Multi-Objective Optimization of Sequence of Layers in Deep Learning Architectures

Mayara S. O. Castro[1], **Giuseppe F. Neto**[1], **Péricles B. C. de Miranda**[1], **Filipe R. Cordeiro**[1], **Ricardo B. C. Prudêncio**[2]

[1]Departamento de Computação – Universidade Federal Rural de Pernambuco (UFRPE)
Recife – PE – Brazil

[2]Centro de Informática – Universidade Federal de Pernambuco
Recife – PE – Brazil

{pericles.miranda}@ufrpe.br

Abstract. *Selecting the best architecture for a Deep Neural Network (DNN) is a non-trivial task since there is a massive amount of possible configurations (layers and parameters) and great difficulty in how to choose them. In order to make this task more independent of human interaction, this paper proposes an intelligent method to optimize the architecture (sequence of layers) of a chain-structured DNN, taking into account multiple criteria: accuracy and* $F_1 score$*. The method was evaluated for performance and compared to the exhaustive and random approaches. The results obtained are promising, showing the potential of the proposed method.*

Resumo. *Selecionar a melhor arquitetura para uma Rede Neural Profunda (DNN) não é uma tarefa trivial, pois há uma enorme quantidade de configurações possíveis (camadas e parâmetros) e grande dificuldade em como escolhê-las. A fim de tornar essa tarefa mais independente da interação humana, este trabalho propõe um método inteligente para otimizar a arquitetura (sequência de camadas) de uma DNN estruturada em cadeia, levando em consideração múltiplos critérios: acurácia e* $F_1 score$*. O método foi avaliado quanto ao desempenho e comparado às abordagens exaustiva e aleatória. Os resultados obtidos são promissores, mostrando o potencial do método proposto.*

1. Introdução

A tarefa de resolver problemas como a identificação de objetos em imagens e a transcrição de voz em texto usa cada vez mais uma classe de técnicas chamadas *Deep Neural Networks* (DNN) [LeCun et al. 2015]. Estas técnicas foram muito bem sucedidas nos últimos anos, alcançando os seres humanos em diferentes tarefas. No entanto, esses algoritmos dependem de sua arquitetura (número de camadas, o tipo de cada camada, posicionamento das camadas e parâmetros) para obter um desempenho satisfatório, o que levanta a questão: Qual é a arquitetura da DNN mais adequada para um determinado problema de classificação?

Os três métodos mais amplamente utilizados para a seleção da arquitetura de DNNs são (1) busca manual, (2) busca exaustiva e (3) busca aleatória [Young et al. 2015]. Busca manual refere-se ao processo de um especialista selecionar manualmente as arquiteturas para avaliar. Isso requer profundo conhecimento sobre o problema e o algoritmo,

sendo difícil para um não especialista definir uma boa configuração. A busca exaustiva é um procedimento computacional que avalia todas as combinações possíveis de arquiteturas para a DNN, não sendo eficiente em um espaço de alta dimensão. Por fim, a busca aleatória tenta encontrar boas soluções avaliando soluções aleatórias do conjunto de candidatos. Nesse caso, quando o problema tem um espaço de alta dimensão, a chance de encontrar uma arquitetura adequada para o DNN por sorte (aleatoriamente) é pequena. Por esta razão, há um interesse crescente na seleção automática de arquitetura de DNNs usando algoritmos inteligentes [Elsken et al. 2018].

Este trabalho trata o problema de seleção de arquitetura da DNN como uma tarefa de otimização com múltiplos objetivos em um contexto combinatório. O método proposto tenta encontrar a melhor composição e sequência de camadas para a arquitetura da DNN. Nesta pesquisa, dois objetivos são considerados na otimização: maximização da acurácia e maximização de F_1score. Uma avaliação quantitativa foi realizada comparando o método proposto com a busca exaustiva e a busca aleatória, em uma gama de problemas de classificação. Os resultados mostraram que o método proposto é eficiente e eficaz para selecionar arquiteturas adequadas para a DNN, considerando os objetivos, acurácia e F_1score. É importante mencionar que nos concentramos na otimização de DNNs estruturadas em cadeia.

Este artigo está organizado da seguinte forma: a seção 2 introduz a DNN e a otimização multi-objetivo. A seção 3 apresenta trabalhos relacionados. A seção 4 descreve o método proposto. A seção 5 apresenta a metodologia experimental utilizada para avaliar o método proposto. A seção 6 apresenta os resultados obtidos. Finalmente, a seção 7 destaca a conclusão e trabalhos futuros.

2. *Background*

Esta seção apresenta conceitos fundamentais sobre DNN e otimização multi-objetivo para uma melhor compreensão do trabalho proposto.

2.1. *Deep Neural Networks*

Diferente dos algoritmos convencionais de aprendizado de máquina, as DNNs pertencem à classe de métodos denominados métodos de aprendizagem representativa. Esses métodos permitem que uma máquina seja alimentada com dados brutos e descubra as representações necessárias para detecção ou classificação automaticamente [LeCun et al. 2015]. As DNNs são métodos com múltiplos níveis de representação, obtidos pela composição de módulos simples, mas não lineares, que modificam a representação em um nível (começando com a entrada bruta) um pouco mais abstrato. Com a composição de tais transformações, funções muito complexas podem ser aprendidas.

Para tarefas de classificação, camadas com nível maior de representação amplificam aspectos da entrada que são importantes para a discriminação e suprimem variações irrelevantes [LeCun et al. 2015]. A Figura 1 mostra um exemplo de uma DNN estruturada em cadeia, que é uma sequência de camadas, onde a $i^{ésima}$ recebe sua entrada da camada $i-1$ e sua saída serve como entrada para a camada $i+1$. Cada camada treina um conjunto distinto de características com base na camada anterior. Quanto mais avançada a rede, mais complexas as características que as camadas podem reconhecer, uma vez que agregam e recombinam as características das camadas anteriores. Este tipo de arquitetura de DNN é comumente usado na literatura.

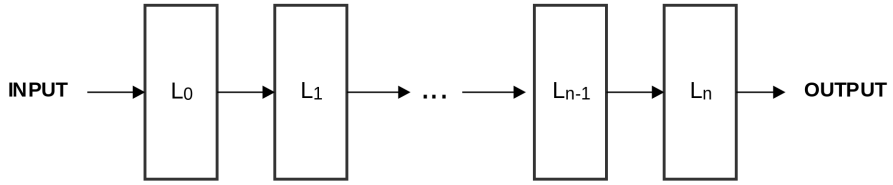

Figura 1. Exemplo de DNN estruturado em cadeia com diferentes camadas. Fonte: [Elsken et al. 2018]

A arquitetura de uma DNN pode ser composta de diferentes tipos e números de camadas. Os tipos de camadas e onde cada uma está posicionada podem ter uma grande influência no desempenho da rede. Aqui, discutimos três camadas populares: *Convolutional*, *Pooling* e *Dense*. A camada *Convolutional* possui um conjunto de filtros que podem ser aprendidos. Convolução é o simples emprego de um filtro em uma entrada que resulta em uma 'ativação'. A aplicação repetida do mesmo filtro a uma entrada retorna um mapa de ativações chamado de mapa de recursos, indicando os locais e a intensidade de um recurso detectado em uma entrada, como uma imagem. As camadas de *Pooling* são responsáveis por calcular o local médio ou máximo e fazer subamostras dos dados da camada anterior, reduzindo o tamanho e selecionando os recursos mais relevantes da entrada [LeCun et al. 2015]. O tipo mais comum de *Pooling* usado é o *Max Pooling*, que é um filtro de tamanho quadrado e *stride* escolhido de acordo com o problema, o que resulta em cada sub-seleção da imagem no maior valor dentro da região selecionada. Uma camada do tipo *Dense* é apenas uma camada regular de neurônios em uma rede neural. Cada neurônio recebe como entrada todas as saídas dos neurônios na camada anterior, portanto, densamente conectados. A camada tem uma matriz de peso, um vetor de polarização e as ativações da camada anterior.

2.2. Otimização Multi-Objetivo

Os problemas de Otimização Multi-Objetivo (MOO) referem-se a problemas que contêm mais de um objetivo conflitante. Algoritmos MOO tentam encontrar soluções, em um espaço de busca de soluções candidatas $\mathcal{S}$, que satisfazem todos os objetivos conflitantes ($\vec{f} = (f_1, f_2, ..., f_j)$) ao mesmo tempo, onde j é o número de objetivos.

Cada solução em $\mathcal{S}$ pode ser representada por um vetor de variáveis de decisão ($\vec{x} = (x_1, x_2, ..., x_k)$), onde k é o número de variáveis de decisão. Cada solução tem sua qualidade (aptidão) avaliada através da execução de cada função objetivo sobre ela. Assim, $\vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), ..., f_j(\vec{x})$ representa um vetor de valores de aptidão que pertence à solução $\vec{x}$. Um problema geral MOO, em que seja deseja minimizar todos os objetivos, pode ser definido como:

$$\text{minimizar } \vec{f}(\vec{x}) := [f_1(\vec{x}), f_2(\vec{x}), ..., f_j(\vec{x})], \tag{1}$$

sujeito a:

$$g_i(\vec{x}) \leq 0 \quad i = 1, 2, ..., q, \tag{2}$$

$$h_j(\vec{x}) = 0 \quad j = 1, 2, ..., s, \tag{3}$$

onde $\vec{x} = (x_1, x_2, ..., x_k)$ é o vetor no espaço de busca de decisão; e $g_i(\vec{x})$, $h_j(\vec{x})$ são as funções de restrição e $q + s$ é o número de restrições do problema.

Como em MOO, a qualidade de cada solução é representada por um vetor de valores de aptidão (ou *fitness*), a comparação entre duas soluções distintas para saber qual é a melhor, é diferente quando comparada a uma otimização de objetivo único. Na MOO, as soluções são geralmente comparadas umas às outras através do domínio de Pareto [Ishibuchi et al. 2008]. Dados dois vetores $\vec{x}$, $\vec{y} \in \mathbb{R}^n$, $\vec{x}$ domina $\vec{y}$ (denotado por $\vec{x} \prec \vec{y}$) se $\vec{x}$ é melhor que $\vec{y}$ em ao menos um objetivo e $\vec{x}$ não é pior que $\vec{y}$ em nenhum objetivo. $\vec{x}$ não é dominado se não existir outra solução na população atual que domine $\vec{x}$. O conjunto de soluções não dominadas no espaço objetivo é conhecido como frente de Pareto. A Figura 2 mostra algumas soluções candidatas em um problema de otimização bidimensional (duas funções objetivo f_1 e f_2) de minimização. As soluções em formato de círculo são consideradas não dominadas entre si, dominando as soluções quadradas. As soluções circuladas compõem a frente de Pareto e são consideradas as melhores soluções entre todas [Wang et al. 2015].

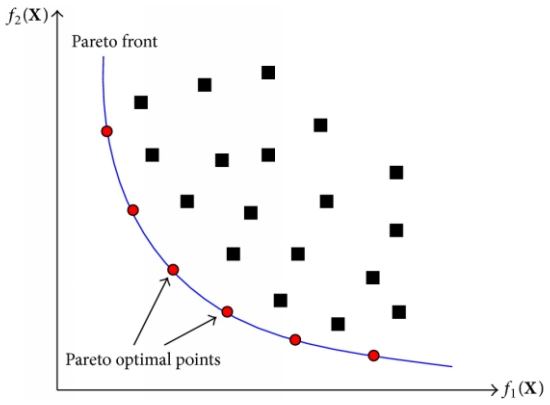

Figura 2. Frente de Pareto em um problema de otimização bidimensional. Fonte: [Wang et al. 2015]

Para um melhor entendimento de como os algoritmos MOO funcionam, o Algoritmo 1 apresenta etapas gerais de todo o processo de busca.

Algoritmo 1: Algoritmo de Otimização Multi-Objetivo.

```
PARAMETERS:
pop_size: O limite de candidatos a serem avaliados por geração.
front: Soluções não dominadas encontradas durante a busca.
    candidates = random_initialization(pop_size)
    evaluate_fitness(candidates)
    front = []
    update_front(front, candidates)
1:  while !stop_criterion do
        parents = select(candidates)
        offspring = crossover(parents)
        mutate(offspring)
        evaluate_fitness(offspring)
        candidates = replace(candidates, offspring)
        update_front(front, candidates)
    end
    return front
```

É importante mencionar que existem algoritmos MOO com diferentes inspirações, e o Algoritmo 1 é uma abordagem evolucionária baseada em população que usa domínio de Pareto. Inicialmente, uma população de pop_size soluções é gerada aleatoriamente. Em seguida, cada solução tem seus valores de aptidão avaliados por cada função objetivo, e a frente de Pareto, vazia no início, é atualizada com soluções não dominadas de $candidates$. Depois disso, um procedimento iterativo começa; enquanto a condição de parada não é satisfeita, operadores evolutivos de seleção, cruzamento e mutação são aplicados. Uma vez que os filhos são gerados a partir dos candidatos selecionados, cada um deles tem sua aptidão avaliada. Esses descendentes promissores substituem as soluções dominadas em $candidates$, de modo que o tamanho da população pop_size seja o mesmo. Finalmente, a frente de Pareto é atualizada. A saída do algoritmo MOO é a frente de Pareto resultante.

3. Trabalhos Relacionados

É importante mencionar que a otimização de arquitetura e parâmetros de Redes Neurais Artificiais (RNAs) não é um problema recente. Diferentes algoritmos de otimização já foram empregados com sucesso para otimizar ANNs [Schaffer et al. 1992]. Entretanto, DNNs foram propostas, e suas arquiteturas se tornaram cada vez mais complexas, de modo que ainda existem poucos estudos que tentam otimizá-las, e algumas lacunas precisam ser melhor investigadas [Elsken et al. 2018]. Neste contexto, apresentamos importantes trabalhos que abordaram o problema de otimização de DNNs estruturadas em cadeia.

David e Greental [David and Greental 2014] usaram um Algoritmo Genético (GA) para otimizar os pesos da camada de codificação da DNN, e adotaram o erro médio quadrático (RMSE) como função objetivo. Young et al. [Young et al. 2015] usou um GA para otimizar o tamanho do kernel e o número de filtros de cada uma das camadas convolucionais, e considerou a taxa de erro como função objetivo. Lorenzo et al. [Lorenzo et al. 2017] usou um algoritmo de Otimização por Enxame de Partículas (PSO) para otimizar os parâmetros de um SimpleNET (modelo de DNN existente). Os parâmetros considerados foram o tamanho e o número de campos receptivos das camadas convolucionais, e o *stride* e o tamanho do campo receptivo da camada de *max pooling*. Eles usaram a taxa de erro de classificação como função objetivo.

Os trabalhos a seguir consideraram não apenas a otimização dos parâmetros de DNNs (como os trabalhos anteriores), mas também a composição e a sequência de suas camadas. Diniz et al. [Diniz et al. 2018] propôs uma abordagem que utiliza *Genetic Based Genetic Programming* (GGP) para otimizar as arquiteturas de rede neural convolucional (parâmetros e camadas) considerando a acurácia como função objetivo. Miikkulainen et al. [Miikkulainen et al. 2019] propôs o CoDeepNEAT, uma técnica de neuroevolução para otimizar arquiteturas da DNN. Essa abordagem otimiza 15 hiper-parâmetros de uma rede neural convolucional, considerando como função de *fitness* a média entre três métricas (BLEU, METEOR e CIDEr) normalizadas pelos seus valores de referência. Assunção et al. [Assunção et al. 2018] recentemente propôs uma abordagem que combina o GA com GGP para gerar uma lista sequencial de camadas e seus parâmetros. Este trabalho usou a acurácia como função objetivo.

Tratar o problema de seleção de arquitetura de DNN como um problema de

otimização com um único objetivo não é adequado [Miikkulainen et al. 2019]. O problema atual é naturalmente um problema multi-objetivo no qual diferentes funções objetivo precisam ser satisfeitas. Ao contrário dos trabalhos mencionados acima, a seguir, apresentamos trabalhos que tratam da seleção de arquiteturas de DNN como um problema multi-objetivo. Liu et al. [Liu et al. 2015] usou o algoritmo de rede neural convolucional multi-objetivo (MOCNN). Este algoritmo não é evolutivo. O processo de treinamento possui um procedimento de otimização interna, que minimiza as perdas de termos unários e pares, respectivamente, através de uma rede convolucional unificada. Este trabalho aplicou o MOCNN para rotulagem facial. Finalmente, Yang et al. [Yang et al. 2019] usou um Algoritmo Evolutivo de Múltiplos Objetivos (MOEA) para otimizar a poda de arquiteturas de DNN pré-existentes de acordo com o *trade-off* entre taxa de erro, custo computacional e dispersão.

Como se pode ver, poucos estudos investigaram a seleção de arquitetura de DNN como um problema multi-objetivo. Diferentemente, dos estudos acima, propomos um método evolutivo multi-objetivo para a seleção automática de arquiteturas DNN estruturadas em cadeia, capazes de construir *novas arquiteturas*, satisfazendo múltiplos objetivos. Neste processo de otimização, nosso método retorna uma arquitetura de DNN com uma sequência de camadas. De acordo com nosso conhecimento, este tipo de otimização ainda não foi investigado no cenário multi-objetivo.

4. Proposta

Neste trabalho, tratamos o problema de selecionar uma arquitetura adequada para DNNs estruturadas em cadeia como uma tarefa de otimização com vários objetivos. Nossa intenção é encontrar a melhor composição e sequência de camadas para a DNN, satisfazendo os objetivos. O problema em questão é combinatório e possui um espaço de soluções de alta dimensão. Devido à natureza do problema, decidimos adotar um algoritmo evolucionário amplamente utilizado na literatura, o *Nondominated Sorting Genetic Algorithm* II (NSGA-II) - mais detalhes deste algoritmo podem ser encontrados em [Deb et al. 2002]).

O problema de selecionar arquiteturas para DNNs é uma tarefa difícil, porque uma única arquitetura pode ser composta por diferentes números e tipos de camadas, cada uma com diferentes hiper-parâmetros, e a ordem, na qual as camadas estão posicionadas, é importante. Assim, a principal contribuição deste trabalho é um método que busca a melhor sequência de camadas para a DNN, dado um problema de classificação, utilizando algoritmos de otimização multi-objetivo. A seguir, apresentamos como o problema de seleção de arquitetura de DNNs estruturadas em cadeia foi formulado para ser resolvido por um algoritmo evolucionário multi-objetivo.

4.1. Representação do Indivíduo

Um indivíduo ou cromossomo C em um algoritmo evolucionário é uma solução para o problema que se quer resolver. No problema em questão, pretende-se alocar uma sequência de camadas $C = (L_1, L_2, ..., L_n)$, onde L_i é a $i^{ésima}$ camada no cromossomo e n é o número total de camadas. Nesse caso, cada cromossomo representa uma possível sequência de camadas (candidata) para a arquitetura da DNN.

Neste trabalho, o processo de otimização de arquitetura de uma DNN considera o número, ordem e tipo das camadas. O cromossomo é um vetor de inteiros de tamanho n, e

cada posição pode assumir um valor no intervalo $[0, 3]$. O valor 0 (zero) indica que nessa posição do vetor nenhuma camada foi escolhida e os valores 1, 2 e 3 representam os tipos de camada *Max Pooling*, *Convolutional* e *Dense*, respectivamente. Todas essas camadas são configuradas com seus parâmetros padrão. É importante mencionar que as primeiras e últimas camadas foram fixadas como *Convolutional* e *Dense*. Essa restrição foi definida para evitar arquiteturas incorretas. A Figura 3 mostra um exemplo de cromossomo (considerando $n = 7$). Como se pode ver, a primeira e última camadas (células pretas) são fixadas como camadas *Convolutional* e *Dense*, respectivamente, e as outras cinco células têm valores variáveis no intervalo $[0, 3]$. Neste exemplo, o cromossomo tem cinco valores diferentes de zero, então existem cinco camadas na DNN, exatamente na mesma ordem em que estão no vetor.

Ainda considerando o exemplo da Figura 3, onde cada cromossomo é representado por um vetor de tamanho $n = 7$ (sendo $n - 2$ camadas variáveis, dado que 2 são fixas), existem 1026 combinações possíveis de arquiteturas. O objetivo do algoritmo MOO é buscar no espaço de soluções aquelas não dominadas que satisfaçam os objetivos. Como o cálculo do vetor de *fitness* de cada solução é a execução da própria DNN, configurada com a sequência de camadas representada pela solução, no problema de entrada, a otimização se torna muito cara, principalmente quando o valor de n é grande.

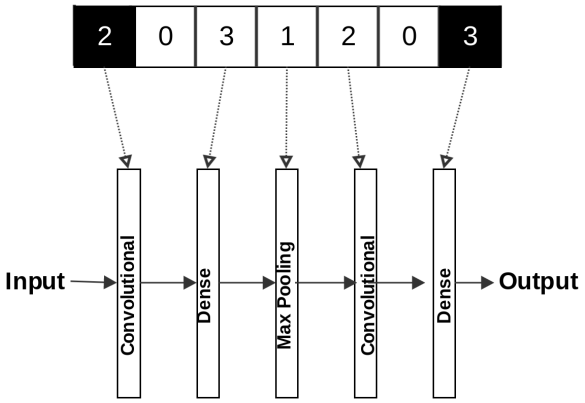

Figura 3. Exemplo de cromossomo em que $n = 7$. Fonte: autor.

4.2. Avaliação do Indivíduo

Uma vez definida a representação dos indivíduos, a seguir, apresentamos as funções objetivo usadas neste trabalho para avaliar a aptidão (*fitness*) dos indivíduos. Para avaliar a aptidão de um indivíduo, é criada uma DNN com a sequência de camadas que o indivíduo representa, e a executamos em um determinado problema. Para isso, aplicamos um procedimento de validação cruzada 10-*fold* repetido 30 vezes. O desempenho de uma arquitetura de DNN não pode ser avaliado considerando apenas a acurácia, porque ela pode não ser representativa. Portanto, adotamos dois objetivos para avaliar o desempenho da arquitetura da DNN:

Acurácia (Acc): Taxa total de acertos do classificador independentemente das classes dos exemplos. Este critério é avaliado através da seguinte equação:

$$Acc = (TP + TN)/(TP + FP + FN + TN), \tag{4}$$

onde TP e TN significam verdadeiro positivo e negativo, e FP e FN significam falso positivo e negativo.

F_1 **score** (F_1): F_1 (também chamada de *f-measure*) é a média harmônica entre *Recall* e *Precision*. O *trade-off* entre *Recall* e *Precision* só tem um valor alto quando ambas as métricas têm valores altos. F_1 é avaliado pela seguinte equação:

$$F_1 = (2 * Recall * Precision)/(Recall + Precision). \quad (5)$$

Os valores de cada objetivo variam de 0 a 1 e o algoritmo de otimização multi-objetivo tenta encontrar uma solução que satisfaça ambos os objetivos.

5. Metodologia

Nesta seção, apresentamos a metodologia utilizada para avaliar o método proposto. Além disso, os conjuntos de dados e as configurações experimentais usadas aqui também são detalhados. Neste trabalho foi realizada uma análise que avalia o desempenho do método proposto, comparando-o com o método exaustivo e aleatório (ambos em suas versões multi-objetivo, considerando as mesmas funções objetivo (Acc e F_1). Para comparações práticas dos resultados, decidimos que, em vez de comparar as frentes de Pareto resultantes encontrados por cada método, apenas uma solução de cada Frente de Pareto seria selecionada. Esta prática é predominante em experimentos envolvendo algoritmos multi-objetivo. Para escolher uma arquitetura de DNN, a partir de um conjunto de soluções não dominadas, considerando os dois critérios de desempenho mencionados anteriormente, aplicamos o método de *Borda count* [Black 1976]. Esse método é um método de eleição de um único vencedor, no qual cada algoritmo é classificado por cada objetivo e, em seguida, uma classificação média é retornada, onde o algoritmo, em primeiro lugar, é o vencedor. O objetivo desta análise é verificar se uma solução não dominada (arquitetura de DNN) encontrada pelo método proposto pode melhorar o processo de classificação, ao invés de apenas fazer uma análise de Pareto.

5.1. Bases de Dados

Para os experimentos, foram utilizados dois bancos de dados: CIFAR 10 [Krizhevsky and Hinton 2009] e MNIST [Deng 2012]. Essas bases de dados são compostas de várias imagens e são comumente usadas como problemas de classificação para avaliar redes neurais. A popularidade destas bases é devido ao fato de que são robustas e possuem imagens reais.

CIFAR 10: Base de dados de 50.000 imagens de treinamento coloridas 32x32, rotuladas em 10 categorias e 10.000 imagens de teste (veja a Figura 4 - esquerda).

MNIST: Base de dados de 60.000 imagens de escala de cinza de 10 dígitos 28x28 (0-9), junto com um conjunto de teste de 10.000 imagens. O banco de dados foi construído a partir do *NIST Special Database 3* e do *Special Database 1*, que contêm imagens binárias de dígitos manuscritos (ver Figura 4 - direita).

5.2. Configurações

Neste trabalho, utilizamos como algoritmo multi-objetivo o NSGA-II. O algoritmo MOO foi definido com seus parâmetros padrão, definidos na biblioteca Platypus [Hadka 2017].

Figura 4. Imagens de amostra da CIFAR 10 (esquerda) e MNIST (direita).

O tamanho populacional igual a 10, o número máximo de gerações igual a 50 (totalizando 500 avaliações de aptidão), bem como o tamanho dos cromossomos $n = 10$, (onde 8 camadas são variáveis e 2 fixas, totalizando 65.538 possíveis candidatos), foram definidos empiricamente, levando em consideração o alto custo computacional do problema. Enfatizamos, mais uma vez, que todas as camadas foram configuradas com seus parâmetros padrão, provenientes da biblioteca Keras [Chollet et al. 2015]. A Tabela 1 mostra os parâmetros padrão de cada tipo de camada. Para a execução da DNN, valores padrão como $batch\_size = 200$, $epochs = 50$, $learning\_rate = 0,1$, $impulse = 0,8$ e $shuffle = True$ foram usados.

Tabela 1. Parâmetros padrão de cada tipo de camada da DNN.

Tipo da Camada	***Parâmetros***
Conv2D	$filter = 5$ $kernelsize = 4$ $padding = "same"$
Max Pooling2D	$poolsize = [2, 2]$ $strides = 2$
Dense	$units = 5$ $activation = "softmax"$

Para os experimentos, o método proposto foi executado 10 vezes para gerar um desempenho médio. A mesma metodologia foi realizada para o método aleatório, uma simulação com 500 avaliações de aptidão e executada 10 vezes para gerar um desempenho médio. No caso do método exaustivo, todas as arquiteturas candidatas de 65.538 foram avaliadas.

Para desenvolver este trabalho, usamos as seguintes bibliotecas: Scikit learn [Pedregosa et al. 2011] para o uso de aprendizado de máquina, Keras [Chollet et al. 2015] para construir DNNs e Platypus [Hadka 2017] para o uso de algoritmos MOO. Essas bibliotecas são robustas e amplamente utilizadas na literatura. Todas as execuções foram realizadas em um laptop com um processador Intel Core i7-7500U com 4Mb de memória cache, velocidade de clock de 2,9 GHz e turbo de até 3,5 GHz, e 16 GB de RAM.

6. Resultados e Discussão

Nesta seção, comparamos o desempenho do método proposto com os das abordagens exaustiva e aleatória. A Tabela 2 e a Tabela 3 mostram os resultados de cada abordagem considerando os banco de dados CIFAR 10 e MNIST, respectivamente. Estes resultados

incluem o melhor valor médio de *fitness* e desvio padrão alcançados em cada objetivo (Acc e F_1), e o tempo médio de desempenho do método proposto e aleatório (medido em segundos), após dez execuções. Além disso, as tabelas mostram os resultados das melhores soluções, obtidas pela abordagem exaustiva, e o seu tempo de execução.

Tabela 2. Resultados das abordagens exaustiva, aleatória e proposta, na base de dados CIFAR10.

	Exaustivo	***Aleatório***	***Proposta***
Acc	0,90	0,82 ($\pm 0,0204$)	0,90($\pm 0,0018$)
F_1	0,49	0,04($\pm 0,0172$)	0,42($\pm 0,0202$)
Tempo (s)	>86.400,00	452,34	79.796,73

Tabela 3. Resultados das abordagens exaustiva, aleatória e proposta, na base de dados MNIST.

	Exaustivo	***Aleatório***	***Proposta***
Acc	0,99	0,88 ($\pm 0,0135$)	0,98($\pm 0,0027$)
F_1	0,90	0,0138($\pm 0,1724$)	0,87($\pm 0,00702$)
Tempo (s)	>86.400,00	364,22	25.538,81

A primeira coluna da Tabela 2 e da Tabela 3 mostram os valores ótimos para Acc e F_1 obtidos pela abordagem exaustiva em cada problema, CIFAR 10 e MNIST. Para encontrar os valores ótimos, o método exaustivo avaliou todas as combinações possíveis para a DNN, uma tarefa cujo custo é extremamente alto. Por outro lado, o método aleatório tem um tempo de execução menor ($452,34s$ no CIFAR 10 e $364,22s$ no MNIST) quando comparado ao método exaustivo. No entanto, devido ao grande número de possíveis soluções, a chance de encontrar um boa arquitetura para a DNN por sorte (aleatoriamente) é pequena. Como se pode ver, o método aleatório alcançou $Acc = 0,82$ e $F_1 = 0,04$ para o CIFAR 10 e $Acc = 0,88$ e $F_1 = 0,0138$.

Em relação ao método proposto, podemos ver que ele foi capaz de encontrar boas soluções em muito menos tempo se comparado à abordagem exaustiva. Para o CIFAR 10, a proposta encontrou uma solução com $Acc = 0,90$ e $F_1 = 0,42$, próximo da solução ótima. O mesmo para a base MNIST, onde a proposta encontrou uma solução com $Acc = 0,98$ e $F_1 = 0,87$. Quando comparado à abordagem aleatória, o método proposto a superou em todos os objetivos e também obteve menores valores de desvio padrão, significando que seus resultados tiveram menos variação. Embora o método proposto e o método aleatório tenham a mesma condição de parada (500 avaliações de aptidão), o método aleatório tem um tempo de execução muito menor que o proposto nos problemas CIFAR 10 e MNIST. Será que, se o método aleatório tivesse o mesmo tempo de execução, ele poderia superar a abordagem proposta? Neste trabalho, também realizamos essa análise. A Tabela 4 mostra os resultados obtidos pelo método aleatório, sendo executado com mesmo tempo e condições de execução da proposta no CIFAR 10 e MNIST, respectivamente. Como se pode ver na Tabela 4, mesmo se executarmos o método aleatório tendo o mesmo tempo do método proposto, a nossa abordagem continua a superá-lo em todos os objetivos.

De acordo com os resultados, a proposta foi capaz de lidar com a tarefa de seleção de arquitetura para a DNN. O método proposto diminui a interferência humana nesta

Tabela 4. Resultados das abordagens aleatória e proposta (ambos com o mesmo tempo e condições de execução) em CIFAR 10 e MNIST.

CIFAR 10		
Acc	Aleatória	0,88 ($\pm 0,0282$)
	Proposta	0.90(± 0.0018)
F_1	Aleatória	0,03 ($\pm 0,0255$)
	Proposta	0,42($\pm 0,0202$)
MNIST		
Acc	Aleatória	0,90 ($\pm 0,0197$)
	Proposta	0.98 (± 0.0027)
F_1	Aleatória	0,095($\pm 0,1880$)
	Proposta	0,87 ($\pm 0,0070$)

tarefa, fornecendo automaticamente uma DNN otimizada, capaz de alcançar resultados satisfatórios, para um dado problema de classificação.

7. Conclusão e Trabalhos Futuros

Neste trabalho, a seleção de arquiteturas para DNNs estruturadas em cadeia é tratada como um problema de otimização multi-objetivo. Para resolver esse problema, um algoritmo evolucionário, denominado NSGA-II, foi adotado e adaptado para otimizar arquiteturas (sequência de camadas) da DNN, considerando dois objetivos: Acurácia e $F_1 score$. Os resultados mostraram que o método proposto foi capaz de alcançar um desempenho próximo ao método exaustivo, mas em muito menos tempo, e superou a busca aleatória em todos os problemas. No futuro, pretende-se considerar outros tipos de camadas, e considerar os hiper-parâmetros das camadas no processo de otimização.

Referências

Assunção, F., Lourenço, N., Machado, P., and Ribeiro, B. (2018). Evolving the topology of large scale deep neural networks. In *European Conference on Genetic Programming*, pages 19–34. Springer.

Black, D. (1976). Partial justification of the borda count. *Public Choice*, 28(1):1–15.

Chollet, F. et al. (2015). Keras. `https://keras.io/`. Accessed: 2019-06-12.

David, O. E. and Greental, I. (2014). Genetic algorithms for evolving deep neural networks. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pages 1451–1452. ACM.

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

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142.

Diniz, J. B., Cordeiro, F. R., Miranda, P. B., and da Silva, L. A. T. (2018). A grammar-based genetic programming approach to optimize convolutional neural network architectures. In *Anais do XV Encontro Nacional de Inteligência Artificial e Computacional*, pages 82–93. SBC.

Elsken, T., Metzen, J. H., and Hutter, F. (2018). Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*.

Hadka, D. (2017). Platypus: A free and open source python library for multiobjective optimization. *Available on Github, vol. https://github. com/Project-Platypus/Platypus*.

Ishibuchi, H., Tsukamoto, N., and Nojima, Y. (2008). Evolutionary many-objective optimization: A short review. In *IEEE World Congress on Computational Intelligence).*, pages 2419–2426. IEEE.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

Liu, S., Yang, J., Huang, C., and Yang, M.-H. (2015). Multi-objective convolutional learning for face labeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3451–3459.

Lorenzo, P. R., Nalepa, J., Kawulok, M., Ramos, L. S., and Pastor, J. R. (2017). Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 481–488. ACM.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. (2019). Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Schaffer, J. D., Whitley, D., and Eshelman, L. J. (1992). Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE.

Wang, W., Zhang, L.-L., Chen, J.-J., and Wang, J.-H. (2015). Parameter estimation for coupled hydromechanical simulation of dynamic compaction based on pareto multiobjective optimization. *Shock and Vibration*, 2015.

Yang, C., An, Z., Li, C., Diao, B., and Xu, Y. (2019). Multi-objective pruning for cnns using genetic algorithm. *arXiv preprint arXiv:1906.00399*.

Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S.-H., and Patton, R. M. (2015). Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, page 4. ACM.