UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
DEPARTAMENTO DE ESTATÍSTICA E INFORMÁTICA
PROGRAMA DE PÓS–GRADUAÇÃO EM INFORMÁTICA APLICADA

CLEBER ALBERTO CABRAL FERREIRA DA SILVA

UM FRAMEWORK DE EVOLUÇÃO GRAMATICAL MULTI-OBJETIVO PARA GERAÇÃO DE ARQUITETURAS DE REDES NEURAIS CONVOLUCIONAIS

RECIFE - PE

2022

CLEBER ALBERTO CABRAL FERREIRA DA SILVA

UM FRAMEWORK DE EVOLUÇÃO GRAMATICAL MULTI-OBJETIVO PARA GERAÇÃO DE ARQUITETURAS DE REDES NEURAIS CONVOLUCIONAIS

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Informática Aplicada do Departamento de Estatística e Informática - DEINFO - Universidade Federal Rural de Pernambuco, como parte dos requisitos necessários para obtenção do grau de Mestre.

ORIENTADOR: Prof. Dr. Péricles Barbosa Cunha de Miranda

RECIFE - PE

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

S586f Silva, Cleber Alberto Cabral Ferreira da
Um framework de evolução gramatical multi-objetivo para geração de arquiteturas de redes neurais convolucionais / Cleber Alberto Cabral Ferreira da Silva. - 2022.
90 f. : il.

Orientador: Pericles Barbosa Cunha de Miranda.
Inclui referências.

Dissertação (Mestrado) - Universidade Federal Rural de Pernambuco, Programa de Pós-Graduação em Informática Aplicada, Recife, 2022.

1. Evolução gramatical. 2. Redes neurais convolucionais. 3. Otimização multi-objetivo. I. Miranda, Pericles Barbosa Cunha de, orient. II. Título

CDD 004

CLEBER ALBERTO CABRAL FERREIRA DA SILVA

UM FRAMEWORK DE EVOLUÇÃO GRAMATICAL MULTI-OBJETIVO PARA GERAÇÃO DE ARQUITETURAS DE REDES NEURAIS CONVOLUCIONAIS

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Informática Aplicada do Departamento de Estatística e Informática - DEINFO - Universidade Federal Rural de Pernambuco, como parte dos requisitos necessários para obtenção do grau de Mestre.

Apresentada em: 15/02/2022

BANCA EXAMINADORA

Prof. Dr. Péricles Barbosa Cunha de Miranda (Orientador)
Universidade Federal Rural de Pernambuco
Departamento de Estatística e Informática

Prof. Ph.D. Ricardo Bastos Cavalcante Prudencio
Universidade Federal de Pernambuco
Centro de Informática

Prof. Dra. Gisele Lobo Pappa
Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

À minha família, fonte inesgotável de apoio e incentivo.

Agradecimentos

Agradeço primeiramente a Deus por todas as oportunidades que me concedeu, dando-me a possibilidade de subir mais esse degrau na escada da vida.

Aos meus pais por todo o empenho, esforço e dedicação que desempenharam desde minha concepção. Sou eternamente grato a eles e o que faço em vida nunca recompensará o que fizeram por mim.

À minha esposa Mirelly, minha companheira e amiga, que num simples olhar entende coisas que nunca falei e me ajuda no que for preciso. Esse período de estudo não seria possível sem sua ajuda. Você é tudo de bom que eu nunca imaginei que poderia encontrar.

Ao meu filho Miguel, que em tempos sombrios de pandemia trouxe luz à casa com seu nascimento. Por toda a alegria que traz com seus sorrisos engraçados.

À minha sogra, Edna Maria, por todo o apoio, ajuda e disponibilidade à nossa família durante este período.

A Daniel Rosa pela ajuda nos experimentos e pela co-autoria nos artigos publicados. Aos professores Péricles Miranda, Filipe Cordeiro, Valmir Macário e Danilo Ricardo que me forneceram subsídios para construir este estudo.

Ao meu orientador, Péricles Miranda, por toda ajuda, orientação e confiança. Deus abençoe imensamente sua vida.

À UFRPE, seus servidores e professores da DEINFO, por terem compartilhado conhecimento e me dado a oportunidade de cursar este mestrado.

Ao IFPE, minha casa, e aos meus colegas servidores da DADT que me incentivaram e apoiaram na busca por este mestrado.

Está consumado.

(João 19:30)

Resumo

Na última década, as Redes Neurais Convolucionais (CNN) chamaram bastante atenção devido ao sucesso na resolução de vários problemas de visão computacional, como a classificação de imagens. No entanto, à medida que novos modelos de redes eram lançados, sua complexidade e número de parâmetros treináveis aumentavam, consumindo mais poder computacional. Encontrar a configuração e arquiteturas ideais, com complexidade reduzida e alta performance, tornou-se uma tarefa difícil, requerendo o conhecimento de especialistas para sua confecção. A fim de auxiliar na busca por essas configurações otimizadas, este trabalho propõe o uso de um framework de evolução gramatical multi-objetivo que gera automaticamente CNNs otimizadas para um determinado problema de classificação de imagens. Nessa estrutura, uma Gramática Livre de Contexto mapeia o espaço de busca das possíveis soluções para a criação de redes. O framework otimiza esta busca levando em consideração duas funções objetivo: acurácia e F_1-score. A proposta foi validada em seis conjuntos de dados: CIFAR-10, MNIST, KMNIST e MedMNIST (PathMNIST, OCTMNIST e OrganMNIST Axial), e os resultados mostram que a proposta é capaz de gerar redes mais simples e com resultados competitivos ou que superam arquiteturas estado-da-arte (mais complexas) bem como outras arquiteturas também geradas por outras gramáticas.

Palavras-chave: Evolução Gramatical. Redes Neurais Convolucionais. Otimização Multi-objetivo

Abstract

In the last decade, Convolutional Neural Networks (CNN) has been highlighted due to their success in solving various computer vision problems, such as image classification. However, as new network models were launched, their complexity and number of trainable parameters increased, consuming more computational power. Finding the ideal configuration and architectures, with reduced complexity and high performance, became a difficult task, requiring the knowledge of specialists for its creation. In order to help in the search for these optimized configurations, this work proposes the use of a multi-objective grammatical evolution framework that automatically generates optimized CNNs for a given image classification problem. In this structure, a Context-Free Grammar maps the search space of possible solutions for the creation of networks. The framework optimizes this search taking into account two objective functions: accuracy and F_1-score. The proposal was validated on six datasets: CIFAR-10, MNIST, KMNIST and MedMNIST (PathMNIST, OCTMNIST and OrganMNIST Axial), and the results show that the proposal is capable of generating simpler networks with competitive results or that they outperform (more complex) state-of-the-art architectures as well as other architectures also generated by other grammars.

Keywords: Grammatical Evolution. Convolutional Neural Networks. Multi-Objective Optimization

Lista de Figuras

Lista de tabelas

Lista de Algoritmos

Lista de Siglas

ANN	*Artificial Neural Network*
BNF	*Backus-Naur Form*
CD	*Crowding Distance*
CFG	*Context-Free Grammar*
CGP	*Cartesian Genetic Programming*
CNN	*Convolutional Neural Network*
DENSER	*Deep Evolutionary Network StructurEd Representation*
DL	*Deep Learning*
DNN	*Deep Neural Network*
DSGE	*Dynamic Structured Grammatical Evolution*
GA	*Genetic Algorithm*
GE	*Grammatical Evolution*
GP	*Genetic Programming*
GPU	*Graphics Processing Unit*
LiTS	*Liver Tumor Segmentation Reference*
MLP	*Multi-Layer Perceptron*
MOEA	*Multi-Objective Evolutionary Algorithm*
MOO	*Multi-Objective Optimization*
TL	*Transfer Learning*
SOTA	*State-of-the-art*
TPU	*Tensor Processing Unit*

Sumário

1 Introdução

1.1 Motivação

Redes Neurais Convolucionais, ou *Convolutional Neural Netoworks* (CNN) é uma classe de métodos de *Deep Learning* (GOODFELLOW et al., 2016b) que tem mostrado significante performance ao lidar com diferentes tarefas de visão computacional, como a classificação de imagens e detecção de objetos (LIU et al., 2020), em diferentes áreas de aplicação (LECUN et al., 2015). Com os holofotes voltados à área, várias arquiteturas de CNNs foram propostas, com tamanhos e complexidades variadas, resultando num amplo número de parâmetros e opções de design que são totalmente dependentes do conjunto de dados associado (LECUN et al., 1998; SIMONYAN; ZISSERMAN, 2014; HE et al., 2016a; SZEGEDY et al., 2016).

Dado um problema de classificação, escolher uma arquitetura, com seus respectivos números de parâmetros, é um trabalho custoso devido a diversidade de possíveis arranjos de sua estrutura. Redes como LeNet-5 (LECUN et al., 1998) e ResNet-152 (HE et al., 2016a) possuem aproximadamente 60 mil e 60 milhões de parâmetros treináveis, respectivamente. No entanto, a primeira possui 5 camadas enquanto a segunda 152. Desta forma, a estrutura da arquitetura, suas camadas e respectiva disposição afetam diretamente na performance e complexidade da rede. O principal desafio ao utilizar uma CNN para um dado problema de classificação é a diversidade de arquiteturas de redes profundas que se pode escolher, além da configuração correta de seus hiper-parâmetros, tornando assim a otimização destas redes um problema complexo.

Frankle e Carbin (2018) demonstraram que é possível obter, através de técnicas de poda, sub-redes com um número reduzido de parâmetros sem comprometer a acurácia do modelo. Porém, técnicas de poda possuem algumas limitações, como, por exemplo o fato da solução final depender diretamente da CNN original e da sequência de suas camadas.

À medida que novas CNNs tornam-se mais complexas, definir uma arquitetura ideal, otimizada tanto em número de camadas quanto em parâmetros, torna-se uma tarefa custosa. No entanto, na busca de resolver este problema, pesquisadores vêm adotando a automatização do design das redes atrelada à tarefas de otimização. Alguns trabalhos otimizaram estas CNNs utilizando Neuro-Evolução (MIIKKULAINEN et al., 2019) e

Algoritmos Genéticos (NETO et al., 2020). Porém, os resultados obtidos em (ASSUNÇÃO et al., 2018; DINIZ et al., 2018; LIMA et al., 2019) mostraram que a Evolução Gramatical (EG) é uma abordagem promissora para a otimização de CNNs. EG é um algoritmo de Programação Genética (PG) que utiliza gramáticas livre de contexto para evoluir programas (em nosso casso, CNNs). A gramática pode incorporar conhecimento prévio sobre o domínio do problema para guiar a navegação do espaço de busca do algoritmo, evitando a incorreção sintática dos programas gerados (O'NEILL; RYAN, 2004). Assim, a EG tem vantagens potenciais para melhorar o desenho das redes uma vez que é flexível para representar e envolver arquiteturas mais complexas (O'NEILL; RYAN, 2004).

Assunção et al. (2018), Diniz et al. (2018), Lima et al. (2019) mostraram o potencial da Evolução Gramatical na geração e otimização de CNNs, no entanto, seus problemas a princípio foram modelados como problemas mono-objetivo cuja função objetivo é a maximização da acurácia das redes. Todavia, de acordo com Neto et al. (2020), gerar modelos de CNNs com um bom balanço entre generalização, acurácia, sensitividade e precisão é uma tarefa desafiadora. Portanto, tratar esta tarefa como um problema mono-objetivo não parece ser adequado, visto que devem ser satisfeitos diferentes aspectos (objetivos) durante a otimização.

Deste modo, neste trabalho, é proposto um framework de evolução gramatical multi-objetivo destinado a otimização de arquiteturas de CNNs, buscando encontrar modelos que apresentem baixa complexidade mas altos valores de acurácia e F_1-score, além de um menor tempo de treinamento e processamento. As contribuições deste trabalho são duas: foi implementado um motor de busca multi-objetivo e criada uma nova gramática livre de contexto associada a um mecanismo de mapeamento dos modelos. Ambas contribuições são voltadas a evoluir modelos de CNNs de baixa complexidade e alto desempenho.

A proposta foi comparada com arquiteturas de CNNs estado-da-arte bem como outras arquiteturas geradas por propostas que também utilizam a evolução gramatical. Todos os modelos tiveram sua eficiência avaliada de acordo com duas métricas: acurácia e F_1-score. Estas métricas foram obtidas após o teste de cada CNN. A execução do treino e teste de cada rede foi feita em 6 conjuntos de dados diferentes e os resultados mostraram que a proposta elaborada gerou modelos competitivos e de baixa complexidade quando comparados às outras arquiteturas.

1.2 Problema da Pesquisa

No processo de criação das redes neurais convolucionais, vários parâmetros devem ser analisados cautelosamente para que os modelos construídos alcancem bons resultados. Na criação de modelos eficientes o número de neurônios que compõem a rede, bem como a quantidade de camadas e respectiva disposição, afetam diretamente o tempo de treinamento do modelo. CNNs mal configuradas podem proporcionar longos treinamentos, que duram dias ou até mesmo semanas para serem concluídos. Desta forma, é importante que sejam tomadas boas decisões quanto a este aspecto.

A escolha da função de otimização, responsável pela atualização dos pesos das redes, também é de suma importância no desenvolvimento. A escolha errada da função pode trazer resultados errados ou até mesmo mais lentos, influenciando no treinamento.

Além da escolha correta da função de otimização, saber escolher a taxa de aprendizado ideal para a arquitetura em desenvolvimento é de vital importância. A taxa de aprendizado é um hiper-parâmetro que controla o quanto alterar o modelo em resposta ao erro estimado cada vez que os pesos são atualizados. Encontrar a taxa correta é uma tarefa desafiadora pois um valor muito pequeno pode resultar em um longo processo de treinamento, enquanto que um valor muito grande pode resultar no aprendizado de um conjunto de pesos abaixo do ideal muito rápido ou em um processo de treinamento instável.

A escolha do ambiente de execução das redes também influencia no tempo de processamento. Máquinas com grande quantidade de memória RAM e com disponibilidade de uso de GPUs[1] ou TPUs[2] tendem a ser mais eficientes no processamento das CNNs.

Todos estes pontos corroboram para a necessidade do conhecimento de um especialista na área de *deep learning*. Sendo assim, a problemática principal deste trabalho é a de fornecer um meio para que modelos eficientes e de baixa complexidade sejam criados sem a necessidade da interferência humana no processo de criação, fazendo o framework atuar com o papel de especialista no processo de elaboração dos modelos otimizados.

[1] *Graphics Processing Unit* (GPU) é um processador especializado originalmente desenhado para acelerar a renderização de gráficos. GPUs podem processar várias pacotes de dados simultaneamente, tornando-se útil na área de aprendizagem de máquina, edição de vídeos e aplicação em games.

[2] *Tensor Processing Unit* (TPU) é um circuito integrado específico de aplicação (ASIC) desenvolvido pela Google especificamente para o tratamento de redes neurais, atuando como um acelerador no aprendizado de máquina.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é o de definir um framework de evolução gramatical, composto por um motor de busca, uma gramática e um processo de mapeamento, responsáveis por criar arquiteturas otimizadas de redes neurais convolucionais que resolvam problemas de classificação de imagens. Tal framework visa auxiliar na obtenção de arquiteturas otimizadas, mais leves (menor número de parâmetros treináveis), e que tenham um tempo de treinamento menor comparado às outras arquiteturas estado-da-arte, mesmo sem o conhecimento de um especialista na área de aprendizagem de máquina.

A otimização é feita através da maximização de duas métricas (Acurácia e F_1-score), de forma que estas arquiteturas forneçam resultados competitivos à outras arquiteturas estado-da-arte, bem como outras arquiteturas também geradas pela técnica de evolução gramatical.

Para se atingir este objetivo geral foram definidos outros objetivos específicos, listados a seguir.

1.3.2 Objetivos Específicos

Buscando atingir o objetivo geral, foram propostos os seguintes objetivos específicos:

- Criar uma Gramática Livre de Contexto que forme um espaço de busca de fácil exploração e que forneça indivíduos válidos para a resolução dos problemas de classificação de imagens em vários conjuntos de dados;
- Configurar um Algoritmo Evolucionário Multi-Objetivo para a otimização de redes neurais convolucionais através da maximização de duas métricas: Acurácia e F_1-score;
- Implementar um processo de mapeamento responsável por converter indivíduos gerados pela gramática em redes neurais convolucionais aptas para a execução;
- Analisar os resultados, comparando-os com arquiteturas estado-da-arte e outras arquiteturas geradas por outras gramáticas livres de contexto listadas na literatura.

1.4 Estrutura do Trabalho

Para facilitar a compreensão deste trabalho, o mesmo encontra-se dividido em 7 capítulos, que são:

- No atual capítulo 1 são apresentados o problema da pesquisa, a motivação do estudo, os objetivos desejados e a estrutura do trabalho;
- No capítulo 2, são apresentados os conceitos necessários para o entendimento dos assuntos abordados no estudo, divididos em três grandes áreas: Fundamentos da Imagem Digital, Aprendizagem de Máquina e Algoritmos Evolucionários;
- No capítulo 3, são listados os trabalhos relacionados, suas características e contribuições no estudo;
- O capítulo 4 traz a proposta do trabalho, descrevendo em detalhes o processo de elaboração do estudo;
- No capítulo 5, a metodologia do estudo é detalhada, trazendo informações como as bases de dados utilizadas, o ambiente de execução dos testes e os métodos comparativos dos dados obtidos;
- Dando continuidade, o capítulo 6 mostra os resultados obtidos com a proposta, trazendo comparativos entre o trabalho atual e outras técnicas estado-da-arte;
- Por fim, o capítulo 7 traz as considerações finais do estudo, suas contribuições e possíveis trabalhos futuros.

2 Fundamentação Teórica

Este capítulo faz uma breve explanação dos conceitos básicos necessários para a compreensão deste trabalho. Ele encontra-se dividido sequencialmente em quatro grandes áreas: 2.1 Onde são apresentados os fundamentos de imagem digital, necessários para a compreensão da maneira como a proposta trata as imagens utilizadas; 2.2 Onde os conceitos de aprendizagem de máquina e inteligência artificial são apresentados, mostrando como a classificação de imagens executada por redes neurais pode ajudar no desenvolvimento do trabalho; Na seção 2.3 os conceitos de algoritmos evolucionários são apresentados e como eles podem auxiliar na busca e otimização de redes neurais convolucionais em problemas de classificação de imagens; Por fim, não menos importante, a seção 2.4 mostra o conceito da otimização multi-objetivo e como esta técnica auxilia na busca das melhores soluções para a resolução do problema desta proposta.

2.1 Fundamentos de Imagem Digital

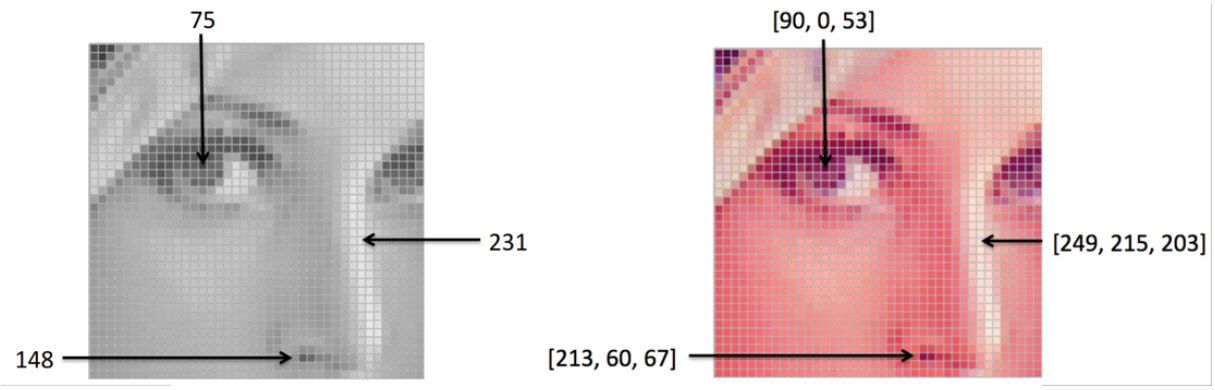
2.1.1 Pixel

A palavra pixel é a combinação dos termos *pix* (vindo de *"pictures"*, abreviado do inglês) e *el* (de *"element"*, do inglês). Ou seja, em imagem digital, um pixel é o menor elemento endereçável de uma imagem, ou o menor elemento controlável de uma imagem (FOLEY; DAM, 1982). Uma imagem digital nada mais é do que uma matriz $x \times y$ onde cada pixel ocupa uma posição x, y da mesma.

O conjunto de pixels em forma de uma matriz forma a imagem em si. A dimensão dessa matriz determina a resolução da imagem, que nada mais é do que a quantidade de pixels contidos na horizontal pela quantidade de pixels contidos na vertical da matriz.

De acordo com o tipo de imagem, cada pixel armazena um tipo de informação. Para imagens em tons de cinza, o pixel armazena a intensidade da cor que nada mais é que um valor variando entre 0 e 255. Enquanto que para imagens coloridas, cada pixel armazena a informação dos três canais de cores da própria cor, conforme o sistema de cores RGB (*Red-Green-Blue*, onde cada canal de cor representa um valor de 0 a 255). A Figura 1 apresenta estes detalhes de acordo com tipo de imagem.

Figura 1 – Representação dos pixels em imagens tons de cinza e coloridas



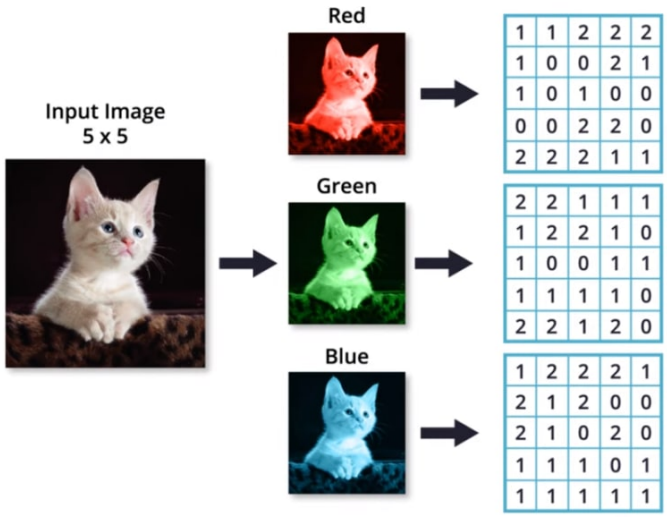
Fonte – Stanford University - CS 131 Computer Vision Foundations and Applications

2.1.2 Sistemas de Cores RGB

O Sistema de Cores RGB, Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*), é um sistema de cores aditivas onde é possível reproduzir um largo espectro cromático de cores a partir da mistura destas três cores primárias. Este sistema tem como principal objetivo a reprodução de cores em dispositivos eletrônicos.

No sistema RGB cada pixel possui a informação dos três canais de cores disponíveis, armazenadas em três *bytes*. Cada canal de cor pode receber valores entre 0 e 255, formando assim a composição da cor que deve ser apresentada no pixel. A Figura 2 mostra um exemplo de uma imagem hipotética de tamanho 5×5.

Figura 2 – Representação dos pixels numa imagem com sistema de cores RGB.



Fonte – Machine Learning - Convolution with color images

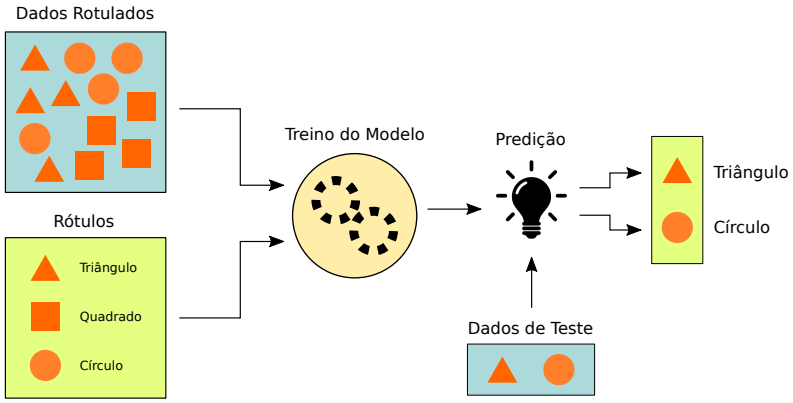
2.2 Aprendizagem de Máquina

Aprendizado de Máquina (AM) é uma área de pesquisa da Inteligência Artificial que visa o desenvolvimento de programas de computador com a capacidade de aprender a executar uma dada tarefa com sua própria experiência (MICHALSKI et al., 2013; LORENA et al., 2000). Em resumo, na AM, um sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas através da solução bem sucedida de problemas anteriores. O objetivo principal da AM é o desenvolvimento de técnicas computacionais sobre o aprendizado, bem como a construção de sistemas capazes de adquirir conhecimento de forma automática.

De forma geral, a maneira como os programas de computador aprendem na AM pode ser divida em três tipos de aprendizado: supervisionado, não-supervisionado e de reforço (RUSSELL; NORVIG, 2010). Neste trabalho, o foco da aprendizagem dos programas está voltada em torno da aprendizagem supervisionada.

Na aprendizagem supervisionada, os programas de computador relacionam uma saída com uma entrada com base em dados rotulados. Sendo assim, o usuário fornece ao programa pares de entradas e saídas conhecidos. Pada cada saída um rótulo é atribuído, que pode ser um valor numérico ou uma classe. O programa então determina uma forma de prever qual o rótulo de saída com base em uma entrada informada (GOODFELLOW et al., 2016b), conforme pode ser visto na Figura 3. Os programas onde sua saída pode assumir somente um conjunto de rótulos pré-definidos são chamados de algoritmos de classificação.

Figura 3 – Esquema de predição na aprendizagem supervisionada.



Fonte – Adaptado do website javatpoint.com

No caso da aprendizagem não-supervisionada, não é atribuído um rótulo para os dados de saída. Analisando uma grande quantidade de dados, o programa busca padrões ou similaridades nos dados, permitindo assim identificar grupos de itens similares ou similaridade de itens novos com grupos já conhecidos.

Por fim, na aprendizagem por reforço o programa realiza uma ação e recebe uma recompensa de acordo com o resultado desta ação, sendo o objetivo do programa receber a maior recompensa possível.

As seções a seguir aprofundam mais os conceitos de aprendizagem de máquina necessários para o entendimento deste trabalho.

2.2.1 Redes Neurais Atificiais

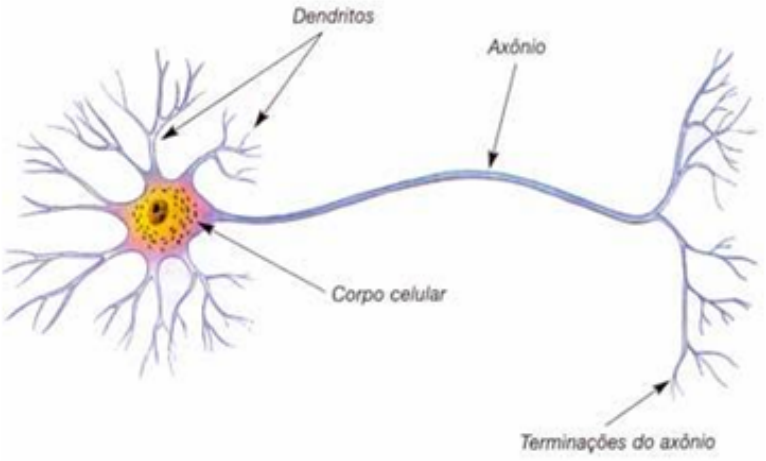
Confome Haykin (1998), uma Rede Neural Artificial (RNA) é um sistema de processamento massivamente paralelo, inspirado nas redes neurais biológicas, composto por unidades simples com capacidade natural de armazenar conhecimento e disponibilizá-lo para uso futuro. Essas unidades são distribuídas em camadas e estão interligadas por conexões normalmente unidirecionais. Na maioria dos modelos de RNAs as conexões possuem pesos associados, que são responsáveis por armazenar o conhecimento representado no modelo. Estes modelos buscam emular o comportamento do cérebro humano.

Uma RNA se inspira no funcionamento de uma rede neural biológica, todavia as RNAs são técnicas computacionais construídas a partir de modelos matemáticos que buscam adquirir conhecimento através de experiências (HAYKIN, 1998). O neurônio biológico tem sua estrutura celular formada pelo corpo celular, dendritos, axônios e terminais sinápticos, conforme pode ser visto na Figura 4. Em condições normais, dois neurônios se associam estabelecendo contato entre o dendrito de um e o axônio de outro formando assim uma sinapse que é onde ocorre a transmissão de impulsos nervosos de uma célula para outra (KOVÁCS, 2002).

A base de uma rede neural artificial é o seu modelo de neurônio, sendo ele uma unidade de processamento de informações onde uma RNA é composta por várias unidades de processamento. A Figura 5 mostra o modelo do neurônio artificial. O modelo é composto por três partes principais: as entradas (ou sinais de entrada, sinapses), o somador e a função de ativação.

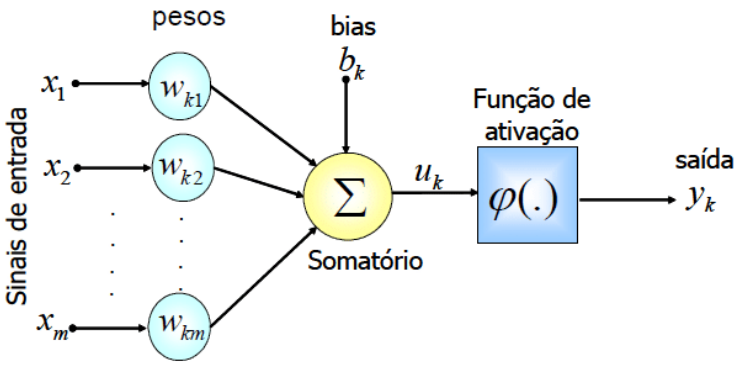
Cada sinal de entrada é composto por um índice e um peso, sendo cada sinal de

Figura 4 – Representação de um neurônio biológico.



Fonte – Adaptado de Haykin (1998).

Figura 5 – Representação de um neurônio artificial.

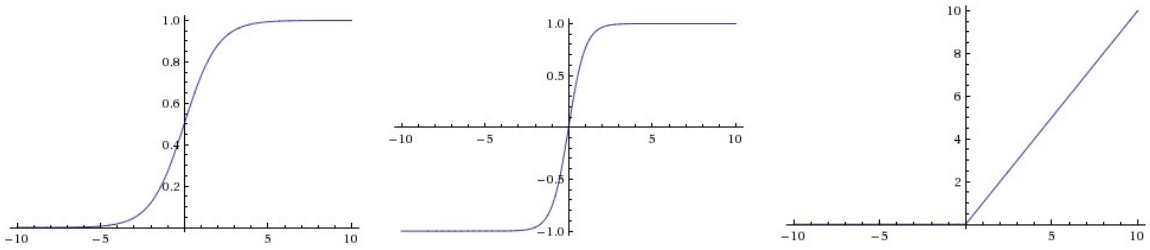


Fonte – Adaptado de Haykin (1998).

entrada multiplicado pelo seu peso associado onde indica a fluência da entrada no valor da saída. A soma dos pesos de cada entrada não indica nenhuma característica do modelo, onde os pesos são variados conforme cada entrada e diferentemente de uma sinapse cerebral, podem ser tanto positivos quanto negativos.

O somador tem a função de somar ponderadamente todas as entradas multiplicadas pelos seus respectivos pesos, produzindo assim uma entrada para a função de ativação. A função de ativação tem como objetivo determinar um limite máximo e mínimo para a amplitude de saída do neurônio, determinando um intervalo de ativação que pode variar de 0 a 1 ou de -1 a 1. Posteriormente, a função de ativação produz uma resposta de saída. Existem vários tipos de função de ativação, variando seu uso de acordo com o projeto, conforme a Figura 6. No entanto, uma das mais utilizadas é a função ReLU (*Rectified Linear Unit*) (DAHL et al., 2013), que é definida por $f(x) = \max(0, x)$.

Figura 6 – Exemplos de funções de ativação.



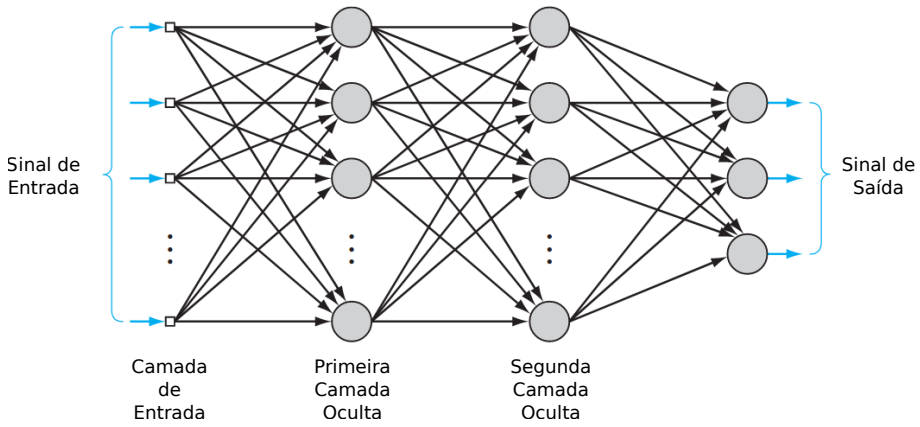
(a) Sigmoid (b) Tangente Hiperbólica (c) ReLU

Fonte – Adaptado do website CS231n Convolutional Neural Networks for Visual Recognition.

A principal vantagem de usar a função ReLU sobre outras funções de ativação é que ela não ativa todos os neurônios ao mesmo tempo. Caso uma entrada contenha um valor negativo, ela será convertida em zero e o neurônio não será ativado. Isso significa que, ao mesmo tempo, apenas alguns neurônios são ativados, tornando a rede esparsa e eficiente e fácil para a computação.

Prosseguindo, as RNAs são modeladas como coleções de neurônios que estão conectadas em um gráfico acíclico. Logo, as saídas de alguns neurônios podem se tornar entradas para outros neurônios. Os modelos de redes neurais costumam ser organizados em camadas distintas de neurônios, que, para redes neurais regulares, o tipo de camada mais comum é a camada totalmente conectada (em inglês, *fully connected layers*) também conhecidas como *Multi-Layer Perceptron* (MLP), na qual os neurônios entre duas camadas adjacentes estão totalmente conectados aos pares. A Figura 7 mostra uma topologia de rede neural que utiliza camadas totalmente conectadas.

Figura 7 – Arquitetura de uma RNA com múltiplas camadas conectadas.



Fonte – Adaptado de Haykin (1998).

Por fim, a arquitetura de uma RNA pode ser modificada para atender o problema associado ou para se adaptar à quantidade de dados que serão processados durante seu treinamento.

2.2.2 *Deep Learning*

O *Deep Learning* (DL), ou Aprendizagem Profunda, é uma área do aprendizado de máquina que permite que modelos computacionais modelados em múltiplas camadas de processamento possam aprender a representação de dados em múltiplos níveis de abstração (BENGIO, 2009; GOODFELLOW et al., 2016b).

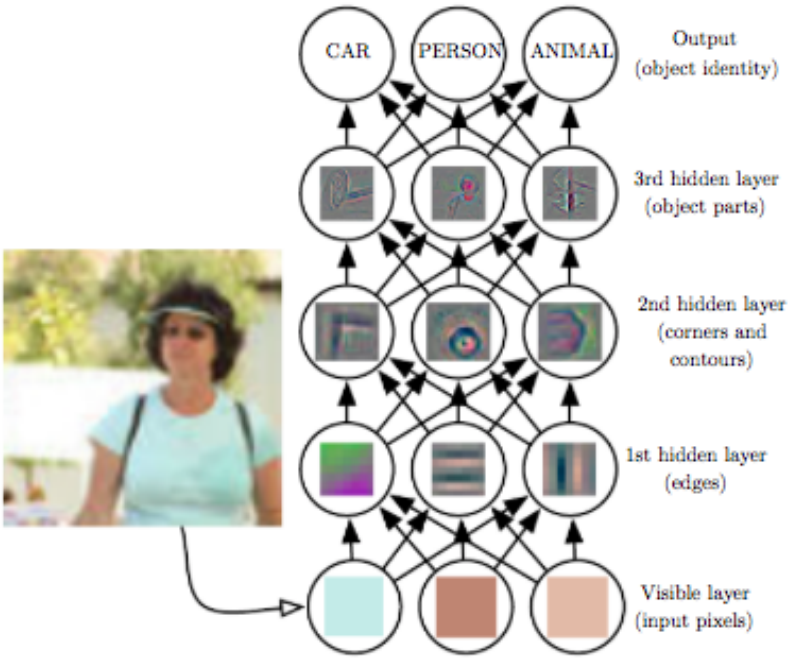
Desta maneira, programas computacionais podem alcançar um grande poder e flexibilidade, aprendendo a representar o mundo como uma hierarquia aninhada de representações (LECUN et al., 2015). Através do DL, os programas podem aprender através de várias etapas, onde cada camada de representação pode ser considerada um estado da memória do programa. Sendo assim, redes com uma maior profundidade podem executar instruções em sequência, onde as camadas mais internas processam os dados provenientes das camadas anteriores, permitindo assim um grande poder de processamento (GOODFELLOW et al., 2016b).

Sendo assim, programas computacionais podem entender o significado de dados brutos provenientes de uma imagem de entrada representada para uma matriz de valores de *pixels*. Por exemplo, um programa pode representar o reconhecimento de imagens de uma pessoa, combinando características simples, como cantos e contornos processados em uma estrutura de camadas distribuídas de forma hierárquica, conforme pode ser visto na Figura 8. Através disso, surge no DL uma nova técnica específica para a reconhecimento de imagens, conhecida como Redes Neurais Convolucionais.

2.2.3 Redes Neurais Convolucionais

Redes Neurais Convolucionais, do inglês *Convolutional Neural Network* (CNN) (LECUN et al., 2015) tentam simular o funcionamento do cérebro humano para o reconhecimento de determinados padrões em dados, sejam eles imagens, sons, vídeos ou dados volumétricos (GOODFELLOW et al., 2016a). Ao final de uma CNN, esses dados são reduzidos a um vetor representativo dos padrões extraídos, que, por sua vez, serão processados por uma rede neural tradicional na forma de camadas totalmente conectadas.

Figura 8 – Exemplo de classificação de uma pessoa através de *Deep Learning*.



Fonte – Adaptado do website towardsdatascience.com.

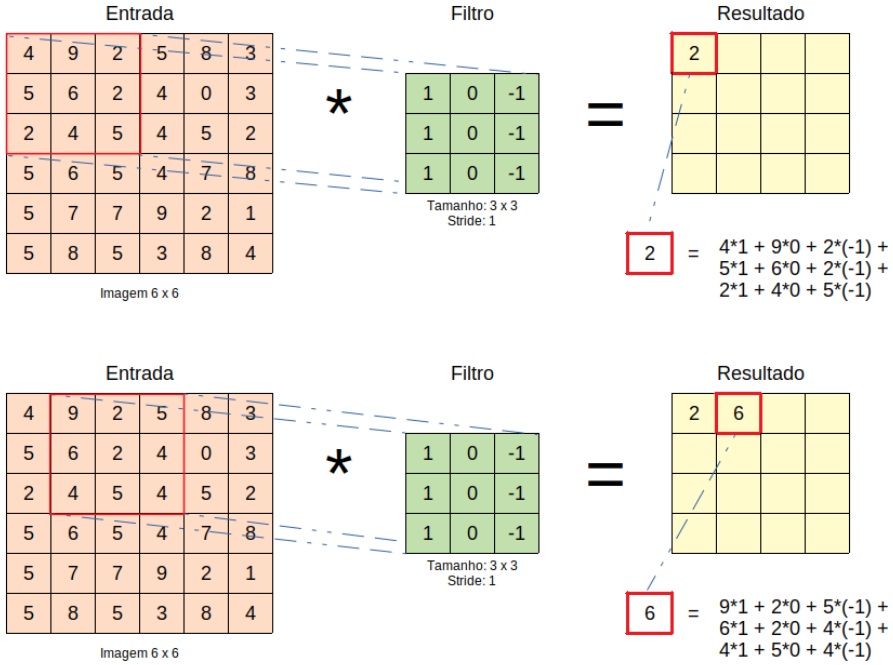
As CNNs são compostas por múltiplas camadas, cada uma responsável por uma determinada tarefa, que processam os dados provenientes das camadas anteriores. A topologia de uma CNN é composta por diferentes tipos e números de camadas. A maneira como essas camadas estão dispostas e seus respectivos tipos atuam diretamente na forma como a CNN irá performar. Habitualmente, há três tipos de camadas mais popularmente usadas: camadas convolucionais, camadas de *pooling* e camadas totalmente conectadas.

As camadas convolucionais formam uma parte primordial no processamento das CNNs. Elas atuam como uma composição de filtros que identificam as principais características de uma determinada entrada. Logo, essas são responsáveis por formar o mapa de características que um determinado dado possui. Numa camada convolucional é realizada uma transformação linear entre uma matriz primária e outra denominada *kernel* ou matriz de convolução. É feita a sobreposição do *kernel* sobre a matriz primária gerando uma terceira matriz.

O *kernel* é composto por um tamanho $m \times n$, seus valores em $f(m, n)$, o valor x de deslocamento (denominado *stride*) e o elemento central que deve ser posicionado sobre o primeiro pixel da imagem. Logo, é feita a soma dos produtos dos elementos que se sobrepõem, gerando um novo valor para a respectiva posição. Após isso, o *kernel*

é deslocado x pixels, onde a mesma operação é repetida até que toda a imagem seja percorrida, conforme pode ser visto na Figura 9. (BENGIO, 2009). Caso uma imagem tenha três canais de cores, o filtro é aplicado em cada um deles.

Figura 9 – Exemplo da aplicação de um filtro 3×3 numa imagem 6×6.

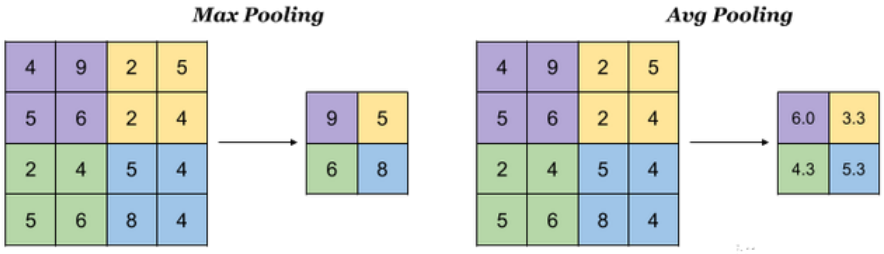


Fonte – Adaptado do website indoml.com.

Enquanto as camadas convolucionais destacam as características de um dado, as camadas de *pooling* implementam uma operação de redução de dimensionalidade dos dados, focando na redução do número de parâmetros treináveis da rede. Dado um mapa de características, estas camadas calculam a média (*average pooling*) ou os máximos (*max pooling*) locais dos dados introduzidos, obtendo assim sub-amostras, de menor tamanho, dos dados principais porém contendo as características mais relevantes identificadas (HAYKIN, 1998). O método mais utilizado é o de maximização dos dados. Geralmente, as camadas de *pooling* encontram-se após as camadas convolucionais. A Figura 10 mostra exemplos da aplicação das duas técnicas de *pooling*.

Já as camadas totalmente conectadas nada mais são do que a rede neural em si e formam a disposição final de uma CNN. Elas interligam uma série de neurônios que recebem os dados processados pelas camadas anteriores e atuam como um classificador da rede, tomando a decisão final sobre o processo de reconhecimento. Para a representação das probabilidades das saídas na última camada, normalmente utiliza-se a função de

Figura 10 – Exemplos da aplicação de um *pooling* com tamanho 2×2 e *stride* 2.



Fonte – Adaptado do website indoml.com.

ativação *softmax* (NWANKPA et al., 2018).

Além das camadas citadas, existem outras responsáveis pela regularização do processo de treinamento da rede, que são as camadas de normalização em lote (*batch normalization*) e camadas de *dropout*. Camadas de normalização em lote têm sido utilizadas em redes modernas e são responsáveis pela normalização da entrada entre as camadas. Geralmente, encontram-se entre as camadas convolucionais e de *pooling*. Camadas de *dropout* são utilizadas para evitar *overfitting* no modelo, desabilitando de forma randômica uma porcentagem dos neurônios presentes na rede. Comumente são utilizadas após camadas de *pooling* ou camadas totalmente conectadas. A Figura 11 mostra uma CNN com diversas camadas, separadas entre as responsáveis pela extração de características de uma imagem e as responsáveis pela classificação final.

2.3 Algoritmos Evolucionários

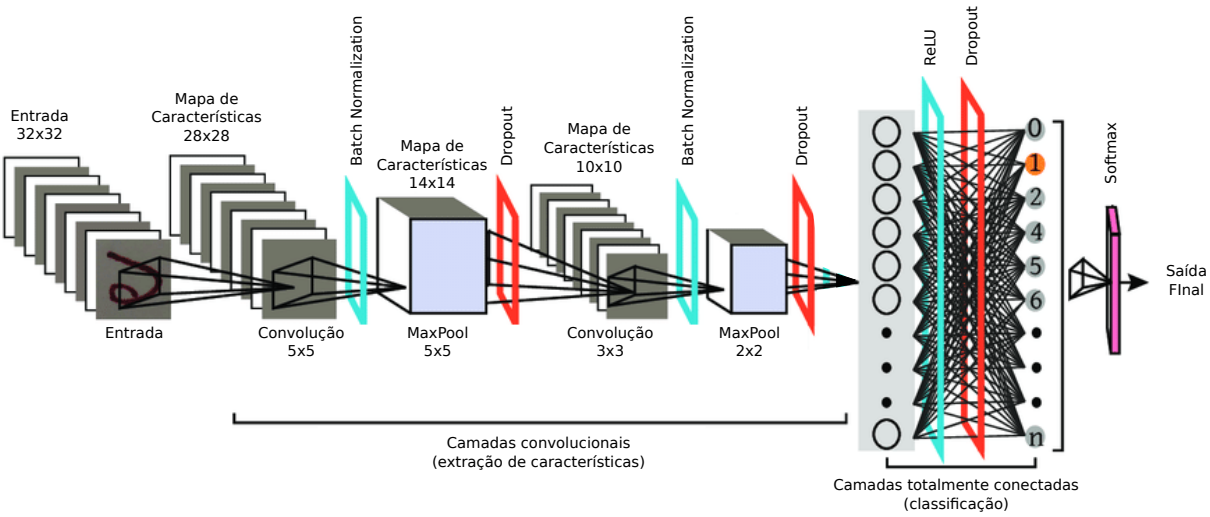
2.3.1 Princípios da Evolução Natural

A Teoria da Evolução, publicada no livro *Sobre a origem das espécies através da seleção natural* por Darwin et al. (1859) fala que espécies são criadas e exterminadas a partir do princípio de tentativa e erro, onde seres vivos superiores desenvolvem-se através de formas menores, inferiores.

Sendo assim, na história de uma espécie seu desenvolvimento é proveniente de um lento processo de alterações que agem numa população. Logo, a teoria propõe que os seres vivos são decorrentes da evolução de seres inferiores, mais simples, que sofreram modificações e foram evoluindo no decorrer de suas gerações.

Segundo Darwin et al. (1859), esta série de modificações ocorrem a partir da interação entre quatro processos distintos: seleção, cruzamento, mutação e competição.

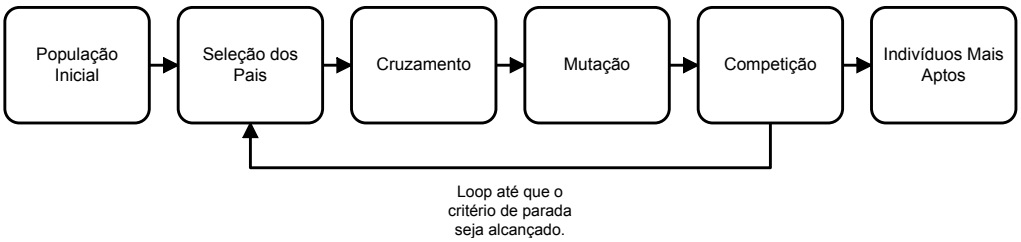
Figura 11 – Exemplo de uma Rede Neural Convolucional.



Fonte – Adaptado de Rahaman et al. (2019).

O processo de seleção natural começa com a seleção dos indivíduos mais aptos vindos de uma população inicial. Estes indivíduos produzem descendentes, pelo cruzamento, que herdam as características dos pais e são adicionados à próxima geração. Estas características podem sofrer pequenas mutações que podem tornar estes descendentes mais aptos que os pais. Estes novos indivíduos competem pelo meio e apenas os mais aptos são selecionados para dar início à próxima população. Este processo continua iterando até o final, onde uma geração com indivíduos mais aptos será encontrada. A Figura 12 mostra o esquema deste processo.

Figura 12 – Esquema de um processo evolutivo.



Fonte – O autor.

2.3.2 Computação Evolutiva

Uma vez formulada a Teoria da Evolução através das observações feitas por Darwin et al. (1859), décadas depois, com o crescimento da ciência da computação, novos algoritmos foram propostos buscando reproduzir os conceitos da evolução biológica (BARRICELLI et al., 1954; FRASER, 1958), dando início ao que foi chamada de Computação Evolutiva (CE).

Os conceitos elaborados por Darwin et al. (1859), listados abaixo, agora fomentados por um arcabouço de conhecimentos genéticos, foram adaptados através da CE dando subsídios para construção de algoritmos voltados à otimização de problemas (DORIN; TAYLOR, 2020).

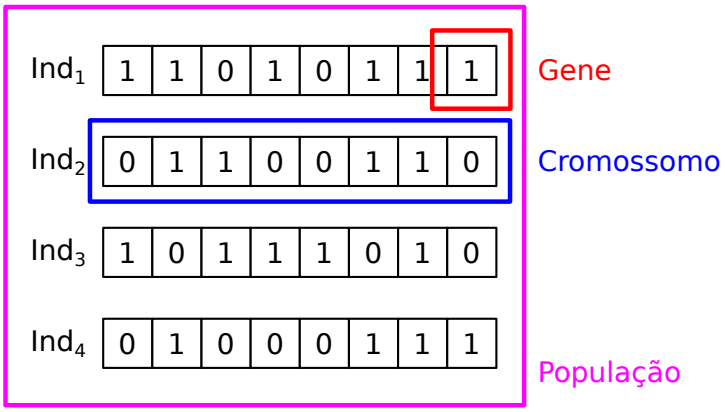
- **Gene:** é a informação mais simples de um indivíduo, que em conjunto forma uma característica de um indivíduo;
- **Cromossomo:** é um conjunto de genes, responsável por formar as informações genéticas de um indivíduo, ou seja, seu genótipo;
- **Indivíduo:** é em si a representação ou manifestação externa do conjunto de suas características, biologicamente representado pelo nome fenótipo;
- **População:** é um conjunto de indivíduos, ou em outras palavras, o conjunto de soluções candidatas disponíveis para execução do algoritmo;
- **Cruzamento:** é o ato de reproduzir dois indivíduos pais na tentativa de gerar um novo indivíduo filho, composto pelas características herdadas;
- **Mutação:** são pequenas variações das características que formam um indivíduo;
- **Seleção:** é o ato de selecionar em meio à população os indivíduos mais aptos ao meio ambiente inserido, e permitir que suas características sejam repassadas para a próxima geração.

A Figura 13 mostra a representação visual dos principais conceitos adaptados, em indivíduos representados por uma cadeia binária.

Num algoritmo evolutivo o primeiro passo dado é o de criar uma população inicial. Os indivíduos desta população podem ser criados com características aleatórias ou seguindo um certo padrão. A aptidão destes indivíduos ao meio é calculada através de uma função chamada *fitness*. Ela é responsável por calcular o grau de adaptação do indivíduo ao meio onde está inserido e varia de acordo com o problema relacionado.

Criada a população inicial, o algoritmo avalia o *fitness* desta população. A partir

Figura 13 – População, cromossomos e genes.



Fonte – Adaptado do website Introduction to Genetic Algorithms

de agora, um laço de repetição é iniciado onde pares de indivíduos são selecionados, de acordo com um viés de aptidão calculado na função de *fitness*. Cada par de indivíduos é cruzado entre si, gerando dois novos indivíduos filhos. Estes novos indivíduos sofrem então uma mutação em seu genótipo de acordo com uma taxa probabilística. Aplicada a mutação em cada indivíduo filho, estes são adicionados à população e novamente a aptidão é calculada pela função de *fitness*. O laço continua até que um critério de parada seja alcançado, seja ele uma certa quantidade de iterações ou um resultado almejado que foi estabelecido (ASHLOCK, 2006). Vale salientar que nem todos os problemas possuem solução. O Algoritmo 1 mostra o pseudocódigo de um algoritmo evolucionário básico.

Algoritmo 1: Pseudocódigo de um algoritmo evolucionário básico (ASHLOCK, 2006).

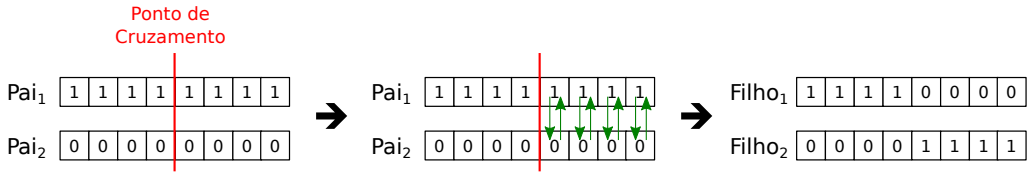
```
populacao ← inicializaPopulacao();
avaliaFitness(populacao);
enquanto critério de parada não for satisfeito faça
    pares ← selecionaPares(populacao);
    filhos ← [];
    para cada parente_1 e parente_2 contido em pares faça
        filho_1, filho_2 ← realizaCruzamento(parente_1, parente_2);
        filhos ← aplicaMutacao(filho_1);
        filhos ← aplicaMutacao(filho_2);
    fim
    acrescenta(filhos, populacao);
    avaliaFitness(populacao);
fim
```

O cruzamento é uma das fases mais importantes num algoritmo genético. A partir

do cruzamento é que o algoritmo navega pelo espaço de busca à procura de regiões mais promissoras para a resolução do problema. Há vários tipos de cruzamento, sendo o *Single Point Crossover* (Cruzamento de Ponto Único) e *Two Point Crossover* (Cruzamento de Dois Pontos) os dois mais utilizados.

No *Single Point Crossover* o algoritmo gera aleatoriamente um divisor chamado *crossover point*, ou ponto de cruzamento, responsável por demarcar uma região de separação dos genes do par de pais. Então, é feita a troca do genes dos pais entre si até que o ponto de cruzamento seja alcançado. Deste modo, a informação de cada filho vem de um pai diferente antes e depois do ponto de cruzamento, conforme pode ser visto na Figura 14.

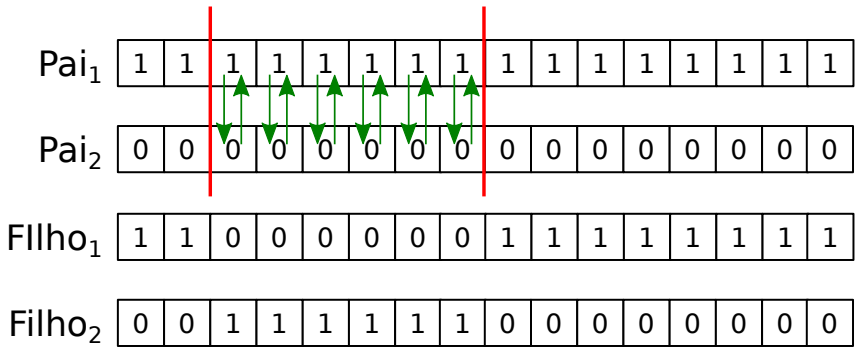
Figura 14 – Cruzamento de ponto único.



Fonte – Adaptado de Ashlock (2006).

Já para o *Two Point Crossover*, dois pontos de cruzamento são gerados, e em seguida os genes dos indivíduos filhos são copiados de um dos pais antes e depois dos pontos de cruzamento, e do outro pai entre os pontos de cruzamento. Este tipo de cruzamento tende a evitar que genes muito próximos um do outro na representação, tenham uma maior probabilidade de aparecer nos indivíduos filhos do que genes que encontram-se mais distantes um do outro. A Figura 15 mostra o esquema deste tipo de cruzamento.

Figura 15 – Cruzamento de ponto duplo.

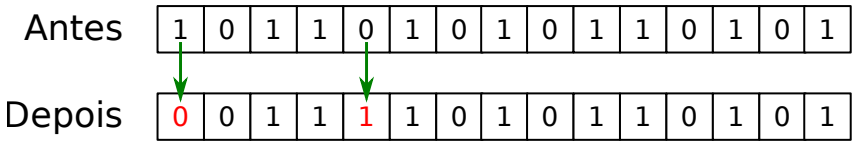


Fonte – Adaptado de Ashlock (2006).

Enquanto o cruzamento mistura e combina indivíduos diferentes, facilitando uma

navegação ampla no espaço de busca, a mutação, por outro lado, faz pequenas mudanças nos indivíduos, favorecendo a busca local e a introdução de pequenas novas ideias na população. Essas mudanças são aplicadas em um ou mais genes do cromossomo do indivíduo, e há diversos tipos de mutação. Uma da mais utilizadas é a *Int Flip Per Codon*, onde o operador de mutação lê de forma linear o cromossomo do indivíduo e aleatoriamente muda o gene de acordo com uma taxa de probabilidade. Geralmente, esta taxa recebe um valor baixo (entre 1% a 3%) para que o indivíduo não sofra grandes modificações e assim a navegação do espaço de busca seja prejudicada. A Figura 16 mostra o esquema de uma mutação deste tipo.

Figura 16 – Mutação *Int Flip Per Codon*.



Fonte – Adaptado de Ashlock (2006).

Todas estas operações proporcionam que os algoritmos evolucionários simulem a evolução biológica. No entanto, representar um indivíduo e portá-lo para um problema evolutivo ainda é uma tarefa difícil, já que se faz necessário manter sua estrutura simples e ao mesmo tempo completa. Nos exemplos das imagens anteriores os indivíduos foram representados por uma cadeia de *bits*. Todavia, indivíduos mais complexos necessitam de outras técnicas para facilitar sua manipulação. Uma dessas técnicas é a Programação Genética (PG).

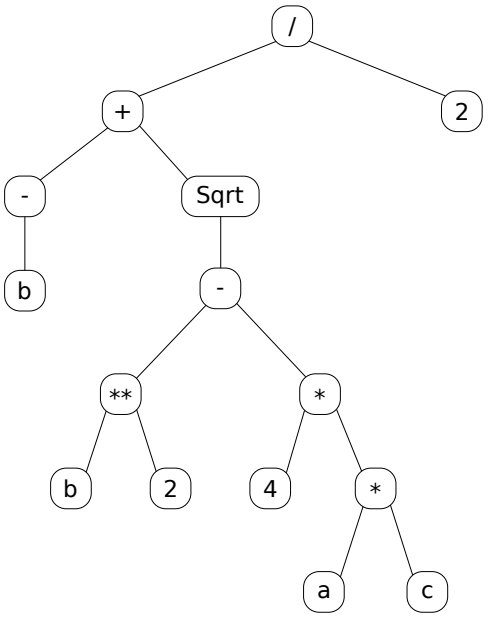
2.3.2.1 Programação Genética

A Programação Genética (PG) é uma técnica proposta por Koza (1990), onde os algoritmos evolucionários são utilizados para evoluir programas de computador, tipicamente representados em estrutura de árvores sintáticas. Desta forma, os programas de computador, quando executados, são as próprias soluções candidatas do espaço de busca (KOZA et al., 1992). Estes programas são otimizados durante várias gerações do algoritmo evolucionário até que um critério de parada ou resultado esperado seja alcançado.

A Figura 17 mostra um exemplo da representação em árvore sintática de um programa que nada mais é que uma função matemática (representada em expressões LISP

[1]) que retorna um resultado dado alguns valores. A árvore tem os nós interiores chamados de *operadores* e as folhas chamadas de *terminais*. Na figura, os operadores são *Sqrt*, /, +, -, * e **. Já os terminais podem conter valores externos passados pelo programa ou constantes. No caso, na figura são valores externos a, b e c, e as constantes 2 e 4.

Figura 17 – Exemplo de uma função matemática representada na estrutura de árvore sintática.



(/ (+ (- b) (Sqrt (- (** b 2) (* 4 (* a c))))) 2)

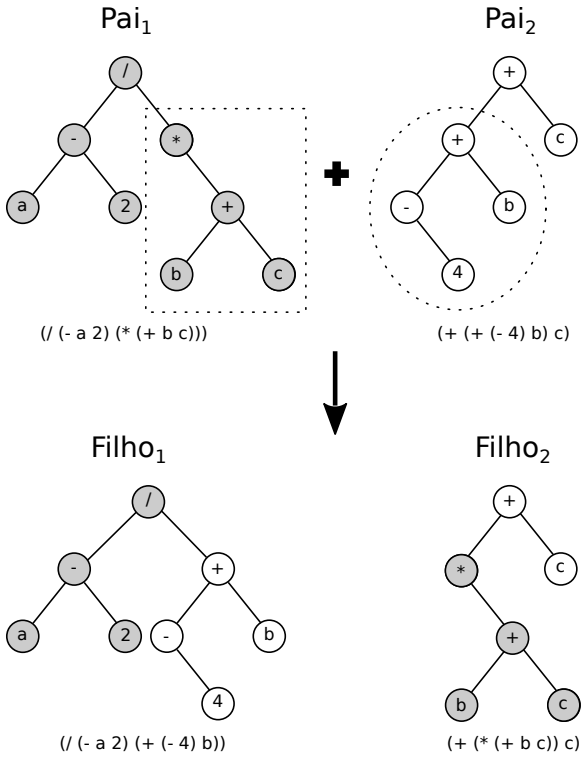
Fonte – Adaptado de Ashlock (2006).

Pelo fato dos indivíduos agora serem representados por árvores sintáticas, os cruzamentos e mutações ocorrem de uma maneira diferente. Basicamente, o cruzamento é performado pela troca de sub-árvores dos pais entre si. Para a escolha das sub-árvores, um ponto aleatório é escolhido na árvore de cada indivíduo pai, marcando assim a origem de cada sub-árvore. Escolhidas as sub-árvores, elas são trocadas entre os indivíduos pais, formando assim novos indivíduos filhos, conforme pode ser visto na Figura 18.

Já para a mutação dos indivíduos, o procedimento mais utilizado é bem simples. Um ponto da árvore é selecionado aleatoriamente, marcando o início de uma sub-árvore. Esta sub-árvore é então apagada e substituída por uma outra gerada aleatoriamente, conforme pode ser visto na Figura 19.

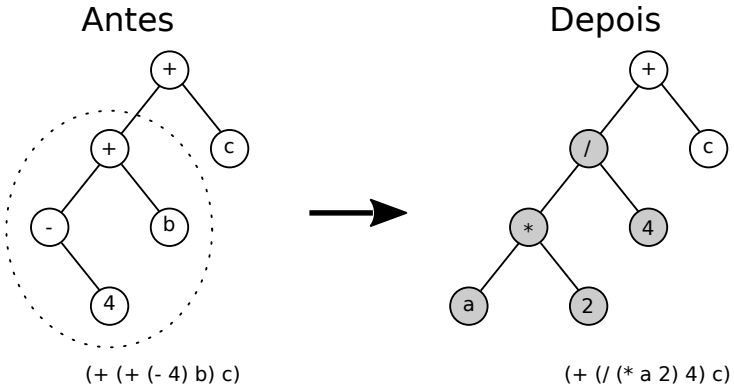
[1] Expressões LISP são representadas de várias formas. A forma mais comum de representação é a de aplicação em funções. LISP representa a chamada de uma função $f(x)$ como (fx). Por exemplo, $\cos x$ é escrito como $(\cos x)$.

Figura 18 – Cruzamento entre duas árvores sintáticas trocando sub-árvores.



Fonte – O autor.

Figura 19 – Mutação de uma árvore sintáticas.



Fonte – O autor.

Apesar de fornecer de uma maneira simples as operações de mutação e cruzamento de programas computacionais, Koza et al. (1992) percebeu que estas operações poderiam gerar indivíduos inválidos. Por isso, para que a busca seja efetiva, ele introduziu o conceito de *closure*, que cita:

> Cada uma das funções no conjunto de funções (deve) ser capaz de aceitar, como seus argumentos, qualquer valor e tipo de dados que possa ser retornado por qualquer função no conjunto de funções e qualquer valor e tipo de dados

que possa ser assumido por qualquer terminal no conjunto de terminais (KOZA et al., 1992).

Sem esta propriedade, programas inválidos poderiam aparecer na inicialização do processo (pela criação aleatória de árvores sintáticas) bem como pelo cruzamento (pela troca arbitrária de sub-árvores) ou mutação (pela substituição arbitrária de sub-árvores por árvores geradas aleatoriamente).

Mesmo com o conceito de *closure* introduzido, percebeu-se que ainda era possível encontrar indivíduos inválidos no espaço de busca. Por exemplo, nos indivíduos que representam uma função matemática, para o operador de divisão, o denominador pode assumir o valor 0 gerando um erro matemático, mesmo que o zero possa fazer parte do conjunto dos terminais. Essa possibilidade da geração de indivíduos inválidos estimulou a criação de soluções que buscassem a resolução deste problema, sendo proposto assim por Gruau (1996) o uso de gramáticas que auxiliassem nas restrições sintáticas dos indivíduos, surgindo posteriormente técnicas como a Programação Genética Orientada a Gramática (PGOG).

2.3.2.2 Programação Genética Orientada a Gramática

Koza et al. (1992) havia enfatizado em seu trabalho que a PG é como os algoritmos genéticos: o algoritmo de busca é independente do problema. Desta forma, a PG traz mais universalidade e a capacidade de buscar rapidamente um espaço de busca desconhecido. No entanto, estas questões motivaram o desenvolvimento de técnicas de programação genética mais avançadas, que restringissem o espaço de busca e abolissem a possibilidade de geração de indivíduos inválidos.

Paralelamente ao desenvolvimento das técnicas de Koza, gramáticas já eram utilizadas em outros ramos da aprendizagem de máquina, como a *Inductive Logic Programming* (ILP) (MUGGLETON, 1994). No entato, o primeiro trabalho notável do uso de gramáticas para controlar a busca dos algoritmos de programação genética veio por Whigham et al. (1995) em *Context-Free Grammar Genetic Programming*, onde eram utilizadas gramáticas livres de contexto.

Context-Free Grammars ou Gramáticas Livres de Contexto são formalmente definidas por uma 4-tupla $G = \{N, T, P, S\}$, onde N é o conjunto de não-terminais, T o conjunto de terminais, P o conjunto de produções e S o símbolo inicial. As produções

possuem a forma $x \models y$, onde $x \in N$ e $y \in \{T \cup N\}$ e há um conjunto de possibilidades para x. Cada uma das possíveis produções é delimitada pelo símbolo disjuntivo '|' (PAPPA; FREITAS, 2009). A Figura 20 mostra um exemplo de uma gramática livre de contexto na notação BNF (BACKUS et al., 1963) que gera expressões aritméticas.

Figura 20 – Gramática livre de contexto que gera expressões aritméticas.

$$
\begin{aligned}
&N = \{\langle \text{exp} \rangle, \langle \text{var} \rangle, \langle \text{op} \rangle\} \\
&T = \{x, +, -, /, *\} \\
&S = \langle \text{exp} \rangle \\
&P = \{ \\
&\qquad \langle \text{exp} \rangle \models \langle \text{var} \rangle \ \mid \ (\langle \text{exp} \rangle \langle \text{or} \rangle \langle \text{exp} \rangle) \\
&\qquad \langle \text{op} \rangle \models \texttt{+} \ \mid \ \texttt{-} \ \mid \ \texttt{/} \ \mid \ \texttt{*} \\
&\qquad \langle \text{var} \rangle \models \texttt{x} \\
&\}
\end{aligned}
$$

Fonte – Adaptado de Rodrigues (2002).

No exemplo da Figura 20, tem-se $\langle \text{exp} \rangle$, $\langle \text{var} \rangle$ e $\langle \text{op} \rangle$ como não-terminais. Por outro lado, o termo x e os operadores aritméticos $+$, $-$, $/$ e $*$ são os terminais. Sendo assim, um possível indivíduo gerado através da gramática seria o $x * (x + x)$. Logo, obedecendo as regras da gramática, é possível gerar toda uma população de indivíduos válidos que obedecem o que foi descrito. Os indivíduos gerados pela gramática, também podem ser representados por árvores sintáticas ou um vetor linear. Sendo assim, o espaço de busca agora é determinado por todas os indivíduos que podem ser criados pela livre combinação dos elementos dos terminais.

O processo de mapeamento do genótipo para um indivíduo, através da gramática, é feito pela leitura dos valores do genótipo que agora é constituído por um vetor de inteiros de 8 *bits* representando os genes.

Nesta operação, cada gene, representado por g, é lido da esquerda para a direita e tem seu valor convertido. Sendo N o total de opções disponíveis para produção atual da gramática, a conversão segue a seguinte fórmula:

$$f(g) = g \mod N$$

A título de exemplificação, na figura 20 o não-terminal ⟨op⟩ possui quatro opções de escolha (+, -, / e *), numeradas de 0 a 3, logo $N = 4$. Supondo que exista um genótipo representado pelo vetor $\vec{V} = \{32, 45, 87, 99, 1, 12, 34\}$ e seu terceiro elemento, 87, esteja sendo convertido seguindo a equação anterior, tem-se:

$$f(87) = 87 \mod 4 \therefore f(87) = 3$$

desta forma, o gene de valor 87 será representado pelo terminal de índice 3 que é o sinal de multiplicação (*).

Através disso, a operação converte as expressões não-terminais em expressões terminais ou outras não-terminais. Caso ainda haja expressões não-terminais quando a leitura do vetor chegar ao fim, o processo é reiniciado a partir do início do vetor até que não existam mais expressões não-terminais. Vale salientar que o primeiro não-terminal utilizado na conversão é o símbolo inicial presente no conjunto S na definição formal da gramática.

2.4 Otimização Multi-Objetivo

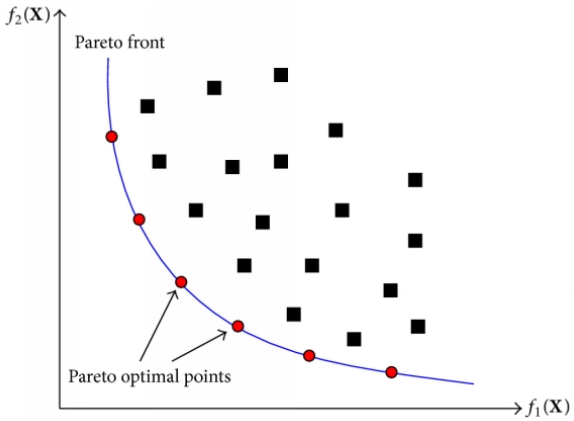
O objetivo dos algoritmos de Otimização Multi-Objetivo (OMO) é otimizar simultaneamente cada uma das avaliações dos objetivos previamente definidos. Isso significa que é necessário otimizar mais de uma única função objetivo. Dado um conjunto de n objetivos $O = \{O_1, O_2, ..., O_n\}$, o algoritmo de OMO precisará receber cada um dos n valores pertencentes à solução proposta. Com estas informações, ele medirá a aptidão final da solução, computada da seguinte forma:

$$\vec{f} = (f_1, f_2, \ldots, f_n)$$

onde $f_i (i \in [1, n])$ é o valor de aptidão da solução, de acordo com seu respectivo n objetivo de O. Para representar cada uma das soluções, a função $\vec{f}$ define um vetor, utilizado em sua comparação para selecionar os melhores.

Diferentemente da otimização mono-objetivo, na OMO, cada qualidade da solução é determinada por um vetor de valores de aptidão ao invés de um único valor. Logo, as solução são geralmente comparadas entre elas através da dominância de Pareto (DEB

Figura 21 – Exemplo de uma frente de Pareto.



Fonte – Neto et al. (2020).

et al., 2002). Formalmente, a dominância de Pareto e definida como dados dois vetores $\vec{x}$, $\vec{y} \in \mathbb{R}^n$, $\vec{x}$ domina $\vec{y}$ (denotado por $\vec{x} \prec \vec{y}$) se $\vec{x}$ supera $\vec{y}$ em ao menos um objetivo e $\vec{x}$ não é pior que $\vec{y}$ em quaisquer outros objetivos. Complementando, $\vec{x}$ é não-dominado se não há solução $\vec{x}_i$ na atual população, de tal modo que $\vec{x}_i \prec \vec{x}$. Todas as soluções não-dominadas no espaço objetivo são conhecidas como a frente de Pareto. A Figura 21 mostra a representação da frente de Pareto num problema de otimização que busca a minimização de duas funções objetivos.

3 Revisão da Literatura

A técnica de utilização de algoritmos baseados em gramáticas não é nova (O'NEILL; RYAN, 2001; RYAN et al., 1998), bem como sua utilização na construção de redes neurais (TSOULOS et al., 2008; AHMADIZAR et al., 2015; BALDOMINOS et al., 2018). Com o passar do tempo, otimizar CNNs tornou-se uma tarefa trabalhosa e recentemente diferentes estudos utilizaram a técnica de evolução gramatical para acelerar este processo.

Soltanian et al. (2013) aplicaram algoritmos evolucionários atrelados a um mapeamento por uma gramática bastante simples, porém recursiva, para geração de redes neurais na forma de MLPs. As redes foram otimizadas buscando a maximização da acurácia sobre a classificação de quatro conjuntos de dados (Breast cancer, Heart-Cleveland, Ionosphere e Iris) e alcançaram resultados superiores a outras técnicas.

Assunção et al. (2018) utilizaram Neuro-Evolução para propor o DENSER (*Deep Evolutionary StructurEd Representation*), que combina a utilização de algoritmos evolucionários com a *Dynamic Structured Grammatical Evolution* (DSGE). O DENSER foi utilizado para gerar CNNs bastante profundas buscando otimizá-las através da maximização da acurácia (mono-objetivo) em problemas de classificação sobre quatro datasets: MNIST, FashionMNIST, CIFAR-10 e CIFAR-100. Os resultados mostraram que a técnica é bastante promissora, produzindo CNNs competitivas a redes estado-da-arte da literatura. No entanto, além de otimizar as redes num processo de otimização mono-objetivo, a gramática utilizada no processo não permite que camadas de regularização do tipo *dropout* sejam adicionadas às redes. Outro ponto que deve ser destacado é o de que a gramática associada possui muitos parâmetros de configuração das redes em sua estrutura (e.g. *stride*, tamanho do *kernel*, número de filtros das camadas convolutivas e valores numéricos entre intervalos). Desta forma, o número de combinações dos terminais torna-se infinito, influenciando no tamanho do espaço de busca. Isto implica que o alto número de combinações pode causar a não convergência para áreas saudáveis do espaço de busca. Últimas pesquisas (ASSUNçãO et al., 2021) mostraram um aperfeiçoamento do DENSER, dando a possibilidade de obtenção redes otimizadas sem a necessidade de um segundo treinamento mais aprofundado.

Diniz et al. (2018) também utilizaram os princípios evolucionários associados a uma gramática livre de contexto para otimizar redes neurais. Sua gramática representava topologias que continham camadas convolucionais, de *pooling* e totalmente conectadas. O

algoritmo genético associado também buscava otimizar as redes através da maximização da acurácia em problemas de classificação, porém, em apenas um conjunto de dados, o CIFAR-10. Os resultados mostraram que o método é promissor, gerando arquiteturas leves e de baixa complexidade. Todavia, sua gramática possui um espaço de busca pequeno (54 possíveis indivíduos) e não permite a adição de algumas camadas de regularização (e.g. *dropout*, *batch normalization*) ao design das redes, além da otimização utilizar apenas um objetivo na convergência do algoritmo evolucionário.

Suganuma et al. (2017) utilizaram um algoritmo evolucionário associado a *Cartesian Genetic Programming* (CGP) para gerar CNNs de acordo com uma estrutura fixa. Sua estrutura permitia que seis tipos de camadas fossem adicionadas ao modelo (ConvBlock, *max pooling*, *average pooling*, *batch normalization*, soma, concatenação e ResBlock). Os testes foram realizados nos datasets CIFAR-10 e CIFAR-100 e o algoritmo evolucionário era guiado a buscar a maximização da acurácia. Os resultados mostraram que a técnica é bastante promissora gerando modelos competitivos a redes estado-da-arte. Por outro lado, o método não permitia a adição de camadas de regularização do tipo *dropout* ao modelo, além de guiar o algoritmo evolucionário num processo mono-objetivo.

Continuando, Lima et al. (2019) também utilizaram um algoritmo de otimização mono-objetivo, associado de uma nova gramática, para a criação de CNNs otimizadas voltadas a resolução de problemas de classificação em dois conjuntos de dados: CIFAR-10 e MNIST. Neste trabalho, a gramática construída é composta de várias regras que permitem a criação de uma espaço de busca infinito, uma vez que seu desenho permite a recursividade de blocos de camadas. Os resultados obtidos mostraram que a técnica conseguiu gerar arquiteturas competitivas a outros modelos mais robustos, porém complexos. Um obstáculo no uso desta abordagem recursiva é o fato do algoritmo evolucionário potencialmente não convergir para boas soluções devido ao tamanho do espaço de busca. Assim como nos outros trabalhos, o foto da otimização é a maximização da acurácia. Além disso, camadas de otimização como *batch normalization* não foram incluídas na gramática.

Conforme pôde ser avaliado, os trabalhos citados utilizaram diversas técnicas promissoras envolvendo a evolução gramatical para construção de modelos otimizados de arquiteturas de RNAs, alcançando resultados competitivos em relação a modelos estado-da-arte ou mais robustos. No entanto, nota-se a predominância da otimização mono-objetivo no processo de exploração do espaço de busca, sempre tendo apenas a

acurácia como função objetivo.

Outro ponto de destaque relaciona-se ao tamanho do espaço de busca das soluções, variando desde àqueles muito pequenos (Diniz et al. (2018)) aos de tamanho muito grande ou infinito (Assunção et al. (2018), Lima et al. (2019). Logo, técnicas que possuem um espaço de busca muito pequeno não proporcionam uma boa diversidade, já que a especialização das gramáticas não permitem tal feito. Enquanto isso, espaços de busca muito grandes podem influenciar na convergência da exploração do espaço de busca, fazendo com que o algoritmo evolutivo necessite de um número maior de gerações para explorar soluções mais especializadas.

Complementando, muitos dos trabalhos listados não permitem que suas gramáticas associadas deem a opção de inclusão de camadas de regularização na estrutura das arquiteturas. Ora, técnicas de regularização são importantes para o incremento da performance das redes, bem como, para reduzir a ocorrência de *overfitting*[1], logo é importante que técnicas como *dropout* e *batch normalization* façam parte do processo de criação das redes.

Por fim, este trabalho propõe a criação de um framework de evolução gramatical multi-objetivo que busque a criação, treinamento e otimização de CNNs otimizadas a partir da maximização de duas funções objetivo: acurácia e F_1-score. A gramática livre de contexto associada ao motor de busca do framework permite que as CNNs otimizadas sejam compostas de camadas convolutivas, de pooling e totalmente conectadas, além de permitir a adição de camadas de regularização como *dropout* e *batch normalization*, ausentes na maioria dos trabalhos relacionados.

A fim de facilitar a assimilação dos pontos citados, a Tabela 1 mostra um resumo comparativo das contribuições feitas pelos trabalhos relacionados e da atual proposta.

[1] *Overfitting*: É um problema comum no aprendizado de máquina que ocorre quando um modelo performa excepcionalmente bem nos dados de treino, porém falha ao predizer os dados de teste.

Tabela 1 – Comparativo entre os trabalhos relacionados e a técnica proposta.

Referência	***Suporte do método***	***Motor de busca***	***Conjuntos de dados***
Soltanian et al. (2013)	MLPs, função de ativação e operadores matemáticos.	Algoritmo Genético Básico	Breast cancer, Heart-Cleveland, Ionosphere e Iris
Assunção et al. (2018)	Camadas convolucionais, camadas de *pooling* (máximo e média), camadas totalmente conectadas, funções de ativação, *batch normalization*, número de filtros convolucionais, número de neurônios, tamanho do filtro convolutivo, *stride*, *padding*, taxa de aprendizado e valores númericos intervalados.	Algoritmo Genético Básico	CIFAR-10, CIFAR-100, MNIST e FashionMNIST
Diniz et al. (2018)	Camadas convolucionais, camadas de *pooling* (máximo) e camadas totalmente conectadas.	NSGA-II	CIFAR-10
Suganuma et al. (2017)	Camadas convolucionais, camadas de *pooling* (máximo e média), camadas totalmente conectadas, *batch normalization*, soma, concatenação e ResBlock.	*Cartesian Genetic Programming*	CIFAR-10
Lima et al. (2019)	Camadas convolucionais, camadas de *pooling* (máximo e média), camadas totalmente conectadas, *dropout*, funções de ativação, número de filtros convolutivos, taxas de aprendizado, *padding* e *stride*.	Algoritmo Genético Básico	CIFAR-10 e MNIST
Proposta	Camadas convolucionais, camadas de *pooling* (máximo), camadas totalmente conectadas, número de neurônios, *batch normalization*, *dropout* e taxas de aprendizado.	NSGA-II	CIFAR-10, MNIST, KMNIST e MedMNIST (PathMNIST, OCTMNIST e OrganMNIST Axial)

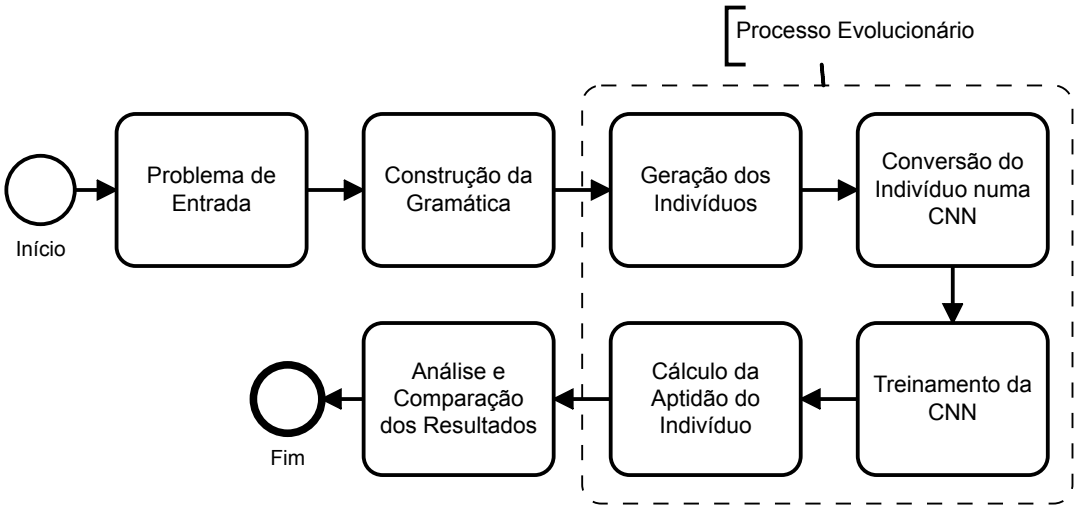
Fonte – O autor.

4 Proposta

Este estudo propõe um framework baseado em evolução gramatical para a geração de arquiteturas de CNNs otimizadas. O framework possui três componentes: o Motor de Busca, a Gramática Livre de Contexto e o Processo de Mapeamento dos indivíduos.

Para o desenvolvimento deste trabalho, optou-se pelo uso da Programação Genética associada a Evolução Gramatical para automação e otimização destas arquiteturas, buscando encontrar designs otimizados que oferecessem um bom equilíbrio entre alta performance e simplicidade, sem a necessidade do auxílio de especialistas. A Figura 22 mostra o processo proposto neste trabalho, destacando em pontilhado os itens que não necessitam da interferência humana. Sendo assim, este capítulo apresenta o framework proposto e as decisões tomadas no desenvolvimento de seus componentes ao longo da pesquisa.

Figura 22 – Processo do framework proposto.



Fonte – O autor.

4.1 Motor de Busca

De acordo com Neto et al. (2020), encontrar modelos de CNNs que possuam um bom balanço entre acurácia, generalização e precisão não é uma tarefa fácil. Logo, não é apropriado otimizar CNNs utilizando apenas algoritmos de otimização mono-objetivo, devido ao fato de que existem vários aspectos que devem ser levados em conta quando se

escolhe uma arquitetura ideal. Portanto, nós decidimos adotar o uso de um Algoritmo Evolucionário Multi-Objetivo (AEMO) amplamente utilizado, o NSGA-II (DEB et al., 2002). Este algoritmo obtém boa performance comparado a outros otimizadores multi-objetivos (SILVA et al., 2021b), sendo a escolha correta para este trabalho.

O NSGA-II é um Algoritmo Evolucionário Multi-Objetivo com uma forte estratégia elitista. Seu pseudocódigo pode ser visto no Algoritmo 2, onde inicialmente, antes do processo iterativo, uma nova população P com tamanho N' é gerada randomicamente. Quando iniciada a população, cada indivíduo tem tem seus valores de aptidão avaliados e, após isso, eles são ordenados de acordo com a ordem de Pareto (linhas 1 a 3). Uma vez concluída esta etapa, os operadores de seleção, cruzamento e mutação são aplicados, gerando assim uma nova população.

Algoritmo 2: Pseudocódigo do NSGA-II.

Entrada: $N', g, f_k(X) \triangleright N'$ membros evoluídos em g gerações para resolver $f_k(X)$

1: Gera uma população randômica P com tamanho N'
2: Avalia os valores dos objetivos;
3: Atribui um ranking (nível) baseado na frente de Pareto ordenada
4: Gera uma população filha: Seleção, Cruzamento e Mutação
5: **para** $i \leftarrow 1$ **até** g **faça**
6: | **para** *cada Pai e Filho em P* **faça**
7: | | Atribui um ranking (nível) baseado na frente de Pareto ordenada
8: | | Gera conjuntos de soluções não-dominadas
9: | | Determina a Distância de Aglomeração
10: | | Loop interno adicionando soluções para a próxima geração começando pela primeira frente até N' indivíduos
| **fim**
11: | Seleciona pontos na frente mais baixa e com alta distância de aglomeração
12: | Gera a próxima geração: Seleção, Cruzamento e Mutação
fim
13: **return** A melhor frente de Pareto

Dando continuidade, o processo iterativo é iniciado e para cada geração o algoritmo classifica os indivíduos de acordo com a não-dominância, produzindo assim algumas frentes de Pareto (linhas 7 e 8). Uma classificação baseada na Distância de Aglomeração (DA) é performada na frente para promover a diversidade. A DA representa a distância entre uma solução e seus vizinhos. Uma vez que cada solução tem sua própria distância calculada, elas são classificadas decrescentemente. Essa estratégia visa beneficiar soluções de fronteira colocadas em regiões do espaço de busca com menos vizinhos.

Na linha 11, a saída do procedimento de classificação é utilizada pelo método de seleção. Soluções com alta DA são selecionadas na frente inferior e providos para os operadores evolucionários (linha 12). Na seleção, um torneio binário é utilizado (nativo do NSGA-II), indivíduos são selecionados da frente inferior e, no caso de frentes iguais, a solução com a maior DA é escolhida. Depois disso, os operadores de cruzamento e mutação são empregados, e uma nova geração é criada. Este processo continua até o critério de parada ser alcançado. A saída do NSGA-II é a melhor frente de Pareto encontrada. É importante salientar que o motor de busca da nossa proposta pode ser substituído por qualquer AEMO que possa realizar o processo de exploração do espaço de busca, não sendo uma escolha exclusiva do NSGA-II. No entanto, para esta proposta, utiliza-se apenas o NSGA-II.

Neste trabalho, o NSGA-II é adotado para otimizar as arquiteturas das CNNs através da maximização de duas funções objetivos: acurácia e F_1-score. Abaixo, estão descritas essas dúas métricas, suas composições e o que representam.

4.1.1 Acurácia

A taxa de acerto total do classificador usado é chamada de acurácia, independentemente das classes no exemplo. Esta variável pode ser obtida por meio da seguinte expressão (THARWAT, 2021):

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.1}$$

Onde, TP e TN são verdadeiro positivo e verdadeiro negativo, e FP e FN são os valores falso positivo e falso negativo, respectivamente.

Acurácia é uma das mais comum métricas para avaliação da performance de um algoritmo em tarefas de classificação. Entretanto, ela é comumente utilizada quando a distribuição das classes do conjunto de dados são similares, onde TP e TN são mais importantes. Todavia, muitos conjuntos de dados da vida real não são balanceados. Desta forma, escolhemos uma métrica adicional para avaliar a qualidade das soluções candidatas.

4.1.2 F_1-score

A média harmônia entre a sensitividade ($Recall = \frac{TP}{TP+TN}$) e a precisão ($Precision = \frac{TP}{TP+FP}$) é chamada de F_1-score (THARWAT, 2021). Ela é primariamente utilizada para comparar a performance entre dois classificadores. A relação entre a sensitividade e precisão é que o F_1-score somente alcança valores altos quando ambas as métricas possuem valores altos. Em casos onde os conjuntos de dados não têm uma distribuição de classes equilibrada ou têm uma sobreposição de exemplos de classes diferentes, o F_1-score é a melhor métrica para avaliar a classificação dos modelos. O F_1-score pode ser representado pela seguinte expressão:

$$F_1\text{-score} = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{4.2}$$

4.2 Gramática Livre de Contexto

Esta seção apresenta o uso da Programação Genética no estudo, aplicada através do uso de Evolução Gramatical, descrevendo o processo de escolha das gramáticas testadas, bem como a composição final da gramática aplicada ao estudo. Dos componentes apresentados na proposta, a gramática é o mais importante. Nela são definidas as regras de composição e distribuição das camadas das redes, bem como o tamanho do espaço de busca pelo qual o AEMO navegará. Sendo assim, o processo de construção da gramática deve ser criterioso e exaustivo, devido a importância que este componente possui na proposta.

4.2.1 Evolução da Gramática Durante a Pesquisa

Durante o processo da pesquisa, várias gramáticas foram testadas até encontrar aquela que seria a proposta da pesquisa. Primeiramente, buscou-se uma gramática que apresentasse uma boa flexibilidade de geração de indivíduos, produzindo redes com pouca a muitas camadas, e que ao mesmo tempo não tivesse um espaço de busca muito grande, visando facilitar a convergência do algoritmo evolucionário.

Inicialmente, utilizou-se como base a gramática proposta por Diniz et al. (2018), que é bastante simples e gera 54 possíveis indivíduos em seu espaço de busca. A Figura 23 apresenta a estrutura desta gramática.

Figura 23 – Gramática de Diniz et al. (2018).

$$
\begin{aligned}
\langle \text{FINAL\_EXP} \rangle &\models \langle \text{EXP\_2} \rangle \langle \text{FC} \rangle \\
\langle \text{EXP\_2} \rangle &\models (\langle \text{EXP\_1} \rangle * \langle \text{M} \rangle) \\
\langle \text{EXP\_1} \rangle &\models (\langle \text{CONV} \rangle \langle \text{POOL} \rangle) \\
\langle \text{FC} \rangle &\models \texttt{fc} * \langle \text{K} \rangle \\
\langle \text{CONV} \rangle &\models \texttt{(conv)} * \langle \text{N} \rangle \\
\langle \text{POOL} \rangle &\models \texttt{pool} \mid \lambda \\
\langle \text{N} \rangle &\models \texttt{1} \mid \texttt{2} \mid \texttt{3} \\
\langle \text{K} \rangle &\models \texttt{0} \mid \texttt{1} \mid \texttt{2} \\
\langle \text{M} \rangle &\models \texttt{1} \mid \texttt{2} \mid \texttt{3}
\end{aligned}
$$

Fonte – Diniz et al. (2018)

Como pode-se perceber, a gramática de Diniz et al. (2018) é bastante simples, gerando indivíduos com três tipos de camadas: convolucionais, de *pooling* e totalmente conectadas, representadas por ⟨CONV⟩, ⟨POOL⟩ e ⟨FC⟩, respectivamente. Ela permite a configuração da profundidade das redes através das produções ⟨N⟩, ⟨K⟩ e ⟨M⟩.

Realizando os primeiros testes no conjunto de dados CIFAR-10 (KRIZHEVSKY; HINTON, 2009), percebeu-se que a estrutura da gramática e suas camadas disponíveis não seriam suficientes para formar resultados competitivos comparados à outras redes. Mesmo alterando as produções de profundidade das redes, a gramática não conseguiu gerar redes competitivas, produzindo indivíduos com valores de acurácia e F_1-score por volta de 76%. Sendo assim, foi criada uma nova gramática que pudesse comportar outras camadas e que fosse munida de mais parâmetros configuráveis, necessários para a construção CNNs mais complexas e competitivas (SILVA et al., 2021b). A primeira gramática elaborada pode ser vista na Figura 24.

A gramática elaborada possui duas novas camadas, *Dropout* e *Batch Normalization*, representadas por ⟨DROPOUT⟩ e ⟨BNORM⟩, respectivamente. Estas são camadas de regularização da rede que podem trazer ganhos significativos de performance às CNNs. As mesmas camadas podem estar presentes ou não no indivíduo final, representando sua ausência pelo símbolo λ. Além disso, foram adicionadas quatro opções de taxa de aprendizado, e outras quatro opções de números de neurônios presentes nas camadas totalmente conectadas. A nova gramática é capaz de gerar um espaço de busca com 2592 indivíduos, em sua configuração padrão. A adição dos novos parâmetros trouxe resultados significativos, aumentando para cerca de 87% os valores de acurácia e F_1-score das redes

Figura 24 – Primeira gramática aplicada no estudo.

$$
\begin{aligned}
\langle \text{FINAL\_EXP} \rangle &\models \langle \text{EXP\_2} \rangle \langle \text{FC} \rangle \texttt{*lr-} \langle \text{LR} \rangle \\
\langle \text{EXP\_2} \rangle &\models (\langle \text{EXP\_1} \rangle * \langle \text{M} \rangle) \\
\langle \text{EXP\_1} \rangle &\models (\langle \text{CONV} \rangle \langle \text{POOL} \rangle) \\
\langle \text{FC} \rangle &\models \texttt{fc} * \langle \text{K} \rangle * \langle \text{FCNEURONS} \rangle \\
\langle \text{POOL} \rangle &\models \texttt{pool-} \langle \text{DROPOUT} \rangle \mid \lambda \\
\langle \text{CONV} \rangle &\models (\texttt{conv} * \langle \text{N} \rangle) \langle \text{BNORM} \rangle \\
\langle \text{DROPOUT} \rangle &\models \texttt{dropout} \mid \lambda \\
\langle \text{BNORM} \rangle &\models \texttt{bnorm-} \mid \lambda \\
\langle \text{LR} \rangle &\models \texttt{0.1} \mid \texttt{0.01} \mid \texttt{0.001} \mid \texttt{0.0001} \\
\langle \text{FCNEURONS} \rangle &\models \texttt{64} \mid \texttt{128} \mid \texttt{256} \mid \texttt{512} \\
\langle \text{N} \rangle &\models \texttt{1} \mid \texttt{2} \mid \texttt{3} \\
\langle \text{K} \rangle &\models \texttt{0} \mid \texttt{1} \mid \texttt{2} \\
\langle \text{M} \rangle &\models \texttt{1} \mid \texttt{2} \mid \texttt{3}
\end{aligned}
$$

Fonte – Silva et al. (2021b).

treinadas sobre o conjunto de dados CIFAR-10.

Todavia, no decorrer do estudo, percebeu-se que o uso das produções ⟨N⟩, ⟨K⟩ e ⟨M⟩ para configuração da profundidade das redes, limitava a disposição das camadas da arquitetura geralmente formados pela sequência: camadas convolucionais, *batch normalization*, *pooling* e *dropout*, respectivamente. Desta forma, para alcançar resultados satisfatórios e indivíduos mais especializados, era necessário aumentar a diversidade de disposição das camadas, influenciando na performance do processo evolucionário e consequentemente na especialização dos indivíduos. Além disso, os experimentos mostraram que a taxa de aprendizado de 0.1 não produzia resultados satisfatórios, gerando ruídos no gráfico de convergência das métricas das redes no ato do treinamento. Sendo assim, fez-se necessário alterar a estrutura da gramática, para que o espaço de busca fosse expandido sem a necessidade de aumentar o número de convoluções das redes, a taxa de aprendizado de 0.1 foi removida. Logo, as produções ⟨N⟩, ⟨K⟩ e ⟨M⟩ foram substituídas por uma estrutura em blocos, que traziam mais diversidade à arquitetura. O resultado desta alteração pode ser visto na Figura 25.

Esta nova versão da gramática substituiu as produções ⟨N⟩, ⟨K⟩ e ⟨M⟩, responsáveis pela configuração de profundidade das redes, para uma estrutura de blocos especializados. A nova gramática apresenta um espaço de busca com cerca de 3871000 possíveis soluções. Novamente, foram realizados testes nesta segunda versão da gramática e percebeu-se que

Figura 25 – Segunda gramática aplicada no estudo.

⟨CNN⟩ ⊨ ⟨BLOCKS⟩⟨FLATTEN⟩⟨FC⟩⟨LR⟩
⟨BLOCKS⟩ ⊨ ⟨BLOCK⟩ | ⟨BLOCK⟩⟨BLOCK⟩ | ⟨BLOCK⟩⟨BLOCK⟩⟨BLOCK⟩
⟨BLOCK⟩ ⊨ ⟨CONVS⟩⟨POOLING⟩
⟨CONVS⟩ ⊨ ⟨CONV⟩ | ⟨CONV⟩⟨CONV⟩ | ⟨CONV⟩⟨CONV⟩⟨CONV⟩
⟨CONV⟩ ⊨ `(Conv` ⟨BNORM⟩`),`
⟨POOLING⟩ ⊨ `(MaxPool` ⟨DROPOUT⟩`),` | λ
⟨FLATTEN⟩ ⊨ `(Flatten),`
⟨FC⟩ ⊨ `(`⟨NFCS⟩⟨UNITS⟩⟨DROPOUT⟩`),` | λ
⟨BNORM⟩ ⊨ `BNorm` | λ
⟨DROPOUT⟩ ⊨ `Dropout` | λ
⟨LR⟩ ⊨ `(Lr`⟨RATES⟩`)`
⟨RATES⟩ ⊨ `0.01` | `0.001` | `0.0001`
⟨UNITS⟩ ⊨ `64` | `128` | `256` | `512`
⟨NFCS⟩ ⊨ `1` | `2`

Fonte – O autor.

o tamanho do espaço de busca da gramática estava influenciando na convergência dos resultados. Devido ao espaço de busca ser muito grande, o algoritmo evolucionário associado à gramática mostrou-se não tão eficiente na convergência dos indivíduos, demorando muito a busca por boas regiões no espaço de busca, não gerando assim resultados competitivos. Desta forma, fez-se necessário encontrar uma gramática que apresentasse um meio termo entre quantidade de indivíduos no espaço de busca e o nível de especialização das camadas das arquiteturas geradas.

4.2.2 Gramática da Proposta

Visando solucionar este balanço entre espaço de busca versus especialização dos indivíduos, foi elaborada uma nova gramática que apresentasse um espaço de busca menor, porém que pudesse, ainda assim, gerar indivíduos mais especializados que a segunda gramática proposta. O resultado pode ser visto na Figura 26.

A gramática proposta consiste num total de 16 regras de produções que permitem a criação de um espaço de busca com 6426 possíveis soluções, número bem menor em relação à gramática anterior, porém, maior que a primeira gramática aplicada ao estudo, que é o de 2592 possíveis indivíduos. Isso se deve ao fato da produção ⟨BLOCKS⟩ ter sido reescrita em forma do uso do parâmetro ⟨NBLOCKS⟩, reduzindo assim exponencialmente

Figura 26 – Gramática final aplicada no estudo.

$$
\begin{aligned}
\langle \text{CNN} \rangle &\models \langle \text{BLOCKS} \rangle \langle \text{FLATTEN} \rangle \langle \text{FC} \rangle \langle \text{SOFTMAX} \rangle \langle \text{LR} \rangle && (4.3)\\
\langle \text{BLOCKS} \rangle &\models \langle \text{NBLOCKS} \rangle \langle \text{BLOCK} \rangle && (4.4)\\
\langle \text{BLOCK} \rangle &\models \langle \text{CONVS} \rangle \langle \text{POOLING} \rangle && (4.5)\\
\langle \text{CONVS} \rangle &\models \langle \text{CONV} \rangle \mid \langle \text{CONV} \rangle \langle \text{CONV} \rangle \mid \langle \text{CONV} \rangle \langle \text{CONV} \rangle \langle \text{CONV} \rangle && (4.6)\\
\langle \text{CONV} \rangle &\models \texttt{(Conv } \langle \text{BNORM} \rangle \texttt{),} && (4.7)\\
\langle \text{POOLING} \rangle &\models \texttt{(MaxPool } \langle \text{DROPOUT} \rangle \texttt{),} \mid \lambda && (4.8)\\
\langle \text{FLATTEN} \rangle &\models \texttt{(Flatten),} && (4.9)\\
\langle \text{SOFTMAX} \rangle &\models \texttt{(Softmax),} && (4.10)\\
\langle \text{FC} \rangle &\models \texttt{(}\langle \text{NFCS} \rangle \langle \text{UNITS} \rangle \langle \text{DROPOUT} \rangle \texttt{),} \mid \lambda && (4.11)\\
\langle \text{BNORM} \rangle &\models \texttt{BNorm} \mid \lambda && (4.12)\\
\langle \text{DROPOUT} \rangle &\models \texttt{Dropout} \mid \lambda && (4.13)\\
\langle \text{LR} \rangle &\models \texttt{(Lr}\langle \text{RATES} \rangle \texttt{)} && (4.14)\\
\langle \text{RATES} \rangle &\models \texttt{0.01} \mid \texttt{0.001} \mid \texttt{0.0001} && (4.15)\\
\langle \text{UNITS} \rangle &\models \texttt{64} \mid \texttt{128} \mid \texttt{256} \mid \texttt{512} && (4.16)\\
\langle \text{NBLOCKS} \rangle &\models \texttt{1} \mid \texttt{2} \mid \texttt{3} && (4.17)\\
\langle \text{NFCS} \rangle &\models \texttt{1} \mid \texttt{2} && (4.18)
\end{aligned}
$$

Fonte – O autor.

o número de possíveis indivíduos do espaço de busca.

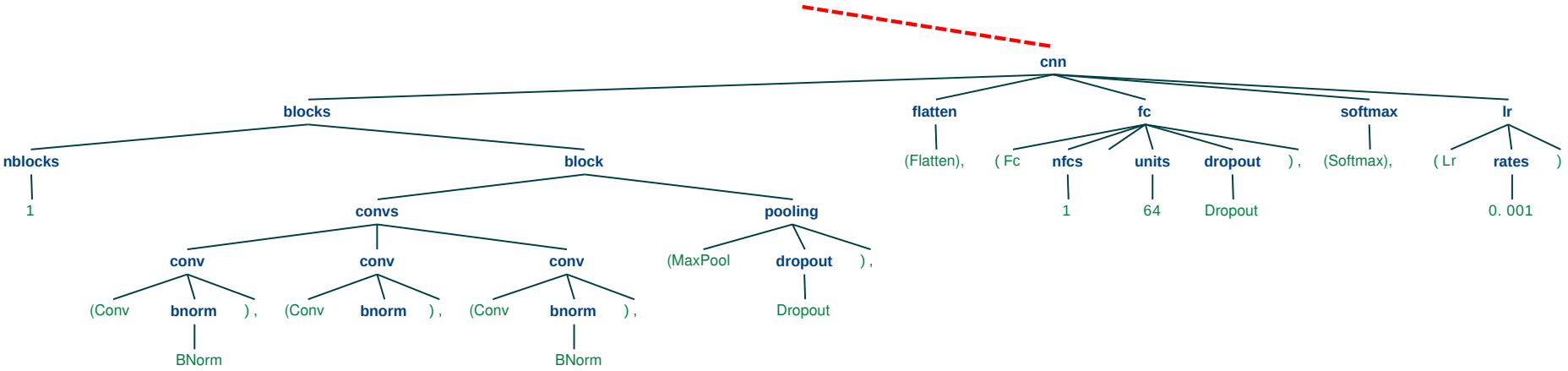
A regra de produção principal ⟨CNN⟩, na linha 4.3, define toda a estrutura da CNN, contendo os blocos convolutivos, a camada de achatamento do dados, as camadas totalmente conectadas e, por fim, a taxa de aprendizado. A regra de produção ⟨BLOCKS⟩ representa os blocos convolutivos da rede. Ela é definida por ⟨NBLOCKS⟩⟨BLOCK⟩, na linha 4.4, onde ⟨NBLOCKS⟩ é o número de vezes com que a regra de produção ⟨BLOCK⟩ irá se repetir na rede, que varia de 1 a 3 conforme a linha 4.17.

Dando prosseguimento, na linha 4.5, a regra de produção ⟨BLOCK⟩ define o conjunto de camadas convolutivas, ⟨CONVS⟩, seguida ou não da camada de *pooling* ⟨POOLING⟩, detalhada na linha 4.8. Na linha 4.6, é definido o bloco convolutivo que pode ser composto por 1, 2 ou 3 camadas convolutivas. Cada camada convolutiva, linha 4.7, pode ser seguida ou não (representado pelo símbolo λ) por camadas de normalização em lotes ⟨BNORM⟩, detalhado na linha 4.12. A regra de produção ⟨POOLING⟩ mostra que as camadas de *pooling* podem ser seguidas de camadas de *dropout* ou não.

As regras de produção ⟨FLATTEN⟩ e ⟨SOFTMAX⟩, linhas 4.9 e 4.10, definem a camada de achatamento dos dados e a camada de classificação da rede, respectivamente. Dando continuidade, a regra de produção ⟨FC⟩ define as camadas totalmente conectadas

que podem existir ou não na gramática. Caso existam, elas podem ter 4 valores de número de neurônios e podem se repetir 1 ou 2 vezes, conforme a regra de produção ⟨NFCS⟩. Além disso, as camadas totalmente conectadas podem ser seguidas ou não por camadas de *dropout*. Mais detalhes estão descritos na linha 4.11. Finalizando, a regra de produção ⟨LR⟩ define a taxa de aprendizado com que a CNN será otimizada, que pode assumir 3 valores conforme definido na linha 4.15. A Figura 27 mostra a representação em árvore de um indivíduo gerado pela gramática.

Figura 27 – Representação em árvore de um indivíduo gerado pela gramática da proposta.



Fonte – O autor.

Como pode ser visto, a gramática proposta é flexível e permite a criação de pequenas a grandes redes, desde àquelas com apenas camadas convolucionais até outras mais complexas com camadas convolucionais, *pooling*, normalização em lotes, *dropout* e totalmente conectadas. Além disso, a gramática permite que produções como ⟨CONVS⟩, ⟨BLOCKS⟩ e ⟨NFCS⟩ sejam alteradas para que gerem redes mais profundas, e consequentemente mais complexas.

Alguns parâmetros importantes na construção de CNNs permaneceram fixos, como função de ativação, *padding*, *stride* e número de filtros convolutivos. Nas camadas convolucionais, a função de ativação utilizada foi a ReLU por ter um baixo poder computacional e apresentar, ainda assim, bons resultados em comparação à outras como sigmóide e tangente hiperbólica (GLOROT et al., 2011). As camadas de *pooling* foram configuradas para o tipo de maximização, realçando assim os aspectos mais importantes das imagens. O otimizador utilizado nos treinamentos foi o Adam por trazer resultados rápidos e satisfatórios comparado a outros otimizadores (KINGMA; BA, 2017). A função de erro utilizada foi a *Categorical Cross-Entropy* (ZHANG; SABUNCU, 2018). Os modelos das CNNs foram configurados para serem treinados em 30 épocas com um tamanho de lote de 128. Mais detalhes sobre estes parâmetros podem ser vistos na Tabela 2.

4.3 Processo de Mapeamento dos Indivíduos

Uma vez definidos o Motor de Busca e a Gramática Livre de Contexto que serão utilizados, é preciso estabelecer o Processo de Mapeamento dos Indivíduos gerados através da gramática para CNNs executáveis. Uma vez que o motor de busca é executado, para cada iteração do processo, a gramática livre de contexto associada cria um genótipo, geralmente uma cadeira de inteiros, que representa o indivíduo. Este genótipo é convertido numa cadeia de caracteres de acordo com as produções definidas na gramática.

Basicamente, o processo de mapeamento deve produzir CNNs a partir de cinco categorias extraídas das cadeias de caracteres resultante da produção principal ⟨CNN⟩, que são: Convoluções, Camada de Achatamento dos Dados, Camadas Totalmente Conectadas, Classificador e Otimizador. Conforme a construção da gramática, cada categoria citada possui uma produção associada, que são: ⟨BLOCKS⟩ para a convoluções, ⟨FLATTEN⟩ para a camada de achatamento dos dados, ⟨FC⟩ para as camadas totalmente conectadas, ⟨SOFTMAX⟩ para a camada de saída que representa o classificador e finalmente ⟨LR⟩

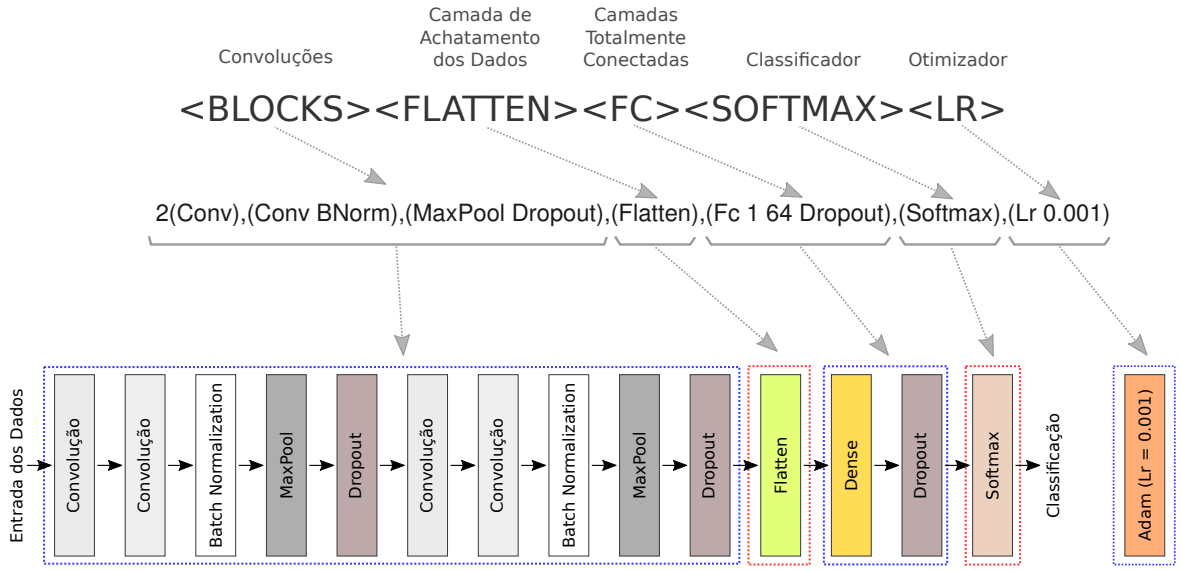
Tabela 2 – Parâmetros fixos nas CNNs.

Parâmetro	***Valor***
Núcleo convolutivo	3 x 3
# de filtros convolutivos	Começa por 32;
	duplica a cada 2 convoluções.
Stride	1
Núcleo de *pooling*	2 x 2
Taxa de *dropout*	0.25 após camada de *pooling*;
	0.50 após camada totalmente conectada.
Função de ativação	ReLU para convoluções;
	Softmax para a última camada totalmente conectada.
Otimizador	Adam
Função de perda	*Categorical Cross-Entropy*
# de épocas	30
Tamanho do lote	128
Parada antecipada	monitor=val_accuracy, mode=max
	patience=10, baseline=0.5

contendo a taxa de aprendizado do otimizador utilizado. A Figura 28 mostra o exemplo de um mapeamento destas produções de um indivíduo para uma CNN.

O indivíduo da Figura 28, conforme a produção ⟨BLOCKS⟩ que representa as convoluções, possui duas sequências de camadas que seguem a ordem: Convolução, Convolução, *Batch Normalization*, *MaxPool* e *Dropout*. Logo após, tem-se a camada de achatamento dos dados, *Flatten*, seguida da produção ⟨FC⟩ que decompõe-se em uma camada totalmente conectada com 64 neurônios, seguida de uma camada de *Dropout*. Seguindo, tem-se a camada de classificação, vinda da produção ⟨SOFTMAX⟩, que nada mais é do que uma camada totalmente conectada com a função de ativação *Softmax* e a quantidade de neurônios igual ao número de classes do conjunto de dados associado. Por fim, não menos importante, tem-se a implementação do otimizador *Adam* configurado com a taxa de aprendizado proveniente da produção ⟨LR⟩.

Figura 28 – Mapeamento de um indivíduo para uma CNN.



Fonte – O autor.

4.4 Considerações do Capítulo

Neste capítulo foi explanado o processo de desenvolvimento e progresso do método proposto, principalmente a gramática livre de contexto associada, a evolução de seu estudo e a maneira como o mapeamento e otimização das CNNs é feito. Além disso, a composição do motor de busca do método proposto foi detalhada, explicando seu AEMO associado e as métricas que guiam o processo de exploração do espaço de busca gerado pela gramática. Mais detalhes sobre a execução do método proposto encontram-se nos próximos capítulos.

5 Metodologia

Esta seção apresenta o conjunto de dados utilizado nos experimentos, detalhes sobre a metodologia e configurações adotadas, métodos comparativos e as métricas utilizadas para a avaliação das soluções. Todas os experimentos foram executados utilizando o ambiente Google Colab [1] com o uso de TPU ativado. A liguaguem utilizada para implementar a abordagem proposta e os métodos comparativos foi a Python™.

5.1 Conjunto de Dados

Para realizar os experimentos, foram utilizados 6 conjuntos de dados para bechmark da proposta, que são o CIFAR-10, MNIST, KMNIST e o MedMNIST (PathMNIST, OCTMNIST e OrganMNIST Axial). A proposta de usar vários conjuntos de dados é a de mostrar o quanto o framework se adapta na resolução de problemas em diferentes tipos e tamanhos de imagens. Todos estes conjuntos de dados são multi-classes e estão divididos naqueles com imagens de 1 ou 3 canais de cores RGB. Cada conjunto de dados é detalhado abaixo.

5.1.1 CIFAR-10

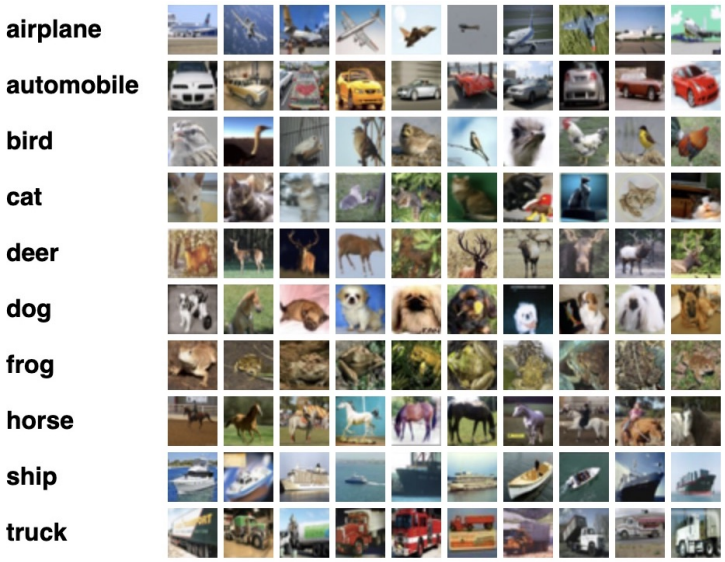
O CIFAR-10 (KRIZHEVSKY; HINTON, 2009) é um conjunto de dados amplamente utilizado por Redes Neurais Profundas para problemas de classificação. Ele tem cerca de 60000 imagens da vida real com 32×32 pixels, todas coloridas e divididas em 10 categorias (6000 imagens por classe). Ele contém cerca de 50000 imagens para treinamento e 10000 imagens para testes. O lote de testes consiste em 10000 imagens selecionada aleatoriamente, sendo 1000 para cada classe. O lotes de treinamento incluem as imagens remanecentes numa ordem aleatória. Portanto, alguns lotes de treinamento podem conter mais imagens de uma classe que de outra. Para cada lote de treinamento há um total de 5000 imagens. A Figura 29 mostra exemplos das imagens contidas no conjunto.

5.1.2 MNIST

O MNIST (LECUN et al., 2010) é um conjunto de dados amplamente utilizado para tarefas de classificação de imagens. É uma dos conjunto de dados mais famosos na

[1] https://colab.research.google.com/

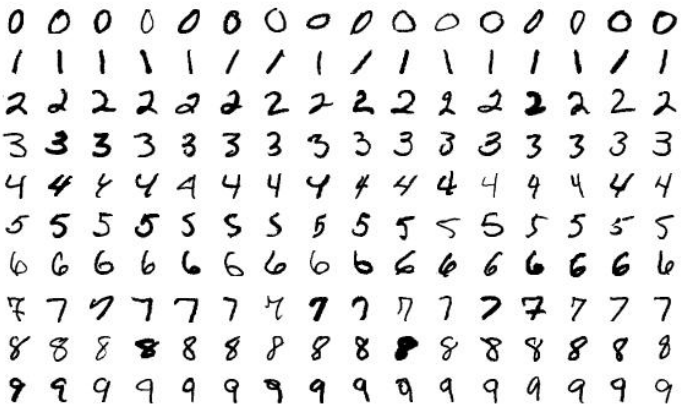
Figura 29 – Exemplos de imagens do conjunto de dados CIFAR-10 (KRIZHEVSKY; HINTON, 2009).



Fonte – Adaptado do website do CIFAR-10.

comunidade de aprendizagem de máquina. Ele consiste em 70000 imagens com 28×28 pixels em tons de cinza, sendo 60000 imagens para treinamento e 10000 imagens para teste. As imagens estão divididas em 10 categorias e representam numerais escritos à mão de 0 a 9. A Figura 30 traz uma amostra das imagens contidas no conjunto.

Figura 30 – Exemplos de imagens do conjunto de dados MNIST.



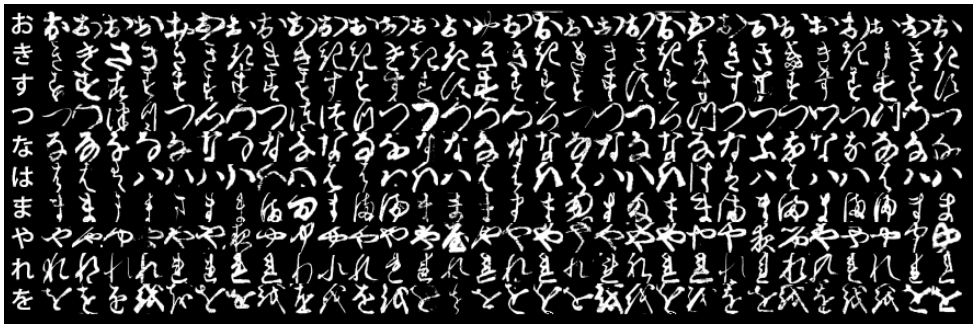
Fonte – Adaptado do website do MNIST.

5.1.3 KMNIST

O conjunto de dados KMNIST ou Kuzushiji-MNIST (CLANUWAT et al., 2018), é um substituto imediato para o conjunto de dados MNIST. Ele contém 70000 imagens com

28×28 pixels em escala de cinza e também contém 10 classes. As classes representam cada uma das 10 linhas do Hiragama japonês. Mesmo que o Kuzushiji-MNIST tenha sido criado como um substituto do MNIST, as características do Hentaigana e números arábicos são completamente diferentes. Desta forma, esta é uma das razões pelas quais o KMNIST ser mais difícil que o MNIST. A Figura 31 traz uma amostra das imagens contidas do KMNIST.

Figura 31 – As 10 classes do KMNIST (CLANUWAT et al., 2018) onde a primeira coluna mostra o contraparte hiragana morderna de cada caractere.



Fonte – Adaptado do website do KMNIST.

5.1.4 MedMNIST

O conjunto MedMNIST (YANG et al., 2021) é uma coleção de dez subconjuntos de imagens médicas pre-processadas e liberadas sob a licença *Creative Commons*. Todas as imagens estão padronizadas na dimensão de 28×28 pixels. Destes dez subconjuntos foram escolhidos três deles, que são os que possuem mais dados de treinamento, validação e testes, além de estarem divididos em múltiplas classes, necessárias para a tarefa de classificação. São eles: PathMNIST, OCTMNIST e OrganMNIST Axial.

5.1.4.1 PathMNIST

Conjunto composto por imagens coloridas histológicas de câncer colorretal coradas de hematoxilina e eosina. Nove tipos de tecidos são envolvidos, resultando numa tarefa de classificação multi-classe. As imagens têm dimensão 28×28 pixels. No total, são 89996, 10004 e 7180 imagens de treinamento, validação e testes, respectivamente.

5.1.4.2 OCTMNIST

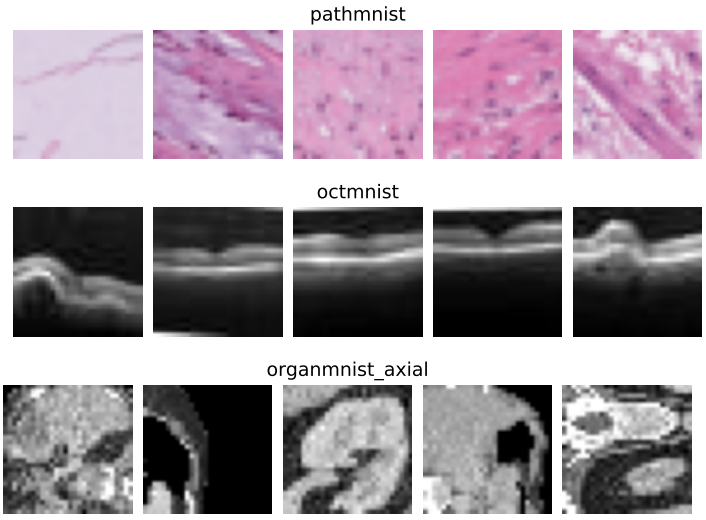
Conjunto composto por imagens de tomografia de coerência óptica (OCT) válidas para doenças da retina. As imagens são divididas em 4 classes, levando a uma tarefa de classificação multi-classe. As imagens têm dimensão 28×28 pixels e estão em escala de cinza. No total, são 97477, 10832 e 1000 imagens de treinamento, validação e testes, respectivamente.

5.1.4.3 OrganMNIST Axial

Conjunto composto por imagens 3D de tomografias computadorizadas da Referência de Segmentação do Tumor do Fígado (LiTS). As imagens 3D são transformadas em escala de cinza e recordadas em planos, neste caso, o plano axial. As imagens têm dimensão 28×28 pixels. No total, são 34581, 6491, 17778 imagens de treinamento, validação e testes, respectivamente.

A Figura 32 traz exemplos de imagens contidas nos três conjuntos de dados selecionados.

Figura 32 – Exemplos de imagens contidas nos conjuntos de dados selecionados no MedMNIST (YANG et al., 2021).



Fonte – Adaptado do website do MedMNIST V1.

5.1.5 Resumo Comparativo dos Conjuntos de Dados

A Tabela 3 traz o quadro comparativo dos conjuntos de dados aplicados ao estudo, mostrando sua dimensão, número de canais de cores, total de classes presentes, total de imagens e a quantidade individual de imagens para treinamento, testes e validação.

Tabela 3 – Quadro comparativo dos conjuntos de dados utilizados.

Nome	*Dimensão*	*# de Canais*	*# de Classes*	*# de Imagens*	*Treinamento*	*Teste*	*Validação*
CIFAR-10	32×32	3	10	60000	50000	8000	2000
MNIST	28×28	1	10	70000	60000	8000	2000
KMNIST	28×28	1	10	70000	60000	8000	2000
PathMNIST	28×28	3	9	107180	89996	10004	7180
OCTMNIST	28×28	1	4	109309	97477	10832	1000
OrganMNIST Axial	28×28	1	11	58850	34581	6491	17778

5.2 Configurações Experimentais

Para a execução do processo de Evolução Gramatical, foi utilizado o framework PonyGE2 (FENTON et al., 2017) que fornece de uma maneira simples e objetiva um ambiente para execução de processos evolucionários associados a gramáticas livres de contexto. O PonyGE2 foi configurado para representar os indivíduos gerados numa estrutura de vetor de inteiros.

Para a construção e execução das redes neurais convolucionais, utilizou-se a biblioteca de código aberto Keras (CHOLLET et al., 2015). Além disso, para a normalização dos dados e assistência no processo de treinamento das CNNs, foi utilizada a biblioteca Scikit-Learn (PEDREGOSA et al., 2011).

O AEMO foi configurado com os seguintes parâmetros: o processo foi executado numa população de 50 indivíduos através de 30 gerações. O operador de seleção adotado foi o de Torneio Binário, nativo do NSGA-II. Para o cruzamento dos indivíduos, foi utilizado o operador *One Point Crossover*, com a probabilidade de 75%. O operador de mutação utilizado foi o *Int Flip Per Codon*, com uma probalidade de 1%. A Tabela 4 sumariza os parâmetros utilizados no algoritmo evolucionário.

Durante o processo evolucionário, para cada geração, cada indivíduo foi convertido numa CNN válida que foi treinada por 3 vezes, tendo as médias da acurácia e F_1-score

Tabela 4 – Parâmetros do Algoritmo Evolucionário.

Parâmetro	***Valor***
Número de gerações	30
Tamanho da população	50
Operador de mutação	*Int Flip Per Codon*
Operador de cruzamento	*One Point Crossover*
Taxa de mutação	0.01
Taxa de cruzamento	0.75
Tamanho do toneio	2
Elite size	1
Algoritmo de seleção	*nsga2_selection*
Algoritmo de reposição	*nsga2_replacement*

calculadas, formando assim o valor de aptidão do indivíduo.

No treinamento das CNNs, as redes foram treinadas num período de 30 épocas configurados com um tamanho de lote de treinamento de 128. Pelo fato do processo evolucionário ter um longo tempo de duração, dependendo do número de épocas configurado no treinamento, optou-se por treinar as redes num menor número de épocas.

Além disso, buscando identificar indivíduos que produzem redes com baixa performance, de acordo com um limiar, e interromper o seu treinamento, foi aplicado o recurso de parada antecipada disponível na biblioteca Keras.

Neste recurso os valores de aptidão dos indivíduos são monitorados, de forma que, durante o processo de treinamento, caso o indivíduo atual em suas primeiras 10 épocas não alcance uma performance de ao menos 50% nas métricas, o seu treinamento é interrompido, dando celeridade ao processo evolucionário.

5.3 Métodos Comparativos

Após executar o processo evolucionário para cada conjunto de dados, o motor de busca retorna a frente de Pareto que contém uma ou mais soluções não-dominadas. No

estudo, para todos os conjuntos de dados, a frente de Pareto retornada continha apenas um indivíduo não-dominado.

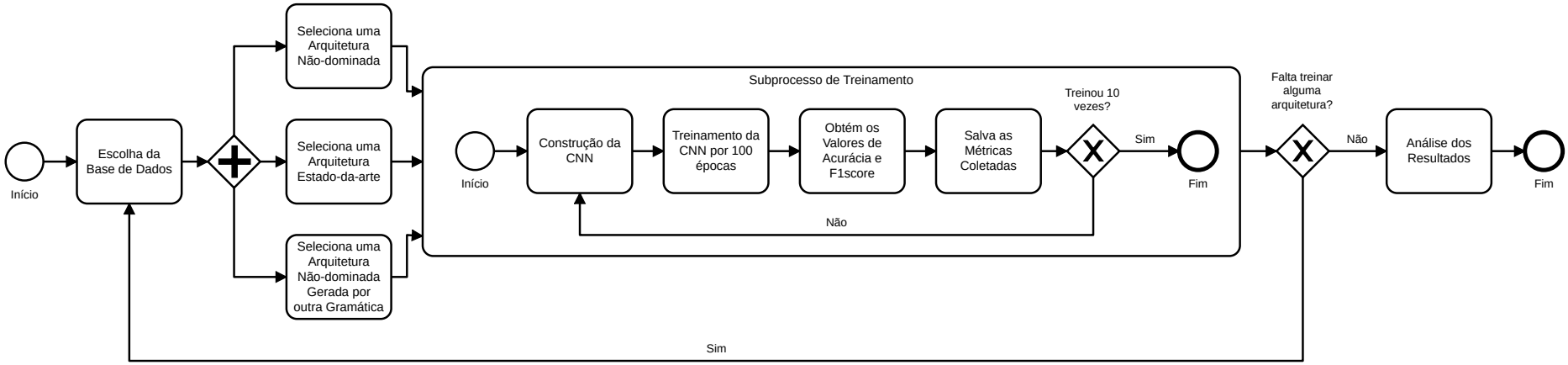
Para cada conjunto de dados, seus correspondentes indivíduos não-dominados são nomeados como: *Proposta CIFAR-10*, *Proposta MNIST*, *Proposta KMNIST*, *Proposta PathMNIST*, *Proposta OCTMNIST* e *Proposta OrganMNIST* para os conjuntos de dados CIFAR-10, MNIST, KMNIST, PathMNIST, OCTMNIST e OrganMNIST Axial, repectivamente.

Um vez identificadas as arquiteturas não-dominadas, cada solução é comparada com outras 4 arquiteturas estado-da-arte: 1) InceptionV3 (SZEGEDY et al., 2016), 2) ResNet50V2 (HE et al., 2016b), 3) EfficientNetB1 (TAN; LE, 2019) e 4) DenseNet169 (HUANG et al., 2017), e com 3 arquiteturas também geradas por outras gramáticas livres de contexto: 1) Diniz et al. (2018), 2) Lima et al. (2019) e 3) Assunção et al. (2018). Suas performances são comparadas em termos de acurácia e F_1-score.

As três arquiteturas geradas por outras gramáticas foram obtidas através da execução do mesmo processo evolucionário utilizado na proposta, todavia utilizando suas respectivas gramáticas. Já para as outras arquiteturas estado-da-arte faz-se necessário que as imagens utilizadas para o seu treinamento sejam redimensionadas para um tamanho mínimo aceitável por todas elas, que neste caso é o de 75×75 píxels.

Dando continuidade, as arquiteturas são treinadas novamente agora com um número de épocas maior, no caso, 100 épocas e sem o critério de parada antecipada. O treinamento é aplicado sob os mesmos parâmetros, tamanho de lote igual a 128, mesma proporção entre dados de treinamento/validação/testes, otimizador Adam (KINGMA; BA, 2017). A taxa de aprendizado utilizada varia de acordo com o indivíduo, sendo utilizada a que foi retornada para cada modelo no processo evolucionário. Já para as CNNs estado-da-arte, a taxa de aprendizado utilizada foi de 0.001. Cada arquitetura, para cada conjunto de dados, é treinada 20 vezes para produzir a média e desvio padrão das métricas coletadas. A Figura 33 mostra o processo utilizado.

Figura 33 – Processo final de treinamento das arquiteturas.



Fonte – O autor.

6 Resultados

Neste capítulo do trabalho são discutidos os resultados obtidos nos experimentos do framework proposto, seguindo a metodologia apresentada no Capítulo 5. Além disso, é elaborado um comparativo da técnica proposta em relação a outros modelos de CNNs estado-da-arte, bem como, modelos obtidos por outras técnicas que também utilizam a evolução gramatical.

6.1 Resultados Experimentais

Conforme a metodologia descrita no Capítulo 5, primeiramente é executado o processo evolucionário do framework proposto para cada conjunto de dados, utilizando a gramática da proposta além das outras três gramáticas escolhidas para comparação (Assunção et al. (2018), Diniz et al. (2018), Lima et al. (2019)). Desta forma são obtidos os indivíduos otimizados segundo o processo proposto. A Tabela 5 mostra a composição das camadas, bem como o otimizador e taxa de aprendizado, de cada um dos indivíduos obtidos pelo framework proposto para cada conjunto de dados. É possível identificar que em quase todos os modelos otimizados obtidos há a presença de camadas de regularização (*dropout*, *batch normalization*), que é um dos diferenciais estabelecidos na estrutura da gramática da proposta em relação às gramáticas dos trabalhos relacionados.

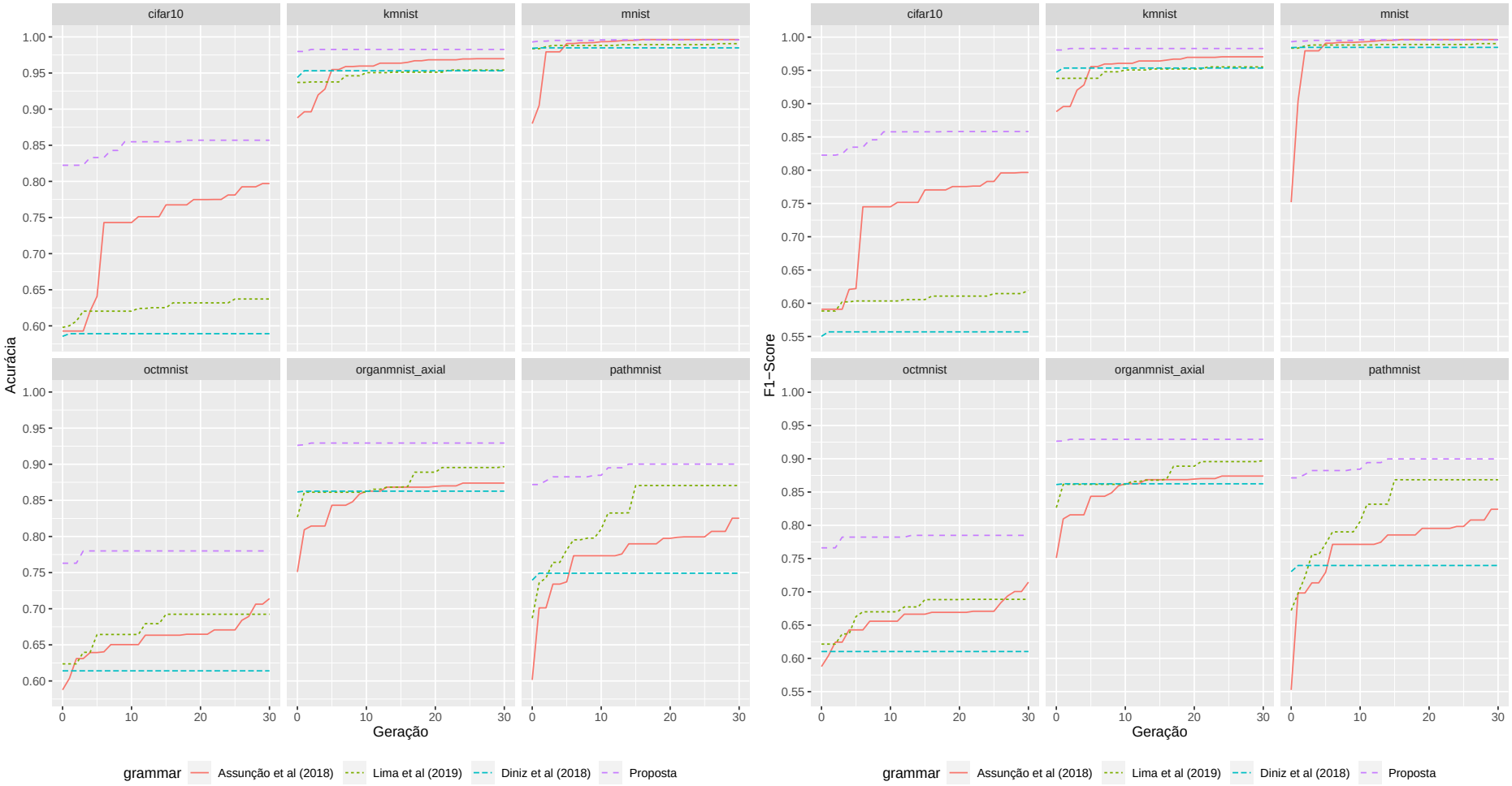
Através da Figura 34 é possível verificar o gráfico de convergência do processo evolucionário para cada métrica em cada conjunto de dados. É possível notar que a gramática da proposta consegue gerar indivíduos que alcançam bons resultados de acurácia e F_1-score já nas primeiras gerações do processo, tomando a liderança na maximização das métricas em quase todos os conjuntos de dados, e o empate com a gramática de Assunção et al. (2018) no conjunto de dados MNIST. Desta forma, o framework proposto consegue gerar bons indivíduos já nas primeiras gerações do processo.

Tabela 5 – Disposição das camadas dos indivíduos obtidos pela técnica proposta em cada conjunto de dados.

CIFAR10	***MNIST***	***KMNIST***	***PathMNIST***	***OCTMNIST***	***OrganMNIST Axial***
Conv2D(32, 'relu')	Conv2D(32, 'relu')	Conv2D(32, 'relu')	Conv2D(32, 'relu')	Conv2D(32, 'relu')	Conv2D(32, 'relu')
Batch Normalization	Batch Normalization	Batch Normalization	Conv2D(32, 'relu')	Conv2D(32, 'relu')	Conv2D(32, 'relu')
Conv2D(32, 'relu')	Conv2D(32, 'relu')	Conv2D(32, 'relu')	Conv2D(64, 'relu')	Batch Normalization	Batch Normalization
Conv2D(64, 'relu')	Conv2D(64, 'relu')	Batch Normalization	MaxPooling2D	Conv2D(64, 'relu')	MaxPooling2D
Batch Normalization	Batch Normalization	MaxPooling2D	Dropout(0.25)	MaxPooling2D	Dropout(0.25)
MaxPooling2D	MaxPooling2D			Dropout(0.25)	
Dropout(0.25)					
Conv2D(64, 'relu')	Conv2D(64, 'relu')	Conv2D(64, 'relu')	Conv2D(64, 'relu')	Conv2D(64, 'relu')	Conv2D(64, 'relu')
Batch Normalization	Batch Normalization	Batch Normalization	Conv2D(128, 'relu')	Conv2D(128, 'relu')	Conv2D(64, 'relu')
Conv2D(128, 'relu')	Conv2D(128, 'relu')	Conv2D(64, 'relu')	Conv2D(128, 'relu')	Batch Normalization	Batch Normalization
Conv2D(128, 'relu')	Conv2D(128, 'relu')	Batch Normalization	MaxPooling2D	Conv2D(128, 'relu')	MaxPooling2D
Batch Normalization	Batch Normalization	MaxPooling2D	Dropout(0.25)	MaxPooling2D	Dropout(0.25)
MaxPooling2D	MaxPooling2D			Dropout(0.25)	
Dropout(0.25)					
Conv2D(256, 'relu')	Conv2D(256, 'relu')	Conv2D(128, 'relu')		Conv2D(256, 'relu')	
Batch Normalization	Batch Normalization	Batch Normalization		Conv2D(256, 'relu')	
Conv2D(256, 'relu')	Conv2D(256, 'relu')	Conv2D(128, 'relu')		Batch Normalization	
Conv2D(512, 'relu')	Conv2D(512, 'relu')	Batch Normalization		Conv2D(512, 'relu')	
Batch Normalization	Batch Normalization	MaxPooling2D		MaxPooling2D	
MaxPooling2D	MaxPooling2D			Dropout(0.25)	
Dropout(0.25)					
Flatten	Flatten	Flatten	Flatten	Flatten	Flatten
Dense(512, 'relu')	Dense(256, 'relu')	Dense(128, 'relu')	Dense(64, 'relu')	Dense(64, 'relu')	Dense(64, 'relu')
Dropout(0.5)	Dense(256, 'relu')	Dropout(0.5)		Dense(64, 'relu')	Dropout(0.5)
		Dense(128, 'relu')			
		Dropout(0.5)			
Dense(10, 'softmax')	Dense(10, 'softmax')	Dense(10, 'softmax')	Dense(9, 'softmax')	Dense(4, 'softmax')	Dense(11, 'softmax')
Adam(lr=0.001)	Adam(lr=0.0001)	Adam(lr=0.001)	Adam(lr=0.001)	Adam(lr=0.001)	Adam(lr=0.001)

Fonte – O autor.

Figura 34 – Convergência do processo evolutivo aplicado às gramáticas nos conjuntos de dados.



Fonte – O autor.

No entato, na maioria dos conjuntos de dados, outras gramáticas como Assunção et al. (2018) e Lima et al. (2019) aparentam necessitar de um maior número de gerações no processo evolucionário para alcançar métricas com valores superiores. Logo, este maior número de gerações requer consequentemente um maior tempo de execução do processo. A gramática de Diniz et al. (2018) gerou seus melhores indivíduos já nas primeiras gerações, porém ainda assim com uma performance inferior comparada às outras gramáticas. Isso ocorre provavelmente pelo fato da gramática ter um espaço de busca de apenas 54 indivíduos.

Uma vez identificados os indivíduos otimizados de cada gramática, seguindo o fluxo da metodologia, executa-se agora o retreino destes indivíduos por um número maior de épocas, no caso, 100 épocas. Além disso, as CNNs estado-da-arte listadas na metodologia também são treinadas por igual período de épocas. No geral, para todas as CNNs (indivíduos otimizados e CNNs estado-da-arte), o treinamento de 100 épocas é executado 20 vezes a fim de obter dados de média e desvio padrão das métricas estabelecidas.

Vale salientar que os modelos de arquiteturas estado-da-arte encontram-se com os seus parâmetros padrão para todos os conjuntos de dados, e podem assim obter resultados superiores ao da técnica proposta caso venham a ser otimizados por um especialista da área. Todavia, para encontrar esses parâmetros otimizados há um custo de tempo e esforço. Sendo assim, a abordagem proposta propõe encontrar redes competitivas, mais simples e que não necessitam de um conhecimento de especialista para encontrar seus parâmetros otimizados, diferindo assim das arquiteturas estado-da-arte.

A Tabela 6 mostra o resultado do retreino dos modelos, bem como detalhes como número de camadas, total de parâmetros treináveis, tempo de treinamento utilizado e tamanho em *megabytes* do modelo treinado. É importante enfatizar que o número de camadas das arquiteturas foi obtido através do tamanho da lista `model.layers` que é um atributo presente na classe `Model` que representa um modelo de CNN na biblioteca Keras.

Tabela 6 – Análise comparativa da performance dos modelos treinados nos seis conjunto de dados.

Dataset	*CNN*	*Acurácia*	*F_1-score*	*Camadas*	*Parâmetros*	*Tamanho(MB)*	*Tempo(s)*
cifar10	Assunção et al. (ASSUNÇÃO et al., 2018)	0,7926 (± 0,0060)	0,7944 (± 0,0055)	12	43.672.266	166,64	1392, 65 (±61, 65)
	DenseNet169 (HUANG et al., 2017)	0,8353 (± 0,0211)	0,8358 (± 0,0207)	598	12.659.530	146,47	4080, 45 (±317, 03)
	Diniz et al. (DINIZ et al., 2018)	0,6015 (± 0,0210)	0,5689 (± 0,0220)	8	38.890	0,49	985, 80 (±65, 56)
	EfficientNetB1 (TAN; LE, 2019)	0,7624 (± 0,0104)	0,7633 (± 0,0103)	343	6.588.049	76,50	2479, 70 (±33, 91)
	InceptionV3 (SZEGEDY et al., 2016)	0,8168 (± 0,0132)	0,8179 (± 0,0135)	314	21.823.274	250,85	2621, 85 (±80, 85)
	Lima et al. (LIMA et al., 2019)	0,6313 (± 0,0079)	0,6135 (± 0,0086)	7	57.482	0,70	928, 80 (±51, 42)
	Proposta	**0,8648 (± 0,0046)**	**0,8665 (± 0,0047)**	26	6.556.586	75,17	1104, 60 (±46, 82)
	ResNet50V2 (HE et al., 2016b)	0,7861 (± 0,0142)	0,7873 (± 0,0137)	193	23.585.290	270,46	1893, 10 (±47, 47)
	p-value	9.94×10^{-23}	8.71×10^{-20}				
kmnist	Assunção et al. (ASSUNÇÃO et al., 2018)	0,9667 (± 0,0025)	0,9671 (± 0,0024)	17	4.977.418	19,05	1160, 65 (±66, 37)
	DenseNet169 (HUANG et al., 2017)	0,9844 (± 0,0033)	0,9846 (± 0,0031)	598	12.653.258	146,40	4633, 25 (±241, 11)
	Diniz et al. (DINIZ et al., 2018)	0,9536 (± 0,0059)	0,9539 (± 0,0059)	8	413.802	4,78	1061, 05 (±54, 13)
	EfficientNetB1 (TAN; LE, 2019)	0,9762 (± 0,0048)	0,9765 (± 0,0048)	343	6.587.469	76,50	3259, 55 (±125, 63)
	InceptionV3 (SZEGEDY et al., 2016)	**0,9859 (± 0,0017)**	**0,9861 (± 0,0017)**	314	21.822.698	250,84	3192, 50 (±104, 27)
	Lima et al. (LIMA et al., 2019)	0,9530 (± 0,0038)	0,9535 (± 0,0035)	7	138.186	1,62	1019, 75 (±69, 62)

Dataset	*CNN*	*Acurácia*	F_1*-score*	*Camadas*	*Parâmetros*	*Tamanho(MB)*	*Tempo(s)*
	Proposta	0,9807 (± 0,0009)	0,9811 (± 0,0009)	22	453.610	5,32	$1181,70$ ($\pm 95,22$)
	ResNet50V2 (HE et al., 2016b)	0,9820 (± 0,0012)	0,9822 (± 0,0011)	193	23.579.018	270,39	$2444,20$ ($\pm 150,03$)
	p-value	1.67×10^{-22}	2.76×10^{-22}				
mnist	Assunção et al. (ASSUNÇÃO et al., 2018)	0,9063 (± 0,2699)	0,8945 (± 0,3059)	15	11.034.346	42,14	$1219,40$ ($\pm 54,72$)
	DenseNet169 (HUANG et al., 2017)	0,9951 (± 0,0011)	0,9950 (± 0,0011)	598	12.653.258	146,40	$4823,65$ ($\pm 65,56$)
	Diniz et al. (DINIZ et al., 2018)	0,9851 (± 0,0025)	0,9849 (± 0,0025)	6	25.258	0,32	$1032,90$ ($\pm 78,86$)
	EfficientNetB1 (TAN; LE, 2019)	0,9930 (± 0,0020)	0,9928 (± 0,0021)	343	6.587.469	76,50	$3204,80$ ($\pm 172,30$)
	InceptionV3 (SZEGEDY et al., 2016)	0,9953 (± 0,0010)	0,9951 (± 0,0009)	314	21.822.698	250,84	$3170,75$ ($\pm 144,20$)
	Lima et al. (LIMA et al., 2019)	0,9845 (± 0,0043)	0,9843 (± 0,0042)	7	13.348.234	152,81	$1245,80$ ($\pm 84,81$)
	Proposta	**0,9955 (± 0,0006)**	**0,9955 (± 0,0006)**	24	3.604.330	41,39	$1313,15$ ($\pm 85,05$)
	ResNet50V2 (HE et al., 2016b)	0,9947 (± 0,0007)	0,9945 (± 0,0008)	193	23.579.018	270,39	$2412,80$ ($\pm 130,29$)
	p-value	6.10×10^{-22}	6.29×10^{-22}				
octmnist	Assunção et al. (ASSUNÇÃO et al., 2018)	0,7041 (± 0,0276)	0,7076 (± 0,0270)	18	3.789.412	14,52	$1754,25$ ($\pm 87,50$)
	DenseNet169 (HUANG et al., 2017)	0,7645 (± 0,0193)	**0,7689 (± 0,0190)**	598	12.643.268	146,28	$7414,15$ ($\pm 262,58$)
	Diniz et al. (DINIZ et al., 2018)	0,6418 (± 0,0384)	0,6390 (± 0,0388)	8	30.372	0,39	$1599,30$ ($\pm 156,61$)
	EfficientNetB1 (TAN; LE, 2019)	0,7350 (± 0,0304)	0,7383 (± 0,0294)	343	6.579.783	76,41	$4728,25$ ($\pm 351,10$)
	InceptionV3 (SZEGEDY et al., 2016)	0,7137 (± 0,0252)	0,7178 (± 0,0249)	314	21.810.404	250,70	$4970,65$ ($\pm 210,63$)

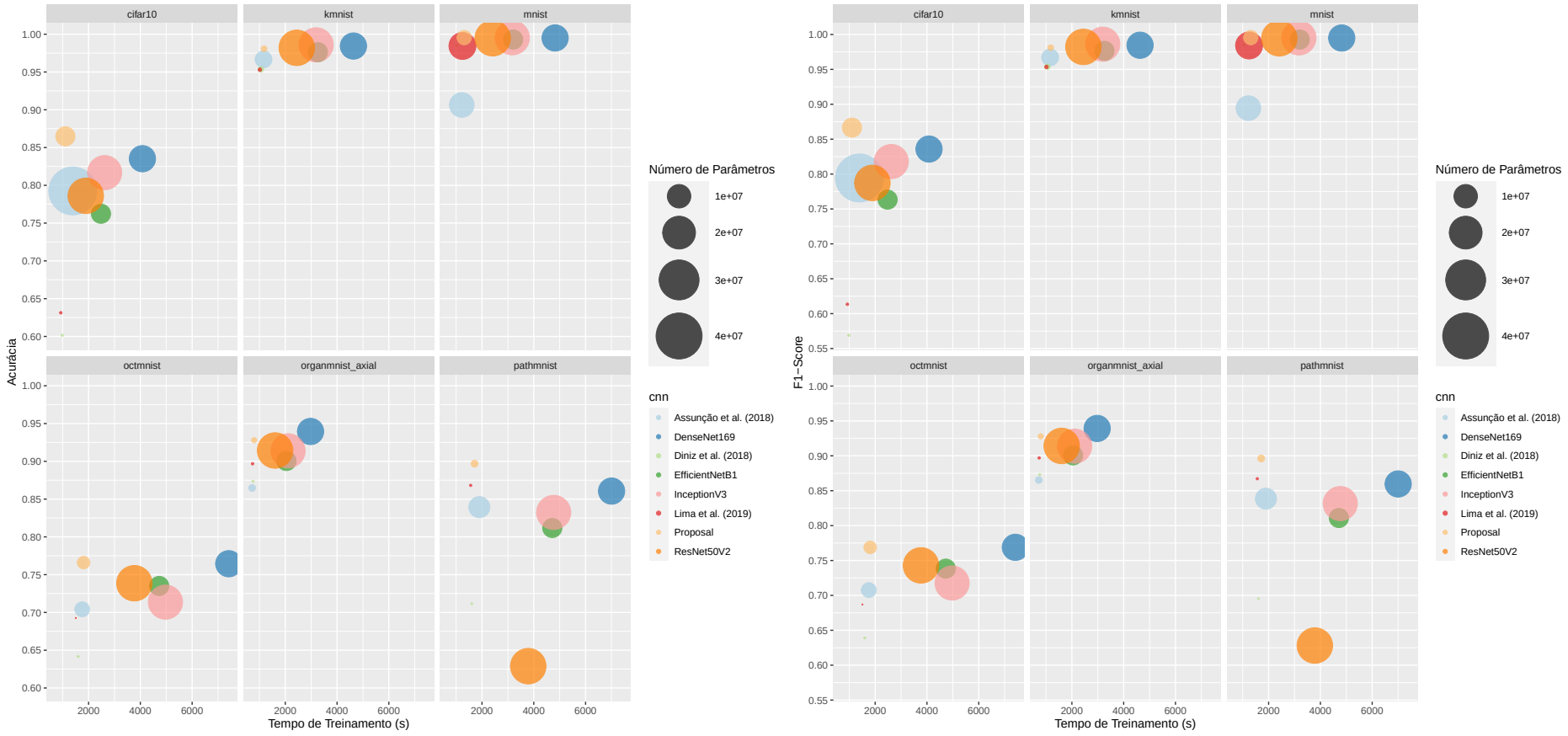
Dataset	*CNN*	*Acurácia*	F_1*-score*	*Camadas*	*Parâmetros*	*Tamanho(MB)*	*Tempo(s)*
	Lima et al. (LIMA et al., 2019)	0,6927 (± 0,0118)	0,6870 (± 0,0118)	7	12.036	0,18	1509, 90 (±70, 73)
	Proposta	**0,7660 (± 0,0125)**	0,7688 (± 0,0119)	24	2.652.900	30,49	1808, 75 (±100, 88)
	ResNet50V2 (HE et al., 2016b)	0,7387 (± 0,0272)	0,7430 (± 0,0273)	193	23.566.724	270,25	3768, 80 (±99, 29)
	p-value	1.02×10^{-22}	1.84×10^{-21}				
organmnist_axial	Assunção et al. (ASSUNÇÃO et al., 2018)	0,8647 (± 0,0026)	0,8653 (± 0,0024)	11	663.499	2,57	720, 55 (±22, 71)
	DenseNet169 (HUANG et al., 2017)	**0,9395 (± 0,0030)**	**0,9392 (± 0,0030)**	598	12.654.923	146,42	2976, 40 (±119, 02)
	Diniz et al. (DINIZ et al., 2018)	0,8736 (± 0,0093)	0,8732 (± 0,0093)	8	34.411	0,44	758, 25 (±19, 14)
	EfficientNetB1 (TAN; LE, 2019)	0,9005 (± 0,0285)	0,9007 (± 0,0283)	343	6.588.750	76,51	2045, 30 (±39, 06)
	InceptionV3 (SZEGEDY et al., 2016)	0,9139 (± 0,0271)	0,9138 (± 0,0269)	314	21.824.747	250,87	2107, 05 (±86, 09)
	Lima et al. (LIMA et al., 2019)	0,8968 (± 0,0073)	0,8970 (± 0,0073)	8	54.091	0,67	736, 70 (±26, 04)
	Proposta	0,9280 (± 0,0037)	0,9279 (± 0,0038)	20	361.835	4,25	799, 10 (±49, 55)
	ResNet50V2 (HE et al., 2016b)	0,9143 (± 0,0779)	0,9142 (± 0,0779)	193	23.581.067	270,42	1607, 10 (±74, 19)
	p-value	9.25×10^{-23}	1.53×10^{-22}				
pathmnist	Assunção et al. (ASSUNÇÃO et al., 2018)	0,8393 (± 0,0131)	0,8386 (± 0,0130)	11	7.979.593	30,48	1897, 55 (±58, 50)
	DenseNet169 (HUANG et al., 2017)	0,8606 (± 0,0547)	0,8597 (± 0,0545)	598	12.657.865	146,45	7004, 35 (±375, 00)
	Diniz et al. (DINIZ et al., 2018)	0,7116 (± 0,1338)	0,6953 (± 0,1695)	6	24.265	0,31	1607, 45 (±70, 19)
	EfficientNetB1 (TAN; LE, 2019)	0,8117 (± 0,0405)	0,8109 (± 0,0405)	343	6.586.768	76,49	4714, 30 (±305, 98)

Dataset	*CNN*	*Acurácia*	F_1*-score*	*Camadas*	*Parâmetros*	*Tamanho(MB)*	*Tempo(s)*
	InceptionV3 (SZEGEDY et al., 2016)	0,8324 (± 0,0906)	0,8316 (± 0,0905)	314	21.821.225	250,82	$4767,60$ ($\pm 179,74$)
	Lima et al. (LIMA et al., 2019)	0,8682 (± 0,0098)	0,8672 (± 0,0097)	8	46.729	0,58	$1563,25$ ($\pm 74,69$)
	Proposta	**0,8969 (± 0,0081)**	**0,8962 (± 0,0081)**	14	689.065	7,96	$1712,30$ ($\pm 35,22$)
	ResNet50V2 (HE et al., 2016b)	0,6289 (± 0,1420)	0,6282 (± 0,1421)	193	23.583.241	270,44	$3786,05$ ($\pm 140,21$)
	p-value	9.00×10^{-23}	1.17×10^{-22}				

Através dos resultados obtidos, pode-se notar que os modelos gerados pela técnica proposta para os conjuntos de dados CIFAR-10, MNIST e PathMNIST conseguem performar melhor que os outros modelos estado-da-arte, bem como, os gerados por outras gramáticas livres de contexto. Quanto ao conjunto de dados OCTMNIST, o modelo gerado pela técnica proposta alcançou resultados compatíveis como a rede DenseNet169 (HUANG et al., 2017), no entanto, o modelo da proposta possui apenas 24 camadas contra 598 computados na DenseNet169. Consequentemente, o modelo da proposta possui aproximadamente 79% menos parâmetros treináveis em relação à outra rede compatível. Já para os conjuntos de dados KMNIST e OrganMNIST Axial, os modelos gerados pela técnica proposta não superam as CNNs estado-da-arte em ambas as métricas, no entanto, ainda assim obtêm performance superior aos modelos gerados pelas outras gramáticas.

Outro ponto importante que pode ser observado através dos dados é o do nível da eficiência na construção dos modelos gerados pela proposta em relação aos outros. Mesmo o motor de busca tendo apenas duas funções objetivo configuradas (acurácia e F_1-score), os modelos obtidos apresentaram números de parâmetros treináveis e tempo de processamento bastante inferiores em relação aos outros modelos de redes estado-da-arte. Sendo assim, mesmo explicitamente não configurado, a técnica proposta tende a gerar modelos mais simples e de treinamento mais rápido. Desta forma, os modelos produzidos pelo framework podem ser utilizados em contextos onde o baixo uso de memória e baixo tempo de processamento são requisitos importantes. Os gráficos de bolhas da Figura 35 mostram esta relação entra o número de parâmetros treináveis, o tempo de treinamento e as métricas obtidas para cada conjunto de dados.

Figura 35 – Relação entre as métricas obtidas e o número de parâmetros treináveis dos modelos.



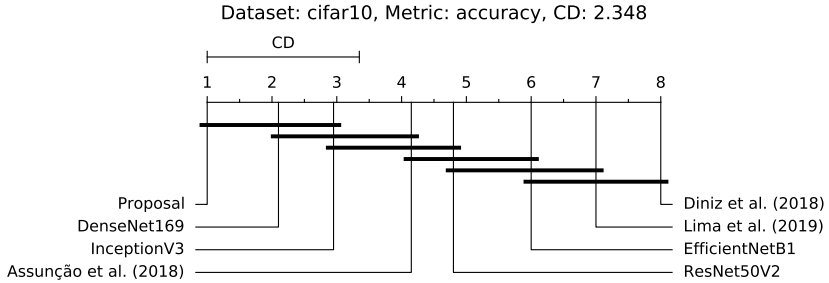
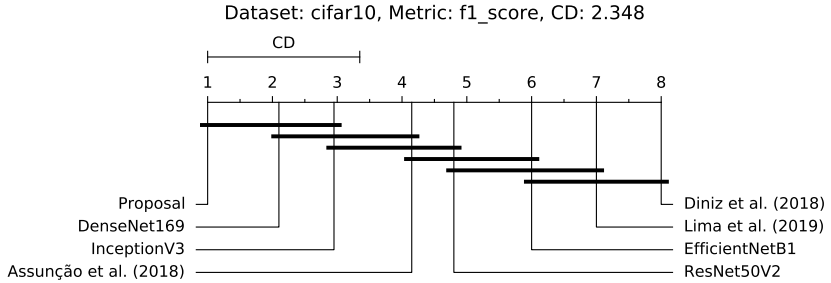
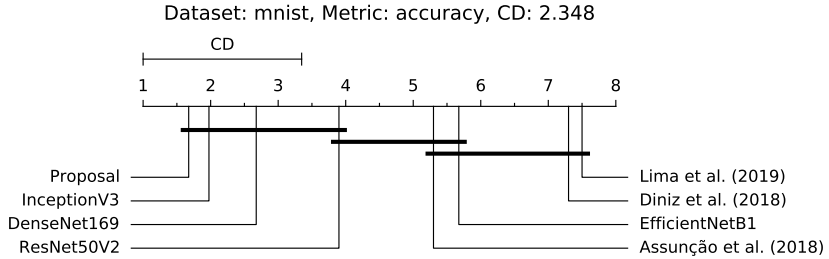
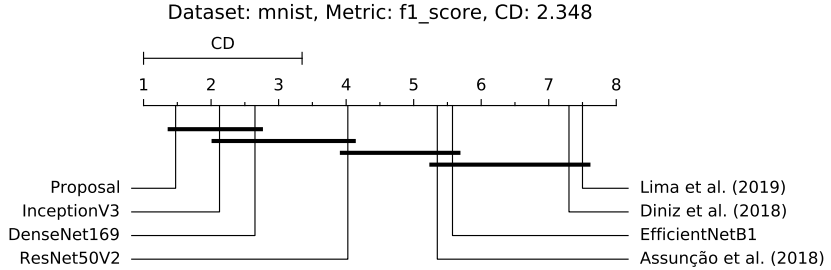
Fonte – O autor.

6.2 Análise estatística

Para realizar uma comparação justa entre os modelos faz-se necessário avaliar a significância estatística dos resultados obtidos. Desta forma, define-se como hipótese nula a de que não há diferença estatística entre as médias das métricas obtidas em cada modelo para cada conjunto de dados. Dando continuidade, aplica-se o teste não-paramétrico de *Friedman* (PEREIRA et al., 2015) com um nível de significância estatística de $\alpha = 0.05$ obtendo-se os valores *p-value* apresentados na Tabela 6. Valores *p-value* menores que α indicam que a hipótese nula foi rejeitada. Como pode ser observado na tabela, em todas as métricas e em todos os conjuntos de dados a hipótese nula foi rejeitada, indicando que há ao menos um resultado estatisticamente diferente dos outros em cada métrica e conjunto de dados.

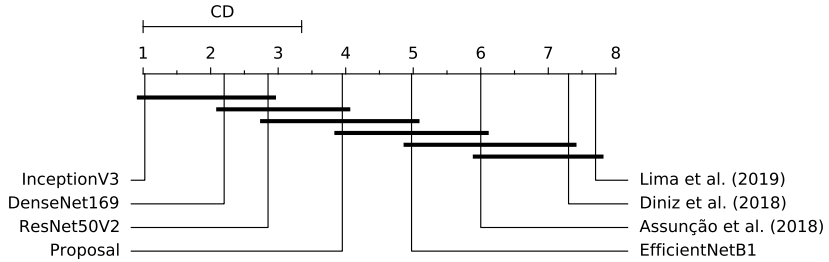
Para identificar esses grupos de resultados que apresentam similaridade estatística em múltiplas comparações, faz-se necessário aplicar o *post hoc Nemenyi* (NEMENYI, 1963) para obter-se a classificação média dos resultados. Uma vez obtida a classificação, é possível calcular a Diferença Crítica (*Critical Difference* ou CD) dos valores dos resultados e assim criar a representação visual destes dados, chamado Diagrama de Diferença Crítica, conforme pode ser visto na Tabela 7. A linha vertical de cada CNN mostra sua classificação média em comparação com os outros modelos de redes em cada conjunto de dados e métrica. As barras horizontais em negrito mostram a diferença crítica da comparação. Modelos separados por uma distância menor que o valor CD são estatisticamente indistinguíveis. Modelos separados por uma distância maior do que a distância crítica têm uma diferença estatisticamente significativa no desempenho.

Tabela 7 – Diagramas de Diferença Crítica para teste estatístico Friedman-Nemenyi.

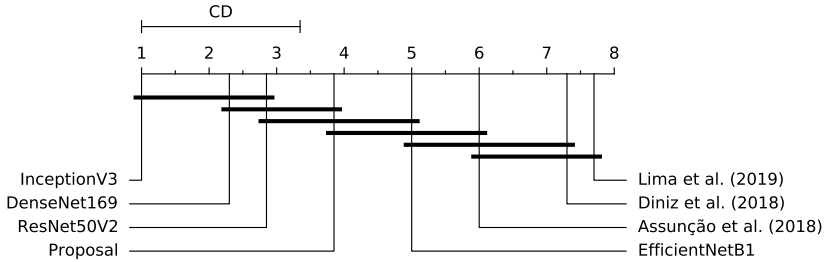
Acurácia	F_1***-score***
	
	

Acurácia	F_1*-score*

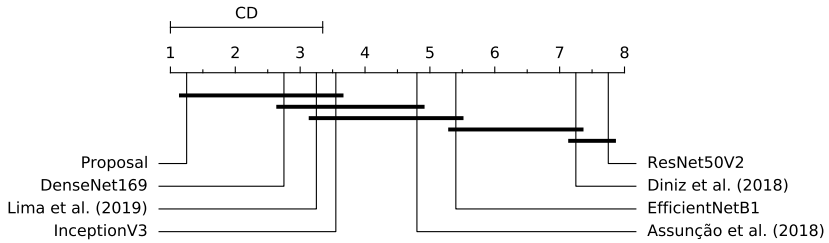
Dataset: kmnist, Metric: accuracy, CD: 2.348



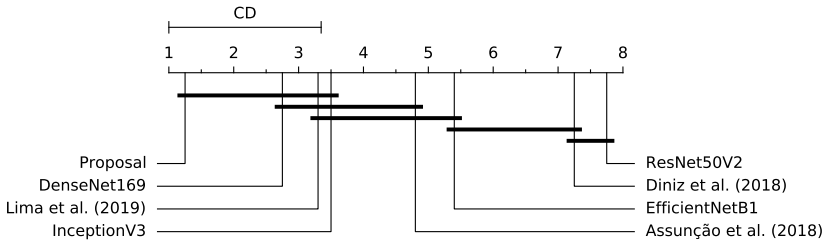
Dataset: kmnist, Metric: f1_score, CD: 2.348



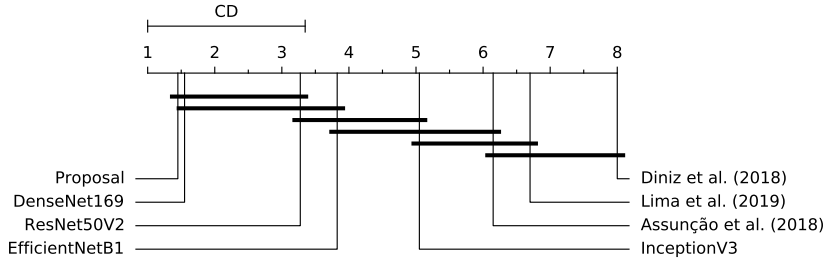
Dataset: pathmnist, Metric: accuracy, CD: 2.348



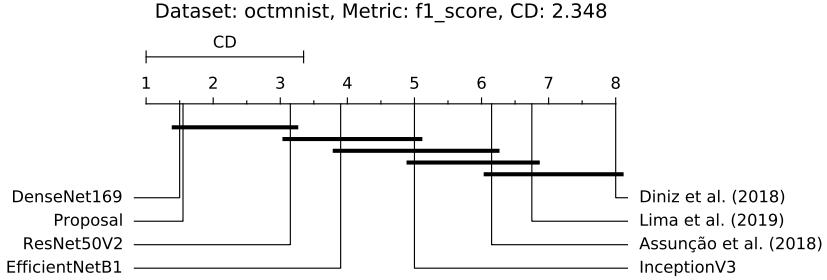
Dataset: pathmnist, Metric: f1_score, CD: 2.348

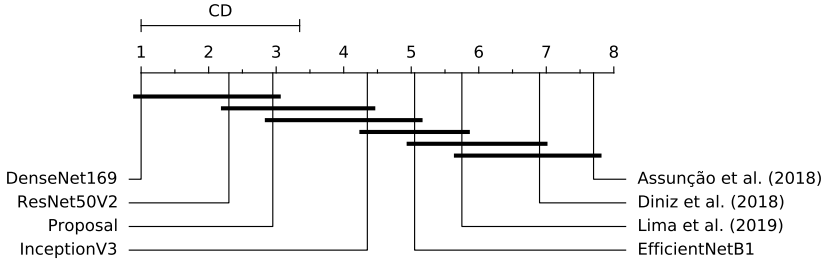
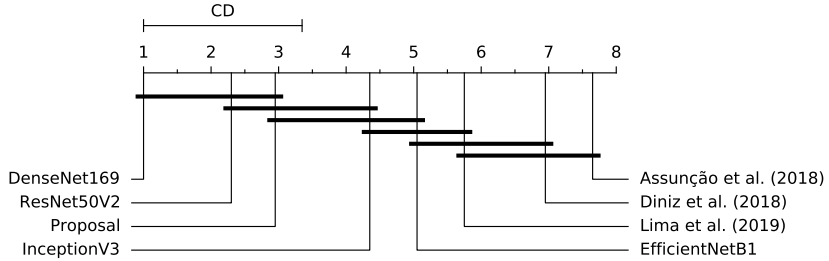


Dataset: octmnist, Metric: accuracy, CD: 2.348



Dataset: octmnist, Metric: f1_score, CD: 2.348



Acurácia	F_1*-score*
Dataset: organmnist_axial, Metric: accuracy, CD: 2.348	Dataset: organmnist_axial, Metric: f1_score, CD: 2.348
	

Assim como foi mencionado anteriormente, agora embasado pelo teste estatístico, os modelos gerados pela técnica proposta tomam a liderança na classificação nos conjuntos de dados CIFAR-10, MNIST e PathMNIST. Quanto ao conjunto de dados OCTMNIST, o modelo da proposta apresenta-se praticamente empatado com a rede DenseNet169. Já no conjunto de dados OrganMNIST Axial, o modelo da proposta não liderou a classificação, mas alcançou similaridade estatística às DenseNet169 e ResNet50V2. No entanto, apenas no conjunto de dados KMNIST o modelo da proposta não obteve performance suficiente para alcançar similaridade estatística às redes estado-da-arte.

6.3 Discussão dos resultados

No processo adotado, foram escolhidas diversas propostas de arquiteturas estado-da-arte, assim como, outras arquiteturas também geradas pelo processo de evolução gramatical, assim como a técnica proposta. No entanto, a maioria dos modelos produzidos pela proposta alcançaram a melhor performance, mesmo tendo uma arquitetura mais simples e consequentemente um menor número de parâmetros treináveis e tempo de processamento.

Outro ponto destacado, é o fato de mesmo não tendo uma função objetivo que busque explicitamente a minimização do número de parâmetros treináveis e tempo de treinamento das redes, o framework foi capaz de naturalmente guiar o processo evolutivo para a criação de arquiteturas mais simples. Consequentemente, as redes obtidas apresentam uma menor quantidade de parâmetros treináveis e tempo de processamento em relação as suas competidoras, destacando assim a eficiência do framework proposto.

Dentre as técnicas que também utilizam a evolução gramatical, os modelos da proposta alcançaram performance superior a estes, mesmo tempo um espaço de busca menor em comparação a Assunção et al. (2018) e Lima et al. (2019), mostrando que a gramática associada ao processo consegue estruturar indivíduos mais efetivos e consequentemente consumindo menos gerações no processo evolucionário.

Pelo fato do framework conseguir gerar modelos simples, porém efetivos, e que não dependem de um especialista da área para sua confecção, vê-se que a proposta torna-se uma técnica interessante quanto à elaboração de CNNs otimizadas para a resolução de problemas de classificação. Aliado a isso, nota-se que os modelos gerados podem ser utilizados em contextos onde o baixo uso de memória e processamento são requisitos

essenciais para a execução de tarefas.

7 Conclusão

7.1 Considerações e Contribuições do Trabalho

Neste trabalho, é proposto um framework de evolução gramatical voltado a otimização de redes neurais convolucionais focadas na resolução de problemas de classificação, guiadas através de duas funções objetivos: acurácia e F_1-score. O framework proposto evoluiu as redes otimizadas em seis conjuntos de dados: CIFAR-10, MNIST, KMNIST, PathMNIST, OCTMNIST e OrganMNIST Axial. Os modelos otimizados foram comparados a quatro arquiteturas estado-da-arte da literatura e outras três abordagem que também utilizam o processo de evolução gramatical.

Nossos modelos otimizados alcançaram a primeira classificação em três conjuntos de dados: CIFAR-10, MNIST e PathMNIST. Além disso, os resultados mostram que os modelos gerados alcançaram similaridade estatística à arquiteturas estado-da-arte nos conjuntos de dados OCTMNIST e OrganMNIST Axial. No entanto, nossos modelos apresentam um menor número camadas e consequentemente menor número de parâmetros treináveis comparados às arquiteturas estado-da-arte que se igualam estatisticamente.

Assim, os resultados mostram que os modelos produzidos são mais simples, menos complexos, consomem um menor poder computacional e ainda assim entregam resultados satisfatórios, podendo ser utilizados em ambientes onde a economia de memória e processamento seja requisitos essenciais.

Complementando, a técnica proposta pode ser utilizada em contextos onde não se tenha a habilidade e conhecimento de uma especialista na área de aprendizagem de máquina, para que sejam realizados ajustes e otimizações em redes como as estado-da-arte, mas que ainda assim possa se obter arquiteturas competitivas e sem a necessidade de conhecimento especializado.

Os resultados iniciais desta pesquisa foram publicados e encontram disponíveis nos anais do *2021 IEEE Congress on Evolutionary Computation (CEC)* com o título de *A Multi-Objective Grammatical Evolution Framework to Generate Convolutional Neural Network Architectures* (SILVA et al., 2021b) e no *SIBGRAPI 2021 - 34th Conference on Graphics, Patterns and Images* com o título *A New Grammar for Creating Convolutional Neural Networks Applied to Medical Image Classification* (SILVA et al., 2021a).

7.2 Trabalhos Futuros

Durante o processo de elaboração deste trabalho, foram observadas possíveis melhorias que podem auxiliar na performance e estrutura das arquiteturas otimizadas. Dentre elas, a adição de camadas de adição, soma e concatenação na estrutura da gramática e a utilização de técnicas conhecidas na literatura como blocos residuais (*ResBlocks*). Além disso, há a possibilidade da adição de outras camadas de *pooling* como a *average pooling* bem como outras técnicas de regularização como L1 e L2.

Quanto à navegação do espaço de busca, métricas como o coeficiente de Kappa Cohen podem ser utilizadas para avaliar a concordância entre os valores previstos e verdadeiros, dando mais subsídios à navegação de uma possível maior gramática. Além disso, há a possibilidade de testar o impacto do uso de diferentes algoritmos multi-objetivos na convergência dos modelos.

Referências

AHMADIZAR, F.; SOLTANIAN, K.; AKHLAGHIANTAB, F.; TSOULOS, I. Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 39, p. 1–13, 2015.

ASHLOCK, D. **Evolutionary computation for modeling and optimization**. [S.l.]: Springer Science & Business Media, 2006.

ASSUNÇÃO, F.; LOURENÇO, N.; MACHADO, P.; RIBEIRO, B. Evolving the topology of large scale deep neural networks. In: SPRINGER. **European Conference on Genetic Programming**. [S.l.], 2018. p. 19–34.

ASSUNçãO, F.; LOURENçO, N.; RIBEIRO, B.; MACHADO, P. Fast-denser: Fast deep evolutionary network structured representation. **SoftwareX**, v. 14, p. 100694, 2021. ISSN 2352-7110. Disponível em: <https://www.sciencedirect.com/science/article/pii/S235271102100039X>.

BACKUS, J. W.; BAUER, F. L.; GREEN, J.; KATZ, C.; MCCARTHY, J.; NAUR, P.; PERLIS, A. J.; RUTISHAUSER, H.; SAMELSON, K.; VAUQUOIS, B. et al. Revised report on the algorithmic language algol 60. **The Computer Journal**, Oxford University Press, v. 5, n. 4, p. 349–367, 1963.

BALDOMINOS, A.; SAEZ, Y.; ISASI, P. Evolutionary convolutional neural networks: An application to handwriting recognition. **Neurocomputing**, v. 283, p. 38–52, 2018. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231217319112>.

BARRICELLI, N. A. et al. Esempi numerici di processi di evoluzione. **Methodos**, v. 6, n. 21-22, p. 45–68, 1954.

BENGIO, Y. **Learning deep architectures for AI**. [S.l.]: Now Publishers Inc, 2009.

CHOLLET, F. et al. **Keras**. 2015. <https://keras.io/>.

CLANUWAT, T.; BOBER-IRIZAR, M.; KITAMOTO, A.; LAMB, A.; YAMAMOTO, K.; HA, D. Deep learning for classical japanese literature. 2018.

DAHL, G. E.; SAINATH, T. N.; HINTON, G. E. Improving deep neural networks for lvcsr using rectified linear units and dropout. In: IEEE. **2013 IEEE international conference on acoustics, speech and signal processing**. [S.l.], 2013. p. 8609–8613.

DARWIN, C.; MURRAY, J.; CLOWES, W.; SONS; EVANS, B. . **On the Origin of Species by Means of Natural Selection, Or, The Preservation of Favoured Races in the Struggle for Life**. J. Murray, 1859. (The World's Classics). Disponível em: <https://books.google.com.br/books?id=jTZbAAAAQAAJ>.

DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. **IEEE transactions on evolutionary computation**, IEEE, v. 6, n. 2, p. 182–197, 2002.

DINIZ, J. B.; CORDEIRO, F. R.; MIRANDA, P. B.; SILVA, L. A. T. da. A grammar-based genetic programming approach to optimize convolutional neural network architectures. In: SBC. **Anais do XV Encontro Nacional de Inteligência Artificial e Computacional**. [S.l.], 2018. p. 82–93.

DORIN, A.; TAYLOR, T. Rise of the self-replicators: Early visions of machines, ai and robots that can reproduce and evolve. Springer, 2020.

FENTON, M.; MCDERMOTT, J.; FAGAN, D.; FORSTENLECHNER, S.; HEMBERG, E.; O'NEILL, M. Ponyge2: Grammatical evolution in python. In: ACM. **Proceedings of the Genetic and Evolutionary Computation Conference Companion**. [S.l.], 2017. p. 1194–1201.

FOLEY, J. D.; DAM, A. V. **Fundamentals of interactive computer graphics**. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1982.

FRANKLE, J.; CARBIN, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. **arXiv preprint arXiv:1803.03635**, 2018.

FRASER, A. Monte carlo analyses of genetic models. **Nature**, Nature Publishing Group, v. 181, n. 4603, p. 208–209, 1958.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: **Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics**. [S.l.: s.n.], 2011. v. 15, p. 315–323.

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

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A.; BENGIO, Y. **Deep learning**. [S.l.]: MIT press Cambridge, 2016. v. 1.

GRUAU, F. On using syntactic constraints with genetic programming. In: ANGELINE, P. J.; Kinnear, Jr., K. E. (Ed.). **Advances in Genetic Programming 2**. Cambridge, MA, USA: MIT Press, 1996. cap. 19, p. 377–394. ISBN 0-262-01158-1.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. 2nd. ed. USA: Prentice Hall PTR, 1998. ISBN 0132733501.

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

_____. Identity mappings in deep residual networks. In: SPRINGER. **European conference on computer vision**. [S.l.], 2016. p. 630–645.

HUANG, G.; LIU, Z.; MAATEN, L. V. D.; WEINBERGER, K. Q. Densely connected convolutional networks. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 4700–4708.

KINGMA, D. P.; BA, J. **Adam: A Method for Stochastic Optimization**. 2017.

KOVÁCS, Z. L. **Redes neurais artificiais**. [S.l.]: Editora Livraria da Fisica, 2002.

KOZA, J.; KOZA, J.; RICE, J. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Bradford, 1992. (A Bradford book). ISBN 9780262111706. Disponível em: <https://books.google.com.br/books?id=Bhtxo60BV0EC>.

KOZA, J. R. **Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems**. [S.l.]: Stanford University, Department of Computer Science Stanford, CA, 1990. v. 34.

KRIZHEVSKY, A.; HINTON, G. **Learning multiple layers of features from tiny images**. Toronto, Ontario, 2009.

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

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, Ieee, v. 86, n. 11, p. 2278–2324, 1998.

LECUN, Y.; CORTES, C.; BURGES, C. Mnist handwritten digit database. **ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist**, v. 2, 2010.

LIMA, R. H. R. de; POZO, A.; SANTANA, R. Automatic design of convolutional neural networks using grammatical evolution. In: IEEE. **2019 8th Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.], 2019. p. 329–334.

LIU, L.; OUYANG, W.; WANG, X.; FIEGUTH, P.; CHEN, J.; LIU, X.; PIETIKÄINEN, M. Deep learning for generic object detection: A survey. **International journal of computer vision**, Springer, v. 128, n. 2, p. 261–318, 2020.

LORENA, A. C.; GAMA, J.; FACELI, K. **Inteligência Artificial: Uma abordagem de aprendizado de máquina**. [S.l.]: Grupo Gen-LTC, 2000.

MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. **Machine learning: An artificial intelligence approach**. [S.l.]: Springer Science & Business Media, 2013.

MIIKKULAINEN, R.; LIANG, J.; MEYERSON, E.; RAWAL, A.; FINK, D.; FRANCON, O.; RAJU, B.; SHAHRZAD, H.; NAVRUZYAN, A.; DUFFY, N. et al. Evolving deep neural networks. In: **Artificial Intelligence in the Age of Neural Networks and Brain Computing**. [S.l.]: Elsevier, 2019. p. 293–312.

MUGGLETON, S. Inductive logic programming: derivations, successes and shortcomings. **ACM SIGART Bulletin**, ACM New York, NY, USA, v. 5, n. 1, p. 5–11, 1994.

NEMENYI, P. B. **Distribution-free multiple comparisons.** [S.l.]: Princeton University, 1963.

NETO, G.; MIRANDA, P. B.; CAVALCANTI, G. D.; SI, T.; CORDEIRO, F.; CASTRO, M. Layers sequence optimizing for deep neural networks using multiples objectives. In: IEEE. **2020 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.], 2020. p. 1–8.

NWANKPA, C.; IJOMAH, W.; GACHAGAN, A.; MARSHALL, S. Activation functions: Comparison of trends in practice and research for deep learning. **arXiv preprint arXiv:1811.03378**, 2018.

O'NEILL, M.; RYAN, C. Grammatical evolution. **IEEE Transactions on Evolutionary Computation**, IEEE, v. 5, n. 4, p. 349–358, 2001.

O'NEILL, M.; RYAN, C. Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In: SPRINGER. **European Conference on Genetic Programming**. [S.l.], 2004. p. 138–149.

PAPPA, G.; FREITAS, A. Evolving rule induction algorithms with multi-objective grammar-based genetic programming. **Knowl. Inf. Syst.**, v. 19, p. 283–309, 06 2009.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V. et al. Scikit-learn: Machine learning in python. **Journal of machine learning research**, v. 12, n. Oct, p. 2825–2830, 2011.

PEREIRA, D. G.; AFONSO, A.; MEDEIROS, F. M. Overview of friedman's test and post-hoc analysis. **Communications in Statistics-Simulation and Computation**, Taylor & Francis, v. 44, n. 10, p. 2636–2653, 2015.

RAHAMAN, M.; MAHIN, M.; ALI, M.; HASANUZZAMAN. Bhcdr: Real-time bangla handwritten characters and digits recognition using adopted convolutional neural network. 04 2019.

RODRIGUES, E. L. M. **Evolução de funções em programação genética orientada a gramáticas**. [S.l.]: Universidade Federal do Paraná, 2002.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3. ed. [S.l.]: Prentice Hall, 2010.

RYAN, C.; COLLINS, J. J.; NEILL, M. O. Grammatical evolution: Evolving programs for an arbitrary language. In: SPRINGER. **European Conference on Genetic Programming**. [S.l.], 1998. p. 83–96.

SILVA, C. A. da; MIRANDA, P. B.; CORDEIRO, F. R. A new grammar for creating convolutional neural networks applied to medical image classification. In: **SIBGRAPI 2021 - 34th Conference on Graphics, Patterns and Images**. [S.l.: s.n.], 2021.

SILVA, C. A. da; ROSA, D. C.; MIRANDA, P. B.; CORDEIRO, F. R.; SI, T.; NASCIMENTO, A. C.; MELLO, R. F.; NETO, P. S. de M. A multi-objective grammatical evolution framework to generate convolutional neural network architectures. In: IEEE. **2021 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.], 2021. p. 2187–2194.

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

SOLTANIAN, K.; TAB, F. A.; ZAR, F. A.; TSOULOS, I. Artificial neural networks generation using grammatical evolution. In: IEEE. **2013 21st Iranian Conference on Electrical Engineering (ICEE)**. [S.l.], 2013. p. 1–5.

SUGANUMA, M.; SHIRAKAWA, S.; NAGAO, T. A genetic programming approach to designing convolutional neural network architectures. In: **Proceedings of the Genetic and Evolutionary Computation Conference**. [S.l.: s.n.], 2017. p. 497–504.

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

TAN, M.; LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2019. p. 6105–6114.

THARWAT, A. Classification assessment methods. **Applied Computing and Informatics**, v. 17, n. 1, p. 168–192, 2021.

TSOULOS, I.; GAVRILIS, D.; GLAVAS, E. Neural network construction and training using grammatical evolution. **Neurocomputing**, Elsevier, v. 72, n. 1-3, p. 269–277, 2008.

WHIGHAM, P. A. et al. Grammatically-based genetic programming. In: CITESEER. **Proceedings of the workshop on genetic programming: from theory to real-world applications**. [S.l.], 1995. v. 16, n. 3, p. 33–41.

YANG, J.; SHI, R.; NI, B. **MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis**. 2021.

ZHANG, Z.; SABUNCU, M. R. **Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels**. 2018.