57º SBPO
SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
Gramado • RS
DE 5 A 9 DE OUTUBRO DE 2025

Avaliação do Uso da Ferramenta irace para a Otimização de Hiperparâmetros de Redes Neurais Convolucionais

Camilla Rodrigues de M. Fernandes, Pedro Nuno de S. Moura, Adriana C. F. Alvim
Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Av. Pasteur 458, Urca – Rio de Janeiro, RJ – Brasil
camilla.r.fernandes@edu.unirio.br, {pedro.moura, adriana}@uniriotec.br

RESUMO

A Aprendizagem Profunda é um campo da Inteligência Artificial que se ocupa das redes neurais artificiais e que tem sido empregada em diversas aplicações e problemas, especialmente em tarefas de Visão Computacional. O treinamento de uma rede neural, etapa fundamental, pode tomar uma quantidade de tempo considerável e demandar bastante esforço computacional, a depender da quantidade de dados e do tamanho da rede. Tipicamente, o seu treinamento também depende da configuração de valores dos seus hiperparâmetros, sendo comumente necessário realizá-lo diversas vezes, considerando diferentes configurações, a fim de escolher aquela que leva a rede ao melhor desempenho. Este trabalho propõe e avalia o uso da ferramenta irace, adotada na área de meta-heurísticas, no processo de otimização dos valores de hiperparâmetros de uma rede neural convolucional treinada para o *dataset* CIFAR-10, realizando experimentos computacionais e comparando os resultados obtidos com aqueles oriundos da aplicação da ferramenta Optuna.

PALAVRAS CHAVE. Aprendizagem Profunda, Redes Neurais Convolucionais, Otimização de Hiperparâmetros, irace, Optuna.

Tópicos (PO e IA, Metaheurísticas)

ABSTRACT

Deep Learning is a field of Artificial Intelligence that deals with artificial neural networks and has been used in various applications and problems, especially in Computer Vision tasks. Training a neural network, a fundamental step, can take a considerable amount of time and demand a lot of computational effort, depending on the amount of data and the size of the network. Typically, its training also depends on the configuration of its hyperparameter values, and it is often necessary to perform it several times, considering different configurations, in order to choose the one that leads the network to the best performance. This work proposes and evaluates the use of the irace tool, adopted in the area of metaheuristics, in the process of optimizing the hyperparameter values of a convolutional neural network trained for the CIFAR-10 dataset, carrying out computational experiments and comparing the results obtained with those from the application of the Optuna tool.

KEYWORDS. Deep Learning, Convolutional Neural Networks, Hyperparameter Optimization, irace, Optuna.

Topics (PO and IA, Metaheuristics)

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

1. Introdução

As redes neurais artificiais (RNAs) fazem parte da área chamada de Aprendizagem Profunda (AP) e consistem em modelos computacionais inspirados pelo funcionamento do cérebro humano que contêm camadas de unidades de processamento chamadas de neurônios artificiais. Esses modelos são capazes de aprender características (*features*) automaticamente a partir de grandes volumes de dados, sem a necessidade de implementação de algoritmos extratores de características. As RNAs conseguem reconhecer padrões complexos e não lineares nos dados e se adaptar a novas informações. Existem também arquiteturas de redes neurais que permitem que, uma vez aprendidas as características que representam os dados de treinamento, sejam gerados dados artificiais como imagens, textos e músicas, por meio de modelos de Inteligência Artificial (IA) generativa. As RNAs revolucionaram diversos campos da IA, como, por exemplo, visão computacional, processamento de linguagem natural, robótica e sistemas de recomendação [Aggarwal, 2023].

Uma RNA é composta de diversas camadas de neurônios, em que cada neurônio possui entradas, pesos e uma função de ativação para cálculo da sua saída. Inicialmente, os pesos de uma RNA são inicializados segundo alguma distribuição de probabilidade relevante e são ajustados ao longo do processo chamado de "treinamento" [Aggarwal, 2023].

O sucesso de uma RNA depende de como ela é montada e treinada. O projeto de uma RNA em si e da sua forma de treinamento dependem de uma série de configurações chamadas de "hiperparâmetros" que são definidas *a priori* pelo projetista da rede. Uma etapa fundamental do processo de treinamento de uma RNA é a escolha dos valores dos hiperparâmetros que levem ao melhor desempenho da rede, a depender da métrica de avaliação adotada. Essa etapa recebe o nome de afinação (*tuning*) dos hiperparâmetros.

Durante a etapa de afinação, alguns dos valores dos hiperparâmetros são modificados para melhorar o resultado final ou o tempo até a convergência. Alguns hiperparâmetros podem influenciar em como a rede vai convergir para uma saída aceitável (ou mesmo não convergir), se vai conseguir escapar de ótimos locais de baixa qualidade e se vai se sobreajustar (*overfit*) aos dados de treinamento, em que a rede possui um ótimo desempenho para os dados de treinamento, mas falha em generalizar para os dados de teste.

A cada mudança de valores de hiperparâmetros, um novo treinamento precisa ser executado no modelo. Isso constitui um grande desafio, pois cada treinamento pode levar uma quantidade de tempo considerável e demandar bastante esforço computacional, a depender do problema, da quantidade de dados e do tamanho da rede, enquanto as combinações de valores de hiperparâmetros crescem rapidamente no número de hiperparâmetros e na variação dos valores de cada um.

Normalmente, o projetista da RNA costuma empregar uma abordagem empírica em que testa diversos valores de hiperparâmetros até conseguir uma combinação que leve a um resultado satisfatório. Uma alternativa para essa abordagem empírica consiste em explorar todas as combinações de valores dos hiperparâmetros considerados, denominada de *grid search*, mas que se mostra muitas vezes infactível.

Portanto, o objetivo deste trabalho é verificar a viabilidade do uso da ferramenta irace [López-Ibáñez et al., 2016] para realizar a otimização dos hiperparâmetros de RNAs. O irace é um pacote em R para otimização e é considerado o estado da arte em termos de desempenho no projeto de algoritmos de otimização combinatória [Bezerra et al., 2020]. Para tal, realizou-se uma avaliação experimental da ferramenta irace na otimização dos hiperparâmetos de uma RNA aplicada para a base de dados CIFAR-10 [Krizhevsky e Hinton, 2009], comparando-se os resultados obtidos com aqueles gerados pelo uso do *framework* Optuna [Akiba et al., 2019], desenvolvido especificamente para a otimização de hiperparâmetros de RNAs.

O restante deste artigo está estruturado da seguinte forma: a Seção 2 apresenta a funda-

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

mentação teórica; já a Seção 3 trata dos trabalhos relacionados; a Seção 4 aborda os experimentos computacionais conduzidos e os resultados obtidos; e, por fim, a Seção 5 apresenta as considerações finais e os trabalhos futuros.

2. Fundamentação Teórica

Esta seção apresenta os principais conceitos necessários ao entendimento deste trabalho.

2.1. Aprendizagem Profunda

A Aprendizagem de Máquina (AM) é um campo da IA que lida com algoritmos que são capazes de aprender a partir de dados. Mais formalmente, tal capacidade de aprender pode ser definida da seguinte forma [Mitchell, 1997]: "Diz-se que um programa de computador aprende com a experiência 'E' com relação a algumas classes de tarefas 'T' e possui desempenho medido por 'P', se seu desempenho nas tarefas em 'T', conforme medido por 'P', melhora com a experiência 'E'." Já a AP é um subcampo da AM que busca se inspirar o funcionamento do cérebro humano para processar dados por meio das RNAs. Os métodos de AP podem ser utilizados para realizar aprendizado supervisionado, não supervisionado, autossupervisionado e por reforço.

Em um contexto de aprendizado supervisionado, na etapa de treinamento, cada exemplo da base de dados é inserido na RNA e a saída associada é calculada. Após cada rodada de treinamento, definida por uma quantidade pré-estabelecida de exemplos, os pesos da RNA são ajustados para que as suas saídas preditas se aproximem das saídas esperadas (*ground truth*) por meio dos algoritmos de Gradiente Descendente e *Backpropagation* [Aggarwal, 2023].

A diferença entre as saídas esperadas e as saídas preditas por uma RNA é chamada de valor de perda. O valor de perda é calculado a partir da aplicação de uma função de perda (ou custo), como, por exemplo, a entropia cruzada. A entropia cruzada é normalmente empregada em problemas de classificação e mede a diferença entre as distribuições de probabilidade das saídas esperadas e das saídas preditas pelo modelo [Aggarwal, 2023].

Terminado o processo de treinamento, o conjunto de dados de teste é utilizado para calcular as métricas de avaliação adotadas, tais como a acurácia e, assim, aferir o desempenho da RNA. Por meio desse processo, espera-se garantir que a rede consiga atingir capacidade de generalização suficiente para dados diferentes daqueles utilizados no treinamento.

A escolha dos hiperparâmetros influencia diretamente em como vai ser o treinamento de uma RNA e o seu desempenho para dados de teste, por isso tem tamanha importância. Os hiperparâmetros podem ser referentes à arquitetura da RNA (por exemplo, número de neurônios ou número de camadas) ou ao seu treinamento (por exemplo, quantas épocas serão realizadas ou o tamanho de lote adotado). Além disso, dependendo da arquitetura da rede, os tipos de hiperparâmetros podem ser diferentes.

2.2. Redes Neurais Convolucionais

Uma rede neural convolucional (*convolutional neural network* - CNN) é um tipo de RNA composta de uma ou mais camadas convolucionais, que são especialmente eficientes para tratar dados multidimensionais e, portanto, costumam ser empregadas em tarefas de visão computacional. Camadas convolucionais são constituídas de filtros convolucionais que visam extrair características (*features*) de uma imagem. Assim, dada uma imagem de entrada, cada filtro convolucional, que possui um determinado tamanho, é sucessivamente deslocado pela imagem, tal como definido por um hiperparâmetro chamado *stride*, e aplicado a cada submatriz da imagem, por meio do cálculo de um produto interno. Os pesos de cada filtro têm seus valores ajustados ao longo do treinamento a fim de possibilitar a extração de características relevantes das imagens e auxiliar na tarefa abordada. A saída de uma camada convolucional é denominada de mapa de características (*feature map*).

Além das camadas convolucionais, as CNNs têm camadas de agrupamento (*pooling*), que são usadas para filtrar as informações mais importantes da camada anterior, reduzindo o número de

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

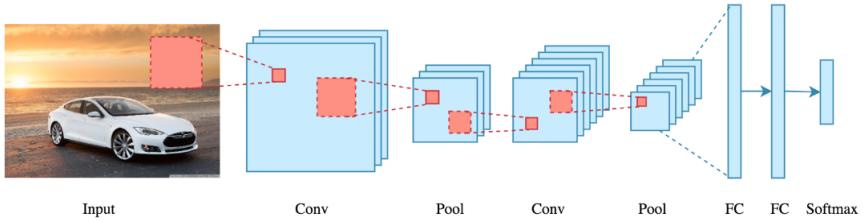


Figura 1: Exemplo de uma CNN completa (extraída de [Int]).

parâmetros a serem processados e a dimensionalidade dos dados, e camadas densas (*fully-connected* - FC), em que cada neurônio está ligado a todos os neurônios da camada anterior. Além disso, uma CNN conta também com uma camada de *flattening* que lineariza os dados ao final da última camada de *pooling* para que possam ser consumidos pelas camadas densas. Por fim, existe ainda a camada *softmax* que tipicamente é utilizada como camada de saída e tem como função transformar a estimativa da rede em uma distribuição de probabilidade considerando as classes existentes na tarefa. A Figura 1 exemplifica uma CNN com duas camadas convolucionais, duas camadas de *pooling*, duas camadas densas e uma camada *softmax*.

2.3. Validação Cruzada

A validação cruzada (*cross-validation* (cv), em inglês) [Burman, 1989] é uma técnica de validação amplamente utilizada em problemas de regressão e classificação para estimar a medida do erro. A principal característica dessa técnica é a separação de uma parte diferente do *dataset* a cada rodada para validação. Uma variante bastante utilizada da validação cruzada é o *k-fold*, em que o conjunto de dados é separado em k partes (*folds*) de tamanho igual ou o mais próximo possível. O algoritmo realiza k iterações e, em cada iteração, um dos *folds* é separado como conjunto de validação, enquanto os outros $k-1$ *folds* são utilizados para o treinamento do modelo. Assim, a técnica permite uma avaliação mais robusta e confiável do desempenho do modelo [Burman, 1989], garantindo que todas as amostras sejam usadas tanto para treinamento quanto para validação.

Foi demonstrado por [Bates et al., 2023] que técnicas de validação cruzada como a *k-fold* são boas em estimar o erro produzido por um modelo quando k tende a infinito, porém são muito otimistas para k menores, ou seja, nesses casos indicam um valor de erro menor do que o real. Os autores propuseram, então, um novo estimador, chamado validação cruzada aninhada (*nested cross validation*, em inglês). Segundo os autores, esse estimador tem a capacidade de gerar intervalos de confiança mais precisos e menos enviesados sobre os erros estimados na previsão.

Um exemplo de validação cruzada aninhada baseada na técnica *k-fold*, consiste em dividir o *dataset* em k *folds* de tamanho igual ou o mais próximo possível. Seguem-se então dois laços aninhados: um laço externo de 1 até k separa um *fold* para a validação final; outro laço interno percorre os demais $k-1$ *folds*, separando a cada iteração um *fold* para testes e os demais $k-2$ *folds* para treinamento do modelo. Dessa forma, há sempre um conjunto de dados para validação do modelo que não foi utilizado no treinamento. Ao final das repetições, a validação cruzada aninhada estima o erro médio sobre conjuntos distintos de treinamento, teste e validação. É importante destacar que essa técnica requer mais recursos computacionais do que a validação cruzada simples.

2.4. irace

O irace [López-Ibáñez et al., 2016] consiste em um pacote em R que implementa o algoritmo *Iterated F-Race* (I/F-Race) [Birattari et al., 2010], comumente adotado na área de Metaheurísticas. *Iterated race*, ou corrida iterativa, é um método que considera várias iterações (ou corridas) para realizar a definição automática dos valores de hiperparâmetros, selecionando as melhores configurações de hiperparâmetros por meio de testes estatísticos sequenciais e levando para

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

as iterações subsequentes as melhores configurações. A corrida iterativa consiste em três passos: (i) selecionar amostras de novas configurações de acordo com alguma distribuição em particular; (ii) selecionar as melhores configurações da nova amostragem por meio de corridas; e (iii) atualizar a distribuição de probabilidade dos hiperparâmetros de forma a criar uma tendência para que boas configurações sejam sorteadas na próxima amostragem. Esses passos se repetem até que seja atingido um critério de parada definido.

Na implementação do pacote irace, cada hiperparâmetro possui uma distribuição de amostragem independente, exceto quando existem condições ou restrições específicas que conectam certos hiperparâmetros. A distribuição inicial da amostragem corresponde a uma distribuição normal para hiperparâmetros numéricos e distribuição contínua para hiperparâmetros categóricos. Já hiperparâmetros ordinais são tratados como numéricos. A cada iteração, a distribuição da amostragem é atualizada mudando-se a média e o desvio padrão nas distribuições normais e o valor de probabilidade nas distribuições discretas. Isso é feito para ajustar a probabilidade de um valor ser escolhido de acordo com os melhores resultados obtidos nas iterações anteriores.

De maneira geral, a corrida iterativa seleciona as melhores configurações de hiperparâmetros testando várias amostras (candidatos) a cada iteração, em que cada amostra corresponde a uma dada configuração. De tempos em tempos, as piores amostras são descartadas após testes estatísticos, até que sobre apenas um determinado número de melhores amostras, chamadas de configurações elite, ou se atinja um limite computacional pré-estabelecido (*budget*). O limite computacional pode ser uma quantidade de tempo definida ou um determinado número de testes, em que um teste consiste em executar a configuração associada a uma amostra em uma dada instância com uma dada semente (*seed*) de aleatoriedade.

No contexto deste trabalho, um teste corresponde a um treinamento completo de uma RNA com a configuração de hiperparâmetros associada a uma amostra para uma dada semente, pois não há o conceito de instância. Note-se que a variação de sementes é fundamental, visto que o treinamento de uma RNA possui diversos componentes estocásticos.

A Figura 2 mostra uma iteração do irace. Inicialmente são escolhidas oito amostras (candidatos), em que cada amostra corresponde a uma configuração de hiperparâmetros. Em seguida, é iniciada a corrida que consiste em uma série de testes que podem eliminar ou não algumas das amostras. Nessa figura, os três primeiros testes não eliminam nenhuma amostra, porém, a partir do quarto teste, designado por I_4, são eliminadas amostras até que sobrem apenas duas. As duas configurações associadas a essas amostras restantes são classificadas como configurações elite e a distribuição de probabilidade do espaço amostral é atualizada, para que, na próxima iteração, as configurações mais próximas dessas duas melhores configurações tenham maiores chances de serem selecionadas.

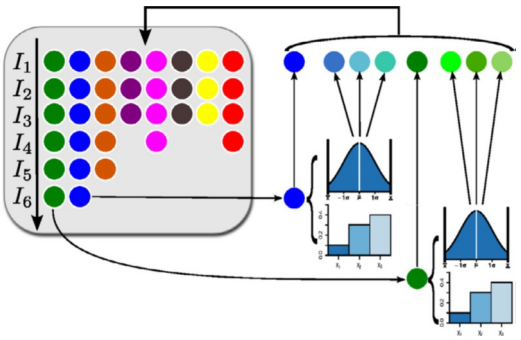


Figura 2: Ilustração de uma iteração do irace (extraída de [López-Ibáñez et al., 2016]).

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

2.5. Optuna

O Optuna [Akiba et al., 2019] é um *framework* em Python que utiliza otimização Bayesiana para estimar os valores de hiperparâmetros de modelos de AM de forma automática. A sua vantagem sobre o irace é que permite realizar a seleção das configurações de hiperparâmetros no próprio código de definição do modelo de AM, além de possuir uma interface gráfica para acompanhamento dos resultados. Outra diferença é que, enquanto no irace os hiperparâmetros a serem otimizados precisam ser predeterminados em um arquivo de configuração, o Optuna permite a criação de hiperparâmetros de forma dinâmica. Os valores de hiperparâmetros são definidos realizando chamadas às funções de sorteio do Optuna no momento da utilização desses hiperparâmetros.

3. Trabalhos Relacionados

Foram encontrados diversos trabalhos que empregam meta-heurísticas na otimização de valores de hiperparâmetros de RNAs, especialmente algoritmos genéticos. Entretanto, optou-se por apresentar nesta seção somente aqueles que utilizam as ferramentas irace ou Optuna e são, portanto, mais relacionados a este trabalho.

Apesar de o irace ser uma ferramenta bastante consolidada na área de meta-heurísticas, existem poucos trabalhos que o empregam na otimização dos hiperparâmetros de modelos de AP. O trabalho de [Vieira, 2021] criou uma ferramenta de *AutoML* chamada iSklearn, baseada no irace. O iSklearn foi utilizado para configurar diferentes modelos de AM e foi comparado a outra ferramenta já existente de *AutoML*, chamada AutoSklearn, em relação a três *datasets* de visão computacional e dois de séries temporais. Os resultados demonstraram que o iSklearn, consistentemente, demonstrou melhores resultados do que os algoritmos de ML com valores padrão e obteve resultados comparáveis com a ferramenta AutoSklearn, sendo melhor em alguns casos. Ademais, os autores demonstraram a utilização do irace como *Neural Architecture Search* (NAS), que é um campo de *AutoML* para o projeto de RNAs complexas.

Costa et al. [2022] utilizaram uma rede LSTM para a predição de valores de ações. Para o ajuste dos hiperparâmetros, os autores empregaram o irace a fim de diminuir o espaço amostral e, em seguida, o *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) [Deb et al., 2002] para fazer a sintonia fina dos hiperparâmetros, usando os valores conforme encontrados pelo irace com uma variação de 10% para mais ou para menos. Como resultados obtidos, os autores constataram que o ajuste dos hiperparâmetros utilizando somente o irace não foi tão promissor, mas reduziu o espaço para a execução do NSGA-II, que levou então a resultados mais efetivos.

Diferentemente do irace, o Optuna tem sido utilizado em muitos trabalhos de AP desde sua criação em 2019. No trabalho de [Aguilera-Martos et al., 2023], os autores utilizaram o Optuna para otimizar diferentes modelos para a detecção de anomalias e os analisa por meio da comparação dos resultados obtidos e do tempo necessário para a estimação dos melhores valores de hiperparâmetros. Em [Hassanpour et al., 2024], os autores utilizaram um *Autoencoder*, que é um tipo de RNA adotada para redução de dimensionalidade, além de algoritmos de clusterização, para detectar falhas de funcionamento em sistemas de aquecimento, ventilação e ar-condicionado. O Optuna foi utilizado a fim de selecionar e otimizar os valores dos hiperparâmetros de todos os algoritmos testados.

4. Experimentos Computacionais

Com o objetivo de verificar a viabilidade de uso da ferramenta irace na otimização de hiperparâmetros de redes neurais, foram realizados dois experimentos computacionais com uma CNN e os resultados obtidos foram comparados com os gerados pelo Optuna. Nas próximas subseções, serão descritos detalhadamente os experimentos e os resultados.

4.1. Ambiente Computacional

O ambiente computacional adotado nos experimentos foi o seguinte: processador i5-14400F, 32GB DDR5 de memória RAM, placa de vídeo NVIDIA GeForce 4060 com 16GB de

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

GDRAM e sistema operacional Windows 11. A implementação da rede neural foi feita na linguagem de programação Python na versão 3.10.14, usando-se as bibliotecas Keras e Tensorflow[1] na versão 2.10. A ferramenta irace[2] foi utilizada na versão 3.3 e o Optuna[3], na versão 3.6. Os experimentos foram executados utilizando-se a placa de vídeo com duas *threads* em paralelo. Os códigos desenvolvidos nesta pesquisa e os resultados dos experimentos estão disponíveis publicamente em um repositório criado no GitHub[4].

4.2. *Dataset*

O *dataset* CIFAR-10[5] corresponde a um subconjunto de um *dataset* maior apresentado em [Krizhevsky e Hinton, 2009], composto por imagens reais com resolução de 32 x 32 *pixels*, contendo 3 canais de cores. O CIFAR-10 é composto por 60.000 imagens divididas em 10 classes, com 6.000 imagens em cada classe, divididas entre 5.000 imagens para treino e 1.000 para teste.

4.3. Arquitetura da Rede

Como o *dataset* CIFAR-10 é composto por imagens, foi utilizada uma CNN nos experimentos, visto que corresponde à arquitetura de rede neural mais apropriada para lidar com dados dessa natureza.

Os hiperparâmetros contidos na Tabela 1 foram escolhidos para serem variados nos experimentos computacionais, pois costumam ser aqueles com maior variação nesse tipo de rede. Nessa tabela, cada linha possui as seguintes colunas: o nome do hiperparâmetro, o tipo de valor assumido, o intervalo de valores variado e alguma condição relacionada. Por exemplo, para o hiperparâmetro "camada de *max pooling* (MP)", o valor assumido é booleano, de modo que os valores variados nos experimentos corresponderam a 0 ou 1, indicando a presença ou não da camada de *max pooling* após uma dada camada convolucional.

Tabela 1: Relação dos onze hiperparâmetros adotados nos experimentos computacionais.

Hiperparâmetro	Tipo	Valores	Condição
número de camadas convolucionais (NC)	inteiro	1 a 8	N/A
camada de *max pooling* (MP)	booleano	0 ou 1	sorteado para cada camada convolucional
batch normalization (BN)	booleano	0 ou 1	sorteado para cada camada convolucional
número de filtros nas camadas convolucionais (NF)	inteiro	32 a 256	potências de 2
tamanho do filtro nas camadas convolucionais (TF)	inteiro	3 a 11	número ímpar
taxa de dropout (TD)	real	0.0 a 0.5	após cada camada densa
número de camadas densas (CD)	inteiro	1 a 3	N/A
quantidade de neurônios nas camadas densas (ND)	inteiro	4 a 128	N/A
taxa de aprendizado (TA)	real	$0,00001$ a 1	ordem de grandeza
tamanho do lote (TL)	inteiro	16 a 1024	potências de dois
número de épocas (NE)	inteiro	10 a 300	N/A

A CNN foi criada de acordo com os valores dos hiperparâmetros da Tabela 1. A primeira camada oculta é uma camada convolucional que recebe a imagem de entrada. Se o hiperparâmetro

[1]https://www.tensorflow.org/?hl=pt-br
[2]https://mlopez-ibanez.github.io/irace/news/index.html#irace-35
[3]https://optuna.readthedocs.io/en/v3.6.0/
[4]https://github.com/camillamelo/deeplearningwithirace
[5]https://www.cs.toronto.edu/∼kriz/cifar.html

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

MP for verdadeiro para essa primeira camada, é adicionada em seguida uma camada de *max pooling* de tamanho 2×2 e de *stride* igual a 2. Caso o hiperparâmetro BN seja verdadeiro, é adicionada uma camada de *batch normalization*. A criação de mais camadas convolucionais em sequência está condicionada ao valor de NC, caso NC seja maior do que 1. Em cada uma destas novas camadas, são adicionadas camadas de *max pooling* e *batch normalization* de acordo com os valores de MP e BN. As camadas convolucionais têm como hiperparâmetros o número e o tamanho dos filtros de acordo com os valores de NF e TF, possuindo um *stride* fixo de valor igual a 1. Após tais camadas, é adicionada uma camada de *flattening*. Em seguida, é criada uma quantidade de camadas densas em função do valor de CD, com o número de neurônios ND em cada uma, e taxa de *dropout* igual a TD. Por fim, a última camada densa possui 10 neurônios com função de ativação *softmax*. Ademais, a função de perda adotada correspondeu à entropia cruzada.

Nos experimentos, a métrica de avaliação utilizada no processo de otimização feito pelo irace e pelo Optuna correspondeu ao valor de perda para o conjunto de validação (*validation loss*). As melhores configurações obtidas foram submetidas a uma validação cruzada aninhada, em que o *dataset* inteiro (incluindo conjuntamente os dados de treino e teste, totalizando os 60.000 exemplos) foi divido em seis *folds*, de modo que, em cada rodada, um *fold* foi separado para validação e, para os demais cinco *folds*, foram feitas cinco rodadas aninhadas à anterior com um *fold* escolhido para teste e os quatro restantes para treinamento.

4.4. Resultados

No primeiro experimento, inicialmente foi executado o Optuna com um limite de 100 testes, em que um teste representa o processo de treinamento completo da rede neural. O tempo de processamento foi de aproximadamente 17 horas e a melhor configuração obteve um valor de perda de 0,6915. Em seguida, foi executado o irace adotando-se como limite de tempo a mesma quantidade de tempo tomada pelo Optuna. Dessa maneira, o irace foi capaz de testar 59 configurações com cinco sementes e a melhor configuração (vencedora) obteve um valor de perda médio de 0,7530, que é 8,9% maior do que a melhor configuração do Optuna. Deve-se observar que o irace somente aponta como vencedora uma configuração que tenha sido testada em todas as sementes, mas também aponta as configurações elite, que são as melhores após cada iteração. Assim, ao final do primeiro experimento, foram apontadas cinco configurações como elite das quais algumas obtiveram menor valor de perda médio do que a melhor configuração gerada pelo Optuna.

A fim de realizar uma comparação mais aprofundada, foram utilizadas as cinco configurações elite do irace e as cinco melhores configurações do Optuna e foi refeito o treinamento de cada uma usando-se a técnica de validação cruzada aninhada com seis *folds* no laço externo e cinco *folds* no laço interno, extraindo-se as médias do valor de perda e da acurácia.

A Figura 3 exibe um *boxplot* comparando o valor de perda médio das cinco configurações elite do irace (IR1 a IR5, em vermelho) e as cinco melhores do Optuna (OP1 a OP5, em azul). As configurações do irace obtiveram, na média, um valor de perda 2,3530% maior do que o Optuna. É possível notar que, apesar de a configuração IR1 ter sido escolhida pelo irace como vencedora, a IR5 foi aquela que obteve o melhor resultado. Isso aconteceu devido ao fato de que, durante a execução do irace, a configuração IR5 só foi testada em quatro sementes, provavelmente por causa do limite de tempo. Além disso, também é possível observar que, para as configurações do Optuna, a configuração OP1 escolhida como vencedora tem uma grande variabilidade entre os *folds*, atingindo uma variação de valor perda entre 0,7095 e 0,8312. Isso provavelmente se deve ao fato de que, diferentemente do irace, o Optuna seleciona as configurações com base em apenas um teste, sem variar sementes, de forma que não leva em consideração os componentes estocásticos contidos no treinamento de uma RNA.

Já a Figura 4 mostra a variação da acurácia para as mesmas cinco melhores configurações

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

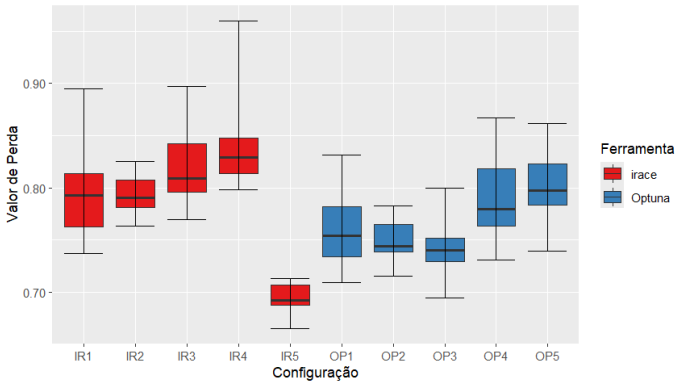


Figura 3: *Boxplot* do valor de perda médio obtido após validação cruzada aninhada com seis *folds* das cinco melhores configurações apontadas pelo irace (em vermelho) e pelo Optuna (em azul).

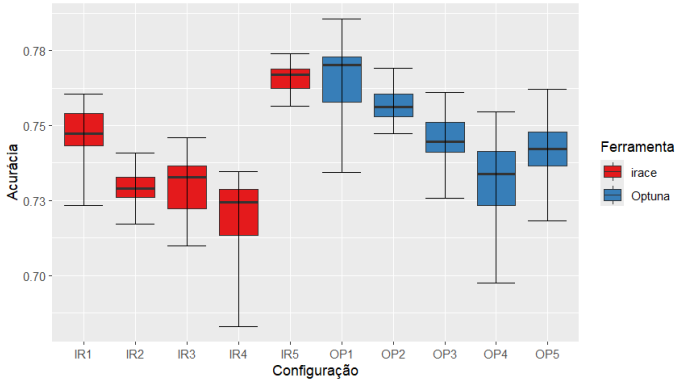


Figura 4: *Boxplot* da acurácia média obtida após validação cruzada aninhada com seis *folds* das cinco melhores configurações apontadas pelo irace (em vermelho) e pelo Optuna (em azul).

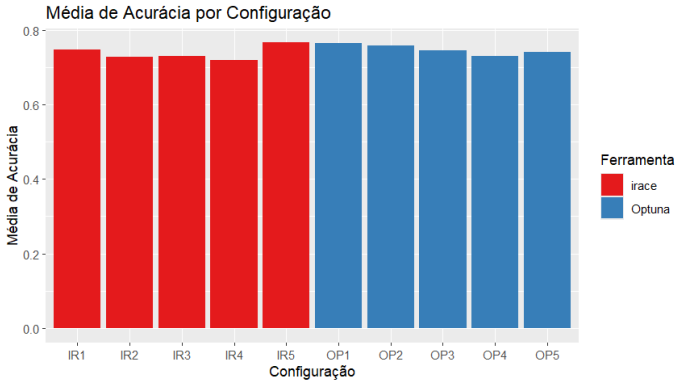


Figura 5: EXP1 - valores médios de acurácia obtidos após validação cruzada aninhada das cinco melhores configurações apontadas pelo irace (em vermelho) e pelo Optuna (em azul).

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

do irace (IR1 a IR5, em vermelho) e do Optuna (OP1 a OP5, em azul). As configurações do Optuna alcançaram, na média, um valor de acurácia $1{,}2741\%$ maior do que as do irace, porém a melhor configuração do irace (IR5) obteve, na média, melhor acurácia do que todas as configurações do Optuna, com um valor de $0{,}7661$, o que pode ser percebido mais detalhadamente na Figura 5. Note-se também que, na Figura 4, a variação da acurácia na melhor configuração obtida pelo irace (IR5) é menor do que nas demais configurações geradas, variando entre $0{,}7564$ e $0{,}7739$, com desvio padrão de $0{,}0043$, como o próprio *boxplot* evidencia.

No segundo experimento, as ferramentas foram executadas na ordem inversa: primeiramente foi executado o irace com o limite de 777 testes, valor apontado pela ferramenta como o mínimo necessário para viabilidade do experimento. Além disso, foi modificado o parâmetro *firstTest* para 4, de tal forma que o irace passasse a testar uma configuração em, ao menos, quatro sementes antes de descartá-la. O tempo de processamento foi de aproximadamente 36 horas, em que o irace foi capaz de testar 138 configurações em 15 sementes. Foram apontadas sete configurações como elite e a melhor foi aquela que obteve um valor de perda médio de $0{,}7028$. Para esse mesmo limite de tempo, o Optuna testou 190 configurações, das quais a melhor teve um valor de perda de $0{,}6353$, que é $9{,}61\%$ menor do que o da melhor configuração do irace.

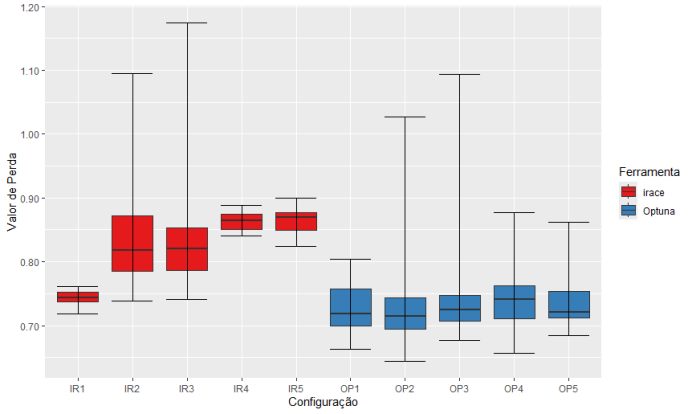


Figura 6: *Boxplot* do valor de perda médio obtido após validação cruzada com cinco *folds* das cinco melhores configurações apontas pelo irace (em vermelho) e pelo Optuna (em azul).

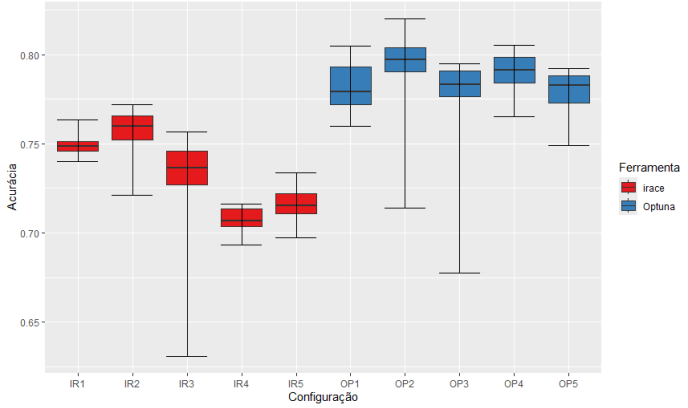


Figura 7: *Boxplot* da acurácia média obtida após validação cruzada com cinco *folds* das cinco melhores configurações apontas pelo irace (em vermelho) e pelo Optuna (em azul).

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

Analogamente ao experimento anterior, foram utilizadas as cinco melhores configurações geradas por cada ferramenta (no caso do irace, as cinco melhores dentre as configurações elite) para se realizar o treinamento de cada uma adotando-se validação cruzada aninhada e se extraírem as médias do valor de perda e da acurácia. As Figuras 6 e 7 exibem um *boxplot* comparando as médias do valor de perda e da acurácia, respectivamente, para as cinco melhores configurações do irace (IR1 a IR5, em vermelho) e Optuna (OP1 a OP5, em azul). É possível observar que as configurações geradas pelo Optuna alcançaram, na média, valores melhores tanto de valor de perda quanto de acurácia. Nesse caso, houve uma variação grande entre as configurações obtidas tanto pelo irace quanto pelo Optuna. Entretanto, destaque-se que, entre as configurações do irace, as configurações IR2 e IR3 obtiveram maior variação entre os *folds* da validação cruzada aninhada, ao passo que as configurações IR1, IR4 e IR5 se mostraram bastante estáveis. Já as configuraçãoes do Optuna se mostraram muito parecidas, com um desvio padrão do valor de perda de $0{,}0350$ para a configuração OP1, que é maior do que o desvio padrão do valor de perda de $0{,}0107$ para a configuração IR1, que foram aquelas apontadas como as melhores pelo Optuna e pelo irace, respectivamente.

5. Conclusão

Este trabalho apresentou experimentos computacionais a fim de verificar a viabilidade do uso da ferramenta irace para a otimização de hiperparâmetros de RNAs. Foi feita uma comparação com a ferramenta Optuna tomando-se como base uma CNN treinada sobre o *dataset* CIFAR-10. Os resultados dos experimentos apontaram que o irace pode obter resultados comparáveis com aqueles do Optuna, embora este último tenha apresentado uma pequena vantagem. O Optuna é capaz de testar mais configurações em um mesmo período de tempo pelo fato de não levar em consideração o caráter estocástico do treinamento de uma RNA. Assim, apesar de ter chegado a configurações com melhor resultado, essas apresentaram alta variabilidade quando treinadas em uma validação cruzada aninhada. Por outro lado, o irace obtém resultados mais estáveis por testar a mesma configuração com diferentes sementes, o que também pode funcionar como uma limitação por haver o descarte de configurações promissoras caso não seja possível testás-las em todas as sementes.

Conforme evidenciado nos experimentos, o irace pode acabar descartando boas configurações ou não as considerando como as melhores pelo fato de essas configurações não terem sido testadas em todas as sementes. Isso aconteceu em alguns experimentos mesmo ainda havendo *budget* no final da execução para que essas sementes fossem testadas. Essa questão foi levada até os criadores da ferramenta irace que reconheceram esse fato como uma possível falha e abriram um *issue* no repositório do *Github*[6] que levou ao desenvolvimento de uma nova versão da ferramenta (versão 4.1)[7]. No tempo de finalização deste artigo, não foi possível testar a nova versão para se verificar se realmente o problema foi corrigido, embora este esteja reportado entre os problemas solucionados na nova versão. Além disso, uma outra contribuição deste trabalho corresponde à observação sobre as melhores combinações de hiperparâmetros encontradas para CNNs em relação ao *dataset* CIFAR-10, encontrando-se disponível em uma planilha no GitHub[8] associado a esta pesquisa.

Trabalhos futuros envolvem a experimentação considerando outras bases de dados e arquiteturas de redes a fim de averiguar a generalização dos resultados, a investigação do impacto oriundo da alteração de outros parâmetros do irace, a adoção de outras ferramentas de otimização de hiperparâmetros para comparação, além do emprego de métodos voltados à diminuição do espaço de busca explorado pelas ferramentas e a avaliação do uso de pós-processamento para concluir testes que apresentam bons resultados, mas que não tenham tido todas as sementes verificadas pelo irace.

[6] https://github.com/MLopez-Ibanez/irace/issues/69#issuecomment-2558143606
[7] https://cran.r-project.org/web/packages/irace/index.html
[8] https://github.com/camillamelo/deeplearningwithirace/blob/main/comparativos/sumario.xlsx

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br

Referências

Introduction to convolutional neural networks. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/. Accessado em out. 2024.

Aggarwal, C. C. (2023). *Neural Networks and Deep Learning: A Textbook*. Springer Publishing Company, Incorporated, 2nd edition. ISBN 3031296419.

Aguilera-Martos, I., García-Barzana, M., García-Gil, D., Carrasco, J., López, D., Luengo, J., e Herrera, F. (2023). Multi-step histogram based outlier scores for unsupervised anomaly detection: Arcelormittal engineering dataset case of study. *Neurocomputing*, 544:126228. ISSN 0925-2312.

Akiba, T., Sano, S., Yanase, T., Ohta, T., e Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD' 19. ACM.

Bates, S., Hastie, T., e Tibshirani, R. (2023). Cross-validation: What does it estimate and how well does it do it? *Journal of the American Statistical Association*, 119(546):1434–1445. ISSN 1537-274X.

Bezerra, L. C. T., López-Ibáñez, M., e Stützle, T. (2020). *Automatic Configuration of Multi-objective Optimizers and Multi-objective Configuration*, p. 69–92. Springer International Publishing, Cham. ISBN 978-3-030-18764-4.

Birattari, M., Yuan, Z., Balaprakash, P., e Stützle, T. (2010). F-race and iterated f-race: An overview. In *Experimental Methods for the Analysis of Optimization Algorithms*, p. 311–336. Springer Berlin Heidelberg.

Burman, P. (1989). A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, 76(3):503–514. ISSN 0006-3444.

Costa, T. F., Wanner, E. F., Martins, F. V. C., e da Cruz, A. R. (2022). A methodology for definition and refinement of a LSTM stock predictor architecture using iRace and NSGA-II. In *Anais do I Brazilian Workshop on Artificial Intelligence in Finance (BWAIF 2022)*. Sociedade Brasileira de Computação.

Deb, K., Pratap, A., Agarwal, S., e Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Hassanpour, H., Hamedi, A. H., Mhaskar, P., House, J. M., e Salsbury, T. I. (2024). A hybrid clustering approach integrating first-principles knowledge with data for fault detection in hvac systems. *Computers & Chemical Engineering*, 187:108717. ISSN 0098-1354.

Krizhevsky, A. e Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., e Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., USA, 1 edition. ISBN 0070428077.

Vieira, C. E. M. (2021). Assessing irace for automated machine and deep learning in computer vision. Master's thesis, Universidade Federal do Rio Grande do Norte.

DOI: 10.59254/sbpo-2025-212654
https://proceedings.science/p/212654?lang=pt-br