UNICAMP

UNIVERSIDADE ESTADUAL DE CAMPINAS

Instituto de Matemática, Estatística e Computação Científica

VICTOR ALEXANDRE GOMES WEIL

Busca de arquiteturas de redes neurais morfológicas

Campinas

2022

Victor Alexandre Gomes Weil

Busca de arquiteturas de redes neurais morfológicas

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Matemática Aplicada.

Orientador: João Batista Florindo

Este trabalho corresponde à versão final da Dissertação defendida pelo aluno Victor Alexandre Gomes Weil e orientada pelo Prof. Dr. João Batista Florindo.

Campinas

2022

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Weil, Victor Alexandre Gomes, 1998-
W429b Busca de arquiteturas de redes neurais morfológicas / Victor Alexandre Gomes Weil. – Campinas, SP : [s.n.], 2022.

Orientador: João Batista Florindo.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Redes neurais (Computação). 2. Morfologia matemática. 3. Classificação de imagem. 4. Busca de arquitetura neural (Aprendizado de máquina). I. Florindo, João Batista, 1984-. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Search for morphological neural network architectures
Palavras-chave em inglês:
Neural networks (Computer science)
Mathematical morphology
Image classification
Neural architecture search (Machine learning)
Área de concentração: Matemática Aplicada
Titulação: Mestre em Matemática Aplicada
Banca examinadora:
João Batista Florindo [Orientador]
Marcos Eduardo Ribeiro do Valle Mesquita
Romis Ribeiro de Faissol Attux
Data de defesa: 14-03-2022
Programa de Pós-Graduação: Matemática Aplicada

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0003-0321-359X
- Currículo Lattes do autor: http://lattes.cnpq.br/2610077926496373

Dissertação de Mestrado defendida em 14 de março de 2022 e aprovada pela banca examinadora composta pelos Profs. Drs.

Prof(a). Dr(a). JOÃO BATISTA FLORINDO

Prof(a). Dr(a). MARCOS EDUARDO RIBEIRO DO VALLE MESQUITA

Prof(a). Dr(a). ROMIS RIBEIRO DE FAISSOL ATTUX

A Ata da Defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria de Pós-Graduação do Instituto de Matemática, Estatística e Computação Científica.

Agradecimentos

- Ao meu orientador, João Batista Florindo, pelo grande auxílio durante o desenvolvimento desse trabalho;
- À toda minha familia, pelo apoio psicológico durante a escrita desse trabalho;
- À minha namorada, Natalia, pelo apoio e ajuda em dúvidas relacionadas a temas gerais de redes neurais;
- Ao professor Sandro Bitar, pelas grandes dicas e conselhos dados durante minha graduação;
- O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil. (CAPES) - Código de Financiamento 001

Resumo

Esse trabalho utiliza técnicas de *Neural Architecture Search* (NAS) com o objetivo de otimizar hiperparâmetros em diferentes arquiteturas que, em suas conexões, possuem operações morfológicas diferenciáveis e atuam em problemas de classificação diversos. Em particular, uma base de texturas é usada para validação do método. Por serem operações diferenciáveis, essas arquiteturas podem ser treinadas por *backpropagation*. Os resultados obtidos mostram que o uso combinado de operações morfológicas e clássicas aumentam a capacidade de generalização da rede, dependendo do problema. Além disso, o uso de técnicas de NAS possibilita a construção de uma arquitetura com desempenho superior se comparado ao de uma construída manualmente.

Palavras-chave: Redes neurais. Morfologia matemática. Classificação de imagens. Busca de arquiteturas de redes neurais.

Abstract

This work proposes the use of *Neural Architecture Search* (NAS) techniques with the objective of optimizing hyperparameters in different architectures possessing, in their connections, differentiable morphological operations. The methodology is applied to different classification problems. In particular, the classification of a texture database is performed. Because these are operations are differentiable, these architectures can be trained by *backpropagation*. In our analysis, the results show that the combined use of morphological and classical operations increase the generalizability of the network. In addition, the use of NAS techniques makes it possible to build an architecture with superior performance compared to one empirically constructed.

Keywords: Neural networks. Mathematical morphology. Image classification. Neural architecture search.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

NAS	*Neural Architecture Search*
CNN	*Convolutional Neural Network*
DNN	*Deep Neural Network*
SMCNN	*Smooth Morphological Convolutional Neural Network*
ANN	*Artificial Neural Network*
RNN	*Recurrent Neural Network*

Sumário

1 Introdução

O aprendizado de máquinas é uma ramificação da inteligência artificial focada na construção de algoritmos que executam tarefas complexas e melhoram seu desempenho ao longo do tempo através de algoritmos auxiliares de otimização sem a necessidade de constante interferência humana. Assim, esse campo da ciência ganhou destaque nas mais diversas tarefas envolvendo processamento e análise de dados como, por exemplo, reconhecimento de voz [1, 2] ou imagem [3, 4].

Contudo, as técnicas clássicas para aprendizado de máquinas são limitadas em sua habilidade de processar os dados de um conjunto na sua forma bruta. Por muito tempo, construir um sistema para reconhecimento de padrões precisou de manipulações cuidadosas e domínio na área para desenvolver um extrator de características que transformava os dados brutos em informação útil para uma posterior análise dos padrões e execução de alguma tarefa [5].

Tendo em vista essa problemática, a aprendizagem de características é um conjunto de métodos que permitem a uma máquina ser alimentada com dados brutos e automaticamente descobrir as características necessárias para classificar ou detectar um objeto. Similarmente, métodos de aprendizagem profunda possuem múltiplos níveis de representação, obtidos pela composição de módulos simples e não lineares que extraem, a cada nível, características cada vez mais complexas. Em quantidade suficiente, os níveis de representação possibilitam que os algoritmos aprendam funções cada vez mais complexas [5].

Com o aumento da capacidade de processamento dos computadores nas últimas décadas, foi possível a utlização de um sistema de computação chamado de redes neurais e, com esse sistema, surgiram aplicações da aprendizagem profunda. Esse tipo de algoritmo se mostrou superior a outros por ter uma alta capacidade de generalização e extração de características peculiares dos dados, consequentemente, possuindo um melhor desempenho em diversas aplicações.

Também, com o objetivo de melhorar a eficiência e possibilitar que diversos tipos de usuários pudessem aplicar esses algoritmos em seus problemas, a automatização das técnicas em aprendizado de máquinas tornou-se um campo de pesquisa popular nos últimos tempos. Em particular, as técnicas de *Neural Architecture Search* (NAS) buscam automatizar o processo da busca e otimização dos hiperparâmetros das arquiteturas de redes neurais.

Embora, em princípio, o objetivo de algoritmos de NAS seja encontrar arquiteturas ótimas em um espaço universal de redes, na prática esse espaço não costuma

englobar redes em que as operações internas não se limitam apenas àquelas das redes neurais clássicas. Um exemplo desse tipo de modelo alternativo que não é coberto por técnicas populares de NAS são as redes morfológicas.

Partindo das ideias de redes neurais e morfologia matemática e tendo como objetivo se assemelhar ao modelo biológico dos neurônios humanos, as redes morfológicas buscam modelar as computações feitas em um neurônio artificial se inspirando nas operações lógicas presentes no neurônio humano, no qual boa parte de suas sinapses ocorrem nos dendritos, que é onde a informação é processada [6]. Como consequência, essas redes se utilizam das operações presentes na morfologia matemática que, por sua vez, são aplicadas, por exemplo, em problemas de detecção de bordas, remoção de ruídos [7] e segmentação de imagem.

1.1 Objetivos

O objetivo geral deste trabalho é avaliar o uso de técnicas de NAS em conjunto com operações morfológicas. As redes morfológicas já apresentam contribuições em algumas áreas, como mencionado anteriormente. As técnicas de NAS surgiram recentemente [8] e podem melhorar a capacidade de generalização de uma rede neural em um conjunto de dados. O objetivo principal do estudo é o seguinte:

- Uso da operação *SMorph* [9] na classificação de dados;
- Aplicação de técnicas de NAS para construção de arquiteturas com operadores morfológicos;
- Avaliar as fronteiras de decisão geradas por arquiteturas que contenham operações morfológicas em camadas densas;
- Aplicação de operações de dilatação e erosão em imagens;
- Avaliação da rede morfológica em uma base de dados de textura;

Para a realização dos testes utilizamos diversos *datasets* diferentes, sendo dois usados para *benchmark* da *SMorph*: MNIST e GTSRB. Também foi utilizado um *dataset* mais desafiador, a FMD, que é uma base de texturas.

1.2 Contribuições

Neste trabalho apresentamos aplicações de NAS em redes morfológicas, algo relativamente pouco estudado na literatura. Fazemos o uso de operações morfológicas diferenciáveis que, portanto, podem ser treinadas com *backpropagation*. Aplicamos a

operação *SMorph* em tarefas para classificação de imagens e mostramos os efeitos da dilatação e erosão nos *datasets* utilizados, algo que até o momento ainda não havia sido avaliado. Com o uso de NAS em conjunto com essa operação, obtemos acurácias competitivas com o estado da arte.

Além disso, com o auxílio dessa operação morfológica, propomos duas operações morfológicas em camadas densas que são diferenciáveis e as usamos em diversos problemas de classificação que superam outros algoritmos nesta tarefa. Por fim, também usamos essas operações morfológicas propostas como classificadores em bases de texturas e obtemos resultados competitivos com o estado da arte.

1.3 Organização

Esta dissertação está divida em 8 capítulos, incluindo a introdução. O capítulo seguinte apresenta referências de trabalhos relevantes de cada um dos principais temas deste estudo, que são: redes neurais, morfologia matemática e NAS. No terceito capítulo fazemos uma fundamentação desses mesmos temas, apresentando conceitos relevantes para o entendimento dos resultados. No quarto capítulo apresentamos a metodologia proposta por este trabalho. No penúltimo capítulo apresentamos informações sobre os *datasets* utilizados, como cada experimento será configurado para a produção dos resultados e, por fim, apresentamos os resultados. No último capítulo apresentamos a conclusão do trabalho bem como sugestões para trabalhos futuros abordando este mesmo tema.

2 Revisão Bibliográfica

2.1 Redes Neurais

Algoritmos de redes neurais não foram propostos recentemente. Já em 1943, os autores de [10] apresentaram um modelo lógico que buscava se comportar de forma semelhante aos neurônios presentes no cérebro humano e construíram a teoria necessária para fundamentar aquele modelo que, até o momento, ainda não era treinável.

Citando trabalhos como o de *McCulloch* e *Pitts* [10], *Rosenblatt* [11] [12] apresentou os chamados *perceptrons*, um modelo projetado com o intuito de ilustrar algumas das propriedades fundamentais presentes em sistemas lógicos de organismos em geral e esse modelo já era treinável através do chamado *supervised learning*. Modelos com ideias similares e treináveis através de *supervised learning* também são apresentados em [13] e [14].

De acordo com Schmidhuber [15], o uso de gradiente descendente para problemas de otimização em espaços de parâmetros complexos, não lineares, diferenciáveis e relacionados a redes neurais era discutido desde meados de 1960 em trabalhos como, por exemplo, o de *Kelley* [16], *Bryson* e *Denham* [17] e *Amari* [18]. Os sistemas dessa época retropropagavam a informação de um camada a outra através da matriz Jacobiana usual, porém não faziam correlações explicítas entre as múltiplas camadas em seu uso nem exploravam potenciais ganhos devido à esparsidade das redes.

Sem referências explicítas a redes neurais, o uso do chamado *efficient error backpropagation* em sistemas similares aos de redes neurais foi descrito em [19] e na dissertação de mestrado do mesmo autor. Em 1980, *Speelpenning* [20] implementou um algoritmo de *backpropagation* para sistemas diferenciáveis.

Um trabalho que contribuiu muito para a popularização do uso de *backpropagation* em redes neurais foi o de *Rumelhart*, *Hinton* e *Williams* [21], que fala sobre a necessidade de representações utéis em camadas ocultas de redes neurais, já que redes sem camadas ocultas não possuiam a habilidade de solucionar alguns problemas como o clássico XOR.

Em 1989, Cybenko [22] provou o *Teorema da Aproximação Universal* para redes neurais compostas por uma camada oculta e uma quantidade arbitrária de neurônios possuindo a função sigmoide como ativação. De acordo com o autor, uma função de decisão qualquer pode ser arbitrariamente aproximada por uma rede neural.

Tornando-se posteriormente muito popular no reconhecimento de padrões,

LeCun *et al.* [23] apresentou um novo tipo de rede neural que podia ser treinada através de *backpropagation*. Chamada de rede convolucional, esse tipo de rede inicialmente foi utilizada para a classificação de dígitos com o objetivo de minimizar a quantidade de parâmetros que precisam ser determinados e, consequentemente, reduzindo o custo computacional, um grande problema já naquela época.

Trabalhos como [24] apresentam exemplos de redes convolucionais para o reconhecimento de imagens. Entretanto, na época, esse tipo de abordagem não se popularizou em conjuntos de dados grandes devido à falta de poder computacional disponível e à necessidade de conjuntos não apenas grandes, mas também bem categorizados.

A partir de 2010, [4] e [2] mostraram o potencial das redes neurais profundas, *deep neural networks* (DNNs), para a classificação de imagens e voz. A *AlexNet* [4] foi a primeira *Convolutional Neural Network* (CNN) em larga escala que superou os algoritmos até então disponíveis no conjunto de dados denominado *ImageNet* [25] e proporcionou novas pesquisas a respeito das arquiteturas de redes convolucionais.

Criada por *Zeiler* e *Fergus* [26], a *ZFNet* possui uma estrutura similar à presente na *AlexNet*, porém com hiperparâmetros aperfeiçoados e, assim, obteve uma precisão melhor, mostrando que ajustes nos hiperparâmetros de uma rede neural possibilitam uma melhora na performance. Outra contribuição importante desse trabalho foi a ideia de deconvolução, que procura mapear as características aprendidas pela rede de volta para os pixels de entrada, possibilitando uma visualização interna da rede. A partir disso, foi constatado que as características em um *feature map* da rede neural estão longe de serem aleatórias e com padrões não interpretáveis. Ainda com essa ideia, foi feito um estudo para avaliar qual a contribuição para o desempenho da rede de cada camada em um modelo e como usar a ideia de deconvolução para identificar problemas no modelo e obter resultados melhores.

Em 2014, a arquitetura chamada de *VGG*, em um trabalho com autoria de *Simonyan* e *Zisserman* [27], foi vice-campeã da competição no banco de dados *ImageNet*. Essa arquitetura é amplamente usada por ser mais simples em comparação com a *AlexNet* e *ZFNet*. A *VGG* utiliza filtros de tamanho 3×3 em comparação com o 11×11 da *AlexNet* e 7×7 da *ZFNet*. A estratégia do autor foi reduzir a quantidade de hiperparâmetros treináveis na rede neural.

Ainda no mesmo ano, e vencedora daquela competição, a *GoogLeNet* [28] propôs um novo modelo de arquitetura chamado de *inception*. A principal característica desse modelo é o aperfeiçoamento da utilização dos recursos computacionais dentro da rede.

Conforme a quantidade de camadas aumenta, as arquiteturas de redes profundas se tornam difíceis de treinar. Em 2015, a *ResNet* [29] propôs uma estrutura de aprendizagem residual com o objetivo de facilitar seu treinamento e mostrando que, quanto mais profunda

a arquitetura, mais é possível reduzir-se a taxa de erro, ainda que com um aumento do custo computacional.

Diversas novas arquiteturas são aperfeiçoamentos dos modelos inovadores já apresentados e algumas são a união das arquiteturas apresentadas nesses trabalhos. Essas novas arquiteturas possuem performances melhores, mas têm ideias parecidas com as que foram apresentadas anteriormente. A seguir, apresentaremos algumas pesquisas recentes em redes neurais, particularmente, focando nas que envolvem redes convolucionais.

Um exemplo relativamente recente, Howard *et al.* [30] é uma simplificação das redes neurais, inicialmente projetadas para facilitar o uso de seus recursos em um aparelho móvel. Chamada de MobileNet, esses modelos são baseados em arquiteturas simplificadas que usa convoluções separáveis em profundidade para construir redes neurais profundas leves.

O trabalho de *Tan* e *Le* [31] propõe um metodo de rede para dimensionamento das redes convolucionais. Em seu estudo foi notado que, balancear a largura, altura e resolução das imagens que passam na rede, é uma característica importante para melhorar a acurácia e eficiência da rede. Usando técnicas de NAS, eles ainda desenvolvem uma nova arquitetura base e chamam a família de modelos com essas técnicas de *EfficientNet*.

2.2 Redes Morfológicas

Os primeiros trabalhos que formularam redes neurais morfológicas surgiram por volta de 1990. Por um lado, Wilson [32] definiu os neurônios morfológicos como sendo filtros de classificação ponderada. Por outro, *Ritter* e *Davidson* [33, 34] definiram de uma forma mais específica, com a rede realizando operações elementares de dilatação ou erosão.

Davidson e *Ritter* [33] buscaram formalizar a teoria fundamental das redes morfológicas que, em vez da multiplicação e somatório, utilizam as operações de adição e máximo, sendo a operação de máximo responsável pela introdução de não-linearidade. Similarmente, *Davidson* e *Ritter* [34] procuraram relacionar a área da Álgebra de Imagens com as redes morfológicas e formalizaram algumas notações para essa rede, além de apresentar os algoritmos para treiná-las.

Em seguida, Ritter e Ucid [6] introduziram e desenvolveram um modelo matemático para a computação de dendritos em neurônios morfológicos baseado na Álgebra de Reticulados [35]. Anos depois, *Angulo* [36] estudou como generalizar abordagens baseadas na difusão buscando introduzir filtros não lineares em que os efeitos simulam a dilatação e erosão morfológica.

Inspirado por *Angulo* [36] e se baseando-se na ideia de *counter-harmonic mean*, Masci *et al.* [37] propuseram uma nova estrutura para aprendizagem de operadores morfo-

lógicos que, ao combinar o conceito de morfologia com o de redes neurais convolucionais, reproduz efeitos de dilatação, erosão, abertura e fechamento em uma imagem.

No trabalho de *Zamora* e *Sossa* [38] foi apresentado um método novo para treinamento de neurônios morfológicos em tarefas de classificação que se baseava em gradiente descendente estocástico, usando ideias adotadas por trabalhos anteriores apenas na inicialização dos pesos da rede morfológica. Além disso, é mostrado que as técnicas usadas anteriormente possuem certas desvantagens, como problemas de generalização em conjuntos não vistos pelo algoritmo.

Em 2019, Mellouli [39] propôs uma rede convolucional morfológica em que os papeis de seus *feature maps* podem ser mais interpretáveis em comparação ao de uma CNN para o reconhecimento de padrões. Chamada de *Morph-CNN*, essa rede usa as noções apresentadas em [37] para gerar melhores *feature maps* nas camadas convolucionas. A *Morph-CNN* apresentou acurácias superiores nas bases de *benchmark MNIST* e *SVHN*.

Pela não diferenciabilidade das operações morfológicas, substituir a convolução usual por uma erosão ou dilatação é um desafio. Portanto, com o crescimento da atenção nos últimos anos à integração de operações morfológicas com redes neurais profundas, Kirszenberg *et al.* [9] apresentaram duas novas operações morfológicas diferenciáveis que resolvem alguns problemas presentes na *Morph-CNN*. A primeira operação, chamada de *LMorph*, é definida com o objetivo de ser compatível com a teoria de morfologia matemática em escala de cinza. Já a segunda, *SMorph*, usa ideias da função *softmax* para reproduzir os resultados desejados além de resolver as limitações presentes na *Morph-CNN*.

Mondal *et al* [40] analisaram teoricamente o uso de operações morfológicas em vetores unidimensionais e mostraram que estas podem ser estendidas para vetores multidimensionais (matrizes). Além disso, também foi mostrado nesse trabalho que o denominado bloco morfológico pode representar uma soma de *hinge functions*, que são aplicáveis em diversas tarefas de classificação e regressão. Por fim, mostraram também que a combinação de blocos morfológicos satisfaz o teorema da aproximação universal.

2.3 Neural Architecture Search

O ajuste de hiperparâmetros se refere à otimização automática dos hiperparâmetros de um modelo de aprendizado de máquinas. Por sua vez restrita a redes neurais, NAS procura otimizar seus hiperparâmetros (taxa de aprendizado, camadas ocultas e números de neurônios, por exemplo) e, portanto, pode ser vista como um subcampo da otimização de hiperparâmetros.

Com a premissa de que modelar arquiteturas de redes neurais requer um estudo especializado, Zoph e *Le* [8] usaram aprendizado por reforço para projetar, na

época, arquiteturas para reconhecimento de imagens e linguagem. Assim, essa abordagem conseguia criar arquiteturas que se comparavam às melhores já criadas por humanos em termos de acurácia no conjunto de teste da base de dados *CIFAR-10*.

Também, publicado na mesma época e com a mesma premissa do trabalho anterior, Baker *et al.* [41] utilizaram técnicas de aprendizado por reforço para modelar arquiteturas focadas na ideia de redes neurais convolucionais competitivas para o reconhecimento de imagens. Portanto, [8] e [41] popularizaram e impulsionaram pesquisas em NAS.

Entretanto, [8], [41] e os primeiros trabalhos em NAS requeriam uma enorme quantidade de recurso computacional, tornando impraticável para diversos usuários utilizarem essa técnica. Abordagens mais recentes exploram formas de reduzir o custo computacional e novos métodos estão sendo introduzidos com frequência na comunidade científica.

Com o objetivo de reduzir o custo computacional de NAS, Pham *et al.* [42] propuseram uma abordagem rápida e eficiente para o design automático de modelos. Nesse trabalho, um controlador descobre redes neurais buscando por um subgrafo ótimo dentro de um grande grafo computacional.

Diversos trabalhos exploraram *weight sharing* nas arquiteturas, focando assim em reduzir o custo da busca. Estes fizeram notáveis progressos em *One-Shot* NAS. Entretanto, foi constatado em [43], [44] e [45] que arquiteturas com alta acurácia no modelo *one-shot* não necessariamente refletem a sua real performance quando treinadas sem o uso de *weight sharing*. Assim, Xia *et al.* [46] propuseram um novo método chamado de *Hierarchical Neural Architecture Search* (HNAS) que, diferente de abordagens anteriores, formula uma estratégia de busca hierárquica baseada na operação de *pruning* e constroi, para cada camada, um espaço de busca que contenha apenas operações relevantes.

Já Real *et al.* [47] adotaram a estratégia dos algoritmos evolutivos como uma forma de procurar pelas arquiteturas de alta performance que, pela primeira vez, superaram modelos cuja arquitetura não é definida por métodos de busca exaustiva. Denominada *AmoebaNet*, eles modificaram uma técnica de algoritmos evolutivos chamada de seleção por torneio de forma a favorecer arquiteturas mais novas. Essa abordagem utilizada por eles foi chamada de *aging evolution* ou *regularized evolution*.

Também adotando algoritmos evolutivos e utilizando a técnica sem modificações de seleção por torneio como sua *search strategy*, Liu *et al.* [48] combinaram um esquema de representação genético hierárquico que imita o padrão de modelagem adotado pelos especialistas com um *search space* expressivo que suporta topologias complexas.

Os autores de [49] propuseram um novo método para aprender a estrutura de redes neurais convolucionais mais eficiente do que os baseados em aprendizado por

reforço e algoritmos evolutivos. Esse método procura por estruturas em ordem crescente de complexidade, enquanto simultaneamente aprende um modelo substituto para guiar a busca no *search space*.

Nas abordagens convencionais de NAS, os algoritmos buscam arquiteturas em um *search space* discreto e não diferenciável. O trabalho apresentado em [50] propõe um relaxamento contínuo da representação da arquitetura, possibilitando uma busca eficiente das arquiteturas através de gradiente descendente.

Também, automatizando o processo de construção das arquiteturas e tendo como objetivo acelerar o processo de busca por arquiteturas, Zhang *et al.* [51] propôs um método que, ao reduzir o número de operações candidatas enquanto mantém os parâmetros da rede neural treinados durante o processo chamado de *pruning*, faz com que a computação necessária em cada estágio seja reduzida, acelerando assim o processo de procura por arquiteturas ótimas.

Nesse contexto, esse trabalho busca a utilização de métodos de NAS em conjunto com o uso de elementos da morfologia matemática em tarefas de classificação, algo não muito comum na área de NAS. Com o uso das operações morfológicas, resultados competitivos são obtidos em *datasets* clássicos.

3 Fundamentação Teórica

3.1 Redes Neurais

O objetivo desta seção será apresentar conceitos fundamentais para que se entenda o funcionamento de uma rede neural e para que se definam algumas das funções de ativação mais utilizadas e como uma arquitetura neural usualmente é composta.

A tarefa de reconhecimento de padrões pode se tornar complexa dependendo do conjunto de dados a ser trabalho. Escrever regras para um programa de computador descrever partes de uma imagem, por exemplo, não é trivial. Portanto, as redes neurais buscam abordar esse problema de uma forma diferente: a ideia é pegar um grande número de exemplos para treino e, a partir deles, desenvolver um sistema que possa aprender características relevantes. Além disso, ao aumentar a quantidade de exemplos para treino, uma rede neural pode melhorar ainda mais sua acurácia em tarefas de classificação já que, quanto maior a quantidade de exemplos, possivelmente haverá uma quantia maior de padrões para a rede aprender.

3.1.1 Perceptrons

Inspirando-se em [10], Rosenblatt [12] apresentou os chamados *perceptrons*. Um *perceptron* recebe várias entradas $(x_1, x_2, \ldots, x_N)$ e produz uma única saída binária que pode ser representada pela seguinte equação:

$$h(x) = \begin{cases} 0, & \text{se } \sum_i w_i x_i \leq b \\ 1, & \text{se } \sum_i w_i x_i > b \end{cases} \tag{3.1}$$

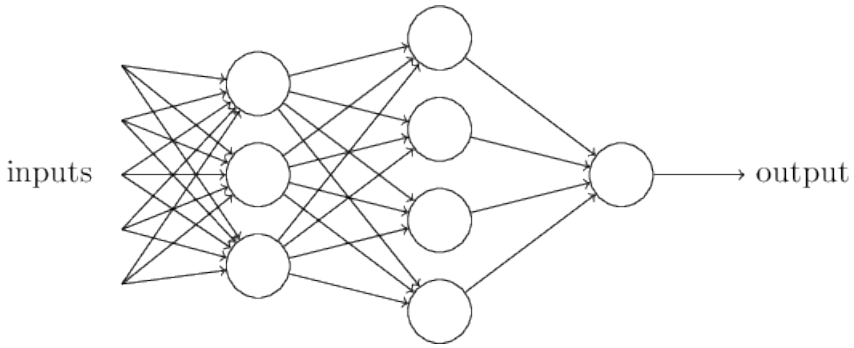
em que $x = (x_1, x_2, \ldots, x_N)$.

A partir dessa equação, um *perceptron* pode ser visto como um dispositivo que realiza suas decisões baseando-se nas combinações lineares do elemento x que contém as informações necessárias [52] com pesos w.

Por exemplo, suponha que uma decisão precise ser feita a partir de 3 informações binárias (1 ou 0) com mesmo peso e que, para essa decisão ser positiva (igual a 1), pelo menos duas dessas informações devam ser positivas. Então podemos representar esse problema como um vetor $x = (x_1, x_2, x_3)$, em que x_i representa uma informação i e w_i o seu peso. Como nesse exemplo todas as informações possuem o mesmo peso e duas delas precisam ser positivas, então $w = (1, 1, 1)$ e para o limiar temos $b = 1$. Portanto, se pelo menos 2 elementos do vetor x forem iguais a 1, teremos pela Equação (3.1) $h(x) = 1$.

Modelos com um único *perceptron* não podem ser considerados como um modelo que tem a capacidade de aproximar qualquer mapeamento de dados. Mas o exemplo anterior mostra que um *perceptron* pode ponderar diversas informações de modo a tomar uma decisão. Então, é plausível pensar que modelos mais complexos contendo vários *perceptrons* podem realizar decisões mais complexas.

Figura 1 – Pequena rede de *perceptrons*.



Fonte: Nielsen [53].

A Figura 1 apresenta um pequeno modelo contendo vários neurônios do tipo *perceptrons* representados pelos círculos. Após a camada de entrada, na primeira camada oculta de *perceptrons*, podemos considerar que seus neurônios estão recebendo as informações de entrada na rede e tomando suas decisões. Em seguida, na próxima camada, a rede está tomando suas decisões se baseando nas informações tomadas pela camada oculta anterior. Dessa forma, é razoável pensar que em redes com várias camadas e neurônios as decisões tomadas serão cada vez mais complexas. De fato, o *Teorema da Aproximação Universal* [54] [22] declara que as redes multicamadas de *perceptrons* utilizando funções de ativação sigmoidais (que seram apresentadas mais a frente) são uma classe de aproximadores universais caso possuam uma quantidade de neurônios suficientes para o problema.

3.1.2 Redes Neurais Artificiais

De acordo com Svozil *et al.* [55], redes neurais artificiais [56] (ou simplesmente redes neurais) são redes compostas por elementos de processamento (chamados de neurônios) que operam usando seus dados e se comunicando com outros elementos. A rede consiste em uma camada de entrada para os dados, camadas ocultas de neurônios (unidades) e uma camada final de saída dos neurônios.

De acordo com Wang [57], podemos definir uma *Fully-connected Neural Network* como uma rede em que cada neurônio j de uma camada $l-1$ se conecta a um neurônio i de uma camada l, esses neurônios são associados a um elemento numérico chamado de *peso* e outro de *viés* (*bias*), que define um limiar para uma função de ativação da rede. A

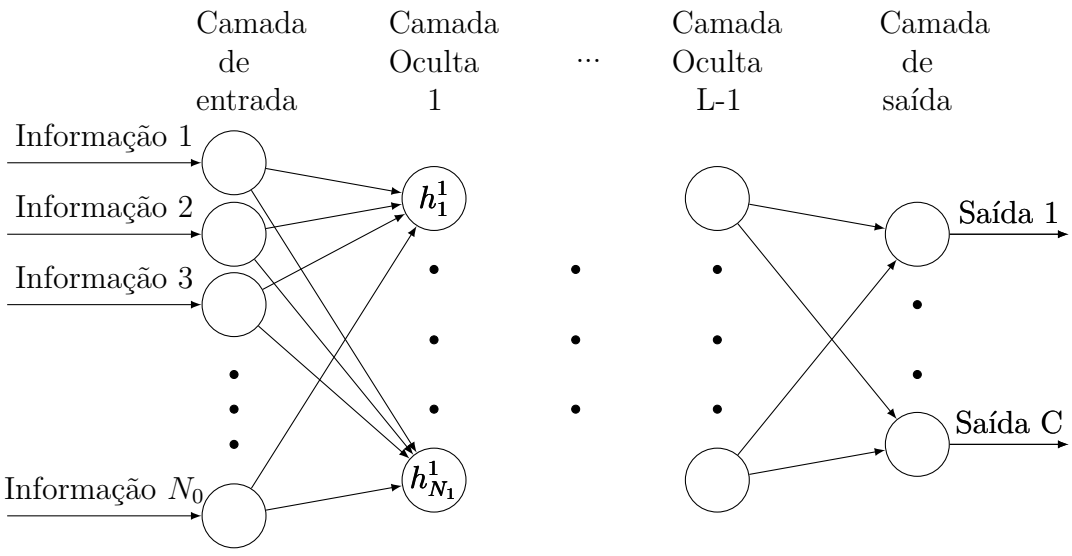
seguir explicamos essa associação entre diferentes camadas de uma rede *fully-connected*, na equação abaixo calculamos a saída h_i^l de um neurônio do tipo *perceptron*, isto é, o valor de saída do neurônio *perceptron* i na camada l como

$$h_i^l = \sigma^l \left(\sum_{j=1}^{N_{l-1}} w_{ij}^l h_j^{l-1} + b_i^l \right) \qquad (l = 1, \ldots, L), (i = 1, \ldots, N_l) \tag{3.2}$$

- h_j^{l-1}: valor de ativação do neurônio j na camada $l-1$;
- Quando $l = 0$, dizemos que h_i^0 é o vetor de entrada dos dados. Então para $l = 0$ será considerado $h_i^0 = x$;
- w_{ij}^l: elemento que liga o neurônio j na camada $l-1$ ao neurônio i na camada l;
- b_i^l: viés do neurônio i na camada l;
- σ^l: função de ativação na camada l;
- N_{l-1}: número de neurônios da camada l;

A Figura 2 abaixo exemplifica a arquitetura de uma *Fullyconnected Neural Network* com $L+1$ camadas que recebe um vetor contendo N informações e retorna como saída um vetor com C elementos que podem informar alguma característica ou as classificações preditas para um conjunto de dados.

Figura 2 – Arquitetura de $L+1$ camadas de uma *Fully-connected Neural Network*



Além disso, podemos reescrever (3.2) em uma expressão matricial e considerando múltiplos exemplos de entrada obtemos:

$$H^l = \sigma^l \left(W^l H^{l-1} + b^l \right), \tag{3.3}$$

com $W^l = [w_{ij}^l]_{i \times j}$ sendo a matriz de dimensão $N_l \times N_{l-1}$ que contém os pesos w_{ij}^l, b^l é um vetor de dimensão N_l que contém os limiares (víes) dos neurônios na camada oculta l e H^{l-1} é a matriz gerada por essa mesma equação, mas calculada na camada $l-1$ para $l = 2, \ldots, L$. Definimos H^0 como os dados de entrada da rede neural, por exemplo, um conjunto de dados quaisquer.

3.1.3 Redes Convolucionais

As *Convolutional Neural Networks* (CNNs) emergiram a partir do estudo do córtex cerebral [58] e são um tipo especializado de rede neural para o processamento de dados que possuem uma topologia no estilo de uma grade [56]. Exemplos de dados com essa topologia são: séries temporais, que podem ser vistas como grades unidimensionais se forem escolhidas amostras em intervalos definidos, e dados compostos por imagens, que formam grades bidimensionais de *pixels*. Esse tipo de rede trouxe resultados inovadores a diversas áreas relacionadas a reconhecimento de padrões como: processamento de imagens [59] e reconhecimento de voz [60].

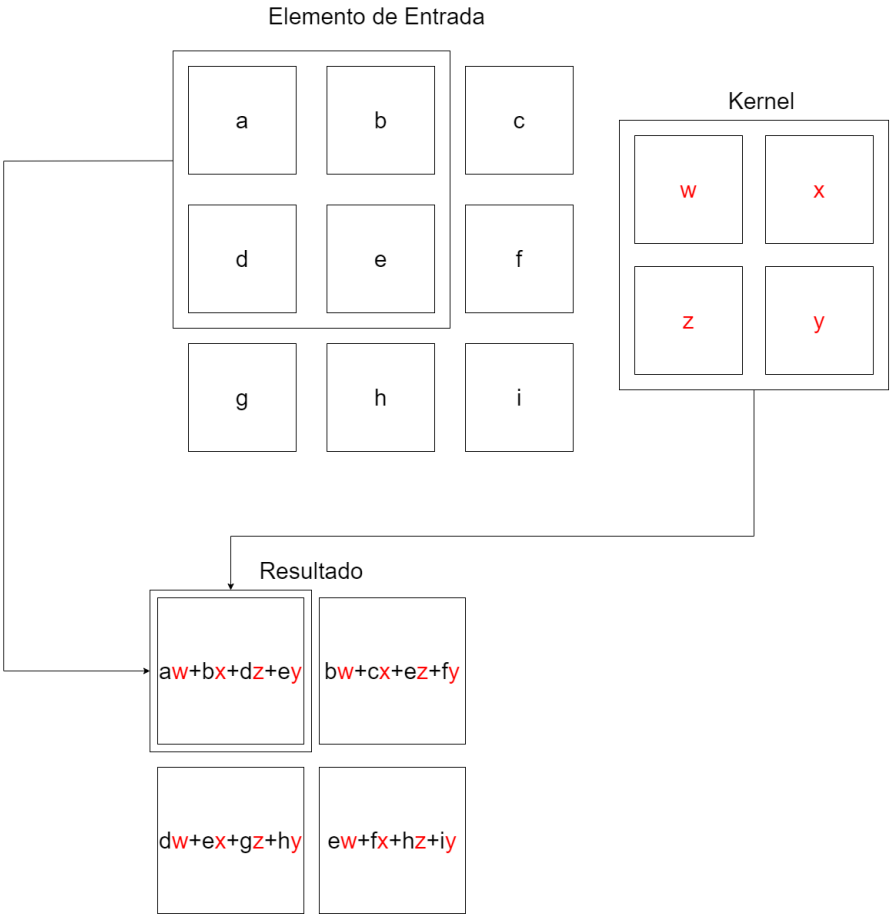
De acordo com Albawi *et al.* [61], definimos a operação realizada por uma convolução como segue

Definição 1 (Convolução 2D)**.** *Dada uma imagem F com dimensão $m \times n$, ela pode ser definida através de uma função $f : \mathbb{R}^2 \to \mathbb{R}$ onde $f(x, y)$ resulta no pixel da coordenada $[x, y]$ da imagem F. Dada uma coordenada $[k, l]$ da imagem F e um kernel w, uma convolução computa o seguinte*

$$(f * w)[k, l] := \sum_{i=1}^{m} \sum_{j=1}^{n} f[i, j] w[k - i, l - j]. \tag{3.4}$$

Apesar de a definição anterior ser baseada em elementos bidimensionais, convoluções podem ser definidas para diferentes dimensões. A Figura 3 exemplifica o cálculo realizado pela Equação (3.4) em um elemento de tamanho 3×3 com um kernel 2×2.

Figura 3 – Exemplo de uma convolução e sua operação computada.



Fonte: Adaptado de [56].

De acordo com Goodfellow *et al.* [56], as convoluções utilizam três ideias que podem ajudar a aperfeiçoar um sistema de aprendizado de máquinas, como o de redes neurais, são eles: *sparse weights*, *parameter sharing* e *equivariant representations*. As redes neurais tradicionais, com camadas *fully connected* de perceptrons, usam multiplicações de matrizes para descrever a interação entre os dados provenientes da camada anterior e os neurônios da camada atual. Isso significa que a quantidade de parâmetros em uma camada *fully connected* cresce baseado na dimensão dos dados de entrada e quantidade de neurônios na camada. As redes convolucionais, entretanto, possuem *sparse weights*. Por exemplo, uma imagem pode ter uma dimensão grande e, consequentemente, milhões de pixels com informações, entretanto, podemos detectar características relevantes, como bordas da imagem, ao utilizar um *kernel* que contenha apenas algumas dezenas de pixels ou até menos. Em outras palavras, uma camada convolucional pode necessitar de menos parâmetros que uma camada *fully connected*, algo que reduz a demanda de memória e melhora sua eficiencia [56].

Ainda na Figura 3, é evidenciada outra característica presente em uma camada convolucional, o *parameter sharing*. Enquanto numa camada *fully connected* de *perceptrons*

utilizamos cada parâmetro (peso) uma única vez para computarmos uma saída, nas camadas convolucionais cada parâmetro é utilizado múltiplas vezes durante a operação de convolução para percorrer o elemento de entrada na camada.

3.1.4 Funções de Ativação

Usualmente após as camadas de uma rede neural, tanto em uma rede totalmente conectada quanto em uma rede convolucional, são usadas funções de ativação para ser obtida a não linearidade necessária para o modelo. Abaixo definimos algumas das funções de ativação existentes e que podem ser utilizadas ao longo deste trabalho. De forma genérica, para as próximas definições considere $x \in \mathbb{R}$.

Definição 2 (Ativação ReLU)**.** *Uma das funções mais usadas em redes convolucionais, a Rectified Linear Unit é definida como segue:*

$$f(x) := max(0, x).$$

Definição 3 (Ativação LeakyReLU)**.** *Uma função derivada da ReLU, a Leaky Rectified Linear Unit é definida como segue:*

$$f(x) := \begin{cases} \alpha x & se\ x < 0 \\ x & se\ x \geq 0 \end{cases}$$

com $\alpha > 0$ onde usualmente é escolhido um valor pequeno menor que 1 (próximo de 0).

Definição 4 (Ativação Logística)**.** *Uma das funções de ativação mais usadas nas primeiras redes neurais, podemos definir a função logística da seguinte forma:*

$$\sigma(x) := \frac{1}{1 + e^{-x}}.$$

Definição 5 (Ativação Tanh)**.** *A função da tangente hiperbólica pode ser definida como segue:*

$$f(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

3.2 Redes Morfológicas

3.2.1 Morfologia Matemática

A morfologia matemática é uma metodologia não linear de processamento de imagens baseada na computação de filtros de máximos e mínimos em vizinhanças locais a partir de um elemento de estruturação [62]. No processamento de imagens em tons de cinza, podemos representar as imagens como uma função bidimensional $f : A \subset \mathbb{R}^2 \to \mathbb{R}$, em que cada $a \in A$ representa uma coordenada na imagem e $f(A) \subset \mathbb{R}$ representa o respectivo valor do seu pixel. Assim quando as imagens estão representadas em escala de cinza podemos definir as operações básicas da morfologia matemática como

Definição 6 (Dilatação)**.**

$$(f \oplus b)(a) := \sup_{y \in A}(f(y) + b(a - y)) \tag{3.5}$$

Definição 7 (Erosão)**.**

$$(f \ominus b)(a) := \inf_{y \in A}(f(y) - b(y - a)) \tag{3.6}$$

A composição dessas operações gera outras duas operações chamadas de *opening* e *closing*, definidas a seguir:

Definição 8 (*Opening*)**.** *Definimos como opening a composição das operações básicas feita da seguinte forma:*

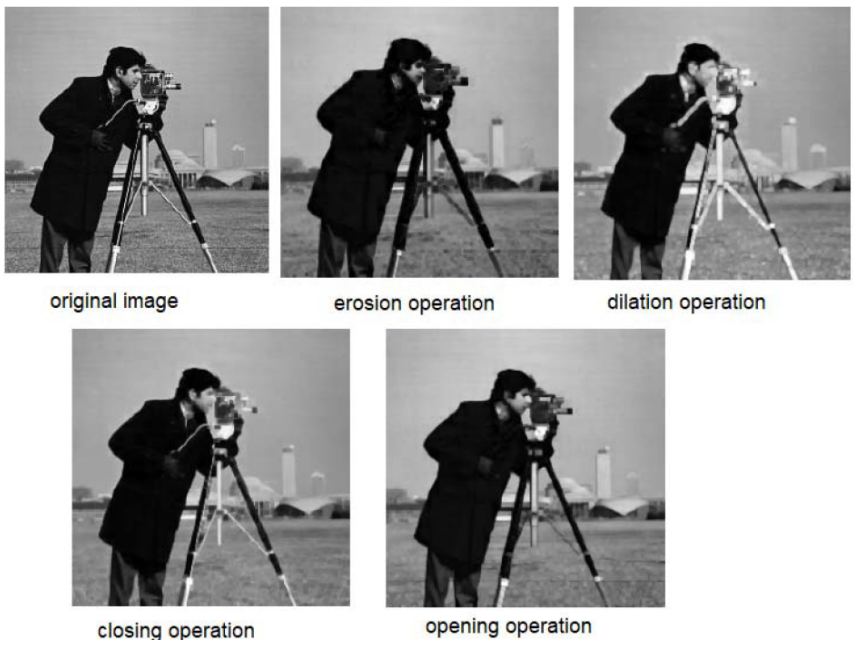
$$(f \circ b)(a) := [(f \ominus b) \oplus b](a)$$

Definição 9 (*Closing*)**.** *Definimos como closing a composição das operações básicas feita da seguinte forma:*

$$(f \bullet b)(a) := [(f \oplus b) \ominus b](a)$$

Basicamente, no ponto de vista algébrico, as operações de *opening* e *closing* mudam a ordem na qual cada uma das operações anteriores são aplicadas em uma imagem f. A Figura 4 representa um exemplo dessas quatro operações e suas diferenças ficam claras.

Figura 4 – Exemplo das operações de dilatação, erosão, *opening* e *closing* com elemento de estruturação escolhido arbitrariamente pelo autor.



Fonte: Figura presente em [39].

Ainda a partir das operações básicas, podemos definir outros operadores mais complexos voltados para detecção de bordas. De acordo com Rivest *et al.* [63], as operações são:

Definição 10 (Gradiente Morfológico)**.**

$$\nabla_M(f) := (f \oplus b) - (f \ominus b)$$

Definição 11 (Gradiente Interno)**.**

$$\nabla_I(f) := f - (f \ominus b)$$

Definição 12 (Gradiente Externo)**.**

$$\nabla_E(f) := (f \oplus b) - f$$

O gradiente interno atua destacando bordas internas de objetos brancos. Para imagens binárias, o gradiente interno irá retornar as bordas internas dos objetos da imagem. Já o gradiente externo busca extrair as bordas externas de um objeto branco. Os gradientes internos e externos normalmente são usados quando contornos finos são necessários para alguma determinada aplicação.

Em adição a essas operações apresentadas, em *Wilson* e *Ritter* [64] temos a noção de máximo e mínimo multiplicativo, que é definido como segue

Definição 13 (Máximo (Mínimo) Multiplicativo)**.** *Seja* $a, b \in (\mathbb{R}^n)_\infty^{\geq 0}$*, então o máximo multiplicativo é computado como segue*

$$a \vee b := \max_j \ (a_j b_j) \tag{3.7}$$

e o minimo multiplicativo por

$$a \wedge b := \min_j \ (a_j b_j) \tag{3.8}$$

Essa operação será utilizada posteriormente na proposição de uma camada de neurônios densos diferente dos *perceptrons*.

3.2.2 Introdução a Redes Morfológicas

As computações realizadas pelos modelos de redes neurais artificiais são baseadas na topologia dessas redes [65]. Para as redes clássicas, o sistema algébrico é o dos números reais munido com as operações de adição e multiplicação em conjunto com as propriedades dessas operações. Esse sistema algébrico é chamado de anel e normalmente é denotado por $(\mathbb{R}, +, .)$.

Em muita das redes morfológicas propostas [65], o sistema algébrico é baseado nos semianeis $(\mathbb{R}_{-\infty}, \vee, +)$ e $(\mathbb{R}_\infty, \wedge, +')$, onde $\mathbb{R}_\infty = \mathbb{R} \cup \{\infty\}$ e $\mathbb{R}_{-\infty} = \mathbb{R} \cup \{-\infty\}$. A aritmética básica e operações lógicas são representadas como seguem:

- O símbolo $+$ denota a operação de adição clássica considerando que dado $a \in \mathbb{R}_{-\infty}$ temos $a + (-\infty) = (-\infty) + a = -\infty$;
- O símbolo $+'$ denota o *dual* de $+$ e é definido por $a +' b = a + b$ sempre que $a, b \in \mathbb{R}$ e $a + (\infty) = (\infty) + a = \infty \; \forall a \in \mathbb{R}_{\infty}$;
- Similar às Equações (3.7) e (3.8), $\wedge$ e $\vee$ representam as operações de mínimo e máximo, respectivamente. Considerando uma regra extra que $a \vee (-\infty) = (-\infty) \vee a = a \; \forall a \in \mathbb{R}_{-\infty}$ e $a \wedge (\infty) = (\infty) \wedge a = a \; \forall a \in \mathbb{R}_{\infty}$.

Com o semianel $(\mathbb{R}_{-\infty}, \vee, +)$, podemos descrever um *perceptron* morfológico [66] [65] como segue:

$$f\left(\vee_j \left[x_j + w_j\right]\right), \tag{3.9}$$

dado que x_j seja uma componente do vetor de entrada, w_j uma componente do vetor estruturante e f uma função de ativação.

Por outro lado, usando o semianel $(\mathbb{R}_{\infty}, \wedge, +')$, um *perceptron* morfológico computaria o seguinte:

$$f\left(\wedge_j \left[x_j +' w_j\right]\right). \tag{3.10}$$

De acordo com [6], apresentamos as seguintes definições para neurônios morfológicos:

$$\tau_j(x) = p_j \vee_{i=1}^{n} \left[r_{ij}(x_i + w_{ij})\right] \tag{3.11}$$

ou

$$\tau_j(x) = p_j \wedge_{i=1}^{n} \left[r_{ij}(x_i +' w_{ij})\right], \tag{3.12}$$

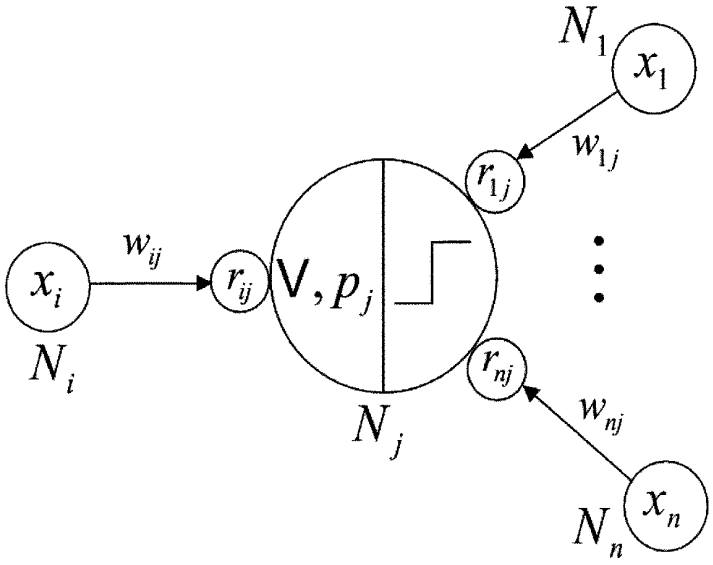
onde $r_{ij} = \pm 1$ indica se o i-ésimo neurônio causa uma ativação ou inibe o j-ésimo neurônio, $p_j = \pm 1$ denota a resposta de saída (ativação ou inibição) do j-ésimo neurônio para os neurônios conectados a ele. Por exemplo, para a ativação dos neurônios poderiamos adotar $r_{ij} = 1$ e $p_j = 1$ e para a inibição, $r_{ij} = -1$ e $p_j = -1$. A partir dessas equações, a computação realizada pelo $k - \acute{e}simo$ dendrito para uma entrada $x \in \mathbb{R}^n$ é dada por:

$$\tau_j(x) = p_j \vee_{i \in I} \vee_{l \in L} \left[r_{ik}^{l}(x_i + w_{ik}^{l})\right], \tag{3.13}$$

onde I corresponde ao conjunto de todos os neurônios N_i que se conectam ao $k - \acute{e}simo$ neurônio e L ao conjunto dos $l - \acute{e}simos$ pesos w_{ik} que se comunicam com o neurônio k.

Na Figura 5 temos uma ilustação de um neurônio que computa a Equação (3.11).

Figura 5 – Computação realizada por um neurônio morfológico.



Fonte: Figura presente em [6].

3.2.3 Media Contra-Harmônica

De acordo com Mellouli *et al.* [39], a média contra-harmônica (sigla CHM em inglês) foi proposta com o objetivo de computar operações não lineares robustas sem o uso dos operadores de máximo e mínimo. Pela definição que será apresentada a seguir, é possível notar-se uma semelhança com a operação de dilatação e erosão. De acordo com Angulo [67], podemos definir essa operação como:

Definição 14 (Média contra-harmônica)**.** *Dados $x, w \in \mathbb{R}_+^n$ e r, definimos a média contra-harmônica como*

$$CHM^r(x,w) = \begin{cases} \dfrac{\sum_{i=1}^n w_i x_i^r}{\sum_{i=1}^n w_i x_i^{r-1}} & se\ r \in \mathbb{R} \\ \max(x_i) & se\ r = \infty \\ \min(x_i) & se\ r = -\infty \end{cases} \tag{3.14}$$

Baseada nessa equação, foi proposta por Mellouli *et al.* [39] a camada MConv, que busca reproduzir operações morfológicas de dilatação e erosão. Definida como

Definição 15 (Camada MConv)**.**

$$MConv(f,w,p)(x) = \frac{(f^{p+1} * w)(x)}{(f^p * w)(x)} \tag{3.15}$$

no qual $*$ denota a operação de convolução definida pela Equação (3.4).

Entretanto, como essa operação é parecida com a proposta por Masci *et al.* [37], Kirszenberg *et al.* [9] comentam que a mesma possui limitações. São elas:

- Se a imagem f^p possuir valores negativos, p é não nulo e $p \in \mathbb{Z}$, então f^p pode conter números complexos;
- Os pesos w da camada são restritos a números positivos para satisfazer a equação da media contra-harmônica;
- Se $(f^p * w)(x) = 0$ para algum x, a operação $\dfrac{1}{(f^p * w)(x)}$ não está definida.

Assim, essas convoluções morfológicas que se baseiam na Equação (3.14) possuem essas limitações. Os autores ainda recomendam que as entradas da camada sejam reescaladas para o intervalo $[1, 2]$.

3.2.4 SMorph

Como vimos na seção anterior, as camadas morfológicas apresentadas que se baseiam no comportamento da média contra-harmônica possuem a desvantagem de as entradas dos dados precisarem estar no intervalo $[1, 2]$. Então, para contornar esse problema, a função α-*softmax* [68] foi empregada para definir o seguinte:

Definição 16 (α-softmax)**.**

$$\begin{aligned} \sigma_\alpha :&\mathbb{R}^n \longrightarrow \mathbb{R}^n \\ &x \longmapsto \sigma_\alpha(x) = \left(\frac{e^{\alpha x_i}}{\sum_{j=1}^n e^{\alpha x_j}} \right)_i \quad \forall i : 1, \ldots, n. \end{aligned}$$

Prova-se que, para essa equação, quando $\alpha \to \infty$ ($\alpha \to -\infty$) temos que $\sigma_\alpha(x) \to argmax(x)$ ($argmin(x)$).

Definição 17. *Para algum $\alpha \in \mathbb{R}$ definimos*

$$\begin{aligned} S_\alpha :&\mathbb{R}^n \longrightarrow \mathbb{R} \\ &x \longmapsto S_\alpha(x) = \langle x, \sigma_\alpha(x) \rangle = \frac{\sum_{i=1}^n x_i e^{\alpha x_i}}{\sum_{j=1}^n e^{\alpha x_j}} \end{aligned}$$

Assim, a partir dessa definição, obtemos as seguintes propriedades:

Proposição 1. *Seja $x \in \mathbb{R}^n, \alpha \in \mathbb{R}$. Temos que*

$$\lim_{\alpha \to \infty} S_\alpha(x) = \max_i x_i \tag{3.16}$$

$$\lim_{\alpha \to -\infty} S_\alpha(x) = \min_i x_i \tag{3.17}$$

e, também,

$$\lim_{\alpha \to 0} S_\alpha(x) = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{3.18}$$

Além disso, note que S_α é menos restritiva que a média contra-harmônica já que não restringe os elementos em um vetor x a serem estritamente positivos pela sua definição. Assim, não é mais necessário exigir-se que as entradas de um conjunto de dados sejam reescaladas para o intervalo $[1, 2]$. Kirszenberg *et al.* [9] apresentam a operação *SMorph* (de *Smooth Morphological*) a partir das propriedades presentes nas Equações (3.16) e (3.17):

$$SMorph(f, w, \alpha)(x) = \frac{\sum_{z \in W(x)} (f(z) + w(x - z)) e^{\alpha(f(z)+w(x-z))}}{\sum_{z \in W(x)} e^{\alpha(f(z)+w(x-z))}}, \tag{3.19}$$

em que $w : W \to \mathbb{R}$ assume o papel da função estruturante. A seguir, apresentamos outra equação da *SMorph* com o intuito de destacar a semelhança entre ela e a equação (3.4) que representa a convolução. Para cada ponto $[k, l]$ de uma imagem $m \times n$ teremos

$$SMorph(f, w, \alpha)[k, l] = \frac{1}{C} \sum_{i=1}^{m} \sum_{j=1}^{n} (f[i, j] + w[k - i, l - j]) e^{\alpha(f[i,j]+w[k-i,l-j])}, \tag{3.20}$$

com $C = \sum_{a=1}^{m} \sum_{b=1}^{n} e^{\alpha(f[a,b]+w[k-a,l-b])}$. Por definição, $C > 0$ para quaisquer que sejam os elementos f e w inseridos.

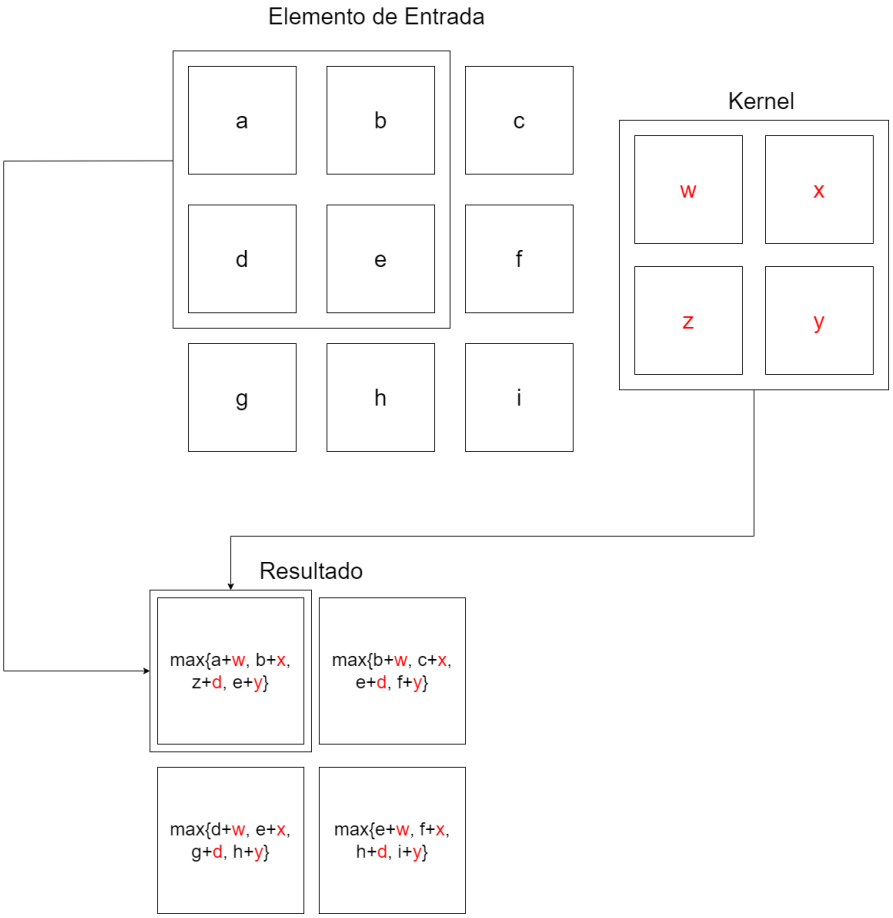
Veja que, por (3.16) e (3.17) temos que:

$$\lim_{\alpha \to \infty} SMorph(f, w, \alpha)(x) = \max_i (f_i + w_i)(x)$$
$$\lim_{\alpha \to -\infty} SMorph(f, w, \alpha)(x) = \min_i (f_i + w_i)(x)$$

Portanto, para $\alpha > 0$ ($\alpha < 0$) dizemos que a operação $SMorph$ obtém o efeito de uma pseudodilatação (pseudoerosão) e nos casos em que $\alpha >> 0$ ($\alpha << 0$) a $SMorph$ se aproxima da definição de dilatação (erosão) como foi apresentado na **Definição 6** e **7**.

Apesar de não ser explicitamente mostrada pelo autor, a Equação (3.19) da SMorph indica que é um tipo de convolução similar à realizada por (3.4) e exemplificado pela Figura 3. A Figura 6 representa um exemplo da operação SMorph computada considerando um α suficientemente grande para a propriedade do máximo ser válida, ficando claras as diferenças em relação a uma convolução clássica.

Figura 6 – Exemplo da operação SMorph.



Fonte: Autoria Própria.

3.3 Neural Architecture Search

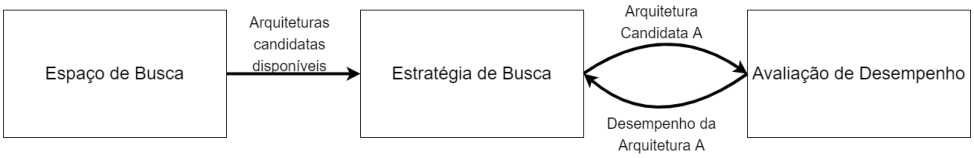
O sucesso do *deep learning* em tarefas perceptivas (imagem ou áudio, entre outros) se deve principalmente à automação do processo de extração de características. Seu sucesso foi acompanhado, porém, de uma demanda crescente para modelagem de arquiteturas, em que arquiteturas neurais cada vez mais complexas eram desenvolvidas [69]. O conceito de NAS, processo de modelagem automática de arquiteturas, é assim o próximo passo lógico. De acordo com [69], podem-se categorizar os métodos de NAS de acordo com 3 focos: *search space*, *search strategy* e *performance estimation strategy*.

- **Search Space:** O *search space* define quais arquiteturas podem ser representadas. A incorporação do conhecimento prévio sobre propriedades típicas das arquiteturas bem adequadas para algum tipo de tarefa pode reduzir o tamanho desse espaço e acelerar a busca por arquiteturas ótimas. Entretanto, essa abordagem introduz o viés humano, que pode impedir o algoritmo de encontrar estruturas inéditas para arquiteturas que vão além do que se pode encontrar fazendo uso da tentativa e erro.

- **Search Strategy:** A *search strategy* detalha como explorar o *search space* (que normalmente é exponencialmente grande ou até ilimitado). Ela engloba a abordagem clássica de *exploration-exploitation*, já que, por um lado, é desejável que se encontrem arquiteturas boas rapidamente, enquanto, por outro, a convergência prematura para um mínimo local "ruim" pode ser um problema.

- **Performance Estimation Strategy:** Tipicamente, o objetivo de NAS é encontrar arquiteturas que alcançam alta acurácia preditiva em dados não vistos durante o treino. *Performance Estimation* se refere ao processo de estimar essa performance: a opção mais simples é treinar e validar a arquitetura da maneira usual nos dados. Porém isso pode ser custoso computacionalmente e limita o número de arquiteturas que podem ser exploradas. Portanto, muitas pesquisas recentes focam em desenvolver métodos para reduzir o custo dessa estimativa de performance.

Figura 7 – Funcionamento de NAS.



Fonte: Autoria Própria.

A Figura 7 ilustra resumidamente como funciona o processo de NAS. Geralmente, os algoritmos de NAS começam utilizando um conjunto predefinido de operações e usa um controlador [70] para obter uma grande quantidade de arquiteturas neurais com base no *search space* que é criado a partir desse conjunto predefinido de operações. As arquiteturas candidatas são então treinadas e são categorizadas com base em uma métrica, usualmente sendo a acurácia da arquitetura em um conjunto, que compõe a *Performance Estimation Strategy*.

De acordo com [70], boa parte dos primeiros métodos de NAS seguiam essa abordagem acima [8, 41]. O trabalho de *Zoph* e *Le* [8] foi apresentado a partir do fato de que uma rede neural pode ser descrita como um vetor de variáveis. Assim, uma ideia intuitiva é que podemos usar uma rede neural recorrente como um controlador para gerar tais arquiteturas novas a partir de um vetor e, através de aprendizagem por reforço, otimizar esse controlador até finalmente ser obtida uma arquitetura ótima. A *Meta-QNN* [41] trata o processo de seleção das arquiteturas neurais como um processo de decisão de *Markov* [70] e usa o chamado $Q - learning$ para gerar recompensas que farão o método obter uma arquitetura neural ótima.

3.3.1 Bayesian Optimization and Hyperband

Nesta seção faremos um breve resumo sobre uma *search strategy* que será muito utilizada para a produção de resultados. Apresentada por *Falkner et al* [71], a técnica chamada de *Bayesian Optimization and Hyperband* (*BOHB*) é o uso combinado de 2 técnicas: *Bayesian Optimization* (BO) e *Hyperband* (HB). Portanto, a seguir fazemos uma breve revisão de ambos os métodos a partir das anotações apresentadas pelos autores de *BOHB*.

Bayesian Optimization: Em cada iteração i, o método de otimização Bayesiana usa um modelo probabilístico $p(g|D)$ para, a partir do conjunto D de pontos observados, modelar uma função objetivo g. De acordo com [71], a otimização Bayesiana usa uma função de aquisição a que se baseia no modelo probabilístico $p(g|D)$ atual. Tomando como base o modelo e a função de aquisição a, em cada iteração são realizados os seguintes passos:

1. Selecionar o ponto (x, y) que maximiza a função de aquisição a;
2. Avaliar a função objetivo nesse ponto;
3. Alimentar o conjunto D com novos pontos a partir dessa seleção e reavaliar o modelo.

Hyperband: Equanto, por um lado, é custoso computacionalmente avaliar uma função objetivo f por esse processo necessitar do treinamento do modelo de *machine learning* com os hiperparâmetros específicos, por outro, na maioria das aplicações é possível avaliar uma aproximação $\bar{f}$ dessa função objetivo f através do que é chamado de *budget*. O *budget* b é definido de forma que, com o *budget* máximo b_{max}, tenhamos $\bar{f}(x, b_{max}) = f(x)$ e para $b < b_{max}$ tenhamos $\bar{f}(x, b) \approx f(x)$, que geralmente se aproxima melhor à medida que $b \to b_{max}$. Assim, o método Hyperband [72] se aproveita dos *budgets* b usando repetidamente uma técnica chamada *SucessiveHalving* [73] para identificar a melhor dentre várias combinações de hiperparâmetros escolhidas aleatoriamente.

De acordo com os autores, BOHB usa o método Hyperband para determinar quantas arquiteturas (ou combinações de arquiteturas) serão avaliadas e qual o *budget* será considerado para cada uma. A inovação é que, em vez de selecionar aleatoriamente cada configuração como é feito no método Hyperband, BOHB substitui essa seleção por uma busca guiada. Assim que o número desejado de configurações é obtido na iteração do algoritmo, o procedimento de divisão usual presente no método Hyperband é feito usando essas configurações. Além disso, o algoritmo BOHB guarda, para cada configuração, as informações do valor da função que estamos avaliando para usar de base em futuras iterações.

A parte do algoritmo BOHB que usa Bayesian Optimization se assemelha muito ao método Tree Parzen Estimator (TPE), com uma grande diferença: o algoritmo BOHB

usa um único *Kernel Density Estimation* (KDE) com múltiplas dimensões, enquanto o TPE usa vários KDE com uma dimensão para manusear a interação no espaço de hiperparâmetros.

4 Metodologia Proposta

4.1 Motivação

Os algoritmos que realizam reconhecimento de padrões buscam tomar decisões a partir das complexas características presentes nos dados. O objetivo é lidar com tarefas que, muitas vezes, são feitas por humanos. Alguns exemplos: reconhecimento de rostos, movimentos no xadrez, decisões na bolsa de valores, diagnósticos médicos e muitos outros [74].

Nessas tarefas, normalmente é apresentado como entrada um conjunto de dados, possivelmente com rótulos associados. A tarefa dos algoritmos é identificar padrões nestes dados e a partir disso auxiliar na tomada de decisões. Os padrões em si costumam ser representados pelas chamadas *features*. Em particular, neste trabalho, focaremos na tarefa de classificação de imagens.

Quando se trata de reconhecimento de padrões e processamento de imagens na visão computacional clássica, uma abordagem bem conhecida é fornecida pela morfologia matemática, em que as imagens são consideradas conjuntos de pontos que são aplicados em operadores morfológicos. A morfologia matemática permite teorias não lineares para processamento de imagens e análise de padrões [75].

De acordo com Shih [75], em aplicações de visão industrial, a morfologia matemática pode ser usada para implementar o reconhecimento de objetos, aperfeiçoamento de imagens, análise de textura e inspeção de defeitos. Além dessas aplicações, as operações morfológicas podem ser usadas para a detecção de bordas em imagens e redução de ruídos, o que pode ser de fundamental importância na classificação das imagens em conjuntos de dados gerais.

Já nos últimos anos, grandes progressos foram realizados aplicando-se técnicas de *deep learning* na visão computacional. Em particular, as *Deep Convolutional Neural Networks* (DCNNs) obtiveram desempenhos no estado da arte em tarefas de reconhecimento de imagens. Entretanto, essas redes usualmente utilizam convoluções clássicas em sua estrutura e operações morfológicas, nas raras vezes em que são utilizadas, são apenas como uma forma de preprocessamento do conjunto de dados, por exemplo: Bnouni *et al.* [76] e Chen *et al.* [77]. Assim, o papel já conhecido da morfologia na visão computacional clássica permite que se conjecture que a substituição de convoluções tradicionais por operações morfológicas, como a de erosão e dilatação, pode trazer benefícios para a tarefa de classificação de imagens.

Atualmente, diversas estruturas morfológicas diferenciáveis, como aquelas em

[37] e [9], foram apresentadas, sendo a descrita em [37] já testada em classificação de imagens e mostrando melhoras na acurácia da rede. Entretanto, no trabalho de Kirszenberg *et al.* [9], que apresentou outra operação para simular a dilatação ou erosão diferencialmente, ainda não foram realizados testes para classificação. A avaliação dessa operação em uma arquitetura de rede neural tem potencial para trazer resultados comparáveis ao estado da arte.

Entretanto, um ponto ainda crítico nestes trabalhos introduzindo operações morfológicas em redes neurais é que geralmente eles são dependentes de uma série de hiper-parâmetros que precisam ser definidos pelo usuário e ajustados para cada nova tarefa. Neste cenário, um processo mais automatizado como o trazido por algoritmos de NAS é bem vindo. E o fato é que, apesar da popularidade de NAS e suas diversas técnicas já disponíveis na literatura, pouco se tem investigado sobre a criação de arquiteturas de redes neurais que contenham estruturas morfológicas, sendo [78] um dos poucos trabalhos disponíveis. Assim, um estudo mais profundo de NAS sobre camadas morfológicas pode trazer resultados promissores à área.

Com base em todo este contexto, o presente estudo se propõe a desenvolver um modelo de NAS com suporte a camadas morfológicas, permitindo assim que representações mais ricas de imagem e outros tipos de dados sejam produzidas pela introdução do operador morfológico ao mesmo tempo em que a automatização propiciada pelo algoritmo de NAS possibilita que menos decisões *ad-hoc* precisem ser tomadas pelo usuário.

4.2 Camadas Morfológicas Propostas

Abaixo apresentamos alguns tipos de neurônios morfológicos que propomos e usamos em nossas redes *feedforward* para a avaliação de resultados. A partir das definições de máximo e mínimo multiplicativo (veja **Definição 13**) e, ao desconsiderar suas restrições do conjunto, podemos usar a função S_α (veja **Definição 17**) para propor uma camada densa que se inspira na mesma computação realizada pelos *perceptrons* (veja Equação (3.2)). Como na definição abaixo essa camada não está respeitando as restrições impostas pela **Definição 13**, não podemos a chamar de morfológica. Entretanto o nome MorphMul é escolhido pois essa camada se inspira no uso da S_α, que é uma função utilizada para a criação de operações morfológicas.

Definição 18 (MorphMul (*Multiplication*))**.** *Seja h_i^l a saída da camada l, então o $i-$ésimo neurônio da camada MorphMul (Multiplication) computará a seguinte operação:*

$$h_i^l = S_{\alpha_i^l}(h^{l-1} w_i^l) + b_i^l \tag{4.1}$$

onde w_i^l é um vetor de estruturação (ou vetor de pesos sinápticos) do $i-$ésimo neurônio na camada l, b_i^l é o viés da $l-$ésima camada, h^{l-1} será o elemento de saída presente na

$(l-1)$-ésima camada da rede e α_i^l será o parâmetro que controlará o comportamento da função S_α do $i-$ésimo neurônio da camada l.

Perceba que, a partir das propriedades apresentadas nas Equações (3.16), (3.17) e (3.18), essa camada morfológica apresentada pode computar as seguintes equações conforme α varia:

(i) Quando $\alpha >> 0$, temos o seguinte cálculo:

$$S_{\alpha_i^l}(h^{l-1}w_i^l) + b_i^l \approx \max\{h^{l-1}w_i^l\} + b_i^l;$$

(ii) Quando $\alpha << 0$, obtemos:

$$S_{\alpha_i^l}(h^{l-1}w_i^l) + b_i^l \approx \min\{h^{l-1}w_i^l\} + b_i^l;$$

(iii) Dado que n é a quantidade de elementos que entram na camada l, conforme α se aproxima de 0 temos

$$S_{\alpha_i^l}(h^{l-1}w_i^l) + b_i^l \approx \frac{h^{l-1}w_i^l}{n} + b_i^l.$$

Observe que os itens (i) e (ii) computam operações não lineares características de operações morfológicas e (iii) computa uma operação similar à Equação (3.2) computada por uma camada com neurônios do tipo *perceptron*. Chamamos uma variante da camada definida na **Definição 18** de PositiveMorphMul, com a definição vista a seguir, a camada respeita as restrições impostas pelo máximo e mínimo multiplicativo e tem características de uma operação morfológica.

Definição 19 (PositiveMorphMul (*Positive Multiplication*))**.** *Seja h_i^l a saída da camada l, então o $i-$ésimo neurônio da camada MorphMul (Multiplication) computará a seguinte operação:*

$$h_i^l = S_{\alpha_i^l}(h^{l-1}w_i^l) + b_i^l \tag{4.2}$$

onde w_i^l é um vetor de estruturação (ou vetor de pesos sinápticos) do $i-$ésimo neurônio na camada l tal que cada uma de suas componentes satisfaçam $(w_i^l)_j \geq 0$, b_i^l é o viés da $l-$ésima camada, h^{l-1} será o elemento de saída resultante da $(l-1)$-ésima camada da rede e, assim como w_i^l, as componentes $(h^{l-1})_j$ deverão ser positivas.Por fim, α_i^l será o parâmetro que controlará o comportamento da função S_α do $i-$ésimo neurônio da camada l.

Em seu trabalho, Silva e Sussner [66] revisam o chamado *perceptron* morfológico que utiliza as noções de erosão e dilatação já citadas anteriormente. Assim, definiremos esse tipo de neurônio em uma camada seguindo as mesmas intuições com o uso da função S_α, onde os itens $(i), (ii)$ e (iii) se aplicam também para essa camada.

Definição 20 (MorphAdd (*Addition*))**.** *Seja h_i^l a saída da camada l, então o $i-$ésimo neurônio da camada MorphAdd (Addition) computará a seguinte operação:*

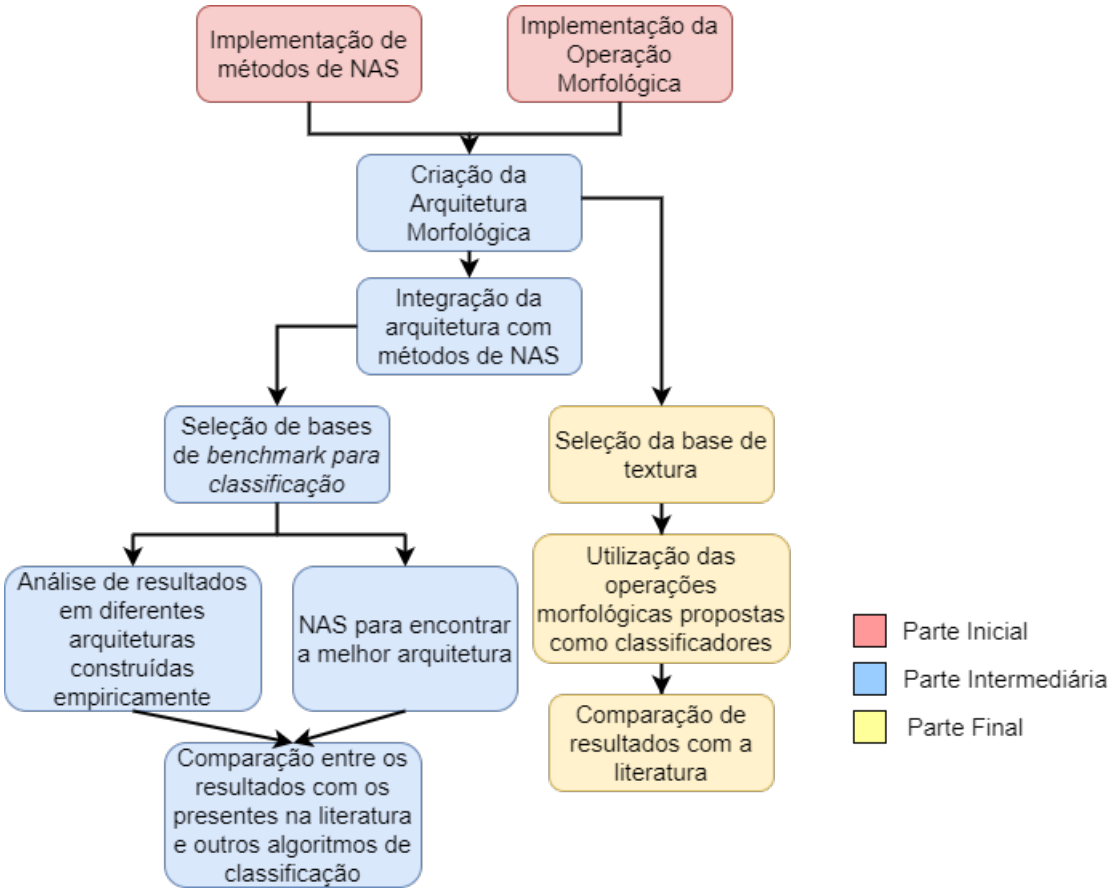
$$h_i^l = S_{\alpha_i^l}(h^{l-1} + w_i^l) + b_i^l \tag{4.3}$$

onde w_i^l é um vetor de estruturação (ou vetor de pesos sinápticos) do $i-$ésimo neurônio na camada l, b_i^l é o viés da $l-$ésima camada, h^{l-1} será o elemento de saída presente na $(l-1)$-ésima camada da rede e α_i^l será o parâmetro que controlará o comportamento da função S_α do $i-$ésimo neurônio da camada l.

4.3 Metodologia Geral

O presente estudo visa investigar técnicas de NAS associadas a arquiteturas de redes neurais com operadores morfológicos. Mais precisamente, este trabalho explorará a implementação das operações morfológicas propostas, em particular as que estão propostas nas equações (4.1), (4.3) e a operação *SMorph*, para a classificação de dados. Todas essas operações ainda não foram avaliadas em tarefas para classificação de dados, permitindo assim incluirmos essa operação no *search space* de NAS agregando um grau maior de liberdade para a busca de hiperparâmetros ótimos. Na Figura 8 vemos um pequeno resumo dos passos lógicos para este trabalho.

Figura 8 – Resumos das etapas do projeto. Aqui vemos os passos lógicos e como cada etapa foi divida para a realização do trabalho.



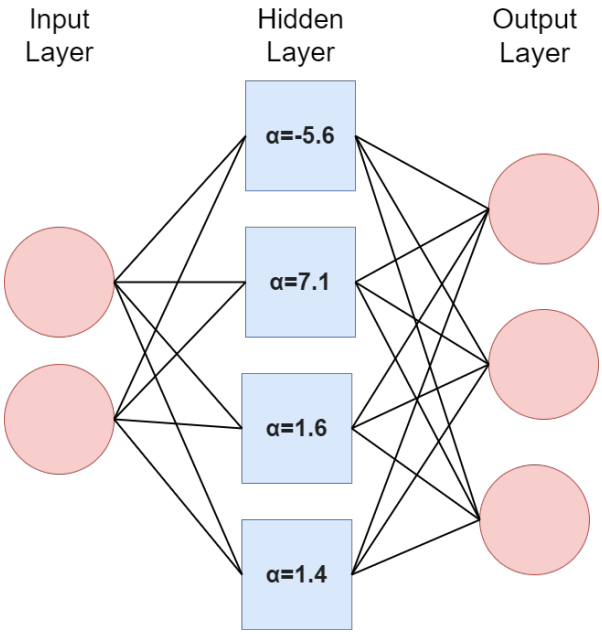
Fonte: Autoria Própria.

Para as camadas densas nas equações (4.1) e (4.3), temos como objetivo compará-las a outros métodos de classificação de dados, a SVM, regressão logística e o método XGBoost (apresentado em [79])em diversos *datasets* com problemas reais e alguns *datasets* sintéticos. Em particular, as arquiteturas contendo essas operações serão feitas com ideias presentes em [40], que faz o uso de arquiteturas híbridas contendo perceptrons clássicos e operações morfológicas em suas camadas. Com essas operações em arquiteturas densas talvez seja possível obter uma generalização melhor com uma menor quantidade de parâmetros dependendo do *dataset* utilizado. Além disso, será feita uma comparação entre as camadas MorphMul e PositiveMorphMul, já que uma é definida sem respeitar as restrições do máximo e mínimo multiplicativo e a outra já é feita as respeitando.

Exemplificando, a Figura 9 mostra uma arquitetura morfológica simples onde os quadrados representam neurônios morfológicos da camada representada pela Equação (4.1) e também mostramos o α aprendido pela rede em um problema sintético representado na Figura 10, que mostra a capacidade de generalização da rede. Como comparação, a Figura 11 mostra uma rede neural com a mesma arquitetura da rede híbrida substituindo os neurônios morfológicos por *perceptrons* clássicos. Podemos ver que a fronteira de decisão

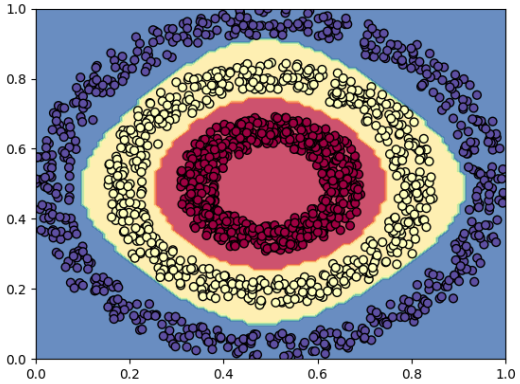
da rede neural falha em generalizar o problema não linear com tão poucos pontos.

Figura 9 – Exemplo de arquitetura híbrida.



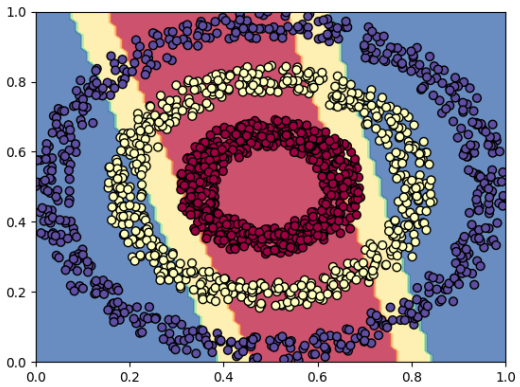
Fonte: Autoria Própria.

Figura 10 – Fronteira de decisão usando arquitetura híbrida.



Fonte: Autoria Própria.

Figura 11 – Fronteira de decisão usando arquitetura contendo apenas *perceptrons* clássicos.

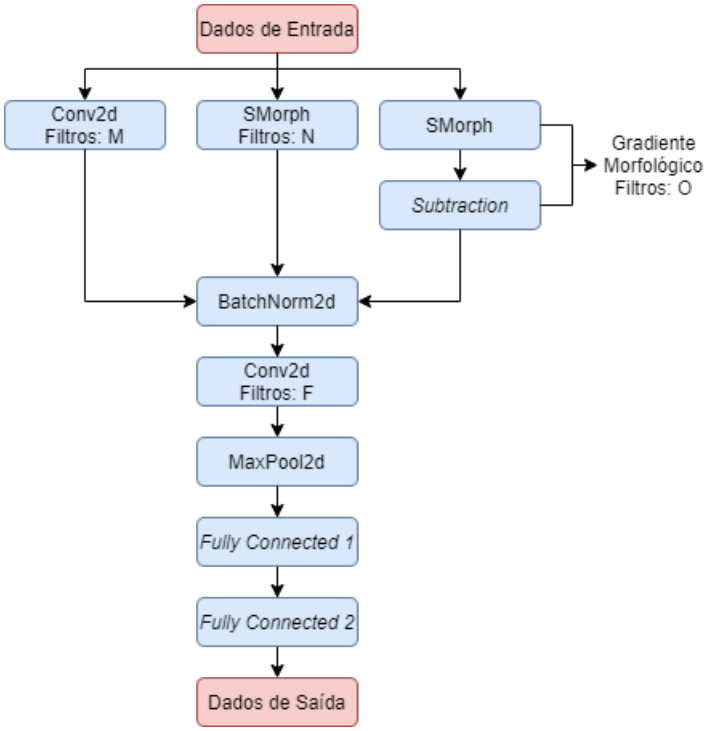


Fonte: Autoria Própria.

Já para a operação *SMorph*, a arquitetura para classificação de dados é proposta na Figura 12, tendo em vista oferecer múltiplas operações na primeira camada para o algoritmo de NAS decidir a configuração de *feature maps* ótima para a rede. Como será explicado na próxima seção, utilizaremos NAS para otimizar um vetor de hiperpârametros que irá denotar a quantidade e o tipo de filtros presentes em cada uma das operações nas duas primeiras camadas da arquitetura.

Vale ressaltar que aqui as operações morfológicas são usadas para auxiliar na tarefa de classificar dados e não necessariamente essa camada morfológica substituirá por completo a convolução clássica. De fato, Hu *et al.* [78] explicam que, apesar de normalmente redes morfológicas apresentarem resultados superiores em várias aplicações, essa metodologia tem seu desempenho comprometido se sua arquitetura não for bem construída. Assim, podemos supor que os resultados gerados por esse tipo de camada podem conter informações úteis para auxiliar a tomada de decisão em outras camadas da rede neural e melhorar a acurácia da rede, mas não deve ser desconsiderado o uso de convoluções clássicas na classificação de imagens pois a combinação dessas operações pode trazer resultados benéficos em uma arquitetura. Assim, procuramos propor uma arquitetura com múltiplas operações na primeira camada, onde a escolha de uma operação não descartará a outra.

Figura 12 – Arquitetura proposta para classificação. Nessa imagem apresentamos as conexões, camadas e tipo de operação que estão contidas na nossa arquitetura para classificação de imagens.



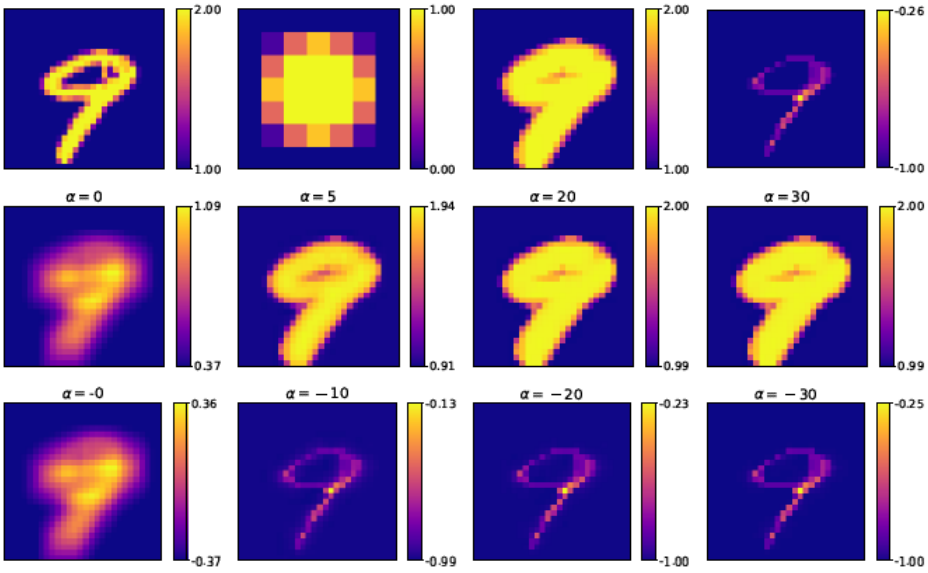
Fonte: Autoria Própria.

Em particular, a escolha da operação morfológica *SMorph* na arquitetura tem como motivo principal a sua diferença em relação a outras operações morfológicas que reside no fato de que sua Equação (3.19) é totalmente diferenciável, incluindo seu parâmetro α, que controla a realização de erosão ou dilatação pela camada. Assim, redes que possuem a *SMorph* em alguma de suas camadas podem ser treinadas com *backpropagation*.

Logo, de acordo com os resultados presentes nos experimentos de Kirszenberg *et al.* [9], uma rede criada com a utilização da *SMorph* pode aprender seu elemento de estruturação, além de também encontrar o parâmetro α correto para reproduzir uma erosão ($\alpha < 0$) ou dilatação ($\alpha > 0$), através do treinamento com *backpropagation*.

A Figura 13 apresenta os resultados obtidos por Kirszenberg *et al.* treinando a operação *SMorph* na base de dados *MNIST* com o objetivo de reproduzir uma erosão ou dilatação dado um elemento alvo, isto é, dada uma dilatação (ou erosão) feita através de operações não diferenciáveis no dígito de entrada. Logo, visualmente é evidente a semelhança e resultados posteriores presentes no trabalho de Kirszenberg *et al.* comprovam que de fato a *SMorph* se aproxima dessas operações.

Figura 13 – Exemplo da operação realizada pela *SMorph*.



Fonte: Kirszenberg *et al.* [9]

Outro ponto importante é que a arquitetura proposta apresenta múltiplas operações na primeira camada convolucional, sendo elas: convolução clássica, *SMorph* e Gradiente Morfológico. Essa terceira operação é exclusiva para redes morfológicas que realizem erosão e dilatação, já que essas operações fazem parte de sua definição e, como foi mostrado anteriormente, a operação escolhida por nossa arquitetura reproduz com sucesso a dilatação e erosão, assim podendo utilizar a técnica de gradiente morfológico. De acordo com Rivest [63], os gradientes morfológicos permitem o destaque de bordas da imagem, como podemos ver em um exemplo na Figura 14.

Figura 14 – Exemplo de uma técnica para extração das bordas.

Fonte: Autoria Própria.

Por fim, cabe a observação de que na morfologia matemática existem diferentes técnicas para a realização de dilatação e erosão em imagens que possuem mais de um canal, por exemplo, imagens RGB. Benavent *et al.* [80] comenta que existem duas possibilidades para que se definam operações morfológicas em imagens com cor, sendo elas: considerar as cores como um rótulo para cada pixel, usar os valores das cores para estabelecer uma

ordem total no espaço das cores e considerar cada canal de cor separadamente. Entretanto, como Kirszenberg *et al.* [9] trata apenas imagens em tons de cinza na equação da *SMorph* (3.19), o presente trabalho busca também explorar apenas imagens em tons de cinza.

5 Discussão e Resultados Experimentais

O presente estudo tem o objetivo de avaliar a performance de arquiteturas morfológicas, em particular com o uso das operações apresentadas nas equações (3.19), (4.1) e (4.3). Para isso, utilizaremos conjuntos de dados $n-$dimensionais para a construção e avaliação das arquiteturas densas morfológicas e conjuntos de dados para classificação de imagens, sendo eles a base *MNIST* e *GTSRB*, que serão úteis para a avaliação da operação $SMorph$. Além disso, definiremos um *search space* e utilizaremos NAS baseado em 2 métodos para otimização de hiperparâmetros, sendo um denominado *Random Search* [81] e outro chamado *BOHB* [71], que otimizará, no caso dos *datasets* de imagem, um vetor definindo a quantidade de filtros presentes nas duas primeiras camadas da arquitetura proposta (Figura 12) e construirá arquiteturas densas para a classificação de *datasets* $n-$dimensionais.

5.1 Implementação e Base de Dados

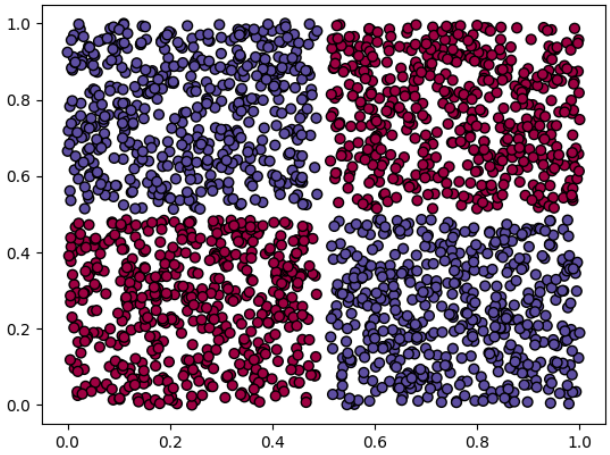
Para a implementação, conforme foi descrito na metodologia proposta, foi utilizada a linguagem *Python* em conjunto com o pacote *PyTorch*, que fornece as operações para a criação, customização e treinamento de redes neurais profundas. Além disso, será utilizado o pacote *NNI* para a aplicação das técnicas de NAS e *Kornia* para a reprodução de uma dilatação (erosão) não diferenciável.

5.1.1 Base de dados multivariada

A seguir, vamos mostrar imagens e tabelas com informações a respeito das bases multivariadas que serão utilizadas para avaliarmos o desempenho das camadas morfológicas dessas em conjunto com métodos de NAS. Em geral, os dados que representam problemas reais foram obtidos em [82], porém também usaremos alguns dados criados artificialmente para avaliarmos a fronteira de decisão gerada pelas arquiteturas e a capacidade de generalização das arquiteturas morfológicas em *datasets* não lineares.

O primeiro *dataset* sintético foi criado com base em um problema clássico, o problema não linear XOR (ou similarmente, XNOR), e pode ser visto na Figura 15.

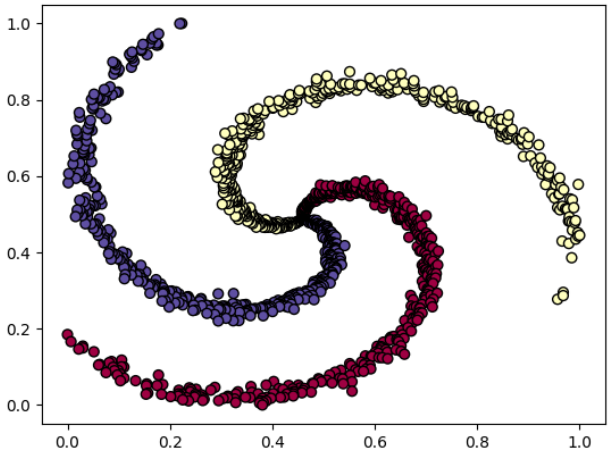
Figura 15 – Dataset sintético XOR.



Fonte: Autoria Própria.

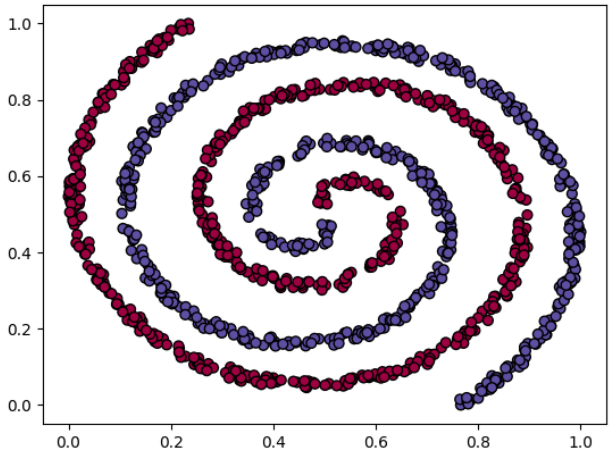
Já para as Figuras 16 e 17 foram usados *datasets* de espirais que são claramente não-lineares, com a diferença que um deles possui mais de 2 classes para a classificação enquanto o outro tem apenas 2 classes porém dão diversas voltas em torno do centro da espiral, o que pode deixar o problema mais complexo.

Figura 16 – Dataset sintético 3 *spirals*.



Fonte: Autoria Própria.

Figura 17 – *Dataset* sintético *spirals with loop*.



Fonte: Autoria Própria.

Tabela 1 – Detalhes sobre os *dataset* multivariados.

Nome	Características (Dimensão)	Número de Classes	Amostras de Treino	Amostras de Teste
XOR	2	2	1400	600
Spirals with loops	2	2	840	360
3 spirals	2	3	840	360
Vertebral Column	6	2	217	93
TicTacToe	9	2	671	287
Wine Quality (White)	11	10	3429	1469
Letter Recognition	16	26	14000	6000
Wine	13	3	178	53
Diabetic Retinopathy Debrecen	19	2	805	346
Default of Credit Cards Clients	24	2	21000	9000
Teaching Assistant Evaluation	5	3	106	45

A Tabela 1 faz uma breve descrição dos *datasets* a serem utilizados no primeiro experimento, que consistirá na avaliação da capacidade de generalização de uma rede neural com o uso de camadas morfológicas.

5.1.2 Base de dados para imagens

A seguir, descrevemos os conjuntos de dados com imagens usados como *benchmark* para a rede convolucional.

- **MNIST:** É uma base de dados amplamente utilizada em *machine learning*, contendo dígitos escritos à mão [24]. Sendo composto por imagens 28×28 em escala de cinza, essa base contém 60000 imagens para treino e 10000 para teste.

- **GTSRB:** Denominada *German Traffic Sign Recognition Benchmark*, essa base de dados foi apresentada em [83] e cada imagem contém placas de transito de tamanho variado. Na *GTSRB*, cada imagem está em RGB, possuindo no total 43 tipos de placas diferentes para a classificação. Seu conjunto de treino é composto por 39208 imagens e o de teste possui 12630. Particularmente neste trabalho, com o objetivo de reduzir o custo computacional e padronizar um tamanho para as imagens, todas elas são redimensionadas para o tamanho 50×50.

- **FMD:** A *Flickr Material Database* (FMD) é uma base de texturas composta por imagens coloridas de superfícies que pertencem a uma das seguintes 10 categorias de materiais: tecido, folhagem, vidro, couro, metal, papel, plástico, pedra, água e madeira. Cada uma dessas classes contém 100 elementos para treino, das quais 50 elementos são fotografias tiradas de perto do elemento e 50 são vistas regulares com algum plano de fundo. As imagens estão padronizadas no tamanho 224×224 para o uso em arquiteturas como a $VGG - 16$.

As Figuras 18 e 19 apresentam alguns exemplos dos dados presentes em cada uma das bases de *benchmark* a serem usadas. A expectativa é que a estratégia adotada para NAS consiga encontrar a melhor combinação para o vetor que indicará a quantidade de filtros nas três operações propostas na arquitetura e consiga obter resultados próximos ao estado da arte em cada um dos conjuntos de dados.

Figura 18 – Exemplo de elementos presentes na *MNIST*.

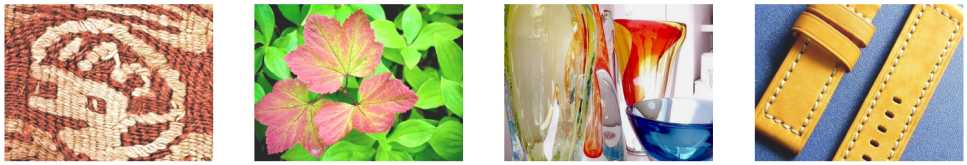
Fonte: Imagens presentes em [24].

Figura 19 – Exemplo de elementos presentes na *GTSRB*.

Fonte: Imagens presentes em [83].

A Figura 20 apresenta exemplos da base de texturas FMD com elementos de diferentes classes. Fica clara a diferença entre os elementos de cada classe e com uma grande variação entre os elementos de uma mesma classe. Além disso, o fundo da imagem, em alguns casos, contém partes que não auxiliam na detecção da classe.

Figura 20 – Exemplo de elementos presentes na base *FMD*.



Fonte: Imagens presentes em [84].

5.2 Ajuste dos Hiperparâmetros

5.2.1 Operação SMorph

Conforme foi descrito na metodologia, o α da operação será o mesmo no cálculo em cada canal de entrada. Porém, o α também pode ser definido para cada canal de saída gerado por uma mesma camada morfológica, tendo assim uma complexidade maior já que, em cada canal de saída, poderemos calcular uma dilatação ou erosão. Como essa abordagem resultará em um custo computacional maior, inicialmente será adotado um único α para todos os canais de saída. Assim o único parâmetro que irá diferenciar os *feature maps* será o elemento de estruturação (*kernel*).

Apesar de Kirszenberg *et al.* [9] conseguir treinar os parâmetros w e α da *SMorph*, nos testes utilizando as bases para classificação descritas, o parâmetro α da operação (3.19) não convergiu corretamente através do treinamento com *backpropagation*, prejudicando assim a acurácia da rede em alguns casos. Possivelmente, as configurações para treinamento adotadas não foram as ideais para resultar na sua convergência. Portanto, na tabela de resultados também serão adicionados casos em que α será considerado como um hiperparâmetro da rede, isto é, definido e fixo antes do treinamento da arquitetura.

5.2.2 Camadas Morfológicas Densas

Diferentemente da operação de convolução $SMorph$, quando as camadas morfológicas densas propostas forem usadas, todos seus parâmetros serão treináveis incluindo o α. Assim cada neurônio das camadas densas morfológicas serão capazes de aprender a operação não linear correta através de *backpropagation*.

5.3 Configuração de Treinamento

5.3.1 Experimento 1 - A

Para o Experimento 1 - A, os *datasets* apresentados na Tabela 1 estarão com todas suas características redimensionadas para o intervalo $[0, 1]$. Esse intervalo foi encontrado empiricamente, por resultar nos melhores desempenhos das arquiteturas nos diferentes *datasets*.

A ideia do Experimento 1 será o uso das camadas propostas nas equações (4.1) e (4.3) para a construção de arquiteturas que, além disso, se inspiram em arquiteturas híbridas como as apresentadas em [40]. Assim, a Tabela 2 apresenta as configurações nas quais cada arquitetura construida empiricamente será testada e comparada com uma SVM. Caso a acurácia atingida com essas arquiteturas não seja satisfatória no *dataset*, são utilizadas técnicas de *Neural Architecture Search* para a construção de arquiteturas mais otimizadas. Um dos motivos para isso ser possível, apesar de normalmente algoritmos de NAS terem um custo computacional muito grande, é que os *datasets* utilizados têm baixa dimensão e poucos exemplos, possibilitando assim a avaliação de diversas arquiteturas.

Como cada *dataset* é um caso particular, o *search space* de NAS será diferente em cada um deles e, quando necessário, a tabela desse espaço será mostrada junto dos resultados.

Chamaremos de *MorphMul* a camada representada pela Equação (4.1) e *MorphAdd* a camada da Equação (4.3). Assim, podemos padronizar um nome para essas camadas.

Tabela 2 – Configuração de treinamento para o Experimento 1.

		Configuração de Treinamento		
Nome	Algoritmo de Otimização	Taxa de Aprendizagem	Épocas	Tamanho de Batch
XOR	Adam	0.002	500	100
Spirals with loops	Adam	0.005	1500	100
3 spirals	Adam	0.005	1500	100
Vertebral Column	Adam	0.001	1500	32
TicTacToe	Adam	0.01	1000	64
Wine Quality (White)	Adam	0.005	2000	250
Letter Recognition	Adam	0.005	1200	512
Wine	Adam	0.005	1000	10
Diabetic Retinopathy Debrecen	Adam	0.001	3000	32
Default of Credit Cards Clients	Adam	0.001	1000	1024
Teaching Assistant Evaluation	Adam	0.005	2000	16

5.3.2 Experimento 1 - B

Esse experimento consistirá na comparação entre duas operações, da camada MorphMul e PositiveMorphMul, apresentadas nas **Definições 18 e 19**. A partir dos resultados do Experimento 1 - A e dos *datasets* em que foi necessário o uso da MorphMul, avaliamos também os resultados da PositiveMorphMul para verificar se, ao respeitar as hipóteses morfológicas, foi possível obter um ganho de acurácia. Em geral, as configurações de treinamento são as mesmas apresentadas na Tabela 2.

5.3.3 Experimento 2

Os *pixels* nas imagens de ambos os conjuntos de dados estão no intervalo $[0, 1]$ e, em particular, as imagens da base *GTSRB* são convertidas para tons de cinza com o objetivo de se adequar à arquitetura proposta. Como nos primeiros experimentos, são efetuados testes para verificar a diferença de acurácia com diferentes hiperparâmetros, de modo que não se utiliza um subconjunto para validar.

A inicialização dos pesos, tanto da convolução (ou *SMorph*) quanto das camadas totalmente conectadas nas arquiteturas, é feita com uma distribuição uniforme com limite inferior e superior dependendo da quantidade de características presentes e, em cada uma das camadas, com exceção das camadas de entrada e saída, terá a função de ativação $LeakyReLU$ para inserção da não linearidade na rede. Além disso, nas camadas totalmente conectadas, usa-se a técnica de *Dropout*, em que uma quantidade aleatória de neurônios,

baseando-se em uma porcentagem pré-definida, é eliminada durante o treino, visando assim atenuar um possível *overfitting*.

Além disso, a arquitetura para as bases *MNIST* e *GTSRB* são treinadas com *backpropagation* utilizando o algoritmo *Adam* [85], com um *learning rate* inicial de 0.01, 25 épocas de treino e com a diferença presente apenas no tamanho de *batch*, o qual na *MNIST* é igual a 128 em todos os testes e na *GTSRB* é igual a 64. Também, em cada uma das bases de dados, a quantidade de neurônios (hiperparâmetro) presentes nas camadas totalmente conectadas será diferente, conforme mostra os valores na Tabela 3.

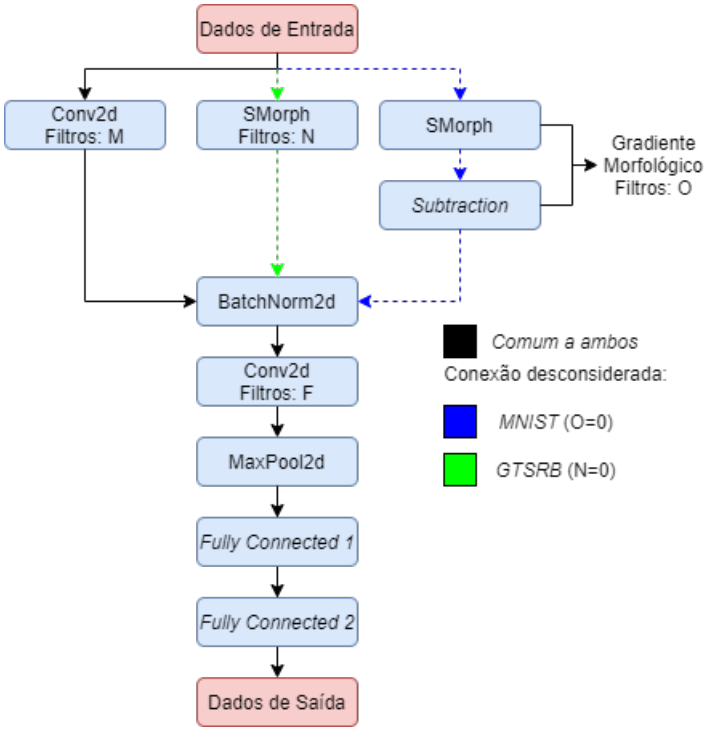
Tabela 3 – Quantidade de Neurônios nas Camadas.

Quantidade de Neurônios		
	MNIST	*GTSRB*
Fully Connected 1	120	256
Fully Connected 2	84	128
Dados de Saída	10	43

Na tabela de resultados, a quantidade de filtros presentes na arquitetura da Figura 12 pode ser representada pelo vetor $v = (M, N, O, F)$, que pretendemos otimizar utilizando NAS. Quando for fixado como hiperparâmetro, o elemento α é destacado também. Como, em um primeiro momento, buscamos apenas comparar a acurácia entre diferentes hiperparâmetros escolhidos manualmente para a arquitetura descrita na Figura 12, então não é separado um conjunto de validação para avaliar as diferentes combinações.

Em particular, a existência de diversas combinações resulta em um tempo de treinamento maior para NAS, então na *MNIST* será considerado $O = 0$ e, na *GTSRB*, $N = 0$ e, quando uma quantidade de filtros for igual a 0, desconsideraremos a respectiva conexão, conforme mostrado na Figura 21. Assim, em vez do vetor $v = (M, N, O, F)$, representaremos a *MNIST* com o vetor $v = (M, N, F)$ e *GTSRB* com $v = (M, O, F)$.

Figura 21 – Conexões desconsideradas em cada base. Traços pontilhados indicam conexões desconsideradas e a cor indica a base de dados no qual isso está se aplicando, traços pretos indicam conexões comum a ambas.



Fonte: Autoria Própria.

Já para a segunda etapa do experimento, como já mencionado, buscamos encontrar, respectivamente, o vetor $v = (M, N, F)$ e $w = (M, O, F)$ com os hiperparâmetros ótimos para *MNIST* e *GTSRB* por meio de NAS usando duas estratégias de busca diferentes, sendo elas: *Random Search* e *BOHB*. Para isso, a rede é treinada nas mesmas bases especificadas anteriormente, sendo essas separadas em: conjunto de treino, validação e teste. O processo segue até que seja encontrado o melhor vetor de hiperparâmetros, isto é, o que atinge maior acurácia no conjunto de validação.

Além disso, para o *Random Search*, durante a busca pelo melhor vetor de hiperparâmetros, a rede é treinada parcialmente com o algoritmo Adam por 15 épocas com o objetivo de reduzir o tempo de execução do algoritmo e, no fim, a configuração ótima é treinada por 25 épocas, como foi feito nos primeiros experimentos, no conjunto de treino + validação e avaliada no de teste. Usando o método *BOHB* a única diferença será que cada arquitetura será treinada parcialmente por entre 5 e 15 épocas, porém a configuração ótima também será treinada durante 25 épocas para a comparação.

Com esse objetivo, será necessário construir um *search space* que defina os possíveis valores para a quantidade de cada um dos filtros da arquitetura e, como foi mencionado anteriormente, também incluiremos α nesse *search space* como um parâmetro treinável ou hiperparâmetro fixo durante o treinamento. A Tabela 4 mostra os possíveis

valores a serem assumidos pelos hiperparâmetros. Devido a limitações computacionais, restringiremos esse espaço de busca a um conjunto pequeno de valores. Ademais, para evitar que o método de NAS escolha sempre a maior quantidade de filtros, uma pequena restrição será considerada nesse *search space*, isto é, $M+N+O \leq 24$ para a arquitetura ser válida. Como *BOHB*, de acordo com o autor do método, converge mais rapidamente que métodos aleatórios como *Random Search*, o *search space* de NAS com *BOHB* na Tabela 5 contém alguns hiperparâmetros extras para otimizar como, por exemplo, o número de neurônios em cada camada escondida após as camadas convolucionais, que no caso de *Random Search* eles continuam fixos seguindo a mesma configuração da Tabela 3.

Tabela 4 – *Search Space* com *Random Search*

Tipo de Operação	*Search Space* com Número de Filtros
Convolução Clássica (M)	{0,4,8,12,16,24}
Operação Morfológica (N)	{0,4,8,12,16,24}
Gradiente Morfológico (O)	{0,4,8,12,16,24}
Convolução Clássica (F) (Segunda Camada)	{4,6,12}
α (Dilatação ou Erosão)	Treinável ou {-5,-2.5,2.5,5}

Tabela 5 – *Search Space* com *BOHB*

Tipo de Operação	*Search Space*
Convolução Clássica (M)	{0,4,8,12,16,24}
Operação Morfológica (N)	{0,4,8,12,16,24}
Gradiente Morfológico (O)	{0,4,8,12,16,24}
Convolução Clássica (F) (Segunda Camada)	{4,6,12}
α (Dilatação ou Erosão)	Treinável ou {-5,-2.5,2.5,5}
Neurônios na Primeira Camada Escondida	{256,240,150,120,100}
Neurônios na Segunda Camada Escondida	{140,128,84,60,40}
Taxa de *Dropout*	Valor no intervalo uniforme $[0, 1]$

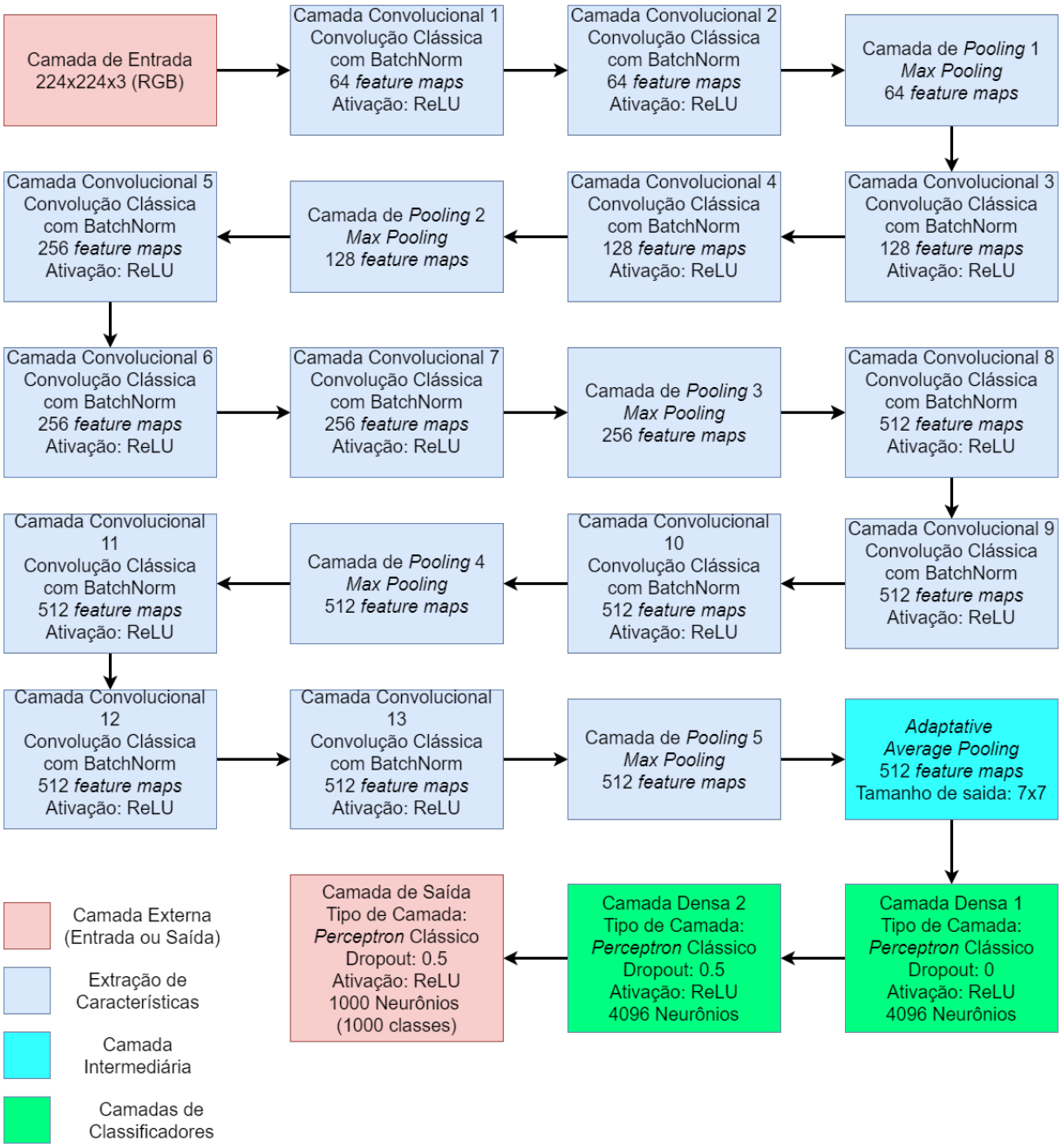
Portanto, a seguir mostraremos os resultados da arquitetura proposta que, quando $N \neq 0$ ou $O \neq 0$, chamaremos de *Smooth Morphological Convolutional Neural Network* (SMCNN) e, caso contrário, apenas de CNN.

5.3.4 Experimento 3

Devido à baixa quantidade de elementos no conjunto de dados FMD, no terceiro e último experimento, aplicaremos *transfer learning* [86] na arquitetura $VGG-16$ com *batch normalization* [27], que foi a arquitetura que melhor se adaptou ao conjunto de dados.

A $VGG-16$ com *batch normalization* é apresentada na Figura 22. Com o uso do *transfer learning*, os pesos de todas as suas camadas estarão pré-treinados na base *ImageNet* e apenas adaptaremos as camadas densas de classificadores para avaliar a acurácia na base.

Figura 22 – Arquitetura da VGG-16 com *batch normalization* para a ImageNet.



Fonte: Autoria Própria.

Adotaremos um método para avaliação da acurácia no *dataset* FMD que já é utilizado por outros trabalhos, como [87] e [88]. O método de avaliação consiste em separar randomicamente 50% de cada classe para treino e teste, avaliar a acurácia, repetir o processo 10 vezes e tirar a média. Trabalhos como [89], [90] e [91] também avaliam esse *dataset* com um método similar.

A ideia desse experimento será usar pelo menos uma das camadas morfológicas MorphMul e MorphAdd propostas, respectivamente, nas Equações (4.1) e (4.3) como camadas densas de classificadores usando as características extraídas pela $VGG-16$ pré treinada como elemento de entrada. Feito isso, comparamos com um classificador clássico composto apenas por *perceptrons* clássicos.

Em ambos os casos, usamos o algoritmo de treinamento Adam por 15 épocas para o treinamento das camadas de classificadores adaptadas. No caso da arquitetura com classificadores de perceptrons clássicos treinaremos com *learning rate* de 0.0001 e no caso dos classificadores morfológicos treinamos com 0.001 de *learning rate*. A adaptação da camada de classificadores (originalmente as camadas são representadas na Figura 22) para a avaliação na FMD será colocada como um resultado, assim como qual dessas camadas estará sendo treinada com as configurações desejadas.

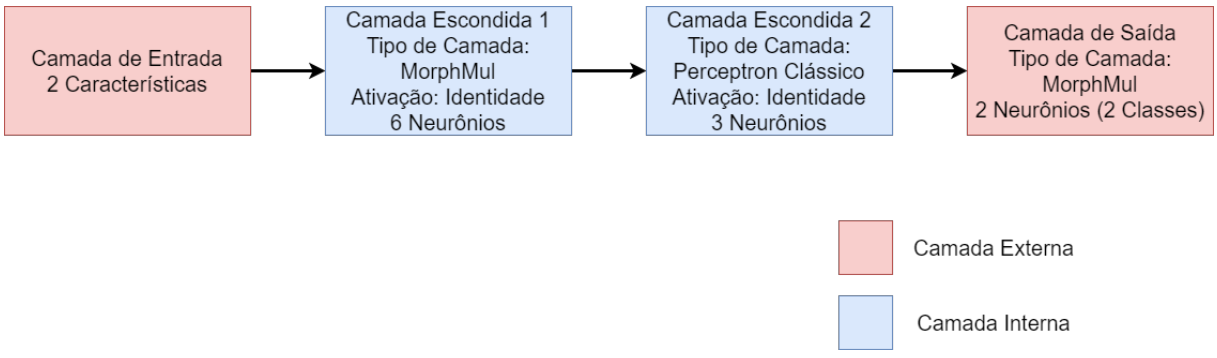
5.4 Resultados

5.4.1 Experimento 1

5.4.1.1 XOR

Para o *dataset* XOR, na Figura 23 apresentamos uma arquitetura hibrida que utiliza a operação MorphMul tanto na primeira camada escondida quanto na camada de saída. Com essa arquitetura e também no método XGBoost, foi possível obter 100% de acurácia tanto no conjunto de treino quanto no de teste. Em comparação, uma SVM obtém 99.5% no conjunto de treino e 99.8% no conjunto de teste, através da regressão logística obtemos 52.35% no conjunto de treino e 48.33% no de teste.

Figura 23 – Arquitetura encontrada empiricamente para o *dataset* XOR.

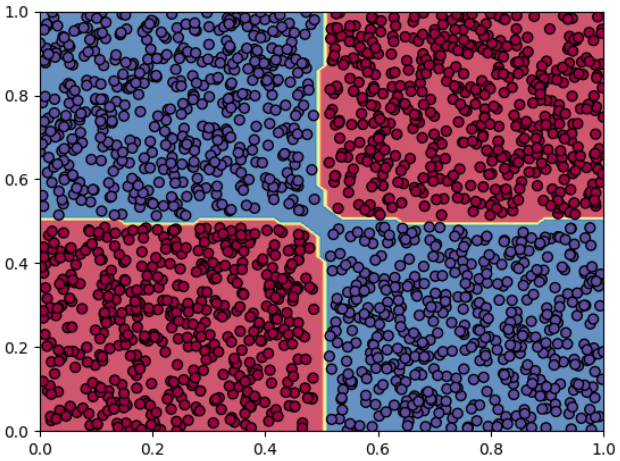


Fonte: Autoria Própria.

Já nas Figuras 24, 25, 26, e 27 temos as fronteiras de decisão geradas pela arquitetura hibrida, SVM, regressão logística e o método XGBoost, respectivamente. É possível notar uma diferença sútil em cada uma das fronteiras, onde a da arquitetura híbrida e do método XGBoost parecem se adaptar melhor aos dados, em comparação a

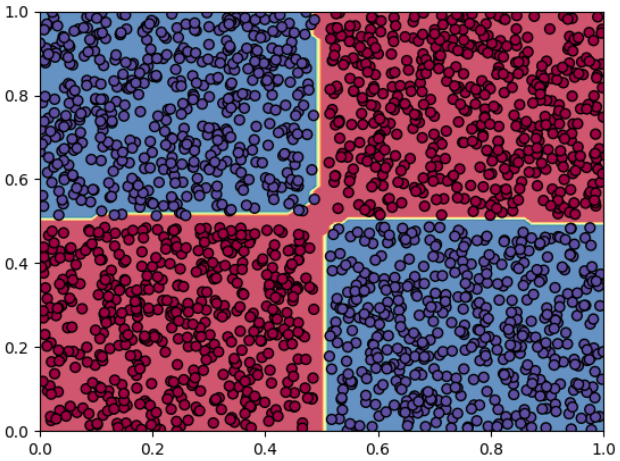
SVM e a regressão logística. Como já foi possível obter 100% de acurácia com a arquitetura híbrida, para esse dataset não será necessário o uso de técnicas de NAS.

Figura 24 – Fronteira de decisão da arquitetura híbrida para o *dataset* XOR.



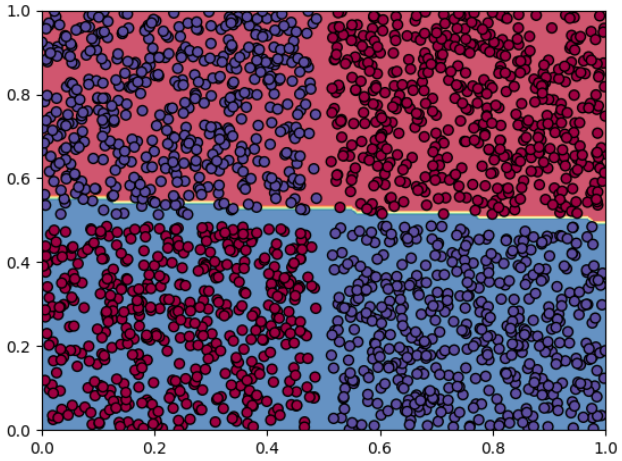
Fonte: Autoria Própria.

Figura 25 – Fronteira de decisão da SVM para o *dataset* XOR.



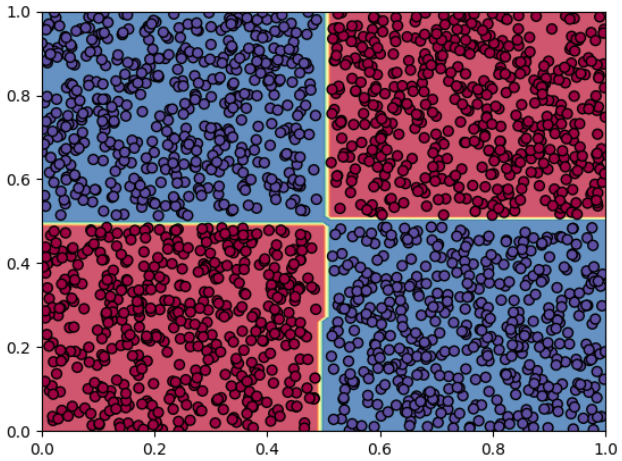
Fonte: Autoria Própria.

Figura 26 – Fronteira de decisão da regressão logística para o *dataset* XOR.



Fonte: Autoria Própria.

Figura 27 – Fronteira de decisão do XGBoost para o *dataset* XOR.



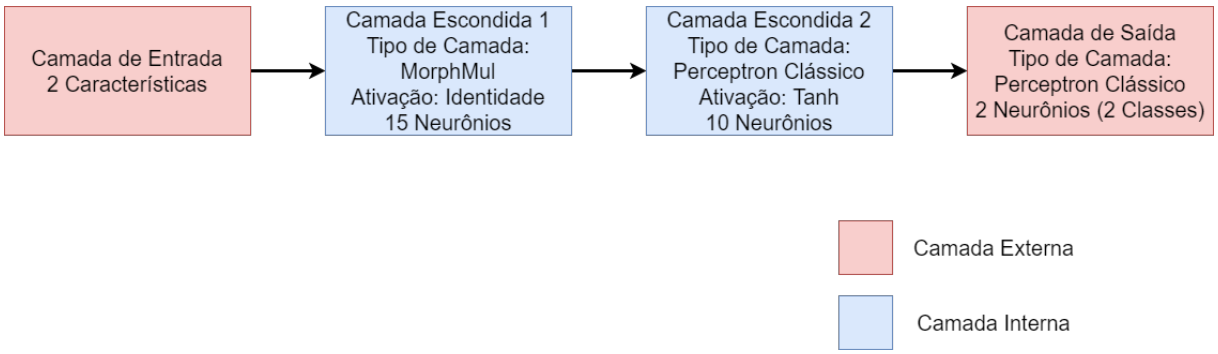
Fonte: Autoria Própria.

5.4.1.2 *Spirals with Loop*

Para o *dataset spirals with loop*, foi criada a arquitetura hibrida presente na Figura 28 que utiliza a função MorphMul apenas na primeira camada sem função de ativação e no restante utiliza perceptrons clássicos como usual. Com essa arquitetura, obtemos 100% tanto no conjunto de treino quanto no de teste. Para comparação, uma SVM atinge apenas 89% no conjunto de treino e 88% no conjunto de teste nesse *dataset*, a regressão logística obtém 61.07% no conjunto de treino e 57.5% no de teste e o método XGBoost obtém 100% no conjunto de treino e 98.88% no de teste. Portanto, nossa

arquitetura hibrida está generalizando melhor os dados que uma SVM e uma regressão logística com desempenho similar ao método XGBoost.

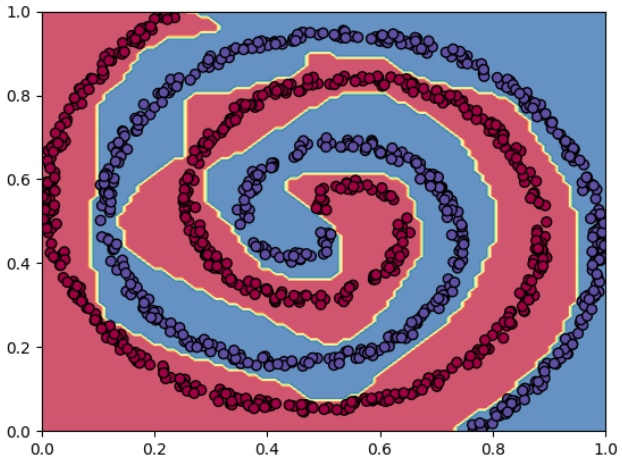
Figura 28 – Arquitetura encontrada empiricamente para o *dataset spirals with loop*.



Fonte: Autoria Própria.

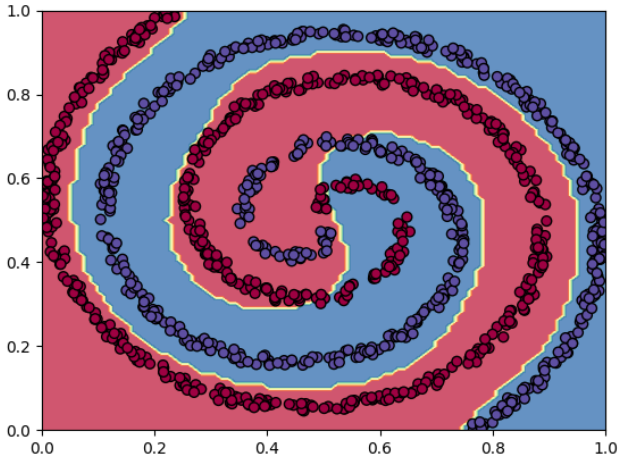
Já na Figuras 29, 30, 31 e 32 temos a comparação entre as fronteiras de decisão geradas pela arquitetura hibrida, SVM, regressão logística e o método XGBoost, respectivamente. Em cada uma das fronteiras, a da arquitetura híbrida e o método XGBoost parecem se adaptar quase que perfeitamente aos dados enquanto a SVM e a regressão logística possuem diversas regiões conflitantes. Como já foi possível obter 100% de acurácia com a arquitetura híbrida, para esse dataset também não será necessário o uso de técnicas de NAS nem a exibição de matrizes de confusão.

Figura 29 – Fronteira de decisão da arquitetura híbrida para o *dataset spirals with loop*.



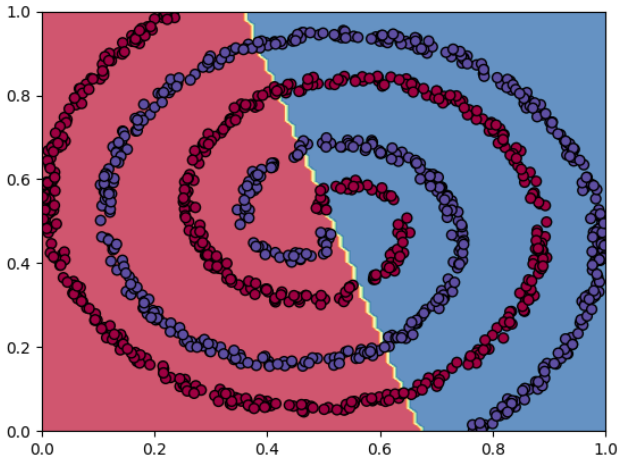
Fonte: Autoria Própria.

Figura 30 – Fronteira de decisão da SVM para o *dataset spirals with loop*.



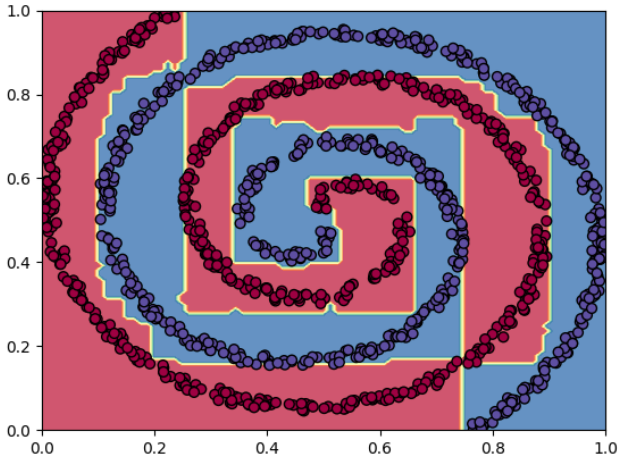
Fonte: Autoria Própria.

Figura 31 – Fronteira de decisão da regressão logística para o *dataset spirals with loop*.



Fonte: Autoria Própria.

Figura 32 – Fronteira de decisão do XGBoost para o *dataset spirals with loop*.

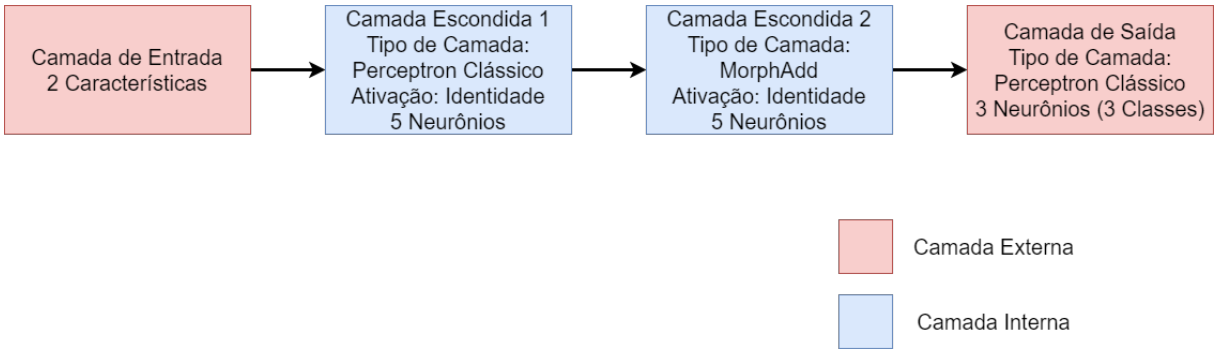


Fonte: Autoria Própria.

5.4.1.3 3 *spirals*

Para o *dataset* 3 *spirals*, a arquitetura apresentada na Figura 33 faz o uso da operação MorphAdd intercalando camadas de perceptrons clássicos, similar a ideia de [40]. Com essa arquitetura foi possível obter acurácias de 99.88% no conjunto de treino e 99.77% no conjunto de teste. Uma SVM obtém 96.1% de acurácia no conjunto de treino e 96.6% no de teste, a regressão logística obtém 52.38% no conjunto de treino e 53.33% no de teste, por último, o método XGBoost obteve 99.88% no conjunto de treino e 99.16% no de teste. Portanto, novamente nossa arquitetura híbrida supera os métodos que estamos usando como comparação.

Figura 33 – Arquitetura encontrada empiricamente para o dataset 3 spirals.

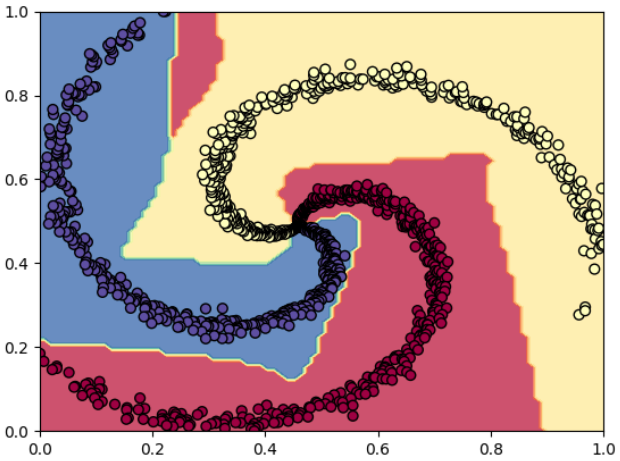


Fonte: Autoria Própria.

Nas Figuras 34, 35, 36 e 37 podemos comparar as fronteiras de decisão encontradas pelos 4 algoritmos de classificação utilizados, enquanto a fronteira da SVM apresenta contornos mais suaves, a fronteira da nossa arquitetura híbrida e do método XGBoost apresentam uma precisão maior próximo ao centro da espiral. Assim como nos

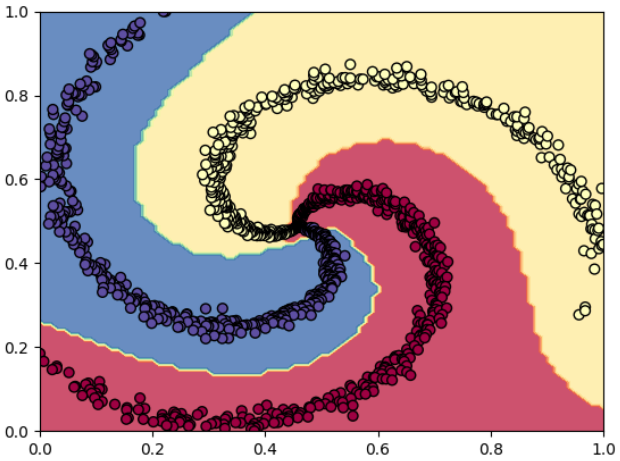
2 *datasets* anteriores, nossa arquitetura híbrida já atingiu valores próximos a 100% de acurácia e, portanto, não usaremos NAS para otimizar essa arquitetura.

Figura 34 – Fronteira de decisão da arquitetura híbrida para a 3 *spirals*.



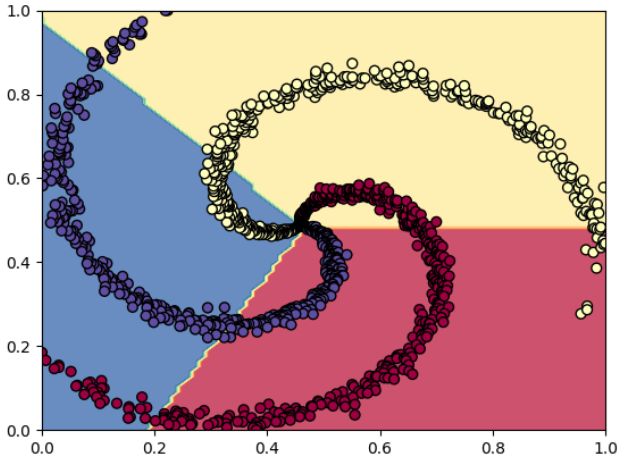
Fonte: Autoria Própria.

Figura 35 – Fronteira de decisão da SVM para a 3 *spirals*.



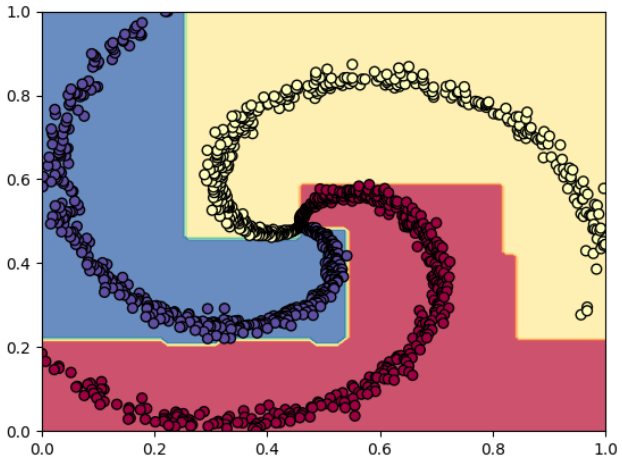
Fonte: Autoria Própria.

Figura 36 – Fronteira de decisão da regressão logística para a 3 *spirals*.



Fonte: Autoria Própria.

Figura 37 – Fronteira de decisão do XGBoost para a 3 *spirals*.



Fonte: Autoria Própria.

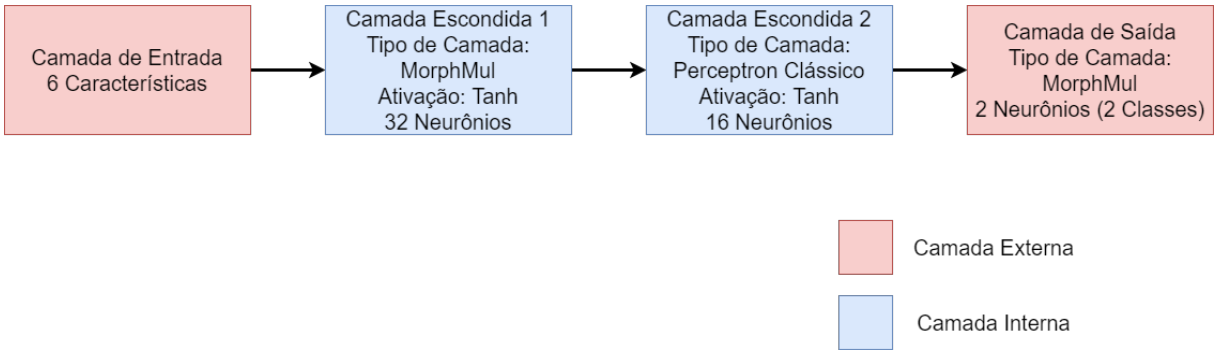
5.4.1.4 *Vertebral Column*

Para o *dataset Vertebral Column*, utilizamos a arquitetura híbrida da Figura 38 para a avaliação da rede. Com essa arquitetura e as configurações citadas na configuração de treinamento, obtemos 85.71% no conjunto de treino e 90.34% no conjunto de teste. Uma SVM obtém 85.71% no conjunto de treino e 90.32% no de teste, a regressão logística obtém 73.73% no conjunto de treino e 74.19% no de teste e, por último, o método XGBoost obtém 100% no conjunto de treino e 88.17% no conjunto de teste.

Portanto, como esse *dataset* ainda necessita de otimizações, NAS será utilizado com BOHB sendo a estratégia de busca e a Tabela 6 sendo o espaço de busca. Como estamos

trabalhando com um *dataset* relativamente pequeno, o custo e tempo computacional para construção e treinamento de cada arquitetura é baixo, permitindo assim usarmos o algoritmo para construir por completo uma arquitetura para esse conjunto de dados. Note que em seu *search space* estamos configurando valores como o *learning rate* que resultará numa diferença do valor especificado anteriormente na tabela de configurações.

Figura 38 – Arquitetura encontrada empiricamente para o *dataset Vertebral Column*.



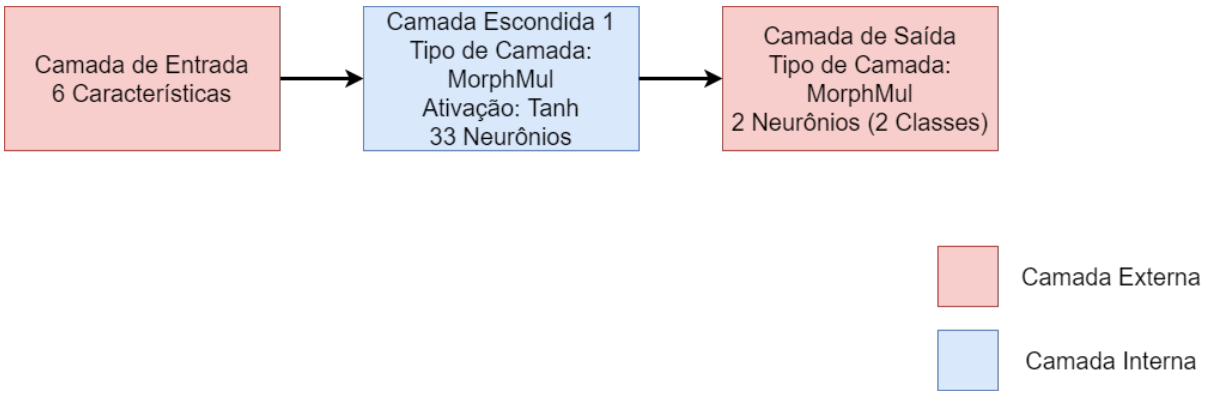
Fonte: Autoria Própria.

Tabela 6 – *Search Space* do *dataset Vertebral Column*

Hiperpârametro	*Search Space*
Número possível de camadas escondidas	{1,2,3}
Número possível de neurônio por camada	Valor inteiro no intervalo uniforme $[6, 64]$
Tipo da Camada	{Perceptron Clássico, MorphMul, MorphAdd}
Ativação da Camada	{Idenditade, Tanh, LeakyReLU, ReLU}
Learning Rate	Valor dentro do intervalo uniforme $[0.0001, 0.01]$

Como resultado, após cerca de 20 horas de treinamento, o algoritmo de NAS avaliou cerca de 1000 arquiteturas e encontrou a melhor configuração de hiperparâmetros como sendo a arquitetura da Figura 39 com *learning rate* igual a 0.00067. Algo interessante dessa arquitetura é que ela é formada totalmente por camadas morfológicas, em particular a MorphMul, e alcança 93.5% de acurácia no conjunto de teste, quase 4% a mais que uma SVM e com uma acurácia 3% superior em comparação a arquitetura criada manualmente.

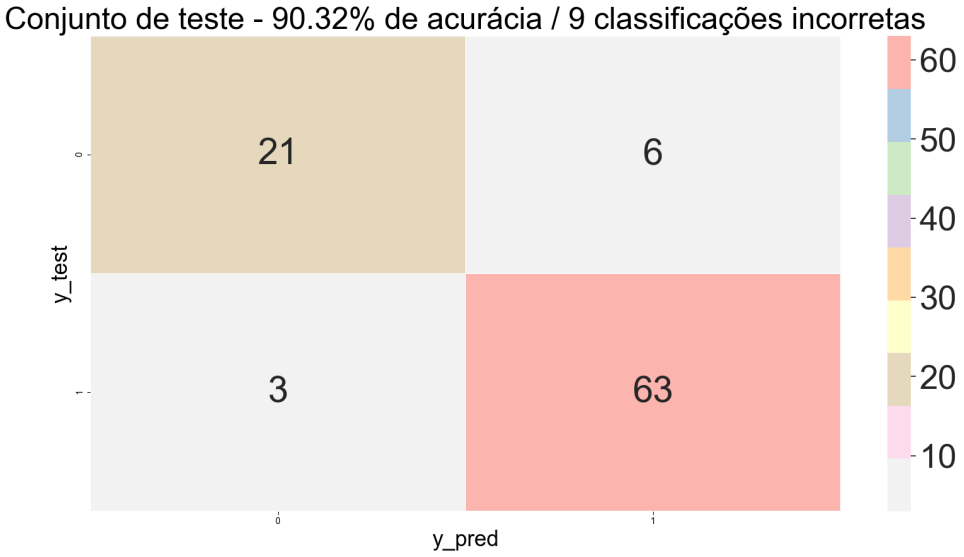
Figura 39 – Arquitetura encontrada com o uso de NAS para o *dataset Vertebral Column*.



Fonte: Autoria Própria.

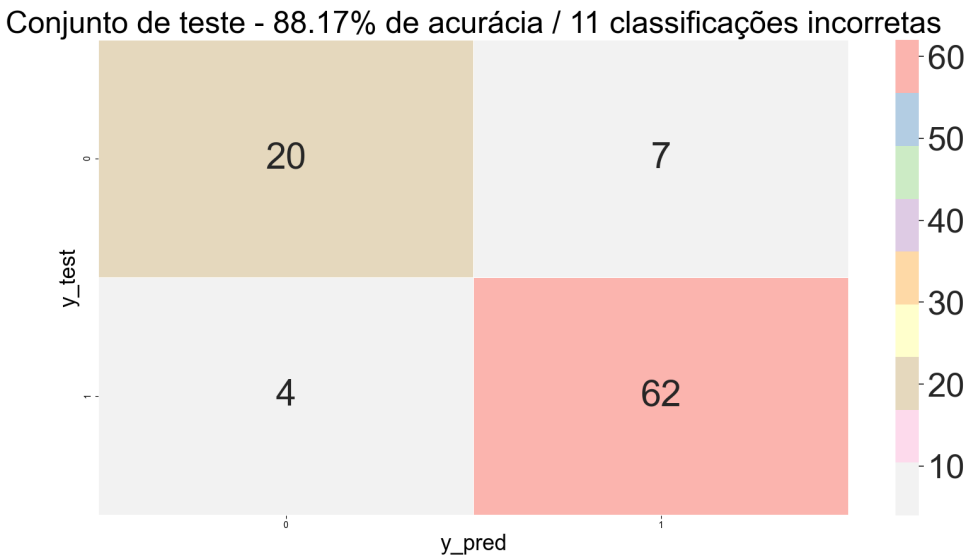
Como esse *dataset* trata de observações sobre colunas vertebrais de pacientes, onde 0 indica uma coluna saúdavel e 1 indica uma coluna com algum tipo de alteração na coluna do paciente, então é interessante observamos o comportamento da matriz de confusão no conjunto de teste. Enquanto nas Figuras 40 e 41 a matriz de confusão da SVM acusou mais pacientes saúdaveis de terem algum problema na coluna e, similarmente, a matriz da XGBoost ocasionou mais erros em ambas as classes. Já na Figura 42, a matriz de coonfusão da arquitetura encontrada por NAS ocasionou menos erros de pacientes saúdaveis serem diagnosticados com algum problema de coluna. Entretanto, no caso da SVM e da arquitetura por NAS, ambos consideraram 2 pacientes com doenças como normais.

Figura 40 – Matriz de confusão da SVM.



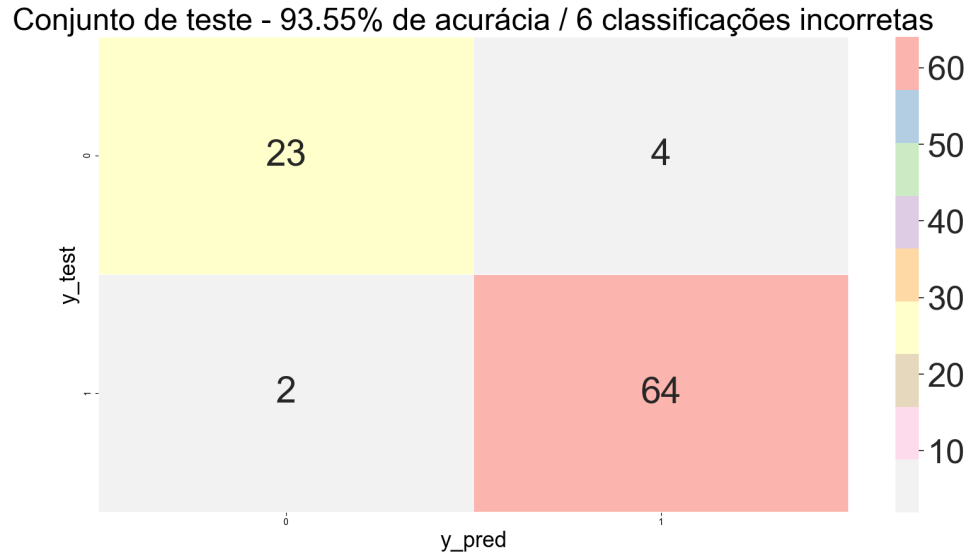
Fonte: Autoria Própria.

Figura 41 – Matriz de confusão da XGBoost.



Fonte: Autoria Própria.

Figura 42 – Matriz de confusão da arquitetura encontrada por NAS.



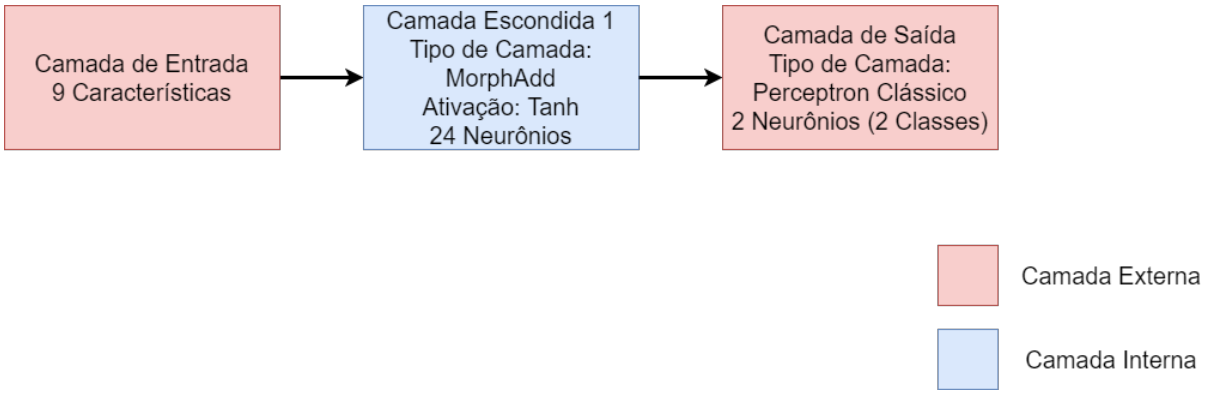
Fonte: Autoria Própria.

5.4.1.5 TicTacToe

Para o *dataset* TicTacToe, a arquitetura criada se encontra na Figura 43. Criada por tentativa e erro, essa arquitetura obteve 100% de acurácia no conjunto de treino e 98.95% no conjunto de teste, para comparação, uma SVM obtém 91.35% no conjunto de treino e 88.50% no de teste, a regressão logística obtém 70.64% de acurácia no conjunto e 67.59% e, por último, o XGBoost obtém 100% de acurácia em ambos os conjuntos.

Portanto, nossa arquitetura híbrida com as operações tem a capacidade de generalização melhor que a de uma SVM e uma regressão logística, mas o algoritmo XGBoost ainda supera o desempenho da nossa rede.

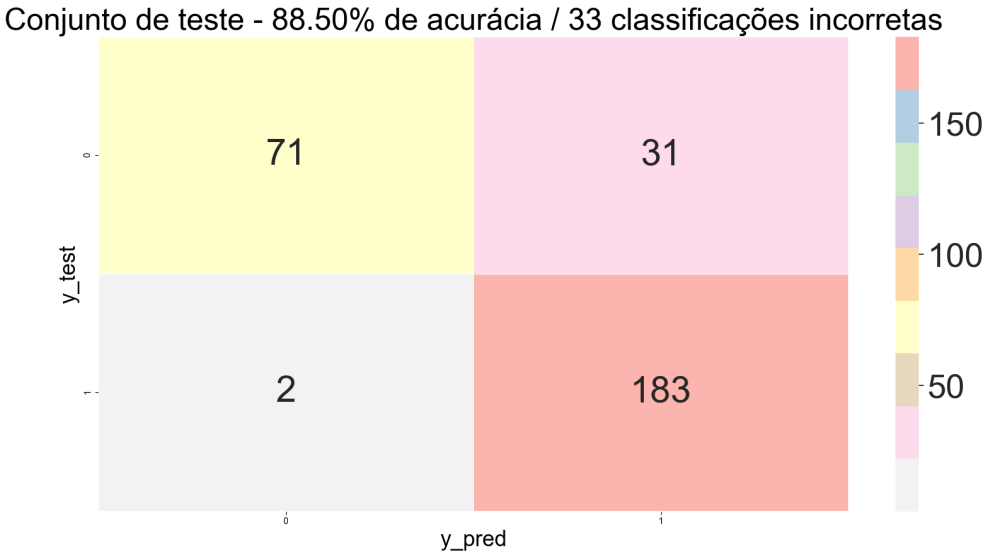
Figura 43 – Arquitetura encontrada empiricamente para o *dataset* TicTacToe.



Fonte: Autoria Própria.

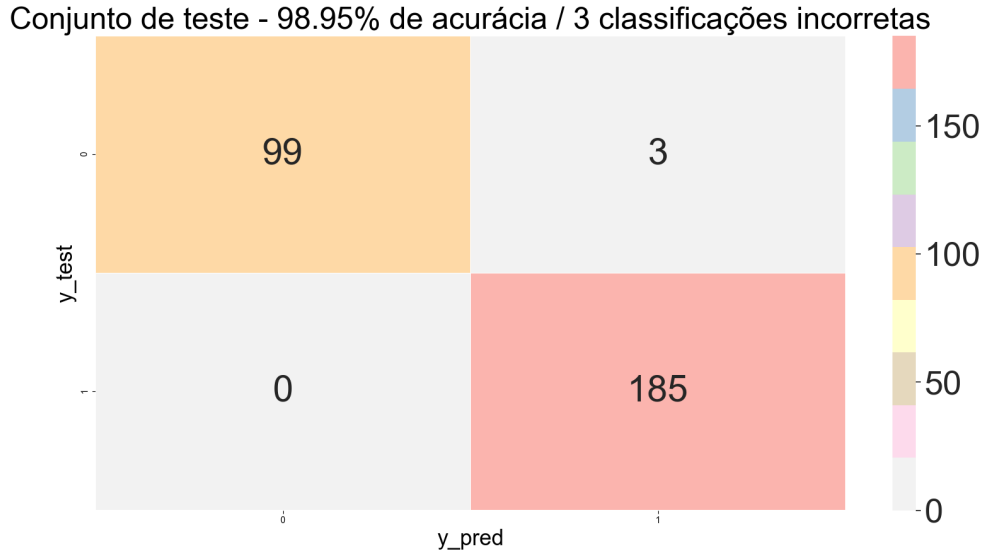
TicTacToe é um *dataset* que simula possíveis partidas desse jogo. Ao analisar a matriz de confusão do conjunto de testes nas Figuras 44 e 45 fica claro que a nossa arquitetura híbrida tem a habilidade de acertar o resultado da maioria dos jogos com mais precisão que uma SVM. Como a quantidade de erro é baixa, não se faz necessário o uso de NAS para otimizar ainda mais a acurácia da arquitetura. Na Figura 46 mostramos a matriz de confusão da XGBoost, superior a ambos os métodos.

Figura 44 – Matriz de confusão da SVM no *dataset* TicTacToe.



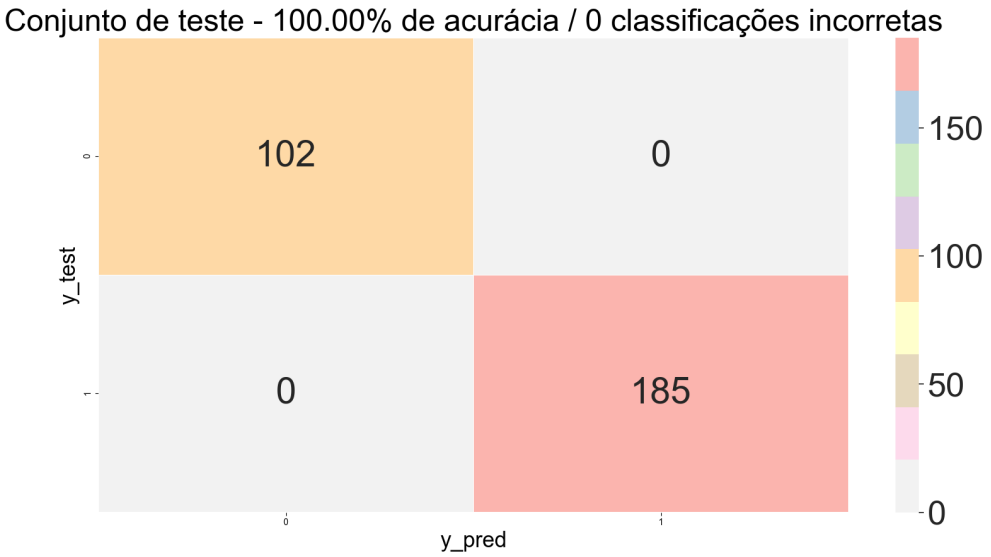
Fonte: Autoria Própria.

Figura 45 – Matriz de confusão da arquitetura encontrada empiricamente no *dataset* TicTacToe.



Fonte: Autoria Própria.

Figura 46 – Matriz de confusão da XGBoost no *dataset* TicTacToe.



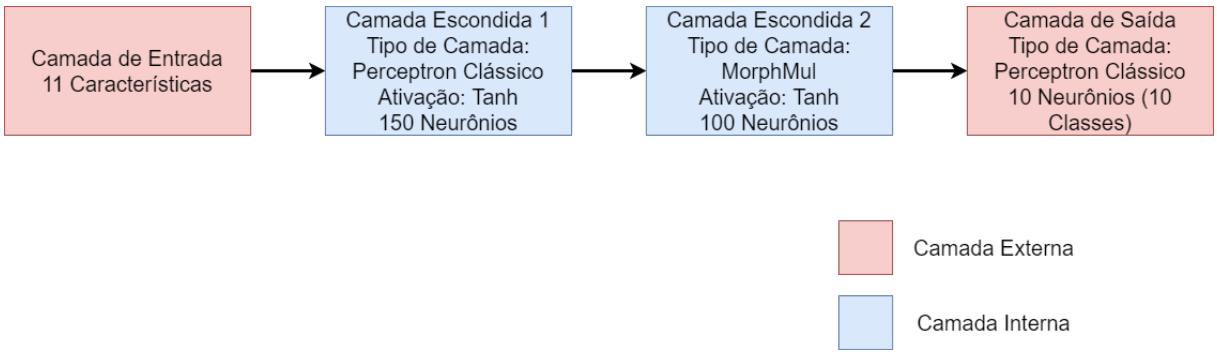
Fonte: Autoria Própria.

5.4.1.6 *Wine Quality* (*White*)

A Figura 47 apresenta a arquitetura para o *dataset Wine Quality* (*White*), que é composta tanto por *perceptrons* clássicos e por uma das operações morfológicas, a Morph-Mul. Com essa arquitetura e a configuração de treinamento especificada anteriormente, a acurácia resultante foi de 100% no conjunto de treino e 61.26% no conjunto de teste,

ou seja, a arquitetura criada empiricamente apresenta um sinal claro de *overfitting*, pois consegue acertar todos os exemplos no conjunto de treino mas não generaliza tão bem no de teste. Já uma SVM é capaz de obter 60.71% de acurácia no conjunto de treino e 59.22% no de teste, uma regressão logística obtém 53.01% no conjunto de treino e 55.13% de teste e o XGBoost obtém 100% no conjunto de treino e 64.87% de teste. Apesar da nossa rede performar melhor que uma SVM e a regressão logística, o XGBoost possui uma acurácia melhor.

Figura 47 – Arquitetura encontrada empiricamente para o *dataset Wine Quality* (*White*).



Fonte: Autoria Própria.

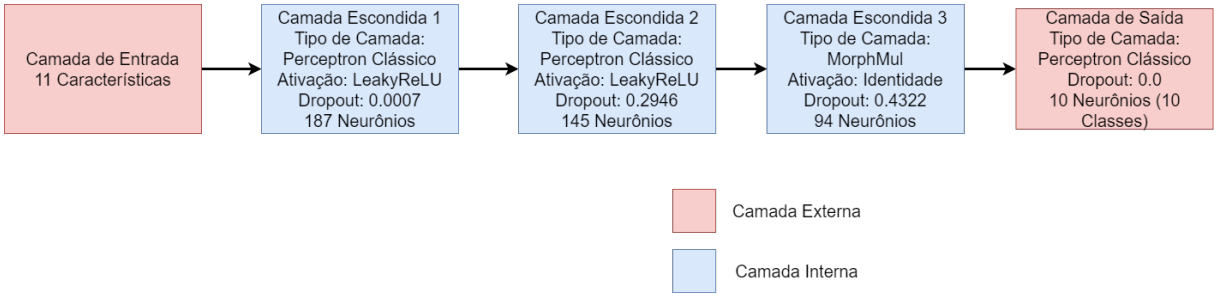
Como talvez o *dataset Wine Quality* (*White*) ainda precise de otimizações, devido ao claro *overfitting*, iremos usar NAS com BOHB para a criação de uma arquitetura e usaremos como espaço de busca os hiperparâmetros descritos na Tabela 7. Note que esse espaço é similar ao do dataset *Vertebral Column* porém com algumas adições como a taxa de dropout por layer que tem o intuito de diminuir o *overfitting* e aumentar a capacidade de generalização da rede.

Tabela 7 – Search Space do *dataset Wine Quality* (*White*)

Hiperparâmetro	*Search Space*
Número possível de camadas escondidas	{1,2,3}
Número possível de neurônio por camada	Valor inteiro no intervalo $[11, 200]$
Tipo de Camada	{*Perceptron* Clássico, MorphMul, MorphAdd }
Ativação da Camada	{Identidade, Tanh, LeakyReLU, ReLU}
Learning Rate	Valor dentro do intervalo uniforme $[0.00001, 0.01]$
Taxa de Dropout por Camada	Valor dentro do intervalo $[0, 1]$

A Figura 48 mostra a arquitetura encontrada com o uso de NAS. Essa arquitetura é composta principalmente por *perceptrons* clássicos, porém também faz uso de uma operação morfológica e obtém 97.31% no conjunto de treino e 63.24% no de teste após o treinamento de cerca de 1200 arquiteturas em 21 horas de execução do algoritmo de NAS.

Figura 48 – Arquitetura encontrada através de NAS para o *dataset Wine Quality* (*White*).

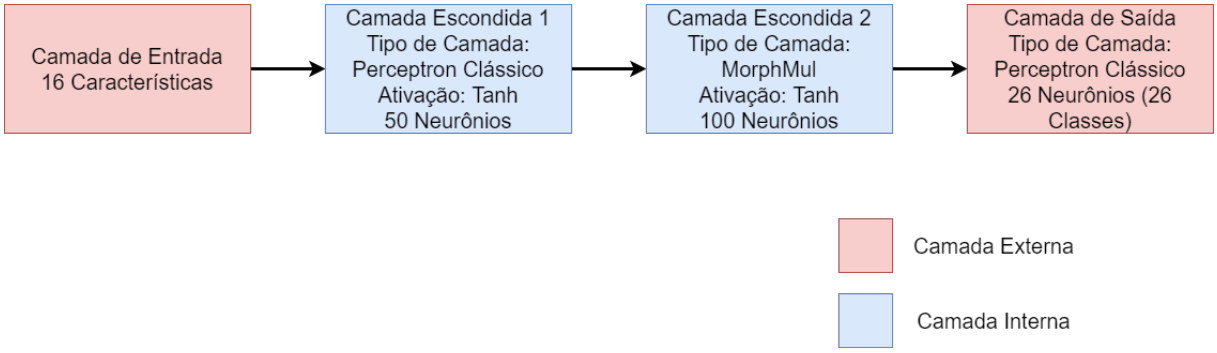


Fonte: Autoria Própria.

5.4.1.7 *Letter Recognition*

Para o *dataset Letter Recognition*, a arquitetura da Figura 49 conseguiu obter uma acurácia de 100% no conjunto de treino e 95.98% no de teste, apresentado uma acurácia satisfatória que, para comparação, uma SVM obtém 95.64% no conjunto de treino e 94.73% no de teste, a regressão logística obtém 74.14% de acurácia no conjunto de treino e 73.25% no de teste e, por último, o XGBoost obtém 100% no conjunto de treino e 96.15% no conjunto de teste. Assim, devido ao tamanho do *dataset* e uma acurácia de teste relativamente alta, não usaremos NAS. Porém, ainda é interessante compararmos as matrizes de confusão.

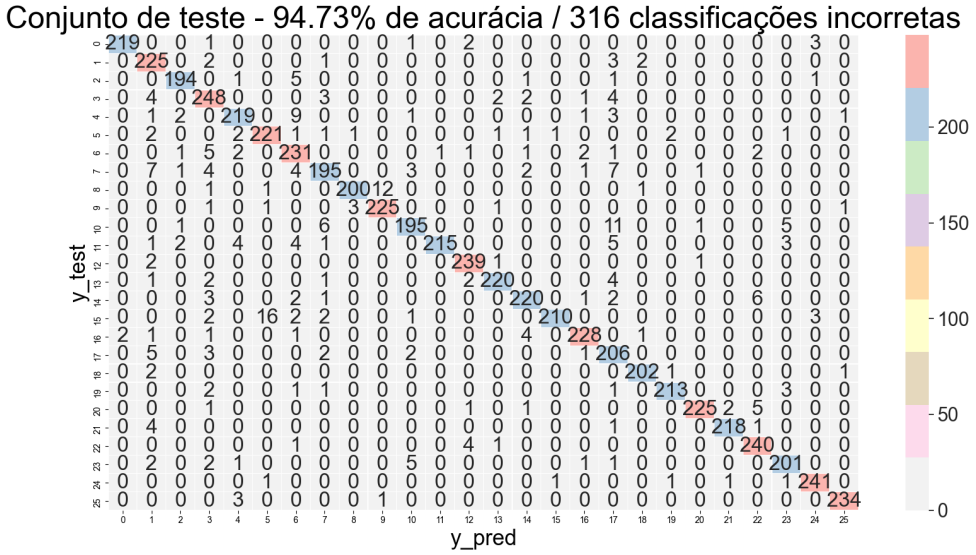
Figura 49 – Arquitetura encontrada empiricamente para o *dataset Letter Recognition*.



Fonte: Autoria Própria.

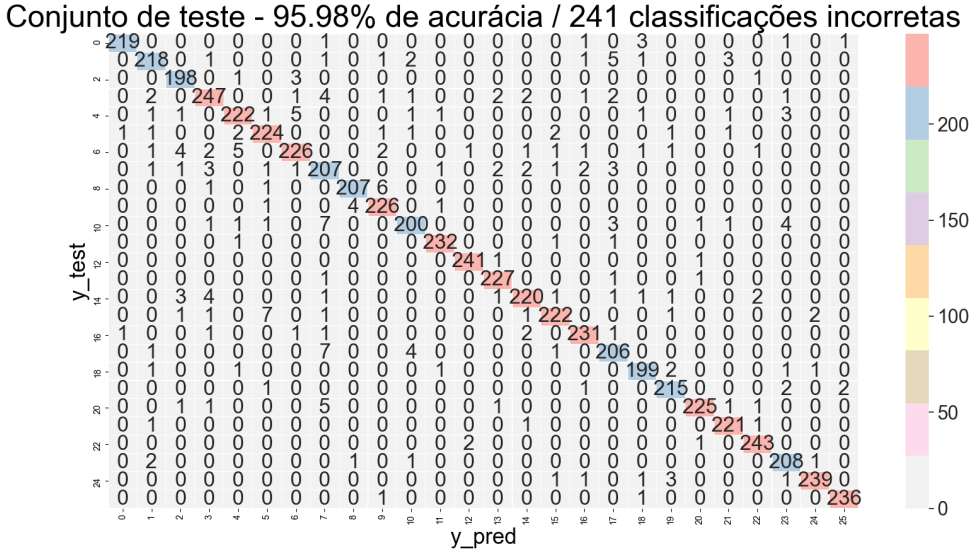
Nas Figuras 60, 61 e 62 vemos as matrizes de confusão dos 3 melhores métodos. Para todos os casos, os erros estão bem distribuídos entre as letras do alfabeto, entretanto a matriz de confusão da XGBoost tem uma quantidade de erros ligereiramente menor.

Figura 50 – Matriz de confusão da SVM no *dataset Letter Recognition*.



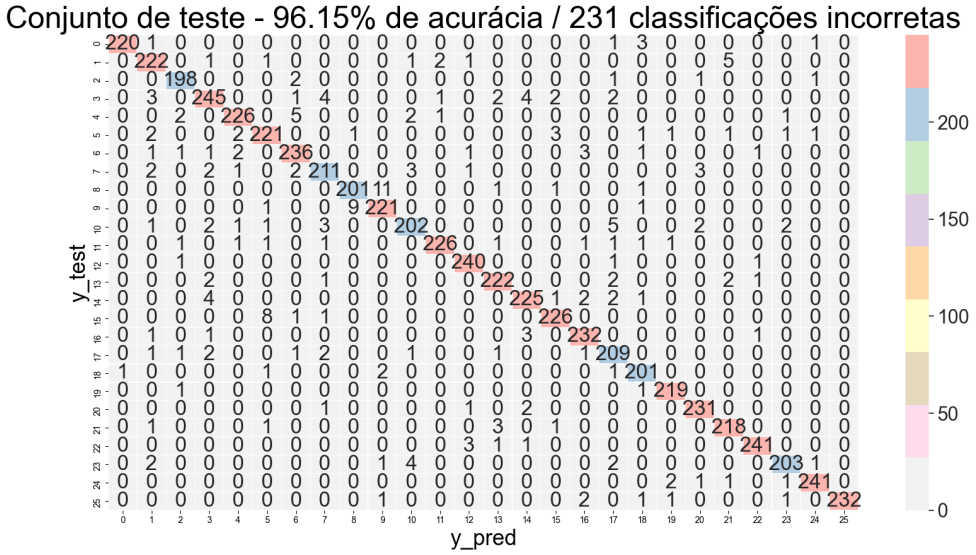
Fonte: Autoria Própria.

Figura 51 – Matriz de confusão da arquitetura encontrada empiricamente no *dataset Letter Recognition*.



Fonte: Autoria Própria.

Figura 52 – Matriz de confusão do XGBoost no *dataset Letter Recognition*.

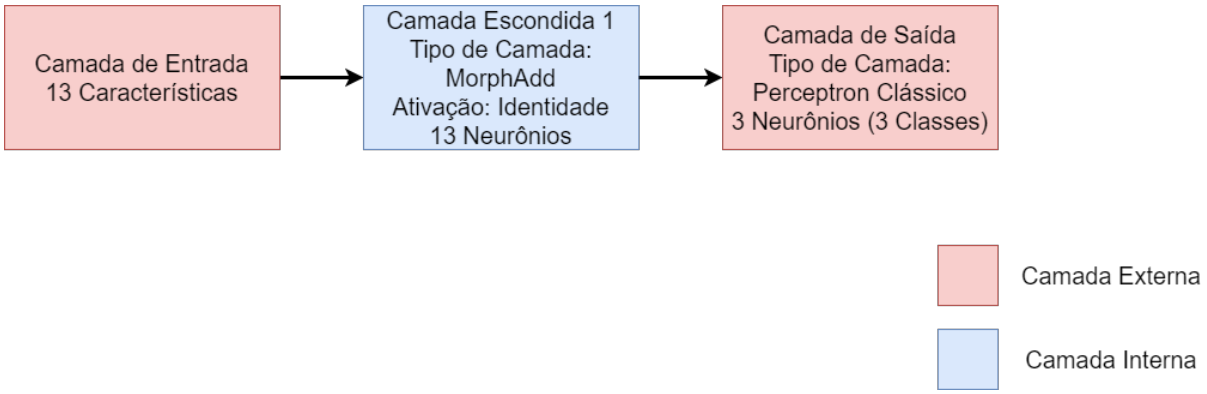


Fonte: Autoria Própria.

5.4.1.8 *Wine*

Para o *dataset Wine*, tanto a arquitetura híbrida encontrada empiricamente na Figura 53 quanto uma SVM conseguem obter 100% de acurácia tanto no conjunto de treino quanto no conjunto de teste, já a regressão logística obtém 98.4% de acurácia no conjunto de treino e 100% no conjunto de teste e a XGBoost obtém 100% e 98.1% nos conjuntos de treino e teste, respectivamente. Assim, não é necessário mais otimizações nesse *dataset*.

Figura 53 – Arquitetura encontrada empiricamente para o *dataset Wine*.



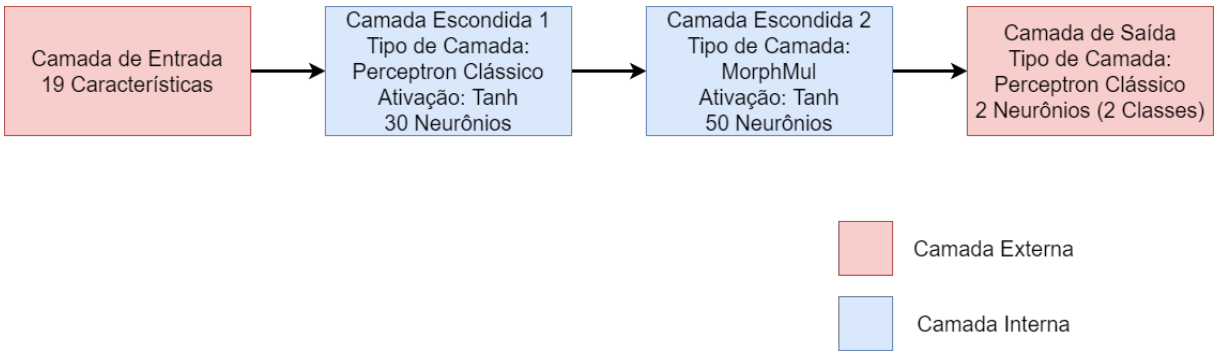
Fonte: Autoria Própria.

5.4.1.9 *Diabetic Retinopathy Debrecen*

Para o *dataset Diabetic Retinopathy Debrecen*, a melhor arquitetura encontrada através de tentativa e erro foi a que se encontra na Figura 54. Com essa arquitetura e

as configurações mencionadas anteriormente obtemos 85.02% de acurácia no conjunto de treino e 75.14% no conjunto de teste. Como comparação, uma SVM obtém 83.85% de acurácia no conjunto de treino e 71.96% no de teste, uma regressão logística obtém 67.82% de acurácia no conjunto de treino e 66.18% no de teste e, por último, o XGBoost atinge 100% e 69% de acurácia no conjunto de treino e teste, respectivamente. Assim, nossa arquitetura construída através de tentativa e erro já supera os outros 3 métodos. Entretanto, ainda é existente uma margem para otimização na acurácia da arquitetura e usaremos NAS para a otimizar.

Figura 54 – Arquitetura encontrada empiricamente para o *dataset Diabetic Retinopathy Debrecen*.



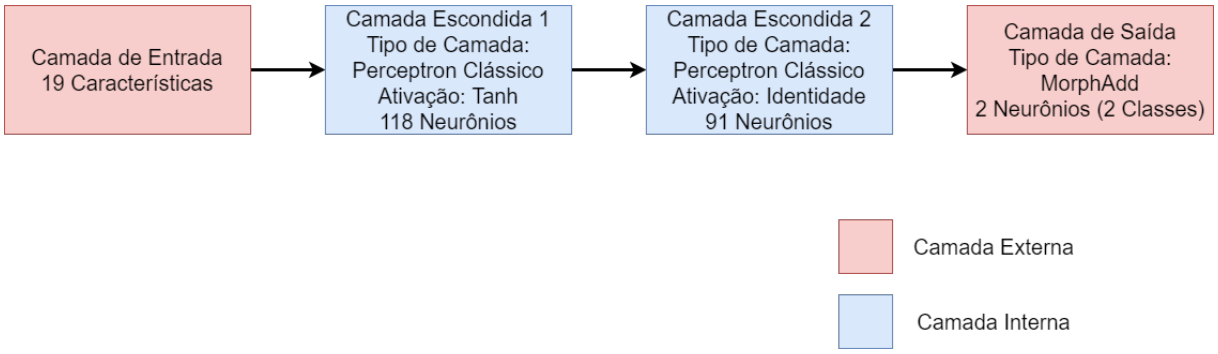
Fonte: Autoria Própria.

Ao utilizar BOHB como estratégia de busca para NAS e a Tabela 8 como seu espaço de busca, a Figura 55 é a arquitetura ótima resultante de NAS, que selecionou um *learning rate* de 0.0006 para seu treinamento. Com essas configurações, a rede obtém 81.62% no conjunto de treino e 76.87% no de teste, uma acurácia ligeiramente menor no conjunto de treino porém um pouco maior no conjunto de teste.

Tabela 8 – *Search Space* do *dataset Diabetic Retinopathy Debrecen*

Hiperpârametro	*Search Space*
Número possível de camadas escondidas	{1,2,3}
Número possível de neurônio por camada	Valor inteiro no intervalo uniforme $[19, 175]$
Tipo da Camada	{Perceptron Clássico, MorphMul, MorphAdd}
Ativação da Camada	{Idenditade, Tanh, LeakyReLU, ReLU}
Learning Rate	Valor dentro do intervalo uniforme $[0.0001, 0.01]$

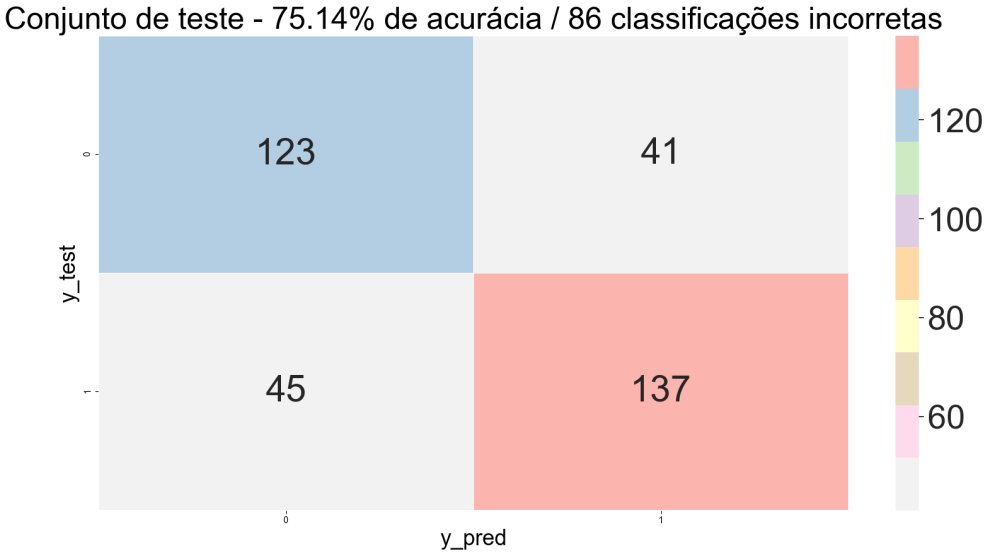
Figura 55 – Arquitetura encontrada através de NAS para o*dataset Diabetic Retinopathy Debrecen*.



Fonte: Autoria Própria.

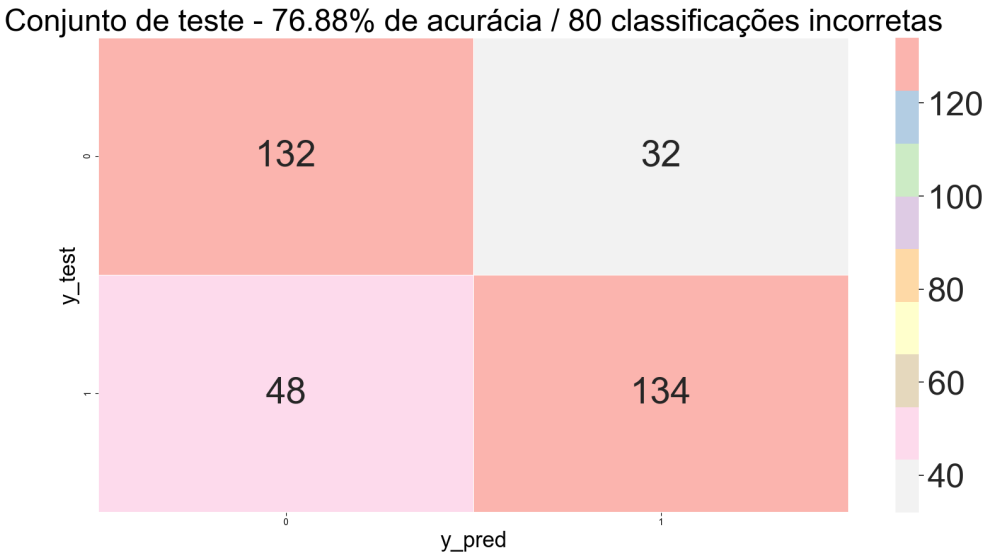
Nas Figuras 56 e 57 vemos as matrizes de confusão para cada arquitetura. Enquanto a arquitetura encontrada empiricamente comete menos falsos negativos, a arquitetura encontrada por NAS comete menos erros falsos positivos.

Figura 56 – Matriz de confusão da arquitetura encontrada empiricamente no *dataset Diabetic Retinopathy Debrecen*.



Fonte: Autoria Própria.

Figura 57 – Matriz de confusão da arquitetura encontrada por NAS no *dataset Diabetic Retinopathy Debrecen*.



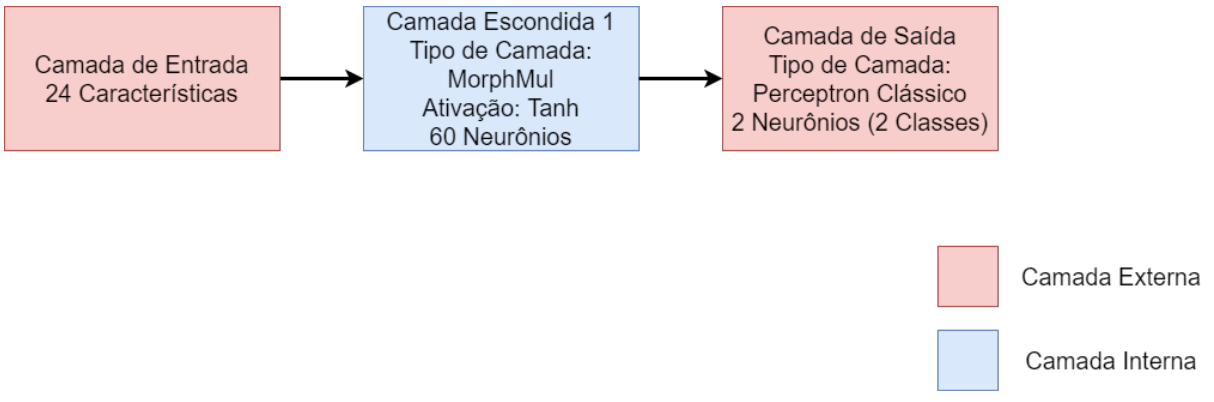
Fonte: Autoria Própria.

5.4.1.10 *Default of Credit Cards Clients*

No *dataset Default of Credit Cards Clients*, apresentamos na Figura 58 a arquitetura híbrida que usa camadas morfológicas e como classificador, uma camada de perceptrons clássicos. Essa arquitetura obtém 82.6% de acurácia no conjunto de treino e 81.15% no conjunto de teste. Uma SVM, como comparação, obtém 82.7% de acurácia no conjunto de treino e 81.58% no conjunto de teste, a regressão logística obtém 77.92% de acurácia no conjunto de treino e 77.76% no de teste e, por último, o XGBoost obtém 99.74% e 80.13% no conjunto de treino e teste, respectivamente.

Assim, usaremos NAS com o *search space* na Tabela 9 para otimizar esse conjunto.

Figura 58 – Arquitetura encontrada empiricamente para o *dataset Default of Credit Cards Clients*.



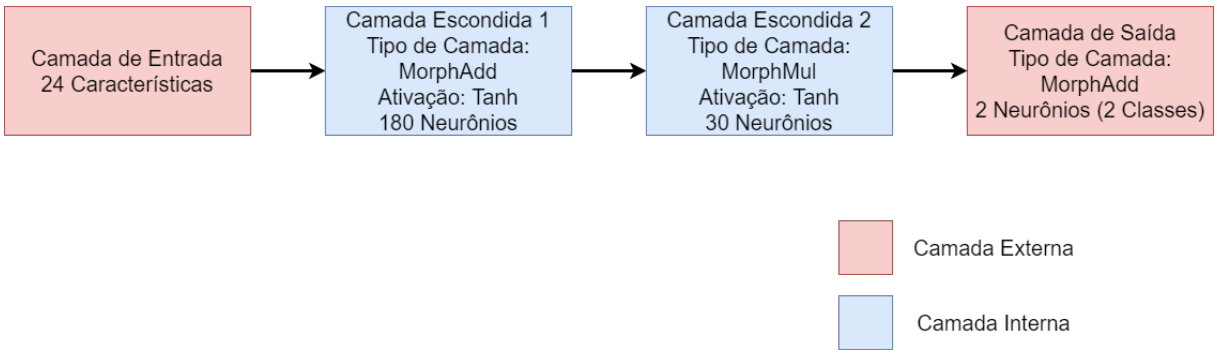
Fonte: Autoria Própria.

Tabela 9 – *Search Space* do *dataset Default of Credit Cards Clients*

Hiperpârametro	*Search Space*
Número possível de camadas escondidas	{1,2,3}
Número possível de neurônio por camada	{30,60,90,120,160,180,200}
Tipo da Camada	{Perceptron Clássico, MorphMul, MorphAdd}
Ativação da Camada	{Idenditade, Tanh, LeakyReLU, ReLU}
Learning Rate	Valor dentro do intervalo uniforme $[0.00001, 0.01]$

Através de NAS, a melhora não foi significativa. A Figura 59 mostra a arquitetura morfológica encontrada para esse dataset, ela obteve $82,3\%$ de acurácia no conjunto de treino e $82,03\%$ no de teste. Uma parte relevante da arquitetura é que ela é formada completamente por operações morfológicas.

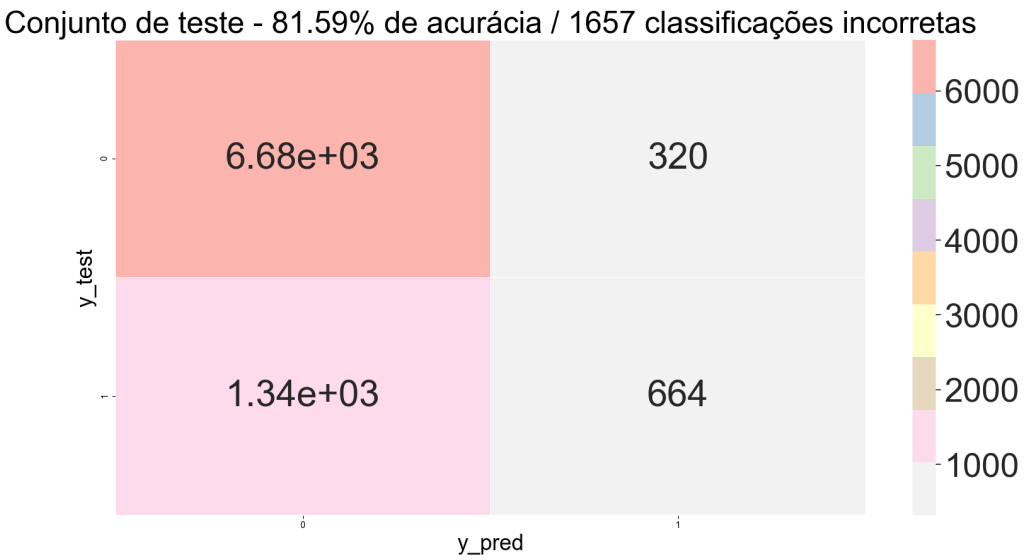
Figura 59 – Arquitetura encontrada com NAS para o *dataset Default of Credit Cards Clients*.



Fonte: Autoria Própria.

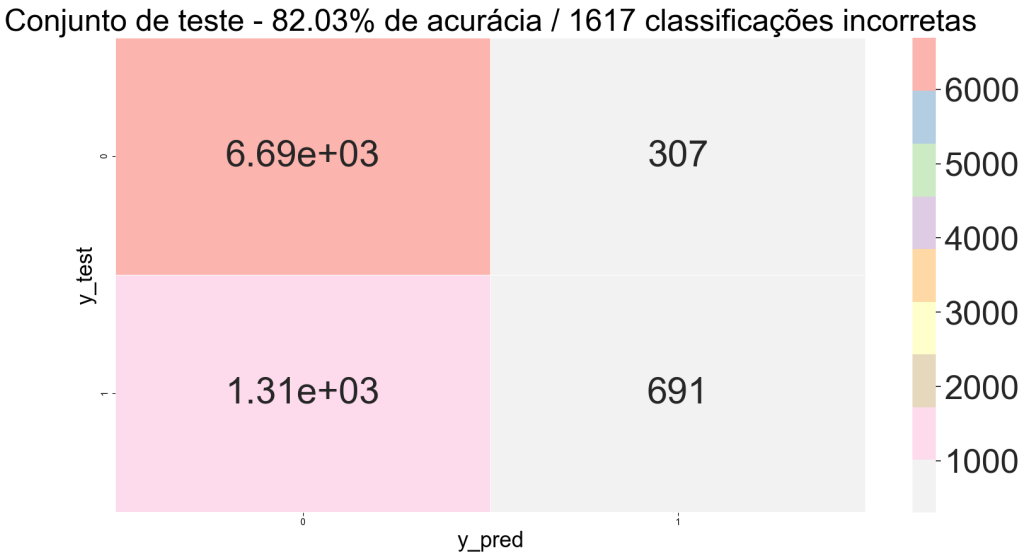
Já as Figuras 60, 61 e 62 mostram as matrizes de confusão para a SVM, a arquitetura de NAS e o método XGBoost, respectivamente. Em geral, a arquitetura de NAS erra menos em ambas as classes com 40 classificações erradas a menos em comparação a SVM e 171 a menos em relação ao XGBoost.

Figura 60 – Matriz de confusão da SVM para o *dataset Default of Credit Cards Clients*.



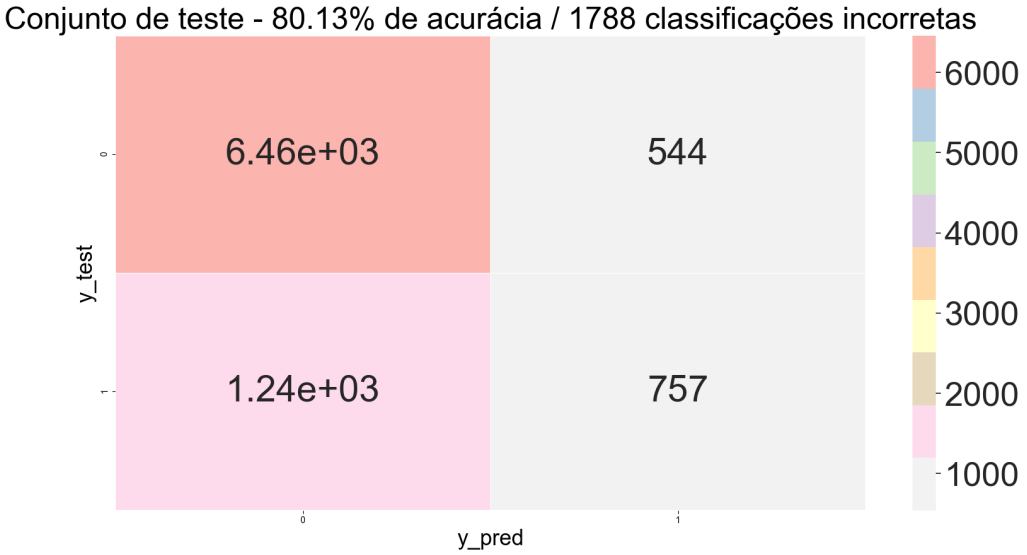
Fonte: Autoria Própria.

Figura 61 – Matriz de confusão da arquitetura encontrada por NAS no *dataset Default of Credit Cards Clients*.



Fonte: Autoria Própria.

Figura 62 – Matriz de confusão da arquitetura encontrada por XGBoost no *dataset Default of Credit Cards Clients*.

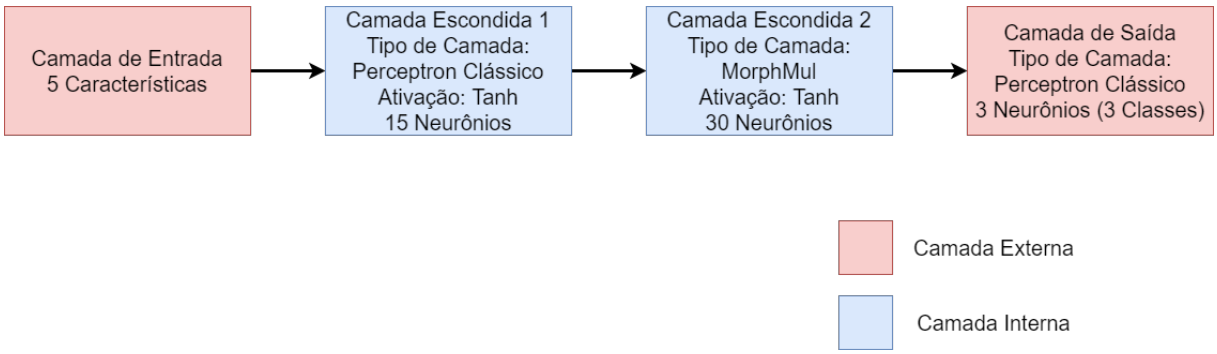


Fonte: Autoria Própria.

5.4.1.11 *Teaching Assistant Evaluation*

Para o *dataset Teaching Assistant Evaluation*, foi feita a arquitetura híbrida na Figura 63. Com essa arquitetura e a configuração para treinamento apresentada na Tabela 2, a rede neural obteve uma acurácia de 98.11% no conjunto de treino e 57.77% no de teste, comportamento que indica um *overfitting*. Como comparação, uma SVM obtém 75.47% de acurácia no conjunto de treino e 57.77% no de teste, a regressão logística obtém 54.71% de acurácia no conjunto de treino e 64.44% no de teste e, por último, o método XGBoost obtém 98.11% e 62.22% de acurácia no conjunto de treino e teste, respectivamente. Dessa vez, os resultados para o conjunto de teste foram iguais para a arquitetura construída empiricamente e para a SVM sendo esse resultado inferior aos métodos XGBoost e de regressão logística. Assim, nesse *dataset* optamos por utilizar um algoritmo de NAS.

Figura 63 – Arquitetura encontrada empiricamente para o *dataset Teaching Assistant Evaluation*.



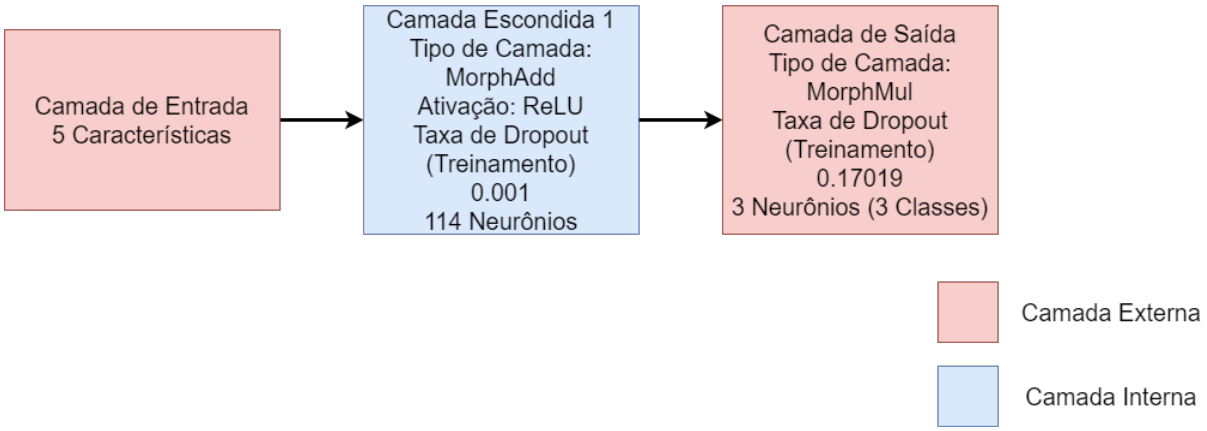
Fonte: Autoria Própria.

Para o algoritmo de NAS foi utilizado BOHB como estratégia de busca e a Tabela 10 como o espaço de busca. Durante 23 horas o algoritmo de NAS buscou mais de 1200 arquiteturas e, com essas configurações, apresentamos na Figura 64 a arquitetura ótima encontrada por NAS e treinada por 750 épocas com *learning rate* de 0.0053. Essa arquitetura obteve 60% de acurácia no conjunto de treino e 71% no de teste e, um detalhe importante, é que sua arquitetura é composta inteiramente por operações morfológicas e elimina o *overfitting* presente na primeira arquitetura criada por tentativa e erro, além de também melhorar a acurácia em cerca de 9%.

Tabela 10 – *Search Space* do *dataset Teaching Assistant Evaluation*

Hiperparâmetro	*Search Space*
Número possível de camadas escondidas	{1,2,3}
Número possível de neurônio por camada	Valor inteiro no intervalo $[5, 250]$
Tipo de Camada	{Perceptron Clássico, MorphMul, MorphAdd}
Ativação da Camada	{Identidade, Tanh, LeakyReLU, ReLU}
Learning Rate	Valor dentro do intervalo uniforme $[0.0001, 0.01]$
Taxa de *Dropout* por Camada	Valor dentro do intervalo $[0, 1]$

Figura 64 – Arquitetura encontrada empiricamente para o *dataset Teaching Assistant Evaluation*.



Fonte: Autoria Própria.

5.4.2 Compilado dos Resultados para o Experimento 1 - A

Colocamos na Tabela 11 os resultados para o primeiro experimento. Fica claro que na maioria dos *datasets*, as operações morfológicas obtiveram um desempenho tão bom quanto os outros métodos avaliados, além disso, o uso de NAS aperfeiçoou a acurácia nos casos em que foi utilizado BOHB. Porém, no caso dos *datasets Wine Quality* (*White*) e *Default of Credit Cards Clients*, esse aperfeiçoamento foi menor talvez devido ao tamanho de cada dataset, a quantidade de arquiteturas treinadas e o *search space* escolhido.

Tabela 11 – Resumo dos resultados apresentados no Experimento 1 - A

	Acurácia Conjunto de Teste				
	Arquitetura Morfológica (Manual)	Regressão Logística	XGBoost	SVM	Arquitetura Morfológica (NAS)
XOR	100.00%	48.33%	100.00%	99.8%	N/A
Spirals with loops	100.00%	57.50%	98.88%	88.00%	N/A
3 spirals	99.88%	53.33%	99.16%	96.6%	N/A
Vertebral Column	90.34%	74.19%	88.17%	83.00%	93.50%
TicTacToe	98.95%	67.59%	100%	88.50%	N/A
Wine Quality (White)	61.26%	55.13%	64.87%	59.22%	63.24%
Letter Recognition	95.98%	73.25%	96.15%	94.73%	N/A
Wine	100.00%	100.00%	98.10%	100.00%	N/A
Diabetic Retinopathy Debrecen	75.14%	66.18%	69.05%	71.96%	77.45%
Default of Credit Cards Clients	81.15%	77.76%	80.13%	81.58%	82.03%
Teaching Assistant Evaluation	57.77%	64.44%	62.22%	57.77%	71.00%

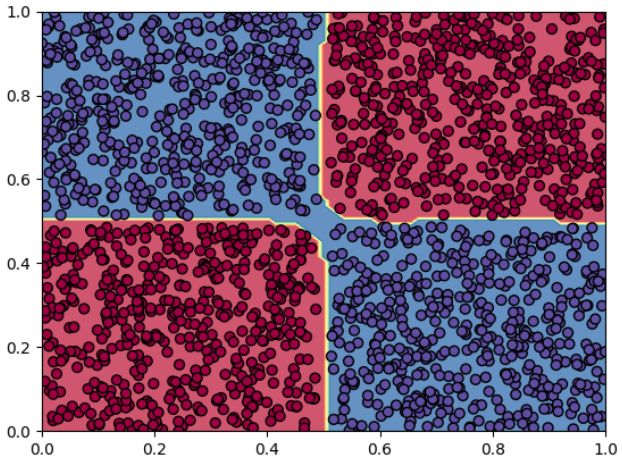
5.4.3 Experimento 1 - B

A partir dos resultados apresentados no Experimento 1 - A e suas arquiteturas, concluímos que a MorphMul foi incluída na arquitetura de alguns *datasets*. São eles: XOR, *Spirals with loop*, *Vertebral Column*, *Wine Quality (White)*, *Letter Recognition*, *Default of Credit Cards Clients* e *Teaching Assistant Evaluation*.

5.4.3.1 XOR

Para o *dataset* sintético XOR, utilizando uma arquitetura similar a apresentada na Figura 23, porém com a operação PositiveMorphMul no lugar da camada MorphMul, a arquitetura foi capaz de obter 100% tanto no conjutno de treino quanto de teste. Abaixo, apresentamos a fronteira de decisão na Figura 65.

Figura 65 – Fronteira de decisão com arquitetura ajustada para o uso da PositiveMorphMul para o *dataset* XOR.

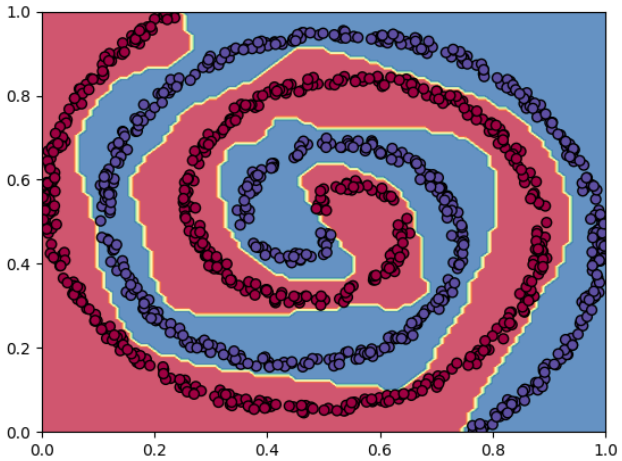


Fonte: Autoria Própria.

5.4.3.2 *Spirals with loop*

Para o *dataset* sintético *Spirals with loop*, utilizando uma arquitetura similar a apresentada na Figura 28, porém com a operação PositiveMorphMul no lugar da camada MorphMul, a arquitetura foi capaz de obter 100% tanto no conjutno de treino quanto de teste. Abaixo, apresentamos a fronteira de decisão na Figura 66.

Figura 66 – Fronteira de decisão com arquitetura ajustada para o uso da PositiveMorphMul para o *dataset Spirals with loop*.

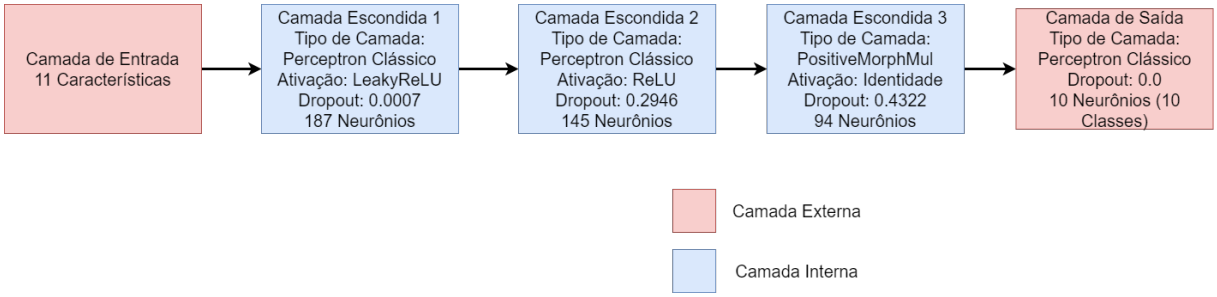


Fonte: Autoria Própria.

5.4.3.3 *Wine Quality* (*White*)

Com uma estrutura parecida com a arquitetura proposta através de NAS para o *dataset Wine Quality* (*White*), a arquitetura apresentada na Figura 67, que faz o uso da PositiveMorphMul, obteve 99.76% e 61.81% no conjunto de treino e teste, respectivamente. Com essa acurácia, podemos afirmar que a arquitetura apresentou *overfitting*.

Figura 67 – Arquitetura ajustada para o PositiveMorphMul *dataset Wine Quality* (*White*).

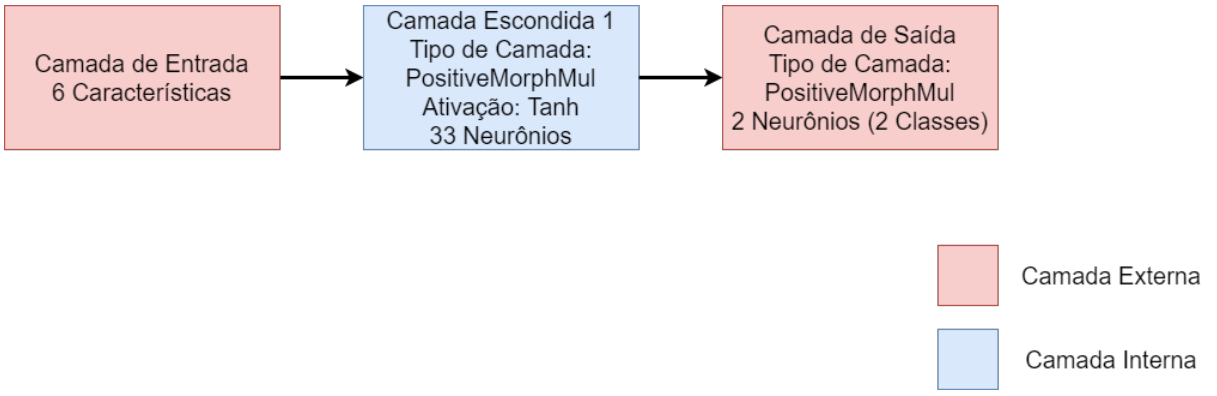


Fonte: Autoria Própria.

5.4.3.4 *Vertebral Column*

Com uma estrutura parecida com a arquitetura proposta através de NAS para o *dataset Vertebral Column*, a arquitetura apresentada na Figura 68, que faz o uso da PositiveMorphMul, obteve 77.41% e 79.56% no conjunto de treino e teste, respectivamente.

Figura 68 – Arquitetura ajustado para o *dataset Vertebral Column*.

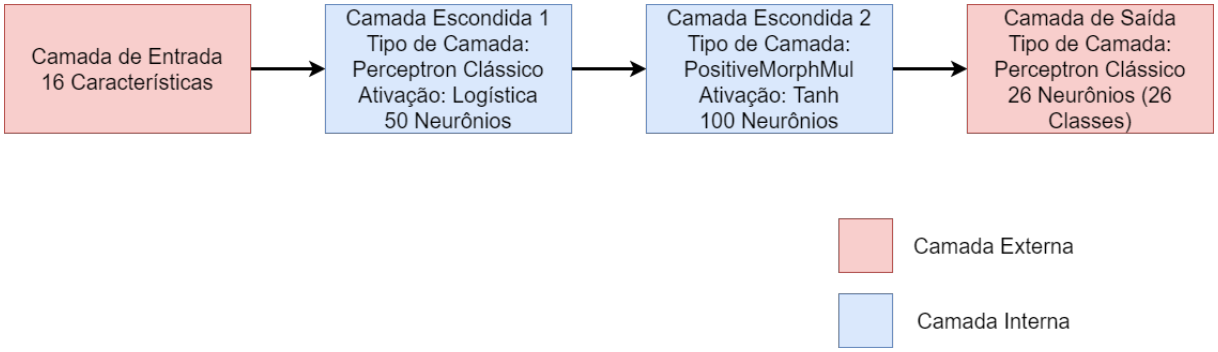


Fonte: Autoria Própria.

5.4.3.5 *Letter Recognition*

Com uma estrutura parecida com a arquitetura proposta para o *dataset Letter Recognition*, a arquitetura apresentada na Figura 69, que faz o uso da PositiveMorphMul, obteve 71.39% e 66.58% no conjunto de treino e teste, respectivamente. Com essa acurácia, podemos afirmar que a arquitetura não se adequou tão bem ao *dataset*.

Figura 69 – Arquitetura ajustada para o *dataset Letter Recognition*.

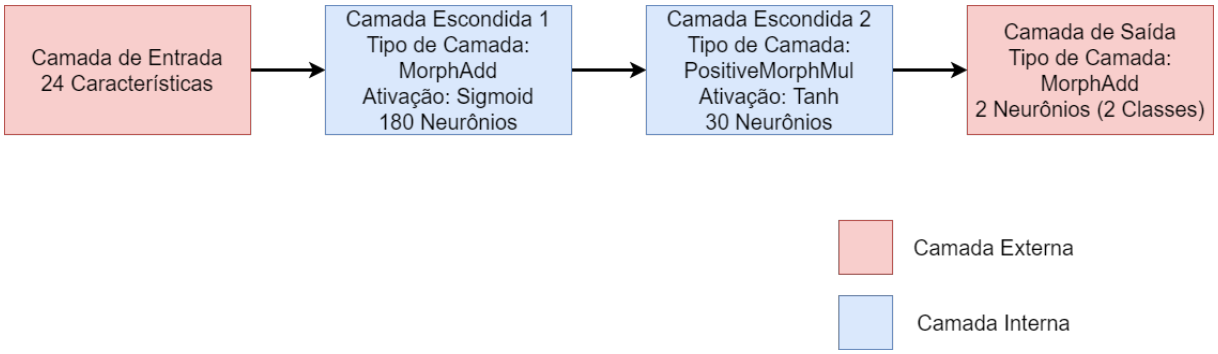


Fonte: Autoria Própria.

5.4.3.6 *Default of Credit Cards Clients*

Com uma estrutura parecida com a arquitetura proposta através de NAS para o *dataset Default of Credit Cards Clients*, a arquitetura apresentada na Figura 70, que faz o uso da PositiveMorphMul, obteve 82.16% e 81.66% no conjunto de treino e teste, respectivamente.

Figura 70 – Arquitetura ajustada para o *dataset Default of Credit Cards Clients*.

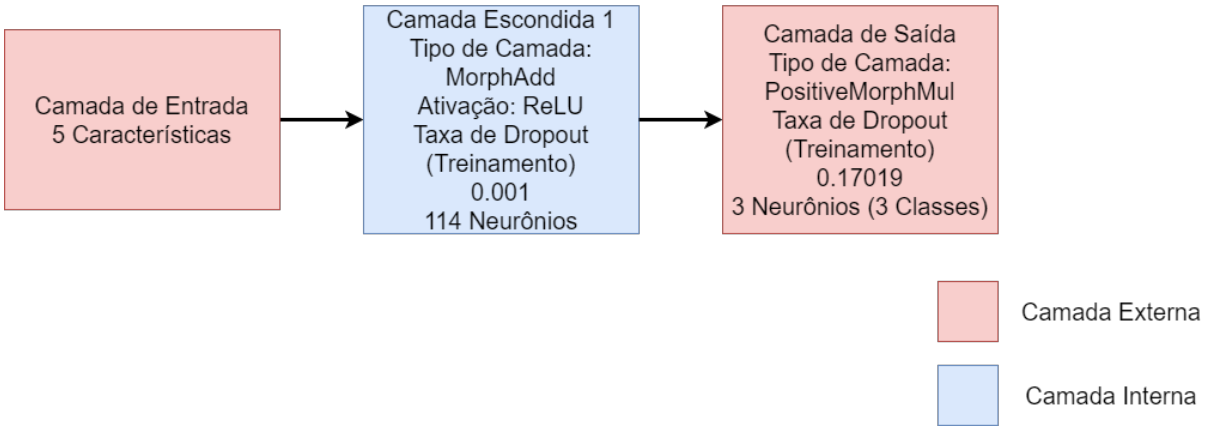


Fonte: Autoria Própria.

5.4.3.7 *Teaching Assistant Evaluation*

Com uma estrutura parecida com a arquitetura proposta através de NAS para o *dataset Teaching Assistant Evaluation*, a arquitetura apresentada na Figura 71, que faz o uso da PositiveMorphMul, obteve 61.05% e 55.55% no conjunto de treino e teste, respectivamente.

Figura 71 – Arquitetura ajustada para o *dataset Teaching Assistant Evaluation*.



Fonte: Autoria Própria.

5.4.4 Comparação entre as operações

Nessa seção comparamos os resultados entre as operações MorphMul e a PositiveMorphMul, que respeita as operações morfológicas. A Tabela 12 mostra que a MorphMul se adaptou melhor a diferentes *datasets*, entretanto um estudo mais aprofundado pode gerar resultados melhores.

Tabela 12 – Comparação entre as acurácias das operações

	Acurácia de Teste	
	MorphMul	PositiveMorphMul
XOR	100%	100%
Spirals with loop	100%	100%
Wine Quality (White)	63.24%	61.81%
Vertebral Column	93.50%	79.56%
Letter Recognition	95.98%	66.58%
Default of Credit Card Clients	82.03%	81.66%
Teaching Assistant Evaluation	71.00%	55.55%

5.4.5 Experimento 2

Seguindo a configuração para treinamento proposta anteriormente, que basicamente restringe alguns valores para os hiperparâmetros em cada base, obtemos os resultados presentes na Tabela 13 ao testar valores para $v = (M, N, F)$ no caso da *MNIST* e $v = (M, O, F)$ para *GTSRB*. É claro que esses são apenas alguns dos testes possíveis, porém existem diversas outras combinações de hiperparâmetros que podem gerar resultados melhores e, manualmente, é difícil de encontrar essa combinação ótima.

Tabela 13 – Resultados dos experimentos variando os hiperparâmetros.

Taxa de Acerto no Treino / Teste		
Arquitetura	MNIST	GTSRB
CNN $(12, 0, 6)$	99.43% / 99.12%	98.94% / 95.70%
SMCNN $(0, 12, 6)$	99.11% / 98.73%	98.45% / 94.62%
SMCNN $(0, 12, 6)$ $\alpha = -5$	99.45% / 99.04%	99.18% / 94.72%
SMCNN $(8, 4, 6)$ $\alpha = -5$	99.44% / 99.13%	98.95% / 94.96%
SMCNN $(4, 8, 6)$ $\alpha = 2.5$	98.36% / 98.91%	99.07% / 95.21%

Com essas configurações apresentadas, na base *MNIST*, a SMCNN (8,4,6) com $\alpha = -5$ fixo, que realiza uma erosão na camada, superou a CNN e todos os resultados com essa arquitetura obtiveram uma acurácia próxima de redes presentes na literatura, como a MCDNN [92] e SpinalNet [93], que atingem no conjunto de teste, respectivamente, acertos de 99.77% e 99.72%. Porém, como podemos observar, a configuração em que o α é treinado com *backpropagation* atingiu uma acurácia menor em relação àquela em que ele foi fixado. Isso pode ocorrer pois, pela equação da *SMorph*, a quantidade de épocas necessárias para a convergência desse parâmetro pode ser maior do que a utilizada.

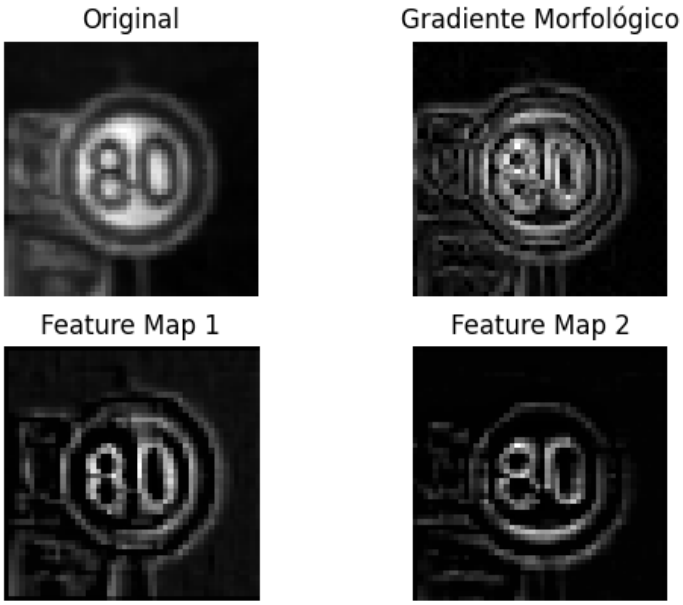
Já na *GTSRB*, a diferença entre a acurácia de treino e teste pode indicar que a rede encontrou um problema para generalizar os dados. Isso sugere que essa base necessita de um pré-processamento mais cuidadoso ou, também, que o algoritmo de treinamento Adam causou um *overfitting*. Porém, ainda assim, os resultados gerados são competitivos mesmo que não tenham, em um primeiro momento, superado a CNN com vetor de hiperparâmetros $v = (12, 0, 6)$ na base da dados.

Além disso, os resultados da aplicação de operações morfológicas de dilatação e erosão em uma imagem podem ser identificados. Assim, como a *SMorph* busca reproduzir essas operações, é de se esperar que uma arquitetura contendo essa camada morfológica reproduza esse efeito e que o mesmo possa ser identificado também. Logo, as Figuras 72 e 73 mostram exemplos da operação de dilatação realizada por métodos não diferenciáveis em um dígito da *MNIST* e do gradiente morfológico em um elemento da *GTSRB*, além de alguns *feature maps* obtidos pela rede com a *SMorph*, que busca reproduzir essas operações. Em ambos, esses *feature maps* foram escolhidos por meio da SMCNN quando seu vetor de hiperparâmetros é igual a $(4, 8, 6)$ com $\alpha = 2.5$ fixo.

Figura 72 – Dígito 7 aplicado à SMCNN $(4, 8, 6)$ com $\alpha = 2.5$.

Fonte: Autoria Própria.

Figura 73 – Elemento da *GTSRB* aplicado à SMCNN $(4, 8, 6)$ com $\alpha = 2.5$.



Fonte: Autoria Própria.

Nota-se que, visualmente, em ambos os casos, seus *feature maps* parecem realizar uma operação semelhante ao que foi proposto na imagem superior direita. Entretanto,

suas pequenas diferenças podem ser geradas pelos diferentes elementos de estruturação presentes em cada uma, sendo que na rede tal elemento foi aprendido por *backpropagation*. Vale ressaltar que, apesar de visualmente a rede conseguir reproduzir operações de dilatação e erosão, não podemos afirmar qual característica exatamente a rede extrai para a classificação.

Como a Tabela 13 mostra os resultados apenas para combinações de hiperparâmetros encontrados manualmente, as Tabelas 14 e 15 mostram resultados com NAS usando *Random Search* para as 5 melhores combinações de hiperparâmetros encontradas, respectivamente, na MNIST e *GTSRB*, as quais resultam na maior acurácia no conjunto de validação. Também mostram o resultado da acurácia nos conjuntos de treino+validação e teste, para comparar com os resultados encontrados manualmente. As arquiteturas usadas são da forma SMCNN (M, N, F) para *MNIST* e SMCNN (M, O, F) para *GTSRB*, equivalentes portanto às usadas nos resultados anteriores.

Tabela 14 – Resultados de NAS com *Random Search* para *MNIST*.

	Taxa de Acerto		
Arquitetura	Conjuntos (Épocas de treino)		
	Validação (15 épocas)	Treino + Validação (25 épocas)	Teste (25 épocas)
SMCNN $(16, 8, 12)$ $\alpha = -5.0$	99.094%	99.730%	99.281%
SMCNN $(12, 8, 12)$ $\alpha = -2.5$	99.083%	99.740%	99.253%
CNN $(24, 0, 12)$	99.077%	99.685%	99.096%
SMCNN $(12, 12, 12)$ $\alpha = 5.0$	99.055%	99.680%	99.232%
CNN $(12, 0, 12)$	99.044%	99.711%	99.170%

Tabela 15 – Resultados de NAS com *Random Search* para *GTSRB*.

Arquitetura	Taxa de Acerto		
	Conjuntos (Épocas de treino)		
	Validação (15 épocas)	Treino + Validação (25 épocas)	Teste (25 épocas)
SMCNN $(8, 12, 12)$ $\alpha = 5.0$	98.962%	99.701%	96.510%
CNN $(16, 0, 12)$	98.937%	99.607%	96.240%
SMCNN $(12, 12, 12)$ $\alpha = -5.0$	98.894%	99.650%	96.579%
SMCNN $(8, 12, 12)$ $\alpha = -5.0$	98.784%	99.658%	96.682%
SMCNN $(12, 8, 12)$ $\alpha = -5.0$	98.775%	99.648%	96.706%

As tabelas mostram que, com NAS usando *Random Search*, foi possível melhorar os resultados obtidos ao se comparar com as combinações escolhidas manualmente na Tabela 13 em ambas as bases de dados. Além disso, podemos observar que os melhores resultados encontrados pelo *search space* definido fazem o uso das operações morfológicas. Similarmente, nas Tabelas 16 e 17, apresentamos os resultados utilizando NAS com *BOHB* no seguinte formato: SMCNN $(M, N, F, h1, h2, drop)$ para *MNIST* e SMCNN $(M, O, F, h1, h2, drop)$ para a *GTSRB*, onde $h1$ e $h2$ indicam a quantidade de neurônios selecionados pelo método de NAS na primeira e segunda camada escondida, respectivamente.

Tabela 16 – Resultados de NAS com *BOHB* para *MNIST*.

Arquitetura	Taxa de Acerto		
	Conjunto		
	Validação (15 épocas)	Treino + Validação (25 épocas)	Teste (25 épocas)
SMCNN(16,4,12,240,128,0.51)	99.161%	99.815%	99.290%
SMCNN(0,24,12,240,60,0.49) $\alpha = -5$	99.092%	99.776%	99.370%
SMCNN(12,4,12,150,60,0.5) $\alpha = 5$	99.080%	99.731%	99.325%
SMCNN(0,24,12,240,128,0.5) $\alpha = -5$	99.065%	99.776%	99.310%
SMCNN(8,16,12,240,84,0.5) $\alpha = 5$	99.062%	99.781%	99.420%

Tabela 17 – Resultados de NAS com *BOHB* para *GTSRB*.

Arquitetura	Taxa de Acerto		
	Conjunto		
	Validação (15 épocas)	Treino + Validação (25 épocas)	Teste (25 épocas)
SMCNN(0,24,12,150,128, 0.22) $\alpha = 5$	99.251%	99.994%	96.832%
SMCNN(0,24,12,100,84, 0.24) $\alpha = 5$	99.149%	99.987%	96.524%
SMCNN(0,24,12,240,128,0.26) $\alpha = -5$	99.132%	99.999%	96.401%
SMCNN(0,24,12,256,140,0.31) $\alpha = 2.5$	99.124%	99.990%	96.205%
SMCNN(0,24,12,120,60,0.43) $\alpha = 5$	99.092%	99.961%	96.350%

No caso da *MNIST*, o uso de *BOHB* ao invés de *Random Search* em conjunto com o aumento do *search space* que pode ser comparado nas Tabelas 4 e 5 conseguiu melhorar ainda mais a acurácia no conjunto de teste e em todos os casos selecionou uma arquitetura contendo operações morfológicas.

Apesar de, na *GTSRB*, os resultados ficarem um pouco aquém de métodos mais complexos apresentados na literatura, como a MCDNN [92] que atinge 99.46% no conjunto de teste, ainda assim nossos resultados superam vários dos trabalhos feitos nesse conjunto de dados, como o estudo de Zaklouta *et al.* [94], que atinge 96.14%. Além disso, vale ressaltar que o método de busca *BOHB* (Tabela 17) selecionou, em todos os casos, filtros morfológicos para compor a arquitetura da rede, o que demonstra a efetividade da combinação de elementos morfológicos com os convencionais em uma arquitetura híbrida, especialmente com o auxílio de técnicas de NAS automatizando os pontos em que cada tipo de neurônio deve ser usado.

5.4.6 Experimento 3

Com as configurações de treinamento mencionadas, a arquitetura VGG-16 pré treinada na ImageNet, adotamos a configuração de classificadores da Tabela 18, que foi encontrada empiricamente e levou aos melhores resultados. Na Tabela 19 mostramos as acurácias de teste atingidas por ambas as arquiteturas e também por resultados da literatura que usam um método similar para avaliar a acurácia.

Tabela 18 – Classificadores na VGG-16 adotados para comparação na base FMD.

	VGG com Classificadores Clássicos (VGG16-c)				VGG com Classificadores Morfológicos (VGG16-m)			
	Tipo de Camada	Ativação + Dropout	Quantidade de Neurônios	Pré-treinada? Treinável?	Tipo de Camada	Ativação + Dropout	Quantidade de Neurônios	Pré-treinada? Treinável?
Camada Densa 1	Perceptron Clássico	ReLU 0.5	4096	Sim Não	Perceptron Clássico	ReLU 0.5	4096	Sim Não
Camada Densa 2	Perceptron Clássico	ReLU 0.5	4096	Sim Não	Perceptron Clássico	ReLU 0.5	4096	Sim Não
Camada Densa 3	N/A	N/A	N/A	N/A N/A	Perceptron Clássico	ReLU 0.0	1000	Sim Sim
Camada Densa 4	N/A	N/A	N/A	N/A N/A	MorphMul	Identidade 0.0	500	Não Sim
Camada de Saída	Perceptron Clássico	N/A 0.0	10	Não Sim	Perceptron Clássico	Identidade 0.0	10	Não Sim

Tabela 19 – Resultado do nosso modelo e outros da literatura.

Modelo	Acurácia no Conjunto de Teste (%)
VGG-16c	82.44
VGG-16m	82.90
OBIFs [88]	55.78
FC-CNN VGGVD [89]	77.40 ± 1.8
FV-CNN VGGVD [89]	79.8 ± 1.8
FC-CNN + FV-CNN VGGVD [89]	82.4 ± 1.5
Fusion CNN [90]	86.1 ± 1.6
Deep-TEN [91]	80.2 ± 0.9

Podemos observar na Tabela 18 a estrutura dos classificadores clássicos, enquanto que na outra estrutura usamos apenas um bloco morfológico similar a ideia de [40] na última camada para classificar. Com isso, vemos na Tabela 19 que obtemos valores competitivos de acurácia em relação a outros resultados da literatura. Em particular, a VGG-16m que possui a operação morfológica MorphMul supera ligereiramente a VGG-16c que usa apenas *perceptrons* clássicos em suas camadas densas.

5.5 Síntese dos Resultados

A partir das fronteiras de decisão, acurácias, arquiteturas contruídas e imagens, podemos concluir o seguinte à respeito das operações morfológicas utilizadas:

- No primeiro experimento, vemos que as operações morfológicas apresentaram acurácias competitivas em relação a outros algoritmos e, além disso, nos *datasets* sintéticos a fronteira de decisão se adaptou melhor em comparação a outros algoritmos;

- A comparação entre as operações MorphMul e PositiveMorphMul mostrou que a operação MorphMul, sem restrições do máximo e mínimo multiplicativo, resultou em melhores acurácias. Esse resultado pode ocorrer devido as propriedades da MorphMul que reproduzem um *perceptron* com α próximo de 0 e operações não lineares de máximo ($\alpha > 0$) e mínimo ($\alpha < 0$);

- Como podemos ver nas imagens da MNIST e GTSRB apresentadas no Experimento 2, a operação SMorph gerou imagens com características que podem ser interessantes se aplicadas a outros *datasets* onde operações de dilatação e erosão podem ser interessantes para destacar características dos dados e, além disso, suas acurácias foram similares ao estado da arte;

- Na base de texturas FMD, o uso de *transfer learning* resultou em acurácias competitivas com o estado da arte, tanto usando classificadores clássicos quanto um bloco morfológico. Entretanto, a construção de arquiteturas morfológicas densas para classificação parece ser sensível pois, após diversas tentativas e erros, apenas a operação MorphMul resultou em uma arquitetura com acurácia de fato interessante;

- O uso de NAS nos Experimentos 1 e 2 possibilitou a melhora da acurácia nos *datasets* testados e também mostrou que as operações morfológicas podem compor uma arquitetura híbrida (contendo operações clássicas e as morfológicas). Porém, algoritmos de NAS ainda são custosos computacionalmente, então nesse trabalho o benéficio que um algoritmo de NAS proporciona foi limitado devido as especificações do computador utilizado.

6 Conclusões

6.1 Conclusões Sobre o Trabalho Realizado

Este trabalho propôs duas operações morfológicas para camadas densas, denominadas nos experimentos de MorphMul e MorphAdd, que podem ser treinadas através de *backpropagation*. Mostramos que o uso combinado dessas operações em conjunto com camadas escondidas de perceptrons clássicos possibilitaram uma melhora na generalização da rede neural nos *datasets* estudados.

Além disso, mostramos que a operação morfológica *SMorph* consegue obter resultados competitivos em comparação aos presentes na literatura em tarefas de classificação de imagens. Também, essa operação teve sucesso em aprender os filtros que reproduzem dilatações, erosões e gradientes morfológicos, em particular, o gradiente externo através do *backpropagation*.

Por fim, mostramos também que a adição de operações morfológicas ao *search space* de *Neural Architecture Search* possibilitou uma melhora na capacidade de generalização da rede em diferentes *datasets*. No caso das operações morfológicas em camadas densas totalmente conectadas, o uso de NAS construiu arquiteturas e usou essas operações em diferentes camadas em conjunto (ou não) com camadas de *perceptrons* clássicos dependendo do *dataset* utilizado. Em particular, isso nos mostra que essas operações podem ser úteis em diferentes situações e problemas da atualidade.

No caso da operação SMorph, o uso de NAS para a otimização de hiperparâmetros possibilitou uma melhor escolha de arquitetura para os *datasets* utilizados e confirmou que as operações morfológicas convolucionais propostas possuem informações interessantes para melhorar a acurácia de teste.

6.2 Sugestões para Trabalhos Futuros

Com as conclusões desse trabalho, o próximo passo lógico é seguir fazendo testes com essas operações em *datasets* mais competitivos e complexos, podendo integrar técnicas inovadoras de NAS para auxiliar a busca de arquiteturas ou hiperparâmetros ótimos. Uma sugestão é a utilização dessas operações em outros *datasets* de texturas, onde a MorphMul e MorphAdd podem servir como camadas de classificadores e a operação *SMorph* pode ser útil para remoção de ruídos e aperfeiçoamento das imagens.

Além disso, o uso de múltiplas camadas da *SMorph* geram operações mais complexas como as de *opening* e *closing*. Neste trabalho, a *SMorph* foi definida para

imagens, gerando uma convolução bidimensional, porém também é possível definir a *SMorph* como uma convolução unidimensional e ser aplicada em *datasets* compostos, por exemplo, por series temporais.

Referências

1 BILMES, J. Graphical models and automatic speech recognition. In: *Mathematical foundations of speech and language processing*. [S.l.]: Springer, 2004. p. 191–245. Citado na página 15.

2 DAHL, G. et al. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, IEEE, v. 20, n. 1, p. 30–42, 2011. Citado 2 vezes nas páginas 15 e 19.

3 AYAT, N. et al. Kmod-a new support vector machine kernel with moderate decreasing for pattern recognition. application to digit image recognition. In: IEEE. *Proceedings of Sixth International Conference on Document Analysis and Recognition*. [S.l.], 2001. p. 1215–1219. Citado na página 15.

4 KRIZHEVSKY, A. et al. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, v. 25, p. 1097–1105, 2012. Citado 2 vezes nas páginas 15 e 19.

5 LECUN, Y. et al. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citado na página 15.

6 RITTER, G.; URCID, G. Lattice algebra approach to single-neuron computation. *IEEE Transactions on Neural Networks*, IEEE, v. 14, n. 2, p. 282–295, 2003. Citado 4 vezes nas páginas 16, 20, 32 e 33.

7 JAMIL, N. Noise removal and enhancement of binary images using morphological operations. In: IEEE. *2008 International Symposium on Information Technology*. [S.l.], 2008. v. 4, p. 1–6. Citado na página 16.

8 ZOPH, B.; LE, Q. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. Citado 4 vezes nas páginas 16, 21, 22 e 37.

9 KIRSZENBERG, A. et al. Going beyond p-convolutions to learn grayscale morphological operators. In: SPRINGER. *International Conference on Discrete Geometry and Mathematical Morphology*. [S.l.], 2021. p. 470–482. Citado 9 vezes nas páginas 16, 21, 33, 35, 41, 47, 48, 49 e 54.

10 MCCULLOCH, W.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943. Citado 2 vezes nas páginas 18 e 24.

11 ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado na página 18.

12 _____. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. [S.l.: s.n.], 1961. Citado 2 vezes nas páginas 18 e 24.

13 WIDROW, B.; HOFF, M. Associative storage and retrieval of digital information in networks of adaptive "neurons". In: *Biological Prototypes and Synthetic Systems*. [S.l.]: Springer, 1962. p. 160–160. Citado na página 18.

14 NARENDRA, K.; THATHACHAR, M. Learning automata-a survey. *IEEE Transactions on systems, man, and cybernetics*, IEEE, n. 4, p. 323–334, 1974. Citado na página 18.

15 SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural networks*, Elsevier, v. 61, p. 85–117, 2015. Citado na página 18.

16 KELLEY, H. Gradient theory of optimal flight paths. *Ars Journal*, v. 30, n. 10, p. 947–954, 1960. Citado na página 18.

17 BRYSON, A.; DENHAM, W. A steepest-ascent method for solving optimum programming problems. 1962. Citado na página 18.

18 AMARI, S. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, IEEE, n. 3, p. 299–307, 1967. Citado na página 18.

19 LINNAINMAA, S. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, Springer, v. 16, n. 2, p. 146–160, 1976. Citado na página 18.

20 SPEELPENNING, B. *Compiling fast partial derivatives of functions given by algorithms*. Tese (Doutorado) — University of Illinois at Urbana-Champaign, 1980. Citado na página 18.

21 RUMELHART, D.; HINTON, G.; WILLIAMS, R. *Learning internal representations by error propagation*. [S.l.], 1985. Citado na página 18.

22 CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, Springer, v. 2, n. 4, p. 303–314, 1989. Citado 2 vezes nas páginas 18 e 25.

23 LECUN, Y. et al. Handwritten digit recognition with a back-propagation network. In: TOURETZKY, D. (Ed.). *Advances in Neural Information Processing Systems*. Morgan-Kaufmann, 1990. v. 2. Disponível em: <https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf>. Citado na página 19.

24 _____. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998. Citado 2 vezes nas páginas 19 e 53.

25 DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: IEEE. *2009 IEEE conference on computer vision and pattern recognition*. [S.l.], 2009. p. 248–255. Citado na página 19.

26 ZEILER, M.; FERGUS, R. Visualizing and understanding convolutional networks. In: SPRINGER. *European conference on computer vision*. [S.l.], 2014. p. 818–833. Citado na página 19.

27 SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. Citado 2 vezes nas páginas 19 e 59.

28 SZEGEDY, C.; LIU, W. et al. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2014. p. 1–9. Citado na página 19.

29 HE, K.; ZHANG, X. et al. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015. p. 770–778. Citado na página 19.

30 HOWARD, A. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. Citado na página 20.

31 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. Citado na página 20.

32 WILSON, S. Morphological networks. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Visual Communications and Image Processing IV*. [S.l.], 1989. v. 1199, p. 483–495. Citado na página 20.

33 DAVIDSON, J.; RITTER, G. Theory of morphological neural networks. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Digital Optical Computing II*. [S.l.], 1990. v. 1215, p. 378–388. Citado na página 20.

34 RITTER, G.; DAVIDSON, J. Recursion and feedback in image algebra. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Image Understanding in the'90s: Building Systems that Work*. [S.l.], 1991. v. 1406, p. 74–86. Citado na página 20.

35 BIRKHOFF, G. *Lattice theory*. [S.l.]: American Mathematical Soc., 1940. v. 25. Citado na página 20.

36 ANGULO, J. Pseudo-morphological image diffusion using the counter-harmonic paradigm. In: SPRINGER. *International Conference on Advanced Concepts for Intelligent Vision Systems*. [S.l.], 2010. p. 426–437. Citado na página 20.

37 MASCI, J. et al. A learning framework for morphological operators using counter-harmonic mean. In: SPRINGER. *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*. [S.l.], 2013. p. 329–340. Citado 4 vezes nas páginas 20, 21, 33 e 41.

38 ZAMORA, E.; SOSSA, H. Dendrite morphological neurons trained by stochastic gradient descent. *Neurocomputing*, Elsevier, v. 260, p. 420–431, 2017. Citado na página 21.

39 MELLOULI, D. et al. Morphological convolutional neural network architecture for digit recognition. *IEEE transactions on neural networks and learning systems*, IEEE, v. 30, n. 9, p. 2876–2885, 2019. Citado 3 vezes nas páginas 21, 30 e 33.

40 MONDAL, R. et al. Morphological network: How far can we go with morphological neurons? *arXiv preprint arXiv:1901.00109*, 2019. Citado 5 vezes nas páginas 21, 44, 55, 66 e 95.

41 BAKER, B. et al. Designing Neural Network Architectures using Reinforcement Learning. *arXiv e-prints*, p. arXiv:1611.02167, nov. 2016. Citado 2 vezes nas páginas 22 e 37.

42 PHAM, H. et al. Efficient neural architecture search via parameters sharing. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2018. p. 4095–4104. Citado na página 22.

43 ADAM, G. et al. Understanding neural architecture search techniques. *arXiv preprint arXiv:1904.00438*, 2019. Citado na página 22.

44 BENDER, G. et al. Understanding and simplifying one-shot architecture search. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2018. p. 550–559. Citado na página 22.

45 CHU, X. et al. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019. Citado na página 22.

46 XIA, X. et al. Hnas: Hierarchical neural architecture search on mobile devices. *arXiv preprint arXiv:2005.07564*, 2020. Citado na página 22.

47 REAL, E. et al. Regularized evolution for image classifier architecture search. In: *Proceedings of the aaai conference on artificial intelligence*. [S.l.: s.n.], 2019. v. 33, n. 01, p. 4780–4789. Citado na página 22.

48 LIU, H. et al. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017. Citado na página 22.

49 LIU, C. et al. Progressive neural architecture search. In: *Proceedings of the European conference on computer vision (ECCV)*. [S.l.: s.n.], 2018. p. 19–34. Citado na página 22.

50 LIU, H. et al. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. Citado na página 23.

51 ZHANG, K. et al. Differentiable neural architecture search augmented with pruning and multi-objective optimization for time-efficient intelligent fault diagnosis of machinery. *Mechanical Systems and Signal Processing*, Elsevier, v. 158, p. 107773, 2021. Citado na página 23.

52 BISHOP, C. et al. *Neural networks for pattern recognition*. [S.l.]: Oxford university press, 1995. Citado na página 24.

53 NIELSEN, M. *Neural networks and deep learning*. [S.l.]: Determination press San Francisco, CA, 2015. v. 25. Citado na página 25.

54 HORNIK, K. et al. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989. Citado na página 25.

55 SVOZIL, D.; OTHERSI. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, Elsevier, v. 39, n. 1, p. 43–62, 1997. Citado na página 25.

56 GOODFELLOW, I. et al. *Deep learning*. [S.l.]: MIT press, 2016. Citado 3 vezes nas páginas 25, 27 e 28.

57 WANG, S. Artificial neural network. In: *Interdisciplinary computing in java programming*. [S.l.]: Springer, 2003. p. 81–100. Citado na página 25.

58 GÉRON, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. [S.l.]: O'Reilly Media, 2019. Citado na página 27.

59 CHEN, Q. et al. Fast image processing with fully-convolutional networks. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2017. p. 2497–2506. Citado na página 27.

60 ABDEL-HAMID, O. et al. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, IEEE, v. 22, n. 10, p. 1533–1545, 2014. Citado na página 27.

61 ALBAWI, S. et al. Understanding of a convolutional neural network. In: IEEE. *2017 International Conference on Engineering and Technology (ICET)*. [S.l.], 2017. p. 1–6. Citado na página 27.

62 SOILLE, P. *Morphological image analysis: principles and applications*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 29.

63 RIVEST, J. Morphological gradients. *Journal of Electronic Imaging*, International Society for Optics and Photonics, v. 2, n. 4, p. 326–336, 1993. Citado 2 vezes nas páginas 31 e 48.

64 WILSON, J.; RITTER, G. *Handbook of computer vision algorithms in image algebra*. [S.l.]: CRC press, 2000. Citado na página 31.

65 RITTER, G.; SUSSNER, P. An introduction to morphological neural networks. In: IEEE. *Proceedings of 13th International Conference on Pattern Recognition*. [S.l.], 1996. v. 4, p. 709–717. Citado 2 vezes nas páginas 31 e 32.

66 SILVA, A.; SUSSNER, P. A brief review and comparison of feedforward morphological neural networks with applications to classification. In: SPRINGER. *International Conference on Artificial Neural Networks*. [S.l.], 2008. p. 783–792. Citado 2 vezes nas páginas 32 e 42.

67 ANGULO, J. Generalised morphological image diffusion. *Nonlinear Analysis*, Elsevier, v. 134, p. 1–30, 2016. Citado na página 33.

68 LANGE, M. et al. Applications of lp-norms and their smooth approximations for gradient based learning vector quantization. In: *ESANN*. [S.l.: s.n.], 2014. p. 271–276. Citado na página 34.

69 ELSKEN, T. et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, v. 20, n. 55, p. 1–21, 2019. Citado na página 36.

70 REN, P. et al. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020. Citado na página 37.

71 FALKNER, S.; KLEIN, A. et al. Bohb: Robust and efficient hyperparameter optimization at scale. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2018. p. 1437–1446. Citado 2 vezes nas páginas 38 e 50.

72 LI, L. et al. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In: *ICLR (Poster)*. [S.l.: s.n.], 2017. Citado na página 38.

73 JAMIESON, K. et al. Non-stochastic best arm identification and hyperparameter optimization. In: PMLR. *Artificial Intelligence and Statistics*. [S.l.], 2016. p. 240–248. Citado na página 38.

74 RIPLEY, B. *Pattern recognition and neural networks*. [S.l.]: Cambridge university press, 2007. Citado na página 40.

75 SHIH, F. *Image processing and mathematical morphology: fundamentals and applications*. [S.l.]: CRC press, 2009. Citado na página 40.

76 BNOUNI, N. et al. Boosting cnn learning by ensemble image preprocessing methods for cervical cancer segmentation. In: IEEE. *2021 18th International Multi-Conference on Systems, Signals & Devices (SSD)*. [S.l.], 2021. p. 264–269. Citado na página 40.

77 CHEN, Y. et al. The application of a convolution neural network on face and license plate detection. In: IEEE. *18th International Conference on Pattern Recognition (ICPR'06)*. [S.l.], 2006. v. 3, p. 552–555. Citado na página 40.

78 HU, Y. et al. Learning deep morphological networks with neural architecture search. *arXiv preprint arXiv:2106.07714*, 2021. Citado 2 vezes nas páginas 41 e 46.

79 CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. [S.l.: s.n.], 2016. p. 785–794. Citado na página 44.

80 BENAVENT, X. et al. Mathematical morphology for color images: An image-dependent approach. *Mathematical Problems in Engineering*, Hindawi, v. 2012, 2012. Citado na página 48.

81 BERGSTRA, J. et al. Random search for hyper-parameter optimization. *Journal of machine learning research*, v. 13, n. 2, 2012. Citado na página 50.

82 DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>. Citado na página 50.

83 HOUBEN, S. et al. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In: *International Joint Conference on Neural Networks*. [S.l.: s.n.], 2013. Citado na página 53.

84 SHARAN, L.; ROSENHOLTZ, R.; ADELSON, E. H. Accuracy and speed of material categorization in real-world images. *Journal of Vision*, v. 14, n. 10, 2014. Citado na página 54.

85 KINGMA, D. et al. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Citado na página 57.

86 WEISS, K. et al. A survey of transfer learning. *Journal of Big data*, SpringerOpen, v. 3, n. 1, p. 1–40, 2016. Citado na página 59.

87 BU, X.; WU, Y.; GAO, Z.; JIA, Y. Deep convolutional network with locality and sparsity constraints for texture classification. *Pattern Recognition*, Elsevier, v. 91, p. 34–46, 2019. Citado na página 60.

88 TIMOFTE, R.; GOOL, L. V. A training-free classification framework for textures, writers, and materials. In: *BMVC*. [S.l.: s.n.], 2012. v. 13, p. 14. Citado 2 vezes nas páginas 60 e 95.

89 CIMPOI, M.; MAJI, S.; VEDALDI, A. Deep filter banks for texture recognition and segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 3828–3836. Citado 2 vezes nas páginas 60 e 95.

90 JBENE, M.; M., A. D. E.; HASSOUNI, M. E. Fusion of convolutional neural network and statistical features for texture classification. In: IEEE. *2019 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. [S.l.], 2019. p. 1–4. Citado 2 vezes nas páginas 60 e 95.

91 ZHANG, H.; XUE, J.; DANA, K. Deep ten: Texture encoding network. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 708–717. Citado 2 vezes nas páginas 60 e 95.

92 CIREGAN, D. et al. Multi-column deep neural networks for image classification. In: IEEE. *2012 IEEE conference on computer vision and pattern recognition*. [S.l.], 2012. p. 3642–3649. Citado 2 vezes nas páginas 90 e 94.

93 KABIR, H. et al. Spinalnet: Deep neural network with gradual input. *arXiv preprint arXiv:2007.03347*, 2020. Citado na página 90.

94 ZAKLOUTA, F. et al. Traffic sign classification using kd trees and random forests. In: IEEE. *The 2011 international joint conference on neural networks*. [S.l.], 2011. p. 2151–2155. Citado na página 94.