez. Se o *stride* for igual a 2, o filtro se move de 2 em 2 pixels, produzindo um mapa de ativação menor, e assim sucessivamente. Outra forma de controlar o tamanho do mapa de ativação produzido por cada filtro é preencher as bordas da imagem com zeros, procedimento chamado de zero *padding*, demonstrado na imagem 16.

Matematicamente, uma camada de convolução aceita um volume de entrada $W1 \cdot H1 \cdot$

[11] Disponível em: https://medium.com/machine-learning-algorithms/what-is-padding-in-convolutional-neural-network-c120077469cc. Acesso em: 4 fev. 2020

Figura 16 – Ilustração de *zero padding*



Fonte: Retirado da internet.[11]

$D1$ e produz um volume de saída $W2 \cdot H2 \cdot D2$, onde

$$\begin{aligned} W_2 &= \frac{(W_1 - F + 2P)}{S} + 1 \\ H_2 &= \frac{(H_1 - F + 2P)}{S} + 1 \\ D_2 &= K \end{aligned} \tag{2.12}$$

Nessas equações, K representa o número de filtros; F, a extensão espacial deles; S, o *stride* e P, a quantidade de pixels de *zero padding*.

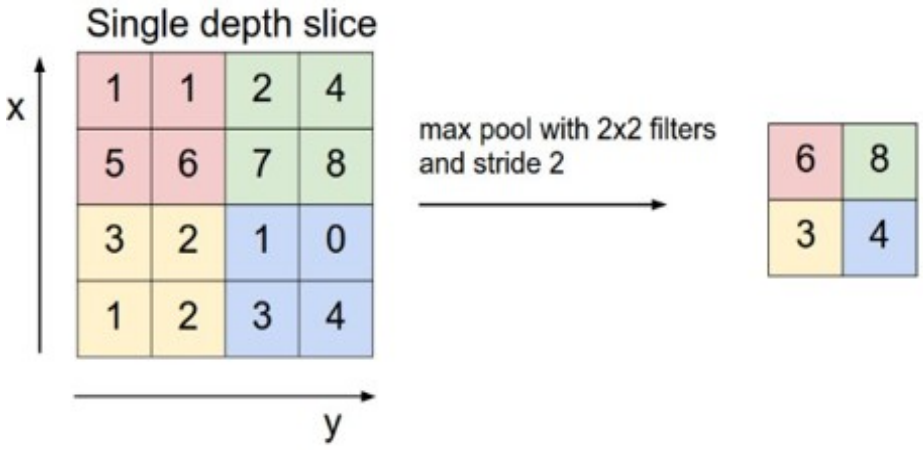
2.3.2 Camada de agrupamento

Na arquitetura das CNNs, a camada de agrupamento, ou camada de *pooling*, é geralmente utilizada após algumas camadas de convolução, com o objetivo de reduzir progressivamente os mapas de ativação ao longo da rede, controlando assim o número de parâmetros do modelo. O procedimento mais comum na implementação da camada de *pooling* é utilizar um filtro *max-pooling*, de tamanho 2x2, que opera de forma independente em cada fatia de profundidade e gera como saída a ativação máxima dentro da região 2x2 de atuação. Esse procedimento é ilustrado na imagem 17 e, quando aplicado com um *stride* de valor igual a 2, descarta 75% das ativações.

Em resumo, a camada de *pooling* aceita um volume $W1 \cdot H1 \cdot D1$ e gera um volume

[12] http://cs231n.github.io/convolutional-networks/pool

Figura 17 – Ilustração do procedimento *max-pooling*



Fonte: Adaptado da internet[12]

$W2 \cdot H2 \cdot D2$ de forma que

$$\begin{aligned} W_2 &= \frac{(W_1 - F)}{S} + 1 \\ H_2 &= \frac{(H_1 - F)}{S} + 1 \\ D_2 &= D_1 \end{aligned} \tag{2.13}$$

onde F representa a extensão espacial dos filtros e S o *stride*. A camada de *pooling* não gera novos parâmetros para a rede, visto que calcula uma função fixa da entrada.

2.4 OTIMIZAÇÃO DE ARQUITETURAS DE CNNS

Essa seção descreve algumas das técnicas de otimização de arquiteturas mais comuns na área de redes neurais convolucionais. A otimização tratada aqui não deve ser confundida com os algoritmos de otimização da função de perda, apresentados na seção 2.2.2.1 (gradiente descendente). Aqui, tratamos de técnicas que buscam otimizar a arquitetura das CNNs, minimizando características como número de parâmetros envolvidos, complexidade computacional e tamanho dos modelos.

As diversas estratégias encontradas na literatura podem, em geral, ser categorizadas em técnicas que comprimem redes pré-treinadas ou técnicas que focam em treinar redes já inicialmente pequenas. Gondim (GONDIM, 2019) faz essa diferenciação denominando a primeira categoria de técnicas de pós-otimização e a segunda de técnicas de pré-otimização.

2.4.1 Técnicas de pós-otimização

Uma das técnicas de pós-otimização bastante utilizada no contexto de redes neurais profundas é a técnica de poda, primeiramente apresentada por (LECUN; DENKER; SOLLA,

1990) como uma forma de reduzir a complexidade e o *overfitting* de um modelo. A poda consiste em remover os parâmetros desnecessários de uma rede. Utilizando essa técnica, (HAN et al., 2015) reduzem em 9x o número de parâmetros da AlexNet, sem perda de acurácia. A estratégia consiste em treinar a rede uma vez pelo método tradicional e em seguida remover da rede conexões que sejam menores do que um valor limiar pré-estabelecido. Em seguida, a rede é treinada novamente, para aprender os pesos finais das conexões que restantes. No ano seguinte, (HAN; MAO; DALLY, 2015) expande esse trabalho, criando a técnica de *Deep Compression*, que une a poda com a quantização e o *Huffman coding*.

A técnica de quantização modifica a representação dos parâmetros de uma rede, procurando reduzir o número de bits utilizados. Em *deep learning*, o formato numérico predominante é o ponto flutuante com precisão de 32-bits. Porém, a partir da quantização, os parâmetros das redes podem ser convertidos para formatos numéricos de precisão mais baixa. Como em (CHOUKROUN; KRAVCHIK; KISILEV, 2019), por exemplo, utiliza uma representação de 4-bits para implantar modelos pré-treinados em plataformas com limitação de *hardware*. Outro trabalho recente (NAYAK; ZHANG; CHAI, 2019) mostra métodos que permitem a representação dos parâmetros em formato de até 3-bits, sem afetar a acurácia dos modelos testados.

Uma outra estratégia interessante é a de destilação (HINTON; VINYALS; DEAN, 2015), que treina uma rede menor a partir de uma rede maior pré-treinada, ou até de um conjunto de redes. O modelo maior é chamado muitas vezes de "professor", enquanto o menor é chamado de "aluno". Aqui, o conhecimento é transferido para uma rede menor a partir da minimização de uma função de perda, onde as saídas corretas são a distribuição de probabilidade predita pela rede professor. A destilação vem mostrando bons resultados, como pode ser visto em (HEGDE et al., 2019) e (MIRZADEH et al., 2019).

2.4.2 Técnicas de pré-otimização

As técnicas de pré-otimização são aquelas utilizadas para a criação de arquiteturas pequenas, que têm como objetivo um número reduzido de parâmetros e de espaço de armazenamento.

A SqueezeNet, rede que serviu de base para este trabalho, é um exemplo de arquitetura que utiliza estratégias de pré-otimização. Três princípios básicos guiaram a escolha da arquitetura da SqueezeNet. O primeiro deles é substituir filtros de convolução 3x3 por filtros 1x1 sempre que possível. O segundo é diminuir o número de canais de entrada dos filtros 3x3 restantes, e o terceiro é realizar *downsample* (utilizando camadas de *pooling*) tardiamente, para que um maior número de camadas tenha mapas de ativação grandes. Esses princípios levaram à idealização do *Fire Module*, um novo bloco para construção de CNNs que utiliza a estratégia de contração e expansão dos mapas de características, reduzindo o número de canais de entrada antes de realizar uma convolução espacial. A

SqueezeNet e os seus *Fire Modules* serão estudados em detalhes na seção 4.3.1. A mesma estratégia de contração e expansão também é vista nas arquiteturas ResNet(HE et al., 2016) e Inception(SZEGEDY et al., 2015).

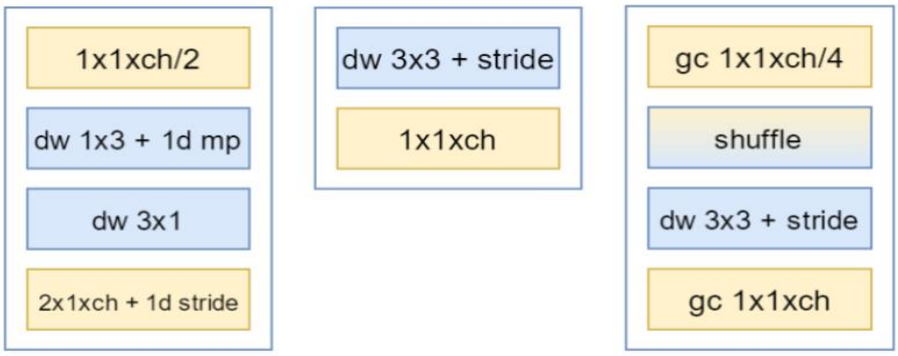
Uma outra estratégia, utilizada por redes como a MobileNet(HOWARD et al., 2017), a ShuffleNet(ZHANG et al., 2018) e a EffNet(FREEMAN; ROESE-KOERNER; KUMMERT, 2018) é modificar a forma como a própria camada de convolução é implementada. Na MobileNet, temos o uso da chamada convolução separável em profundidade, ou DSC, que fatora a convolução comum em uma convolução em profundidade seguida de uma convolução 1x1, denominada convolução pontual, ou convolução ponto a ponto. Por fazer parte da implementação desse trabalho, as DSCs serão apresentadas novamente na seção 4.3.5, juntamente com os benefícios da sua utilização.

Já na ShuffleNet, temos o uso de convoluções de grupo juntamente com operações de *channel shuffle*, ou mistura de canais. Uma convolução de grupo nada mais é do que um conjunto de várias convoluções, com cada uma delas pegando uma parte do canal de entrada, enquanto a operação de mistura de canais consiste em embaralhar aleatoriamente os canais de saída da convolução de grupo. Essa operação é introduzida para que não haja um enfraquecimento da representação, uma vez que as saídas das convoluções de um determinado grupo seriam relacionadas apenas às entradas daquele grupo.

Por fim, a EffNet propõe um novo bloco a ser utilizado ao invés das convoluções tradicionais. O bloco proposto basicamente divide a convolução separável em profundidade em duas camadas lineares. Isso permite realizar uma operação de *maxpooling* depois da primeira camada espacial, diminuindo a quantidade de cálculos necessários para a segunda. Uma outra otimização que a EffNet propõe é substituir a primeira camada de convolução tradicional, presente tanto na MobileNet quanto na ShuffleNet, por um bloco EffNet.

A imagem 18 apresenta uma comparação entre as camadas de convolução utilizadas na EffNet, na MobileNet e na ShuffleNet.

Figura 18 – Comparação entre as camadas de convolução utilizadas na EffNet, na MobileNet e na ShuffleNet. 'ch' indica o número de canais de saída, enquanto 'dw', 'gc' e 'mp' indicam, respectivamente, camadas de convolução em profundidade, convolução em grupo e *max-pooling*.



Fonte: Retirado da internet.[13]

2.5 ARQUITETURAS INVESTIGADAS

Nesta seção, apresentaremos as arquiteturas de CNNs utilizadas neste trabalho para fins de comparação da performance da SqueezeNet-DSC. Por ser a base da SqueezeNet-DSC, a SqueezeNet original será vista com mais detalhes em um capítulo posterior. As informações apresentadas nessa seção foram retiradas dos artigos originais que introduziram as arquiteturas em questão.

2.5.1 AlexNet

A AlexNet, apresentada por Krizhevsky et. al em 2012 (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), contém oito camadas com parâmetros livres, sendo as três primeiras camadas de convolução e as três últimas, camadas completamente conectadas. A última camada alimenta uma Softmax, que produz a distribuição de probabilidade entre as 1000 classes do dataset ImageNet.

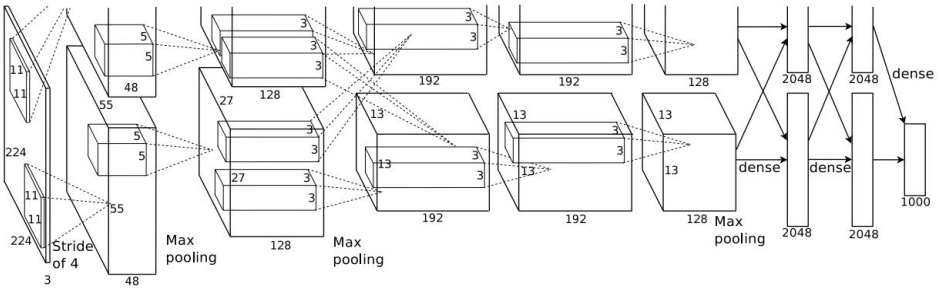
Krizhevsky utiliza o conceito de convolução de grupo para realizar o treinamento da rede utilizando duas GPUs, com o detalhe adicional de que ela controla as camadas nas quais as GPUs podem trocar informação entre si. Nesse caso, a troca acontece na terceira camada de convolução e nas camadas completamente conectadas.

A AlexNet utiliza a função de ativação ReLU nos seus neurônios, filtros *max-pooling* na sua camada de agrupamento e as estratégias de *data augmentation* e *dropout* para a

[13] Disponível em: https://towardsdatascience.com/3-small-but-powerful-convolutional-networks-27ef86faa42d. Acesso em: 4 fev. 2020

redução do seu erro de generalização. Dentro da estratégia de *data augmentation*, duas técnicas são utilizadas. A primeira delas consiste em gerar novas imagens a partir de translações e reflexões horizontais das imagens e a segunda a partir da alteração da intensidade dos seus canais RGB.

Figura 19 – Arquitetura AlexNet



Fonte: (KRIZHEVSKY; SUTSKEVER; HINTON, 2012)

2.5.2 Visual Graphic Group - VGG

O trabalho que apresentou a arquitetura VGG(SIMONYAN; ZISSERMAN, 2014) tinha como objetivo investigar o impacto da profundidade de uma CNN na sua acurácia. Para isso, vários experimentos foram realizados utilizando redes com a mesma arquitetura básica, porém com diferentes quantidades de camadas de convolução, as *ConvNets*.

Nesta arquitetura, uma pilha de *ConvNets*, de tamanho variado, é seguida por três camadas completamente conectadas e uma camada *Softmax*. Assim como a AlexNet, as redes VGG também utilizam a função de ativação ReLU e os filtros de *max-pooling*.

A imagem abaixo mostra as configurações da arquitetura para diferentes quantidades de *ConvNets*. A configuração VGG com 19 camadas, mais conhecida como VGG-19, foi a vencedora do desafio ILSVRC no ano de 2014.

Figura 20 – Arquitetura VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fonte: (SIMONYAN; ZISSERMAN, 2014)

2.5.3 MobileNet

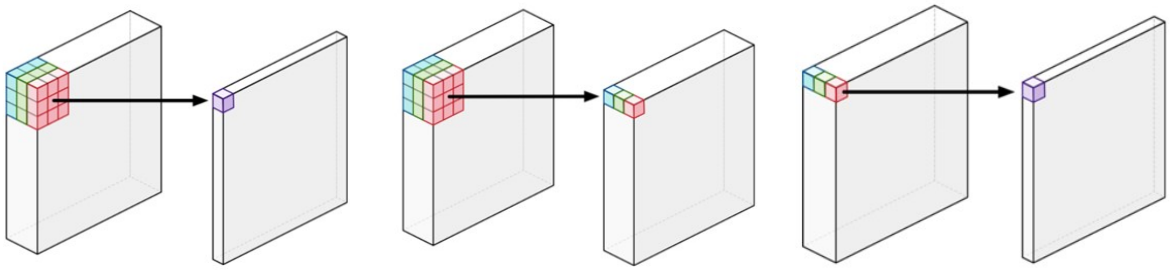
A arquitetura da rede MobileNet, apresentada por (HOWARD et al., 2017), se baseia na utilização das DSCs. Publicadas inicialmente em (SIFRE; MALLAT, 2014) e utilizadas posteriormente nos módulos Inception(IOFFE; SZEGEDY, 2015), as DSCs fatoram a convolução tradicional, que passa a ser executada em dois passos. O primeiro passo passa a ser uma convolução em profundidade, que aplica um único filtro para cada canal de entrada. As saídas desses filtros são em seguida combinadas através do segundo passo, uma convolução ponto a ponto.

A figura abaixo ilustra, respectivamente, uma convolução tradicional na figura 21 (a), uma convolução em profundidade na figura 21 (b) e uma convolução ponto a ponto na figura 21 (c).

Como mencionado anteriormente, a arquitetura da MobileNet é baseada em camadas de convolução DSC, com exceção da primeira camada, que é uma convolução tradicional. A última camada é completamente conectada e seguida por uma Softmax. Essa arquitetura pode ser vista na figura 22. A MobileNet usa a função de ativação ReLU, a estratégia de regularização *batchnormalization*.

[14] Disponível em: http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/. Acesso em: 4 fev. 2020

Figura 21 – Comparação entre os tipos de convolução.



(a) Tradicional. (b) Profundidade. (c) Ponto-a-ponto.

Fonte: Retirado da internet.[14]

Figura 22 – Arquitetura MobileNet

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fonte: (HOWARD et al., 2017)

3 MODELO PROPOSTO

A SqueezeNet, rede neural convolucional que o presente trabalho objetiva aprimorar, foi proposta inicialmente em 2016 por pesquisadores da DeepScale, da Universidade da Califórnia, Berkley e da Universidade de Stanford. O objetivo inicial da pesquisa que levou à SqueezeNet era construir uma nova arquitetura de CNNs que necessitasse de uma menor quantidade de parâmetros, porém mantendo um nível de precisão equivalente à de um outro modelo bem conhecido, o modelo AlexNet (IANDOLA et al., 2016).

No artigo publicado por Iandola et.al (2016), três grandes vantagens de modelos de aprendizagem de máquina com arquitetura pequena são apresentadas. São elas, um treinamento distribuído mais eficiente, menos sobrecarga ao exportar novos modelos para clientes e a possibilidade de implementação dessas redes em sistemas embarcados. A busca por uma nova arquitetura de redes pequenas é motivada principalmente por essas vantagens e se baseia em três estratégias principais.

A primeira delas é substituir os filtros 3x3 por filtros 1x1, dado que estes possuem 9 vezes menos parâmetros do que aqueles. A segunda constitui em diminuir o número de canais de entrada de filtros 3x3, visto que a quantidade de parâmetros também é proporcional a este número. Por fim, a terceira consiste em adiar o uso de técnicas de *downsampling* (redução da amostragem) na rede para que um número maior de camadas de convolução tenham mapas de ativação grandes. A intuição por trás dessa estratégia é que, mantidas constantes as outras variáveis, mapas de ativação maiores levam a maiores níveis de precisão na tarefa de classificação.

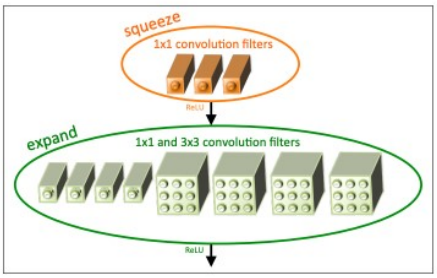
Dessa forma, temos que as duas primeiras estratégias focam em diminuir a quantidade de parâmetros tentando preservar a acurácia, enquanto a última busca maximizar a acurácia em face de um número já limitado de parâmetros. Fundamentado nelas, um novo bloco básico para construção de CNNs é sugerido, sendo denominado de *Fire Module*.

De acordo com Iandola et. al (2016), um *Fire Module* é composto de uma camada de convolução compressora (*squeeze layer*), que possui apenas filtros 1x1, alimentando uma camada expansora (*expand layer*). A camada expansora, por sua vez, possui uma mistura de filtros de convolução 1x1 e 3x3. Essa estrutura é demonstrada na figura 23.

Um *Fire Module* possui então 3 hiperparâmetros, sendo eles o número de filtros na camada compressora, s1x1, o número de filtros e1x1 na camada expansora, 1x1, e o número de filtros 3x3 ainda na camada expansora, e3x3. É recomendado que s1x1 seja menor do que (e1x1 + e3x3), para que a camada compressora ajude a limitar o número de canais de entrada para os filtros 3x3, de acordo com a estratégia 2, detalhada acima.

A macroarquitetura da SqueezeNet é exibida na figura 24, começando com uma camada de convolução isolada (conv1), seguida de 8 *Fire Modules* (fire2-9) e finalizando com uma última camada de convolução (conv10). O número de filtros por *Fire Module* é

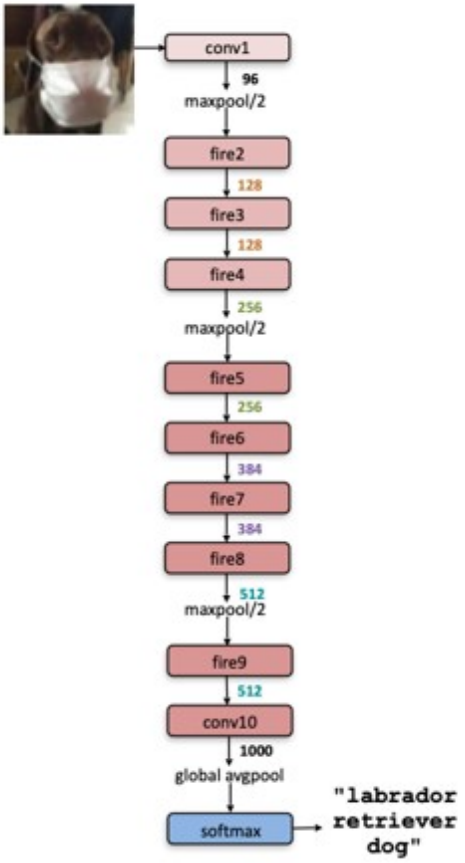
Figura 23 – Ilustração de um *Fire Module*



Fonte: (IANDOLA et al., 2016)

incrementado gradualmente do começo da rede até o final. Além disso, é aplicado *max-pooling* com um *stride* de 2 depois das camadas de conv1, fire4, fire8 e conv10. Todas as escolhas mencionadas acima foram tomadas seguindo as três estratégias que norteiam todo o trabalho.

Figura 24 – Macroarquitetura da SqueezeNet



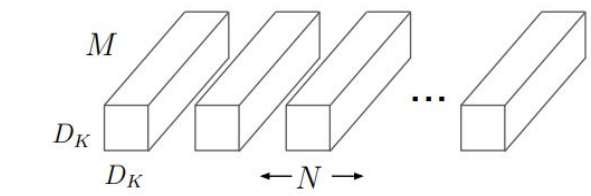
Fonte: Adaptado de (IANDOLA et al., 2016)

Como pôde ser visto na descrição da SqueezeNet, a rede possui filtros de convolução 3x3 tradicionais na estrutura de seus *Fire Modules*. Neste trabalho, propomos substituir

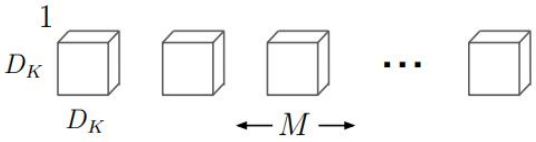
esses filtros de convolução tradicionais por filtros de convolução separáveis em profundidade, os filtros DSC, e em seguida avaliar o impacto dessa substituição em algumas métricas da rede. A esta nova proposta de implementação da SqueezeNet foi dado o nome de SqueezeNet-DSC.

Como visto anteriormente, a DSC foi inicialmente introduzida por (SIFRE; MALLAT, 2014) e fatora uma convolução tradicional em duas partes, a primeira sendo uma convolução em profundidade (*depthwise convolution*) e a segunda uma convolução 1x1 chamada convolução pontual (*pointwise convolution*). Essa fatoração busca reduzir a quantidade de parâmetros utilizados. A figura 25 mostra a diferença entre um filtro de convolução normal e um filtro DSC.

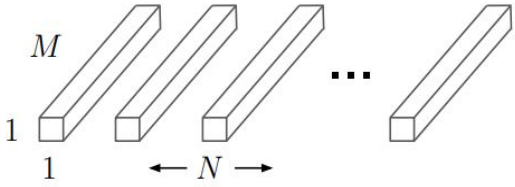
Figura 25 – Os filtros convolucionais padrão em (a) são substituídos por duas camadas: convolução em profundidade em (b) e convolução ponto a ponto em (c) para criar um filtro separável em profundidade.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Fonte: (HOWARD et al., 2017)

Sabemos que o número de parâmetros P_S de uma camada de convolução padrão pode ser calculado como:

$$P_S = D_k * D_k * M * N, \tag{3.1}$$

no qual D_k é o tamanho do kernel, M é o número de canais de entrada e N é o número de

canais de saída. Já o número de parâmetros de uma camada DSC é calculado como:

$$P_{\text{DSC}} = M * D_k * D_k + M * N. \tag{3.2}$$

Dessa forma, sendo N_1 o número de canais de saída de uma *squeeze layer* e N o número de canais de saída de uma *expand layer*, o número de parâmetros de um *Fire Module* tradicional pode ser escrito como:

$$P_{\text{FIRE}} = M * N_1 + N_1 * \frac{N}{2} + D_k * D_k * N_1 * \frac{N}{2}, \tag{3.3}$$

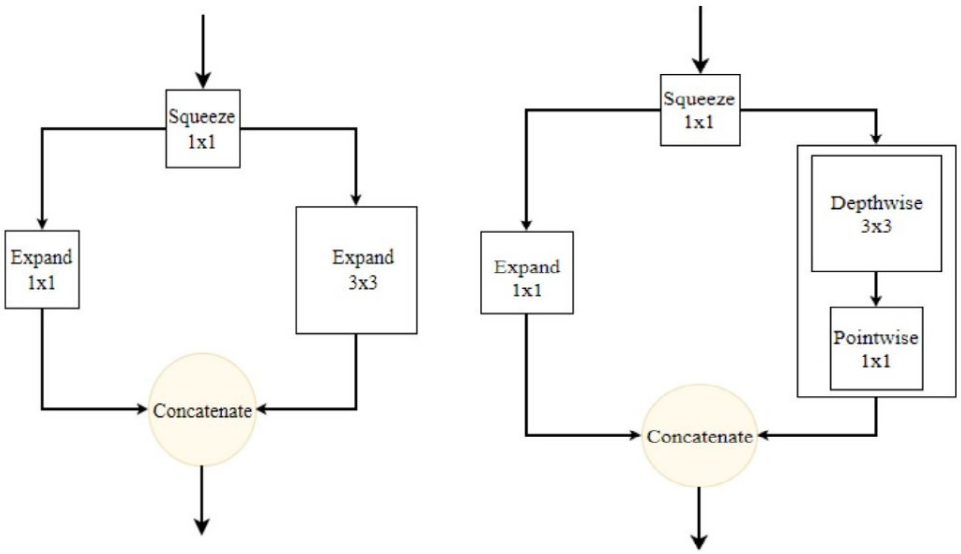
o que faz com que o número de parâmetros de um *Fire Module* implementado com convoluções do tipo DSC seja

$$P_{\text{FIRE - DSC}} = M * N_1 + N_1 * N + N_1 * D_k * D_k. \tag{3.4}$$

Para uma melhor visualização, vamos tomar como exemplo uma camada com um número N de canais de saída igual a 128, um número de canais de entrada M igual a 96, um número de canais intermediários N_1 igual a 16 e um kernel de tamanho D_k igual a 3. Nesse caso, utilizando a implementação tradicional da convolução, teríamos 11.776 parâmetros contra apenas 3.728 da implementação DSC, uma redução de 68.34%. Tendo isto em vista, é esperada uma redução no número de parâmetros da SqueezeNet uma vez que os filtros de convolução dos *Fire Modules* forem substituídos por filtros DSC.

A figura 26 ilustra a diferença entre um *Fire Module* tradicional e um *Fire Module* modificado.

Figura 26 – Ilustração da diferença de um *Fire Module* normal para um modificado



Fonte: (SANTOS et al., 2018)

Para uma melhor compreensão dos efeitos desta substituição, a nova rede proposta será aplicada ao problema de classificação de imagens nos datasets CIFAR-10 e CIFAR-

100, além disso, quatro métricas serão avaliadas e em seguida comparadas às de outras redes já conhecidas. Essas métricas são:

1. Acurácia
2. Número de parâmetros
3. Tamanho de armazenamento
4. Tempo de inferência de um único exemplo de teste

Os experimentos realizados, assim como seus resultados e a análise dos dados obtidos serão explicados com detalhes no capítulo seguinte.

4 EXPERIMENTOS

Este capítulo apresenta os resultados dos experimentos realizados durante o desenvolvimento desde trabalho. A seção 4.1 introduz as bases de dados utilizadas durante os testes. Em seguida, a seção 4.2 descreve detalhes dos experimentos realizados, bem como as métricas usadas para avaliá-los. Por fim, a seção 4.3 apresenta as redes neurais convolucionais avaliadas, com alguns detalhes referentes às adaptações feitas para que estas pudessem ser treinadas nas bases de dados escolhidas.

4.1 BASES DE DADOS

Apesar das redes apresentadas neste capítulo terem sido inicialmente idealizadas para a base de dados Imagenet, os experimentos de classificação de imagens realizados durante este trabalho utilizaram as bases de dados CIFAR-10 e CIFAR-100. Essa escolha foi feita devido à menor quantidade de imagens e ao tamanho reduzido das mesmas, diminuindo assim o tempo gasto com cada experimento e possibilitando a comparação da SqueezeNet-DSC com uma maior quantidade de redes, além de uma maior variação de hiperparâmetros testados.

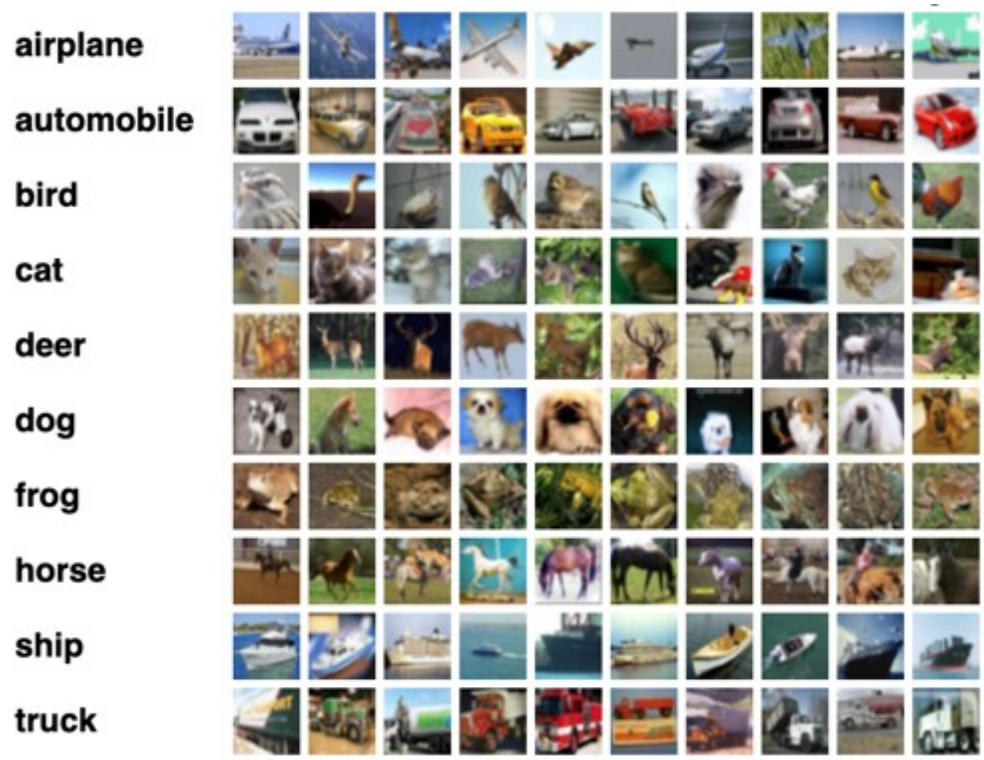
As bases CIFAR, amplamente utilizadas em pesquisas na área de aprendizagem de máquina, são subconjuntos rotulados da base de dados *80 million tiny images* (TORRALBA; FERGUS; FREEMAN, 2008). Ambas foram coletadas pelos pesquisadores Alex Krizhevsky, Vinod Nair e Geoffrey Hinton. O nome CIFAR vem da instituição que financiou o projeto de criação das bases, a *Canadian Institute for Advanced Research*, uma organização global sem fins lucrativos que tem como objetivo reunir pesquisadores de todo o mundo, dando a eles suporte para abordar questões importantes para a ciência e a humanidade. Detalhes da metodologia utilizada na elaboração dessas bases podem ser encontrados no relatório técnico disponibilizado por Alex Krizhevsk(KRIZHEVSKY; SUTSKEVER; HINTON, 2012), de onde foram retiradas as informações mencionadas nesta seção.

4.1.1 CIFAR-10

A CIFAR-10 possui 60.000 imagens coloridas de tamanho 32x32 pixels, divididas em 10 classes de 6.000 imagens. As classes são “avião”, “automóvel”, “pássaro”, “gato”, “veado”, “cachorro”, “sapo”, “cavalo”, “navio” e “caminhão”. As imagens são ainda divididas em cinco lotes de treinamento e um lote de teste, cada um deles com 10.000 imagens. O lote de teste é construído de forma balanceada, contendo exatamente 1.000 imagens de cada classe, selecionadas de forma aleatória. As imagens de treinamento também são escolhidas de forma balanceada, com quantidade de 5.000 imagens de cada classe. É interessante ainda ressaltar que as classes são mutuamente exclusivas, não havendo, por exemplo,

imagens que possam ser rotuladas tanto como “automóvel” quanto como “caminhão”. A imagem 27 mostra 10 exemplos de imagens de cada uma das classes existentes na CIFAR-10.

Figura 27 – Classes disponíveis na base de dados CIFAR-10



Fonte: Retirado da internet.[1]

4.1.2 CIFAR-100

A CIFAR-100 é composta por um outro subconjunto de imagens da *80 million tiny images*, coletadas com a mesma metodologia da CIFAR-10, porém agora contendo 100 classes, cada uma com 600 imagens. Para cada uma das classes, existe um total de 500 imagens de treinamento e 100 de teste. As 100 classes são ainda agrupadas em 20 superclasses, fazendo com que cada item tenha um rótulo mais genérico (*fine label*) e um rótulo mais específico (*coarse label*). Segundo Krizhevsky, a ideia é que classes dentro da mesma superclasse são mais difíceis de serem distinguidas do que classes que pertencem a superclasses diferentes. Uma lista com essa estrutura de classes pode ser encontrada no relatório técnico (KRIZHEVSKY; NAIR; HINTON, 2009) das bases e sua versão resumida é apresentada na imagem 28.

Da mesma forma que na base CIFAR-10 a CIFAR-100 também só possui imagens rotuladas com uma classe. O *torchvision*, pacote do framework Pytorch, disponibiliza aos usuários bases de dados populares, entre elas a CIFAR-10 e a CIFAR-100, que

[1] Disponível em: https://www.cs.toronto.edu/ kriz/cifar.html. Acesso em: 4 fev. 2020

podem ser acessadas a partir do comando **torchvision.datasets.cifar10** e **torchvision.datasets.cifar100**. Os comandos possuem ainda opções para realizar transformações nas imagens e para realizar ou não o download delas em um diretório local.

Atualmente, a menor taxa de erro nas bases de dados CIFAR foi reportada por Kolesnikov et. al. (2019) e utiliza um método de *transfer learning* denominado *Big Transfer* (BiT) em redes com arquitetura ResNet. O BiT apresenta uma taxa de erro de 0,7% na CIFAR-10 e 6,4% na CIFAR-100.

Figura 28 – Classes disponiveis na base de dados CIFAR-100

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Fonte: Retirado da internet.[2]

4.2 METODOLOGIA EXPERIMENTAL

Os experimentos realizados neste trabalho têm como objetivo avaliar a performance do modelo CNN proposto, o modelo SqueezeNet-DSC. Para isso, a rede é aplicada ao problema de classificação de imagens nas bases de dados CIFAR, juntamente com outras quatro redes. Essas redes são a SqueezeNet na sua estrutura original, a MobileNet, a AlexNet e a VGG19. A MobileNet foi escolhida por também ser uma rede projetada para ambientes com restrição de poder computacional, além de usar a própria DSC em sua arquitetura. A AlexNet foi a rede utilizada como base de comparação no artigo que introduz a SqueezeNet e a VGG19 é uma rede amplamente utilizada na literatura para comparação de performance de novos modelos propostos. As métricas de desempenho utilizadas para realizar a comparação de performance das redes escolhidas são:

[2] Disponível em: https://www.cs.toronto.edu/ kriz/cifar.html. Acesso em: 4 fev. 2020

- Acurácia: taxa de decisões corretas realizadas pelo classificador. Calculada como o número de decisões corretas dividido pelo número total de decisões tomadas, no qual $acc = 100 * (decisões corretas)/(total de decisões)$;
- Número de parâmetros: o número de parâmetros treináveis de uma rede é calculado como a soma dos pesos e dos vieses de cada uma de suas camadas;
- Tamanho de armazenamento: após o treinamento, cada modelo é armazenado em um arquivo t7. O tamanho de armazenamento da rede é tido então como o tamanho do arquivo t7 que contém a rede treinada. Onde .t7 designa a extensão do arquivo que é escrito/lido pelo pytorch;
- Tempo de inferência de um único exemplo de teste: durante a fase de teste, o tempo de cada inferência é salvo e, para o valor final, é considerada a média desses tempos, juntamente com o desvio padrão.

A fim de comparar a performance dos cinco diferentes modelos analisados, foi necessário criar um padrão para que o treinamento dos modelos fosse o mais homogêneo possível. Antes de iniciar o treinamento das redes, as imagens da base CIFAR foram tratadas utilizando normalização e *data augmentation*. A técnica de *data augmentation* utilizada foi a realização de um *flip* horizontal randômico nas imagens de entrada. Em seguida, o treinamento é realizado em lotes de tamanho 128, por no máximo 200 épocas.

Para a taxa de aprendizagem, ou *learning rate*(lr), foi escolhido utilizar um método de ajuste automático baseado na estabilização da perda. Para isso, o treinamento é iniciado com um valor de lr de 0.01, que é atualizado automaticamente, sendo dividido por 10 quando a perda se torna estável. A perda é considerada estável quando o seu valor varia menos do que 0.01 em relação à época anterior. Após a quarta estabilização do valor da perda, o treinamento é interrompido. A função de perda utilizada foi a de entropia cruzada ou *cross-entropy loss*. A escolha dos hiperparâmetros foi feita de forma manual, após diversos testes realizados com valores usuais encontrados na literatura.

Os cinco modelos foram treinados e testados em uma GPU NVIDIA TITAN Xp com 12GB de memória e em seguida testados novamente em uma CPU Intel® Core™i7-7700K e 16GB RAM para que o tempo de inferência das redes em GPU e em CPU pudesse ser comparado, a fim de analisar a dependência do modelo ao hardware utilizado na sua fase de teste.

4.3 ARQUITETURAS INVESTIGADAS

Esta seção descreve as arquiteturas dos cinco modelos utilizados durante os experimentos e eventuais adaptações que tenham sido necessárias para que eles pudessem ser treinados nas bases CIFAR, dado que as suas implementações originais foram realizadas

para usarem dados da base de dados ImageNet, que possui imagens com diferentes dimensões. Algumas adaptações de modelos populares na literatura para as bases CIFAR foram disponibilizadas por membros da comunidade *open source* de deep learning através da plataforma GitHub. Entre elas, podemos citar dois repositórios que serviram de base para as nossas implementações. O primeiro deles, o pytorch-playground[3], foi criado por Aaron Chen da Universidade Tsinghua e contém adaptações da SqueezeNet, da AlexNet e da VGG19. O segundo, o pytorch-cifar [4], foi criado por Kuang Liu da universidade de Zhejiang e contém uma implementação da MobileNet.

4.3.1 Modelo SqueezeNet

A SqueezeNet foi originalmente implementada utilizando o Caffe, um framework de *deep learning* desenvolvido pelo laboratório Berkeley AI Research (BAIR) e colaboradores, e o seu código pode ser encontrado no GitHub de Forrest Iandola[5]. Apesar de ter sido desenvolvido para o Caffe, a rede foi adaptada para outras plataformas por membros da comunidade de deep learning e algumas dessas adaptações são mencionadas por Iandola. Entre elas, podemos citar a implementação da SqueezeNet no framework Pytorch por Marat Dukhan[6], que serviu como guia para a nossa própria adaptação da rede.

Na implementação da SqueezeNet, feita para o *framework* Pytorch, um *Fire Module* é composto por três contêineres sequenciais. O primeiro deles implementa a camada de compressão, ou *squeeze layer*, formada por uma convolução 2d com kernel igual 1. O segundo representa a parte da camada de expansão que contém os filtros 1x1 e também é implementado por uma camada de convolução 2d com kernel igual a 1, seguida de *batch normalization* e Relu. Por fim, o terceiro contêiner representa a parte da camada de expansão responsável pelos filtros 3x3 e difere dos dois primeiros por ter o kernel da sua convolução 2d igual a 3. A camada de expansão é fatorada dessa forma porque o Pytorch não suporta de forma nativa uma camada de convolução com filtros de tamanhos distintos. Tanto a convolução 2d, quanto as operações de *batch normalization* e Relu já são disponibilizadas no módulo torch.nn.

Entre as modificações necessárias para que o modelo fosse treinado nas bases CIFAR, temos a adição de *batch normalization* após as camadas de convolução do Fire Module e a remoção do *dropout* na última camada de convolução. Além disso, na primeira camada de convolução, o tamanho do kernel foi diminuído de 7x7 para 3x3 e o *stride* passou de 2 para 1. Essas modificações foram retiradas da implementação disponibilizada no repositório pytorch-playground e são necessárias para que a informação não desapareça ao longo da rede, dado que as imagens da CIFAR são de tamanho 32x32 enquanto a maioria das aplicações da ImageNet trabalha com imagens de tamanho 256x256.

[3] https://github.com/aaron-xichen/pytorch-playground
[4] https://github.com/kuangliu/pytorch-cifar
[5] https://github.com/forresti/SqueezeNet
[6] https://github.com/pytorch/vision/blob/master/torchvision/models/squeezenet.py

Em sua implementação original, a SqueezeNet apresentou uma acurácia top-1 e top-5 de 57.5% e 80.3%, respectivamente, na base de dados ImageNet.

4.3.2 Modelo MobileNet

A arquitetura MobileNet foi apresentada originalmente em 2017 por Howard et. al (HOWARD et al., 2017) como uma classe de modelos eficientes para aplicações embarcadas de visão computacional e se baseia no conceito de convoluções separáveis em profundidade para criar redes neurais convolucionais pequenas.

Considerando que uma DSC possui duas camadas, uma relativa à convolução em profundidade e outra à convolução pontual, a MobileNet possui 28 camadas, das quais apenas a primeira delas é uma convolução tradicional. Todas as camadas são seguidas de *batchnormalization* e ReLU, com exceção da última, totalmente conectada, que é seguida de uma camada Softmax. Além disso, uma camada de *average pooling* precede a camada totalmente conectada, a fim de reduzir a resolução espacial para 1. A MobileNet foi inicialmente treinada no framework Tensorflow e uma implementação desse modelo pode ser encontrada no repositório do Tensorflow[7] no GitHub. Nossa implementação, porém, se baseou no código disponibilizado pelo repositório pytorch-playground. As adaptações feitas na rede para o treinamento na CIFAR incluem diminuir o *stride* de 2 para 1 na primeira e na penúltima camada de convolução, além de modificar o tamanho do kernel da camada de *average pooling* de 7x7 para 2x2. Em sua implementação original, a MobileNet apresentou uma acurácia top-1 de 70.6% na base de dados ImageNet.

4.3.3 Modelo AlexNet

A AlexNet(KRIZHEVSKY; SUTSKEVER; HINTON, 2012) aumentou a popularidade das redes neurais convolucionais ao ganhar o desafio ILSVRC em 2012, atingindo uma acurácia top-5 de 84,6% na ImageNet. Sua arquitetura original consiste em cinco camadas de convolução, três camadas completamente conectadas e uma camada Softmax. Camadas de *max-pooling* seguem a primeira, segunda e quinta convoluções e a não-linearidade ReLU é aplicada à saída de todas as camadas de convolução e de todas as camadas completamente conectadas. Além disso, a técnica de *dropoput* é utilizada nas duas primeiras camadas completamente conectadas.

As duas primeiras convoluções da AlexNet sofreram alterações para o treinamento nas bases CIFAR. Na primeira delas, o tamanho do kernel foi modificado de 11x11 para 3x3, o *padding* de 2 para 1 e o *stride* de 4 para 1. Na segunda, o kernel foi modificado de 5x5 para 3x3, enquanto o padding passou de 2 para 1. Por fim, o *dropout* nas duas camadas completamente conectadas foi substituído por uma camada de *batch normalization*.

[7] https://github.com/tensorflow/tensorflow

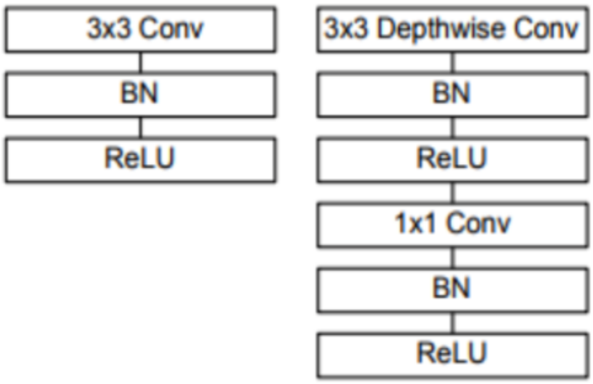
4.3.4 Modelo VGG-19

Criada a partir do desejo de investigar o impacto da profundidade de uma CNN na sua acurácia, a VGG-19(SIMONYAN; ZISSERMAN, 2014) ganhou o desafio ILSVRC no ano de 2014 com uma acurácia top-1 de 74,5% e top-5 de 92% na ImageNet. A numeração 19 se deve ao fato da rede possuir 19 camadas com pesos, sendo as primeiras 16 camadas de convolução, seguidas de 3 camadas completamente conectadas. As convoluções utilizam sempre kernels de tamanho 3x3 e *stride* 1. Camadas de *max-pooling* seguem as convoluções de número 2, 4, 8, 12 e 16. Após a última camada completamente conectada existe uma camada Softmax. A implementação da VGG-19 para CIFAR, encontrada no repositório pytorch-playground, mantém as camadas de convolução e *max-pooling*, porém conta com apenas uma camada completamente conectada.

4.3.5 Modelo SqueezeNet-DSC

Como visto anteriormente, o *Fire Module* da SqueezeNet é composto por uma camada de compressão, contendo apenas filtros de convolução 1x1, e uma camada de expansão, contendo tanto filtros 1x1, quanto filtros 3x3. São exatamente os filtros de convolução 3x3 na camada de expansão que nós pretendemos modificar, substituindo-os por filtros de convolução separáveis em profundidade. A figura 29 representa o diagrama de um *Fire Module* tradicional ao lado de um *Fire Module* modificado, com o filtro de convolução tradicional 3x3 sendo substituído por uma convolução em profundidade seguida de uma convolução pontual.

Figura 29 – Esquerda: Camada convolucional padrão com *Batch Normalization* e ReLU. Direita: Convolução em Profundidade e Convolução ponto a ponto seguidas de *Batch Normalization* e ReLU



Fonte: (HOWARD et al., 2017)

A SqueezeNet-DSC[8] usa como base a implementação da SqueezeNet para CIFAR, po-

[8] https://github.com/ags5/DL2017.2

rém modificando o terceiro contêiner do *Fire Module*, com a convolução em profundidade sendo implementada com a ajuda do módulo torch.nn.

De acordo com a documentação do Pytorch [9], um dos parâmetros que a convolução 2d pode receber é o número de grupos. Esse número controla as conexões entre as entradas e as saídas da camada. Se o número de grupos for igual a 1, todas as entradas são convoluídas com todas as saídas. Por outro lado, se o número de grupos for igual a 2, teremos o equivalente a duas camadas de convolução, cada uma vendo metade dos canais de entrada. Quando o número de canais de entrada, $in_{channels}$, se iguala ao número de grupos, n_{groups}, cada canal de entrada é convoluído com seu próprio conjunto de filtros, de tamanho $(in_{channels})/(out_{channels})$, onde $out_{channels}$ representa o número de canais de saída.

A convolução em profundidade acontece quando o número de grupos é igual ao número de canais de entrada e o número de canais de saída é múltiplo do número de canais de entrada. Ou seja, quando $n_{groups} = in_{channels}$ e $out_{channels} = in_{channels} * K$, onde K é um número inteiro positivo.

[9] https://pytorch.org/docs/stable/index.html

5 RESULTADOS E DISCUSSÃO

Os cinco modelos implementados neste trabalho foram treinados de acordo com o procedimento descrito na seção 4.2. Em seguida, para cada modelo, os valores de acurácia, número de parâmetros, tamanho do modelo treinado e tempo de inferência de um único exemplo de teste foram analisados. Este capítulo descreve os resultados obtidos a partir desses experimentos.

Durante o capítulo, por vezes vamos nos referir à base de dados CIFAR-10 por C10 e à base CIFAR-100 como C100. Além disso, a implementação de uma rede neural específica para uma determinada base de dados será indicada pelo nome do modelo seguido pelo nome do dataset. A implementação da rede SqueezeNet feita para o dataset CIFAR-10, por exemplo, poderá ser referenciada como SqueezeNet-C10.

5.1 MODELO SQUEEZENET

Enquanto a implementação original da SqueezeNet atingiu uma acurácia top-1 de apenas 57.5% na base de dados ImageNet, o resultado atingido pela adaptação da rede para a base CIFAR-10 foi de 89.39%. Esse aumento era esperado, dado que enquanto as bases CIFAR possuem um total de 60.000 imagens de tamanho 32x32x3, a ImageNet possui 1,2 milhões de imagens, de tamanhos variados, divididas em 1000 categorias, aumentando assim em muito a complexidade da tarefa de classificação de imagens. É natural, também, esperar que a acurácia da tarefa de classificação na CIFAR-100 seja um pouco menor do que aquela na CIFAR-10, já que o classificador precisa aprender um número maior de classes. Essa suposição é confirmada pelo resultado de 70.58% na acurácia top-1.

Na CIFAR-10, devido ao pequeno número de classes, não faz sentido se falar em acurácia top-5. No CIFAR-100, porém, introduzimos essa métrica, que checa se a classe correta está entre as 5 principais predições do classificador. No caso da SqueezeNet-C100, essa taxa é de 91.91%. Os resultados detalhados da arquitetura SqueezeNet são apresentados na tabela 1.

Dado o seu poder de paralelismo, é esperado que a utilização de uma GPU durante a fase de teste acelere o tempo de inferência de uma rede neural. No caso da SqueezeNet-C10, a utilização da GPU acelerou o tempo de inferência em aproximadamente duas vezes.

Notamos que a métrica mais impactada foi a acurácia, apresentando uma diferença de 18,81%, com os valores das outras métricas permanecendo bastante próximos entre as duas bases. Da CIFAR-10 para a CIFAR-100 a SqueezeNet aumentou o número de parâmetros em 6,2%, o tamanho em 3,7% e o tempo de inferência em 1,9% na GPU e 3% na CPU.

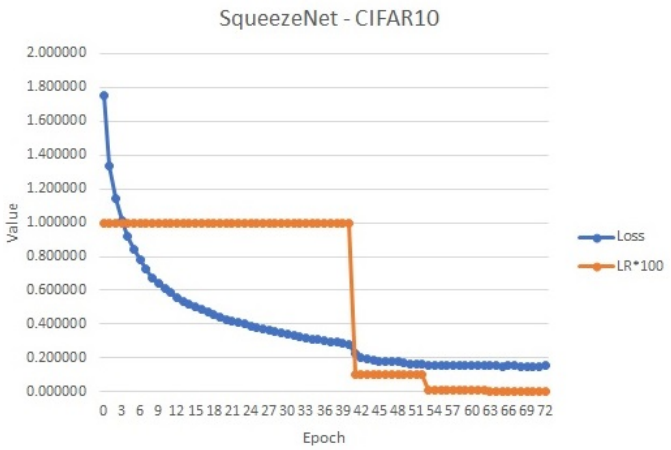
Tabela 1 – Resultados para a SqueezeNet.

SqueezeNet						
Base	Acurácia		Parâmetros	Tamanho	Tempo	
	Top-1	Top-5			GPU	CPU
CIFAR-10	89.39%	-	746,638	2.99 MB	2.64ms ± 0.10	5.35ms ± 0.31
CIFAR-100	70.58%	91.91%	792,988	3.10 MB	2.69ms ± 0.09	5.51ms ± 0.20

Fonte: A autora (2020)

O gráfico apresentado na figura 30 tem como objetivo facilitar a visualização do método utilizado para o ajuste da taxa de aprendizagem, o platô da função de perda. Podemos verificar que, sempre que a função de perda se torna estável, há uma queda na *lr*. Esse processo se repete por quatro vezes e, quando há uma nova estabilização, o treinamento é interrompido. O impacto da queda da *lr* é mais claro nas primeiras iterações do processo e se torna mais sutil nas últimas.

Figura 30 – Gráfico que relaciona *lr* e *loss* durantes as épocas na SqueezeNet.



Fonte: A autora (2020)

5.2 MODELO MOBILENET

A adaptação da MobileNet para as bases CIFAR atingiu 90.42% de acurácia top-1 no CIFAR-10. Já no CIFAR-100, o resultado foi de 69.94% na taxa de acurácia top-1 e 89.85% na taxa top-5. Os resultados detalhados de todos os parâmetros avaliados durante os experimentos podem ser vistos na tabela 2. A taxa de acurácia nos dois casos foi mais elevada do que aquela atingida pela implementação original na ImageNet, de 70.6%. Como já mencionado anteriormente, esse resultado já era esperado, dada a diferença da complexidade das tarefas de classificação entre as duas bases de dados.

Algumas das características apresentadas pela SqueezeNet são igualmente observadas na MobileNet, tais como a melhor performance da rede no desafio CIFAR-10 do que no CIFAR-100 e uma maior acurácia top-5 do que top-1. Podemos observar também que a maior diferença entre as métricas foi novamente na taxa de acurácia, que variou 22,6% entre o top-1 dos datasets, enquanto o número de parâmetros variou 2,9%, o tamanho, 0,5%, e o tempo de inferência, 1,1% em GPU e 0,8% em CPU.

Um dado interessante a ser notado no modelo MobileNet é que o tempo de inferência dessa rede na GPU é maior do que na CPU. Esse fato será discutido com mais detalhes na seção 5.5, na qual será realizada uma análise do impacto do uso de DSCs no tempo de treinamento e teste de uma CNN.

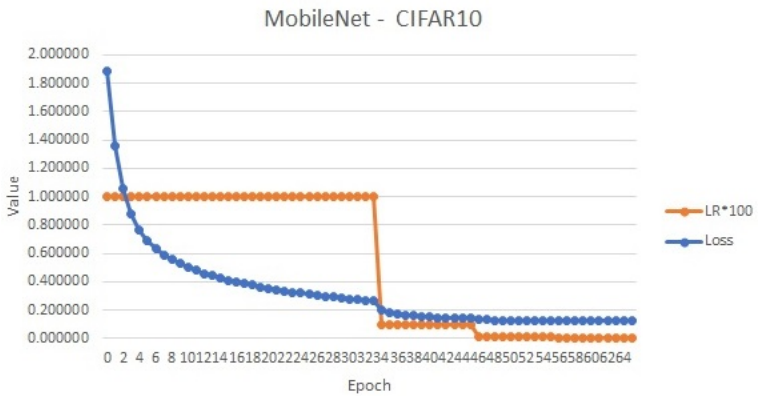
Tabela 2 – Resultados para a MobileNet.

MobileNet						
Base	Acurácia		Parâmetros	Tamanho	Tempo	
	Top-1	Top-5			GPU	CPU
CIFAR-10	90.42%	-	3,217,226	12.69 MB	36.47ms ± 0.95	26.53ms ± 1.06
CIFAR-100	69.94%	89.85%	3,309,476	12.75 MB	36.88ms ± 0.75	26.32ms ± 0.60

Fonte: A autora (2020)

Assim como na seção referente à SqueezeNet, apresentamos na figura 31 um gráfico onde pode ser visto o ajuste da taxa de aprendizagem a partir do platô na função de perda .

Figura 31 – Gráfico que relaciona *lr* e *loss* durantes as épocas na MobileNet.



Fonte: A autora (2020)

5.3 MODELO ALEXNET

A AlexNet, que atingiu uma taxa de acurácia top-1 de 63,3% na base de dados Imagenet, teve esse valor aumentado para 89,17% na CIFAR-10, para 72,97% no top-1 da

CIFAR-100 e para 91,16% no top-5 da CIFAR-100. Todas as métricas de desempenho avaliadas neste trabalho podem ser encontradas na tabela 3.

Notamos que algumas das mesmas características apresentadas pelas outras redes são mantidas pela AlexNet, como uma maior taxa de acurácia no top-5 em relação ao top-1 e o maior impacto de mudança entre os *datasets* sendo observado na acurácia, que varia 16,2%, enquanto o número de parâmetros apresenta uma variação de 1%, o tamanho de armazenamento, de 1,3%, e o tempo de inferência, de 6,6% em GPU e 1,1% em CPU.

As duas redes vistas até aqui empregavam alguma estratégia de otimização de arquitetura, tendo sido idealizadas para ambientes com poder computacional limitado e, consequentemente, sendo consideradas redes pequenas. A AlexNet não emprega estratégias desse tipo, tendo sido idealizada com o objetivo de obter um ganho de acurácia no dataset ImageNet em relação ao estado da arte da época. Ela é considerada uma rede grande, cuja implementação para a base de dados CIFAR conta com cerca de 35 milhões de parâmetros e ocupa 140MB.

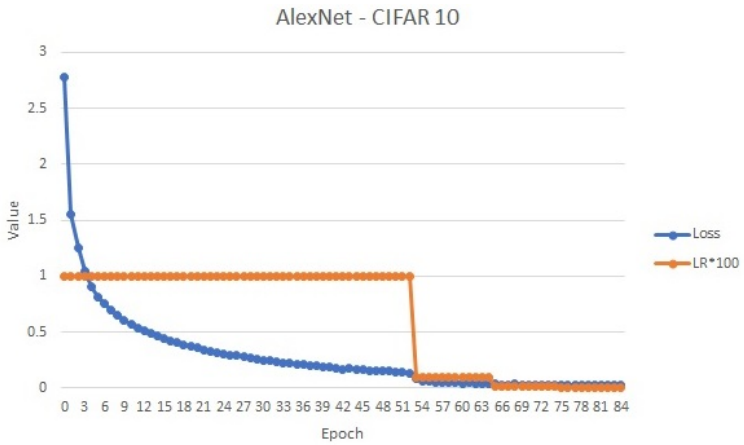
Tabela 3 – Resultados para a AlexNet.

AlexNet						
Base	Acurácia		Parâmetros	Tamanho	Tempo	
	Top-1	Top-5			GPU	CPU
CIFAR-10	89.17%	-	35,871,562	140.21 MB	0.76ms ± 0.08	15.39ms ± 0.33
CIFAR-100	72.97%	92.16%	36,240,292	138.33 MB	0.81ms ± 0.09	15.56ms ± 0.15

Fonte: A autora (2020)

Trazemos novamente o gráfico da queda da taxa de aprendizagem a partir do platô da função de perda, que pode ser observado na imagem 32.

Figura 32 – Gráfico que relaciona *lr* e *loss* durantes as épocas na AlexNet



Fonte: A autora (2020)

5.4 MODELO VGG-19

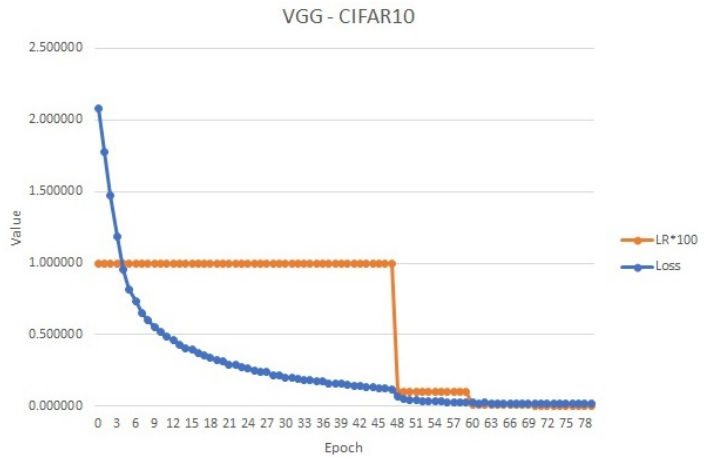
Com uma acurácia de 74,5% na ImageNet, a VGG19 adaptada para as bases CIFAR atingiu uma acurácia de 91,13% na CIFAR-10, 73,74% no top-1 da CIFAR-100 e 91,09% no top-5 da CIFAR-100. A rede apresenta ainda cerca de 20 milhões de parâmetros, ocupa em média 77,5MB e leva 1.66ms para realizar inferências em GPU e 16.5ms para realizar inferências em CPU. Todos esses dados podem ser vistos na tabela 4. Dentre as métricas analisadas, o maior impacto continua sendo na acurácia, que varia 17,39% entre as bases CIFAR, enquanto o número de parâmetros, o tamanho de armazenamento e o tempo de inferência variam 0,2%, 2,1% e 0,2% respectivamente. Foi trazido novamente o gráfico da queda da taxa da acurácia em relação ao platô da função de perda, que pode ser visto na figura 33.

Tabela 4 – Resultados para a VGG19.

VGG19						
Base	Acurácia		Parâmetros	Tamanho	Tempo	
	Top-1	Top-5			GPU	CPU
CIFAR-10	91.13%	-	20,040,522	78.36 MB	1.66ms ± 0.09	16.54ms ± 0.56
CIFAR-100	73.74%	91.09%	20,086,692	76.71 MB	1.66ms ± 0.08	16.50ms ± 0.21

Fonte: A autora (2020)

Figura 33 – Gráfico que relaciona *lr* e *loss* durantes as épocas na VGG



Fonte: A autora (2020)

5.5 MODELO SQUEEZENET-DSC

A tabela 5 apresenta os resultados da SqueezeNet-DSC, arquitetura proposta neste trabalho, que modifica os *Fire Modules* da SqueezeNet, substituindo as suas convoluções tradicionais por convoluções separáveis em profundidade, as DSCs.

Na base de dados CIFAR-10, a SqueezeNet-DSC atingiu uma acurácia de 88.32%, contando com 248,670 parâmetros e ocupando 1.08MB. Já na base de dados CIFAR-100, a acurácia obtida foi de 64.87% no top-1 e 88.86% no top-5, para uma arquitetura com 295,020 parâmetros, que ocupou 1.21MB. Assim como na SqueezeNet original, a diferença de cerca de 46 mil parâmetros entre os *datasets* vem da última camada de convolução tradicional, que possui 10 vezes mais parâmetros na SqueezeNet-DSC-100.

Em relação ao tempo de inferência, relativamente constante entre os *datasets*, vemos que os tempos em GPU e em CPU são parecidos, o que significa que a arquitetura não consegue tirar muito proveito do poder de paralelização das GPUs. Esse fato será discutido com mais detalhes na seção 5.6, na qual faremos uma análise mais profunda do impacto da utilização das DSCs na arquitetura da rede.

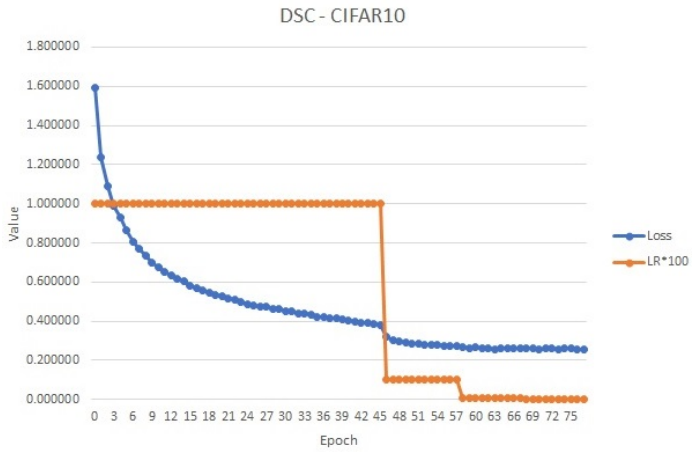
Tabela 5 – Resultados para a SqueezeNet-DSC.

SqueezeNet-DSC						
Base	Acurácia		Parâmetros	Tamanho	Tempo	
	Top-1	Top-5			GPU	CPU
CIFAR-10	88.32%	-	248,670	1.08 MB	5.25ms ± 0.10	6.06ms ± 0.36
CIFAR-100	64.87%	88.85%	295,020	1.21 MB	5.32ms ± 0.10	6.07ms ± 0.16

Fonte: A autora (2020)

O gráfico da figura 34 mostra a redução feita na taxa de aprendizagem durante o treinamento da rede, uma vez que um platô da função de perda fosse atingido.

Figura 34 – Gráfico que relaciona *lr* e *loss* durantes as épocas na SqueezeNet-DSC



Fonte: A autora (2020)

5.6 COMPARAÇÃO ENTRE ARQUITETURAS AVALIADAS

Dado que o objetivo principal deste trabalho é avaliar a SqueezeNet-DSC, optamos por apresentar os resultados obtidos a partir do seu treinamento em conjunto com os

resultados das outras redes, previamente apresentados. O objetivo dessa escolha foi facilitar a comparação da SqueezeNet-DSC com os outros modelos implementados. Para uma melhor visualização, as métricas avaliadas para cada dataset foram divididas em duas tabelas, uma delas contendo os dados de acurácia, número de parâmetros e tamanho de armazenamento, e outra contendo os dados de tempo de inferência em GPU e em CPU.

Através da tabela 6, podemos notar que a substituição das convoluções tradicionais dos blocos *Fire Module* por convoluções separáveis em profundidade surtiu o efeito desejado de reduzir o número de parâmetros da SqueezeNet. Na CIFAR-10, a nova proposta, a SqueezeNet-DSC apresenta apenas 33,3% do número de parâmetros da SqueezeNet e 36,1% do seu tamanho de armazenamento, sendo a menor de todas as redes implementadas. Essa melhoria, porém, não vem sem algum custo, já que podemos observar uma redução de 1,07% na taxa de acurácia.

O melhor resultado em relação aos acertos do classificador foi apresentado pela VGG19, 91,13%, 2,81% melhor do que a SqueezeNet-DSC. A VGG, porém, possui cerca de 20 milhões de parâmetros, contra os 248,670 parâmetros na nova rede proposta. É importante ressaltar que, para algumas aplicações, é aceitável abrir mão de um pouco de acurácia em prol de trabalhar com uma rede menor. Existem inclusive métodos que podem ser utilizados para contornar o problema da perda de acurácia, como, por exemplo, utilizar mais de um classificador, cada um contribuindo com um voto em uma combinação de modelos, por exemplo. Utilizando esse método, seria possível implementar aproximadamente 70 redes SqueezeNet-DSC, ocupando o mesmo espaço de uma rede VGG.

Tabela 6 – Tabela com o comparativo de resultados pelas redes avaliadas no CIFAR-10

SqueezeNet-DSC CIFAR-10 Resultados			
Modelo	Acurácia	Parâmetros	Tamanho
SqueezeNet-DSC	88.32%	**248,670**	**1.08 MB**
SqueezeNet	89.39%	746,638	2.99 MB
MobileNet	90.42%	3,217,226	12.69 MB
AlexNet	89.17%	35,871,562	140.21 MB
VGG19	**91.13%**	20,040,522	78.36 MB

Fonte: A autora (2020)

Em relação aos tempos de inferência de um exemplo de teste, pode ser observado um padrão interessante nos nossos experimentos. Nas redes AlexNet e VGG19, que possuem apenas convoluções tradicionais, se verifica que o uso da GPU consegue acelerar bastante o tempo de inferência, oferecendo uma diminuição de 95% e 90% nesses tempos de processamento, respectivamente. A SqueezeNet, que começa a inserir o conceito de convoluções pontuais, já não apresenta uma diferença tão grande entre o tempo de GPU

e o de CPU, mostrando uma redução de 50.6%. Isso acontece porque as convoluções 1x1 não são paralelizáveis, limitando o potencial de aceleração das GPUs.

A SqueezeNet-DSC, ao substituir as convoluções tradicionais do *Fire Module* por convoluções separáveis em profundidade, limita ainda mais o potencial das GPUs, mostrando uma redução de 13.4% no desempenho. Isso acontece porque, como também reportado em (QIN et al., 2018), a DSC possui uma baixa intensidade aritmética (proporção de cálculos por acessos à memória), introduzindo assim um gargalo no fluxo de dados. Isso significa que o acesso à memória leva mais tempo do que um cálculo, tornando esse tipo de convolução mais difícil de implementar de forma eficiente do que uma convolução padrão.

Em resumo, as camadas DSC apresentam uma menor quantidade de parâmetros em relação às camadas de convolução tradicionais, porém, se tornam mais lentas por não conseguirem utilizar completamente a capacidade das GPUs. Esse fato pode ser observado ainda mais claramente na MobileNet que, por ter a sua arquitetura completamente baseada no uso de DSCs, apresenta um aumento no tempo de inferência na GPU.

Tabela 7 – Tabela com os tempos inferência em GPU e CPU para o CIFAR-10

SqueezeNet-DSC CIFAR-10 Resultados de tempo		
Modelo	GPU	CPU
SqueezeNet-DSC	5.25ms ± 0.10	6.06ms ± 0.36
SqueezeNet	2.64ms ± 0.10	**5.35ms ± 0.31**
MobileNet	36.47ms ± 0.95	26.53ms ± 1.06
AlexNet	**0.76ms ± 0.08**	15.39ms ± 0.33
VGG19	1.66ms ± 0.09	16.54ms ± 0.56

Fonte: A autora (2020)

Na CIFAR-100, era esperado que as redes em geral apresentassem uma queda na performance. Nesta base de dados, a SqueezeNet-DSC, quando comparada à SqueezeNet, apresentou uma queda na taxa de acurácia de 5,71% no top-1 e 3,06% no top-5. Entretanto, apesar da diminuição mais significativa na acurácia, a rede proposta apresentou uma melhoria de 62,8% no número de parâmetros e de 61% no espaço de armazenamento.

Nesta base de dados a maior taxa de acurácia foi novamente apresentada pela VGG, com 73,74% no top-1 e o maior número de parâmetros, e o maior tamanho de armazenamento foram do modelo AlexNet. A tabela 8 contém um resumo dos resultados obtidos por todas as arquiteturas implementadas.

Em todas as arquiteturas, podemos ver um pequeno aumento no número de parâmetros nas implementações realizadas na base de dados CIFAR-100, que se dá pelo maior número de classes desse *dataset*. A última camada da arquitetura de cada rede introduz um número de parâmetros proporcional ao número de canais de saída, que no caso da tarefa de

classificação, é justamente o número de classes do problema. Isso faz com que a última camada das implementações para o CIFAR-100 introduza 10 vezes mais parâmetros do que a última camada das implementações para o CIFAR-10.

Tabela 8 – Tabela com o comparativo de resultados pelas redes avaliadas no CIFAR-100

SqueezeNet-DSC CIFAR-100 Resultados				
Modelo	Acurácia		Parâmetros	Tamanho
	Top-1	Top-5		
SqueezeNet-DSC	64.87%	88.85%	**295,020**	**1.21 MB**
SqueezeNet	70.58%	91.91%	792,988	3.10 MB
MobileNet	69.94%	89.85%	3,309,476	12.75 MB
AlexNet	72.97%	**92.16%**	36,240,292	138.33 MB
VGG19	**73.74%**	91.09%	20,086,692	76.71 MB

Fonte: A autora (2020)

A diferença dos tempos de inferência entre a CIFAR-10 e a CIFAR-100 é bem pequena e a mesma dinâmica se mantém em relação às redes apresentadas.

Tabela 9 – Tabela com os tempos inferência em GPU e CPU para o CIFAR-100

SqueezeNet-DSC CIFAR-100 Resultados de tempo		
Modelo	GPU	CPU
SqueezeNet-DSC	5.32ms ± 0.10	6.07ms ± 0.16
SqueezeNet	2.64ms ± 0.09	**5.51ms ± 0.20**
MobileNet	36.88ms ± 0.75	26.32ms ± 0.60
AlexNet	**0.81ms ± 0.09**	15.56ms ± 0.15
VGG19	1.66ms ± 0.08	16.50ms ± 0.21

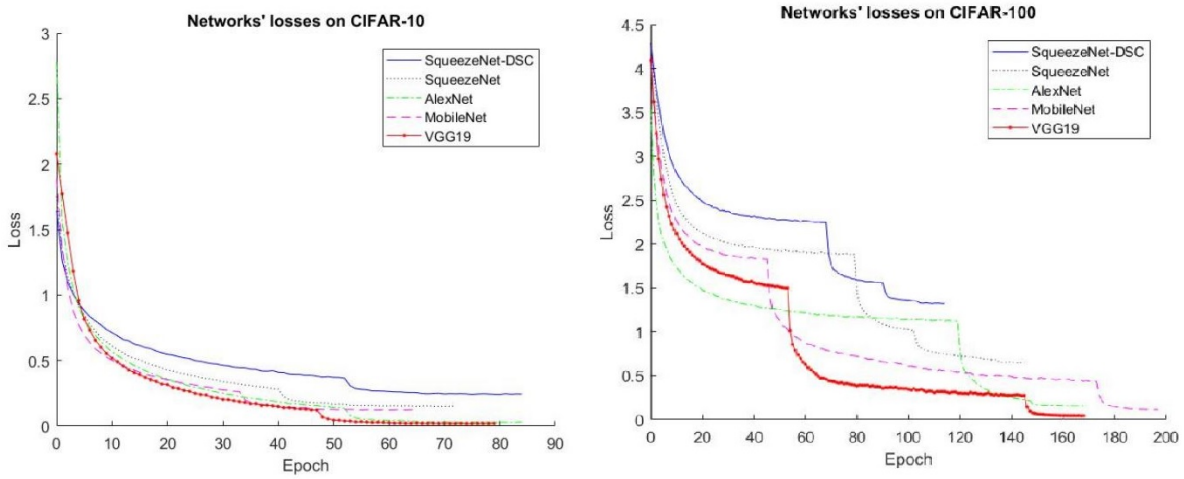
Fonte: A autora (2020)

Os gráficos da figura 35 representam as funções de perda de todas as redes testadas na base de dados CIFAR-10 e na base CIFAR-100, respectivamente. Pode ser observado que, nos dois casos, a SqueezeNet-DSC apresenta a função de perda mais alta em relação às demais redes. Visto que a SqueezeNet-DSC tem menos parâmetros do que os outros modelos, é natural esperar que ela tenha uma alta taxa de perda. Entretanto, isso também significa que ela é menos inclinada a apresentar *overfitting*.

Alguns testes foram realizados com o intuito de obter uma melhor performance da SqueezeNet-DSC. Um deles foi, por exemplo, diminuir o critério a partir do qual era considerado que a função de perda estava em um platô, fazendo com que a queda da learning

rate demorasse mais a acontecer e, consequentemente, a rede fosse treinada por mais épocas. Diferentes taxas de aprendizagem iniciais também foram testadas. Esses experimentos, no entanto, não levaram a melhorias na acurácia da rede e, por não terem sido experimentos sistemáticos, foram deixados de fora deste trabalho. Apesar disso, acreditamos que a rede proposta apresenta espaço para melhoria, podendo se beneficiar de um maior refinamento nas configurações de treinamento, que pode ser feito em um trabalho futuro.

Figura 35 – Perdas das redes testadas com CIFAR-10 e CIFAR-100



Fonte: A autora (2020)

6 CONCLUSÕES

As redes neurais convolucionais vêm sendo aplicadas ao problema de classificação de imagens há muitos anos, em alguns casos apresentando taxas de acerto maiores até do que os próprios seres humanos. As taxas de acurácia sempre crescentes, no entanto, não se traduzem necessariamente em benefícios para outras métricas, como, por exemplo, tamanho da rede ou velocidade de inferência do modelo. Para muitas aplicações do mundo real, porém, principalmente para aquelas em que o poder computacional disponível é limitado, essas métricas são extremamente importantes e podem inclusive se tornar um impeditivo para que as CNNs sejam empregadas.

Um exemplo de aplicação que vem ganhando força nos últimos anos são os aplicativos para *smartphones*. Segundo (XU et al., 2019), entre os meses de junho e setembro de 2018, o número de aplicativos Android que utilizam técnicas de *Deep Learning* cresceu em 27.1%. O ambiente mobile, porém, é bastante restrito não só em termos de poder computacional, como também em relação ao uso da bateria e da capacidade de memória do hardware. Ainda segundo (XU et al., 2019), o tamanho dos modelos de DL utilizados nessas aplicações é bastante pequeno, uma média de 2.5MB, quando comparados a modelos clássicos como VGG-16 (cerca de 500MB) e MobileNet (cerca de 16MB).

Por esses motivos, trabalhos que visam reduzir o número de parâmetros e o espaço de armazenamento das CNNs vêm ganhando cada vez mais relevância. Entre esses trabalhos, destacamos aqueles que investigam formas de alcançar essa redução alterando a forma como a própria convolução é realizada, como a MobileNet, que utiliza o conceito de convoluções separáveis em profundidade (DSCs), para criar uma nova arquitetura de CNN.

Neste trabalho, nós investigamos o efeito do uso das DSC na rede neural convolucional SqueezeNet, cuja proposta é ser uma rede pequena e eficiente, porém mantendo uma acurácia competitiva. A SqueezeNet apresenta novos blocos básicos para a construção de CNNs, os *Fire Modules*. Em uma tentativa de reduzir o número de parâmetros da SqueezeNet, foi proposta a troca das convoluções tradicionais utilizadas nos *Fire Modules* por convoluções separáveis em profundidade. A rede resultante foi chamada de SqueezeNet-DSC e apresentou resultados interessantes. Na bases de dados CIFAR-10, a rede apresentou uma perda de acurácia de 1,07% quando comparada à implementação original da SqueezeNet, porém ocupando apenas 36,1% do seu espaço de armazenamento. Na base CIFAR-100, a perda de acurácia em relação a implementação original foi de 5,71%, com a rede proposta ocupando 39% do espaço do modelo original. A performance da SqueezeNet-DSC também foi comparada a outros modelos populares na literatura, como a VGG19 e a AlexNet obtendo também resultados promissores.

Esses resultados foram publicados em um artigo intitulado *"Reducing SqueezeNet Sto-*

rage Size with Depthwise Separable Convolution" na *International Joint Conference on Neural Networks* (IJCNN) de 2018, que aconteceu no Rio de Janeiro e o código-fonte do modelo proposto está disponibilizado no repósitorio do GitHub[1].

Em trabalhos futuros, planejamos realizar um ajuste mais preciso nos hiperparâmetros da SqueezeNet-DSC, com o objetivo de conseguir melhores resultados na taxa de acurácia da rede. Um dos ajustes idealizados é a utilização de outras taxas de aprendizagem iniciais, além de diferentes formas de controle da taxa de aprendizagem durante o treinamento, além de treinar a rede por mais épocas com o objetivo de diminuir o erro obtido na função de perda. Planejamos também treinar a nova rede proposta em bases de dados mais desafiadoras, como a ImageNet, avaliando as mesmas métricas utilizadas durante este trabalho.

[1] *https://github.com/camilaodsouza/DL2017.2*

REFERÊNCIAS

ADELI, H.; PANAKKAT, A. A probabilistic neural network for earthquake magnitude prediction. *Neural networks*, Elsevier, v. 22, n. 7, p. 1018–1024, 2009.

CAMPBELL, M.; JR, A. J. H.; HSU, F.-h. Deep blue. *Artificial intelligence*, Elsevier, v. 134, n. 1-2, p. 57–83, 2002.

CAUCHY, A. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, v. 25, n. 1847, p. 536–538, 1847.

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.

CHOUKROUN, Y.; KRAVCHIK, E.; KISILEV, P. Low-bit quantization of neural networks for efficient inference. *arXiv preprint arXiv:1902.06822*, 2019.

CUN, Y. L.; FOGELMAN-SOULIÉ, F. Modèles connexionnistes de l'apprentissage. *Intellectica*, Persée-Portail des revues scientifiques en SHS, v. 2, n. 1, p. 114–143, 1987.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, Springer, v. 2, n. 4, p. 303–314, 1989.

DAHAB, D. A.; GHONIEMY, S. S.; SELIM, G. M. et al. Automated brain tumor detection and identification using image processing and probabilistic neural network techniques. *International journal of image processing and visual communication*, v. 1, n. 2, p. 1–8, 2012.

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.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, v. 12, n. Jul, p. 2121–2159, 2011.

FREEMAN, I.; ROESE-KOERNER, L.; KUMMERT, A. Effnet: An efficient structure for convolutional neural networks. In: IEEE. *2018 25th IEEE International Conference on Image Processing (ICIP)*. [S.l.], 2018. p. 6–10.

GONDIM, A. Análise de métodos de otimização de parâmetros e tempo de inferência para modelos de aprendizagem profunda. *Tese de Mestrado, Universidade Federal de Pernambuco*, 2019.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

GOODMAN, J. Classes for fast maximum entropy training. In: IEEE. *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*. [S.l.], 2001. v. 1, p. 561–564.

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.; ZHANG, X.; REN, S.; SUN, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 37, n. 9, p. 1904–1916, 2015.

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.

HEGDE, S.; PRASAD, R.; HEBBALAGUPPE, R.; KUMAR, V. Variational student: Learning compact and sparser networks in knowledge distillation framework. *arXiv preprint arXiv:1910.12061*, 2019.

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

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, MIT Press, v. 18, n. 7, p. 1527–1554, 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.

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.

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

JARRETT, K.; KAVUKCUOGLU, K.; RANZATO, M.; LECUN, Y. What is the best multi-stage architecture for object recognition? In: IEEE. *2009 IEEE 12th international conference on computer vision*. [S.l.], 2009. p. 2146–2153.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

KRIZHEVSKY, A.; NAIR, V.; HINTON, G. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, v. 6, 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.

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.

LINNAINMAA, S. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, p. 6–7, 1970.

LIU, Y.; WANG, Y.; WANG, S.; LIANG, T.; ZHAO, Q.; TANG, Z.; LING, H. Cbnet: A novel composite backbone network architecture for object detection. *arXiv preprint arXiv:1909.03625*, 2019.

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In: *Proc. icml*. [S.l.: s.n.], 2013. v. 30, n. 1, p. 3.

MACCULLOCH, W.; PITTS, W. A logical calculus of ideas immanent in nervous activity. bull. mathematical biophysics. vol. 5. 1943.

MACEDO, D. Enhancing deep learning performance using displaced rectifier linear unit. *Master's Thesis , Universidade Federal de Pernambuco*, 2017.

MIKOLOV, T.; SUTSKEVER, I.; CHEN, K.; CORRADO, G. S.; DEAN, J. Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2013. p. 3111–3119.

MINSKY, M.; PAPERT, S. An introduction to computational geometry. *Cambridge tiass., HIT*, 1969.

MIRZADEH, S.-I.; FARAJTABAR, M.; LI, A.; GHASEMZADEH, H. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *arXiv preprint arXiv:1902.03393*, 2019.

MOR-YOSEF, S.; SAMUELOFF, A.; MODAN, B.; NAVOT, D.; SCHENKER, J. G. Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study. *Obstetrics and gynecology*, v. 75, n. 6, p. 944–947, 1990.

MÜNKER, C. Using convolutional neural networks to distinguish vehicle pose and vehicle class. *Master's Thesis , Technische Universität Darmstadt*, 2016.

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.

NAYAK, P.; ZHANG, D.; CHAI, S. Bit efficient quantization for deep neural networks. *arXiv preprint arXiv:1910.04877*, 2019.

NIELSEN, M. A. *Neural networks and deep learning*. [S.l.]: Determination press San Francisco, CA, USA:, 2015. v. 2018.

POLYAK, B. T. Gradient methods for solving equations and inequalities. *USSR Computational Mathematics and Mathematical Physics*, Elsevier, v. 4, n. 6, p. 17–32, 1964.

QIN, Z.; ZHANG, Z.; LI, D.; ZHANG, Y.; PENG, Y. Diagonalwise refactorization: An efficient training method for depthwise convolutions. In: IEEE. *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2018. p. 1–8.

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.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

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

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.

SELVIN, S.; VINAYAKUMAR, R.; GOPALAKRISHNAN, E.; MENON, V. K.; SOMAN, K. Stock price prediction using lstm, rnn and cnn-sliding window model. In: IEEE. *2017 international conference on advances in computing, communications and informatics (icacci)*. [S.l.], 2017. p. 1643–1647.

SHORTEN, C.; KHOSHGOFTAAR, T. M. A survey on image data augmentation for deep learning. *Journal of Big Data*, Springer, v. 6, n. 1, p. 60, 2019.

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

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

SOLOMON, C.; BRECKON, T. *Fundamentos de processamento digital de imagens: uma abordagem prática com exemplos em Matlab*. [S.l.]: Grupo Gen-LTC, 2000.

SWE, T.; TIN, P. Recognition and translation of the myanmar printed text based on hopfield neural network. In: IEEE. *6th Asia-Pacific Symposium on Information and Telecommunication Technologies*. [S.l.], 2005. p. 99–104.

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.

TAN, M.; LE, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

TIELEMAN, T.; HINTON, G. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 2012.

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.

TOUVRON, H.; VEDALDI, A.; DOUZE, M.; JÉGOU, H. Fixing the train-test resolution discrepancy. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2019. p. 8250–8260.

XIE, Q.; HOVY, E.; LUONG, M.-T.; LE, Q. V. Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252*, 2019.

XU, B.; WANG, N.; CHEN, T.; LI, M. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

XU, M.; LIU, J.; LIU, Y.; LIN, F. X.; LIU, Y.; LIU, X. A first look at deep learning apps on smartphones. In: *The World Wide Web Conference*. [S.l.: s.n.], 2019. p. 2125–2136.

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.