Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação
Mestrado Acadêmico em Sistemas e Computação

FiberNet: Um Modelo de Rede Neural Convolucional Simples e Eficiente

Por: Verner Rafael Ferreira

Natal – RN

Agosto de 2024

Verner Rafael Ferreira

FiberNet: Um Modelo de Rede Neural Convolucional Simples e Eficiente.

Tese de doutorado apresentada ao Programa de Pós Graduação de Sistemas e Computação do Departamento de Informática e Matemática aplicada da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Doutor em Sistemas e Computação.

Linha de Pesquisa: Inteligência Artificial Aplicada

Orientadora:
Profa. Dr. Anne Magaly de Paula Canuto

PPgSC – Programa de Pós-Graduação em Sistemas e Computação
DIMAp – Departamento de Informática e Matemática Aplicada
CCET – Centro de Ciências Exatas e da Terra
UFRN – Universidade Federal do Rio Grande do Norte

Natal – RN
Agosto / 2024

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET

Ferreira, Verner Rafael.
FiberNet: um modelo de rede neural convolucional simples e eficiente / Verner Rafael Ferreira. - 2024.
125 f.: il.

Tese (doutorado) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Programa de Pós-Graduação em Sistemas e Computação. Natal, RN, 2024.
Orientação: Profa. Dra. Anne Magaly de Paula Canuto.

1. Computação - Tese. 2. Rede neural convolucional - Tese. 3. Aprendizado profundo - Tese. 4. Classificação de imagens - Tese. 5. Inteligência artificial - Tese. 6. Planta Agave Sisalana - Tese. I. Canuto, Anne Magaly de Paula. II. Título.

RN/UF/CCET CDU 004(043.2)

Elaborado por Joseneide Ferreira Dantas - CRB-15/324

VERNER RAFAEL FERREIRA

“ *FiberNet: um modelo de Rede Neural Convolucional simples e eficiente*”

Esta Tese foi julgada adequada para a obtenção do título de Doutor(a) em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

Prof. Dr. NÉLIO ALESSANDRO AZEVEDO CACHO
Coordenador do PPgSC

Banca Examinadora

Documento assinado digitalmente
gov.br
ARAKEN DE MEDEIROS SANTOS
Data: 02/09/2024 14:05:53-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Externo(a): **Dr. ARAKEN DE MEDEIROS SANTOS**

Documento assinado digitalmente
gov.br
DIEGO SILVEIRA COSTA NASCIMENTO
Data: 31/08/2024 19:38:07-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Externo(a): **Dr. DIEGO SILVEIRA COSTA NASCIMENTO**

Documento assinado digitalmente
gov.br
BRUNO MOTTA DE CARVALHO
Data: 12/09/2024 00:15:54-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Interno(a): **Dr. BRUNO MOTTA DE CARVALHO**

Documento assinado digitalmente
gov.br
JOAO CARLOS XAVIER JUNIOR
Data: 12/09/2024 08:54:28-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Externo(a): **Dr. JOÃO CARLOS XAVIER JUNIOR**

Documento assinado digitalmente
gov.br
ANNE MAGALY DE PAULA CANUTO
Data: 12/09/2024 09:20:54-0300
Verifique em https://validar.iti.gov.br

Presidente: **Dr.ª ANNE MAGÁLY DE PAULA CANUTO**

Documento assinado digitalmente
gov.br
VERNER RAFAEL FERREIRA
Data: 12/09/2024 13:59:50-0300
Verifique em https://validar.iti.gov.br

Discente: **VERNER RAFAEL FERREIRA**

Natal, 28 de agosto de 2024

“Aquele que habita no esconderijo do Altíssimo, à sombra do onipotente descansará. Direi do Senhor: Ele é o meu Deus, o meu refúgio, a minha fortaleza, e Nele confiarei.” Salmo 91.

Agradecimento

Agradeço primeiramente a Deus por tudo de que o Senhor tem feito por mim e pela permissão para que eu pudesse concluir mais essa etapa de minha vida acadêmica.

Agradeço a toda a minha família por ter me dado o suporte necessário para que eu conseguisse finalizar meu Doutorado. Em especial ao meu pai que foi o maior exemplo de caráter enquanto esteve neste mundo. Que Deus o acolha na morada Eterna!

Agradeço também ao Estado da Bahia, na figura da Universidade do Estado da Bahia – UNEB, Instituição Pública de Ensino Superior no qual eu trabalho como servidor público, pela liberação para realizar meu doutorado.

“Tenho a impressão de ter sido uma criança brincando à beira-mar, divertindo-me em descobrir uma pedrinha mais lisa ou uma concha mais bonita que as outras, enquanto o imenso oceano da verdade continua misterioso diante de meus olhos”. (Isaac Newton)

FiberNet: Um Modelo de Rede Neural Convolucional Simples e Robusto

Autor: Verner Rafael Ferreira

Orientadora: Profa. Dra. Anne Magaly de Paula Canuto

RESUMO

Redes neurais convolucionais (CNNs) são arquiteturas poderosas e eficazes para extrair informações significativas de imagens e identificar objetos. No entanto, seu alto custo computacional pode limitar sua adoção em cenários com recursos computacionais limitados como, por exemplo, em dispositivos móveis. Para resolver esse problema, nos propomos uma nova arquitetura de CNN no qual incluímos uma nova camada denominada Defiber que atua na fase de convolução da CNN. Essa nova camada, pertencente à estratégia de *down sampling*, tem por finalidade reduzir o quantitativo de parâmetros treináveis da rede sem que isso afete sua capacidade de predição. Para testar nossa abordagem nós criamos a FiberNet. Um protótipo de CNN pequena e simples que possui um número reduzido de parâmetros treináveis. Isso resultou em uma rede com alta velocidade de inferência e custos computacionais reduzidos. A FiberNet foi avaliada em dois conjuntos de dados, Sisal e CIFAR10. No conjunto Sisal, a FiberNet alcançou uma precisão de 96,25%. No conjunto CIFAR10, a FiberNet alcançou uma precisão de 74,9%. Nossos resultados mostraram que a camada Defiber é uma alternativa viável para a construção de CNNs de baixo custo. Sua aplicação na arquitetura do modelo proposto resultou em uma alta acurácia e capacidade de processamento, mesmo com um número reduzido de parâmetros treináveis.

Palavras-chave: Rede Neural Convolucional, classificação de imagens, planta Agave Sisalana.

FiberNet: A Simple and Robust Convolutional Neural Network Model

Author: Verner Rafael Ferreira

Supervisor: Profa. Dra. Anne Magaly de Paula Canuto

ABSTRACT

Convolutional neural networks (CNNs) are powerful and effective tools for extracting meaningful information from images and identifying objects. However, their high computational cost can limit their adoption in scenarios with limited computational resources, such as mobile devices. To address this issue, we propose a new CNN architecture in which we include a new layer called Defiber which operates in the convolution phase of the CNN. This new layer, belonging to the down sampling strategy, aims to reduce the number of trainable parameters of the network without affecting its prediction ability. To test our approach, we created FiberNet. A prototype of a small and simple CNN that has a reduced number of trainable parameters. This resulted in a network with high inference speed and reduced computational costs. FiberNet was evaluated on two datasets, Sisal and CIFAR10. On the Sisal dataset, FiberNet achieved a precision of 96.25%. On the CIFAR10 dataset, FiberNet achieved a precision of 74.9%. Our results show that the Defiber layer is a viable alternative for building low-cost CNNs. Its application in the architecture of our model resulted in high accuracy and processing capacity, even with a reduced number of trainable parameters.

Keywords - Convolutional Neural Network, Image Classification, Agave Sisalana plant.

Lista de Figuras

Figura 1: Representação do cálculo de convolução entre imagem e filtro. Fonte: https://www.researchgate.net/profile/Jose-Padarian 32
Figura 2: Exemplo de aplicação do filtro de convolução. Fonte: https://commons.wikimedia.org/wiki/File:Convolution_arithmetic_-_Padding_*stride*s.gif 37
Figura 3: Ilustração da GoogleNet exemplificando uma arquitetura profunda (Fonte: https://medium.com/the-modern-scientist/exploring-googlenet-a-revolutionary-deep-learning-architecture-8bb176a0facc) 43
Figura 4: Exemplo de aplicação de camada residual em uma CNN (Fonte: https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4) 43
Figura 5: Arquitetura módulo FIRE da SqueezeNet 44
Figura 6: Descrição da arquitetura da DenseNet 44
Figura 7: Descrição do bloco de convolução da MobileNet 45
Figura 8: Funcionamento do *Channel shuffle* 45
Figura 9: Arquitetura do bloco de convolução da ShuffleNet V2. Fonte: https://paperswithcode.com/method/shufflenet-v2-block 46
Figura 10: Descrição dos blocos de convolução da MobileNet V2 47
Figura 11: Descrição da aplicação NetAdapt da arquitetura da MobileNet V3 47
Figura 12: Descrição da escalabilidade composta proposta por Tan e Le na arquitetura da EfficientNet (Fonte: https://paperswithcode.com/method/efficientnet) 48
Figura 13: Fluxograma do modelo proposto (figura do autor) 57
Figura 14: Fluxograma da camada Defiber (figura do autor) 60
Figura 15: Ilustração do processo de redução da imagem ao passar pela camada Defiber (figura do autor) 61
Figura 16: Ilustração do processo de redimensionamento da matriz a partir do código em linguagem Python 62
Figura 17: Sumário do modelo ao processar uma imagem com a resolução de 1080x1080 63
Figura 18: Fluxograma da Função de probabilidade (figura do autor). 65
Figura 19: Imagens retiradas do banco de imagens do SISAL produzido em nossa pesquisa. No exemplo temos imagens da classe Fibra e da classe Sisal (figura do autor) na qual estão inclusas também imagens da palma do sisal. 68
Figura 20: Imagens da base CIFAR10. Fonte: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html 69
Figura 21: Fluxograma do algoritmo da FiberNet (figura do autor) 71
Figura 22: Fluxograma da função CDR. 75
Figura 23: Gráfico da Diferença Crítica para a acurácia entre os modelos profundos 92
Figura 24: Gráfico da Diferença Crítica para a acurácia entre os modelos móveis 92
Figura 25: Gráfico da Diferença Crítica para TI entre os modelos profundos 93
Figura 26: Gráfico da Diferença Crítica para TI entre os modelos móveis 93
Figura 27: Gráfico da Distância Crítica para TT entre os modelos profundos 93
Figura 28: Gráfico da Distância Crítica para TT entre os modelos móveis 94
Figura 29: Gráfico da Distância Crítica para a acurácia entre os modelos profundo 97
Figura 30: Gráfico da Distância Crítica para a Acurácia entre os modelos móveis 97
Figura 31: Gráfico da Distância Crítica para o critério de TT entre os modelos profundos 98
Figura 32: Gráfico da Distância Crítica para o critério de TT entre os modelos móveis 98
Figura 33: Gráfico da Distância crítica para o critério de TI entre os modelos profundos 98

Figura 34: Gráfico da Distância Crítica para o critério de TI entre os modelos móveis 98
Figura 35: Gráfico da Distância Crítica para o critério da acurácia entre as versões da FiberNet 105
Figura 36: Gráfico da Distância Crítica do critério de TT para as versões da FiberNet 106
Figura 37: Gráfico da Distância Crítica do critério de TI para as versões da FiberNet 106
Figura 38: Gráfico da Distância Crítica para a acurácia entre as versões da FiberNet 110
Figura 39: Gráfico da Distância Crítica do critério de TT entre as versões da FiberNet com a base CIFAR10 111
Figura 40: Gráfico da Distância Crítica do critério de TI entre as versões da FiberNet com a base CIFAR10 111

Lista de Tabelas

Tabela 1: Descrição de hiperparâmetros de convolução do modelo proposto (FiberNet) 71
Tabela 2: Pseudocódigo responsável pelo cálculo do número de repetições da camada Defiber 72
Tabela 3: Simulação da Camada Defiber utilizada pela função CDR 74
Tabela 4: Resultado de estudo comparativo da arquitetura da FiberNet utilizando a camada Defiber e outras metodologias de down sample sobre a base CIFAR10 75
Tabela 5: Variações da FiberNet testadas com a base Sisal 77
Tabela 6: Variações da FiberNet testadas com a base CIFAR10 77
Tabela 7: Total de arquiteturas testadas por quantidade de parâmetros e suas respectivas acurácias médias por grupo (bases Sisal e CIFAR10) 87
Tabela 8: Resultado do teste de arquitetura por variação de limite na camada Defiber e seus resultados (parâmetros e acurácias) 88
Tabela 9: Modelos distribuídos por quantidade de parâmetros treináveis e tamanho do arquivo 89
Tabela 10: Resultados de precisão dos modelos profundos e móveis com a base Sisal 90
Tabela 11: Resultados de taxa de transferência (TT) e taxa de inferência (TI) para os modelos profundos com a base Sisal 91
Tabela 12: Resultados de TT e TI para os modelos móveis com a base Sisal 91
Tabela 13: Resultado do teste de Friedman para ACC, TT e TI com a base Sisal 92
Tabela 14: Resultados de Post-hoc da Acurácia para os modelos móveis e profundos com a base Sisal 92
Tabela 15: Resultados do teste de Post-hoc de TT e TI para os modelos profundos com a base Sisal 92
Tabela 16: Resultados do teste de Post-hoc de TT e TI para os modelos móveis para a base Sisal 93
Tabela 17: Resultados da acurácia para os modelos profundos e móveis para a base CIFAR10 94
Tabela 18: Resultados de TT e TI para os modelos profundos com a base CIFAR10 95
Tabela 19: Resultados de TT e TI para os modelos móveis com a base CIFAR10 95
Tabela 20: Resultado do teste de Friedman para acurácia, TT e TI com a base CIFAR10 97
Tabela 21: Resultado do teste Post-hoc para a acurácia dos modelos profundos e móveis com a base CIFAR10 97
Tabela 22: Resultado do teste de Post-hoc para TT e TI dos modelos profundos com a base CIFAR10 97
Tabela 23: Resultado do teste de Post-hoc para TT e TI dos modelos móveis com a base CIFAR10 . 98
Tabela 24: Configuração dos modelos avaliados com a base Sisal (NA=Não se Aplica) por Total de Parâmetros 100
Tabela 25: Configurações dos modelos avaliados com a base CIFAR10 por Total de Parâmetros ... 100
Tabela 26: Resultados da acurácia com a base Sisal 101
Tabela 27: Resultado da taxa de transferência (TT) com a base Sisal 102
Tabela 28: Resultado da taxa de inferência (TI) com a base Sisal 103
Tabela 29: Resultado do teste de Friedman para os critérios de Acurácia, TT e TI da Fase 2 104
Tabela 30: Resultado post-hoc da acurácia na fase 02 com a base Sisal 105
Tabela 31: Resultados post-hoc para TT e TI da fase 02 com a base Sisal 106
Tabela 32: Resultado da acurácia para as versões da FiberNet para os dados da base CIFAR10 108
Tabela 33: Resultados para o critério de TT na Fase 2 com os dados da CIFAR10 108
Tabela 34: Resultado para o critério de TI da Fase 2 com a base CIFAR10 109
Tabela 35: Resultado do teste de Friedman para os dados da Fase 2 com a base CIFAR10 109
Tabela 36: Resultado post-hoc para o critério da acurácia na Fase 2 com a base CIFAR10 110

Tabela 37: Resultado post-hoc para os critérios de TT e TI dos resultados da Fase 2 com a base CIFAR10 111
Tabela 38: Posição geral da FiberNet nos resultados dos experimentos realizados 113

Lista de Siglas e Abreviaturas

ACC	Acurácia
AM	Aprendizado de Máquina
AVG	Average
CNN	Convolutional Neural Network
Conv	Convolução
DL	Deep Learning
DP	Desvio padrão
ELU	Exponential Linear Unit
GAN	Generative Adversarial Network
GPU	Graphic Processing Unit
IA	Inteligência Artificial
NAS	Neural Architecture Search
PLN	Processamento de Linguagem Natural
ReLU	Rectified Linear Unit
RNA	Rede Neural Artificial
TI	Taxa de Inferência
TPU	Tensor Processing Unit
TT	Taxa de Transferência
VC	Visão Computacional
VGG	Very Deep Convolutional Network

Sumário

CAPÍTULO 1. INTRODUÇÃO 18
1.1. MOTIVAÇÃO 20
1.2. PROBLEMAS DE PESQUISA E HIPÓTESES 22
1.3. OBJETIVOS 23
1.4. PRINCIPAIS CONTRIBUIÇÕES 24
1.5. ORGANIZAÇÃO DO TRABALHO 26
CAPÍTULO 2. CONCEITOS RELACIONADOS 28
2.1. APRENDIZADO DE MÁQUINA 28
2.1.1. Tipos de Aprendizado de Máquina 28
2.2. REDES NEURAIS ARTIFICIAIS (RNA) 29
2.3. APRENDIZADO PROFUNDO (DO INGLÊS DEEP LEARNING) 30
2.3.1. Convolutional Neural Network (Cnn) 31
2.3.1.1. O processo de convolução 31
2.3.1.2. Hiperparâmetros 33
2.3.1.3. Arquitetura 38
2.3.1.4. Upsampling e Downsampling 39
2.3.1.5. Tipos de CNN 40
2.3.1.5.1. Redes neurais convolucionais profundas (deep) 41
2.3.1.5.2. Redes neurais convolucionais para dispositivos móveis (mobile) 41
2.3.1.5.3. Comparação entre redes neurais convolucionais profundas e móveis 42
2.4. ARQUITETURAS DE CNN 42
2.4.1. GoogleNet (2014) 42
2.4.2. ResNet (2015) 43
2.4.3. SqueezeNet (2016) 44
2.4.4. DenseNet (2016) 44
2.4.5. MobileNet V1 (2017) 45
2.4.6. ShuffleNet V1 (2018) 45
2.4.7. ShuffleNet V2 (2018) 46
2.4.8. Mobilenet V2 (2018) 46
2.4.9. Mobilenet V3 (2019) 47
2.4.10. Efficientnet V1 (2019) 48
2.4.11. Efficientnet V2 (2021) 48

2.5. CONSIDERAÇÕES FINAIS 48
CAPÍTULO 3. TRABALHOS RELACIONADOS 50
3.1. CNNS MOBILE NA CLASSIFICAÇÃO DE FRUTAS E HORTALIÇAS 50
3.2. CNNS MOBILE NA CLASSIFICAÇÃO DE DOENÇAS EM PLANTAS 51
3.3. CNN NA CLASSIFICAÇÃO DE PLANTAS FIBROSAS 52
3.4. PONTOS IMPORTANTES SOBRE OS TRABALHOS RELACIONADOS 53
3.5. CONSIDERAÇÕES FINAIS 54
CAPÍTULO 4. O MODELO PROPOSTO 56
4.1. A ARQUITETURA BASE DO MODELO PROPOSTO 56
4.2. A CAMADA DEFIBER 59
4.3. A FUNÇÃO DE PROBABILIDADE 63
4.4. CONSIDERAÇÕES FINAIS 66
CAPÍTULO 5. METODOLOGIA EXPERIMENTAL 67
5.1. BASES DE DADOS UTILIZADAS 67
5.2. MÉTODOS E MATERIAIS 69
5.3. O DESENVOLVIMENTO DO ALGORITMO FIBERNET 70
5.3.1. O desenvolvimento da função de probabilidade 72
5.3.2. As simulações das camadas de convolução e Defiber 73
5.4. DEFINIÇÃO DO PROCESSO DE ANÁLISE COMPARATIVA ENTRE ALGORITMOS. 76
5.4.1. Fase 01: Comparação da FiberNet com outras arquiteturas 76
5.4.2. Fase 02: Comparação da FiberNet com variações de sua própria arquitetura 76
5.5. HIPERPARÂMETROS DE TREINAMENTO 78
5.5.1. Batch-size 78
5.5.2. Épocas 79
5.5.3. Taxa de aprendizado e otimizador 80
5.5.4. Função de custo e aspectos gerais 81
5.6. MATRIZ DE CONFUSÃO: VERIFICANDO A ACURÁCIA 82
5.7. MÉTODO ESTATÍSTICO DE FRIEDMAN 83
5.8. MÉTODO ESTATÍSTICO DE NEMENYI 84
5.9. CONSIDERAÇÕES FINAIS 85
CAPÍTULO 6. ANÁLISE DOS RESULTADOS 87
6.1. RESULTADOS DA ANÁLISE EMPÍRICA 87
6.2. FASE 1: RESULTADO DOS EXPERIMENTOS PELA COMPARAÇÃO ENTRE ALGORITMOS 88
6.2.1. Comparação da quantidade de parâmetros treináveis dos algoritmos 89

6.2.2. Resultado dos algoritmos, separando-os por categoria (profundos e móveis) e treinados com a base Sisal 90
6.2.3. Resultados dos algoritmos, separando-os por categoria (profundos e móveis) e treinados com a base CIFAR10 94
6.3. FASE 2: COMPARANDO OS RESULTADOS DA FIBERNET COM OUTRAS ESTRATÉGIAS DE DOWN SAMPLING 99
6.3.1. Comparação dos resultados da FiberNet com outras estratégias de *down sampling* analisados com a base Sisal 101
6.3.2. Comparando os resultados da FiberNet com outras estratégias de *down sampling* analisados com a base CIFAR10 107
6.4. ANÁLISE GERAL DOS RESULTADOS 112
6.5. CONSIDERAÇÕES FINAIS 114
CAPÍTULO 7. CONCLUSÃO 116
7.1. TRABALHOS FUTUROS 117
REFERÊNCIAS 119
APÊNDICE I

CAPÍTULO 1.

INTRODUÇÃO

A utilização de tecnologia como ferramenta para amplificar a produtividade humana remonta a períodos anteriores, evidenciando a incessante busca por instrumentos que facilitem a realização de atividades e alcance de metas. Desde a Revolução Industrial, com a introdução dos carros a vapor, até a era moderna, com a concepção da Máquina Diferencial em 1822, a humanidade demonstrou a necessidade de soluções tecnológicas.

O computador, conforme observado por Ponte (1989), emerge como uma ferramenta versátil, sustentando diversas disciplinas. Destaca-se a Inteligência Artificial (IA), um campo da ciência da computação explorado por Gomes (2010), cujo objetivo é reproduzir habilidades humanas como aprendizado, raciocínio, reconhecimento de fala e imagem, tomada de decisão e resolução de problemas.

Utilizando técnicas como Aprendizado de Máquina (AM), Processamento de Linguagem Natural (PLN) e algoritmos de busca, os sistemas de IA são capazes de analisar dados, aprender com exemplos e fazer previsões ou tomar decisões. Dentro do escopo da IA, o Aprendizado de Máquina, conforme delineado por Mitchell (1997), é um domínio que se concentra no desenvolvimento de algoritmos capazes de aprender tarefas específicas com base em dados de entrada, sem a necessidade de programação explícita.

Esses algoritmos, ao invés de serem programados de forma determinística, aprendem com exemplos fornecidos por especialistas ou dados coletados automaticamente, representando um avanço significativo na área da computação e ciência de dados. Esse conceito pode ser mais bem entendido se a dividirmos em três partes principais (Domingos, 2015):

- Experiência: O Aprendizado de Máquina se baseia em exemplos de dados para aprender a tomar decisões ou realizar tarefas. Esses exemplos podem ser fornecidos por um especialista humano ou podem ser coletados automaticamente a partir de dados brutos.
- Tarefas: O Aprendizado de Máquina é aplicado a uma classe de tarefas, que pode ser qualquer coisa, desde reconhecimento de fala e imagem até previsão de preços de ações ou diagnóstico médico.

- Desempenho: O desempenho do algoritmo de Aprendizado de Máquina é medido em termos de uma métrica definida, que pode ser a taxa de erro, a precisão, o tempo de execução ou qualquer outra medida relevante para a tarefa em questão.

O objetivo do Aprendizado de Máquina é criar um modelo ou algoritmo que possa aprender a realizar uma tarefa com base em dados, para que possa generalizar e realizar essa tarefa com precisão em novos dados. Existem muitos algoritmos diferentes de Aprendizado de Máquina, como árvores de decisão, redes neurais, *Support Vector Machines* (SVM) e regressão linear, cada um adequado para diferentes tipos de problemas.

E explorando ainda mais o universo do Aprendizado de Máquina, associado à sua ampla gama de aplicações, chegamos ao conceito de *Deep Learning* ou aprendizado profundo na tradução literal. O aprendizado profundo pode ser definido como um subcampo do Aprendizado de Máquina que se concentra no desenvolvimento de redes neurais profundas, que são modelos computacionais inspirados na estrutura e função do cérebro humano (Goodfellow, et.al., 2016).

Essas redes neurais são treinadas com grandes quantidades de dados para identificar padrões complexos e tentar fazer previsões precisas. No aprendizado profundo, os dados passam por várias camadas de neurônios, onde cada camada aprende características mais abstratas. Redes profundas podem lidar com problemas mais complexos devido à sua capacidade de aprendizado em várias camadas, enquanto redes rasas, com menos camadas, são adequadas para problemas mais simples e de menor escala.

Em termos de execução, a eficácia de uma abordagem de aprendizado profundo repousa sobre três pilares fundamentais: a seleção criteriosa do modelo apropriado, a disponibilidade de volumes substanciais de dados e a capacidade computacional necessária para o treinamento do modelo (Ng, 2017).

A conjunção harmoniosa entre um modelo bem concebido, uma profusão de dados e recursos computacionais suficientes é crucial para garantir o êxito das aplicações de aprendizado profundo em uma miríade de domínios, como Visão Computacional (VC), Processamento de Linguagem Natural (PLN), reconhecimento de voz, entre outros contextos desafiadores.

O emprego de um modelo adequado desempenha um papel decisivo na garantia da adequação da arquitetura da rede neural à tarefa específica. Por exemplo, na Visão Computacional, as redes neurais convolucionais (CNNs) ganham destaque devido à sua habilidade de detectar características visuais em variadas escalas e orientações, tornando-as

ideais para tarefas como identificação de objetos em imagens. Por outro lado, para atividades de Processamento de Linguagem Natural, modelos baseados em redes neurais recorrentes podem se mostrar mais apropriados.

Aquisições extensas de dados rotulados são imperativas para um treinamento efetivo da rede neural. A exploração de conjuntos de dados públicos, como o ImageNet, que engloba milhões de imagens com anotações, promoveu avanços notáveis no desempenho das redes neurais em desafios de Visão Computacional. Além disso, técnicas de pré-treinamento, onde uma rede neural é inicialmente treinada em um conjunto de dados abrangente e genérico, seguido de ajustes finos para tarefas específicas, também potencializam o desempenho em contextos com escassez de dados anotados.

No trabalho com Visão Computacional, por exemplo, existem algoritmos como a CNN que são utilizados para solucionar problemas de classificação de imagens. Essas redes são compostas por várias camadas que desempenham funções específicas, desde a camada de entrada até as camadas de saída.

As camadas convolucionais extraem características das imagens através da convolução com filtros e reduzem a dimensionalidade dos dados. As camadas de *pooling* reduzem ainda mais a dimensionalidade dos dados, reduzindo a resolução da imagem e aumentando a eficiência do processamento. As camadas totalmente conectadas combinam as informações extraídas pelas camadas anteriores e geram as saídas finais (Chollet, 2021).

1.1. MOTIVAÇÃO

A elaboração de um algoritmo de aprendizado profundo não é uma tarefa simples. A habilidade de construir um algoritmo eficaz perpassa pela capacidade do programador em ajustar de maneira eficaz os hiperparâmetros, essenciais para capacitar o algoritmo a atingir seu melhor desempenho. Isto é fundamental porque, de maneira similar a muitos outros algoritmos de Inteligência Artificial, o êxito de um algoritmo de aprendizado profundo está intrinsecamente ligado à sua habilidade de generalização diante dos desafios de classificação e à sua acurácia.

A convergência dos hiperparâmetros ideais, capazes de conferir eficácia a um algoritmo, é o cerne do sucesso dos modelos de aprendizado profundos mais renomados até o presente momento. Algumas dessas características demonstram serem tão eficazes que acabam por se tornar práticas padronizadas, princípios não oficialmente declarados, mas que são incorporados a outros algoritmos na busca por melhores resultados.

Uma ilustração desse princípio é encontrada nas camadas residuais. Essa técnica, que implica na propagação de uma cópia do *input* entre os blocos de convolução, foi concebida pelos desenvolvedores da ResNet para mitigar os desafios de desaparecimento do gradiente, acabando por se tornar um padrão no desenvolvimento de CNNs até hoje. Esse método é especialmente utilizado em algoritmos com um grande número de camadas.

A profundidade nas redes neurais convolucionais emergiu como um paradigma predominante na Visão Computacional. A busca por melhorias em tarefas complexas de reconhecimento visual e Processamento de Imagens incentivou a construção de modelos cada vez mais profundos, aproveitando-se de arquiteturas multicamadas para capturar hierarquias complexas de características visuais. Entretanto, a corrida pela profundidade também suscitou questões inerentes a essa característica.

Dessa forma, embora a profundidade das redes tenha provado sua eficácia em uma variedade de aplicações, também suscitou preocupações substanciais, como o aumento acentuado no número de parâmetros treináveis (Domingos, 2012). Em diversos cenários, um grande número de parâmetros pode conduzir ao *overfitting* (Dolphin, 2022), onde o modelo se adapta excessivamente aos dados de treinamento, prejudicando sua capacidade de generalizar para novos exemplos, por problemas relacionados à quantidade de parâmetros treináveis e a quantidade de dados de treinamento.

Além disso, essa expansão na arquitetura não apenas aumenta a demanda por poder computacional, muitas vezes requerendo *hardware* especializado, como *Graphic Processing Unit* (GPU) ou *Tensor Processing Unit* (TPU), para treinamento e inferência eficazes, mas também pode resultar em uma taxa de inferência mais lenta, o que pode limitar a aplicabilidade prática da rede em cenários de tempo real.

Nesse contexto, torna-se crucial encontrar um equilíbrio entre a profundidade da rede e a quantidade de parâmetros treináveis. Criar uma estratégia que viabilize a simplificação do modelo, como a abordagem proposta neste estudo, emerge como soluções promissoras em situações onde a aplicação deve operar em dispositivos de menor capacidade.

Tais estratégias têm como objetivo preservar a habilidade da rede em capturar informações relevantes, embora com uma contagem de parâmetros mais reduzida, na tentativa de resultar em uma rede com maior eficiência e eficácia. A reflexão sobre o paradigma da profundidade em redes neurais convolucionais instiga a ponderar cuidadosamente a interação entre a complexidade do modelo e sua capacidade de generalização, especialmente quando se busca modelos mais enxutos e viáveis para *hardwares* de menor custo.

Outra motivação que consideramos importante para uma pesquisa científica é a sua capacidade de aplicação social. Para além do contexto acadêmico, a implementação prática de uma pesquisa, incluindo está sobre tecnologia, pode desempenhar um papel crucial no progresso social das comunidades onde é conduzida.

Ao contribuir para o desenvolvimento social de uma região específica, uma pesquisa adquire um valor adicional ao fornecer insights que podem ser empregados na solução de desafios gerais ou contextuais (Cavalcante, 2021).

A proposta de nossa pesquisa se alinha a essa perspectiva, pois buscamos através do desenvolvimento tecnológico fomentar novas formas de aperfeiçoamento de técnicas e práticas da produção do Sisal. Esperamos que nossos resultados tenham um impacto positivo significativo no processo produtivo do sisal, refletindo-se também na melhoria da qualidade de vida das comunidades envolvidas.

1.2. PROBLEMAS DE PESQUISA E HIPÓTESES

Dessa maneira, levando em consideração o que foi definido, elencamos os seguintes questionamentos que servirão de norte para o decorrer de nossa pesquisa:

Questão de Pesquisa 1: É possível desenvolver um modelo cuja eficiência, e consequentemente a totalidade de seus parâmetros treináveis, não esteja vinculada à sua profundidade?

Questão de Pesquisa 2: É possível desenvolver um modelo cuja eficiência não dependa do tamanho da imagem para o qual seja apresentado?

Questão de Pesquisa 2a: Adjacente à hipótese anterior, surge a indagação se é viável desenvolver um modelo cuja eficiência não esteja condicionada à aplicação de melhorias na imagem original?

Questão de Pesquisa 3: É possível desenvolver um modelo que realize a classificação correta da planta e da fibra do sisal, mantendo uma boa acurácia e que possa ser utilizada em um dispositivo de baixo custo?

A partir dos problemas elencados nós supomos a seguinte hipótese que será testada no decorrer de nossa pesquisa:

Hipótese 1: É viável desenvolver um modelo cuja eficiência, e a totalidade de seus parâmetros treináveis, não dependam completamente da sua profundidade, nem dependam do tamanho da imagem apresentada.

Adjacente a essa hipótese principal nós formulamos também outras duas hipóteses específicas onde modelo é aplicado a um contexto de imagens sem nenhum tipo de tratamento prévio, processo análogo ao que utilizamos em nossa pesquisa com o banco de imagens do Sisal.

Hipótese 1a: É viável desenvolver um modelo cuja eficiência não esteja condicionada à aplicação de melhorias na imagem original;

Hipótese 1b: É possível realizar a classificação correta da planta e da fibra da planta do Sisal, mantendo uma boa acurácia, e que possa ser utilizado em um dispositivo de baixo custo.

1.3. OBJETIVOS

Na maioria dos casos, a obtenção de resultados de classificação satisfatórios por meio de modelos de CNN implica na adoção de uma estrutura profunda com várias camadas. Contudo, essa profundidade muitas vezes traz consigo um aumento substancial na quantidade de parâmetros treináveis. Essa correlação entre a profundidade da rede e a complexidade dos parâmetros é um aspecto crítico a ser considerado.

De maneira geral, a execução de algoritmos de CNN normalmente requer unidades de processamento gráfico (GPUs) ou unidades de processamento tensorial (TPUs) devido à intensidade computacional dessas tarefas. Isso traz a necessidade de recursos computacionais substanciais, especialmente em casos em que a tarefa envolve um modelo com um grande número de parâmetros.

Nesse contexto, torna-se essencial avaliar a relação custo-benefício desses algoritmos, visto que a complexidade resultante pode se traduzir em altos custos tanto em termos de tempo de treinamento quanto de investimentos em *hardware*. Por essa razão e considerando a necessidade de redução máxima de custos financeiros e operacionais de implementação de um modelo nesse cenário, objetivamos**:**

- **Propor uma arquitetura de CNN capaz de viabilizar a elaboração de modelos mais eficientes, diminuindo o número de parâmetros treináveis, e permitindo sua utilização em contextos com hardware restrito, sem comprometer a precisão.**

A viabilidade do uso desses modelos pode vir a depender também das demandas de tarefas específicas e dos recursos disponíveis. Em muitos casos, os custos associados a esses

fatores podem se tornar significativos a ponto de inviabilizar a utilização desses algoritmos em cenários de recursos limitados.

Levando em consideração essas condições e considerando também o contexto regional de nossa pesquisa, nós optamos por desenvolver um algoritmo que leve em consideração também os seguintes objetivos específicos:

- **No algoritmo proposto, apresentamos uma nova funcionalidade que possibilita uma menor quantidade de parâmetros treináveis no modelo, por meio da redução escalável da resolução da imagem;**
- **Ao incorporar esta nova funcionalidade, propor que essa nova arquitetura seja ativa e ajustável de maneira autônoma. Considerando um elemento importante da imagem, a sua resolução;**
- **Propor um algoritmo que possibilite a análise de imagens de resoluções diversas, sem que para isso seja necessária a aplicação de aprimoramentos que beneficiem ou melhorem a efetividade do modelo;**

1.4. PRINCIPAIS CONTRIBUIÇÕES

As Redes Neurais Convolucionais consideradas profundas, do termo em inglês *deep*, se refere à CNNs com muitas camadas, o que lhes permite extrair recursos complexos e de alto nível das imagens de entrada, enquanto o termo *shallow* se refere à CNNs com poucas camadas (Lei et.al., 2020).

As redes *shallow* podem ser úteis em casos onde há pouca variação ou complexidade nas imagens de entrada, como, por exemplo, em tarefas de reconhecimento de caracteres. Já as redes profundas são mais adequadas para tarefas de Visão Computacional mais complexas, como classificação de imagens em grandes conjuntos de dados, detecção de objetos, reconhecimento facial, entre outras (LeCun, Bengio e Hinton, 2015).

Uma CNN típica é composta por várias camadas, sendo as camadas convolucionais as mais importantes. Essas camadas aplicam filtros que deslizam sobre a imagem de entrada e realizam operações de convolução para extrair recursos relevantes, como bordas, texturas e padrões. As camadas convolucionais são seguidas por camadas de *pooling*, que reduzem a dimensão espacial dos recursos extraídos, tornando a rede mais eficiente computacionalmente.

Tanto CNNs profundas quanto CNNs rasas podem lidar com a redução da resolução do *input*, mas a profundidade da rede pode influenciar o quão bem ela lida com essa tarefa. Uma CNN rasa pode ter dificuldade em preservar informações de alta resolução em imagens, pois ela não tem muitas camadas para aprender representações hierárquicas.

As CNNs profundas têm muitas camadas, o que lhes permite aprender representações hierárquicas mais complexas e de alto nível a partir das imagens de entrada. À medida que as camadas convolucionais se aprofundam, a dimensão dos dados de saída é reduzida, o que leva a uma redução da resolução espacial dos dados da imagem.

No entanto, redes muito profundas podem ter dificuldade em preservar informações importantes em imagens com alta resolução, pois as camadas convolucionais podem reduzir a dimensão dos dados de saída a um ponto em que informações críticas podem ser perdidas. Essa discussão sobre a dimensionalidade das CNNs em torno de paradigmas da sua profundidade e suas implicações foram abordadas em nosso artigo:

- FERREIRA, Verner Rafael; DE PAULA CANUTO, Anne Magaly. Potencializando a classificação de imagens com eficiência e robustez através da FiberNet. Revista de Ciência da Computação, v. 5, n. 1, p. 7-20, 2023.

Por outro lado, CNNs rasas têm menos camadas, o que lhes permite processar as informações de entrada de forma mais direta e manter uma resolução espacial mais alta. No entanto, isso pode limitar a capacidade da rede de aprender representações mais complexas e de alto nível. Para superar essa limitação, as CNNs rasas podem incorporar técnicas de *upsampling* (Dong et.al., 2014) em camadas deconvolutivas para ajudar a preservar a resolução em imagens de entrada.

Em contraponto à técnica de *upsampling* existem também as técnicas de *downsampling* que se refere à redução da resolução espacial da imagem ou dos recursos gerados pelas camadas anteriores da rede. O *downsampling* é frequentemente realizado por meio de camadas de *pooling*, como a camada *maxpooling* ou *averagepooling*, que reduzem o tamanho da imagem ou dos recursos ao selecionar o valor máximo ou médio de um grupo de pixels adjacentes (LeCun et.al., 1998).

A redução da resolução espacial por meio do *downsampling* ajuda a reduzir a dimensionalidade dos dados, tornando-os mais fáceis de processar pelas camadas posteriores da rede. Além disso, o *downsampling* ajuda a aumentar a eficiência computacional,

permitindo que a rede processe imagens maiores e mais complexas com menos recursos computacionais (He et.al., 2016).

Em nosso trabalho, nós desenvolvemos uma nova camada que pode ser considerada como um *downsampling* pré-convolutivo, pois nós aplicamos o nosso método antes das camadas de convolução. Mesmo não existindo uma regra que determine onde as camadas de *downsampling* podem ser inseridas, é mais comum encontrarmos técnicas de *downsampling* sendo utilizadas na fase de convolução das redes convolutivas. Os detalhes concernentes ao desenvolvimento dessa nova camada foram detalhados no seguinte artigo:

- FERREIRA, Verner Rafael; DE PAULA CANUTO, Anne Magaly. FiberNet: A Compact and Efficient Convolutional Neural Network Model for Image Classification. In: Anais do XX Encontro Nacional de Inteligência Artificial e Computacional. SBC, 2023. p. 257-271.

Nosso método é compreensível e consiste em remover linhas, através de uma função específica, nos eixos da altura e largura do *input*. Esse processo é realizado para reduzir a resolução da imagem ou *input* antes das camadas de convolução. Com nosso método nós intencionamos promover uma redução gradual no *input* para consequentemente forçar uma redução no número de parâmetros treináveis da rede e com isso obtermos ganho de velocidade nas taxas de inferência e transferência da rede.

1.5. ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho está organizado em 07 capítulos: o Capítulo 2 apresenta os principais conceitos utilizados para o desenvolvimento deste trabalho. No Capítulo 3 apresentaremos alguns estudos relacionados com o tema deste trabalho, apresentando uma visão geral do que foi desenvolvido na aplicação de CNNs na identificação e proposição de solução de problemas que envolvem redes de pequeno porte e alguns tipos de plantas e frutas.

No Capítulo 4 será apresentada a nossa arquitetura de CNN bem como a inovação desenvolvida para os propósitos de nossa pesquisa enquanto que no Capítulo 5 apresentamos as informações relacionadas à nossa metodologia incluindo a descrição detalhada dos bancos de imagens (*dataset*s) utilizados, a descrição da análise empírica realizada durante a nossa pesquisa e uma descrição sobre o nosso protótipo utilizado, a FiberNet.

Os resultados obtidos são apresentados no Capítulo 06, enquanto que no Capítulo 07 nós apresentamos nossa conclusão incluindo nossas perspectivas para trabalhos futuros.

CAPÍTULO 2.

CONCEITOS RELACIONADOS

2.1. APRENDIZADO DE MÁQUINA

Aprendizado de Máquina (do inglês *Machine Learning*) é uma subárea da Inteligência Artificial (IA) que se concentra no desenvolvimento de algoritmos e técnicas que permitem que sistemas de computadores aprendam e melhorem a partir de dados (Murphy, 2012). O objetivo da Aprendizagem de Máquina é criar algoritmos capazes de aprender com dados, reconhecer padrões e tomar decisões inteligentes sem intervenção humana direta. Isso é alcançado através do uso de técnicas estatísticas e de análise de dados para identificar correlações, tendências e padrões em grandes conjuntos de dados (Flach, 2012).

Os algoritmos de Aprendizagem de Máquina são capazes de analisar e encontrar padrões em grandes volumes de dados, que seriam impossíveis de serem analisados por seres humanos sozinhos (Bishop, 2006). Os grandes conjuntos de dados podem ser de diferentes tipos, como dados de transações comerciais, dados de redes sociais, dados de sensores, entre outros.

Através do uso de técnicas de Aprendizagem de Máquina, é possível extrair informações valiosas desses dados, como *insights* de negócios, tendências de mercado, comportamento do consumidor, previsões de demanda, entre outros. Por exemplo, em um conjunto de dados de registros médicos de pacientes, a Aprendizagem de Máquina pode ser usada para identificar padrões que indiquem uma maior probabilidade de desenvolvimento de uma determinada doença em uma determinada população (Tekale et. al., 2018).

Além disso, a Aprendizagem de Máquina pode ser usada para automatizar tarefas complexas, como a classificação de imagens ou a identificação de fraudes financeiras. Em ambos os casos, o volume de dados é muito grande para ser processado manualmente, e a Aprendizagem de Máquina é capaz de identificar padrões e anomalias, neste contexto, de forma muito mais eficiente do que um ser humano (Géron, 2022).

2.1.1. Tipos de Aprendizado de Máquina

A Aprendizagem de Máquina é uma área da Inteligência Artificial que se concentra no desenvolvimento de sistemas que podem aprender e se adaptar sem serem explicitamente

programados. Esse tipo de aprendizado é baseado no conceito de que os sistemas podem aprender a partir de dados, identificando padrões e tendências que podem ser usados para tomar decisões ou realizar tarefas.

A Aprendizagem de Máquina pode ser dividida em quatro tipos principais:

- Aprendizado supervisionado: Nesse tipo de aprendizado, o sistema é treinado com dados rotulados, ou seja, dados que já possuem uma resposta correta. O sistema aprende a associar os dados de entrada com a resposta correta, e pode então usar esse conhecimento para fazer previsões sobre dados novos.
- Aprendizado não supervisionado: Nesse tipo de aprendizado, o sistema é treinado com dados não rotulados. O sistema aprende a identificar padrões nos dados, sem ter acesso a respostas corretas. Esse tipo de aprendizado pode ser usado para tarefas como agrupamento de dados, redução de dimensionalidade e detecção de anomalias.
- Aprendizado semisupervisionado: Nesse tipo de aprendizado, o sistema é treinado com uma combinação de dados rotulados e não rotulados. O sistema usa os dados rotulados para aprender a associar os dados de entrada com a resposta correta, e usa os dados não rotulados para refinar seu aprendizado.
- Aprendizado por reforço: Nesse tipo de aprendizado, o sistema é premiado por tomar decisões que levam a resultados desejados. O sistema aprende a tomar decisões de forma mais eficaz, experimentando diferentes possibilidades e observando as consequências de suas ações.

2.2. REDES NEURAIS ARTIFICIAIS (RNA)

Redes neurais artificiais (RNA) são modelos matemáticos inspirados na estrutura neural do cérebro humano. Elas são compostas por um conjunto de unidades, chamadas de neurônios, que estão interconectadas. Cada neurônio recebe um conjunto de entradas e produz uma saída.

As RNAs são usadas para resolver uma ampla gama de problemas, incluindo classificação, regressão, reconhecimento de padrões e Aprendizado de Máquina. Elas são particularmente eficazes para tarefas que requerem aprendizado de padrões complexos, como a identificação de objetos em imagens ou a tradução de idiomas. Os neurônios da fase densa de uma CNN, por exemplo, funcionam como uma RNA tradicional. A fase densa, conhecida como camada totalmente conectada (do inglês, *fully connected layer)* é composta de

neurônios conectados a todos os neurônios da camada anterior, assim como em uma RNA clássica.

A arquitetura de uma RNA é composta por três componentes principais:

- Camadas de entrada: recebem os dados de entrada do problema a ser resolvido;
- Camadas ocultas: são responsáveis pela aprendizagem dos padrões dos dados;
- Camada de saída: produz a saída da rede, que pode ser um valor, uma classificação ou outra representação do problema.

O número de camadas ocultas e o número de neurônios em cada camada são parâmetros que podem ser ajustados para melhorar o desempenho da rede. E o aprendizado nas redes neurais artificiais se dá através de um processo de ajuste dos pesos das sinapses entre os neurônios. Esse ajuste é feito usando um algoritmo de treinamento, como o *backpropagation* (Rumelhart, Hinton e Williams, 1986). É este algoritmo que calcula o erro da rede em relação aos dados de saída esperados. Esse erro é então usado para ajustar os pesos das conexões de forma a reduzir o erro.

2.3. APRENDIZADO PROFUNDO (DO INGLÊS DEEP LEARNING)

O aprendizado profundo é um campo da Inteligência Artificial que usa RNA para resolver problemas complexos. Ele se baseia na ideia de que os neurônios podem aprender a identificar padrões nos dados, mesmo que esses padrões sejam sutis ou complexos.

As RNA usadas para aprendizado profundo geralmente têm uma arquitetura mais complexa e elas podem ter várias camadas ocultas, com milhões ou até bilhões de neurônios. As camadas ocultas são responsáveis pela aprendizagem dos padrões dos dados. O número de camadas ocultas e o número de neurônios em cada camada são parâmetros que podem ser ajustados para melhorar o desempenho da rede.

Existem diversos tipos de algoritmos para o aprendizado profundo que podem ser utilizadas em uma ampla gama de aplicações, incluindo Visão Computacional, Processamento de Linguagem Natural, geração de dados artificiais e controle e tomada de decisão. São exemplos de aprendizado profundo: as redes neurais convolucionais (CNN), as redes neurais recorrentes (RNN), redes neurais generativas (GANs) entre outras.

2.3.1. Convolutional Neural Network (Cnn)

Na área da Inteligência Artificial, é comum utilizar algoritmos que permitem a sistemas de computadores aprenderem com os dados, identificarem padrões e tomarem decisões de forma autônoma, sem a necessidade de intervenção humana direta. Grandes volumes de dados (do inglês, *big data*) tem um papel importante na Aprendizagem de Máquina pois, oferecem uma vasta quantidade de informações que os algoritmos podem explorar para aprimorar sua capacidade de aprendizado.

As redes neurais convolucionais são um tipo de Inteligência Artificial que se destaca na análise de grandes conjuntos de dados, especialmente em tarefas relacionadas a imagens e Visão Computacional, como o trabalho realizado pelas GANs (Goodfellow, Ian et al., 2020). Elas são inspiradas na estrutura do cérebro humano e são compostas por várias camadas de neurônios artificiais interconectados (Li et. al., 2015).

As CNNs são especialmente adequadas para lidar com grandes conjuntos de dados de imagens, pois possuem a capacidade de identificar e extrair características importantes das imagens, como bordas, texturas e formas, que podem ser usadas para classificar objetos e reconhecer padrões. As CNNs são compostas por várias camadas, incluindo camadas de convolução, camadas de *pooling* e camadas totalmente conectadas, e são treinadas usando um conjunto de dados rotulados para identificar padrões e aprender a classificar novas imagens (Goodfellow, 2016).

A camada de convolução é a principal camada das redes neurais convolucionais e é responsável por detectar recursos nas imagens de entrada. O processo dessa camada envolve a aplicação de um conjunto de filtros, também conhecidos como *kernels*, sobre a imagem de entrada para gerar mapas de recursos.

Cada filtro é uma matriz de pesos que é aplicada a uma pequena região da imagem de entrada, chamada de janela de convolução. A janela de convolução é deslizada pela imagem de entrada, e o filtro é aplicado a cada posição da janela, gerando um valor escalar correspondente a essa posição. O resultado dessa operação é um mapa de características (*feature map*), que destaca características específicas da imagem, como bordas, texturas e padrões. Esses mapas de características são passados para a próxima camada da rede, onde são combinados e processados para gerar uma representação mais abstrata da imagem.

2.3.1.1. O processo de convolução

A convolução é a primeira fase do processamento de uma imagem em uma rede neural convolucional (CNN). Nessa etapa, um filtro é deslizado sobre a imagem de entrada para identificar padrões (LeCun et.al., 1998). O filtro é uma matriz de números menores do que a imagem de entrada e é composto por valores que são inicializados aleatoriamente, mas que serão ajustados durante o treinamento da CNN para melhorar a precisão da classificação.

O filtro é posicionado no canto superior esquerdo da imagem de entrada e é deslizado horizontalmente e verticalmente para percorrer toda a imagem de entrada. Em cada posição, o filtro é multiplicado pelos valores correspondentes da imagem de entrada. Esses produtos são somados para produzir um único valor que é armazenado em um mapa de características. Esse processo é representado através da Figura 1.

Por exemplo, suponha que a imagem de entrada seja uma matriz com resolução de 5x5 de pixels em escala de cinza e o filtro seja uma matriz de resolução de 3x3. Na primeira posição do canto superior esquerdo, o filtro é sobreposto nos primeiros 3x3 pixels da imagem e a multiplicação de elementos é realizada entre os valores do filtro e os valores correspondentes da imagem de entrada que estão sob o filtro. Os resultados são somados para produzir um único valor, que é armazenado na matriz resultante como um mapa de características (Goodfellow, Bengio e Courville, 2016).

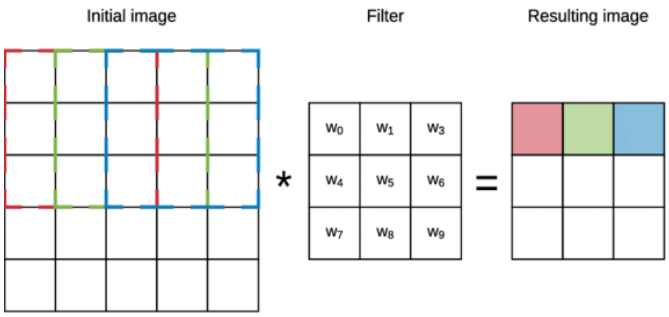


Figura 1: Representação do cálculo de convolução entre imagem e filtro. Fonte: https://www.researchgate.net/profile/Jose-Padarian

Cada filtro usado na convolução é ajustado durante o treinamento para detectar características específicas da imagem, como bordas, linhas, formas ou outras características relevantes para a tarefa de classificação. Esses filtros são inicializados com valores aleatórios e, durante o treinamento, são ajustados para *max*imizar a precisão da classificação.

O processo de convolução é repetido várias vezes, usando diferentes filtros para extrair diferentes características da imagem de entrada. Cada filtro produz um mapa de

características separado, que representa a presença ou ausência de características específicas na imagem.

O resultado final da etapa de convolução é um conjunto de mapas de características que foram extraídos da imagem de entrada. Esses mapas de características são então passados para a próxima camada da CNN, onde serão aplicadas funções de ativação e operações de *pooling* para reduzir a dimensionalidade dos dados e extrair características de níveis mais abstratos da imagem.

2.3.1.2. Hiperparâmetros

Para realizar a aplicação do filtro durante o processo de convolução algumas configurações são necessárias, essas configurações são denominadas de hiperparâmetros. Alguns desses hiperparâmetros são: *kernel size*, *pooling*, *padding*, função de ativação (do inglês, *activation function*), regularização e o *stride*.

O *kernel size* (tamanho do *kernel*) é um dos hiperparâmetros da camada de convolução em uma rede neural convolucional (CNN). O *kernel* é uma matriz que é deslizada sobre a imagem de entrada durante a operação de convolução, gerando um mapa de características (Cun, 1998). O tamanho do *kernel* determina as dimensões da matriz, tipicamente uma matriz quadrada de dimensão (*k, k*), onde *k* é um valor que pode ser ajustado.

No entanto, recomenda-se atualmente o uso de valores ímpares, como 3, 5 ou 7, conforme sugerido por (Chollet, 2021), pois quando se usa um *kernel* com um tamanho ímpar (por exemplo, 3x3), ele possui um ponto central, o que pode ser benéfico para operações de convolução.

Ao usar um filtro com um ponto central, a aplicação da convolução mantém um aspecto mais simétrico, o que pode ser útil em muitos contextos. A simetria é desejável porque ajuda a preservar as características espaciais e a manter a coerência nos padrões que a rede está aprendendo.

O *pooling* é uma técnica usada para reduzir a dimensionalidade dos mapas de características produzidas pela camada de convolução. Essa técnica envolve a divisão da imagem em regiões (por exemplo, em blocos de 2x2 pixels) e a escolha do valor máximo ou médio em cada região (LeCun et. al., 1998). O *pooling* tem dois principais objetivos: reduzir a complexidade do modelo e fornecer alguma invariância a pequenas transformações na imagem de entrada, como rotações ou deslocamentos (Raschka, 2015).

Existem dois tipos principais de *pooling*: o *max pooling* (LeCun, 1998) e o *average pooling* (Krizhevsky, 2012). No *max pooling*, o valor máximo em cada região é escolhido como o valor de saída da região. No *average pooling*, o valor médio dos valores em cada região é escolhido como o valor de saída da região. Em geral, o *max pooling* é mais comum na prática, pois tende a funcionar melhor para tarefas de classificação de imagem. No entanto, em algumas situações, o *average pooling* pode ser mais adequado, por exemplo, quando a preservação de informações de baixa frequência é importante.

O *padding* é uma técnica usada em redes neurais convolucionais (CNN) para aumentar o tamanho da imagem de entrada, adicionando pixels à borda da imagem, sem alterar o tamanho da imagem real. A técnica é usada para preservar informações importantes nas bordas da imagem durante a convolução e evitar que as dimensões da imagem diminuam rapidamente em camadas convolucionais (LeCun, 1989).

Dentre os vários modos existentes atualmente, podemos citar dois como exemplos de aplicação de *padding*: o "*valid"* e o "*same"*. O *padding* "*valid"* não adiciona pixels à imagem de entrada e a convolução é aplicada apenas em áreas da imagem onde o *kernel* pode ser aplicado completamente. Como resultado, a saída é menor que a imagem de entrada. O *padding* "*same"* adiciona pixels à imagem de entrada para que a saída tenha as mesmas dimensões que a imagem de entrada.

Além desses tipos, existe o *padding* causal, que é usado em modelos que precisam prever a saída apenas com base nas entradas anteriores, como em séries temporais. O *padding* causal adiciona valores somente à esquerda da imagem (ou à direita, dependendo do tipo de convolução) e somente nas camadas posteriores, para evitar vazamentos de informações do futuro para o presente. O objetivo é garantir que a saída da rede só dependa das entradas passadas e não das entradas futuras.

As funções de ativação, do inglês *activation function*, são um dos principais blocos de construção das redes neurais, incluindo as redes convolucionais (CNNs). Essas funções são aplicadas após cada camada convolucional para introduzir não linearidade na saída, o que permite que a rede neural modele relações complexas e não lineares entre as entradas e as saídas (Ramachandran, Zoph e Le et.al., 2017).

A função de ativação é responsável por introduzir a não linearidade em uma CNN. Sem ela, a CNN seria apenas uma transformação linear dos dados de entrada. A função de ativação age como um "interruptor" que ativa ou desativa a saída de cada neurônio da rede, dependendo da entrada que recebe. Isso permite que a rede neural capture relações não lineares entre as entradas e as saídas, o que é essencial para muitas tarefas de Aprendizado de

Máquina, como reconhecimento de imagem, Processamento de Linguagem Natural, entre outros.

Diversas funções de ativação são amplamente empregadas nas redes CNN, contribuindo para aprimorar sua capacidade de aprendizado. Uma das opções mais populares é a função ReLU (do inglês, *Rectified Linear Unit*), que se destaca por retornar a entrada se esta for maior que zero; caso contrário, é atribuído o valor zero. Em termos computacionais, a ReLU também desempenha um papel crucial na mitigação do problema do desaparecimento do gradiente (Hahnloser et al., 2000).

A função *sigmoid*e, por sua vez, funciona com saídas na faixa de 0 a 1. Essa característica a torna especialmente adequada para problemas de classificação binária. Entretanto, é importante mencionar que a *sigmoid*e enfrenta desafios relacionados ao desaparecimento dos gradientes quando as entradas se tornam excessivamente grandes ou pequenas. Essa situação pode afetar negativamente o processo de treinamento da rede.

Outra função de ativação recorrente é a tangente hiperbólica que projeta valores dentro do intervalo de -1 a 1. Comumente aplicada em problemas de regressão e classificação. Além destas citadas existem também algumas outras variações como a função *softmax*, empregada em situações de classificação multiclasse e a função de ativação ELU (Exponential Linear Unit) que é uma alternativa à ReLU, pois ela se apresenta como uma solução para o problema de *dying* ReLU, permitindo uma faixa negativa suave ao mesmo tempo em que mantém uma ativação não linear na faixa positiva. Isso contribui para uma melhor convergência do treinamento e evita problemas de gradiente zero.

A regularização é uma técnica, ou conjunto de técnicas, usadas para evitar o *overfitting*, que é o problema de a rede neural se ajustar muito bem aos dados de treinamento, mas não generalizar bem para dados novos. Existem várias técnicas de regulação que podem ser usadas em CNNs, e algumas das mais comuns são:

Dropout (Srivastava, 2014): é uma técnica que consiste em desligar aleatoriamente alguns neurônios da rede durante o treinamento. Isso força a rede a encontrar caminhos alternativos para mapear as entradas para as saídas, tornando-a mais robusta e menos propensa a *overfitting*.

Normalização de lote, do inglês *batch normalization*, é uma técnica que normaliza as ativações de cada camada da rede, tornando mais fácil o treinamento em camadas profundas. A normalização de lote também pode ajudar a regularizar a rede, pois introduz algum ruído nas ativações da rede (Ioffe and Szegedy, 2015).

Aumento de dados, do inglês *data augmentation*, é uma técnica que consiste em gerar novos dados de treinamento a partir dos dados existentes, aplicando transformações aleatórias, como rotações, ampliações e deslocamentos. Isso aumenta a diversidade dos dados de treinamento e ajuda a evitar o *overfitting*.

O *overfitting* e o *underfitting* são dois problemas comuns em redes neurais convolucionais (CNNs) que podem afetar negativamente seu desempenho e precisão na tarefa de classificação (Samuel, 1967). O *overfitting* ocorre quando a CNN aprende a mapear muito bem os dados de treinamento, mas falha em generalizar para novos dados não vistos no treinamento.

Isso pode acontecer quando a CNN é muito complexa em relação ao conjunto de dados de treinamento, levando a um ajuste excessivo aos dados de treinamento e ignorando padrões mais gerais que seriam importantes para a classificação correta dos dados de teste. Outra causa comum de *overfitting* é o conjunto de treinamento ser muito pequeno, o que pode levar a uma memorização dos dados de treinamento pela CNN, em vez de aprender padrões gerais.

Por outro lado, o *underfitting* ocorre quando a CNN não é suficientemente complexa para aprender as características relevantes do conjunto de dados. Nesse caso, a CNN falha em capturar todos os padrões importantes dos dados de treinamento e não consegue generalizar para novos dados.

Para evitar o *overfitting*, é comum usar técnicas como *dropout*, regularização L2, *data augmentation*, *early stopping* e *cross-validation*, que ajudam a impedir que a CNN se torne muito complexa e a memorizar os dados de treinamento. Já para evitar o *underfitting*, é importante garantir que a CNN tenha camadas suficientes e que seja treinada por tempo suficiente para aprender as características relevantes dos dados.

O *stride* (Krizhevsky, 2012) é um parâmetro utilizado na camada de convolução de uma rede neural convolucional (CNN) que define o passo em que o filtro se move ao longo da imagem de entrada. É basicamente a quantidade de pixels que o filtro se move horizontalmente e verticalmente em cada passo.

O filtro de convolução é uma matriz pequena que é deslocada pela entrada de dados, elemento por elemento. Durante cada deslocamento, a multiplicação pixel a pixel é realizada entre o filtro e a parte correspondente da entrada de dados como ilustrado na Figura 2. Essa operação resulta em um valor que é colocado em uma nova matriz de saída também conhecida como mapa de características. Na ilustração nós temos uma imagem com a resolução de 5x5 e uma linha de *padding* representados por linhas de pixels pontilhados ao

redor da imagem. A tela de convolução (*kernel*) é representada por uma matriz de resolução de 3x3.

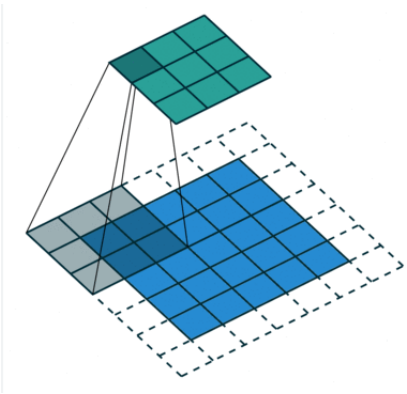


Figura 2: Exemplo de aplicação do filtro de **convolução. Fonte:** **https://commons.wikimedia.org/wiki/File:Convolution_arithmetic_-_Padding_*strides*.gif**

Nesse exemplo, o filtro se desloca um pixel por vez da esquerda para a direita. Ao chegar ao final da linha, ele desce um pixel e reinicia o deslocamento. Se o valor for definido como 2, o filtro avança dois pixels, e assim por diante. Uma conclusão é que um *stride* maior reduz a resolução da saída e acelera o processamento, enquanto um *stride* menor aumenta a resolução da saída e diminui a velocidade do processamento. O uso de um *stride* maior que 1 reduz o tamanho da saída do mapa de características em relação à entrada original, pois o filtro pula pixels durante o deslocamento. Isso pode ser útil para reduzir a resolução espacial dos dados ou extrair características em uma escala maior.

O uso *stride* permite controlar a resolução do mapa de características produzido pela convolução. Um *stride* maior reduz a resolução do mapa de características, o que pode ser útil em algumas tarefas, como a detecção de objetos em imagens, onde a identificação precisa da localização de objetos é mais importante do que a sua forma exata. Dentre outros benefícios da variação do valor do *stride* para o processo de convolução podemos citar:

- Redução de *overfitting*: Usar um *stride* maior também pode ajudar a reduzir o *overfitting*, pois diminui o número de parâmetros treináveis na rede e aumenta a capacidade da rede de generalizar para dados novos e não vistos durante o treinamento.
- Aceleração de computação: O uso de *stride* maior também pode acelerar a computação durante a convolução, pois reduz o número de operações necessárias para calcular a saída do mapa de características.

- Controle de tamanho do filtro: Em alguns casos, usar um *stride* maior permite que sejam usados filtros maiores sem aumentar significativamente o tempo de computação necessário para a convolução.
- Aumento de robustez: O uso de *stride* maior também pode aumentar a robustez da rede a pequenas variações na entrada de dados, pois a convolução é menos sensível a pequenos deslocamentos na entrada.

2.3.1.3. Arquitetura

Uma rede neural convolucional é uma arquitetura de rede neural profunda especialmente projetada para processar dados em formato de grade, como imagens, vídeos ou sinais de áudio. A CNN é composta por várias camadas, cada uma realizando um conjunto de operações que transformam os dados de entrada em uma saída que pode ser usada para classificação, segmentação ou outras tarefas (LeCun et.al., 1998).

Todas as partes de uma CNN são importantes e trabalham juntas para realizar tarefas específicas, mas algumas partes têm mais impacto no desempenho e eficiência da rede do que outras, dependendo da tarefa em questão (Bengio, 2009). As camadas convolucionais são frequentemente consideradas a parte mais importante de uma CNN, pois elas extraem recursos relevantes da entrada e são responsáveis por grande parte da capacidade de generalização da rede (Simonyan e Zisserman, 2014).

As camadas de ativação são cruciais para a rede neural aprender a relação não linear entre os recursos extraídos pelas camadas convolucionais e a saída final. A função de ativação mais comum é a função ReLU, que é fácil de calcular e tem se mostrado eficaz na prática.

As camadas totalmente conectadas também são importantes, pois elas combinam os recursos extraídos pelas camadas anteriores em uma saída final (Zeiler e Fergus, 2014). No entanto, em algumas arquiteturas mais recentes, as camadas totalmente conectadas são substituídas por camadas convolucionais adicionais, permitindo que a rede tenha mais parâmetros e seja mais profunda.

A camada de saída é crítica para a tarefa específica que a rede está tentando resolver, pois ela produz a saída final da rede (Michelucci, 2018). A escolha do número e do tipo de neurônios na camada de saída depende do problema, e uma escolha adequada pode melhorar significativamente o desempenho da rede.

Na camada totalmente conectada ou camada densa cada neurônio dessa camada está conectado a todos os neurônios da camada anterior. Essa camada é responsável por combinar os recursos extraídos pelas camadas anteriores em uma saída final, que pode ser usada para realizar a tarefa específica que a rede está tentando resolver.

Por exemplo, em uma CNN para classificação de imagem, as camadas convolucionais são responsáveis por extrair recursos relevantes da imagem, como bordas, texturas e padrões. Esses recursos são então passados para uma ou mais camadas totalmente conectadas, que os combina em uma saída final. É a partir destes dados combinados que o modelo avalia a probabilidade da imagem pertencer a cada classe de interesse.

As camadas totalmente conectadas podem ter um grande número de neurônios, o que pode levar a um grande número de parâmetros na rede. Isso pode levar a problemas de *overfitting*, onde a rede se ajusta demais aos dados de treinamento e não generaliza bem para novos dados. Para lidar com esse problema, muitas redes modernas usam técnicas como regularização, *dropout* e camadas de *pooling*, que podem reduzir o número de parâmetros e ajudar a evitar o *overfitting*.

Um exemplo de uma CNN com camadas totalmente conectadas é a VGG16, uma rede neural profunda para classificação de imagens desenvolvida pelo Visual Geometry Group da Universidade de Oxford. A *Very Deep Convolutional Networks* (VGG16) consiste em 13 camadas convolucionais e 3 camadas totalmente conectadas, e é conhecida por seu desempenho de ponta em vários conjuntos de dados de classificação de imagem.

2.3.1.4. Upsampling e Downsampling

Em redes neurais convolucionais (CNN), o *upsampling* é uma técnica usada para aumentar o tamanho de uma imagem ou mapa de características, que geralmente é reduzido usando uma camada de *pooling* ou convolução. O objetivo é restaurar a resolução espacial da imagem ou mapa de características para uma representação mais próxima da imagem original, antes de ser submetida a camadas posteriores da rede neural (Long, Shelhamer e Darrell, 2015).

Existem diferentes técnicas de *upsampling* que podem ser usadas em CNNs. A técnica mais comum é o *upsampling* bilinear, que utiliza interpolação linear para aumentar o tamanho da imagem (Ronneberger, Fischer e Brox, 2015). Nesta técnica, cada pixel da imagem original é interpolado em quatro pixels na imagem resultante, adicionando novos pixels com valores intermediários entre os pixels originais (Szegedy, 2016).

Outra técnica de *upsampling* comumente utilizada é o *upsampling* por transposição de convolução, também conhecida como deconvolução. Nessa técnica, uma camada de convolução transposta é usada para aumentar o tamanho da imagem, aplicando um filtro de convolução invertido que cria pixels adicionais e restaura a resolução espacial.

O *upsampling* é frequentemente utilizado em arquiteturas de redes neurais convolucionais que envolvem tarefas de segmentação de imagem, como a segmentação semântica, onde o objetivo é atribuir um rótulo a cada pixel da imagem. Nesses casos, o *upsampling* é usado para restaurar a resolução espacial da imagem, antes que a rede neural seja usada para classificar cada pixel em uma classe específica.

Já o *downsampling* refere-se à redução da resolução espacial da imagem ou dos recursos gerados pelas camadas anteriores da rede. O *downsampling* é frequentemente realizado por meio de camadas de *pooling*, como a camada *maxpooling* ou *averagepooling*, que reduzem o tamanho da imagem ou dos recursos ao selecionar o valor máximo ou médio de um grupo de pixels adjacentes.

A redução da resolução espacial por meio do *downsampling* ajuda a reduzir a dimensionalidade dos dados, tornando-os mais fáceis de processar pelas camadas posteriores da rede. Além disso, o *downsampling* ajuda a aumentar a eficiência computacional, permitindo que a rede processe imagens maiores e mais complexas com menos recursos computacionais.

O *downsampling* pode ser aplicado em diferentes pontos da rede, dependendo da arquitetura da CNN e da tarefa em questão. Em algumas arquiteturas, como a LeNet, o *downsampling* é aplicado logo após as primeiras camadas convolucionais para reduzir a dimensão dos recursos gerados. Em outras arquiteturas, como a ResNet, o *downsampling* é aplicado em camadas de bloco específicas que são responsáveis por reduzir a resolução espacial dos recursos.

No entanto, é importante notar que o *downsampling* também pode levar a perda de informações importantes, especialmente quando a redução da resolução espacial é muito agressiva. Isso pode levar a uma precisão reduzida em tarefas como classificação de objetos e segmentação de imagens. Portanto, em algumas arquiteturas, como a U-Net, a resolução espacial é aumentada por meio de camadas de *upsampling* após o *downsampling*, permitindo que a rede capture informações detalhadas em diferentes escalas.

2.3.1.5. Tipos de CNN

As redes neurais convolucionais (CNNs) usam uma arquitetura em camadas para extrair características de imagens. Conceitualmente, quanto mais camadas uma CNN tiver, melhor será seu desempenho. No entanto, nem sempre é possível aplicar uma CNN com muitas camadas a todas as tarefas, pois cada demanda possui características próprias que devem ser consideradas.

Nesse sentido, é importante conhecermos também os conceitos de CNNs profundas (*deep*) e móveis (*mobile*) como forma de entendimento do contexto geral da aplicação de uma CNN bem como para entendermos também o contexto geral da inovação apresentada em nossa pesquisa.

2.3.1.5.1. Redes neurais convolucionais profundas (deep)

Redes neurais convolucionais profundas são CNNs que possuem várias camadas convolucionais. Essas camadas adicionais permitem que as CNNs aprendam representações mais complexas das imagens, o que pode levar a um melhor desempenho em tarefas de Visão Computacional.

As CNNs profundas foram responsáveis por avanços significativos na área de Visão Computacional. Em 2012, o AlexNet, uma CNN profunda, venceu o ImageNet Large Scale Visual Recognition Challenge (ILSVRC) e nessa competição os pesquisadores apresentaram a inovação do paralelismo que é a utilização de múltiplas GPUs para o processamento de imagens. Essa inovação nos dá a dimensão do custo computacional envolvido quando tratamos de redes neurais profundas.

Desde o AlexNet, várias outras CNNs profundas foram desenvolvidas, incluindo a VGGNet, a GoogleNet, a ResNet e a DenseNet. Essas CNNs continuaram a melhorar o desempenho na classificação de imagens, chegando a acurácias de até 99%.

2.3.1.5.2. Redes neurais convolucionais para dispositivos móveis (mobile)

Redes neurais convolucionais móveis são uma variante das CNNs profundas que são projetadas para serem mais eficientes em termos de recursos. Essas CNNs são geralmente menores e usam menos parâmetros do que as CNNs profundas tradicionais, o que as torna mais adequadas para dispositivos móveis, como *smartphones* e *tablets*.

As CNNs móveis foram desenvolvidas em resposta ao crescente uso de dispositivos móveis. Esses dispositivos geralmente têm recursos limitados, como memória e

processamento, o que torna difícil executar CNNs profundas. As redes neurais convolucionais móveis oferecem uma solução para esse problema, permitindo que dispositivos móveis executem tarefas de Visão Computacional com eficiência.

Algumas das redes neurais convolucionais móveis mais populares incluem a MobileNet, a ShuffleNet e a SqueezeNet. Essas CNNs foram usadas em uma variedade de aplicações em dispositivos móveis, incluindo a fotografia, a detecção de objetos e a realidade aumentada.

2.3.1.5.3. Comparação entre redes neurais convolucionais profundas e móveis

As redes neurais convolucionais profundas e móveis têm vantagens e desvantagens distintas. As redes neurais convolucionais profundas geralmente têm melhor desempenho do que as redes neurais móveis, mas também são mais exigentes em termos de recursos. As redes neurais móveis são mais eficientes em termos de recursos, mas geralmente têm um desempenho inferior ao das redes neurais profundas.

A escolha entre redes neurais convolucionais profundas e móveis depende da aplicação específica. Para aplicações que requerem o melhor desempenho possível, as redes neurais profundas são a melhor escolha. Para aplicações que são limitadas em termos de recursos, as redes neurais móveis são uma boa opção.

2.4. ARQUITETURAS DE CNN

Nesta Seção, descrevemos as principais arquiteturas de rede neural convolucional que se destacaram por suas inovações em diversos aspectos. Por essa razão, esses modelos foram selecionados como referências comparativas para a arquitetura do nosso próprio modelo. Realizamos uma análise sucinta de cada arquitetura, focando nos aspectos mais relevantes que cada uma apresenta e as descrições estão organizadas em ordem cronológica de publicação.

2.4.1. GoogleNet (2014)

A GoogleNet é uma CNN proposta pela equipe do *Google Research* em 2014. Ela é conhecida por sua arquitetura inovadora, que utiliza módulos *Inception.* A arquitetura da GoogleNet consiste em 22 camadas de processamento, incluindo 9 módulos *Inception*, e sua

profundidade é ilustrada através da Figura 3. A rede começa com uma camada convolucional de tamanho 7x7 com *stride* 2, seguida de uma camada de *pooling* máxima de tamanho 3x3 e *stride* 2.

Os módulos *Inception* consistem em camadas de convolução e *pooling* executadas em paralelo e concatenadas para extrair recursos de diferentes tamanhos de janela de convolução. O primeiro módulo *Inception* possui quatro caminhos diferentes, incluindo convoluções de 1x1, 3x3 e 5x5, e uma camada de *pooling* seguida por convolução.

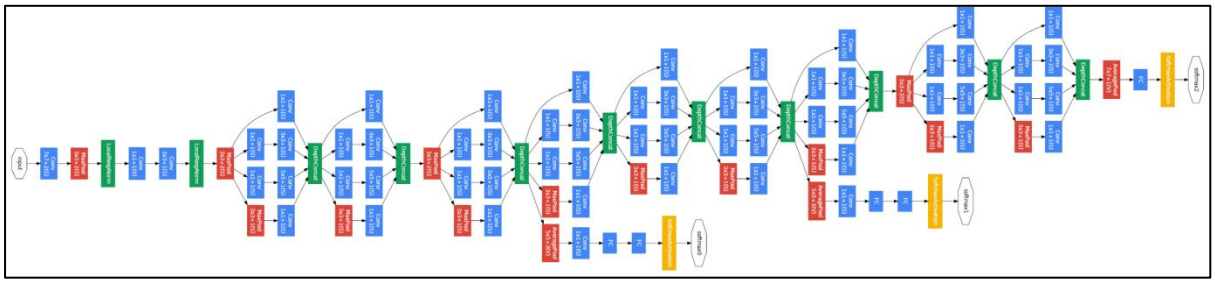


Figura 3: Ilustração da GoogleNet exemplificando uma arquitetura profunda (Fonte: https://medium.com/the-modern-scientist/exploring-googlenet-a-revolutionary-deep-learning-architecture-8bb176a0facc)

2.4.2. ResNet (2015)

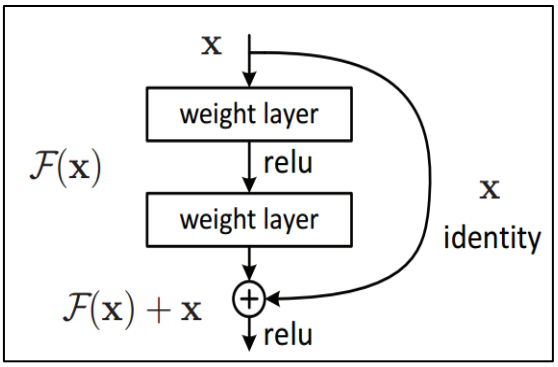


Figura 4: Exemplo de aplicação de camada residual em uma CNN (Fonte: https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4)

A ResNet, ou Rede Neural com Conexões Residuais, foi proposta por Kaiming He, Xiangyu Zhang, Shaoqing Ren e Jian Sun em 2015. A ResNet é conhecida por suas conexões residuais, ilustrado na Figura 4, fundamentais para resolver o problema de desaparecimento do gradiente (do inglês, "*vanish gradiente*") em redes profundas. A ResNet é caracterizada por um número variável de camadas. A ResNet-18, por exemplo, utiliza valores de *kernel* como 3x3 em diferentes blocos, oferecendo um equilíbrio entre eficiência e desempenho em reconhecimento facial e detecção de objetos.

2.4.3. SqueezeNet (2016)

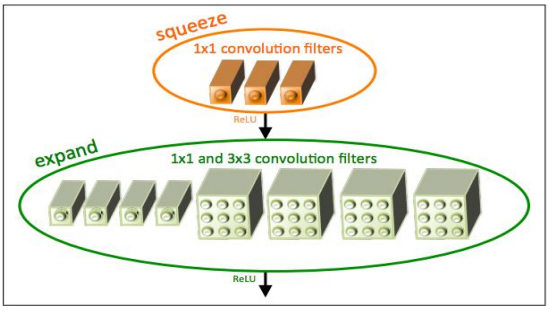


Figura 5: Arquitetura módulo FIRE da SqueezeNet
Fonte: https://paperswithcode.com/method/fire-module

A SqueezeNet, desenvolvida em 2016 pelo UC Berkeley's Berkeley Vision and Learning Center, é conhecida por sua eficiência em classificação de imagens em dispositivos com recursos limitados (Iandola et.al., 2016). Em sua arquitetura ela utiliza combinações de camadas em bloco denominadas de módulo *Fire* como ilustrado na Figura 5. Cada módulo *Fire* possui combinações de camadas de convolução em 1x1 e 3x3 que promovem a redução das dimensões da imagem.

2.4.4. DenseNet (2016)

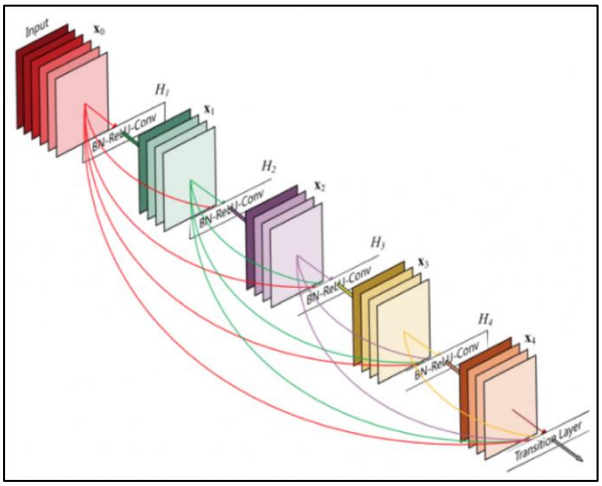


Figura 6: Descrição da arquitetura da DenseNet
Fonte:
https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a

A DenseNet, ou Rede Neural com Conexões Densas, é uma arquitetura profunda para classificação de imagens proposta por pesquisadores da Universidade de Tsinghua em 2016 (Huang, 2016). A Figura 6 ilustra a arquitetura, baseada no conceito de “conexões densas”, onde cada camada se conecta diretamente a todas as subsequentes em um bloco. Cada bloco tem duas ou mais camadas convolucionais com filtro fixo (geralmente 3x3), seguidas por normalização por lote e ativação ReLU.

2.4.5. MobileNet V1 (2017)

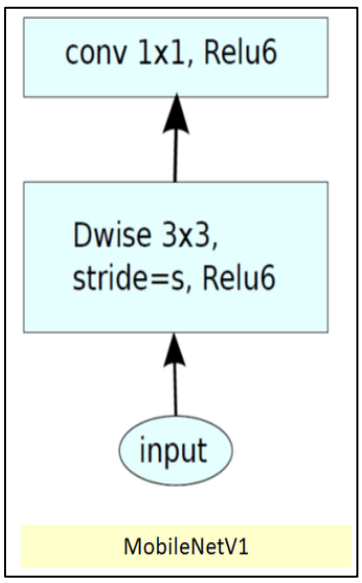


Figura 7: Descrição do bloco de convolução da MobileNet
Fonte: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c

A MobileNet V1 é uma rede neural convolucional projetada para dispositivos móveis, e desenvolvida pela equipe do *Google Research* em 2017 (Howard et.al., 2017). A principal característica da MobileNetV1 é o uso de operações de convolução separável em profundidade, como ilustrado na Figura 7, que são compostas por duas etapas: convolução em profundidade separada (*depthwise convolution*) e convolução em ponto final (*pointwise convolution*).

A convolução de profundidade separável é uma técnica que divide a convolução em duas etapas: primeiro, aplica-se convoluções separadas para cada canal de entrada, reduzindo o número de parâmetros. Em seguida, usa-se convoluções em ponto final para combinar as características aprendidas. Essa abordagem economiza recursos computacionais, tornando as redes mais leves sem comprometer o desempenho.

2.4.6. ShuffleNet V1 (2018)

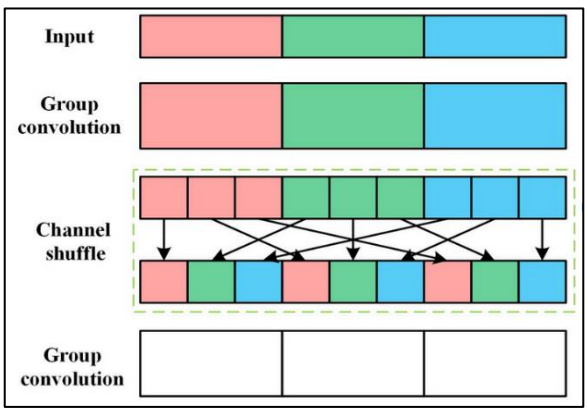


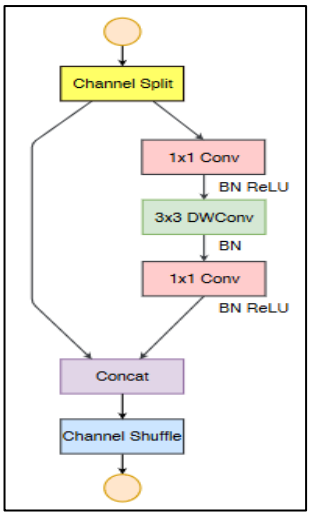
Figura 8: Funcionamento do *Channel shuffle*
Fonte: https://www.researchgate.net/figure/The-basic-principle-of-channel-shuffle_fig1_365399846

A ShuffleNet é uma arquitetura de CNN que foi projetada para dispositivos móveis (Zhang et.al., 2018). Ela apresenta uma técnica chamada "*channel shuffle*", que permite a combinação eficiente de informações de diferentes canais em camadas convolucionais.

A ShuffleNet é composta por várias unidades de bloco *shuffle unit*, cada uma contendo uma sequência de camadas convolucionais, seguida por uma operação de "*channel shuffle*" como ilustrado na Figura 8. Essa operação reorganiza os canais de saída de diferentes convoluções em uma nova matriz de canais, que é usada como entrada para a próxima camada convolucional. Isso permite a

fusão eficiente de informações de diferentes canais, o que pode melhorar o desempenho da rede sem aumentar significativamente o número de parâmetros.

2.4.7. ShuffleNet V2 (2018)



A ShuffleNet V2 é uma arquitetura de cnn projetada como uma evolução da versão 1 (Ma et.al., 2018). A principal novidade na arquitetura ShuffleNet V2 é a utilização de uma função denominada separador de canais. No início de cada unidade, o *channel split*, originária da primeira versão, divide os canais de entrada em dois ramos. Em um dos ramos, a entrada é mantida idêntica. No outro ramo, são realizadas três convoluções com o mesmo número de canais de entrada e saída para satisfazer uma condição específica como ilustrado na Figura 9.

Figura 9: Arquitetura do bloco de convolução da ShuffleNet V2. Fonte: https://paperswithcode.com/method/shufflenet-v2-block

2.4.8. Mobilenet V2 (2018)

A MobileNetV2 é uma CNN desenvolvida pela Google em 2018, como uma evolução da MobileNetV1 originalmente publicada no artigo "MobileNetV2: *Inverted residuals and linear bottleneck*s" (Sandler, 2018). Sua arquitetura é baseada em blocos residuais com *skip connection*s, como na MobileNetV1, mas com algumas modificações para melhorar o desempenho.

A principal diferença é o uso de um bloco chamado *inverted residuals and linear bottleneck*, que consiste em três camadas consecutivas: uma convolução 1x1, seguida de uma convolução 3x3 com dilatação, isto é, com lacunas entre os pixels, e outra convolução 1x1. Essas camadas são seguidas por uma conexão residual que junta à entrada com a saída do bloco, como ilustrado na Figura 10.

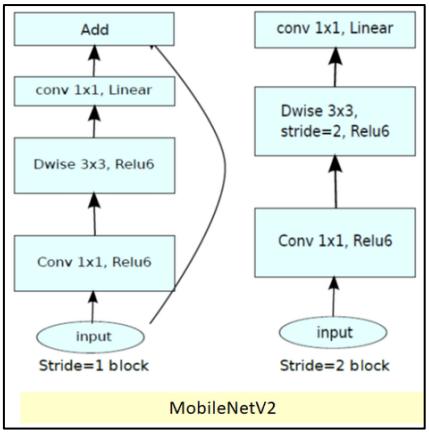


Figura 10: Descrição dos blocos de convolução da MobileNet V2
Fonte: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c .

2.4.9. Mobilenet V3 (2019)

A MobileNetV3 é uma arquitetura de CNN desenvolvida pela Google (Howard, 2019) que possui duas inovações principais. A primeira, a técnica de pesquisa de arquitetura neural, do inglês *neural architecture search* (NAS), que permite otimizar automaticamente a arquitetura da rede para diferentes requisitos de aplicação. E a segunda é o algoritmo de NetAdapt ilustrado na Figura 11 que permite que a arquitetura da MobileNetV3 seja personalizada e otimizada para atender às necessidades específicas de diferentes cenários de aplicação, resultando em uma rede mais eficiente e eficaz.

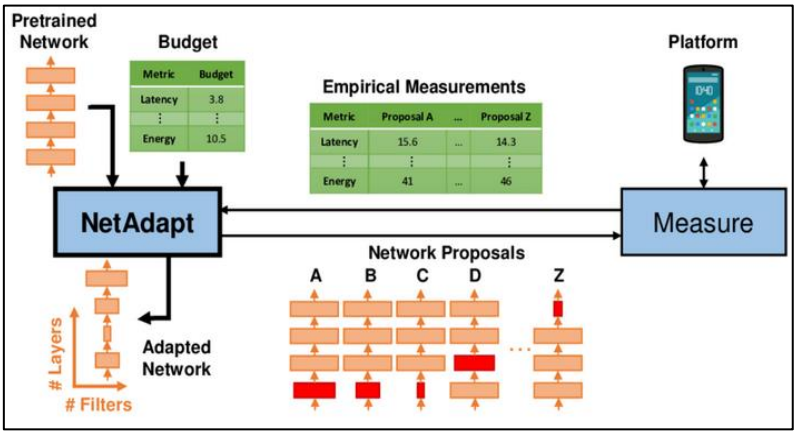


Figura 11: Descrição da aplicação NetAdapt da arquitetura da MobileNet V3
Fonte: https://www.researchgate.net/figure/NetAdapt-automatically-adapts-a-pretrained-network-to-a-mobile-platform-given-a-resource_fig1_324435700

2.4.10.Efficientnet V1 (2019)

A EfficientNet foi criada em 2019 pela equipe do Google (Tan e Le, 2019). Ela utiliza o conceito de escalabilidade composta para equilibrar eficientemente a profundidade da rede, a largura dos canais e a resolução da imagem de entrada, como ilustrado na Figura 12. Em vez de aumentar apenas uma dimensão da rede, como profundidade ou largura dos canais, busca-se encontrar o conjunto ótimo desses fatores para maximizar o desempenho e a eficiência computacional.

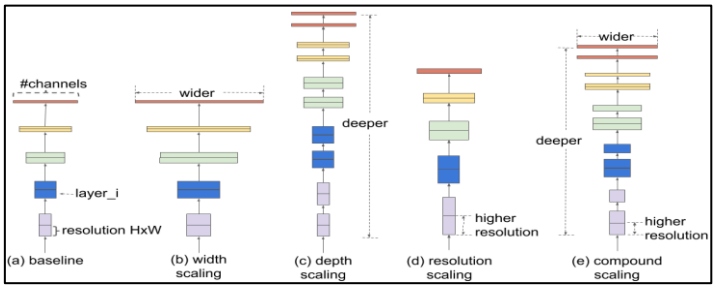


Figura 12: Descrição da escalabilidade composta proposta por Tan e Le na arquitetura da EfficientNet (Fonte: https://paperswithcode.com/method/efficientnet)

2.4.11.Efficientnet V2 (2021)

A EfficientNet V2 é uma versão aprimorada da arquitetura EfficientNet original (Tan e Le, 2021)**.** Ela traz como inovação a estratégia de escalabilidade composta, combinando elementos de escalabilidade uniforme e direcionada. Enquanto a escalabilidade uniforme ajusta todos os componentes da rede proporcionalmente, garantindo um equilíbrio entre as camadas, a escalabilidade direcionada permite um aumento seletivo na profundidade, largura ou resolução da rede para tarefas específicas, como a detecção de objetos em imagens de alta resolução.

2.5. CONSIDERAÇÕES FINAIS

Neste capítulo, os principais pontos que compõem a arquitetura de uma CNN foram examinados em detalhes, incluindo elementos fundamentais como fase de convolução, fase densa, *kernel*, *stride*, convolução, entre outros. Ao entender profundamente esses componentes, adquirimos o conhecimento necessário para compreender o funcionamento interno das CNNs e suas aplicações em diversas áreas.

Apresentamos também algumas arquiteturas de CNN que são reconhecidas cientificamente pelas suas contribuições para a área do aprendizado profundo. Analisamos suas estruturas e ressaltamos os pontos principais que representam as inovações que cada modelo possui.

Essas inovações representam o que há de melhor nas arquiteturas das CNNs e nos ajudou a elaborar a nossa proposta que será apresentada posteriormente. Nosso objetivo é criar um modelo que, além de alcançar uma alta acurácia, característica predominante em redes profundas, também ofereça um desempenho eficiente em termos de *hardware*, atributo essencial para modelos voltados a dispositivos móveis.

No próximo capítulo, exploraremos os trabalhos relacionados, mergulhando na literatura existente para entender as abordagens, técnicas e avanços mais recentes no campo das CNNs e áreas afins. Essa análise nos permitirá posicionar nosso trabalho dentro do contexto atual da pesquisa, identificar lacunas de conhecimento e destacar a contribuição única de nossa proposta para o avanço do campo.

CAPÍTULO 3.

TRABALHOS RELACIONADOS

Neste capítulo apresentamos trabalhos relacionados à aplicação de Redes Neurais Convolucionais, que, pela sua contagem reduzida de parâmetros treináveis, podem ser consideradas como redes móvel, aplicadas na tarefa de identificação de imagens com diversas resoluções. Os estudos relacionados nesse capítulo exploram o uso de CNNs na detecção de imagens de plantas e frutas, contexto que se assemelha ao nosso trabalho que realizamos com a planta do Sisal nesta pesquisa.

3.1. CNNS MOBILE NA CLASSIFICAÇÃO DE FRUTAS E HORTALIÇAS

O estudo realizado pelos autores Abu-Saqer, Abu-Naser e AlShawwa (2020) sobre a aplicação prática do uso de redes neurais convolucionais (CNNs) para classificar diferentes tipos de *grapefruit*. A escolha de usar CNNs se justifica pelo fato de que essas redes são capazes de identificar e extrair características relevantes nas imagens que estão sendo classificadas. O modelo proposto continha apenas quatro camadas de convolução acompanhadas de quatro camadas de *maxpooling* e duas camadas lineares na fase densa.

No processo de pré-processamento, as imagens foram normalizadas, ajustadas ao tamanho de 256x256 pixels e convertidas para escala de cinza. Além disso, foram realizadas outras técnicas de pré-processamento, como remoção de ruído e aumento do conjunto de dados por meio de técnicas de aumento de dados.

Os resultados mostraram que a CNN treinada foi capaz de classificar os três tipos de *grapefruit* com alta precisão, alcançando uma acurácia de 100%. Isso indica que a abordagem proposta pode ser útil em aplicações práticas de classificação automatizada dessa fruta na indústrias de alimentos e na seleção de amostras no plantio das grandes fazendas, por exemplo.

Em Al-Daour, Al-Shawwa e Abu-Naser (2020) os autores descrevem um sistema de classificação de bananas usando aprendizado profundo. O objetivo do sistema é classificar as bananas em três categorias: banana comum, banana *lady finger* e *red* banana. As imagens usadas no treinamento da rede foram obtidas de um banco de imagens de bananas de diferentes variedades e estados de maturação. O banco de dados utilizado é formado por

8.554 imagens divididas em *dataset*s de treino e validação. As imagens possuem a resolução de 128x128.

O modelo utilizado possui quatro camadas de convolução acompanhadas de outras quatro camadas de *maxpooling* e duas camadas lineares na fase densa. Os resultados mostraram que a CNN foi capaz de classificar as bananas em suas categorias corretas com uma taxa de acerto de 100%. Essa precisão alta sugere que o sistema proposto pode ser útil para os produtores de bananas em todo o mundo, ajudando a melhorar a qualidade e a eficiência da produção.

Em AlShawwa et.al. (2020), os autores também tratam da tarefa de classificação de frutas. Nesse artigo os autores utilizam uma rede neural convolucional para realizar a tarefa de identificação de seis tipos diferentes de cereja.

Em Al-Shawwa (2020), o autor apresenta uma abordagem para classificar as maçãs em treze tipos diferentes usando técnicas de aprendizado profundo. O conjunto de dados utilizado contendo imagens de 8.554 possuía uma divisão de 1.928 imagens no conjunto de validação e 2.138 imagens no conjunto de teste. E as imagens estavam configuradas na resolução de 150x150 pixels e não receberam nenhum tipo de tratamento. Os resultados mostraram que o modelo alcançou uma precisão de 100%.

Em Mettleq et.al. (2020) apresentam uma abordagem de classificação de dois tipos de manga usando um conjunto de dados com cerca de 1.200 imagens. Os autores apresentam a metodologia utilizada, incluindo detalhes sobre o conjunto de dados utilizado, a arquitetura do modelo CNN e a camada totalmente conectada utilizada para classificação. O modelo proposto contém quatro camadas de convolução acompanhadas de quatro camadas de *maxpooling* e duas camadas lineares na fase densa.

O objetivo do artigo foi desenvolver um modelo de classificação de manga que pudesse ajudar as pessoas a identificar corretamente o tipo de manga com base em suas características visuais, como tamanho, forma, cor da casca e da polpa. O modelo proposto alcançou uma acurácia de 100% no conjunto de teste, o que demonstrou a viabilidade dessa abordagem e o resultado alcançado com essa abordagem pode ser aplicado em automação de classificação de manga, o que poderia ajudar produtores e vendedores a categorizar seus produtos de forma mais precisa e eficiente.

3.2. CNNS MOBILE NA CLASSIFICAÇÃO DE DOENÇAS EM PLANTAS

Kumar et.al. (2021) aborda a detecção precoce de doenças em plantas e como isso pode ser importante para melhorar a qualidade e o rendimento das colheitas. Para isso, os autores propõem um método baseado em aprendizado profundo onde eles aplicam três arquiteturas principais de rede neural: *Faster Region-based Convolution Neural Network (Faster R-CNN), Region-based Fully CNN (R-CNN) e Single Shot Multibook Detector (SSD)* e utilizam o resultado do algoritmo mais eficiente como referência.

Os autores usaram *dataset* Plant Village, que contém mais de 50.000 imagens de folhas de plantas saudáveis e doentes, divididas em 38 categorias por espécie e doença e técnicas de *augmentation* para aumentar a variedade de dados no conjunto de treinamento e eles alcançaram uma precisão de 94,6% na detecção de diferentes tipos de doenças em plantas, o que indica a viabilidade de soluções de aprendizado profundo para esse problema complexo.

Em Hassan et al. (2021), os autores exploram a identificação e diagnóstico de doenças em plantas a partir de suas folhas, empregando redes neurais convolucionais e uma abordagem de transferência de aprendizado. Enquanto os modelos de CNN convencionais geralmente requerem um grande número de parâmetros e um custo computacional elevado, os autores adotam a convolução *depth-separable* para reduzir tanto o número de parâmetros quanto o custo computacional.

Os autores utilizaram implementações já conhecidas como modelos para a pesquisa e alcançaram taxas de precisão na classificação de doenças de 98,42%, 99,11%, 97,02% e 99,56% usando InceptionV3, InceptionResNetV2, MobileNetV2 e EfficientNetB0, respectivamente. Segundo os autores, e embora não tenha sido o melhor resultado, eles consideraram o resultado da MobileNetV2 satisfatório, pois agregou um bom desempenho e por ter exigido menos tempo de treinamento.

3.3. CNN NA CLASSIFICAÇÃO DE PLANTAS FIBROSAS

Montañez e Barrera (2019) apresentam um estudo sobre a classificação automatizada de fibras de abacá utilizando uma rede neural convolucional. O estudo proposto pelos autores visa justamente explorar o uso de redes neurais convolucionais na classificação de fibras de abacá, um material de grande importância na indústria têxtil. Para tanto, os autores criaram uma base de dados de imagens de fibras de abacá em diferentes graus de qualidade, e treinaram a rede neural convolucional para classificar essas imagens de forma automatizada.

Um dos pontos interessantes do artigo é a comparação dos resultados obtidos com a utilização da rede neural convolucional proposta com outras técnicas de classificação de fibras de abacá. Os autores compararam os resultados da CNN com a análise de imagem baseada em regras e a análise de componentes principais (PCA), técnicas tradicionalmente utilizadas na classificação de fibras de abacá.

Outro ponto interessante do artigo é a discussão sobre a escolha dos hiperparâmetros da rede neural convolucional. Os autores discutem a importância da escolha correta dos hiperparâmetros, como o tamanho do filtro, o número de camadas e a taxa de aprendizado, para garantir a eficácia da rede neural convolucional na classificação de fibras de abacá.

Os resultados mostraram que a rede neural convolucional proposta obteve uma precisão de classificação de 96,7%, o que indica que a abordagem proposta pode ser eficaz na classificação automatizada de fibras de abacá. Além disso, o estudo mostrou que a CNN proposta apresentou uma melhoria significativa em relação a outras técnicas de classificação de fibras de abacá, como a análise de imagem baseada em regras e a análise de componentes principais.

3.4. PONTOS IMPORTANTES SOBRE OS TRABALHOS RELACIONADOS

Os trabalhos relacionados nessa Seção apresentam pesquisadores empregando CNNs em desafios de classificação de imagens de diversas origens. Cada uma delas com suas respectivas características e que demandaram estruturas de CNNs diferentes em quase todos os artigos analisados. Entretanto, podemos identificar alguns pontos que são relevantes e que merecem ser ressaltados nesse momento.

- O primeiro deles é a diversidade do domínio de aplicação das CNNs móveis, pois apresentamos exemplos de imagens diversas com graus de dificuldade de classificação igualmente diverso. Essa característica demonstra alguns cenários que viabilizam a realização de tarefas de alta complexidade mesmo utilizando modelos de pequeno porte. E isso é uma das qualidades que buscamos com nossa pesquisa ao apresentarmos um algoritmo que consegue aliar qualidade na precisão e eficiência no desempenho de hardware a partir da capacidade de se ajustar através da resolução da imagem para o qual é apresentado.
- O segundo ponto faz referência ao fato de todas as pesquisas apresentarem analises que foram baseadas em imagens com resoluções superiores à, no

mínimo, 150x150. Esse ponto é importante, pois demonstra que mesmo sendo algoritmos móveis as pesquisas demandaram imagens com resoluções suficientemente grandes para que se pudesse obter a acurácia desejada. Esse aspecto se alinha com o questionamento (Seção 1.2 Questão 2) que fazemos em nosso trabalho se seria possível desenvolver uma arquitetura que possa ser livre desse contexto. E que permita ao modelo obter uma boa acurácia mesmo sendo considerada como uma arquitetura do tipo móvel.

- Um terceiro ponto relevante nessa nossa análise faz referência à arquitetura utilizada pelos autores. De alguma maneira, todos os modelos possuem em comum o fato de utilizarem estratégias de *down sampling* em sua arquitetura. Dentre as estratégias utilizadas temos as camadas de *pooling*.

Ao contrário da maioria dos trabalhos existentes, em nossa pesquisa, apresentamos uma nova camada que se alinha à categoria de *down sampling* e que pode ser utilizada para reduzir o número de parâmetros treináveis da rede. Essa camada apresenta algumas vantagens em relação às camadas de *pooling* que precisam ser bem planejadas para evitar prejuízos à acurácia do modelo.

Adicionalmente aos pontos comentados, temos também o fato de alguns dos pesquisadores citados aplicarem o uso de CNNs na identificação de doenças em plantas e na identificação de frutas. Esse aspecto demonstra a preocupação com a dinâmica econômica e busca por possíveis aplicações práticas para esse tipo de algoritmo. Essa é outra característica que pretendemos alcançar com nossa pesquisa também através da análise de imagens da planta do sisal.

3.5. CONSIDERAÇÕES FINAIS

Neste capítulo, apresentamos diversos algoritmos de CNNs que pela sua arquitetura podem ser consideradas como redes móveis e que são aplicadas na análise de imagens de plantas em diversas situações. Examinando como essas estruturas se encaixam no contexto de nossa pesquisa e mais precisamente da nossa inovação, a camada Defiber.

Considerando que muitos desses modelos apresentam, ao menos, uma camada de pooling em sua arquitetura, julgamos natural avaliarmos a aplicação dessas camadas em comparação com nossa proposta, a fim de examinar a qualidade desses algoritmos. Nosso objetivo é demonstrar, por meio de nossa abordagem, a superior eficiência da camada Defiber na tarefa de reduzir a resolução do *feature map* durante a fase de convolução.

Além disso, ao analisar os diferentes modelos, pudemos identificar suas vantagens e limitações, nos fornecendo insights valiosos para o desenvolvimento e aprimoramento de nossa proposta, visando assim a criação de uma solução ainda mais eficiente para problemas de Visão Computacional em dispositivos móveis.

A seguir, apresentaremos nossa proposta de arquitetura de CNN, desenvolvida com todos os elementos essenciais para um modelo móvel eficiente. Com base nas considerações discutidas até o momento, incorporamos uma camada inovadora que, aliada a uma configuração precisa de hiperparâmetros, possibilita uma significativa redução na quantidade de parâmetros do modelo.

CAPÍTULO 4.

O MODELO PROPOSTO

No presente capítulo, abordamos os elementos essenciais para a construção de um protótipo base, que serviu como arcabouço para nossa inovação. Esse protótipo emergiu após uma análise abrangente que explorou os conceitos fundamentais das Redes Neurais Convolucionais (CNNs) e suas características principais, além de examinar os modelos tidos como estado da arte. Nossa abordagem representa a materialização da proposta subjacente ao nosso algoritmo, que busca equilibrar a acurácia desejada com a redução do número de parâmetros treináveis. Além disso, neste capítulo, descreveremos em detalhes a camada Defiber, o componente central de nossa proposta de inovação.

4.1. A ARQUITETURA BASE DO MODELO PROPOSTO

Com base nos princípios essenciais das CNNs, a proposta emprega uma configuração mínima tradicionalmente reconhecida por sua eficácia, enquanto também introduz inovações que serão descritas mais adiante. O objetivo central deste esforço é criar um modelo viável economicamente, sem comprometer sua capacidade de alcançar resultados precisos e confiáveis.

Assim sendo, iniciamos, portanto, a apresentação da estrutura do nosso algoritmo que reproduz a arquitetura fundamental de uma CNN utilizando apenas os requisitos mínimos de uma CNN como ilustrado na Figura 13. A fase de convolução da rede é representada por uma camada de convolução, originalmente projetada para processamento de imagens coloridas (Conv2D), e os valores dos hiperparâmetros definidos para cada camada serão detalhadas na Tabela 1 da Seção 5.3 desta tese.

Além disso, como parte integrante da arquitetura base do nosso algoritmo, nós utilizamos o recurso do *BatchNorm* (*batch normalization*) para melhorar a velocidade de aprendizado e evitar *overfitting* no treinamento. Nós utilizamos também a função de ativação ReLU nas camadas de convolução e na parte densa da rede. Essa parte é formada por uma única camada linear de 1.000 neurônios. Esses recursos foram acrescentados por fazerem parte da arquitetura base de uma CNN, sendo recursos amplamente utilizados e até obrigatórios como no caso da função de ativação.

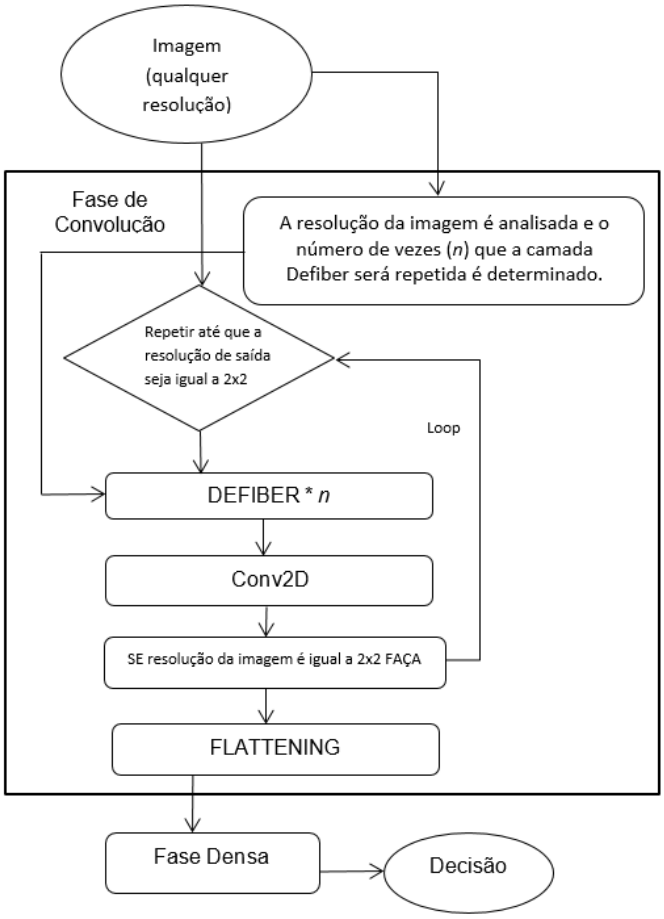


Figura 13: Fluxograma do modelo proposto (figura do autor)

Como inovação, nosso algoritmo conta com uma nova camada denominada Defiber. Essa camada é aplicada antes de cada camada de convolução e ela funciona como um redutor do *input* antes de todas as camadas de convolução. E a camada Defiber pode ser executada uma ou múltiplas vezes antes de cada camada de convolução.

Quem determina a quantidade de vezes que as camadas Defiber são repetidas é um algoritmo específico que toma como base a resolução da imagem de entrada. Esse comportamento das camadas Defiber associado às configurações das camadas de convolução é o que permite que o modelo proposto consiga uma boa acurácia mesmo com um número de parâmetros treináveis baixo.

Analisando a arquitetura do modelo proposto, a imagem de entrada segue para a primeira camada Defiber e também segue para a função que determina quantas vezes todas as camadas Defiber serão repetidas. Essa função analisa a resolução da imagem e define quantas

vezes cada camada Defiber será repetida ao longo do processo de convolução até que o tamanho da imagem seja compatível com a resolução de 2x2.

A arquitetura proposta consiste na aplicação de uma camada de convolução após cada camada Defiber. O valor do *kernel* da camada de convolução pode ser ajustado para otimizar a extração de características da imagem para a tarefa desejada. No entanto, é importante realizar uma análise prévia da configuração desse hiperparâmetro para evitar perda de desempenho ou *overfitting*.

Utilizamos essa análise para garantir que a configuração do *kernel* nas camadas de convolução esteja alinhada com o controle de redução do *input* no fluxo de convolução. Esse controle é necessário porque não estamos considerando a utilização de camadas residuais nessa arquitetura para seguirmos em conformidade com o estabelecido em nossos objetivos. Esse fato impede que nossa rede alcance grande profundidade, porém possibilita que nossa rede seja mais rápida no quesito de processamento.

Relembrando o que vimos na Seção 2.4, sobre arquiteturas, quando descrevemos a arquitetura da ResNet e da DenseNet, camadas residuais são uma técnica usada para evitar a saturação e a explosão de gradientes em redes profundas. A ideia é adicionar uma camada adicional à rede que simplesmente adiciona a entrada à saída da camada anterior. Essa camada adicional é chamada de camada residual.

Um ponto importante da nossa pesquisa é a utilização de composições de *kernel* com valores menores nas primeiras camadas e valores maiores nas camadas finais. Essa configuração permite que o modelo proposto aprenda representações mais básicas e locais das imagens nas primeiras camadas, enquanto aprende representações mais globais e complexas nas camadas finais (Simonyan e Zisserman, 2014).

Entretanto, é importante ressaltar que é plenamente possível a utilização de nossa camada Defiber com qualquer outra combinação de *kernel* e até mesmo com a utilização de modelos com blocos de convolução[1] e camadas residuais. Assim como também é possível à aplicação da camada Defiber em arquiteturas que possuam muita largura, isto é, que possuam muitos filtros de convolução por camada.

[1] **Blocos de convolução**: são unidades de processamento compostas por múltiplas camadas convolucionais. Eles são usados para agrupar funções relacionadas de convolução em uma única unidade, o que pode ajudar a melhorar a eficiência do treinamento e da inferência das CNNs.

Adicionalmente a essa informação, lembramos que a função de probabilidade deve ser configurada com as informações de quantas camadas Defiber e de convolução o modelo terá, independentemente da quantidade de camadas utilizadas para implementá-lo. Essas informações são necessárias para que a função de probabilidade defina quantas vezes cada camada Defiber deve ser repetida, ou se precisam ser repetidas, a partir da configuração atual do modelo desejado.

Como um dos objetivos é obtermos ganhos de desempenho a partir da redução da quantidade de parâmetros treináveis, nossa proposta oferece o suporte necessário para que o modelo, a partir de qualquer arquitetura de convolução, possa reduzir, gradualmente, a resolução da imagem entre camadas, permitindo que o modelo mantenha sua precisão e ainda assim obter ganhos de desempenho de *hardware*.

4.2. A CAMADA DEFIBER

A camada Defiber, cujo algoritmo é ilustrado na Figura 14, consiste em um algoritmo simples que tem como principal objetivo reduzir o *input* (uma matriz multidimensional construída a partir de informações da imagem) a partir do valor dos índices dos eixos que o compõem.

O fluxo principal do modelo proposto começa com a imagem, convertida em um tensor de dados, seguindo para a primeira camada Defiber (DEFIBER * *n*) e o tensor retornado (*output*) por ela segue para a primeira camada de convolução. No contexto do aprendizado profundo, o tensor é uma estrutura de dados que pode representar dados com qualquer dimensão. No caso de imagens, um tensor é frequentemente usado para representar os dados de uma imagem como uma matriz de múltiplas dimensões.

Esse procedimento se repete até que na última camada de convolução o *output* resultante possua a resolução de 2x2. O tensor resultante da última camada de convolução é, então, achatado (*flattening*) e encaminhado para a camada linear da fase densa.

O algoritmo da camada Defiber recebe uma variável de repetição (*NumRep*) e uma estrutura de dados do tipo *tensor* (X). Esses dados passam por dois *loops* aninhados. O primeiro *loop* executa a quantidade de vezes que a camada Defiber precisa ser repetida (*NumRep*). A maneira como obtemos o valor correspondente à variável *NumRep* será detalhada no próximo capítulo.

O segundo *loop* é executada com valores entre zero e o tamanho da última dimensão de (X). A partir dos valores iterados, se a variável de controle (J) não for divisível pelo valor

do limite (L), o valor de (J) é armazenado em *NewList*. Ao término do segundo *loop*, o conteúdo dessa lista é utilizado para redimensionar (T) sendo este retornado pela função. No próximo capítulo, que versa sobre a metodologia, nós detalhamos todo o processo para chegarmos até o valor estabelecido como limite (L) na camada Defiber.

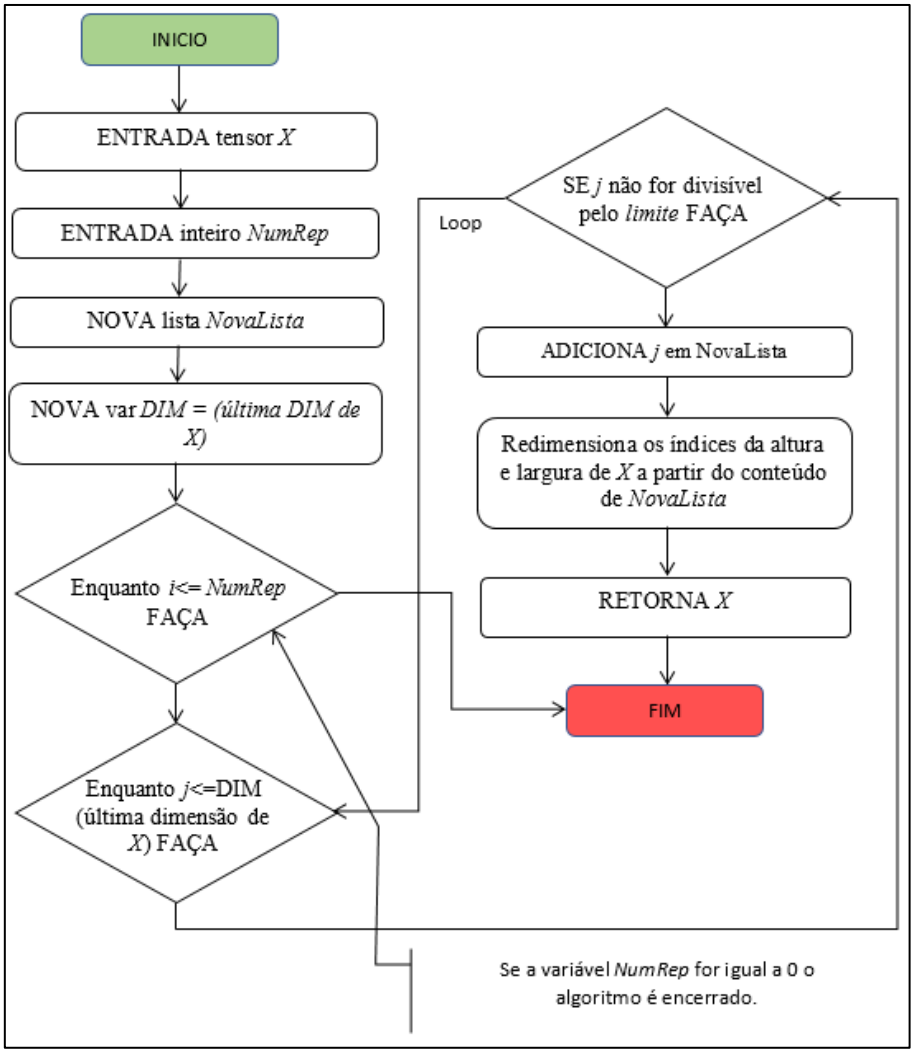


Figura 14: Fluxograma da camada Defiber (figura do autor)

Na Figura 15, nós apresentamos uma ilustração do funcionamento da camada Defiber. Dada uma imagem com o tamanho de 3x50x50, o valor 3 representa a quantidade de canais da imagem, isto é, é uma imagem colorida, ela possui 50 eixos de altura por 50 eixos de largura que começa do índice 0 e vai até o índice 49.

Dessa forma, o algoritmo da camada Defiber analisa os valores destes cabeçalhos e remove os eixos cujos índices forem divisores por três e é dessa maneira que conseguimos produzir um *output* menor antes que a imagem passe por cada camada de convolução.

O processo, relativamente simples, utiliza os índices (*index*) do tensor como parâmetros para remoção de linhas inteiras na imagem. Esses índices são removidos tomando como base um limite. Os valores dos índices que são divisíveis por esse limite são removidos da imagem original resultando em uma imagem menor redimensionada.

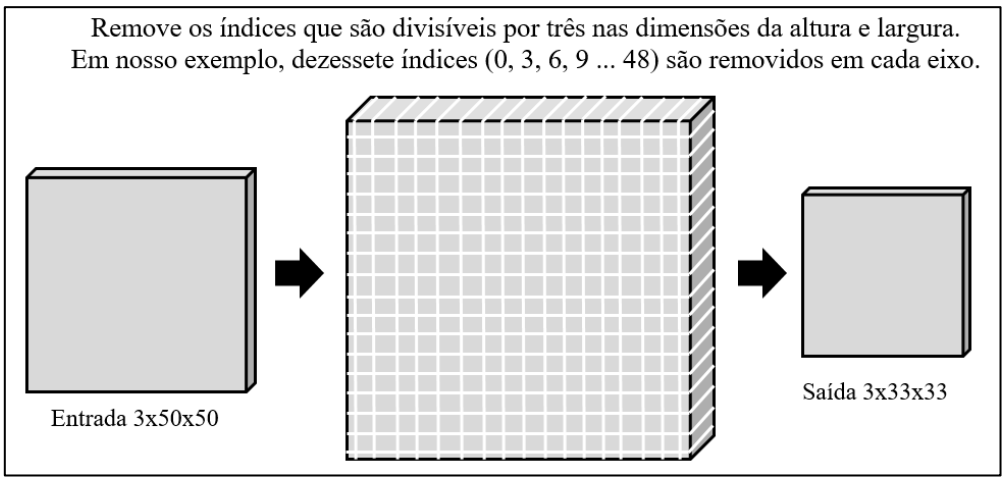


Figura 15: Ilustração do processo de redução da imagem ao passar pela camada Defiber (figura do autor)

A partir do redimensionamento dos índices do tensor original obtemos um novo tensor menor, que segue o fluxo principal da fase de convolução. É importante destacar que esse procedimento pode ser realizado em tensores com resoluções variadas, incluindo imagens de alta resolução, como 1080x1080 ou superior.

Podemos também a partir desta definição de como é executado o algoritmo da camada Defiber, explicar também na prática como uma matriz multidimensional é processada por ela. A partir de um algoritmo em linguagem Python podemos observar como o processo de redimensionamento da matriz se realiza conforme ilustrado na Figura 16.

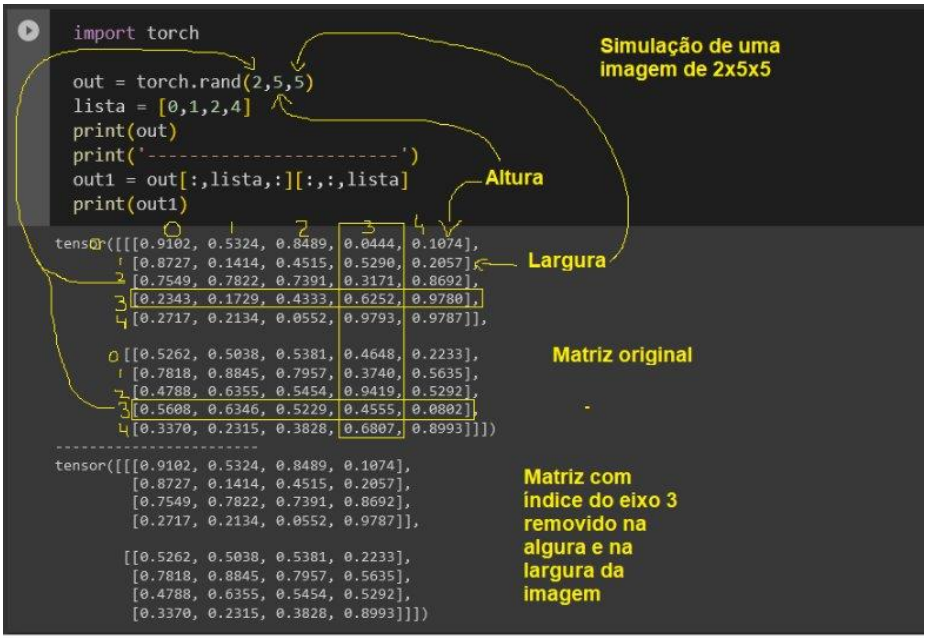


Figura 16: Ilustração do processo de redimensionamento da matriz a partir do código em linguagem Python

Observamos a partir da ilustração que o procedimento afeta a matriz multidimensional nas duas dimensões, como mencionado anteriormente, e também na quantidade de canais que a mesma possua. A figura na dimensão (2,5,5) significa que temos uma imagem composta de 2 canais de entrada na dimensão de 5x5 de altura e largura. E após a execução da camada Defiber ela passou a ter uma dimensão de (2,4,4).

Esse procedimento pode ser realizado em imagens de diversas resoluções. Durante nossa fase de análise empírica nós conseguimos obter redimensionamento de imagens de alta resolução como pode ser observado na ilustração da Figura 17. Nessa figura demonstramos a sumarização do modelo a partir do processamento de uma imagem com a resolução de 1080x1080.

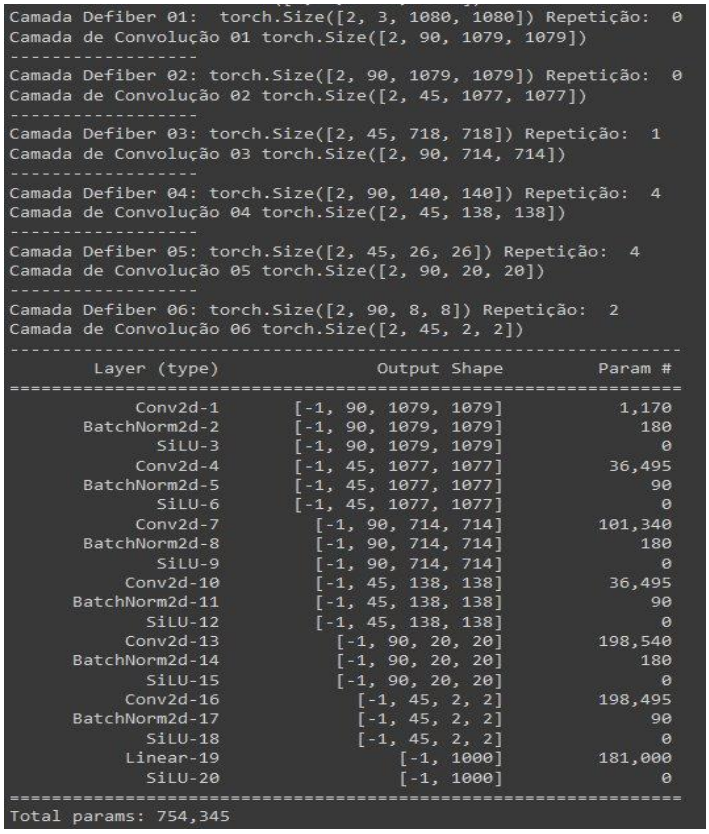


Figura 17: Sumário do modelo ao processar uma imagem com a resolução de 1080x1080

Neste exemplo, podemos observar também que, mesmo a partir de uma imagem de alta definição, nosso modelo conseguiu manter a mesma quantidade de parâmetros treináveis e as características de um modelo do tipo móvel.

4.3. A FUNÇÃO DE PROBABILIDADE

Para controlar dinamicamente a execução da camada Defiber nós definimos uma função de probabilidade. Ela é a responsável por determinar se ou quantas vezes cada camada Defiber será executada. Para isso, definimos um algoritmo que determina, a partir da resolução da imagem de entrada, um valor de repetição para cada camada Defiber definida no modelo.

A função de probabilidade possui em seu algoritmo uma cópia do fluxo principal da fase de convolução do modelo. Todas as camadas Defiber e de convolução existentes no fluxo principal são representadas nesta função por uma cópia que simula a função das camadas originais, porém com um retorno de valor diferente.

As camadas Defiber e de convolução do fluxo original do modelo processam um tensor multidimensional que representam os canais de uma imagem ou do *feature map* produzido. Enquanto que nas suas versões simuladas essas camadas processam um valor que representa a resolução da imagem e que será decomposta durante o fluxo principal.

Esse processamento ocorre para que a função de probabilidade consiga, a partir da resolução da imagem, deduzir se as camadas Defiber precisarão ser acionadas ou quantas vezes cada camada precisará funcionar para que a resolução final da imagem, no fluxo principal, seja um tensor na dimensão de 2x2.

Explicando o algoritmo, uma instância da imagem segue para a função de probabilidade onde sua resolução é capturada. A partir dessa informação o algoritmo utiliza o valor da última dimensão (*X*) como referência para calcular, camada por camada, a quantidade da redução da *feature map*.

São definidos também valores em uma *LISTA* a partir do qual cada camada Defiber poderá ser configurada para simular o processamento da *feature map*. Os valores da *LISTA* são percorridos a partir de estruturas de repetição (*loops*) em quantidade igual à quantidade de camadas Defiber do modelo. Cada *loop* contém uma variável (*A1, A2, A3, ... Ai*) que são preenchidas pelos valores da *LISTA*.

No *loop* mais interno é onde estão contidos as camadas que simulam a função das camadas Defiber e de convolução (Conv2D) originais. A diferença entre elas é que as camadas simuladas processam um valor inteiro que representa a resolução da imagem e, ao final do fluxo simulado, esse valor deverá ser igual à resolução da *feature map* ao final da fase de convolução.

Ao final do processo, se o valor resultante for igual a 2 (simulando um tensor de dimensões de 2x2), os valores utilizados para a repetição das camadas Defiber (*A1, A2, A3, ..., Ai*) serão retornados para o fluxo original conforme ilustrado na Figura 18.

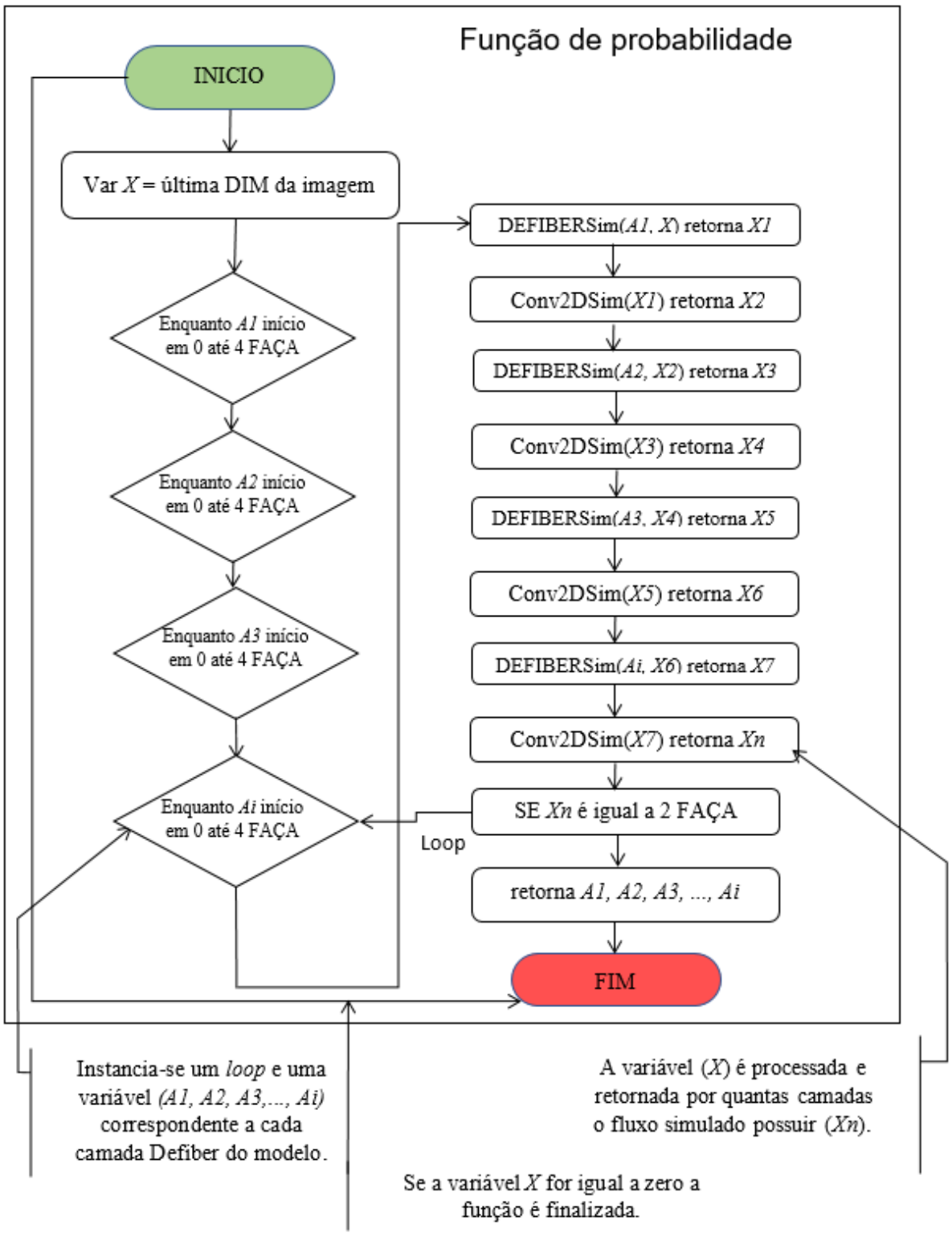


Figura 18: Fluxograma da Função de probabilidade (figura do autor).

Após a função de probabilidade devolver os valores que serão utilizados para determinar quantas vezes cada camada Defiber será repetida, o fluxo de convolução segue seu processamento natural. Os dados gerados pela rede durante a fase de convolução passam

pelo processo de achatamento (*flattening*) e são direcionados para a fase densa seguindo o fluxo tradicionalmente descrito em qualquer arquitetura de CNN.

Sendo este um procedimento natural para qualquer algoritmo de CNN podemos afirmar, portanto, que a proposta da camada Defiber pode ser utilizada em qualquer arquitetura, pois a função de probabilidade se encarrega de adaptar a aplicação da camada Defiber ao contexto de processamento da imagem desejado.

4.4. CONSIDERAÇÕES FINAIS

Apresentamos neste Capítulo uma arquitetura inovadora que conta com a camada Defiber, uma inovação desenvolvida como parte desta pesquisa. A camada Defiber, presente na fase de convolução da CNN, desempenha o papel de redutor dos *feature maps*, antes de cada camada de convolução, a partir da remoção de linhas de dados.

Essa redução, típica em operações de *down sampling*, implica na diminuição do tamanho do input sem afetar adversamente a habilidade preditiva da rede, oferecendo assim uma estratégia eficiente para lidar com cenários de *hardwares* de baixo custo ou móveis.

Como continuidade, no próximo Capítulo, abordaremos a metodologia utilizada em nossa pesquisa, detalhando os procedimentos adotados para avaliar a eficácia e o desempenho do nosso algoritmo, no qual está incluso a camada Defiber, em diferentes cenários.

CAPÍTULO 5.

METODOLOGIA EXPERIMENTAL

Neste capítulo, iremos detalhar a metodologia adotada em nossa pesquisa para a realização dos experimentos, bem como a seleção das bases de dados utilizadas para avaliar o desempenho do modelo proposto. Além disso, forneceremos uma descrição completa do processo investigativo que resultou nas informações da análise empírica e que serviu de base para a criação do código da FiberNet. Também exploraremos em detalhes as metodologias empregadas para avaliar os modelos utilizados em nossa investigação.

No desenvolvimento de nosso estudo, optamos por empregar o método de validação cruzada aninhada (do inglês, *nested cross-validation*) para avaliar o desempenho do modelo. Primeiro, o conjunto de dados é dividido em k partes iguais, geralmente 5 ou 10 partes, onde *k* representa o número de *folds* no loop externo. Em cada iteração do loop externo, um *fold* é reservado como conjunto de teste. Os *k-1 folds* restantes são então divididos novamente em m partes, onde m representa o número de *folds* no loop interno.

No loop interno, cada iteração utiliza *m-1 folds* para treinamento e um *fold* separado para validação. Esse processo é utilizado para otimizar os hiperparâmetros do modelo, enquanto o *fold* de validação avalia o desempenho com diferentes configurações de hiperparâmetros. Após a otimização, o modelo é treinado usando os *k-1 folds* do loop externo com os melhores hiperparâmetros obtidos. Em seguida, o modelo é avaliado no *fold* reservado para teste no loop externo.

A avaliação final do modelo é obtida pela média dos resultados alcançados nas iterações do *loop* externo. Essa abordagem é importante para prevenir o *overfitting*, pois separa claramente as fases de treinamento, validação e teste, garantindo uma estimativa mais precisa do desempenho do modelo em dados não vistos.

5.1. BASES DE DADOS UTILIZADAS

Para nosso experimento, nós utilizamos duas bases de dados, contendo imagens diversas, para treino e teste dos modelos. A primeira base de dados, contendo imagens da temática do Sisal, foi dividida em pastas com duas classes: **sisal** e **fibra** como ilustrado na

Figura 19. Cada pasta foi subdividida em três grupos: treino, teste e validação. Na pasta de treino, foram armazenadas 70% das imagens, totalizando 560 imagens.

Na pasta de teste, utilizamos de 20% das imagens, perfazendo um total de 160 imagens. Na pasta de validação, foram utilizadas 10% das imagens, totalizando 80 imagens. Somando todos os conjuntos de imagens obtivemos um total de 800 imagens distribuídas aleatoriamente nas três pastas.

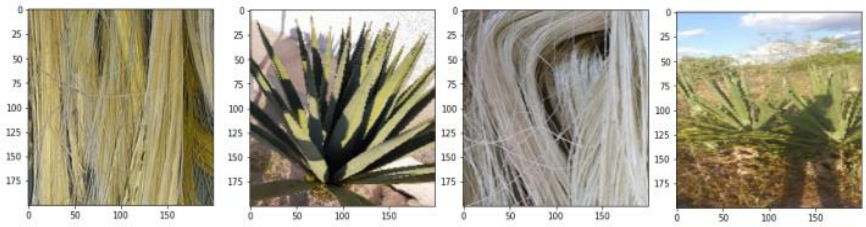


Figura 19: Imagens retiradas do banco de imagens do SISAL produzido em nossa pesquisa. No exemplo temos imagens da classe Fibra e da classe Sisal (figura do autor) na qual estão inclusas também imagens da palma do sisal.

Para garantir a aleatoriedade dos dados, realizamos o embaralhamento das imagens e distribuímos as imagens nas pastas de treino, teste e validação por dez vezes após cada embaralhamento. Ao final do processo, obtivemos 10 pastas com 800 arquivos cada, todas com dados totalmente aleatórios.

A segunda base de dados utilizada em nossa pesquisa é a base CIFAR10. A base é formada por um conjunto de imagens que contém um total de 60.000 imagens coloridas de tamanho 32x32 pixels, divididas em 10 classes igualmente distribuídas: aviões, carros, pássaros, gatos, veados, cães, sapos, cavalos, navios e caminhões conforme ilustrado na Figura 20.

As imagens são pré-processadas e normalizadas para garantir que os dados tenham a mesma escala e distribuição, o que é importante para o bom funcionamento dos modelos de Aprendizado de Máquina.

Esta base é amplamente utilizada na pesquisa de Aprendizado de Máquina, pois é uma tarefa desafiadora de classificação de imagens devido à sua complexidade por conta das imagens possuírem tamanho reduzido. Além disso, a base de dados possui um alto grau de variabilidade e uma grande diversidade de classes, o que torna o problema de classificação ainda mais desafiador.

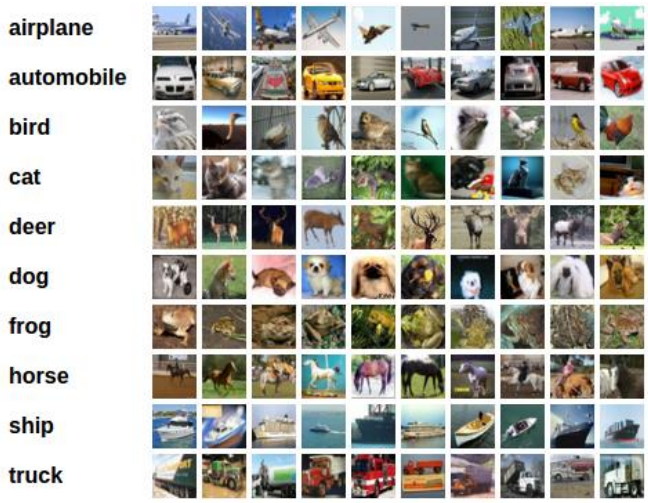


Figura 20: Imagens da base CIFAR10. Fonte: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Das 60.000 imagens, 42.000 são utilizadas para treinamento, 12.000 são utilizadas para testes e 6.000 são utilizadas para validação. A base de dados é dividida dessa forma para que os modelos possam ser treinados com uma grande quantidade de dados, enquanto a capacidade de generalização do modelo é avaliada com dados que ele nunca viu antes.

5.2. MÉTODOS E MATERIAIS

A partir dos objetivos de nossa pesquisa iniciamos o desenvolvimento de uma abordagem que atendesse aos requisitos estabelecidos. A ideia é a de que o nosso algoritmo seja estruturalmente pequeno, para ser rápido, e eficiente, por ter uma boa acurácia e poder de generalização. Como esperado de algoritmos que seguem o paradigma das redes neurais convolucionais móveis (He; Li, 2020)**.**

Além disso, inspiramo-nos também na Teoria de Reconhecimento de Componentes de Biederman (1987). Sua teoria enfatiza que, para reconhecer um objeto, não é necessário ter uma visão completa do mesmo; apenas parte das informações sobre o objeto é suficiente para compreender sua identidade.

Transpondo esse conceito para o campo da computação, podemos afirmar que um software de Visão Computacional pode vir a ter também a capacidade de identificar um objeto mesmo quando apenas informações parciais sobre ele estão disponíveis. Nossa pesquisa foi orientada por essa ideia, com o reconhecimento de que o desafio residiria na criação de um algoritmo de aprendizado profundo capaz de concretizar essa afirmação.

Durante nossa revisão da literatura, identificamos diversos modelos capazes de abordar essa realidade. Isso inclui algoritmos que empregam estratégias de *down sampling* que são usadas para reduzir e controlar o total de parâmetros treináveis nos modelos. No entanto, é importante destacar que essas abordagens também apresentam algumas limitações.

Em alguns casos, modelos que utilizam essas estratégias enfrentam desafios relacionados ao desempenho, uma vez que sua eficácia pode ser altamente sensível à resolução das imagens. Ou seja, a resolução das imagens nas quais esses modelos são treinados estabelece um limite mínimo que não pode ser ultrapassado, pois fazê-lo poderia resultar em perda de desempenho ou até mesmo na inutilização do modelo.

Entretanto, observamos também que nesses casos passa a existir certos pré-requisitos para que determinados algoritmos funcionem corretamente, como por exemplo, a necessidade de aplicar *data augmentation* (*resize*, *crop*, *flip*, entre outras) nas imagens para que ocorra sua efetiva inferência. Além do que, sem esses aprimoramentos de imagem, outros problemas como *overfitting* e *underfitting*, derivados da redução excessiva das *feature map*s durante a execução do modelo, podem surgir (Yang et.al., 2022).

Assim sendo, optamos por desenvolver um modelo próprio que fosse capaz de equilibrar eficiência e desempenho sem que fosse necessária a utilização de recursos de aprimoramento (*data augmentation*). Para esse novo modelo nós introduzimos essa nova camada, denominada Defiber, e que sua função seria a de reduzir o tamanho do *input* antes de entrar nas camadas de convolução.

Durante a fase de testes, exploramos diversas possibilidades arquitetônicas, incluindo a incorporação da camada Defiber, e conduzimos a execução de alguns milhares de combinações de hiperparâmetros, com diferentes combinações de camadas e diferentes configurações de limites na camada Defiber.

5.3. O DESENVOLVIMENTO DO ALGORITMO FIBERNET

Em consonância com os objetivos de nossa pesquisa, selecionamos um modelo protótipo com cerca de 700 mil parâmetros. Essa escolha se baseou na relação comprovada entre a quantidade de parâmetros e a eficiência da inferência (Han, et al., 2015), bem como na demonstração de desempenho satisfatório em testes preliminares com os conjuntos de imagens do Sisal e CIFAR10. Os detalhes sobre o processo de seleção do protótipo será descrito em detalhes no próximo capítulo.

Com base na análise de todos os resultados decorrentes das diferentes combinações de camadas e hiperparâmetros, estabelecemos a arquitetura definitiva do modelo proposto, conforme detalhado na Tabela 1. Esta arquitetura corresponde à implementação do modelo proposto denominado FiberNet e ela segue os conceitos básicos para uma modelo de CNN padrão.

Tabela 1: Descrição de hiperparâmetros de convolução do modelo proposto (FiberNet)

Camada	IN CHANNEL	OUT CHANNEL	KERNEL	PADDING	STRIDE	BIAS
01	3	90	2x2	0	1	TRUE
02	90	45	3x3	0	1	TRUE
03	45	90	5x5	0	1	TRUE
04	90	45	3x3	0	1	TRUE
05	45	90	7x7	0	1	TRUE
06	90	45	7x7	0	1	TRUE

A configuração das camadas de convolução em conjunto com as camadas Defiber pode ser visto na Figura 21 bem como os hiperparâmetros de convolução de cada camada.

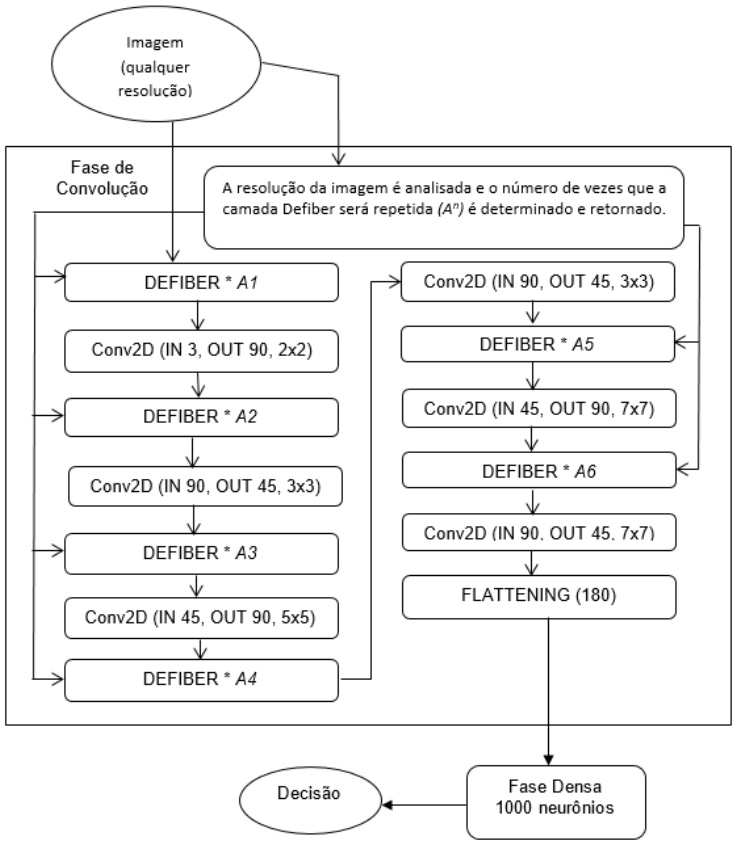


Figura 21: Fluxograma do algoritmo da FiberNet (figura do autor)

5.3.1. O desenvolvimento da função de probabilidade

Em nossa abordagem, incorporamos também uma função probabilística que desempenha um papel na determinação do número de repetições da camada Defiber. A função CDR opera utilizando o valor de uma das dimensões do tensor de entrada como base para realizar os cálculos da simulação do *pipeline* de convolução do modelo proposto. A função responsável pela quantificação das vezes que a camada Defiber deve ser repetida chama-se *Calculate Defiber Repeats* (CDR) e seu algoritmo é descrito na Tabela 2.

Tabela 2: Pseudocódigo responsável pelo cálculo do número de repetições da camada Defiber

01:	**Procedimento** Calculate_Defiber_Repeats (CDR)
02:	**ENTRADA tensor IN**
03:	**NOVA (inteiro)** variável ***X*** preenchida com o valor da última dimensão do tensor **IN**
04:	**PARA** variável ***A*** igual a zero até o valor 4 **FAÇA:**
05:	**PARA** variável ***B*** igual a zero até o valor 4 **FAÇA:** **(…)** criar mais 4 loops para ***C***, ***D***, ***E*** e ***F***.
06:	// Dentro do loop **F** nós começamos a simulação do forward pass. ***X*** e ***A*** são passadas como parâmetro para a primeira camada DefiberSim que retorna ***X1*** ao final.
07:	***X1*** é passado como parâmetro pela primeira Conv2DSim, o tamanho da saída é calculado e ***X2*** é retornado. ***X2*** e ***B*** são passados como parâmetro para a próxima camada DefiberSim que retorna ***X3***. **(…)** Repetimos esse procedimento ao longo de todas as camadas DefiberSim e Conv2DSim até que um valor seja retornado para ***X(…)***.
08:	**SE *X(…)*** for igual a 2 **ENTÃO:**
09:	**RETORNE** os valores de *A1*, *A2*, *A3*, *A4*, *A5* e *A6* para serem configurados nas camadas Defiber do fluxo principal.
10:	**FIM SE**
11:	**FIM PARA**
12:	**FIM PARA**
13:	**FIM PARA**
14:	**FIM PARA**
15:	**FIM PARA**
16:	**FIM PARA**
17:	**FIM do procedimento**

O algoritmo CDR recebe um tensor como parâmetro de entrada. Desse tensor é extraído o valor de sua última dimensão que é utilizado para alimentar uma variável de controle do tipo inteiro (X). Essa função cria também uma lista que é preenchida com valores

de 0 a 4 e ela é utilizada por uma sequência de *loops* aninhados conforme ilustrado na Figura 22.

Cada *loop* executa uma cópia dessa lista onde a variável de iteração assume um dos valores. Assim, em cada iteração, uma combinação de valores vai sendo montada. Por exemplo, na primeira iteração temos o valor: 0,0,0,0,0,0 na segunda iteração temos o valor: 0,0,0,0,0,1, na terceira iteração: 0,0,0,0,0,2 e assim por diante até as combinações montem o valor de: 4,4,4,4,4,4.

No algoritmo da Tabela 2 esses valores são representados pelas variáveis *A, B, C, D, E* e *F* e dentro do último *loop* ocorre uma simulação (réplica) das camadas Defiber e das camadas de convolução, seguindo a mesma sequência e representando cada uma das camadas do fluxo original (*forward*). No entanto, existem diferenças entre as camadas simuladas e as camadas originais.

5.3.2. As simulações das camadas de convolução e Defiber

Nas camadas originais, as camadas Defiber e de Convolução processam um tensor ao longo do seu fluxo. Já na simulação, o valor processado corresponde a uma variável do tipo inteiro (*int*) que é preenchida e percorre todo o fluxo simulado até que o resultado final seja um valor igual a 2. Esse valor 2 simula o que seria um tensor de 2x2 no fluxo original.

A simulação das camadas Defiber, como descrito na Tabela 3, são cópias do algoritmo original e funcionam de maneira similar quanto ao processo de remoção de eixos a partir de um valor limite. A diferença é que na cópia simulada esta função retorna o tamanho da lista no qual são armazenados os valores dos índices dos eixos do tensor que não são divisíveis pelo valor do limite.

O retorno da simulação da camada Defiber segue para a simulação da primeira camada de convolução. A simulação da camada de convolução ocorre a partir de uma fórmula que, dadas às configurações de hiperparâmetros da camada, consegue calcular o tamanho da saída do tensor que passa por ela. Essa simulação é realizada através da aplicação da seguinte fórmula:

*output_size = (input_size + (padding_size * 2) - kernel_size) // stride_size + 1.*

Se o valor retornado pela última camada Defiber for igual a 2, a função CDR retorna para o fluxo original do modelo os valores das variáveis de A, B, C, D, E e F. Esses valores

são aqueles através do qual as camadas Defiber, do fluxo original, serão repetidos no decorrer do processamento de uma imagem.

Tabela 3: Simulação da Camada Defiber utilizada pela função CDR

01:	**PROCEDIMENTO** DefiberSimulation
02:	**ENTRADA** inteiro LastDimSize
03:	**ENTRADA** inteiro *NumRepeatTimes*
04:	**NOVA LISTA** inteiro *NewList*
05:	**PARA** var *i* igual a zero até o tamanho de *NumRepeatTimes* FAÇA:
06:	**PARA** var *j* igual a zero até o tamanho de *LastDimSize* FAÇA:
07:	**SE** var *j* não é divisível pelo limite *(t)* ENTÃO: **ADICIONAR** var j em *NewList*
08:	**FIM SE**
09:	**FIM PARA**
10:	**FIM PARA**
11:	**RETORNE** o tamanho de *NewList*
12:	**END** procedure

Além disso, nós realizamos também testes para aferir a qualidade do modelo utilizando a camada Defiber comparando-o com o mesmo algoritmo utilizando, no lugar da camada Defiber, camadas de *pooling* (Tabela 4). Nessas versões, assim como na FiberNet, nós utilizamos apenas configurações consideradas básicas nas principais partes do código como nas camadas de convolução, função de ativação, normalização e camada linear.

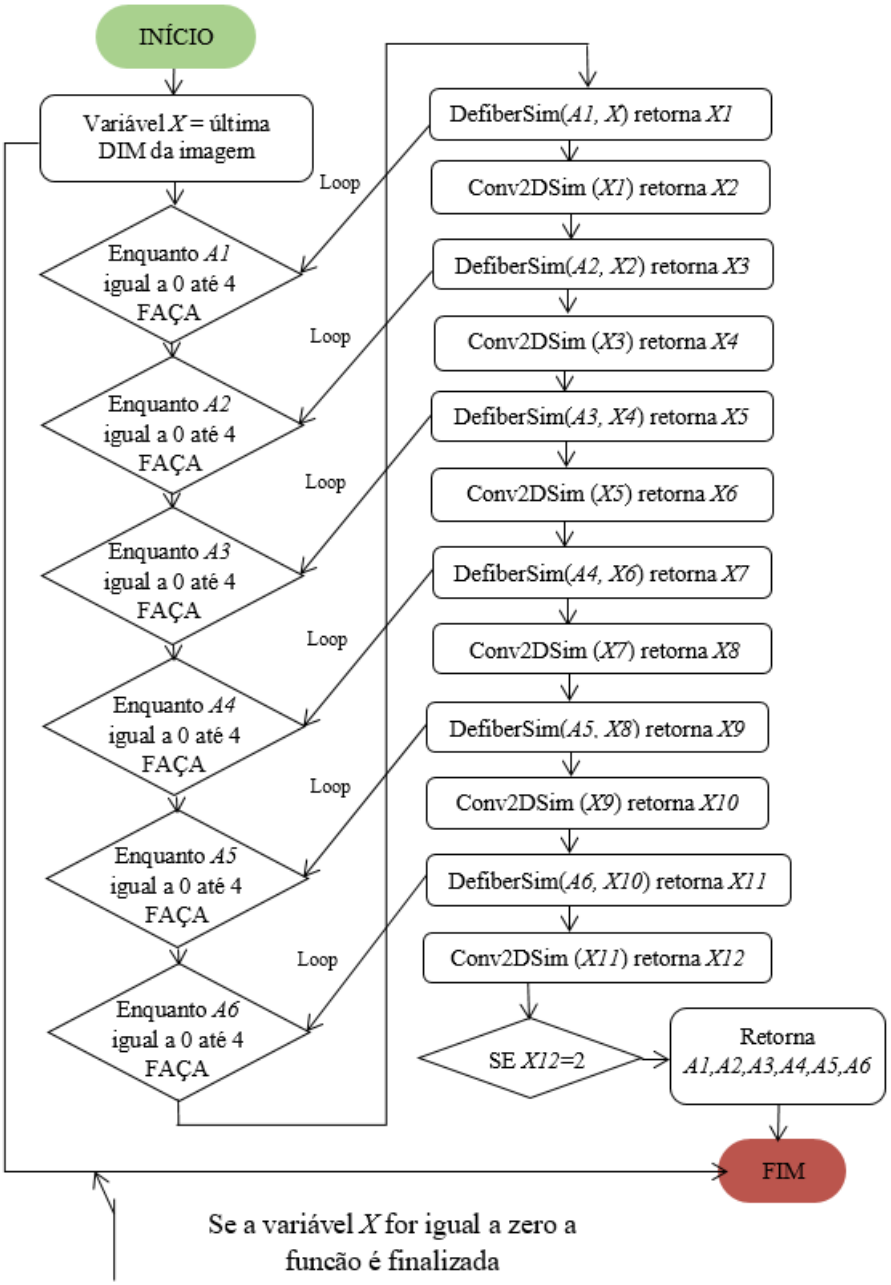


Figura 22: Fluxograma da função CDR.

Tabela 4: Resultado de estudo comparativo da arquitetura da FiberNet utilizando a camada Defiber e outras metodologias de down sample sobre a base CIFAR10

Modelos	Parâmetros	Camadas	Kernel
FiberNet com a camada Defiber	754.345	6	---
FiberNet com camada de *Max-Pooling*	6.019.345	6	7x7
FiberNet com camada de Min-*Pooling*	6.019.345	6	7x7
FiberNet com camada de SUM-*Pooling*	6.019.345	6	7x7

5.4. DEFINIÇÃO DO PROCESSO DE ANÁLISE COMPARATIVA ENTRE ALGORITMOS.

A partir da elaboração de um modelo próprio avaliamos nossa nova camada, e consequentemente nosso protótipo, a partir de um processo comparativo. Esse processo compreende a avaliação qualitativa, a partir de método que aferiu sua acurácia, e a avaliação quantitativa, a partir de método que aferiu o seu desempenho a nível *hardware*.

5.4.1. Fase 01: Comparação da FiberNet com outras arquiteturas

Para isso, nós elaboramos uma sequência de avaliações comparativas da FiberNet contra outros algoritmos considerados como referência na área de aprendizado profundo. A maioria deles são descritos no Capítulo 4, que trata sobre as arquiteturas de CNN, e são os seguintes: DenseNet(), EfficientNetV1(), EfficientNetV2(), GoogleNet(), MobileNetV2(), MobileNetV3(), ResNet(), ShuffleNetV2(),e a SqueezeNet().

Esses modelos se notabilizaram pela alta eficiência em determinados desafios de classificação de imagens e agrupam modelos que podem ser considerados como profundos, pela sua arquitetura que contém uma grande quantidade de camadas, e outros modelos que podem ser considerados como móveis, por possuírem uma menor quantidade de parâmetros treináveis e serem adaptáveis à *hardwares* de baixo custo.

O objetivo dessas comparações é de demonstrar que nossa proposta pode ser efetiva na qualidade das predições, tanto quanto os modelos considerados como profundos, quanto podem ser utilizáveis em *hardwares* de baixo custo, tanto quanto os modelos considerados como próprios para ambientes móveis.

5.4.2. Fase 02: Comparação da FiberNet com variações de sua própria arquitetura

Adicionalmente, realizaremos uma comparação abrangente entre nossa camada Defiber e outras estratégias de *down sampling*. Para isso, empregaremos a arquitetura modelo FiberNet como base, desenvolvendo variantes onde substituímos a camada Defiber por alternativas, como camadas de *pooling*. Essa análise permitirá uma avaliação minuciosa do desempenho da camada Defiber em relação a outras técnicas de *down sampling*.

Nossa meta ao realizar essa comparação é explorar os limites do modelo proposto, com o intuito de examinar detalhadamente o desempenho da camada Defiber em comparação com outras estratégias de mesmo grupo. Dessa forma, pretendemos avaliar as vantagens de

nossa camada, ao mesmo tempo em que identificamos os possíveis limites que surgem ao compará-la com outras estratégias similares.

Os modelos empregados nesta etapa da pesquisa são descritos na Tabela 5 para os algoritmos avaliados com a base de imagens do Sisal, e na Tabela 6 para os algoritmos testados com a base de imagens da CIFAR10. Na tabela, os algoritmos foram nomeados de forma a indicar claramente a estratégia de *down sampling* aplicada em cada arquitetura.

Tabela 5: Variações da FiberNet testadas com a base Sisal

Modelo	Parâmetros	Camadas	Kernel	Stride
FiberNet_Defiber	754.345	Variável	Não utiliza	Não utiliza
FiberNetAVG_700K	754.345	30	1	1
FiberNetMAX_700K	754.345	30	1	1
FiberNetSUM_700K	754.345	30	1	1
FiberNetMIN_700K	754.345	30	1	1
FiberNetAVG_1M	1.699.345	30	1	1
FiberNetMAX_1M	1.699.345	30	1	1
FiberNetSUM_1M	1.699.345	30	1	1
FiberNetMIN_1M	1.699.345	30	1	1
FiberNetAVG_6M	6.019.345	30	1	1
FiberNetMAX_6M	6.019.345	30	1	1
FiberNetSUM_6M	6.019.345	30	1	1
FiberNetMIN_6M	6.019.345	30	1	1

Tabela 6: Variações da FiberNet testadas com a base CIFAR10

Modelo	Parâmetros	Camadas	Kernel	Stride
FiberNet_Defiber	754.345	1	Não utiliza	Não utiliza
FiberNetAVG_700K	754.345	9	1	1
FiberNetMAX_700K	754.345	9	1	1
FiberNetSUM_700K	754.345	9	1	1
FiberNetMIN_700K	754.345	9	1	1
FiberNetAVG_1M	1.699.345	6	1	1
FiberNetMAX_1M	1.699.345	6	1	1
FiberNetSUM_1M	1.699.345	6	1	1
FiberNetMIN_1M	1.699.345	6	1	1
FiberNetAVG_6M	6.019.345	0	1	1
FiberNetMAX_6M	6.019.345	0	1	1
FiberNetSUM_6M	6.019.345	0	1	1
FiberNetMIN_6M	6.019.345	0	1	1

Nesse contexto, a nomenclatura empregada é a soma do nome FiberNet com uma abreviação do nome da estratégia utiliza. A sigla AVG, por exemplo, é uma abreviação do termo *average pooling* ou agrupamento por média, essa técnica reduz o mapa de

características de entrada a partir da média dos valores do grupo de *pixels* avaliados pelo filtro.

Já a abreviação MAX faz referência à técnica de *max pooling* que funciona reduzindo o mapa de características de entrada a partir da captura do maior valor do grupo de *pixels*. Por sua vez a SUM, advinda da técnica de *sum pooling*, reduz o mapa de características de entrada a partir da somatória dos valores do grupo de *pixels* avaliados pelo filtro. E por último temos o MIN que faz referência à técnica de *min pooling* que por sua vez reduz o mapa de características de entrada a partir da captura do menor valor do grupo de *pixels*.

Com relação à quantidade de camadas descritas nas Tabelas 5 e 6, observa-se uma variação para o modelo FiberNet_Defiber entre uma tabela e outra. Essa variação ocorre devido à presença de uma função de probabilidade em nosso algoritmo, que avalia a necessidade de ativar a camada Defiber com base na resolução da imagem de entrada.

Portanto, nos momentos em que o algoritmo determina que a camada Defiber não é necessária, ela é essencialmente desconsiderada durante a fase de convolução. Assim, apenas as ativações são contabilizadas para representar a presença real da camada durante o processamento da imagem.

5.5. HIPERPARÂMETROS DE TREINAMENTO

Hiperparâmetros de treinamento são parâmetros configuráveis externamente que controlam o comportamento de um algoritmo de Aprendizado de Máquina. Ao contrário dos parâmetros do modelo, que são aprendidos durante o treinamento, esses parâmetros são definidos pelo desenvolvedor antes do processo de treinamento e afetam diretamente o desempenho e a capacidade de generalização do modelo.

Em nossa pesquisa, nós utilizamos um conjunto de hiperparâmetros que foram definidos, antes das sessões de treinamento, levando em consideração as restrições do *hardware* e os ajustes realizados durante o processo de análise empírica. Nesse processo nós ajustamos as configurações gradualmente até encontrarmos os melhores valores para cada um dos hiperparâmetros utilizados. Nos próximos parágrafos, nós descrevemos os detalhes sobre os hiperparâmetros utilizados, valores definidos para nossa pesquisa e o impacto esperado nos modelos avaliados.

5.5.1. Batch-size

O tamanho do lote (*batch-size*) é um hiperparâmetro que controla o número de amostras de treinamento processadas em uma única atualização dos pesos do modelo durante o treinamento. Ele define o tamanho dos grupos de dados, chamados lotes, que são utilizados para calcular os gradientes e ajustar os pesos da rede. Em nossa pesquisa nós utilizamos um valor de *batch-size* compatível com a quantidade de imagens de base de dados utilizada nos experimentos (base Sisal e base CIFAR10) e que foi ajustado durante a fase de análise empírica.

Para a base do Sisal nós observamos que, por conta de restrições de *hardware*, quando nós utilizamos valores muito grandes o sistema do *Google Colab* apresentava falhas. Dessa forma restou definido o valor 4 para o treinamento com essa base. Já com a base CIFAR10 nós começamos a análise a partir do valor 8 e seguimos aumentado o valor, com múltiplos de 8, até o valor limite que foi de 64.

Com os valores utilizados até o limite nós não observamos nenhum decréscimo de acurácia e o tempo de treinamento, com cada modelo, continuou relativamente igual. Os resultados apresentados no Capítulo 6 refletem o processo de inferência dos modelos treinados com estes valores.

Adicional a este hiperparâmetro, nós temos também um segundo *batch-size* que é utilizado pelo método que avalia a taxa de transferência dos modelos. A taxa de transferência é a quantidade de dados que a rede pode processar por unidade de tempo e a taxa de inferência é a quantidade de tempo que a rede pode levar para processar uma saída. Em nossa pesquisa nós utilizamos os algoritmos propostos por Geifman (2020) para aferir essas duas métricas.

Para esse *batch-size* nós também utilizamos o valor limite de 64 e os resultados obtidos para este critério, descritos no Capítulo 6, expressam a quantidade de dados que podem ser transferidos pelos modelos em um determinado período de tempo.

5.5.2. Épocas

As épocas são uma unidade de medida que representa uma passagem completa por todo o conjunto de dados de treinamento durante o processo de aprendizado do modelo. Em cada época, o algoritmo de treinamento utiliza todo o conjunto de dados para atualizar os pesos das conexões entre os neurônios da rede, buscando minimizar o erro entre as previsões do modelo e os rótulos corretos dos dados de treinamento.

Da mesma maneira, considerando a limitação de *hardware* disponível e nivelando gradualmente o valor desse parâmetro, optamos por fixar o limite de 50 épocas para treinar todos os modelos em nossa pesquisa. Essa escolha foi fundamentada na constatação de que, durante a análise empírica, todos os modelos apresentaram um desempenho notavelmente estável com este valor. Assim, buscamos equilibrar a eficiência do treinamento com a garantia de uma boa capacidade de generalização dos modelos, considerando as limitações do ambiente de processamento.

Além disso, optamos por conduzir o experimento de forma imparcial, assegurando que nenhum modelo fosse favorecido em detrimento dos outros, uma vez que todos foram treinados com exatamente as mesmas configurações de hiperparâmetros. Essa abordagem nos proporcionou resultados comparáveis e confiáveis, permitindo uma avaliação mais justa e precisa do modelo proposto e concorrentes.

5.5.3. Taxa de aprendizado e otimizador

Outro hiperparâmetro utilizado foi a taxa de aprendizado. Ela representa o tamanho do passo que a rede neural dá em direção ao mínimo local durante o processo de otimização dos pesos. Em outras palavras, a taxa de aprendizado controla o quão rápido o modelo ajusta seus parâmetros em resposta ao erro calculado durante o treinamento.

Adicionalmente à taxa de aprendizado, utilizamos também um otimizador adaptativo que é um tipo de algoritmo usado para ajustar os parâmetros da rede durante o processo de treinamento. Esse algoritmo ajusta a taxa de aprendizado individualmente para cada parâmetro, permitindo uma adaptação mais precisa e dinâmica durante o treinamento. Esse otimizador leva em consideração o histórico dos gradientes calculados durante o treinamento para determinar a taxa de aprendizado para cada parâmetro.

Dessa forma, parâmetros que estão convergindo mais lentamente ou com oscilações podem receber uma taxa de aprendizado menor, enquanto parâmetros que estão progredindo rapidamente podem receber uma taxa de aprendizado maior. Em nossa pesquisa nós utilizamos o otimizador ADAM (Kingma, 2014) através do qual nós escalonamos o valor da taxa de aprendizado.

O otimizador Adam (*Adaptive Moment Estimation*) é um algoritmo de otimização estocástica baseado em gradiente descendente que é frequentemente usado para treinar redes neurais profundas. Ele combina as ideias de dois outros algoritmos de otimização populares, o Momentum (Hinton, Srivastava e Swersky, 2012) e o RMSprop (Hinton, 2012).

O Momentum é um algoritmo que usa os erros passados para estimar a direção geral do aprendizado. Isso ajuda a evitar que o algoritmo fique preso em um ponto onde os erros não mudam. O RMSprop é um algoritmo que usa os erros passados para estimar a rapidez com que os erros estão mudando. Isso ajuda o algoritmo a ajustar a velocidade de aprendizado de acordo com a rapidez com que está aprendendo.

Em nossa pesquisa nós começamos a análise empírica com a variação da taxa de aprendizado a partir do valor de 0,1 até o valor definido, e com o qual treinaríamos os modelos, de 0,0001. Com esse valor nós observamos que houve uma boa convergência dos modelos atingindo o ápice das acurácias sem desestabilizar a curva de aprendizado do gradiente.

5.5.4. Função de custo e aspectos gerais

Por fim, dentre os hiperparâmetros mais relevantes, nós utilizamos também uma função de custo (*criterion*). Essa função é responsável por calcular a diferença entre as previsões feitas pelo modelo e os rótulos verdadeiros dos dados de treinamento. O objetivo do treinamento é minimizar essa função de perda para que o modelo seja capaz de fazer previsões mais precisas nos dados de teste.

Uma vez que nossa pesquisa se trata de um problema de classificação nós escolhemos a função de perda de entropia cruzada (*Cross-Entropy*) que mede a discrepância entre as probabilidades de classe previstas e as probabilidades de classe verdadeiras.

Com tudo que foi definido, fica claro que nós buscamos avaliar os modelos de forma cuidadosa e eficiente. Portanto, duas importantes decisões foram tomadas: (a) optamos por não utilizar nenhum tipo de aumento de dados (*data augmentation*) e (b) todos os modelos foram treinados sem utilizar inicialização de pesos. Essas escolhas foram fundamentadas em suas vantagens específicas. Ao não aplicar o aumento de dados, permitimos que os modelos fossem treinados exclusivamente com os dados originais, aproveitando a representatividade do conjunto de dados em sua forma natural.

Essa abordagem foi especialmente útil quando o conjunto de dados original continha informações suficientes para a tarefa em questão. Além disso, ao evitar a inicialização de pesos, garantimos que os modelos começaram o treinamento com valores neutros, o que reduziu possíveis vieses ou estados desfavoráveis.

Dessa forma, o aprendizado ocorreu de maneira mais imparcial, permitindo a descoberta eficiente de padrões relevantes nos dados. Vale ressaltar que essas decisões foram

consideradas em função da natureza específica da tarefa e do conjunto de dados utilizados em nossa pesquisa, visando maximizar a eficácia na avaliação dos modelos (desenvolvido e concorrentes).

5.6. MATRIZ DE CONFUSÃO: VERIFICANDO A ACURÁCIA

A matriz de confusão é uma Tabela que permite visualizar o desempenho de um algoritmo de classificação (Karl Pearson, 1904). Ela mostra o número de classificações corretas e incorretas em cada classe do problema de classificação.

A matriz é geralmente representada por uma matriz quadrada, onde cada linha representa as instâncias em uma classe real, enquanto cada coluna representa as instâncias em uma classe prevista pelo modelo. A diagonal principal da matriz de confusão mostra o número de classificações corretas para cada classe. Já as outras células mostram os erros de classificação, onde uma célula fora da diagonal representa uma classificação incorreta.

Existem várias métricas de avaliação de desempenho que podem ser calculadas a partir da matriz de confusão, como a acurácia (proporção de classificações corretas em relação ao total de classificações), a sensibilidade (proporção de instâncias positivas corretamente classificadas em relação ao total de instâncias positivas), a especificidade (proporção de instâncias negativas corretamente classificadas em relação ao total de instâncias negativas), entre outras.

- Acurácia (do inglês, *accuracy*): É a proporção de classificações corretas em relação ao total de classificações. É uma métrica comum para avaliar o desempenho geral do modelo, mas pode ser enganosa quando as classes são desbalanceadas: (VP[2]+VN[3]) / (VP+FP[4]+FN[5]+VN);
- Sensibilidade (VP / (VP + FN)): mede a capacidade de um modelo de classificação em prever corretamente os casos positivos.
- Especificidade (*True Negative Rate*): É a proporção de instâncias negativas corretamente classificadas em relação ao total de instâncias negativas. É uma métrica

[2] **Verdadeiro Positivo (VP):** Ocorre quando o modelo prediz corretamente que uma amostra pertence à classe positiva.
[3] **Verdadeiro Negativo (VN):** Ocorre quando o modelo prediz corretamente que uma amostra pertence à classe negativa.
[4] **Falso Positivo (FP):** Ocorre quando o modelo prediz incorretamente que uma amostra pertence à classe positiva, mas na verdade ela pertence à classe negativa.
[5] **Falso Negativo (FN):** Ocorre quando o modelo prediz incorretamente que uma amostra pertence à classe negativa, mas na verdade ela pertence à classe positiva.

importante quando o objetivo é minimizar os falsos positivos. Ela é expressa pela fórmula: (VN / (VN+FP)).

Além da acurácia, sensibilidade e especificidade mencionadas anteriormente, a matriz de confusão permite calcular outras métricas importantes para avaliar o desempenho de um modelo de classificação. Algumas delas são:

- Precisão: a precisão é a proporção de instâncias positivas classificadas corretamente em relação ao total de instâncias classificadas como positivas. Essa métrica é útil em problemas em que os falsos positivos são considerados mais graves do que os falsos negativos, como em testes de diagnóstico de doenças: ((VP) / (VP+FP)).
- *Recall*: o *recall* (também conhecido como taxa de verdadeiros positivos) é a proporção de instâncias positivas classificadas corretamente em relação ao total de instâncias que realmente pertencem à classe positiva. O *recall* é útil em problemas em que os falsos negativos são considerados mais graves do que os falsos positivos, como em testes de segurança: ((VP) / (VP+FN)).
- F1-score: o F1-score é uma medida que combina a precisão e o *recall* em uma única métrica. É útil em problemas em que é importante obter um equilíbrio entre a precisão e o recall: (Precisão * Recall) / (Precisão + Recall).

Para avaliar o desempenho dos algoritmos de classificação utilizados em nossa pesquisa, utilizamos a matriz de confusão pela métrica da acurácia. A partir dos resultados obtidos, realizamos um comparativo qualitativo entre os algoritmos, compreendendo quais são os possíveis melhores resultados.

5.7. MÉTODO ESTATÍSTICO DE FRIEDMAN

O teste de Friedman (Friedman, 1937) é um método estatístico não paramétrico que utilizamos em nossa pesquisa para avaliar a diferença entre as classificações médias dos grupos amostrais. Esse teste é frequentemente utilizado para comparar o desempenho de vários algoritmos em problemas de classificação. A hipótese nula do teste é que as classificações médias de todas as amostras são iguais. A hipótese alternativa é que pelo menos uma amostra é diferente das outras.

O teste de Friedman usa uma tabela de classificação para comparar o desempenho de cada algoritmo em cada amostra. Para cada algoritmo, as amostras são classificadas em ordem crescente de desempenho. O número de pontos que cada algoritmo recebe em cada amostra é registrado na tabela. O teste de Friedman calcula uma estatística de teste chi-

quadrado a partir da tabela de classificação e dos valores de classificação média para cada algoritmo. O valor da estatística de teste é então comparado com uma distribuição de referência chi-quadrado com graus de liberdade apropriados.

Na prática, o teste de Friedman funciona quando calculamos o *p-value*. Esse valor indica a probabilidade de obter uma diferença significativa entre as classificações médias dos modelos testados, se a hipótese nula, supondo que não há diferença significativa entre as classificações médias, for verdadeira. Em outras palavras, o *p-value* informa o quão provável é que as diferenças observadas entre as classificações médias dos modelos testados tenham ocorrido ao acaso.

Um valor de *p* menor do que o nível de significância pré-determinado (geralmente 0,05) sugere que as diferenças observadas são estatisticamente significativas e que a hipótese nula pode ser rejeitada, indicando que pelo menos um dos modelos testados é significativamente diferente dos outros. Por outro lado, um valor de *p* maior do que o nível de significância sugere que não há evidência suficiente para rejeitar a hipótese nula e que as diferenças observadas podem ter ocorrido ao acaso.

Se o valor da estatística de teste for maior do que o valor crítico na distribuição de referência, a hipótese nula é rejeitada e conclui-se que há pelo menos uma diferença significativa entre as classificações médias de pelo menos dois algoritmos. Em seguida, é feito um teste de comparação múltipla para determinar quais algoritmos são significativamente diferentes entre si.

Por exemplo, suponha que temos três modelos de classificação, A, B e C, e queremos determinar se há diferença significativa entre suas classificações médias em um determinado conjunto de dados. Para isso, aplicamos o teste de Friedman com um nível de significância de 0,05. O resultado do teste é um valor estatístico de 6,0 e um valor de *p* de 0,05.

Isso significa que, se a hipótese nula for verdadeira (ou seja, não há diferença significativa entre as classificações médias dos modelos), há uma probabilidade de 5% de obter uma estatística de teste tão grande ou maior do que 6,0, ou seja, as diferenças observadas podem ter ocorrido ao acaso.

Nesse caso, como o valor de *p* é menor do que o nível de significância pré-determinado de 0,05, podemos concluir que há evidência suficiente para rejeitar a hipótese nula e afirmar que pelo menos um dos modelos é significativamente diferente dos outros.

5.8. MÉTODO ESTATÍSTICO DE NEMENYI

O teste estatístico de Nemenyi (Nemenyi, 1963)**,** também conhecido como Teste de Nemenyi-Damico-Wolfe-Holm (Nemenyi-DWH), é um teste de comparação múltipla não paramétrico utilizado para avaliar se há diferenças significativas entre grupos de dados independentes. O teste é uma extensão do teste de Friedman, que é utilizado para comparar classificações médias em dois ou mais tratamentos independentes.

O teste de Nemenyi compara todos os pares de tratamentos para determinar quais têm diferenças significativas. A estatística de teste é baseada na distância média entre as classificações de dois tratamentos independentes, em relação a todas as outras classificações, e é comparada com um valor crítico determinado a partir de tabelas específicas.

Uma das principais vantagens do teste de Nemenyi é que ele não requer que os dados atendam a uma distribuição normal, o que o torna adequado para conjuntos de dados que não seguem uma distribuição específica. No entanto, é importante ressaltar que o teste assume que as amostras têm variâncias iguais.

O resultado do teste de Nemenyi é um valor de estatística de teste e um valor de *p*, que indica a probabilidade de obter uma estatística de teste tão grande ou maior do que o valor observado, se a hipótese nula de igualdade entre os grupos for verdadeira. Se o valor de *p* for menor do que um nível de significância pré-determinado (por exemplo, 0,05), podemos concluir que há diferenças significativas entre pelo menos um par de tratamentos independentes.

O teste de Nemenyi é comumente utilizado para comparar múltiplos modelos ou algoritmos em um conjunto de dados. Por exemplo, suponha que temos um conjunto de dados com 10 algoritmos diferentes que foram testados em 5 conjuntos de dados diferentes.

Para comparar o desempenho dos algoritmos em todos os conjuntos de dados, podemos realizar o teste de Friedman para determinar se há diferenças significativas entre os algoritmos. Se o teste de Friedman for significativo, podemos então aplicar o teste de Nemenyi para determinar quais algoritmos diferem significativamente uns dos outros em termos de desempenho.

5.9. CONSIDERAÇÕES FINAIS

Neste capítulo, detalhamos todos os aspectos metodológicos empregados em nossa pesquisa. Desde a descrição das bases de imagens utilizadas até a definição dos hiperparâmetros de treinamento, desde a sua fase de planejamento até a sua execução.

Além disso, discutimos os métodos estatísticos empregados para avaliar os resultados. Utilizamos a matriz de confusão para analisar a performance dos modelos, bem como algoritmos que medem a taxa de transferência e inferência, a fim de avaliar o desempenho de hardware. Aplicamos também os testes de Friedman e Nemenyi para comparar e validar as diferenças significativas entre os resultados obtidos.

No próximo capítulo, apresentaremos os dados gerados a partir de nossos experimentos, discutindo detalhadamente cada um dos resultados alcançados.

CAPÍTULO 6.

ANÁLISE DOS RESULTADOS

Neste capítulo, forneceremos uma visão detalhada dos resultados que emergiram da análise comparativa dos modelos. Aqui, apresentaremos os resultados provenientes das análises exploratórias buscando desenvolver nosso algoritmo e os resultados das classificações obtidas pelos modelos em resposta ao desafio de analisar as imagens das bases de dados Sisal e CIFAR10. Além disso, conduzimos uma discussão aprofundada sobre os resultados, destacando os aspectos mais significativos no contexto da comparação entre os modelos.

6.1. RESULTADOS DA ANÁLISE EMPÍRICA

Os modelos resultantes do processo de análise algorítmica preliminar variaram significativamente em tamanho, desde arquiteturas muito robustas com mais de 10 milhões de parâmetros até modelos mais compactos com apenas 100 mil. O protótipo apresentando na Seção 5.3 é fruto do resultado da exploração apresentada na Tabela 7 onde nós testamos diversas combinações de modelos construídos a partir da variação de camadas convolutivas e alteração de hiperparâmetros de convolução.

Tabela 7: Total de arquiteturas testadas por quantidade de parâmetros e suas respectivas acurácias médias por grupo (bases Sisal e CIFAR10)

Total de parâmetros	Total de modelos	(%)	Sisal (acurácia)	CIFAR10 (acurácia)
>=10.000.000	584	5,83%	98,9%	68,4%
Entre 10.000.000 e 5.000.000	17	0,17%	53,7%	48,9%
Entre 5.000.000 e 3.000.000	14	0,14%	71,7%	54,4%
Com 2.000.000	817	8,15%	96,9%	51,6%
Com 1.000.000	738	7,36%	85,4%	52,7%
Com 900.000	260	2,59%	92,7%	54,9%
Com 800.000	2211	22,06%	92,7%	56,8%
Com 700.000	848	8,46%	96,2%	65,7%
Com 600.000	18	0,18%	92,7%	65,0%
Com 500.000	690	6,88%	92,7%	58,8%
Com 400.000	1309	13,06%	94,6%	61,4%
Com 300.000	2492	24,86%	93,3%	34,3%
Com 200.000	16	0,16%	88,3%	37,2%

Com 100.000	6	0,06%	92,7%	5,0%
<= 100.000	3	0,04%	82,3%	0,0%
Total Geral	10023	100%		

A partir do nosso algoritmo nós conduzimos também testes de variações no limite aplicado na camada Defiber. Esse limite é o responsável por determinar a intensidade da redução da imagem antes de cada camada de convolução. Durante os testes nós observamos que a acurácia dos modelos não era afetada pela variação de limites, porém houve uma redução significativa na quantidade de parâmetros quando utilizamos o limite 3 (Tabela 8) motivo pelo qual o escolhemos como referência para nossa pesquisa.

Tabela 8: Resultado do teste de arquitetura por variação de limite na camada Defiber e seus resultados (parâmetros e acurácias)

Limite	Sisal (%)	Parâmetros	CIFAR10 (%)	Parâmetros
01	---		---	Reduz em excesso
02	---		---	Reduz em excesso
03	0,96	754.345	0,78	754.345
04	0,96	15.154.345	0,78	3.454.345
05	0,95	52.594.345	0,79	8.179.345
06	0,95	113.074.345	0,79	13.579.345
07	0,95	179.179.345	0,79	16.819.345
08	0,95	260.494.345	0,79	22.354.345
09	Out of memory[6]	333.394.345	0,76	28.699.345
10	Out of memory	398.194.345	0,77	30.994.345

6.2. FASE 1: RESULTADO DOS EXPERIMENTOS PELA COMPARAÇÃO ENTRE ALGORITMOS

Nessa Seção, começamos a avaliar o modelo proposto, a FiberNet, em comparação com outras arquiteturas e estratégias de *down sampling* conforme descrito no capítulo sobre a metodologia. Os resultados provenientes dessa comparação ajudarão a entendermos os limites da nossa proposta, ressaltando seus pontos fortes e fracos.

Para este experimento, destacamos seções específicas para avaliar os resultados, realizando comparações diretas entre nosso algoritmo e outras arquiteturas proeminentes na

[6] Em determinadas situações, durante a fase de treinamento, quando o modelo possui uma quantidade de parâmetros treináveis muito grande, a GPU online do Google Colab, ferramenta que utilizamos em nossa pesquisa, devolve uma mensagem de erro: "*out of memory*" por ter excedido o limite de memória estabelecido para a operação do serviço no momento de execução.

área. Além disso, comparamos nossos resultados com os de outros algoritmos, categorizando-os entre modelos profundos e móveis.

6.2.1. Comparação da quantidade de parâmetros treináveis dos algoritmos

O resultado apresentado na Tabela 9 representa a comparação entre os modelos a partir de suas respectivas quantidades de parâmetros treináveis. A quantidade de parâmetros treináveis de um modelo CNN refere-se ao número total de pesos e vieses que podem ser ajustados durante o processo de treinamento.

Esses parâmetros são essenciais para a aprendizagem da rede, pois são atualizados iterativamente através do algoritmo de otimização para minimizar a função de perda e melhorar a precisão do modelo.

Em uma CNN, os parâmetros treináveis incluem os pesos das conexões entre neurônios nas camadas convolucionais e totalmente conectadas, bem como os vieses associados a essas conexões. A quantidade de parâmetros treináveis é um indicador da complexidade do modelo, influenciando tanto sua capacidade de generalização quanto os requisitos computacionais para treinamento e inferência.

Tabela 9: Modelos distribuídos por quantidade de parâmetros treináveis e tamanho do arquivo

Modelos	Parâmetros ▼	Tipo	(MB) ▼
FiberNet	754.345	Móvel	2.9
SqueezeNet	1.248.424	Móvel	5.0
ShuffleNet	2.278.604	Móvel	9.0
MobileNetV3	2.542.856	Móvel	9.7
MobileNetV2	3.504.872	Móvel	14.0
EfficientNetV1	5.288.548	Profundo	20.0
GoogleNet	6.624.904	Profundo	25.0
DenseNet	8.534.408	Profundo	31.0
ResNet	11.689.512	Profundo	45.0
EfficientNetV2	22.103.832	Profundo	84.0

Em modelos mobile, o valor da quantidade de parâmetros treináveis é especialmente crítico devido às limitações de recursos computacionais e de memória dos dispositivos móveis. Modelos desenhados para esses ambientes são projetados para manter um equilíbrio entre eficiência e precisão, reduzindo significativamente o número de parâmetros treináveis sem comprometer a performance do modelo.

Como resultado de nossos experimentos, a FiberNet, obteve o melhor desempenho geral. A FiberNet possui -39,57 p.p. (pontos percentuais) menos parâmetros treináveis do que o segundo colocado, a SqueezeNet, que também é considerada um modelo do tipo móvel. Além disso, nossa FiberNet possui -85,73 p.p. menos parâmetros do que o menor modelo entre os algoritmos do tipo profundo, a EfficientNet V1.

Destacar esse contexto é relevante porque, juntamente com outros resultados que serão apresentados ao longo deste capítulo, demonstra que nosso algoritmo atingiu um patamar de igualdade e, em muitos casos, superioridade em relação a ambos os tipos de CNN estudados. A leveza do modelo proposto, com menor consumo de energia e maior velocidade, o torna ideal para aplicações em tempo real em dispositivos com menor poder computacional.

6.2.2. Resultado dos algoritmos, separando-os por categoria (profundos e móveis) e treinados com a base Sisal

Os resultados da acurácia de todos os 10 métodos analisados, separados por tipo, são apresentados na Tabela 10. Além disso, os resultados de taxa de transferência (TT) e tempo de inferência (TI) são apresentados nas Tabelas 11 e 12.

Com base nos resultados obtidos, podemos observar que o modelo proposto, a FiberNet, alcançou um resultado equilibrado entre desempenho e precisão. A FiberNet está entre os três melhores resultados em ambos os modelos profundos e móveis, apenas (+2,4 p.p.) atrás do DenseNet e (+2,7 p.p.) atrás do SqueezeNet.

Ao considerarmos a diferença no número de parâmetros treináveis de cada tipo de modelo, em comparação com DenseNet (+91,16 p.p.) para modelos profundos e SqueezeNet (+39,57 p.p.) para modelos móveis, podemos afirmar que o modelo proposto alcançou um equilíbrio satisfatório entre capacidade de predição e aplicabilidade em dispositivos de baixo custo.

Tabela 10: Resultados de precisão dos modelos profundos e móveis com a base Sisal

Modelos profundos	Acurácia ▲	DP(desvio padrão)	Modelos Móveis	Acurácia ▲	DP
DenseNet	98,65%	0,010	SqueezeNet	98,96%	0,010
ResNet	98,02%	0,019	FiberNet	96,25%	0,024
FiberNet	96,25%	0,024	MobileNetV2	83,23%	0,044
GoogleNet	95,94%	0,026	ShuffleNet	81,25%	0,070
EfficientNetV2	94,79%	0,035	MobileNetV3	73,75%	0,142
EfficientNetB0	69,48%	0,109			

(V1)					

Tabela 11: Resultados de taxa de transferência (TT) e taxa de inferência (TI) para os modelos profundos com a base Sisal

	Modelos	TT ▲	DP	Modelos	TI ▼	DP
01	FiberNet	4464	18,333	FiberNet	3,89	0,047
06	ResNet	406	2,884	EfficientNetV1	29,62	1,726
07	GoogleNet	330	2,403	ResNet	37,37	0,097
08	EfficientNetV1	314	2,496	GoogleNet	44,58	1,622
09	EfficientNetV2	299	1,897	EfficientNetV2	48,22	1,052
10	DenseNet	136	0,632	DenseNet	106,72	2,667

Tabela 12: Resultados de TT e TI para os modelos móveis com a base Sisal

	Modelos	TT ▲	DP	Modelos	TI ▼	DP
01	FiberNet	4464	18,333	FiberNet	3,89	0,047
02	MobileNetV3	2586	10,382	MobileNetV3	10,66	0,590
03	ShuffleNet	720	5,417	SqueezeNet	11,45	0,151
04	SqueezeNet	561	5,034	MobileNetV2	21,19	0,871
05	MobileNetV2	428	4,320	ShuffleNet	25,26	0,910

Esta afirmação pode ser comprovada pelos resultados obtidos para os critérios de desempenho. Entre os melhores modelos de cada tipo (profundo e móvel), o modelo proposto obteve um resultado superior em termos de taxa de transferência em comparação com outros modelos: MobileNetV3 (-42,06 p.p.), SqueezeNet (-87,43 p.p.), MobileNetV2 (-90,41 p.p.), ShuffleNet (-83,87 p.p.), ResNet (-90,90 p.p.) e DenseNet (-96,95 p.p.).

A mesma observação foi feita para os resultados de tempo de inferência (IT), onde houve uma diminuição na velocidade de processamento entregue pelo FiberNet quando comparado com MobileNetV3 (+63,5 p.p.), SqueezeNet (+66,02 p.p.), MobileNetV2 (+81,64 p.p.), ShuffleNet (+84,6 p.p.), EfficientNetV1 (+86,86 p.p.) e DenseNet (+96,35 p.p.).

Para apoiar os resultados, foram realizados testes estatísticos com os resultados do conjunto de dados sisal para ambos os tipos de modelos (profundo e móvel). Como resultado, obtivemos um valor de *p* significativo abaixo de 0,05, como descrito na Tabela 13, o que levou à rejeição da hipótese nula (H0). Em seguida, procedemos com o teste não paramétrico de Nemenyi para comparar os resultados individualmente.

O teste de Nemenyi funciona comparando a diferença entre cada par de grupos a um nível de significância de 0,05. Valores iguais ou maiores que o nível de significância podem ser considerados equivalentes. Portanto, podemos concluir, com base nos resultados

apresentados nas Tabelas 14, 15 e 16, e ilustrados nas Figuras 23, 24, 25, 26, 27 e 28, que o resultado do modelo proposto, a FiberNet, é melhor ou equivalente ao melhor resultado de precisão tanto para modelos profundos quanto para móveis.

Tabela 13: Resultado do teste de Friedman para ACC, TT e TI com a base Sisal

Critério	P-Value (Profundos)	P-Value (Móveis)
Acurácia	1.02308e-05	1.01743e-06
Taxa de Transferência (TT)	1.38579e-09	4.32842e-08
Taxa de Inferência (TI)	1.38579e-09	6.09770e-08

Tabela 14: Resultados de Post-hoc da Acurácia para os modelos móveis e profundos com a base Sisal

Modelos Profundos	Post-hoc	Modelos Móveis	Post-hoc
DenseNet	1,000	SqueezeNet	1,000
ResNet	0,900	FiberNet	0,900
FiberNet	0,577	MobileNetV2	0,007
GoogleNet	0,232	ShuffleNet	0,001
EfficientNetB0 (V1)	0,232	MobileNetV3	0,001
EfficientNetV2	0,001		

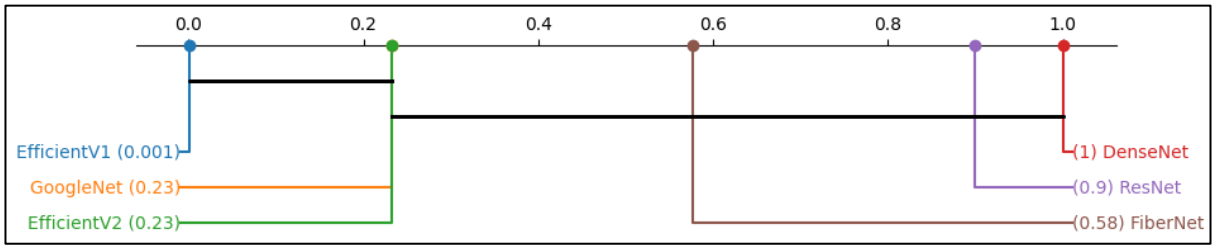


Figura 23: Gráfico da Diferença Crítica para a acurácia entre os modelos profundos

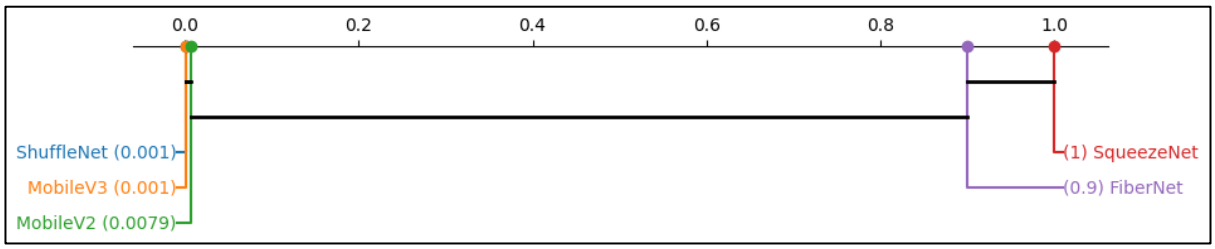


Figura 24: Gráfico da Diferença Crítica para a acurácia entre os modelos móveis

Tabela 15: Resultados do teste de Post-hoc de TT e TI para os modelos profundos com a base Sisal

Modelos Profundos	TT	Modelos Profundos	TI
FiberNet	1,000	FiberNet	1,000
ResNet	0,818	EfficientNetV1	0,818
GoogleNet	0,159	ResNet	0,159

EfficientNetV1	0,004	GoogleNet	0,004
EfficientNetV2	0,001	EfficientNetV2	0,001
DenseNet	0,001	DenseNet	0,001

Tabela 16: Resultados do teste de Post-hoc de TT e TI para os modelos móveis para a base Sisal

Modelos Móveis	TT	Modelos Móveis	TI
FiberNet	1,000	FiberNet	1,000
MobileNetV3	0,602	MobileNetV3	0,522
ShuffleNet	0,038	SqueezeNet	0,056
SqueezeNet	0,001	MobileNetV2	0,001
MobileNetV2	0,001	ShuffleNet	0,001

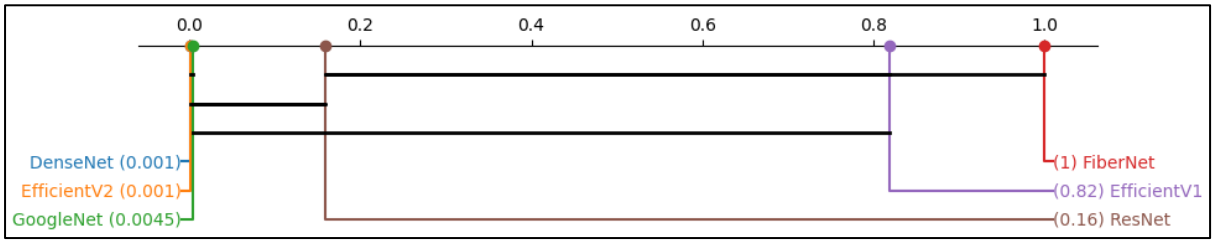


Figura 25: Gráfico da Diferença Crítica para TI entre os modelos profundos

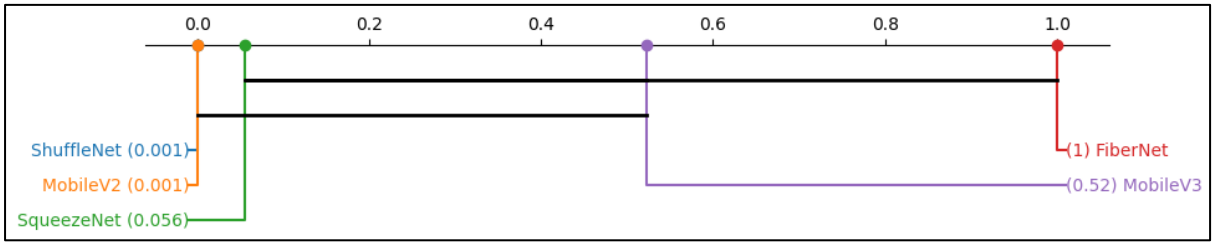


Figura 26: Gráfico da Diferença Crítica para TI entre os modelos móveis

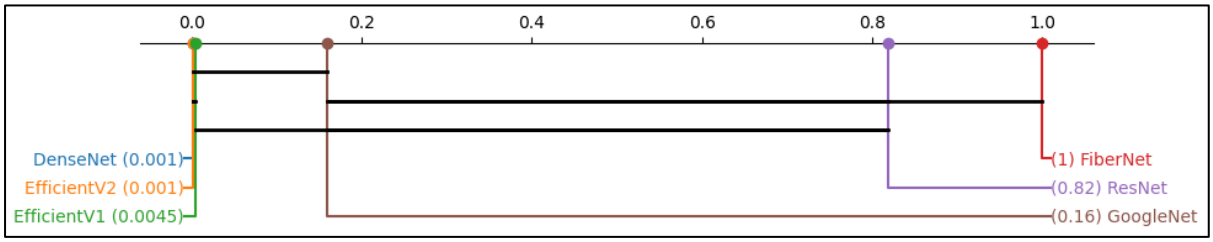


Figura 27: Gráfico da Distância Crítica para TT entre os modelos profundos

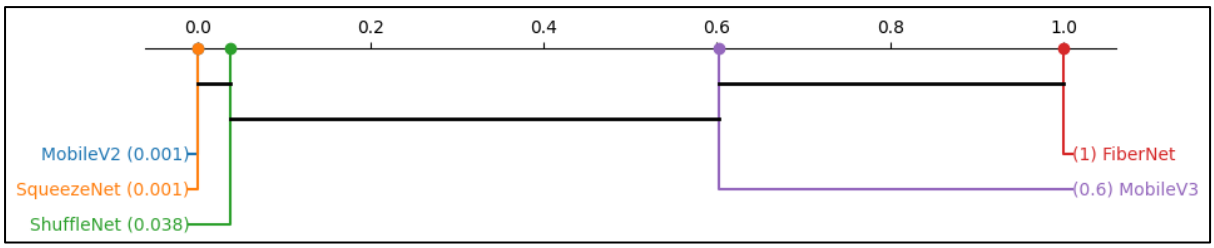


Figura 28: Gráfico da Distância Crítica para TT entre os modelos móveis

6.2.3. Resultados dos algoritmos, separando-os por categoria (profundos e móveis) e treinados com a base CIFAR10

Nesta Subseção, apresentamos os resultados dos modelos treinados no conjunto de dados CIFAR10. Neste experimento, avaliamos tanto modelos profundos quanto móveis em um conjunto de dados com um número maior de classes, o que representa um desafio maior para a discriminação de modelos.

O modelo proposto alcançou uma precisão apenas 1 p.p. atrás do melhor desempenho entre os modelos profundos, o GoogleNet. Este resultado demonstra a alta eficácia do modelo proposto e, consequentemente, da camada Defiber, alcançando uma precisão quase idêntica com 88,61 p.p. menos parâmetros treináveis.

O modelo proposto obteve a maior precisão entre os modelos móveis, superando a SqueezeNet (+16,5 p.p.) como descrito na Tabela 17. Esta diferença de desempenho é estatisticamente significativa e indica que o modelo proposto pode ser considerado mais eficaz do que o SqueezeNet neste contexto.

Este resultado se torna ainda mais significativo quando consideramos que há uma diferença de quase 40 p.p. no número de parâmetros entre os dois modelos. Esses resultados demonstram que a redução no número de parâmetros treináveis promovida pela camada Defiber não afetou a precisão do modelo proposto.

Tabela 17: Resultados da acurácia para os modelos profundos e móveis para a base CIFAR10

Modelos Profundo	Acurácia ▲	DP	Modelos Móveis	Acurácia ▲	DP
GoogleNet	75,9%	0,003	FiberNet	74,9%	0,007
FiberNet	74,9%	0,007	SqueezeNet	58,4%	0,010
ResNet	63,4%	0,008	MobileNetV3	44,3%	0,010
DenseNet	61,4%	0,007	ShuffleNet	42,5%	0,007
EfficientNetV2	57,2%	0,006	MobileNetV2	40,3%	0,009
EfficientNetV1	36,6%	0,008			

Em termos de TT e TI para modelos profundos, o modelo proposto alcançou os melhores resultados como descrito na Tabela 18. No o critério de TT, a diferença para a ResNet foi inferior a 2 p.p., o que pode ser considerado como uma margem estatisticamente irrelevante. No critério de TI, o modelo proposto foi mais rápido que o ResNet (-23,69 p.p.), o segundo colocado. Estes resultados sugerem que a camada Defiber não teve impacto no desempenho da FiberNet em termos de *hardware*.

A variação apresentada nos resultados do critério TT dos modelos profundos, para ambos os conjuntos de dados, Sisal e CIFAR10, mostrou certa disparidade em termos de distanciamento do modelo proposto para o segundo colocado.

A variação nos valores alcançados pelo modelo proposto pode sugerir que a diferença entre o número de classes em cada conjunto de dados foi o diferencial nessa questão. Ainda assim o resultado alcançado pela FiberNet continuou sendo o melhor.

Tabela 18: Resultados de TT e TI para os modelos profundos com a base CIFAR10

	Modelos Profundos	**TT ▲**	**DP**	**Modelos Profundos**	**TI ▼**	**DP**
01	FiberNet	1196	16,130	FiberNet	5,28	0,037
02	ResNet	1181	5,720	ResNet	6,92	0,069
03	GoogleNet	903	10,820	GoogleNet	16,23	0,120
04	EfficientNetV1	746	3,370	EfficientNetV1	16,73	1,258
05	DenseNet	314	1,580	DenseNet	41,41	2,239
06	EfficientNetV2	308	2,350	EfficientNetV2	43,35	1,180

Essa diferença ajudaria também a explicar a disparidade do resultado do modelo proposto nos critérios TT e TI para modelos móveis. No critério TT, o resultado da FiberNet (-53,09 p.p.) foi inferior ao resultado do primeiro colocado, a MobileNetV3. No entanto, no critério de TI, a FiberNet (+18,56 p.p.) ficou em segundo lugar atrás apenas do resultado da SqueezeNet, conforme apresentado na Tabela 19.

Tabela 19: Resultados de TT e TI para os modelos móveis com a base CIFAR10

	Modelos Móveis	**TT ▲**	**DP**	**Modelos Móveis**	**TI ▼**	**DP**
01	MobileNetV3	2550	125,340	SqueezeNet	4,30	0,059
02	ShuffleNet	1893	102,810	FiberNet	5,28	0,037
03	SqueezeNet	1290	17,860	MobileNetV3	10,20	0,158
04	FiberNet	1196	16,130	MobileNetV2	11,54	0,123
05	MobileNetV2	927	2,250	ShuffleNet	13,93	0,089

Também podemos observar, por meio desses resultados, que o modelo proposto, apesar das variações, manteve um desempenho geral muito bom. Semelhante a outros

modelos, como a SqueezeNet, por exemplo, que também sofreu variações nos resultados entre os dois conjuntos de dados.

E ao contrário de modelos como a MobileNetV3, que foi extremamente consistente nos resultados de desempenho (TT e TI), mas obteve resultados fracos em termos de precisão em ambos os conjuntos de dados. É importante destacar que o modelo proposto, a FiberNet, possui menos parâmetros treináveis do que todos os outros modelos (profundo e móvel). E isso foi alcançado através da aplicação da camada Defiber, proposta em nossa pesquisa.

Da mesma forma que fizemos para o conjunto de dados Sisal, realizamos testes estatísticos com os resultados do conjunto de dados CIFAR10. É a partir dos resultados estatísticos que conheceremos o verdadeiro grau de similaridade entre os resultados e destacaremos os modelos que podem ser considerados equivalentes.

Os resultados dos testes de Friedman estão detalhados na Tabela 20. Além disso, os resultados dos testes post-hoc para a acurácia dos modelos, tanto profundos quanto móveis, estão disponíveis na Tabela 21 e ilustrados nas Figuras 29 e 30, respectivamente.

Já os resultados dos testes post-hoc para TT e TI dos modelos profundos podem ser encontrados na Tabela 22, acompanhados pelas Figuras 31 e 32. Da mesma forma, os resultados dos testes post-hoc para a TT e a TI dos modelos móveis estão na Tabela 23, acompanhados pelas Figuras 33 e 34.

Os resultados do teste de Friedman mostraram que as médias dos valores observados para cada tratamento foram estatisticamente diferentes das médias esperadas, com um valor de $p < 0,05$. Portanto, rejeitamos a hipótese nula de que não há diferença significativa entre os tratamentos. Em seguida, realizamos o teste não paramétrico para identificar quais tratamentos diferem entre si.

Através da análise não paramétrica podemos constatar que os resultados alcançados pela FiberNet podem ser considerados como equivalente aos demais resultados, iguais ou superiores ao valor de referência, para quase todos os critérios. Tanto para os modelos considerados como profundos quanto para os modelos considerados como móveis.

Exceto na análise dos resultados da TT na categoria de dispositivos móveis. Nesse critério apenas os resultados da MobileNetV3 e a ShuffleNet podem ser considerados como equivalentes. E, de fato, nesse quesito a FiberNet apresentou um resultado inferior ao resultado da MobileNetV3 (-53,09 p.p.).

Tabela 20: Resultado do teste de Friedman para acurácia, TT e TI com a base CIFAR10

Critério	P-Value (Profundos)	P-Value (Móveis)
Acurácia	1.64209e-09	4.71778e-08
Taxa de Transferência (TT)	2.43825e-09	4.32842e-08
Taxa de Inferência (TI)	3.10549e-09	4.32842e-08

Tabela 21: Resultado do teste Post-hoc para a acurácia dos modelos profundos e móveis com a base CIFAR10

Modelos Profundos	Post-hoc	Modelos Móveis	Post-hoc
GoogleNet	1,000	FiberNet	1,000
FiberNet	0,900	SqueezeNet	0,602
ResNet	0,181	MobileNetV3	0,030
DenseNet	0,008	ShuffleNet	0,001
EfficientNetV2	0,001	MobileNetV2	0,001
EfficientNetV1	0,001		

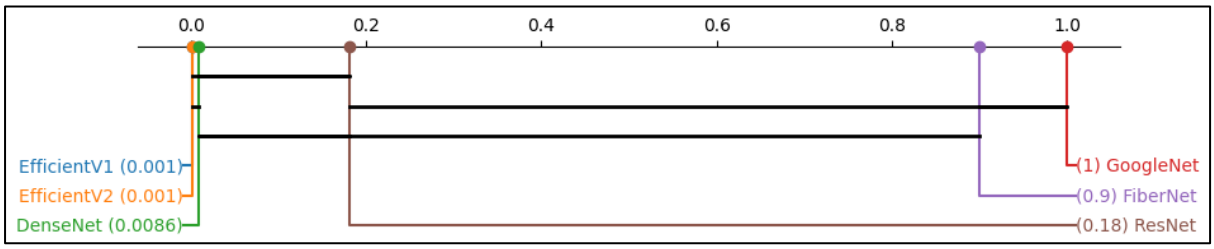


Figura 29: Gráfico da Distância Crítica para a acurácia entre os modelos profundo

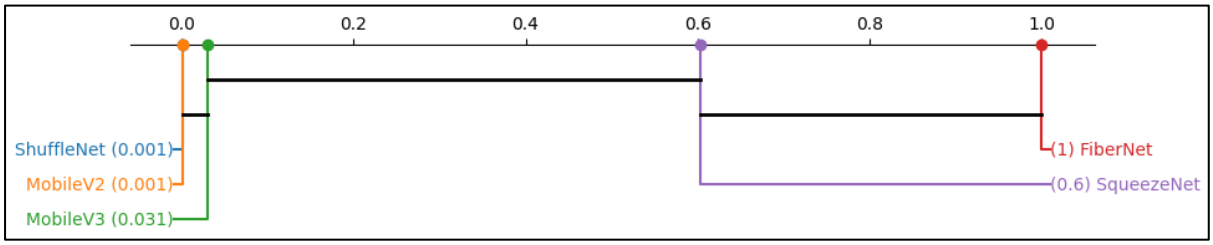


Figura 30: Gráfico da Distância Crítica para a Acurácia entre os modelos móveis

Tabela 22: Resultado do teste de Post-hoc para TT e TI dos modelos profundos com a base CIFAR10

Modelos Profundos	TT	Modelos Profundos	TI
FiberNet	1,000	FiberNet	1,000
ResNet	0,900	ResNet	0,818
GoogleNet	0,323	GoogleNet	0,066
EfficientNetV1	0,016	EfficientNetV1	0,016
DenseNet	0,001	DenseNet	0,001
EfficientNetV2	0,001	EfficientNetV2	0,001

Tabela 23: Resultado do teste de Post-hoc para TT e TI dos modelos móveis com a base CIFAR10

Modelos Móveis	TT	Modelos Móveis	TI
MobileNetV3	1,000	SqueezeNet	1,000
ShuffleNet	0,602	FiberNet	0,602
SqueezeNet	0,038	MobileNetV3	0,038
FiberNet	0,001	MobileNetV2	0,001
MobileNetV2	0,001	ShuffleNet	0,001

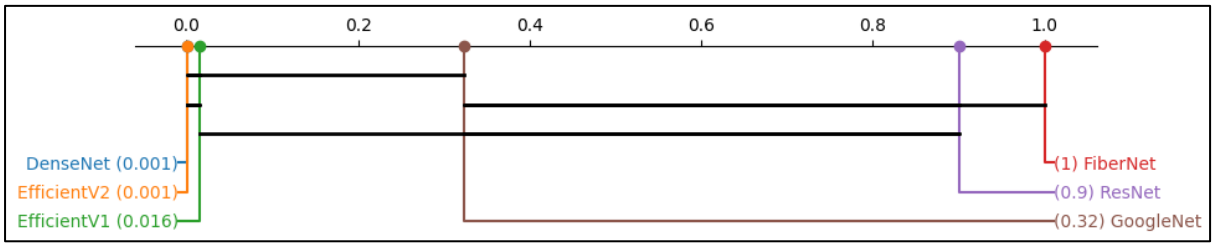


Figura 31: Gráfico da Distância Crítica para o critério de TT entre os modelos profundos

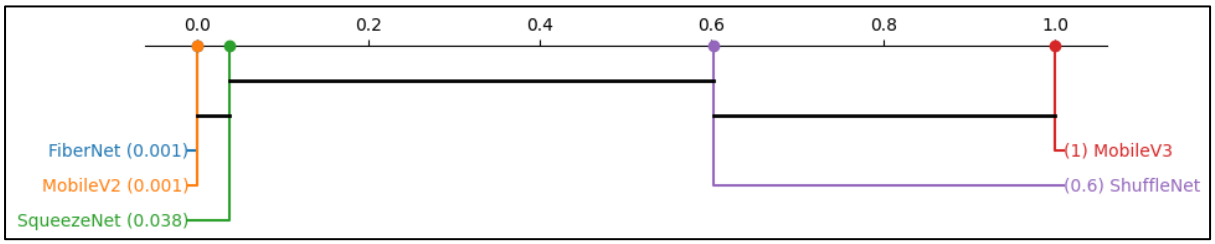


Figura 32: Gráfico da Distância Crítica para o critério de TT entre os modelos móveis

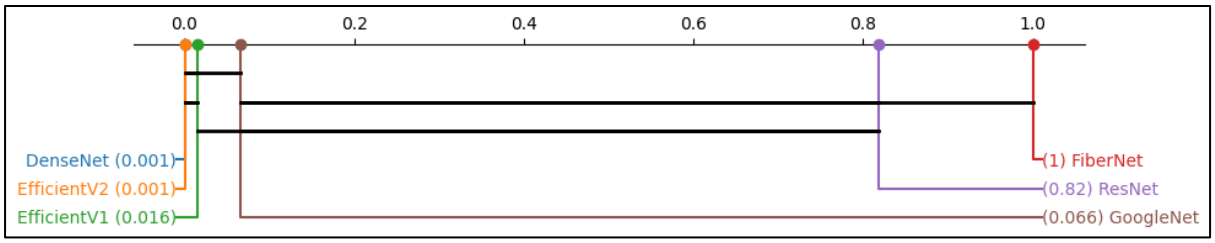


Figura 33: Gráfico da Distância crítica para o critério de TI entre os modelos profundos

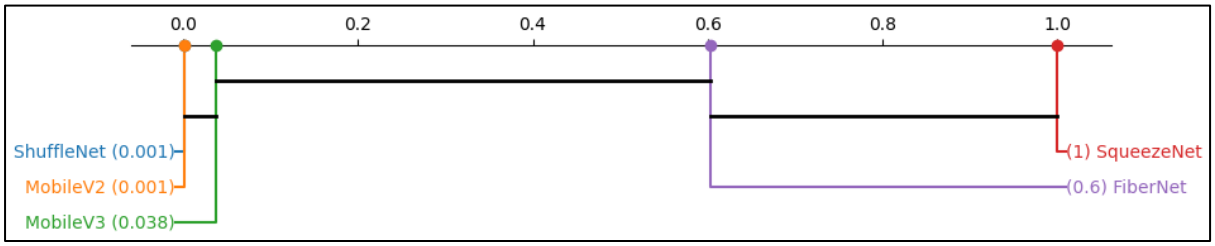


Figura 34: Gráfico da Distância Crítica para o critério de TI entre os modelos móveis

Esse resultado sugere que a arquitetura da MobileNetV3 é mais eficiente do que os demais modelos do tipo móvel nesse quesito de TT, pois ele conseguiu processar um número maior de instâncias em um determinado período de tempo. O que pode ser benéfico quando precisamos de uma arquitetura com maior capacidade de carga de processamento.

Entretanto, esse resultado contrasta com a diferença entre o resultado da FiberNet, primeiro colocado, para o resultado da MobileNetV3 (+40,85 p.p.), terceiro colocado no critério da acurácia. E, além disso, podemos citar também que a FiberNet é proporcionalmente menor do que a MobileNetV3 (-70,33 p.p.).

Esse resultado pode ser analisado também sobre uma ótica qualitativa no sentido da eficiência. Embora a FiberNet não tenha conseguido classificar tantas instâncias quanto a MobileNetV3, os resultados sugerem, entretanto, que dada a sua precisão e tamanho, a FiberNet obteve uma eficiência maior do que a MobileNetV3 no computo geral.

6.3. FASE 2: COMPARANDO OS RESULTADOS DA FIBERNET COM OUTRAS ESTRATÉGIAS DE DOWN SAMPLING

Neste experimento, comparamos os resultados de nossa proposta, a FiberNet com a camada Defiber, com outras versões da FiberNet que empregam camadas de *pooling* em vez da camada Defiber. Utilizamos os mesmos critérios de avaliação: acurácia, total de parâmetros, taxa de transferência e taxa de inferência, aplicados na Fase 1, agora também na Fase 2.

Conforme delineado no capítulo metodológico, nesta fase experimental empregamos recursos de *hardware* distintos daqueles utilizados anteriormente. Por conseguinte, os resultados obtidos nesta segunda etapa podem não ser diretamente comparáveis aos da primeira fase, uma vez que as arquiteturas foram treinadas e testadas em configurações de *hardware* diversas.

Assim sendo, começamos a análise dos resultados apresentando as arquiteturas separadas por quantidade de parâmetros treináveis. Uma vez que utilizamos a FiberNet como arquitetura padrão para os testes da Fase 2, nós adequamos o código substituindo as camadas Defiber pela quantidade correlata de camadas de *pooling* suficientes para avaliarmos em três perspectivas:

a) Manter a arquitetura com a mesma quantidade de parâmetros treináveis objetivando avaliar os modelos em condições de igualdade;

b) Aumentar moderadamente a quantidade de parâmetros treináveis da arquitetura para observar a evolução gradativa do modelo;

c) Aumentar a quantidade de parâmetros da arquitetura até atingir uma profundidade que a caracterize como um modelo profundo, permitindo uma comparação do desempenho da FiberNet, com a camada Defiber, em cenários de diferentes complexidades.

Como resultado, nós obtivemos os seguintes modelos com suas respectivas configurações descritos nas Tabelas 24 e 25. No caso específico da FiberNet com a camada Defiber, pelo fato do algoritmo utilizar uma função que ajusta automaticamente a quantidade de ativações da camada, considerou-se para efeitos de pesquisa, que cada ativação corresponderia a uma única camada Defiber.

Tabela 24: Configuração dos modelos avaliados com a base Sisal (NA=Não se Aplica) por Total de Parâmetros

Modelo	Camada (Kernel)	Total	Stride	Parâmetros	Tamanho (MB)
FiberNet (Defiber)	NA	06	NA	754.345	2,9
FiberNet (*Pooling*)	01 (02, 03) e 29 (07)	30	01	754.345	2,9
FiberNet (*Pooling*)	02 (02, 03) e 28 (07)	30	01	1.699.345	6,5
FiberNet (*Pooling*)	04 (02, 03) e 26 (07)	30	01	6.019.345	23,0

Tabela 25: Configurações dos modelos avaliados com a base CIFAR10 por Total de Parâmetros

Modelo	Camada (Kernel)	Total	Stride	Parâmetros	Tamanho (MB)
FiberNet (Defiber)	NA	01	NA	754.345	2,9
FiberNet (*Pooling*)	09 (02)	09	01	754.345	2,9
FiberNet (*Pooling*)	06 (02)	06	01	1.699.345	6,5
FiberNet (*Pooling*)	NA	NA	01	6.019.345	23,0

Dessa maneira, no de correr dessa fase da pesquisa, o valor aplicado à quantidade de camadas para o modelo da FiberNet, que utiliza a camada Defiber, será sempre correspondente à quantidade de ativações que o modelo utilizou nas tarefas de classificação propostas.

Percebe-se, de início, que há uma diferença considerável na quantidade de camadas se compararmos o modelo com a camada Defiber com os modelos que utilizam camadas de *pooling*. Essa disparidade revela que a FiberNet com a camada Defiber requer, pelo menos, 80 p.p. a menos de camadas do que qualquer um dos outros modelos comparados, que utilize camadas de *pooling*, para produzir a mesma contagem de parâmetros treináveis.

A diferença torna-se mais aparente ao compararmos os modelos testados com bancos de imagens de resoluções mais altas, como é o caso das imagens da base do Sisal. Para

alcançar um modelo proporcionalmente similar, foi necessário um número significativamente maior de camadas de *pooling*, com configurações de *kernel* mais extensas.

Essa particularidade teve um impacto direto nos resultados obtidos com essa base, sugerindo que quanto maior a resolução das imagens, maior será a quantidade necessária de camadas de *pooling* ou configurações de *kernel* para produzir um modelo equivalente à FiberNet com a camada Defiber.

6.3.1. Comparação dos resultados da FiberNet com outras estratégias de *down sampling* analisados com a base Sisal

Para uma melhor compreensão de qual modelo se trata cada um dos resultados, os resultados foram separados e nomeados a partir da seguinte configuração: FiberNet[tipo de camada]_[total de parâmetros]. Sendo que a letra "M" é utilizada para representar a escala dos milhões e a letra "K" é utilizada para representar a escala dos milhares. Dessa maneira fica mais intuitivo a identificação do resultado com a versão da FiberNet utilizada.

Em termos de acurácia, a FiberNet_Defiber apresentou o melhor desempenho, superando o segundo colocado, a FiberNetMAX_6M, em 26,72 p.p. como apresentado na Tabela 26. Além disso, o total de parâmetros da FiberNetMAX_6M é 87,46 p.p. maior do que a da FiberNet_Defiber. Esse resultado sugere que a camada Defiber, quando configurada com hiperparâmetros equivalentes, supera as camadas de *pooling* avaliadas nesse cenário.

Tabela 26: Resultados da acurácia com a base Sisal

Ordem	Modelo	Acc (%)	Desvio Padrão (DP)
1	FiberNet_Defiber	95,8	0,0204
2	FiberNetMAX_6M	70,2	0,0376
3	FiberNetAVG_6M	69,6	0,0363
4	FiberNetMAX_700K	68,4	0,0583
5	FiberNetMAX_1M	65,3	0,0365
6	FiberNetAVG_700K	60,0	0,0450
7	FiberNetAVG_1M	58,3	0,0283
8	FiberNetMIN_700K	57,2	0,0424
9	FiberNetSUM_6M	56,3	0,0481
10	FiberNetMIN_1M	54,3	0,0435
11	FiberNetSUM_1M	53,8	0,0253
12	FiberNetMIN_6M	53,5	0,0314
13	FiberNetSUM_700K	48,7	0,0645

Entre os modelos com quantidade equivalente de parâmetros treináveis houve também uma diferença significativa em relação ao modelo proposto para o quarto colocado, a FiberNetMAX_700K. A diferença de 28,60 p.p. demonstra uma efetividade maior da nossa proposta em relação à estratégia de *pooling* do tipo *max* que também é utilizada no segundo colocado desse mesmo quesito.

A efetividade do modelo proposto contrasta com todas as outras estratégias de *pooling*, principalmente devido à capacidade da camada Defiber de manter a precisão do modelo, mesmo com uma quantidade menor de parâmetros treináveis.

Em contrapartida, os resultados sugerem que os modelos que empregam estratégias de *pooling*, apesar de usarem uma quantidade significativamente maior de camadas para alcançar uma equivalência no total de parâmetros em relação ao modelo proposto, têm seu desempenho prejudicado pelo uso dessa estratégia.

Esse prejuízo em termos de desempenho se estende também para os critérios de *hardware* estabelecidos para nossa pesquisa. É o que podemos observar no critério da taxa de transferência (TT) onde a FiberNet_Defiber liderou, superando o segundo colocado, FiberNetAVG_1M, em 97,47 p.p.

É relevante destacar que, apesar da diferença de 55,60 p.p. em parâmetros treináveis entre o segundo e o primeiro colocado, o modelo proposto continuou se destacando, como evidenciado na Tabela 27.

Além disso, é importante fazer uma observação neste ponto. Conforme mencionamos no Capítulo 5, o valor apresentado nesta Tabela representa a média dos resultados obtidos em dez testes distintos com modelos treinados da FiberNet_Defiber. Houve uma oscilação significativa entre esses resultados, com alguns modelos apresentando valores superiores e outros inferiores à média geral para este critério. Essa oscilação explica o desvio padrão relativamente alto da FiberNet_Defiber em comparação aos demais modelos.

Entretanto, consideramos essa variação compreensível, dado que há uma flutuação natural na contagem de *batches* entre os modelos, refletindo tanto as diferenças no aprendizado entre modelos quanto na execução em uma GPU online do Google Colab. Apesar disso, o desempenho final da FiberNet_Defiber permaneceu significativamente superior aos demais modelos avaliados.

Tabela 27: Resultado da taxa de transferência (TT) com a base Sisal

Ordem	Modelo	Taxa de Transferência	DP
01	FiberNet_Defiber	4778	427,6008

02	FiberNetAVG_1M	121	1,5550
03	FiberNetSUM_700K	118	1,9660
04	FiberNetAVG_6M	118	3,1956
05	FiberNetAVG_700K	117	0,7868
06	FiberNetSUM_6M	117	0,6588
07	FiberNetSUM_1M	116	1,5523
08	FiberNetMAX_700K	89	0,0994
09	FiberNetMAX_1M	89	1,3650
10	FiberNetMAX_6M	89	0,9597
11	FiberNetMIN_6M	85	1,0726
12	FiberNetMIN_700K	84	0,0526
13	FiberNetMIN_1M	84	0,5605

Nessa mesma perspectiva, destaca-se que o modelo proposto alcançou o melhor desempenho em termos de taxa de inferência (TI), superando o segundo colocado, a FiberNetAVG_1M, em 51,16 p.p., conforme revelado pelos dados na Tabela 28.

Esse critério é importante, pois sugerem que o modelo proposto conseguiu manter uma boa velocidade de predição mesmo com quantidade de parâmetros totais menor. O modelo equivalente mais próximo, a FiberNetSUM_700K, terceiro colocado, foi 51,96 p.p. mais lenta em relação ao modelo proposto.

Além disso, colaborando com os resultados obtidos a partir dos experimentos da Fase 1, os novos resultados sugerem que o modelo proposto conseguiu manter sua capacidade de transitar entre os modelos que apresentam certo grau de precisão, característicos dos modelos profundos, ao mesmo tempo em que se mostrou eficaz em termos de recursos de *hardware*, uma característica típica dos modelos móveis.

Tabela 28: Resultado da taxa de inferência (TI) com a base Sisal

Ordem	Modelo	Taxa de Inferência	DP
01	FiberNet_Defiber	5,26	0,8313
02	FiberNetAVG_1M	10,77	0,2331
03	FiberNetSUM_700K	10,95	0,2555
04	FiberNetAVG_6M	11,10	0,3486
05	FiberNetAVG_700K	11,14	0,2800
06	FiberNetSUM_1M	11,16	0,3467
07	FiberNetSUM_6M	11,24	0,2335
08	FiberNetMAX_700K	13,86	0,2904
09	FiberNetMAX_6M	13,86	0,3961
10	FiberNetMAX_1M	13,98	0,3297
11	FiberNetMIN_1M	15,03	0,3539
12	FiberNetMIN_700K	15,09	0,3575
13	FiberNetMIN_6M	15,10	0,3964

Os resultados relacionados aos critérios de Taxa de Transferência (TT) e Taxa de Inferência (TI) indicam que o modelo proposto, FiberNet_Defiber, alcançou um desempenho notável, mantendo excelentes resultados nos três critérios avaliados.

Isso se contrasta, por exemplo, com o modelo DefiberSUM_700K, que utiliza *sum pooling*. Esta última figura entre os cinco melhores nos critérios TT e TI, porém, na avaliação de acurácia, encontra-se na última posição, com uma defasagem de 49,16 p.p. em relação ao resultado mais alto.

Tal comportamento sugere que, na avaliação global dos três critérios, pode não existir uma congruência entre eles. Apenas modelos que conseguem alcançar um desempenho elevado em todas as frentes são dignos de destaque. Um exemplo disso é o modelo FiberNetAVG_6M, que está entre os cinco primeiros em todos os critérios, mesmo estando 27,34 p.p. abaixo do resultado de acurácia mais elevado.

Agora que os resultados quantitativos dos modelos avaliados com o conjunto de dados do Sisal foram apresentados, prosseguiremos com a análise estatística desses dados. Iniciaremos com os resultados do teste de Friedman, aplicado aos critérios avaliados, como ilustrado na Tabela 29.

Tabela 29: Resultado do teste de Friedman para os critérios de Acurácia, TT e TI da Fase 2

Critério	Valor P
Acurácia	1.109076e-15
Taxa de Transferência (TT)	1.303252e-18
Taxa de Inferência (TI)	7.728475e-18

Os resultados obtidos indicam que os valores dos conjuntos de dados analisados são significativamente menores que o nível de significância estabelecido, que é de 0,05. Isso sugere que há uma diferença estatisticamente significativa entre os resultados. Portanto, podemos rejeitar a hipótese nula de que não há diferença entre os grupos, conforme evidenciado pela análise estatística.

Com base nesses resultados, aplicamos o teste post-hoc de Nemenyi aos conjuntos de dados. Esse teste visa determinar quais valores dos conjuntos de dados são estatisticamente significantes em comparação com um valor de referência. Os valores listados na Tabela 30 e visualizados na Figura 35 representam as classificações obtidas pelo teste de Nemenyi, utilizando como referência o melhor valor alcançado em cada critério.

Os valores obtidos a partir do teste de Nemenyi mostram o grau de equivalência entre os elementos de todos os conjuntos avaliados. Os valores obtidos mostram o grau de proximidade entre os elementos e o teste utiliza também o valor de significância de 0.05 para delimitar quais modelos são estatisticamente equivalentes independente dos valores reais de cada conjunto.

Tabela 30: Resultado post-hoc da acurácia na fase 02 com a base Sisal

Ordem	Modelo	Post-hoc Acurácia
01	FiberNet_Defiber	1,000
02	FiberNetMAX_6M	0,900
03	FiberNetAVG_6M	0,900
04	FiberNetMAX_700K	0,880
05	FiberNetMAX_1M	0,662
06	FiberNetAVG_700K	0,027
07	FiberNetAVG_1M	0,007
08	FiberNetMIN_700K	0,001
09	FiberNetSUM_6M	0,001
10	FiberNetMIN_1M	0,001
11	FiberNetSUM_1M	0,001
12	FiberNetMIN_6M	0,001
13	FiberNetSUM_700K	0,001

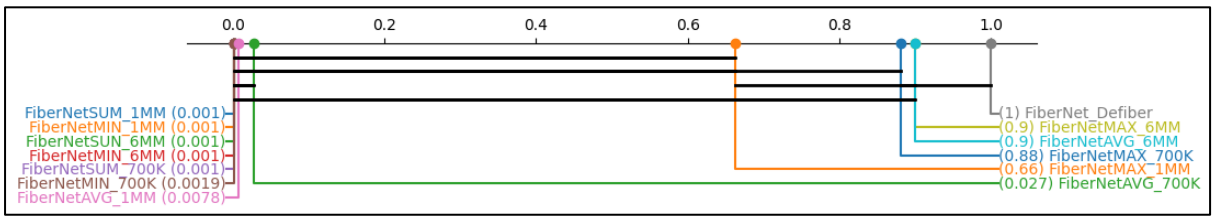


Figura 35: Gráfico da Distância Crítica para o critério da acurácia entre as versões da FiberNet

Embora haja diferenças numéricas, o teste de Nemenyi esclarece, por meio da proximidade dos valores, quais resultados são estatisticamente significativos. Valores mais próximos ao do líder da classificação (1.000) indicam uma equivalência maior entre os conjuntos avaliados.

Assim sendo, apenas os cinco primeiros resultados podem ser considerados como relevantes para o critério da acurácia. Destes, temos dois modelos com cerca de 700 mil parâmetros treináveis, que são a FiberNet_Defiber e a FiberNetMAX_700K, dois modelos com cerca de 6 milhões de parâmetros, que são a FiberNetMAX_6M e a FiberNetAVG_6M e um modelo com cerca de 1 milhão de parâmetros, que é a FiberNetMAX_1M.

A classificação destes modelos sugere novamente, que o modelo proposto, imbuído com a camada Defiber, apresenta uma acurácia típica de modelos profundos ao mesmo tempo em que apresenta também um desempenho de *hardware* compatível com modelos reconhecidamente do tipo móvel.

A capacidade do modelo proposto de transitar entre estes dois tipos de CNNs (profunda e móvel) se torna mais evidente com a continuação de nossa pesquisa e a análise dos dados de Taxa de Transferência (TT) e Taxa de Inferência (TI), coletados durante a Fase 2 utilizando o conjunto de dados do Sisal. Os detalhes desses resultados são apresentados na Tabela 31 e visualizados através das Figuras 36 e 37.

Tabela 31: Resultados post-hoc para TT e TI da fase 02 com a base Sisal

Ordem	Modelos	Post-hoc TT	Modelos	Post-hoc TI
01	FiberNet Defiber	1,000	FiberNet_Defiber	1,000
02	FiberNetAVG_1M	0,900	FiberNetAVG_1M	0,900
03	FiberNetSUM_700K	0,900	FiberNetSUM_700K	0,844
04	FiberNetAVG_6M	0,607	FiberNetAVG_6M	0,753
05	FiberNetAVG_700K	0,498	FiberNetAVG_700K	0,571
06	FiberNetSUM_6M	0,479	FiberNetSUM_1M	0,358
07	FiberNetSUM_1M	0,127	FiberNetSUM_6M	0,320
08	FiberNetMAX_700K	0,001	FiberNetMAX_700K	0,001
09	FiberNetMAX_1M	0,001	FiberNetMAX_6M	0,001
10	FiberNetMAX_6M	0,001	FiberNetMAX_1M	0,001
11	FiberNetMIN_6M	0,001	FiberNetMIN_1M	0,001
12	FiberNetMIN_700K	0,001	FiberNetMIN_700K	0,001
13	FiberNetMIN_1M	0,001	FiberNetMIN_6M	0,001

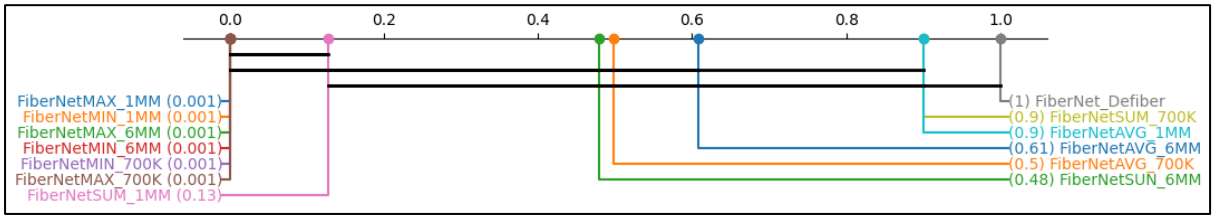


Figura 36: Gráfico da Distância Crítica do critério de TT para as versões da FiberNet

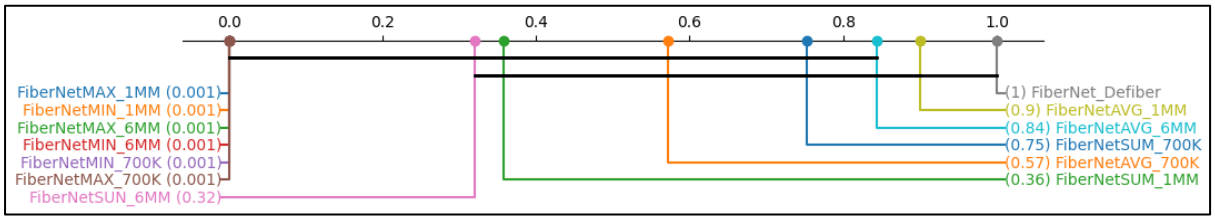


Figura 37: Gráfico da Distância Crítica do critério de TI para as versões da FiberNet

Com base nos dados da base Sisal, ficou claro que as configurações da FiberNet que incorporam as camadas Defiber, Avg_1M e Sum_700k exibiram os melhores resultados nos critérios de *hardware*. Em contraste, as variantes da FiberNet que utilizam camadas de *Min* e *Max Pooling* registraram os resultados mais baixos, não sendo consideradas estatisticamente equivalentes aos modelos de melhor desempenho.

A partir dos resultados obtidos nestes critérios podemos observar que o modelo proposto além de obter o melhor resultado também apresentou estatisticamente uma equivalência com outros modelos com quantidades de parâmetros treináveis variáveis. Essa equivalência reforça, a partir destes outros dois critérios, a ideia anterior de que o modelo proposto, configurado com a camada Defiber, pode assumir características destes dois tipos de CNN.

6.3.2. Comparando os resultados da FiberNet com outras estratégias de *down sampling* analisados com a base CIFAR10

Avançamos agora para a análise dos resultados derivados dos modelos treinados e testados utilizando a base de imagens CIFAR10. No critério de acurácia, o modelo proposto, FiberNet_Defiber, se posicionou como o quarto melhor, ficando a uma distância de aproximadamente 3,5 p.p. do líder, que foi o modelo FiberNetSemCamada_6M.

Aqui é importante fazermos uma breve observação. Por conta da característica específica da base CIFAR10, de fornecer imagens com apenas 32x32 de resolução, não foi possível ajustar a configuração das camadas de *pooling* para o total de 6 milhões de parâmetros (6M). Só foi possível alcançarmos esse patamar a partir de uma versão da FiberNet sem camadas de *pooling* ou Defiber. Por essa razão, estamos utilizando a versão "FiberNetSemCamada_6M" como representante geral desta arquitetura.

Continuando, esse desempenho adquire uma perspectiva mais clara quando observamos a diferença no número de parâmetros treináveis entre os dois modelos. A FiberNet_Defiber (-87,46 p.p.) possui menos parâmetros do que o modelo de melhor desempenho, a FiberNetSemCamada_6M, o que proporcionalmente representa uma diferença significativa.

Vale destacar que outros modelos também atingiram resultados semelhantes nesse aspecto. Pois a distância relativa entre o primeiro colocado e o oitavo melhor colocado é de aproximadamente 6 p.p. como pode ser observado na Tabela 32.

Tabela 32: Resultado da acurácia para as versões da FiberNet para os dados da base CIFAR10

Ordem	Modelos	ACC (%)	DP
01	FiberNetSemCamada_6M	77,0	0,0132
02	FiberNetMAX_1M	75,5	0,0074
03	FiberNetAVG_1M	75,0	0,0032
04	FiberNet_Defiber	75,0	0,0052
05	FiberNetMAX_700K	74,0	0,0063
06	FiberNetAVG_700K	73,0	0,0070
07	FiberNetSUM_1M	72,0	0,0191
08	FiberNetSUM_700K	71,0	0,0042
09	FiberNetMIN_1M	68,0	0,0067
10	FiberNetMIN_700K	66,0	0,0052

Um ponto interessante a se observar são os resultados dos modelos com 700k de parâmetros treináveis, mesma categoria da FiberNet_Defiber. Nessa perspectiva, o modelo proposto apresentou o melhor desempenho, seguido de perto pelo modelo FiberNetMAX_700K, com uma diferença de 1 p.p.

Sendo que, entre os modelos com 700K de parâmetros, a FiberNet_Defiber obteve o melhor resultado, mesmo utilizando menos camadas do que os modelos de 700K (-88,88 p.p.) configurados com camadas de *pooling*, como informado anteriormente na Tabela 25.

Este resultado é significativo, pois evidencia a eficácia de nossa abordagem no contexto geral da pesquisa, ao oferecer uma camada redutora de parâmetros que, conforme sugerem os resultados, não compromete a capacidade preditiva do modelo.

Além disso, outros dados que colaboram com essa afirmação são os resultados obtidos a partir dos critérios de *hardware*. Nas Tabelas 33 e 34 nós apresentamos os resultados para os critérios de TT e TI testados a partir das imagens da base CIFAR10.

Tabela 33: Resultados para o critério de TT na Fase 2 com os dados da CIFAR10

Ordem	Modelos	TT	DP
01	FiberNet Defiber	8843	476,9771
02	FiberNetMAX_700K	7828	265,3317
03	FiberNetSUM_700K	7718	525,1012
04	FiberNetSUM_1M	7623	231,0055
05	FiberNetAVG_700K	7574	540,9691
06	FiberNetAVG_1M	7459	454,9905
07	FiberNetMAX_1M	7333	517,2198
08	FiberNet_SemCamada_6M	7159	653,2651
09	FiberNetMIN_700K	6782	317,8295
10	FiberNetMIN_1M	6741	358,4063

Tabela 34: Resultado para o critério de TI da Fase 2 com a base CIFAR10

Ordem	Modelos	TI	DP
01	FiberNetSUM_700K	2,38	0,2521
02	FiberNet Defiber	2,46	0,3853
03	FiberNetMAX_700K	2,47	0,2659
04	FiberNetAVG_700K	2,55	0,3020
05	FiberNetAVG_1M	2,56	0,2077
06	FiberNet_SemCamada_6M	2,63	0,2306
07	FiberNetMAX_1M	2,67	0,3169
08	FiberNetMIN_700K	2,83	0,3497
09	FiberNetMIN_1M	2,94	0,3527
10	FiberNetSUM_1M	3,04	1,2063

Nesses critérios, observamos uma predominância dos modelos com 700k de parâmetros treináveis, onde o modelo proposto, a FiberNet_Defiber, se destacou entre os três melhores resultados em ambos os critérios. No critério de TT, o modelo proposto obteve o melhor resultado, enquanto no critério de TI alcançou o segundo lugar, ficando apenas a menos de 3 p.p. de diferença para o primeiro colocado, a FiberNetSUM_700K.

Observamos que, nesse sentido, há uma proximidade nos resultados obtidos para o critério de TI, enquanto que para o critério de TT o modelo proposto obteve uma vantagem de 11 p.p. sobre o segundo colocado, a FiberNetMAX_700K. Essa proximidade nos resultados, para os critérios de TI e TT, entre os modelos da categoria de 700K de parâmetros é compreensível, visto que esses modelos menores tendem a ser mais rápidos.

Para analisarmos com mais propriedade os resultados obtidos, vamos agora apresentar os resultados a partir da análise estatística. Na Tabela 35 apresentamos o resultado do teste de Friedman aplicado aos resultados dos critérios da acurácia, TT e TI com os dados da base CIFAR10.

De forma semelhante ao observado com a base Sisal, os dados derivados da base CIFAR10 também apresentaram valores inferiores ao limiar de significância definido, estabelecido em 0,05. Essa constatação indica a existência de uma diferença estatisticamente significativa entre os resultados analisados.

Tabela 35: Resultado do teste de Friedman para os dados da Fase 2 com a base CIFAR10

Critério	Valor P
Acurácia	3.361334e-14
Taxa de Transferência (TT)	9.122298e-09
Taxa de Inferência (TI)	0.0013086708

Dessa forma, podemos rejeitar a hipótese nula de que não há diferença entre os conjuntos de dados, conforme evidenciado pela análise estatística. Em seguida, aplicamos o teste post-hoc de Nemenyi para analisar os conjuntos de dados e determinar o grau de equivalência entre eles. Os resultados do teste post-hoc com os dados da acurácia com a base CIFAR10 são apresentados na Tabela 36 e podem ser visualizados também através da Figura 38.

Tabela 36: Resultado post-hoc para o critério da acurácia na Fase 2 com a base CIFAR10

Ordem	Modelos	Post-hoc Acurácia
01	FiberNetSemCamada_6M	1,000
02	FiberNetMAX_1M	0,900
03	FiberNetAVG_1M	0,900
04	FiberNet_Defiber	0,724
05	FiberNetMAX_700K	0,449
06	FiberNetAVG_700K	0,007
07	FiberNetSUM_1M	0,002
08	FiberNetSUM_700K	0,001
09	FiberNetMIN_1M	0,001
10	FiberNetMIN_700K	0,001

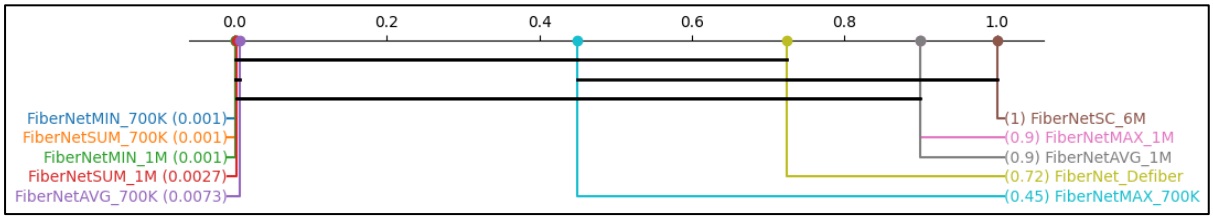


Figura 38: Gráfico da Distância Crítica para a acurácia entre as versões da FiberNet

A análise post-hoc indica que, entre os modelos avaliados quanto à acurácia, apenas os cinco primeiros resultados se destacam como estatisticamente significativos, superando o limiar de significância de 0,05. Esta análise leva em consideração a comparação com o desempenho superior alcançado pelo modelo FiberNetSemCamada_6M.

De maneira similar ao que foi observado com a base Sisal, constatamos que o desempenho do modelo proposto, FiberNet_Defiber, alinha-se aos modelos que operam na mesma escala de 1 milhão de parâmetros. Isso se estende também à comparação com o líder de performance, FiberNetSemCamada_6M, e à variante com 700k de parâmetros, FiberNetMAX_700K.

Esta equivalência sugere, assim como observado com a base Sisal, que o modelo proposto, equipado com a camada Defiber, alia a precisão elevada dos modelos de categoria profunda à performance otimizada de hardware característica dos modelos móveis. Dando continuidade à análise, observamos agora os resultados destes modelos à luz da análise estatística para os critérios de TT e TI com a base CIFAR10 e apresentados na Tabela 37 e podem ser visualizadas nas Figuras 39 e 40.

Tabela 37: Resultado post-hoc para os critérios de TT e TI dos resultados da Fase 2 com a base CIFAR10

Nº	Modelo	TT	Nº	Modelos	TI
01	FiberNet Defiber	1,000	01	FiberNetSUM_700K	1,000
02	FiberNetMAX_700K	0,861	02	FiberNet Defiber	0,900
03	FiberNetSUM_700K	0,633	03	FiberNetMAX_700K	0,900
04	FiberNetSUM_1M	0,133	04	FiberNetAVG_700K	0,900
05	FiberNetAVG_700K	0,133	05	FiberNetAVG_1M	0,883
06	FiberNetAVG_1M	0,060	06	FiberNet_SemCamada_6M	0,656
07	FiberNetMAX_1M	0,001	07	FiberNetMAX_1M	0,423
08	FiberNet_SemCamada_6M	0,001	08	FiberNetMIN_700K	0,082
09	FiberNetMIN_700K	0,001	09	FiberNetMIN_1M	0,001
10	FiberNetMIN_1M	0,001	10	FiberNetSUM_1M	0,001

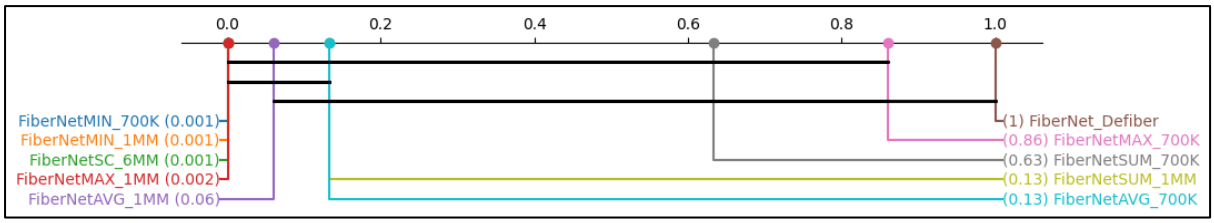


Figura 39: Gráfico da Distância Crítica do critério de TT entre as versões da FiberNet com a base CIFAR10

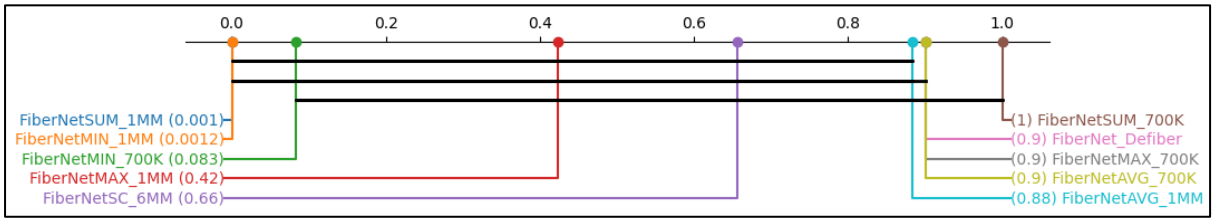


Figura 40: Gráfico da Distância Crítica do critério de TI entre as versões da FiberNet com a base CIFAR10

De maneira distinta aos resultados dos modelos com a base Sisal, observamos para os resultados com a base CIFAR10 um comportamento de equivalência maior entre os

resultados de modelos considerados como móveis. Esse comportamento indica uma qualidade técnica do modelo proposto com relação à seu desempenho de *hardware*.

Considerando a variação no grau de equivalência, que varia de 0,05 a 1,000, que define a magnitude das equivalências entre os conjuntos de dados, podemos afirmar que pelo menos três conjuntos de dados na faixa de 700K de parâmetros foram considerados equivalentes em ambos os critérios: FiberNet_Defiber, FiberNetMAX_700K e FiberNetSUM_700K.

Além destes, os resultados alcançados pelo modelo FiberNetAVG_700K nos dois critérios o posicionam no mesmo grupo, pois seus resultados, em ambos os critérios, estão acima de 0,05. No entanto, é importante observar que o grau de equivalência no critério de TT o distancia um pouco dos demais modelos considerados melhores.

O mesmo princípio se aplica aos modelos com 1 milhão de parâmetros treináveis. Nesse aspecto, os resultados dos modelos FiberNetSUM_1M, FiberNetAVG_1M e FiberNetMAX_1M também foram considerados estatisticamente equivalentes em pelo menos um dos critérios analisados. Embora alguns resultados possam não estar tão próximos, esses modelos obtiveram resultados logo abaixo dos três melhores modelos, que estão na faixa dos 700k de parâmetros.

Confirmando nossas observações anteriores, o modelo proposto continuou demonstrando uma equivalência estatística com os resultados de maiores destaques. Notamos que essa equivalência persiste quando comparamos a FiberNet àqueles caracterizados por sua profundidade quanto àqueles modelos caracterizados pelo seu desempenho de *hardware*. Essa constância na equivalência reforça a capacidade do modelo proposto, especialmente com a inclusão da camada Defiber, de aliar as principais vantagens de ambos os tipos de arquiteturas.

6.4. ANÁLISE GERAL DOS RESULTADOS

Durante nossa pesquisa, conduzimos uma análise comparativa entre nossa arquitetura, a FiberNet, e outras 25 arquiteturas por meio dos experimentos planejados. Isso incluiu modelos amplamente reconhecidos por sua excelência técnica, bem como diferentes variantes da FiberNet que incorporam metodologias distintas de *down sampling*, igualmente reconhecidas por sua relevância científica. A síntese do ranqueamento da FiberNet em todos os experimentos realizados encontra-se descrita na Tabela 38.

Tabela 38: Posição geral da FiberNet nos resultados dos experimentos realizados

Experimento	Critério	Base de imagem	Posição
Comparação arquiteturas	Total de Parâmetros	Geral	1
Comparação tipo (*deep*)	Acurácia	Sisal	3
Comparação tipo (*deep*)	TT	Sisal	1
Comparação tipo (*deep*)	TI	Sisal	1
Comparação tipo (*mobile*)	Acurácia	Sisal	2
Comparação tipo (*mobile*)	TT	Sisal	1
Comparação tipo (*mobile*)	TI	Sisal	1
Comparação tipo (*deep*)	Acurácia	CIFAR10	2
Comparação tipo (*deep*)	TT	CIFAR10	1
Comparação tipo (*deep*)	TI	CIFAR10	1
Comparação tipo (*mobile*)	Acurácia	CIFAR10	1
Comparação tipo (*mobile*)	TT	CIFAR10	4
Comparação tipo (*mobile*)	TI	CIFAR10	2
Comparação *down sampling*	Acurácia	Sisal	1
Comparação *down sampling*	TT	Sisal	1
Comparação *down sampling*	TI	Sisal	1
Comparação *down sampling*	Acurácia	CIFAR10	4
Comparação *down sampling*	TT	CIFAR10	1
Comparação *down sampling*	TI	CIFAR10	2

Nesta pesquisa, quatro critérios foram avaliados em cada experimento: (a) total de parâmetros treináveis, (b) acurácia, (c) taxa de transferência e (d) taxa de inferência, sendo aplicados a todos os modelos avaliados em duas fases distintas. Dos resultados obtidos, a FiberNet se destacou em 12 dos 19 experimentos (63,15 p.p.), enquanto nos demais experimentos ela se posicionou, pelo menos, entre os quatro primeiros colocados.

Os resultados indicam que o sucesso do nosso modelo se deve, entre outros fatores, à integração da camada Defiber, que possibilitou à nossa arquitetura operar com um número reduzido de parâmetros treináveis. Além disso, a capacidade do modelo de ajustar automaticamente a quantidade de parâmetros treináveis em função da resolução das imagens analisadas foi fundamental para proporcionar uma performance equilibrada, garantindo o cumprimento dos nossos objetivos.

Outro ponto importante foi a combinação eficaz dos hiperparâmetros, como o tamanho do kernel nas camadas de convolução, que desempenhou um papel determinante na extração e reconhecimento das características presentes nas imagens processadas.

Além disso, os resultados apontam que a FiberNet foi, pelo menos, equivalente na maioria dos resultados pelo critério da acurácia, enquanto observamos também que nosso algoritmo figurou entre os melhores resultados pelos critérios de desempenho de *hardware*.

Isso nos permite afirmar que nosso algoritmo conseguiu combinar a melhor característica de ambos os tipos de algoritmos (profundos e móveis). Ele se aproximou dos resultados superiores de acurácia, característicos de algoritmos de CNN profundas, ao mesmo tempo que demonstrou um excelente desempenho de *hardware*, o que é típico de algoritmos de CNN otimizados para dispositivos móveis.

6.5. CONSIDERAÇÕES FINAIS

No decorrer deste capítulo, apresentamos os resultados de nossa pesquisa, os quais foram obtidos por meio de experimentos conduzidos com nosso protótipo a FiberNet, que incorpora a camada Defiber em sua arquitetura. Demonstramos a eficácia de nossa abordagem ao analisar os resultados obtidos com uma base de imagens própria, a base Sisal, e uma base de imagens clássica e mundialmente utilizada, a CIFAR10.

Nosso experimento foi conduzido em duas fases. Na primeira, nós comparamos o resultado do modelo proposto a partir de um processo comparativo com outras arquiteturas mundialmente reconhecidas por sua capacidade preditiva.

Além disto, algumas destas arquiteturas foram escolhidas também por possuírem uma grande quantidade de parâmetros treináveis, tornando-as familiar ao conceito de modelos do tipo profundo, ou por apresentarem um excelente desempenho de hardware tornando-as assim familiar ao conceito de modelos do tipo móvel.

A partir deste experimento conseguimos demonstrar a capacidade do modelo proposto de alcançar resultados tão bons quanto os modelos profundos como também conseguimos demonstrar a viabilidade da nossa proposta como um modelo apto a ser utilizado em dispositivos móveis. Restou provado também que esta qualificação foi possível de ser alcançada graças à utilização de nossa inovação, a camada Defiber.

Na segunda fase, aprofundamos ainda mais nossos experimentos a fim de demonstrar a viabilidade de nossa proposta. Nesse estágio, adotamos a arquitetura FiberNet como

plataforma de investigação para explorar o desempenho da nossa inovação, a camada Defiber, em comparação com outras camadas dedicadas à redução de parâmetros.

Engajamos neste processo quatro variantes de camadas de *pooling:* MAX, MIN, SUM e AVG, reconhecidas como estratégias de *down sampling*, para atuar como *benchmarks* comparativos. Utilizando as variantes da FiberNet equipadas com camadas de pooling, realizamos uma comparação entre o modelo proposto, agora denominado FiberNet_Defiber, e outras variações que possuem diferentes quantidades de parâmetros treináveis.

Os resultados obtidos evidenciam a relevância de nossa proposta em aspectos específicos, particularmente nos critérios de taxa de transferência e taxa de inferência, utilizando a base de dados de imagens Sisal, em ambas as fases de nossa investigação.

Ao prosseguir com a análise usando a base CIFAR10, caracterizada por uma maior complexidade na classificação de imagens, o modelo proposto conseguiu manter surpreendentemente um desempenho equiparável às demais variantes da FiberNet e demonstrou ser superior em comparação a várias outras arquiteturas consagradas daquelas que foram utilizadas em comparativo na primeira fase.

CAPÍTULO 7.

CONCLUSÃO

Em nosso estudo, apresentamos o algoritmo FiberNet, um algoritmo de CNN que se destaca pela simplicidade e robustez. Ela foi concebida para funcionar como arquitetura base de uma nova camada através do qual utilizamos para reduzir o quantitativo de parâmetros treináveis do modelo. Essa nova camada, opera reduzindo as imagens antes das camadas de convolução e dessa estratégia resulta uma arquitetura otimizada com desempenho e eficiência.

Nossa camada, denominada Defiber, opera reduzindo o tamanho da imagem a partir da sua estrutura mais básica, os pixels. Iniciado o processo de convolução a imagem é convertida em uma matriz de dados que possui eixos nas dimensões da altura e largura.

O algoritmo proposto analisa e reduz proporcionalmente, a partir de um valor limite estabelecido, linhas inteiras de pixels da imagem em ambas as dimensões. Esta redução gera uma imagem proporcionalmente menor, mantendo, contudo, dados suficientes para que o modelo possa realizar inferências eficazes.

Em nossa pesquisa, empregamos um algoritmo de CNN, um tipo específico de rede neural, projetado para a classificação de imagens, fazendo uso de dois bancos de imagens distintos para a análise.

O primeiro deles é um banco de imagens do Sisal, cientificamente conhecido como Agave Sisalana, que é uma planta característica das regiões semiáridas do Nordeste do Brasil e é amplamente utilizada como forma de subsistência econômica e geração de renda.

O segundo banco de imagens utilizado foi a CIFAR10. Um banco de imagens clássico que já é utilizado pela comunidade científica como base para testes de diversos modelos. Ela é composta por aproximadamente 60.000 imagens divididas em dez classes diferentes. Dentre as classes que a compõem temos imagens de aviões, caminhões, animais, objetos, dentre outros.

Para avaliar a viabilidade do método proposto, foi realizada uma análise empírica. Por meio dessa análise, a CNN proposta neste trabalho apresentou 754,345 parâmetros treináveis e, na primeira fase, foi capaz de classificar a planta de sisal com uma precisão de 96,25%.

E através dos testes com a base CIFAR10 a FiberNet obteve o segundo melhor resultado de precisão, perdendo para o GoogleNet pela diferença de apenas 1 p.p. No entanto,

observamos que a FiberNet é muito mais eficiente que o GoogleNet em termos de critérios como TT (taxa de transferência), TI (taxa de inferência) e parâmetros treináveis.

Na segunda fase, realizamos uma análise comparativa envolvendo a arquitetura FiberNet configurada com a camada Defiber e outras variantes da mesma arquitetura, desta vez configuradas com outros quatro tipos de camadas de *pooling* (*averange*, *maxpooling*, *minpooling* e *sumpooling*).

Como resultado desse segundo experimento nós obtivemos um resultado similar aos da primeira fase. O modelo proposto alcançou um resultado de 95,8% de acurácia com a base Sisal e 75,0% com a base CIFAR10.

Além disso, o modelo proposto alcançou valores excepcionais no desempenho de *hardware* com ambas as bases de dados. Isso demonstra que o modelo proposto, equipado com a camada Defiber, superou as variantes da FiberNet que incorporam camadas de *pooling*, em grande parte dos critérios avaliados.

Com base nas análises e descobertas apresentadas nesta pesquisa, é evidente que o desenvolvimento e implementação da camada Defiber em nossa arquitetura, a FiberNet, constitui um avanço significativo na otimização de Redes Neurais Convolucionais (CNNs) para o processamento e classificação de imagens.

A inclusão desta camada inovadora demonstrou não apenas a capacidade de reduzir o número de parâmetros treináveis sem comprometer a acurácia do FiberNet, mas também de garantir uma performance comparável, e em muitos aspectos superior, às arquiteturas clássicas. Tais resultados reafirmam a relevância da pesquisa em busca de modelos mais eficientes e acessíveis, capazes de operar com alta precisão em diferentes conjuntos de dados e complexidades de tarefas.

7.1. TRABALHOS FUTUROS

A convergência entre a eficiência computacional e a manutenção de altos níveis de desempenho, conforme demonstrado nesta pesquisa, sugere um vasto território a ser explorado para aprimoramentos futuros.

O sucesso da nossa proposta abre novos caminhos para pesquisas subsequentes, especialmente na exploração de técnicas avançadas de otimização e na ampliação da aplicabilidade de modelos leves a partir da reconfiguração ou exploração de novos limites para a camada Defiber.

Assim como o surgimento de novas arquiteturas de configurações simples, porém eficazes, ou da própria adaptação da camada Defiber à arquiteturas já estabelecidas que possam se beneficiar das melhorias apresentadas neste trabalho. Como, por exemplo, investigarmos a possibilidade de inclusão da Camada Defiber em módulos *Fire* da SqueezeNet ou da incorporação da nossa camada na arquitetura *bottleneck* da MobileNet.

Uma outra perspectiva promissora que consideramos é a descoberta de novas combinações de kernel dentro da própria FiberNet, para aplicar a técnica residual. Embora não tenha sido empregada neste estágio da pesquisa, essa técnica continua sendo valiosa e pode se tornar um elemento potencializador para a criação de arquiteturas mais robustas e eficientes.

Portanto, espera-se que os *insights* e metodologias propostas aqui inspirem outras investigações, contribuindo para a evolução contínua no campo da Visão Computacional e do Aprendizado de Máquina.

REFERÊNCIAS

ABU-SAQER, Mohammed M.; ABU-NASER, Samy S.; AL-SHAWWA, Mohammed O. Type of Grapefruit Classification Using *Deep learning*. International Journal of Academic Information Systems Research (IJAISR), v. 4, n. 1, 2020.

AL-DAOUR, Ahmed F.; AL-SHAWWA, Mohammed O.; ABU-NASER, Samy S. Banana classification using *deep learning*. International Journal of Academic Information Systems Research (IJAISR), v. 3, n. 12, 2020.

ALSHAWWA, Izzeddin A. et al. Analyzing Types of Cherry Using *Deep learning*. International Journal of Academic Engineering Research (IJAER), v. 4, n. 1, 2020.

AL-SHAWWA, Mohammed O. Classification of apple fruits by *deep learning*. 2020.

BENGIO, Yoshua et al. Learning deep architectures for AI. Foundations and trends® in *Machine Learning*, v. 2, n. 1, p. 1-127, 2009.

BIEDERMAN, Irving. Recognition-by-components: a theory of human image understanding. Psychological review, v. 94, n. 2, p. 115, 1987.

BISHOP, Christopher M.; NASRABADI, Nasser M. Pattern recognition and *Machine Learning*. New York: Springer, 2006.

CAVALCANTE, Gustavo Teotônio de Oliveira et al. Diagnóstico dos Principais Indicadores Socioambientais do Sisal no Recorte Geográfico de Pocinhos/PB. 2021.

CHOLLET, Francois. *Deep learning* with Python. Simon and Schuster, 2021.

CUN, Y. L. et al. Efficient backprop, neural networks: Tricks of the Trade. Lecture Notes in Computer Sciences, v. 1524, p. 5-50, 1998.

DOLPHIN, R. Overfitting in ML: Understanding and Avoiding the Pitfalls. Disponível em: <https://towardsdatascience.com/overfitting-in-ml-avoiding-the-pitfalls-d5225b7118d>. Acessado em: 28 de Agosto de 2023.

DOMINGOS, Pedro. A few useful things to know about machine learning. Communications of the ACM, v. 55, n. 10, p. 78-87, 2012.

DOMINGOS, Pedro. The master algorithm: How the quest for the ultimate learning machine will remake our world. Basic Books, 2015.

DONG, Chao et al. Learning a deep convolutional network for image super-resolution. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part IV 13. Springer International Publishing, 2014. p. 184-199.

FLACH, Peter. *Machine Learning*: The art and science of algorithms that make sense of *data*. Cambridge University Press, 2012.

FRIEDMAN, Milton. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the american statistical association, v. 32, n. 200, p. 675-701, 1937.

GEIFMAN, A. The Correct Way to Measure Inference Time of Deep Neural Networks. Disponível em: <https://towardsdatascience.com/the-correct-way-to-measure-inference-time-of-deep-neural-networks-304a54e5187f>. Acesso em: 12 dez. 2023.

GÉRON, Aurélien. Hands-on *Machine Learning* with Scikit-Learn, Keras, and TensorFlow. "O'Reilly Media, Inc.", 2022.

GOMES, D. dos S. Inteligência Artificial: Conceitos e aplicações. Olhar Científico. v1, n. 2, p. 234-246, 2010.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep learning*. MIT press, 2016.

GOODFELLOW, Ian et al. Generative adversarial networks. Communications of the ACM, v. 63, n. 11, p. 139-144, 2020.

HAHNLOSER, Richard HR et al. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature, v. 405, n. 6789, p. 947-951, 2000.

HASSAN, Sk Mahmudul et al. Identification of plant-leaf diseases using CNN and transfer-learning approach. Electronics, v. 10, n. 12, p. 1388, 2021.

HE, Kaiming et al. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 770-778.

HE, Yi; LI, Tianli. A lightweight CNN model and its application in intelligent practical teaching evaluation. In: MATEC Web of Conferences. EDP Sciences, 2020. p. 05016.

HINTON, Geoffrey. Neural networks for machine learning, lecture 15a. 2012.

HINTON, Geoffrey; SRIVASTAVA, Nitish; SWERSKY, Kevin. Lecture 6a overview of mini–batch gradient descent. Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online, 2012.

HOWARD, Andrew et al. Searching for Mobilenetv3. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019. p. 1314-1324.

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

HUANG, Gao et al. Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. p. 4700-4708.

IANDOLA, Forrest N. et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. arXiv preprint arXiv:1602.07360, 2016.

IOFFE, Sergey; SZEGEDY, Christian. *Batch normalization*: Accelerating deep network training by reducing internal covariate shift. In: International Conference on *Machine Learning*. pmlr, 2015. p. 448-456.

JOCHER, Glenn et al. ultralytics/yolov5: v3. 0. Zenodo, 2020.

KARL PEARSON, F. R. S. Mathematical contributions to the theory of evolution. Dulau and Co, 1904.

KINGMA, Diederik P. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

KRIZHEVSKY, Alex. One weird trick for parallelizing convolutional neural networks. arXiv preprint arXiv:1404.5997, 2014.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. Communications of the ACM, v. 60, n. 6, p. 84-90, 2017.

KUMAR, Sumit et al. Plant disease detection using CNN. Turkish Journal of Computer and Mathematics Education (TURCOMAT), v. 12, n. 12, p. 2106-2112, 2021.

LECUN, Yann et al. Backpropagation applied to handwritten zip code recognition. Neural computation, v. 1, n. 4, p. 541-551, 1989.

LECUN, Yann et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, v. 86, n. 11, p. 2278-2324, 1998.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. *Deep learning*. Nature, v. 521, n. 7553, p. 436-444, 2015.

LEI, Fangyuan et al. Shallow convolutional neural network for image classification. SN Applied Sciences, v. 2, p. 1-8, 2020.

LIU, Huajie et al. Lightweight convolutional neural network for counting densely piled steel bars. Automation in Construction, v. 146, p. 104692, 2023.

LONG, Jonathan; SHELHAMER, Evan; DARRELL, Trevor. Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. p. 3431-3440.

MA, Ningning et al. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European Conference on Computer Vision (ECCV). 2018. p. 116-131.

METTLEQ, Alaa Soliman Abu et al. Mango classification using *deep learning*. International Journal of Academic Engineering Research (IJAER), v. 3, n. 12, 2020.

MICHELUCCI, Umberto. Applied *deep learning*. A Case-Based Approach to Understanding Deep Neural Networks, 2018.

MITCHELL, Tom M. Does *Machine Learning* really work? AI Magazine, v. 18, n. 3, p. 11-11, 1997.

MONTAÑEZ, Neptali; BARRERA, Jomari Joseph. Automated Abaca Fiber Grade Classification Using Convolutional Neural Network. Visayas State University, Visayas, 2019.

MURPHY, Kevin P. *Machine Learning*: a probabilistic perspective. MIT press, 2012.

NEMENYI, Peter Bjorn. Distribution-free multiple comparisons. Princeton University, 1963.

NG, Andrew. *Machine Learning* yearning. URL: http://www. mlyearning. org/(96), v. 139, 2017.

PONTE, João Pedro da. O computador como ferramenta: Uma aposta bem sucedida. Inovação, p. 41-48, 1989.

RAMACHANDRAN, Prajit; ZOPH, Barret; LE, Quoc V. Searching for *activation function*s. arXiv preprint arXiv:1710.05941, 2017.

RASCHKA, Sebastian. Python Machine Learning. Packt publishing Ltda, 2015.

RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. Springer International Publishing, 2015. p. 234-241.

RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning representations by back-propagating errors. nature, v. 323, n. 6088, p. 533-536, 1986.

SAMUEL, Arthur L. Some studies in *Machine Learning* using the game of checkers. II—Recent progress. IBM Journal of Research and Development, v. 11, n. 6, p. 601-617, 1967.

SANDLER, Mark et al. Mobilenetv2: *Inverted residuals and linear bottleneck*s. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018. p. 4510-4520.

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

SRIVASTAVA, Nitish et al. *Dropout*: a simple way to prevent neural networks from *overfitting*. The Journal of Machine Learning Research, v. 15, n. 1, p. 1929-1958, 2014.

SZEGEDY, Christian et al. Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015. p. 1-9.

SZEGEDY, Christian et al. Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. p. 2818-2826.

TAN, Mingxing; LE, Quoc. Efficientnet: Rethinking model scaling for convolutional neural networks. In: International conference on Machine Learning. PMLR, 2019. p. 6105-6114.

TAN, Mingxing; LE, Quoc. Efficientnetv2: Smaller models and faster training. In: International Conference on Machine Learning. PMLR, 2021. p. 10096-10106.

TEKALE, Siddheshwar et al. Prediction of chronic kidney disease using *Machine Learning* algorithm. International Journal of Advanced Research in Computer and Communication Engineering, v. 7, n. 10, p. 92-96, 2018.

YANG, Suorong et al. Image data augmentation for deep learning: A survey. arXiv preprint arXiv:2204.08610, 2022.

ZEILER, Matthew D.; FERGUS, Rob. Visualizing and understanding convolutional networks. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13. Springer International Publishing, 2014. p. 818-833.

ZHANG, Xiangyu et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018. p. 6848-6856.

APÊNDICE

Anexo 1 – Algoritmo FiberNet

```
import torch.nn as nn
import torch.nn.functional as F
from copy import deepcopy
import math

'''
This method creates the structure of the convolution layer that we use
in FiberNet.
'''
def conv_bn2(in_i, out_o, k_k, pad_p, bias_b, s_s, g_g):
  return nn.Sequential(
    nn.Conv2d(in_i,out_o,kernel_size=k_k,padding=pad_p,bias=bias_b,stride=s_s,
groups=g_g),
    nn.BatchNorm2d(num_features=out_o)
  )

'''
This method that defibres the input tensioner from the height and width axes.
The method analyzes which lines are divisible by three and removes them from
the tensor that will be returned.
'''
def defibrate(x):
  listaA = []
  a = math.ceil(x.size(dim=2)/2)
  for i in range(x.shape[-1]):
    if(i%3):
      listaA.append(i)
  return x[:,:,:,listaA][:,:,listaA,:]

'''
Method responsible for repeating the defibrate function starting from a
certain value.
'''
def repeat_defibration(x,defVez):
  for _ in range(defVez):
    x = defibrate(x)
  return x

'''
Method responsible for analyzing the input and determining six specific values
for each fiber layer in the FiberNet algorithm. These values are represented by
the letters A,B,C,D,E,F and the value described for each one of them represents
the number of times the defibrate function is executed before each of the
convolution layers.
```

```
'''
def times_defibration(out):
  if(out.size(dim=2)>=200):
    return 1,2,1,1,1,1
  elif(out.size(dim=2)>=150):
    return 0,0,1,1,2,3
  elif(out.size(dim=2)>=100):
    return 1,2,1,1,0,0
  elif(out.size(dim=2)>=50):
    return 1,1,1,1,0,0
  else:
    return 0,0,0,1,1,2

'''
FiberNet algorithm main class.
'''
class FiberNet(nn.Module):
  def __init__(self):
    feat = 90
    super(FiberNet,self).__init__()
    self.camada1 = conv_bn2(3,feat,2,18,True,1,1)
    self.camada2 = conv_bn2(feat,feat//2,3,0,True,1,1)
    self.camada3 = conv_bn2(feat//2,feat,5,0,True,1,1)
    self.camada4 = conv_bn2(feat,feat//2,3,0,True,1,1)
    self.camada5 = conv_bn2(feat//2,feat,7,0,True,1,1)
    self.camada6 = conv_bn2(feat,feat//2,7,0,True,1,1)
    self.relu = nn.ReLU()
    self.fc1 = nn.LazyLinear(1000,bias=True)

  def forward(self,x):
    a,b,c,d,e,f = times_defibration(x)
    out = repeat_defibration(x,a)
    out = self.relu(self.camada1(x))
    out = repeat_defibration(out,b)
    out = self.relu(self.camada2(out))
    out = repeat_defibration(out,c)
    out = self.relu(self.camada3(out))
    out = repeat_defibration(out,d)
    out = self.relu(self.camada4(out))
    out = repeat_defibration(out,e)
    out = self.relu(self.camada5(out))
    out = repeat_defibration(out,f)
    out = self.relu(self.camada6(out))
    out = out.view(x.size(0),-1)
    out = self.relu(self.fc1(out))
    return out
```