Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

AutoML e sua contribuição para o desenvolvimento de modelos para a Aprendizagem Profunda

Lucas Henrique Lírio Costa

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Prof. Talles Henrique de Medeiros

Fevereiro, 2024
João Monlevade–MG

Lucas Henrique Lírio Costa

AutoML e sua contribuição para o desenvolvimento de modelos para a Aprendizagem Profunda

Orientador: Prof. Talles Henrique de Medeiros

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Fevereiro de 2024

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

C837a Costa, Lucas Henrique Lirio.
AutoML e sua contribuição para o desenvolvimento de modelos para a aprendizagem profunda. [manuscrito] / Lucas Henrique Lirio Costa. - 2024.
41 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Talles Henrique de Medeiros.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia de Computação .

1. Aprendizado do computador. 2. Entropia (Teoria da informação). 3. Inteligência artificial. 4. Otimização matemática. 5. Probabilidades. 6. Python (Linguagem de programação de computador). I. Medeiros, Talles Henrique de. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.8

Bibliotecário(a) Responsável: Flavia Reis - CRB6-2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Lucas Henrique Lírio Costa

AutoML e sua Contribuição para o Desenvolvimento de Modelos para a Aprendizagem Profunda

Monografia apresentada ao Curso de Engenharia da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação

Aprovada em 21 de fevereiro de 2024.

Membros da banca

Doutor - Talles Henrique de Medeiros - Orientador - Universidade Federal de Ouro Preto
Doutor - Eduardo da Silva Ribeiro - Universidade Federal de Ouro Preto
Doutor - Rafael Frederico Alexandre - Universidade Federal de Ouro Preto

Talles Henrique de Medeiros, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 02/03/2024.

Documento assinado eletronicamente por **Talles Henrique de Medeiros**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 04/03/2024, às 21:08, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0671176** e o código CRC **6101B969**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Agradecimentos

Agradeço, primeiramente a Deus, pela vida, saúde e motivação para seguir em frente e alcançar meus objetivos. Agradeço por ter iluminado meus passos, e colocado pessoas importantes no meu caminho durante toda essa jornada.

Agradeço a mim, pela persistência, coragem e determinação para não desistir diante dos desafios vividos.

Agradeço aos meus pais, pelo sustento financeiro e apoio psicológico durante toda a minha formação como estudante, em especial à minha irmã e minha namorada que sempre me apoiaram no meu desenvolvimento pessoal.

Agradeço ao meu professor orientador, pelos ensinamentos, paciência e confiança, e também aos demais professores do ICEA que foram essenciais na graduação.

Agradeço a meus amigos da faculdade, em especial aos do período 2017/1, pela amizade, parceria e pelos bons momentos passados juntos.

"A sabedoria é a coisa principal; adquire, pois, a sabedoria; sim, com tudo o que possuis, adquire o conhecimento."

— Provérbios 4:7

Resumo

O Aprendizado de Máquina Automatizado (AutoML) é uma área de estudo da Inteligência Artificial (IA) que propõe a criação e configuração de modelos de aprendizado de máquina com mínima ou nenhuma interferência humana. A abordagem deste trabalho foca na otimização de hiperparâmetros, que são parâmetros de configuração do modelo não treináveis que afetam diretamente seu desempenho. Para isto, foi empregada a Otimização Bayesiana e uma função de perda multiobjetivo baseada na Escalarização Chebyshev, cujo os objetivos são o erro de treinamento, medido pela função de entropia cruzada, e a complexidade do modelo, medida pela regularização L2. Os experimentos foram realizados através de códigos na linguagem Python, utilizando a biblioteca KerasTuner e escritos em notebooks do Google Colaboratory. O trabalho obteve resultados satisfatórios, como um modelo que apresentou 99,34% de acurácia ao ser aplicado na base de dados MNIST, e foram identificados pontos de melhorias e aproveitamento do estudo em trabalhos futuros.

Palavras-chaves: AutoML. Inteligência Artificial. Aprendizado de máquina. Otimização de hiperparâmetros. Otimização Bayesiana. Escalarização Chebyshev. Entropia cruzada. Regularização L2. KerasTuner.

Abstract

Automated Machine Learning (AutoML) is a subfield of Artificial Intelligence (AI) that proposes the creation and configuration of machine learning models with minimal or no human interference. The approach of this work focuses on hyperparameter optimization, which refers to the optimization of non-trainable model configuration parameters, that directly impact its performance. For this purpose, were used Bayesian Optimization and a multi-objective loss function based on Chebyshev Scalarization, wich objectives include minimizing the training error, measured by the cross-entropy loss function, and the model's complexity, measured by L2 regularization. The experiments were conducted using Python code, through the KerasTuner library and implemented in Google Colaboratory notebooks. The results were satisfactory, showcasing a model with 99.34% accuracy, applied on the MNIST dataset, and areas for improvement and study for future work were highlighted.

Key-words: AutoML. Artificial Intelligence. Machine Learning. Hyperparameter Optimization. Chebyshev Scalarization. Cross-entropy. L2 Regularization. KerasTuner.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IA Inteligência Artificial

AutoML Automated Machine Learning

AM Aprendizado de Máquina

NAS Neural Architecture Search

OB Otimização Bayesiana

CNN Convolutional Neural Network

HPs Hiperparâmetros

MLP MultiLayer Perceptron

Sumário

1 INTRODUÇÃO

Nos últimos anos, o aprendizado de máquina, especificamente no campo da aprendizagem profunda, tem sido amplamente aplicado em várias áreas de estudo e obtendo resultados satisfatórios na resolução de desafios de Inteligência Artificial (IA), como nas áreas de detecção de objetos (BAI et al., 2020), classificação de imagens (NAHID; KONG et al., 2017), processamento de linguagem natural (KHAN et al., 2016), entre outros. Com essa crescente evolução, têm surgido modelos de redes neurais artificiais cada vez mais profundos e complexos, porém eles dependem de serem projetados manualmente por especialistas através de um processo de tentativas que pode ser longo, exaustivo e computacionalmente custoso, para criar modelos que obtenham um bom desempenho. Para reduzir todo esse esforço no desenvolvimento de uma rede neural, foi criada a ideia de automatizar, ao máximo possível, o fluxo de trabalho do aprendizado de máquina, permitindo que pessoas construam automaticamente aplicações em IA sem a exigência de conhecimento profundo ou tempo de prática na área. Assim, surgiu o *Automated Machine Learning (AutoML)*, trazendo consigo suas inovações e desafios.

1.1 Elaboração do capítulo

Este trabalho visa estudar o AutoML, especificamente na etapa da otimização de hiperparâmetros, onde são procurados automaticamente os melhores valores que configuram uma rede neural. Foram usados notebooks em Python para implementar casos de exemplo a partir de bases de dados famosas na área de classificação de imagens como MNIST[1] e FashionMNIST[2], a fim de mostrar a aplicabilidade dos métodos para a busca dos melhores hiperparâmetros da rede.

1.2 O problema de pesquisa

O problema de pesquisa consiste em configurar redes neurais com hiperparâmetros obtidos por meio de uma busca automática em conjunto com a utilização de uma função de perda multiobjetivo, para auxiliar na busca pelos melhores valores de hiperparâmetros em um modelo de rede neural.

[1] <https://keras.io/api/datasets/mnist/>
[2] <https://keras.io/api/datasets/fashion_mnist/>

1.3 Objetivos

O objetivo deste trabalho é mostrar a aplicação da otimização de hiperparâmetros, um campo do AutoML, a fim de automatizar o processo de configuração de uma rede neural.

Este trabalho possui aos seguintes objetivos específicos:

- Implementar códigos para exemplificar e introduzir a busca automática de hiperparâmetros.
- Comparar os resultados obtidos em duas bases de dados famosas.
- Demonstrar que a busca de hiperparâmetros pode gerar modelos com um bom desempenho.
- Avaliar o impacto dos pesos da função de perda multiobjetivo.
- Visualizar a geração do conjunto de valores ótimos para a função multiobjetivo.

1.4 Metodologia

O objeto de pesquisa deste trabalho é o processo do AutoML, focado na etapa da otimização de hiperparâmetros e sua aplicação na automação da configuração de redes neurais.

Os passos para execução deste trabalho foram assim definidos:

- Revisão da literatura.
- Busca por um *framework*[3] de AutoML.
- Desenvolvimento de códigos para experimentos.
- Execução dos programas num ambiente em nuvem.
- Análise e discussão dos resultados obtidos das execuções.

1.5 Organização do trabalho

Este trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta a revisão bibliográfica de toda a base teórica utilizada no trabalho. O Capítulo 3 aborda como o trabalho foi desenvolvido, desde a escolha da linguagem e das bibliotecas, até a execução

[3] Conjunto de ferramentas e bibliotecas que facilitam o desenvolvimento de aplicações.

dos programas de testes. No Capítulo 4 são mostrados e comentados os resultados obtidos da execução dos programas e, por fim, no Capítulo 5 são feitas as considerações finais e são apresentadas propostas de trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

A Inteligência Artificial (IA) é um campo da ciência da computação que se dedica a criar sistemas capazes de realizar tarefas que normalmente exigem inteligência humana (SARKER, 2022). Uma de suas vertentes mais proeminentes é o Aprendizado de Máquina (AM), do inglês *Machine Learning*, sendo uma subcategoria da IA concentrada no desenvolvimento de algoritmos capazes de aprender padrões a partir de dados, e tomar decisões ou fazer previsões sem a necessidade de programação explícita (RASCHKA; PATTERSON; NOLET, 2020). Os modelos de aprendizado de máquina são capazes de aprender com experiências passadas e ajustar seu comportamento para melhor se adaptar a novos dados. Com o avanço e democratização da tecnologia nos últimos anos, o aprendizado de máquina tem sido explorado em muitas aplicações práticas, graças à sua versatilidade e capacidade de aprendizado de seus modelos.

Sendo um subcampo do aprendizado de máquina, a aprendizagem profunda (*Deep Learning*) estuda algoritmos que modelam e abstraem dados por meio de redes neurais artificiais com múltiplas camadas de processamento. Essas redes neurais artificiais são inspiradas no funcionamento do cérebro humano e são capazes de aprender representações complexas de dados através do ajuste dos seus pesos que conectam os neurônios uns aos outros. A aprendizagem profunda é frequentemente utilizada em tarefas como reconhecimento de padrões (BISHOP, 2006), processamento de linguagem natural (GLAZ et al., 2021), visão computacional (KHAN; LAGHARI; AWAN, 2021) e muitas outras aplicações de inteligência artificial.

2.1 AutoML

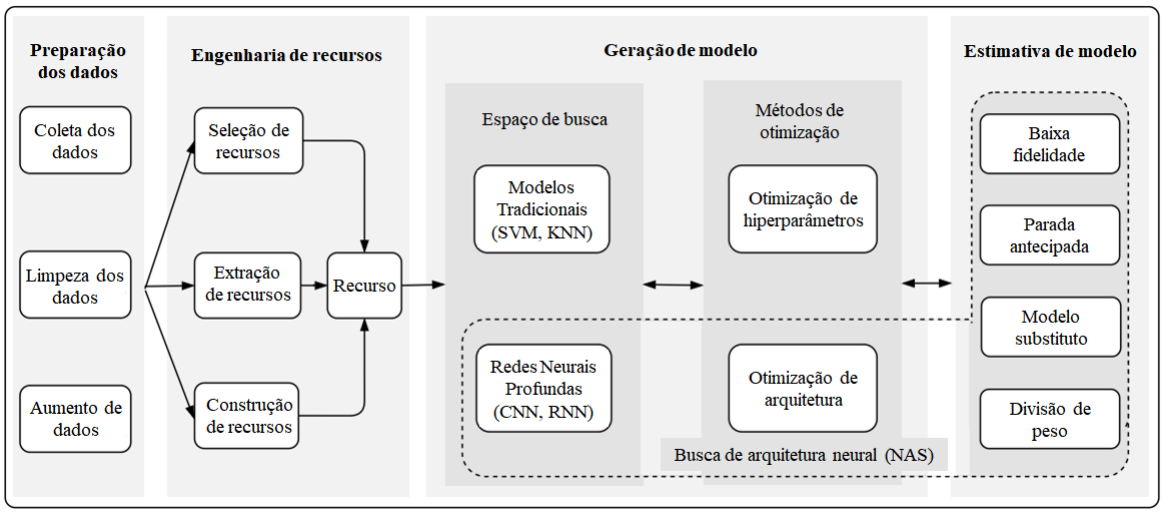
O termo AutoML se refere a um conjunto de técnicas e ferramentas que automatizam o processo de construção, treinamento e otimização de modelos de aprendizado de máquina, a fim de simplificar e acelerar o desenvolvimento desses modelos (HE; ZHAO; CHU, 2021). Além disso, permite que usuários com diferentes níveis de experiência em inteligência artificial possam criar modelos eficazes, sem a necessidade de ter um conhecimento especializado na área.

Segundo (HE; ZHAO; CHU, 2021), o *pipeline* (sequência de processos) do AutoML é dividido em quatro principais etapas, como mostradas na Figura 1, sendo elas:

- Preparação dos dados.
- Engenharia de recursos.

- Geração de modelo.
- Estimativa de modelo.

Figura 1 – *Pipeline* do AutoML e suas principais etapas.



Fonte: Adaptado de (HE; ZHAO; CHU, 2021).

As etapas de preparação dos dados e engenharia de recursos são cruciais para o pré processamento desses dados antes de serem aplicados em um modelo de IA (GARCÍA; LUENGO; HERRERA, 2015). Na primeira etapa, os dados são coletados e é realizada a limpeza e o aumento dos dados[1], a fim de melhorar a qualidade dos mesmos e, consequentemente, o desempenho do modelo a ser aplicado. A etapa da engenharia de recursos é responsável por selecionar as informações mais relevantes para o modelo, incluindo transformações e combinações de informações existentes para melhorar a representação dos dados. Na etapa da geração de modelo, o espaço de busca se refere ao conjunto de configurações possíveis a serem exploradas, e os métodos de otimização são aplicados ao modelo, a fim de obter a melhor configuração final. A última etapa possui métodos para avaliar os modelos gerados e escolher aquele que possui um melhor desempenho.

Na Figura 1 são destacadas no canto inferior direito as etapas que envolvem a Busca de arquitetura neural, do inglês *Neural Architecture Search (NAS)*, que é uma técnica do AutoML que visa encontrar automaticamente a arquitetura mais adequada para uma determinada tarefa de AM. Nesta técnica, é explorado o espaço de busca envolvendo redes neurais profundas e o método de otimização é utilizado para a melhorar a arquitetura do modelo gerado (ELSKEN; METZEN; HUTTER, 2019; KARMAKER et al., 2021).

[1] Técnica que aplica transformações simples nas imagens da base de dados para gerar novas imagens.

2.2 Busca de hiperparâmetros

Os Hiperparâmetros (HPs), são parâmetros que não são aprendidos diretamente pelo modelo durante o treinamento, mas que afetam diretamente o processo de treinamento e o desempenho do modelo de AM. Eles são definidos durante a configuração do modelo, antes do início do treinamento, e influenciam como o algoritmo de aprendizado aprende o problema e ajusta os pesos do modelo a partir dos dados. Como exemplos dos hiperparâmetros mais comuns em redes neurais, podemos citar a taxa de aprendizado, o número de épocas, o tamanho do lote, taxa de *dropout*, número de camadas ocultas, função de ativação, entre outros. A busca automática dos hiperparâmetros é uma etapa importante do AutoML, devido ao fato de que uma escolha adequada dos hiperparâmetros do modelo é crucial para sua eficácia. Esta etapa envolve técnicas como busca em grade, busca aleatória, otimização bayesiana (seção 2.3), ou algoritmos genéticos para encontrar a combinação ideal de hiperparâmetros que maximize o desempenho do modelo.

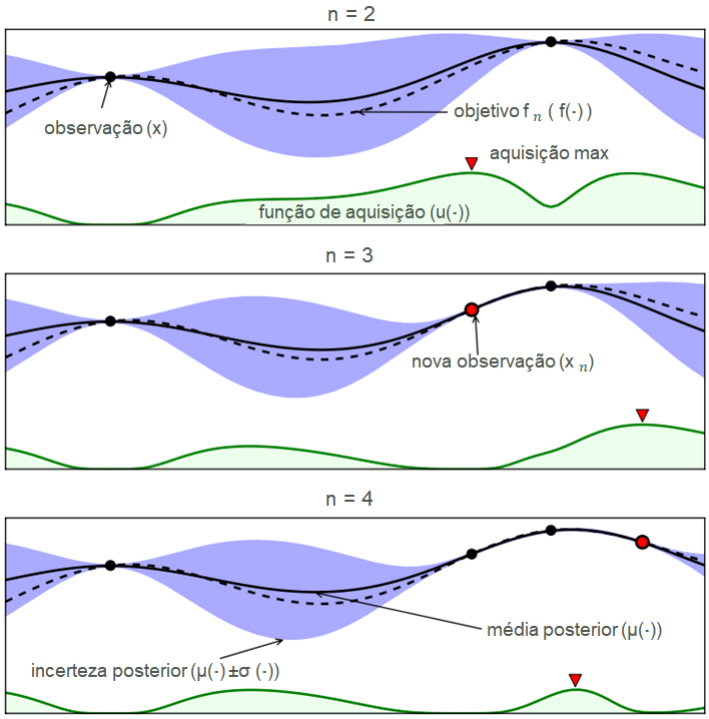
2.3 Otimização bayesiana

Para realizar a busca de hiperparâmetros é necessário a utilização de um método de otimização, que guiará a busca de forma a encontrar os melhores valores. A Otimização Bayesiana (OB) é uma classe de métodos de otimização baseados em aprendizado de máquina, focados em encontrar o ponto ótimo em funções objetivo que possuem um elevado custo computacional para serem avaliadas. Segundo (FRAZIER, 2018), a OB teve sua origem em meados dos anos 60, obtendo contribuições ao longo dos anos, mas recebeu significativamente mais atenção após o trabalho popularizado por (JONES; SCHONLAU; WELCH, 1998) e seu algoritmo *Efficient Global Optimization* (EGO).

O principal objetivo da otimização bayesiana é encontrar o máximo (ou mínimo) de uma função objetivo desconhecida de forma rápida e com o menor número de iterações possível. De acordo com (SHAHRIARI et al., 2015), o processo da OB envolve dois componentes principais: um modelo substituto probabilístico e uma função de perda. O modelo substituto probabilístico, é uma representação estatística da função objetivo que é construída com base em dados observados. Na OB, o modelo substituto probabilístico é frequentemente implementado usando a regressão de processo gaussiano, que é uma técnica de modelagem estatística que utiliza dos processos gaussianos para realizar regressão em dados. Esse modelo captura a média estimada da função objetivo, e também a incerteza associada a essa estimativa, permitindo que a OB tome decisões informadas sobre onde explorar o espaço de busca para encontrar o máximo global da função objetivo. Já a função de perda, é usada para avaliar o desempenho das diferentes configurações de hiperparâmetros ou pontos de amostragem, durante o processo de otimização. Ela quantifica o quão "custoso" é selecionar uma determinada configuração em relação ao

verdadeiro valor ótimo desconhecido da função objetivo. Desse modo, a função de perda desempenha um papel crucial na escolha de qual direção tomar para explorar o espaço de busca, ajudando a buscar com um menor tempo, novas regiões promissoras da função objetivo, a fim de encontrar o máximo global. O autor (FRAZIER, 2018) cita a importância da função de aquisição, que trabalha em conjunto com esses dois componentes principais (modelo substituto probabilístico e função de perda), para decidir a direção a ser tomada de onde realizar a próxima amostragem dentro do espaço de busca.

Figura 2 – *Ilustração do processo de otimização bayesiana.*



Fonte: Adaptado de (SHAHRIARI et al., 2015).

A Figura 2 ilustra o processo da Otimização Bayesiana e, em seguida, é mostrada uma tradução da descrição da imagem feita pelos autores:

> Ilustração do procedimento de otimização bayesiana ao longo de três iterações. Os gráficos mostram a média e os intervalos de confiança estimados com um modelo probabilístico da função objetivo. Embora a função objetivo seja mostrada, na prática, ela é desconhecida. Os gráficos também mostram as funções de aquisição nos gráficos inferiores sombreados. A aquisição é alta onde o modelo prevê um alto valor objetivo e onde a incerteza da previsão é alta. Observe que a área à extrema esquerda permanece não amostrada, pois, embora tenha alta incerteza, é corretamente previsto oferecer pouca melhoria em relação à observação mais alta.

A Figura 3 mostra um pseudocódigo da OB apresentado no trabalho (SHAHRIARI

et al., 2015). Inicialmente, o algoritmo cria um modelo probabilístico que é ajustado aos dados observados, representando a função objetivo e sua incerteza. Com base nesse modelo, uma função de aquisição é utilizada para selecionar o próximo ponto a ser avaliado e, após avaliar a função objetivo nesse ponto, o modelo é atualizado com os novos dados, refinando a representação da função objetivo. Esse processo é repetido iterativamente, buscando convergir para o ótimo global de forma eficiente, minimizando o número de avaliações necessárias.

Figura 3 – *Pseudocódigo da otimização bayesiana.*

Algoritmo 1: Otimização Bayesiana

1: **para** n = 1, 2, ..., **faça:**
2: selecione o novo x_{n+1} otimizando a função de aquisição α
$x_{n+1} = \arg\max \alpha(x; D_n)$
3: consulte a função objetivo para obter y_{n+1}
4: faça o aumento de dados $D_{n+1} = \{D_n, (x_{n+1}, y_{n+1})\}$
5: atualize o modelo estatístico
6: **fim para**

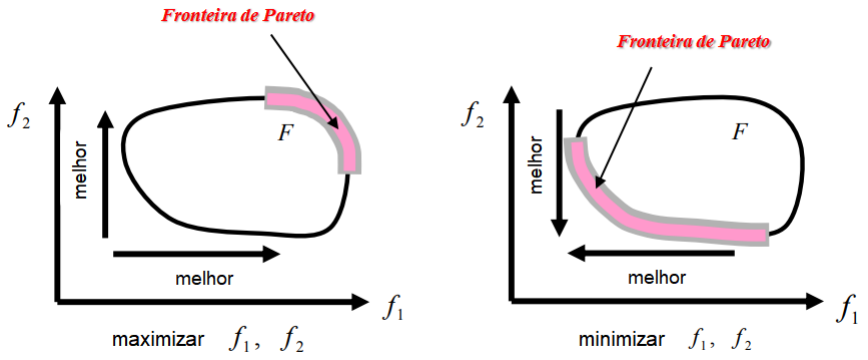
Fonte: Adaptado de (SHAHRIARI et al., 2015).

2.4 Otimalidade de Pareto

Num contexto de otimização multiobjetivo, é necessário que os objetivos sejam otimizados mutuamente para encontrar uma combinação ótima entre eles. O conceito da otimalidade de Pareto desempenha um papel crucial na otimização multiobjetivo, oferecendo uma maneira sistemática de avaliar e comparar soluções que consideram múltiplos objetivos conflitantes. O conceito define que uma solução é dada como "ótima de Pareto", se não houver outra solução viável que possa melhorar um objetivo sem piorar o outro. Esta solução representa o melhor ajuste possível entre dois objetivos conflitantes (NGATCHOU; ZAREI; EL-SHARKAWI, 2005).

Em um gráfico onde cada eixo é um objetivo e os pontos são os pares de valores para aqueles objetivos, o conjunto de todas estas soluções Pareto ótimas em um espaço de busca pode ser observado formando uma fronteira, como mostrado na Figura 4, recebendo o nome de fronteira de Pareto. Essa fronteira delimita a região onde nenhuma solução pode ser melhorada em um objetivo sem piorar o outro.

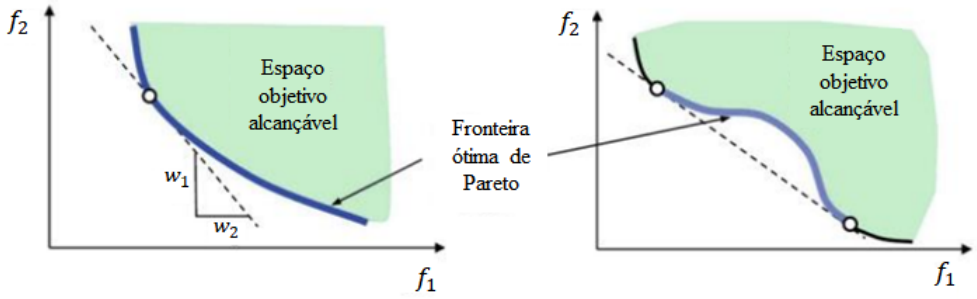
Figura 4 – *Representação da fronteira de Pareto para um problema de maximização (à esquerda) e um problema de minimização (à direita), dentro do conjunto de possibilidades F.*



Fonte: Adaptado de (NGATCHOU; ZAREI; EL-SHARKAWI, 2005)

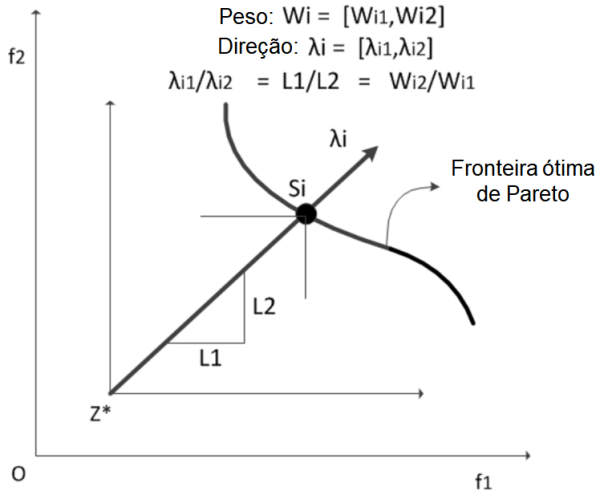
A exploração da fronteira de Pareto dentro das possibilidades em um espaço de busca auxilia na tomada de decisão sobre qual modelo escolher, com base nos critérios definidos. Existem formulações que visam encontrar valores ótimos na fronteira de Pareto, como exemplo da soma ponderada dos pesos e da formulação Chebyshev (seção 2.5). Diferente da soma ponderada dos pesos, as propriedades da formulação Chebyshev garantem que ela consegue encontrar soluções em regiões não convexas da fronteira de Pareto, como ilustram as Figuras 5 e 6:

Figura 5 – *Ilustração de duas regiões da fronteira de Pareto. O gráfico à esquerda mostra a formulação da soma ponderada dos pesos, representada como uma reta que tangencia a fronteira, onde a inclinação da reta é dada pela combinação dos pesos w_1 e w_2. O gráfico à direita mostra um exemplo onde há uma região não convexa da fronteira e, consequentemente, não é possível encontrar uma reta que tangencie a fronteira pela formulação da soma dos pesos.*



Fonte: Adaptado de (NAGY; MANSOUR; ABDELMOHSEN, 2020)

Figura 6 – *Ilustração do funcionamento da formulação Chebyshev com relação à fronteira de Pareto. Esta formulação é retratada no gráfico como um vetor que parte do ponto ótimo z* e sua inclinação é dada pela relação entre os pesos L1 e L2. Com esta formulação, é possível alcançar soluções em qualquer região da fronteira de Pareto.*



Fonte: Adaptado de (WANG; ZHANG; GUO, 2013)

2.5 Função de escalarização Chebyshev

As funções de escalarização são comumente utilizadas para converter um problema de otimização multiobjetivo em um problema de otimização de único objetivo e, combinadas com a otimização bayesiana, essas funções possuem grande utilidade na resolução de problemas de otimização multiobjetivo que são computacionalmente caros (CHUGH, 2020). Dependendo da função de escalarização escolhida, a qualidade e o número de avaliações necessárias para realizar a otimização são afetados. A função de escalarização Chebyshev é uma abordagem comum para esse tipo de problema, e é baseada na máxima distância (norma) entre a solução atual e o ponto ideal em cada objetivo. Ela é derivada da função de norma ponderada (*weighted norm*) que, por sua vez, é baseada na função soma ponderada (*weighted sum*). Matematicamente, a função de escalarização Chebyshev é definida pela Equação 2.1:

$$g = max_i[w_i|f_i - z_i^*|] \tag{2.1}$$

onde:

- g é o valor da função escalarizada para o vetor de objetivos;
- w_i é o peso associado ao objetivo i;
- f_i é o valor atual do objetivo i;

- z_i^* é o valor ideal do objetivo i.

Diferente da soma ponderada ($\sum w_i \cdot f_i$), a função Chebyshev possui a vantagem de garantir encontrar soluções na região ótima da fronteira de Pareto.

2.6 Função de entropia cruzada e de regularização L2

Para o uso da otimização multiobjetivo neste trabalho, foram escolhidos dois objetivos, sendo eles a entropia cruzada representando o erro de treino do modelo, e a regularização L2 representando a complexidade do modelo. De acordo com (ZHANG; SABUNCU, 2018), a função de perda de entropia cruzada é uma das funções de perda mais comuns para treinar modelos de redes neurais profundas, especialmente na sua versão categórica, para resolver problemas de classificação multi classes. Ela funciona calculando a discrepância entre a distribuição de probabilidade prevista pelo modelo e a distribuição real dos rótulos para uma determinada entrada, e essa diferença é chamada de erro de entropia cruzada. Dentre as características que fazem a entropia cruzada ser popular em tarefas de classificação, estão a sensibilidade a erros, penalizando fortemente previsões incorretas, e a eficiência em treinamento de modelos de aprendizado profundo, especialmente combinada com técnicas como gradiente descendente estocástico (SGD). Segundo (ACADEMY, 2022), a formulação matemática da função de entropia cruzada é dada pela Equação 2.2:

$$C = -\frac{1}{n} \sum_x [y \; ln \; a + (1 - y) \; ln(1 - a)] \tag{2.2}$$

onde:

- a é a saída do neurônio;
- n é o número total de itens de dados de treinamento;
- x são as entradas de treinamento;
- y é a saída desejada correspondente.

A regularização L2, também conhecida como decaimento de peso (*weight decay*), é uma técnica comumente utilizada em redes neurais profundas para evitar o *overfitting*[2] (LAARHOVEN, 2017), que envolve adicionar um termo de penalidade à função de perda, durante o treinamento da rede neural. A norma L2 é calculada como segue a Equação 2.3:

$$L_2 = \sum_i w_i^2 \tag{2.3}$$

[2] Condição em que um modelo se ajusta demasiadamente aos dados de treinamento, e tem dificuldade em generalizar para novos dados.

onde w_i são os pesos da camada densa da rede.

Essa penalidade L2 aplicada à função de custo faz com que os pesos sejam forçados a permanecerem pequenos, o que ajuda a prevenir o *overfitting* do modelo.

3 DESENVOLVIMENTO

Para o desenvolvimento dos experimentos, foi escolhida a linguagem de programação Python[1]. Ela é uma linguagem bem consolidada atualmente em várias áreas de desenvolvimento de software, mas se popularizou muito no campo da inteligência artificial devido às várias bibliotecas que suportam e facilitam o desenvolvimento de aplicações de IA. Como ambiente de desenvolvimento, foi escolhido o ambiente Google Colaboratory (Colab)[2], pois apresenta várias vantagens como:

- O código é executado em servidores remotos que dispõem de hardware avançado para aplicações de IA como GPUs e TPUs[3];
- dispensa a instalação de softwares e bibliotecas na sua máquina;
- implementa nativamente as versões mais atualizadas do Python e das principais bibliotecas populares;
- possui código organizado em blocos, o que permite a execução de porções individuais do código, facilita a organização e a manutenção;
- possui blocos de texto usando a formatação *markdown*, o que permite uma melhor apresentação visual e compreensão do código escrito;
- o código pode ser facilmente compartilhado e replicado para outros usuários via link;
- possui versionamento automático do código e integração com ferramentas como Google Drive e GitHub[4].

Uma desvantagem é que, na sua versão gratuita, o Colab oferece valores limitados de tempo de execução e uso de recursos computacionais (memória, disco e acesso a GPUs e TPUs), valores estes que podem ser expandidos nas modalidades pagas.

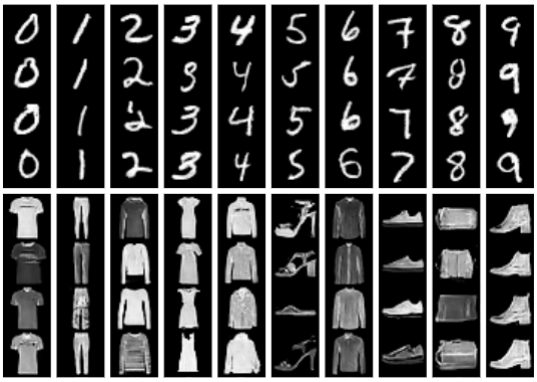
Como escolha da biblioteca, foi utilizada a biblioteca Keras, que é baseada no *framework* Tensorflow. Segundo (TENSORFLOW, 2020), "A *tf.keras* é a API de alto nível do TensorFlow para criar e treinar modelos de aprendizado profundo. Ela é usada para prototipagem rápida, pesquisa de ponta e produção".

[1] <https://www.python.org/>
[2] <https://colab.research.google.com/>
[3] GPUs e TPUs são processadores dedicados a realizar grande quantidade de operações com alto grau de paralelismo, muito utilizados em Aprendizado de Máquina.
[4] Ferramenta de repositório de código online, muito utilizada na programação.

Para o uso de ferramentas de AutoML e com o foco na otimização de hiperparâmetros, foi escolhida a biblioteca KerasTuner (O'MALLEY et al., 2019). Por ser baseada no Keras, ela possui fácil integração com as ferramentas e tipos de dados utilizados, além de implementar diversos algoritmos de otimização e permitir customizações das métricas de desempenho e funções de custo, funcionalidades estas que foram aproveitadas neste trabalho.

De posse do recurso teórico e das ferramentas de software, foram desenvolvidos notebooks para implementar a otimização de hiperparâmetros em duas bases de dados comumente utilizadas para o problema de classificação de imagens: as bases MNIST e FashionMNIST, ambas possuindo 60 mil imagens de 28x28 pixels, em escala de cinza, e com 10 classes de saída, sendo o primeiro conjunto de dados feito de imagens de dígitos, e o segundo de imagens de peças de roupa, como mostra a Figura 7. A escolha dessas bases de dados foi feita com base em sua popularidade, fácil integração por meio do Keras e, por serem bases mais simples, com imagens pequenas e em escala de cinza, são mais rápidas de treinar modelos para elas. Além disso os resultados obtidos para essas bases são amplamente conhecidos e, por isso, servem de linha base para comparação.

Figura 7 – *Exemplos de imagens das bases de dados MNIST (acima) que representa dígitos de 0 a 9 escritos à mão, e da base FashionMNIST (abaixo) que representa classes de peças de roupas.*



Fonte: (BOOTH, 2020)

Com relação aos modelos de aprendizado de máquina, foram utilizados dois tipos: uma rede neural *perceptron* de múltiplas camadas para um experimento mais rápido, e uma rede neural convolucional de várias camadas, para um experimento mais completo. Estas redes neurais serão abordadas com mais detalhes nas seções 3.2 e 3.3. Para a busca de hiperparâmetros dentro do modelo, foi utilizado um objeto que implementa o método *kt.BayesianOptimization*[5] da biblioteca KerasTuner.

[5] <https://keras.io/api/keras_tuner/tuners/bayesian/>

Nas funções de perda (*loss*), foram utilizadas duas funções: a primeira sendo a função nativa do Keras: *SparseCategoricalCrossentropy*[6], pois ela é comumente utilizada em problemas de classificação de imagens com rótulos numéricos. Foi criada uma segunda função customizada que utiliza uma abordagem multiobjetivo, segundo a escalarização Chebyshev (seção 2.5), utilizando como objetivos a entropia cruzada para representar o erro de ajuste em relação aos dados, e a regularização L2 para representar a complexidade do modelo (seção 2.6). Para cada um desses objetivos, foi associado um peso que varia no intervalo contínuo entre 0 e 1, e esses pesos foram definidos como hiperparâmetros, para observar como o algoritmo os escolheria. A formulação matemática da função customizada é dada pela Equação 3.1:

$$ChebLoss = max(w1 \cdot EC \mid w2 \cdot NL2) \tag{3.1}$$

sendo:

$$EC = -\frac{1}{n} \sum_{x} [y \; ln \; a + (1 - y) \; ln(1 - a)] \tag{3.2}$$

e

$$NL2 = \sum_{i} w_i^2 \tag{3.3}$$

onde EC é o valor do erro de entropia cruzada, NL2 é o valor da norma L2, $w1$ é o peso associado ao erro de entropia cruzada, e $w2$ é o peso associado à regularização L2. O resultado da função *ChebLoss* será o máximo entre os dois termos, mas é importante ressaltar que a função objetivo busca o valor mínimo encontrado, ou seja, o menor valor geral de *ChebLoss*.

Para a execução dos notebooks de teste da busca de hiperparâmetros, foi configurada uma busca de HPs feita com 15 *trials*[7] de 50 épocas cada, e para o treinamento do modelo foram definidas 50 épocas e uma divisão da base de dados de 60% para treinamento, e 40% para teste. Estes valores foram definidos com base em valores comumente utilizados e testes realizados, porém com a limitação de que a versão gratuita do Colab não permite tempos muitos longos de execução e isto impede que modelos possam ser treinados com valores grandes de épocas de treinamento e *trials* da busca de hiperparâmetros. Como exemplo, foi feito um notebook de teste que executa a rede neural MLP e a função customizada Chebyshev, porém com 100 *trials* de 10 épocas cada na busca de HPs, e 100 épocas de treino do modelo. Ao final, se constatou que este demorou 3 vezes mais para executar, e não obteve resultados relevantes de desempenho em relação ao mesmo notebook com as configurações de épocas citadas anteriormente.

[6] <https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy>
[7] Execuções da busca de hiperparâmetros.

3.1 Estruturação do código

O código dos notebooks pode variar de acordo com a base de dados e as funções implementadas, porém segue um mesmo padrão de organização. Por se tratar de execução em blocos, é importante que a sequência correta de execução esteja mantida. Os notebooks implementados neste trabalho seguem a seguinte estrutura:

1. Importação das bibliotecas necessárias (TensorFlow e KerasTuner) e da base de dados;
2. tratamentos na base de dados (normalização dos pixels e dos rótulos);
3. criação da função de custo customizada, caso se aplique;
4. definição da função *model_builder*, onde são definidos os hiperparâmetros a serem otimizados e a estrutura da rede, a função de custo e, por fim, o modelo compilado é passando como retorno da função;
5. instanciação do objeto *tuner*, responsável por fazer a otimização dos hiperparâmetros, recebendo o método de otimização bayesiana;
6. criação de uma função de *callback* para parar o treinamento, caso obtenha um erro muito grande;
7. execução da busca de hiperparâmetros;
8. treinamento do modelo que utiliza os hiperparâmetros selecionados automaticamente (hipermodelo);
9. avaliação do hipermodelo.

Os notebooks utilizados tiveram como base o notebook *Introduction to the Keras Tuner*[8], disponibilizado no Github do TensorFlow. Eles foram escritos e comentados em inglês, para que possam ser compreendidos e até reaproveitados em outros trabalhos globalmente, e estão disponibilizados no repositório do autor[9].

3.2 Experimentos com a rede MLP

Os primeiros experimentos foram realizados numa rede neural *perceptron* de múltiplas camadas, do inglês *MultiLayer Perceptron (MLP)*, composta somente por uma camada oculta (densa) entre a camada de entrada e de saída. Os dois hiperparâmetros utilizados

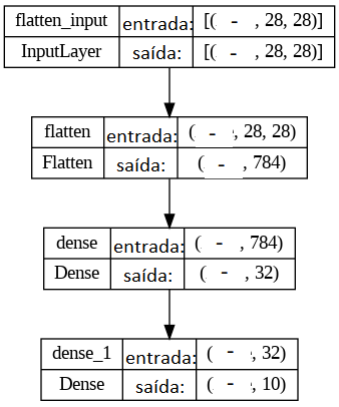
[8] <https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/keras_tuner.ipynb>

[9] <https://github.com/LucasHLirio/TCC_II>

foram a quantidade de unidades (neurônios) da camada densa, e a taxa de aprendizado do modelo. O intervalo de variação escolhido do número de unidades foi de 32 a 512, aumentando ou diminuindo num passo de 32 unidades. O intervalo escolhido para os valores da taxa de aprendizado foi entre $1 \cdot 10^{-1}$, $1 \cdot 10^{-2}$ e $1 \cdot 10^{-3}$.

No caso da rede possuir a função de perda customizada Chebyshev, há mais dois hiperparâmetros, que são os pesos associados para cada objetivo da função, sendo eles o peso W1 para o erro de entropia cruzada, e o peso W2 para a regularização L2. Ambos os pesos associados tiveram um intervalo de variação contínuo de 0 a 1. Portanto, nos notebooks que implementaram a função de perda *SparseCategoricalCrossentropy*, foram otimizados apenas 2 hiperparâmetros (unidades da camada densa e a taxa de aprendizado), enquanto que nos notebooks que implementaram a função customizada Chebyshev foram adicionados os dois hiperparâmetros relacionados à função, totalizando 4 hiperparâmetros a serem otimizados. A estrutura da rede MLP gerada é exibida na Figura 8:

Figura 8 – *Estrutura da rede MLP. Cada bloco representa uma camada, sendo a primeira a camada de entrada e a última a camada de saída. Nas linhas de cada bloco estão representados o nome e o formato (shape) de entrada e saída da camada.*



Fonte: Produzido pelo autor.

3.3 Experimentos com a rede CNN

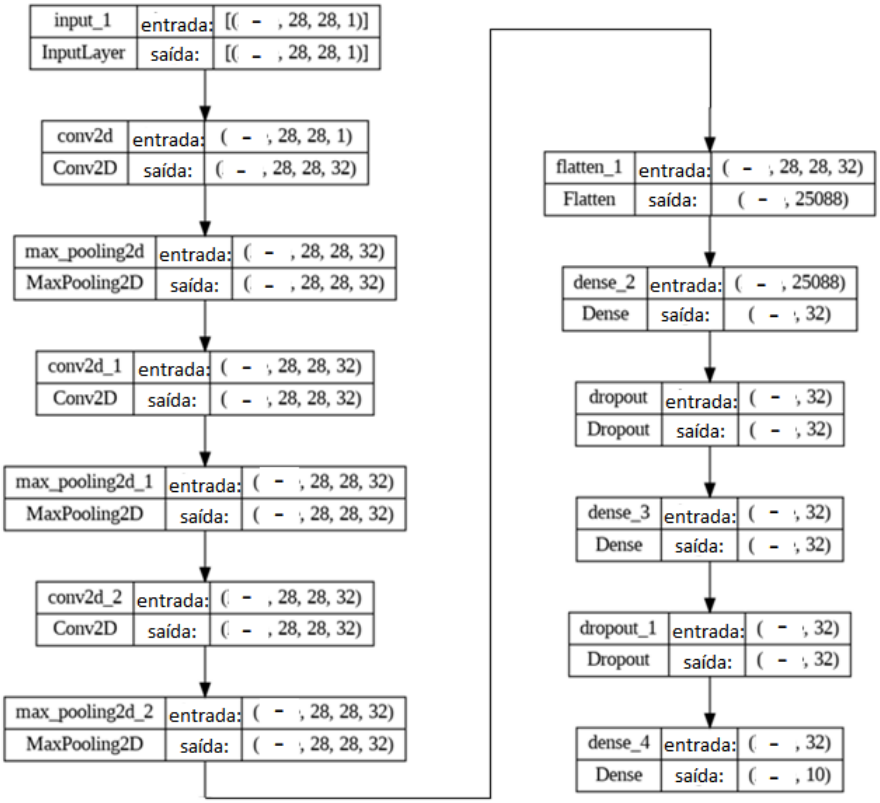
Para testar os métodos desenvolvidos em uma rede neural profunda, foi utilizada uma rede neural convolucional, do inglês *Convolutional Neural Network (CNN)*. A escolha se deu porque as CNNs tem demonstrado uma grande eficiência para resolução de problemas de reconhecimento de imagens. Portanto, foram adicionadas camadas convolucionais ao modelo anterior, obtendo a seguinte estrutura:

- Camada de entrada;

- 3 pares de uma camada de convolução (*Conv2D*) seguida de uma camada de pooling (*MaxPooling2D*), nesta ordem;
- uma camada *flatten*;
- 2 pares de uma camada densa, seguida por uma camada de *Dropout*;
- camada de saída.

Para melhor visualização da estrutura, ela também é apresentada no esquema gerado pelo método *plot_model* do Keras, como é mostrado na Figura 9:

Figura 9 – *Estrutura da rede CNN. A imagem foi dividida manualmente a fim de caber melhor na página.*



Fonte: Produzido pelo autor.

As camadas de convolução recebem este nome, porque realizam uma operação de convolução na imagem para extrair informações relevantes dela. A camada de *pooling* é utilizada normalmente após as camadas de convolução, e sua função é reduzir as dimensões dos parâmetros a serem treinados sem perder as principais características, e tornar a rede invariante a transformações geométricas (NOÉ, 2021). Por fim, a camada *flatten* reduz a matriz gerada pelas camadas anteriores para um vetor unidimensional, e a camada de *dropout* é responsável por remover alguns neurônios da rede a fim de prevenir o *overfitting*.

Na questão dos hiperparâmetros, foram mantidos os dois da rede MLP (unidades na camada densa e taxa de aprendizado). Para cada camada convolucional, foi adicionados os hiperparâmetros:

- Unidades na camada de convolução - intervalo de 32 a 512, com passo 32;
- tamanho do *kernel* (filtro de convolução) - intervalo de 3 a 5, com passo 1;
- tamanho do *stride* (deslocamento do filtro) -intervalo de 1 a 5, com passo 1.

Para as camadas de *pooling* também foram definidos os hiperparâmetros de *stride*, seguindo a mesma definição para a camada de convolução, e o tamanho do filtro de *pooling*, variando de 2 a 4, com passo 1. As camadas densas, cada uma recebeu seu HP de número de unidades, também variando de 32 a 512 com passo 32, e também uma função de ativação comum entre elas, sendo a função "relu" (*Rectified Linear Unit*), "tanh" (*Hyperbolic Tangent*) ou "sigmoid" (*Sigmoid*). Por fim, as camadas de *dropout* tiveram como HP cada uma, a sua taxa de *dropout*, variando de 0 a 0.5, com passo 0.1, e a função otimizadora do modelo foi escolhida entre "adam" (*Adaptive Moment Estimation*), "SGD" (*Stochastic Gradient Descent*) e "rmsprop"(*Root Mean Square Propagation*). Totalizando, foram 22 hiperparâmetros otimizados para os notebooks que utilizam a função de perda *SparseCategoricalCrossentropy*, e 24 hiperparâmetros para os notebooks que utilizam a função de perda Chebyshev, pois há a adição dos dois pesos associados à função.

3.4 Testes com pesos fixos para a função de perda Chebyshev

Para observar o impacto dos objetivos dentro da função de perda Chebyshev, foi reaproveitado um notebook utilizando a rede neural MLP e a base FashionMNIST, porém os pesos associados aos dois objetivos da função de perda (entropia cruzada e regularização L2) deixaram de ser escolhidos pela busca de hiperparâmetros, e passaram a ser valores fixos de 0 ou 1, combinados entre si. Logo, foram feitas 4 execuções com as combinações de pesos: [0,0], [0,1], [1,0] e [1,1], e os resultados obtidos são apresentados no Capítulo 4.

3.5 Geração do conjunto Pareto ótimo

A fim de observar graficamente a geração do conjunto Pareto ótimo (seção 2.4), foi feito um desenvolvido um outro código para gerar um conjunto de modelos utilizando combinações aleatórias dos pesos da função de perda multiobjetivo Chebyshev, e a rede neural MLP dos exemplos anteriores. Para ter acesso aos valores dos objetivos após cada execução, as funções de entropia cruzada e regularização L2 foram implementadas separadamente através de funções simples, e foram chamadas como métricas do modelo,

através do método *model.compile*. Após isso foram feitos dois *loops* de treinamento: um utilizando a função de perda Chebyshev, e outro utilizando a função soma dos pesos, para efeito de comparação entre ambas. Cada *loop* foi executado 50 vezes, ou seja, foram treinados 50 modelos diferentes, e os pares de pesos foram gerados aleatoriamente, sendo um peso o complemento de 1 do outro.

Ressalta-se que esta etapa do trabalho foi feita com o intuito de exibir a fronteira de Pareto e medir o gasto de tempo necessário para gerar um conjunto de modelos candidatos, mas o AutoML é responsável por escolher automaticamente o melhor modelo, sem que seja necessária a realização de todos os passos aqui descritos.

4 RESULTADOS E DISCUSSÃO

Após a execução dos notebooks e a realização de variados testes, os resultados foram coletados e organizados, a fim de validar os assuntos discutidos na fundamentação teórica, e gerar conclusões sobre os experimentos.

4.1 Resultados da busca de hiperparâmetros

Primeiramente, os resultados dos experimentos feitos com a rede neural MLP (seção 3.2), foram agrupados na Tabela 1 a seguir:

Tabela 1 – Tabela comparativa dos modelos de rede neural MLP.

Modelo	1	2	3	4
Base de dados	MNIST	MNIST	FashionMNIST	FashionMNIST
Função de perda	CustomCheb	CrossEnt	CustomCheb	CrossEnt
Peso W1	0.9	-	0.5	-
Peso W2	0.0	-	0.0	-
Unidades da camada densa	512	256	384	448
Taxa de aprendizado	0.001	0.001	0.001	0.001
Perda de teste	0.1450	0.1471	0.2839	0.5861
Acurácia de teste	98.13 %	97.87%	88.51 %	88.78%
Tempo da busca de HPs (min)	38.68	16.43	64.17	20.53
Tempo de treino (min)	9.38	5.06	7.95	4.30

Fonte: Produzido pelo autor.

Nesta comparação foram usados 4 modelos, sendo eles combinações entre as duas bases de dados (MNIST e FashionMNIST) e as duas funções de perda: *CategoricalCrossEntropy* (CrossEnt) e Chebyshev customizada (CustomCheb). A tabela apresenta os dois hiperparâmetros utilizados, além dos dois HPs de pesos da função de perda Chebyshev, sendo eles o número de unidades na camada densa/oculta, a ,taxa de aprendizado do modelo, o peso W1 associado ao erro (entropia cruzada) e o peso W2 associado à complexidade (regularização L2).

Nas quatro últimas linhas, há os valores de perda e da porcentagem de acurácia obtidos na avaliação do modelo, e por último, os tempos de busca dos hiperparâmetros e de treinamento do modelo, ambos em minutos.

Seguindo a mesma organização, porém com mais hiperparâmetros, a Tabela 2 mostra a comparação da execução dos modelos que usaram a rede neural convolucional

(CNN), abordada na seção 3.3.

Tabela 2 – Tabela comparativa dos modelos de rede neural convolucional. Alguns termos foram mantidos em inglês por serem popularmente usados.

Modelo	1	2	3	4
Base de dados	MNIST	MNIST	FashionMNIST	FashionMNIST
Função de perda	CrossEnt	CustomCheb	CrossEnt	CustomCheb
Un. camada de convolução 1	352	128	128	512
Un. camada de convolução 2	352	96	384	192
Un. camada de convolução 3	192	128	192	128
Tam. kernel conv. 1	5	5	5	4
Tam. kernel conv. 2	5	3	4	4
Tam. kernel conv. 3	4	5	3	4
Tam. kernel pooling 1	2	2	4	3
Tam. kernel pooling 2	4	2	2	2
Tam. kernel pooling 3	3	2	3	3
Deslocamento kern. conv. 1	3	1	1	4
Deslocamento kern. conv. 2	1	2	5	3
Deslocamento kern. conv. 3	1	5	1	5
Deslocamento kern. pool. 1	1	2	3	3
Deslocamento kern. pool. 2	2	2	2	4
Deslocamento kern. pool. 3	4	5	5	4
Função de ativação	tanh	tanh	relu	relu
Unidades da camada densa 1	64	128	416	480
Unidades da camada densa 2	288	128	512	384
Taxa de dropout 1	0.1	0.4	0.2	0.0
Taxa de dropout 2	0.0	0.5	0.0	0.0
Função otimizadora	SGD	adam	adam	rmsprop
Taxa de aprendizado	0.1	0.001	0.001	0.001
Peso W1	-	0.7	-	0.8
Peso W2	-	0.0	-	0.0
Perda de teste	0.2710	0.0970	0.7022	1.510
Acurácia de teste	99.34 %	95.79 %	90.17%	82.90 %
Tempo da busca de HPs (min)	46.68	101.85	48.58	80.52
Tempo de treino (min)	38.38	13.30	12.38	14.15

Fonte: Produzido pelo autor.

Pela análise das tabelas, os modelos para a base de dados MNIST obtiveram uma acurácia maior que os da base FashionMNIST, mostrando que a acurácia está fortemente atrelada à base de dados utilizada, teoria esta que se confirma também em uma terceira

base de dados, que será abordada na seção 4.4. A utilização, ou não, da função de perda customizada Chebyshev teve pouco impacto na acurácia da rede MLP, porém na rede CNN houve um decréscimo na porcentagem. Isto pode se dar ao fato de que, por se tratar de uma função customizada, ela pode não estar otimizada com relação à uma função nativa da ferramenta, causando a discrepância no valor. Outra observação que envolve a função de perda Chebyshev é que, para todos os notebooks, a busca de hiperparâmetros sempre escolheu o peso associado à regularização L2 (W2) como sendo zero. Isso implica que a regularização L2 não estava causando impacto significativo para esses modelos, e isto levou à realização do teste com os pesos fixos (seção 3.4) para avaliar os impactos de cada peso.

Com relação aos hiperparâmetros escolhidos pelo algoritmo, pode-se notar que há uma preferência pela taxa de aprendizado valendo $1 \cdot 10^{-3}$, e para os modelos com a função Chebyshev houve preferência de valores entre 0,7 e 0,9 para o peso associado ao erro de entropia cruzada (W1), e zero para o peso associado à regularização L2 (W2). Para os valores de unidades em camadas densas e convolucionais, não houve preferência por parte do algoritmo mas, por terem um intervalo de valores grande, seriam necessárias muitas tentativas isoladas a fim de observar se há a convergência para um valor comum entre os modelos. Nos modelos que utilizam a CNN, também houve uma variação nos valores dos HPs que delimitam os tamanhos dos filtros e deslocamentos relacionados às camadas de convolução e pooling. Novamente, essa não convergência para um valor comum pode ser explicada pela grande quantidade de HPs associados pois, para cada hiperparâmetro adicionado, multiplica-se o número de combinações de modelos possíveis, o que faz com que o tempo da busca dos HPs seja cada vez mais demorado.

Nos modelos CNN, a função de ativação das camadas densas foi escolhida como a "tanh" (*Hyperbolic Tangent*) para a base MNIST, e "relu" (*Rectified Linear Unit*) para a base FashionMNIST. No HP da função otimizadora do modelo, houve uma pequena preferência para a função "adam"(*Adaptive Moment Estimation*), porém as funções "SGD"(*Stochastic Gradient Descent*) e "rmsprop"(*Root Mean Square Propagation*) também foram escolhidas, mostrando que é necessário fazer vários testes para se certificar se há uma função predileta para os modelos implementados.

Por fim, a análise dos tempos de execução nos mostra que a função de perda Chebyshev possui um tempo de busca dos hiperparâmetros cerca de duas vezes maior, comparado ao tempo de busca dos HPs com a função de perda *Categorical Crossentropy*, reiterando o argumento de que a função Chebyshev, por ser customizada, pode não ser tão otimizada quanto a outras implementadas nativamente pela biblioteca do Keras. Sobre o tempo de treinamento, não há muito o que se concluir, visto que ele depende diretamente do tamanho da base de dados, da máquina onde está sendo executado, e do número de épocas de treinamento. Visto que os recursos de hardware oferecidos pelo Colab são bem

variáveis, logo os tempos de treino e busca de HPs podem variar a cada execução.

Por fim, os valores de acurácia foram satisfatórios na maioria dos casos, mas um valor que obteve um grande destaque foi a acurácia de teste para o modelo 1 do tipo CNN (99,34%). Segundo o ranking do site Papers With Code[1], que reúne dados de modelos apresentados em artigos, o modelo 1 do tipo CNN ocuparia a 17^a posição global[2], com o primeiro lugar sendo ocupado pelo modelo "Branching/Merging CNN + Homogeneous Vector Capsules", possuindo uma acurácia de 99,87% para a base de dados MNIST. Este é um resultado satisfatório, visto que o modelo aqui apresentado difere de 0,53 pontos percentuais do melhor modelo atual para este conjunto de dados.

4.2 Resultados do teste com pesos fixos

Com o intuito de avaliar a influência dos pesos associados aos objetivos da função de perda Chebyshev no desempenho do modelo, foi feito um notebook de teste que utiliza os dois pesos fixos em valores de 0 ou 1, combinados entre si, resultando em quatro execuções (seção 3.4). A busca de hiperparâmetros possui 20 *trials* de 50 épocas cada, e o modelo é treinado com 50 épocas e 40% da base de dados usada para validação, gerando os resultados apresentados na Tabela 3:

Tabela 3 – Testes com pesos fixos para a função Chebyshev.

Modelo	1	2	3	4
Wheights (a,b)	**(0,0)**	**(0,1)**	**(1,0)**	**(1,1)**
Dense units	320	160	352	128
Learning rate	0.001	0.01	0.001	0.001
Test loss	0.00	0.12	0. 1706	1.7547
Test accuracy	7.77 %	9.45 %	97.87 %	71.64 %

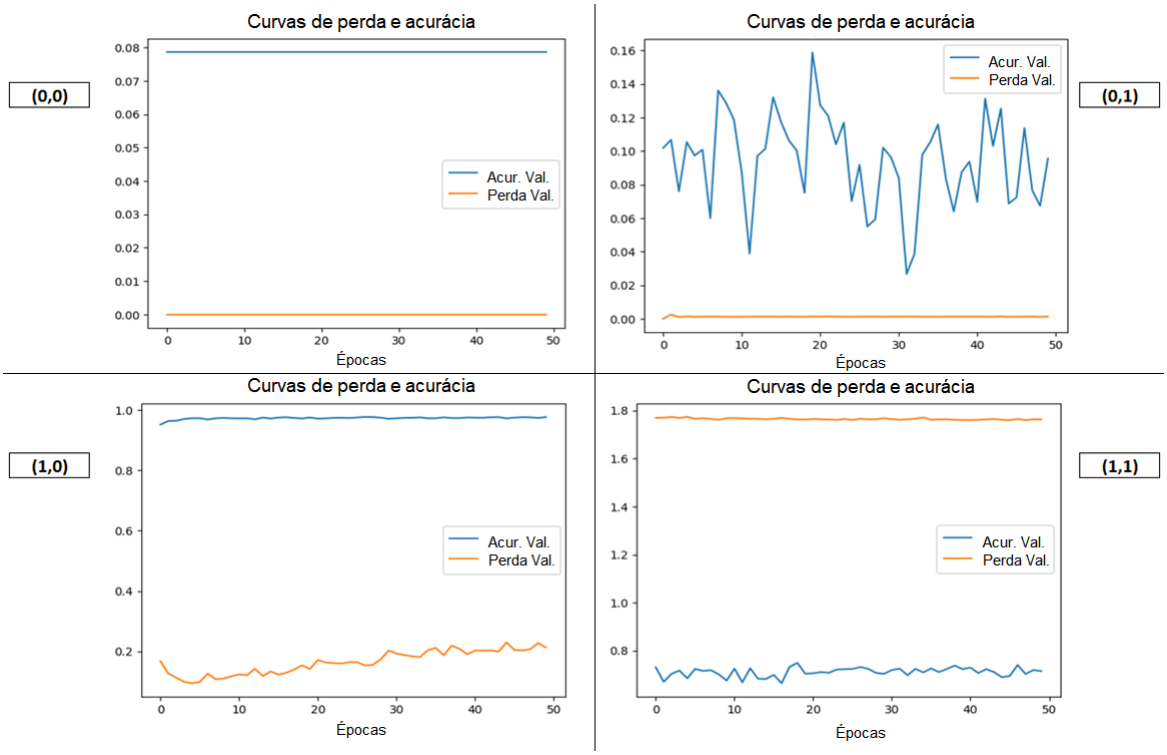
Fonte: Produzido pelo autor.

A Tabela 3 apresenta em cada modelo a tupla de pesos (a,b), onde "a" é o peso associado ao erro de entropia cruzada, e "b" é o peso associado à regularização L2. Assim pode-se notar que, como esperado, o modelo 1 obteve um valor de perda (*loss*) zerado, visto que a saída dos dois objetivos era multiplicada por zero, e consequentemente o modelo obteve uma acurácia extremamente baixa, algo que aconteceu também para o modelo 2, devido ao fato de não estar sendo considerado o valor de erro das amostras. O modelo 3 comprova que uma configuração próxima a 1 para o peso "a", e próxima a zero para o peso "b", é a mais ideal, e foi exatamente o que mostrou o resultado das buscas de

[1] <https://paperswithcode.com/sota/image-classification-on-mnist?metric=Accuracy>
[2] Dados considerando a data de acesso em 15/02/2024.

hiperparâmetros na seção anterior, comprovando a eficácia do método. O modelo 4 mostra que ambos os objetivos com peso 1 não alcança uma acurácia tão boa, e o valor de *loss* é 10 vezes mais alto que o do modelo 3, reforçando que o peso da regularização L2 deve ser aproximado de zero, como foi obtido na busca automática dos HPs. Na Figura 10 é possível visualizar as curvas de perda e acurácia para cada configuração:

Figura 10 – *Curvas de perda (laranja) e acurácia (azul) para cada uma das quatro combinações de pesos fixos da função objetiva customizada. Pelos gráficos é possível notar que, para este caso, o par de pesos (1,0) é o que obtém um melhor desempenho, o que condiz com os resultados encontrados pela busca automática de hiperparâmetros.*



Fonte: Produzido pelo autor.

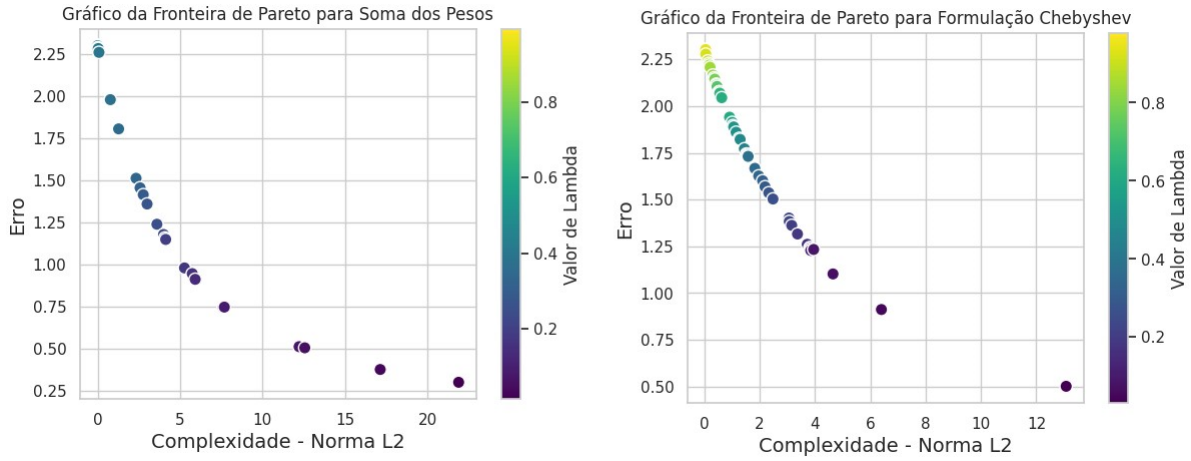
4.3 Resultados da fronteira de Pareto

Os resultado da execução do notebook mencionado na seção 3.5, são exibidos na Figura 11.

Pela análise dos gráficos pode-se notar que, para este problema em específico, as duas formulações produziram curvas bem semelhantes. Por se tratar de um programa onde muitas redes são criadas e treinadas, o custo de tempo é alto, como por exemplo a duração de cerca de 2 horas para executar os *loops* para gerar os dois gráficos, sendo que cada modelo foi treinado apenas com 10 épocas cada. Para obter resultados mais robustos,

seria necessário executar o programa com um número maior de modelos, e realizar testes com modelos mais avançados e bases de dados mais complexas. Ressalta-se, porém, que a formulação Chebyshev garante o encontro de soluções na região ótima da fronteira de Pareto (seção 2.5), onde a função de soma dos pesos não é capaz de encontrar, justificando a utilização da formulação neste trabalho. Também não foram utilizados métricas de espalhamento ou outras medidas de comparação entre o gráfico das duas formulações, pois o foco era gerar uma análise visual e mostrar a inviabilidade de tempo de execução quando as soluções são procuradas manualmente, e não pela busca automática de HPs.

Figura 11 – *Visualização da fronteira de Pareto para os modelos gerados, onde Lambda é o peso associado à norma L2.*



Fonte: Produzido pelo autor.

4.4 Desafios encontrados

O primeiro desafio encontrado no desenvolvimento deste trabalho, foi uma série de problemas tentando implementar a busca de arquiteturas neurais Neural Architecture Search (NAS), assunto este tratado no artigo (HE; ZHAO; CHU, 2021). Obtiveram-se, por exemplo, problemas com relação à bibliotecas desatualizadas e incompatibilidade entre tipos de dados utilizados nos programas, o que gerava muitos erros no código. Estes problemas levaram à mudança de abordagem do NAS para a otimização de hiperparâmetros.

Outra dificuldade encontrada foi o limite de hardware próprio para execução de programas de aprendizado de máquina, visto que demandam um alto poder computacional. Também os recursos limitados do Colab, impediram que fossem feitos testes mais demorados (com um maior número de *trials* da busca de HPs, e épocas de treinamento), testes com modelos mais robustos, ou até em bases de dados maiores, que demandam mais memória e capacidade de disco para serem processadas. Esta limitação de recursos de hardware,

também foi uma causa impedidora da tentativa de implementar a busca de hiperparâmetros em um caso real, para otimizar uma rede CNN de reconhecimento facial de pessoas vestindo máscaras (NOÉ, 2021), que era um dos objetivos finais deste trabalho.

Foram feitos testes com uma terceira base de dados CIFAR10[3], porém para todas as configurações testadas, a acurácia foi muito baixa (por volta de 40 a 45%), mesmo em testes com a rede CNN. Não se sabe ao certo o porquê, mas visto que, para uma mesma arquitetura, o desempenho do modelo é atrelado à base de dados utilizada, pode ser que essa base não se comporte bem para o modelo proposto, fazendo necessária a realização de testes com outros outros modelos e hiper parâmetros que se adaptem melhor a este conjunto de dados.

[3] <https://keras.io/api/datasets/cifar10/>

5 CONCLUSÃO E TRABALHOS FUTUROS

Através deste trabalho, conclui-se que a otimização de hiperparâmetros é um etapa crucial no processo do AutoML, campo de estudo da inteligência artificial que propõe, de forma automatizada, construir arquiteturas neurais eficientes, e buscar configurações que obtenham um desempenho satisfatório, dispensando a necessidade de um especialista em aprendizado de máquina para tal. Através de métodos de otimização, como a otimização bayesiana, podem ser encontrados bons hiperparâmetros com uma exploração eficiente do espaço de busca e, utilizando-se de uma função de perda multiobjetivo, como a escalarização Chebyshev, é possível encontrar combinações de parâmetros que gerem modelos pertencentes ao conjunto Pareto ótimo.

Como trabalhos futuros, sugere-se que seja implementada a otimização de hiperparâmetros em um modelo mais robusto e que resolva algum caso de uso real, como no caso do trabalho de (NOÉ, 2021), para mostrar que a otimização de hiperparâmetros pode gerar modelos melhores, ou tão bons quanto os já existentes, porém beneficiando-se da configuração automática do modelo. Uma segunda sugestão é abordar e implementar o problema da busca de arquiteturas neurais (NAS), em conjunto com a otimização de hiperparâmetros para obter modelos otimizados tanto na sua arquitetura, quanto na sua configuração. Outras sugestões podem ser obtidas ao estudar formas de contornar os problemas descritos na seção 4.4.

Desta maneira, conclui-se que, juntamente com a Inteligência Artificial, o AutoML está em constante evolução, e traz diversos benefícios na construção de modelos cada vez mais completos e complexos, que podem impactar positivamente a vida das pessoas e o meio em que vivem.

REFERÊNCIAS

ACADEMY, D. S. *Deep Learning Book*. Data Science Academy, 2022. Disponível em: <https://www.deeplearningbook.com.br/cross-entropy-cost-function/>. Acesso em: 13 fev 2024. Citado na página 22.

BAI, Q. et al. Object detection recognition and robot grasping based on machine learning: A survey. *IEEE access*, IEEE, v. 8, p. 181855–181879, 2020. Citado na página 12.

BISHOP, C. Pattern recognition and machine learning. *Springer google schola*, Springer, v. 2, p. 5–43, 2006. Citado na página 15.

BOOTH, S. *Bayes-Probe: Distribution-Guided Sampling for Prediction Level Sets*. [S.l.]: ResearchGate, 2020. <https://www.researchgate.net/figure/Inferred-high-classification-samples-for-CLEVR-top-MNIST-middle-and-Fashion-MNIST_fig3_339471054>. Acesso em: 13 fev 2024. Citado na página 25.

CHUGH, T. Scalarizing functions in bayesian multiobjective optimization. In: IEEE. *2020 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.], 2020. p. 1–8. Citado na página 21.

ELSKEN, T.; METZEN, J. H.; HUTTER, F. Neural architecture search: A survey. *The Journal of Machine Learning Research*, JMLR. org, v. 20, n. 1, p. 1997–2017, 2019. Citado na página 16.

FRAZIER, P. I. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018. Citado 2 vezes nas páginas 17 e 18.

GARCÍA, S.; LUENGO, J.; HERRERA, F. *Data preprocessing in data mining*. [S.l.]: Springer, 2015. v. 72. Citado na página 16.

GLAZ, A. L. et al. Machine learning and natural language processing in mental health: systematic review. *Journal of Medical Internet Research*, JMIR Publications Toronto, Canada, v. 23, n. 5, p. e15708, 2021. Citado na página 15.

HE, X.; ZHAO, K.; CHU, X. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, Elsevier, v. 212, p. 106622, 2021. Citado 3 vezes nas páginas 15, 16 e 37.

JONES, D. R.; SCHONLAU, M.; WELCH, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, Springer, v. 13, p. 455–492, 1998. Citado na página 17.

KARMAKER, S. K. et al. Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys (CSUR)*, ACM New York, NY, v. 54, n. 8, p. 1–36, 2021. Citado na página 16.

KHAN, A. A.; LAGHARI, A. A.; AWAN, S. A. Machine learning in computer vision: a review. *EAI Endorsed Transactions on Scalable Information Systems*, v. 8, n. 32, p. e4–e4, 2021. Citado na página 15.

KHAN, W. et al. A survey on the state-of-the-art machine learning models in the context of nlp. *Kuwait journal of Science*, v. 43, n. 4, 2016. Citado na página 12.

LAARHOVEN, T. V. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017. Citado na página 22.

NAGY, M.; MANSOUR, Y.; ABDELMOHSEN, S. Multi-objective optimization methods as a decision making strategy. *Int. J. Eng. Res. Technol.(IJERT)*, v. 9, n. 3, p. 516–522, 2020. Citado na página 20.

NAHID, A.-A.; KONG, Y. et al. Involvement of machine learning for breast cancer image classification: a survey. *Computational and mathematical methods in medicine*, Hindawi, v. 2017, 2017. Citado na página 12.

NGATCHOU, P.; ZAREI, A.; EL-SHARKAWI, A. Pareto multi objective optimization. In: IEEE. *Proceedings of the 13th international conference on, intelligent systems application to power systems*. [S.l.], 2005. p. 84–91. Citado 2 vezes nas páginas 19 e 20.

NOÉ, Í. T. Redes neurais convolucionais aplicadas ao reconhecimento facial em indivíduos com máscara. *Trabalho de Conclusão de Curso – Instituto de Ciências Exatas e Aplicadas -Universidade Federal de Ouro Preto*, 2021. Citado 3 vezes nas páginas 29, 38 e 39.

O'MALLEY, T. et al. *KerasTuner*. 2019. <https://github.com/keras-team/keras-tuner>. Citado na página 25.

RASCHKA, S.; PATTERSON, J.; NOLET, C. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, MDPI, v. 11, n. 4, p. 193, 2020. Citado na página 15.

SARKER, I. H. Ai-based modeling: Techniques, applications and research issues towards automation, intelligent and smart systems. *SN Computer Science*, Springer, v. 3, n. 2, p. 158, 2022. Citado na página 15.

SHAHRIARI, B. et al. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, IEEE, v. 104, n. 1, p. 148–175, 2015. Citado 3 vezes nas páginas 17, 18 e 19.

TENSORFLOW. *Keras | Tensorflow Core*. TensorFlow, 2020. Disponível em: <https://www.tensorflow.org/guide/keras?hl=pt-br>. Citado na página 24.

WANG, R.; ZHANG, T.; GUO, B. An enhanced moea/d using uniform directions and a pre-organization procedure. In: . [S.l.: s.n.], 2013. p. 2390–2397. ISBN 978-1-4799-0453-2. Citado na página 21.

ZHANG, Z.; SABUNCU, M. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, v. 31, 2018. Citado na página 22.