Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, v. 12, n. 1, 2026.

Trabalho apresentado no XLIV CNMAC, Fundação Getulio Vargas - Rio de Janeiro - RJ, 2025.

**Proceeding Series of the Brazilian Society of Computational and Applied Mathematics**

# Comparando Métodos de Aprendizagem Profunda para a Seleção de Precondicionadores

Michael Souza[1]
UFC, Fortaleza, CE,
Luiz Mariano Carvalho[2]
UERJ, Rio de Janeiro, RJ,
Douglas Augusto[3]
FOC, Rio de Janeiro, RJ ,
Jairo Panetta[4]
ITA, São José dos Campos, SP,
Paulo Goldfeld[5]
UFRJ, Rio de Janeiro, RJ,
José R. P. Rodrigues[6]
CENPES/Petrobras, Rio de Janeiro, RJ

**Resumo**. Em computação de alto desempenho (HPC), a solução eficiente de grandes sistemas lineares esparsos é fundamental, sendo os métodos iterativos a escolha predominante, cuja performance está ligada ao precondicionador escolhido. Apresentamos uma representação da esparsidade de matrizes por meio de imagens RGB. Utilizando uma rede neural convolucional (CNN), a tarefa de seleção do precondicionador se transforma em um problema de classificação multi-classe. Testes com 126 matrizes da coleção SuiteSparse enfatizam a adequação do modelo CNN, observando um aumento de 32% na acurácia e uma redução de 25% no tempo de execução.

**Palavras-chave**. computação de alto desempenho (HPC), sistemas lineares esparsos, métodos iterativos, escolha de precondicionadores, rede neural convolucional, classificação multi-classe.

## 1 Introdução

Os métodos de Krylov são os solvers lineares preferidos para simulações numéricas em, por exemplo, engenharia de reservatórios que exigem a solução de grandes sistemas lineares com matrizes esparsas [5]. No entanto, seu sucesso depende da escolha de um precondicionador adequado [14]. Discutiremos uma técnica para representar graficamente propriedades estruturais ou matemáticas das matrizes esparsas envolvidas na escolha de precondicionadores. Esta representação deve ser escalável e, para grandes sistemas esparsos, o ideal é que a complexidade computacional seja linear em relação ao número de elementos não nulos.

Entre os atributos da matriz, devemos destacar a ordem, seus autovalores e valores singulares, o número de condicionamento, o padrão de esparsidade, a densidade e a dominância diagonal, entre outros. Com a exceção do padrão de esparsidade, esses atributos são numéricos. A esparsidade

[1]michael@ufc.br
[2]luizmc@ime.uerj.br
[3]daa@fiocruz.br
[4]jairo.panetta@gmail.com
[5]goldfeld@matematica.ufrj.br
[6]jrprodrigues@petrobras.com.br

 

encapsula atributos topológicos sobre a conexão entre os elementos não nulos da matriz. Embora não seja o único fator determinante, o padrão de esparsidade afeta o nível de paralelismo possível tanto na aplicação quanto na construção de precondicionadores como ILU($k$) [14].

Exploramos algumas técnicas de aprendizagem profunda (AP) para obter automaticamente representações compactas de matrizes esparsas para selecionar precondicionadores. Ampliando a abordagem de Yamada et al. [16], usamos imagens RGB para codificar espacialmente os padrões de esparsidade. Diferentemente do trabalho de Yamada, nossa metodologia incorpora a classificação de vários rótulos para identificar uma gama de precondicionadores adequados para uma determinada matriz esparsa. Para processar as representações baseadas em imagens, empregamos uma rede neural convolucional (CNN) [10]. Há vários trabalhos que utilizam AP para tratar problemas em álgebra linear computacional, por exemplo [1, 3, 6], para maiores detalhes, ver [15].

Nossas contribuições são as seguintes: adotamos um modelo multi-rótulo para a seleção de diversos precondicionadores para uma única matriz, lidando melhor com casos em que vários precondicionadores têm desempenhos semelhantes; comparamos modelos baseados em atributos numéricos com modelos baseados em imagens. Os modelos baseados em imagens superam os modelos baseados em dados numéricos, com uma probabilidade 32% maior de sugestão de precondicionador ideal e uma chance 26% maior de baixo impacto computacional.

Este texto resume alguns dos resultados de um trabalho em desenvolvimento cuja versão inicial se encontra disponível em [15]; vários detalhes omitidos aqui, devido ao limite do número de páginas, podem ser encontrados nesse trabalho. O restante deste artigo está assim estruturado: a Seção 2 descreve a metodologia, a Seção 3 discute os resultados numéricos e a Seção 4 conclui o artigo, destacando sua importância e sugerindo futuras direções de pesquisa.

## 2 Metodologia

Propomos codificar a esparsidade da matriz e alguns atributos facilmente computáveis através de uma imagem RGB. Realizamos experimentos comparando modelos de AP treinados em atributos numéricos da matriz com aqueles treinados em imagens. Empregamos 126 matrizes simétricas, positivo-definidas (SPD) e não diagonais da coleção de matrizes SuiteSparse[8].

Utilizamos um conjunto usual de parâmetros escalares para caracterizar as matrizes: densidade, número de linhas, número de elementos não nulos, número médio de elementos não nulos por linha, condicionamento, autovalores extremos, total de linhas com diagonal dominância e razão mínima entre o valor absoluto do elemento diagonal e a soma das entradas não diagonais. Entre esses parâmetros, o número da condicionamento e os autovalores foram calculados usando o MATLAB. Embora o número de condicionamento de uma matriz SPD seja definido por seus autovalores extremos, os atributos que empregamos são estimativas, o que os torna não redundantes.

Para representar visualmente as propriedade da matriz, criamos um conjunto de dados de imagem com base no método proposto por Yamada et al. [16]. Cada matriz $A$ é particionada em blocos $A_{ij}$ de dimensão $b \approx N/m$, em que $N$ é a dimensão da matriz e $m$ é a resolução da imagem. Esses blocos são representados como pixels $p_{ij}$ em uma imagem de $m \times m$ pixels. Geramos quatro conjuntos de imagens com $m \in \{32, 64, 128, 256\}$.

Cada pixel $p_{ij}$ consiste em três componentes alinhados com os canais RGB: vermelho, verde e azul. O canal vermelho captura a magnitude dos elementos diferentes de zero no bloco da matriz, o azul incorpora as dimensões da matriz e o verde transmite a densidade do bloco. Como resultado, as características distintas de cada matriz são visualmente representadas, com os atributos de cada bloco influenciando a cor do pixel correspondente.

A Figura 1 ilustra a conversão de uma matriz $20 \times 20$ em uma imagem de $5 \times 5$ pixels. Aqui, cada pixel na imagem corresponde a um bloco de $4 \times 4$ da matriz.

A representação da esparsidade como imagens captura naturalmente as informações topológicas,

DOI: 10.5540/03.2026.012.01.0262  

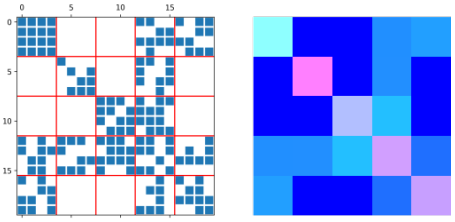


Figura 1: Representação de matriz $20 \times 20$ em blocos $5 \times 5$. Fonte: Os autores.

codificando-as em relações geométricas, que são mais difíceis de serem obtidas com representações escalares. Esse processo gerou quatro conjuntos de atributos escalares estendidos, correspondentes a cada $m \in \{32, 64, 128, 256\}$. Para gerenciar o aumento da dimensionalidade e reter os aspectos mais informativos, usamos a análise de componentes principais (PCA), capturando 99% da variação nos dados [7].

Consideramos dez precondicionadores: Fatoração LU Incompleta com níveis 0 (ILU-0) e 1 (ILU-1), Bloco Jacobi (BJACOBI), Sobrerelaxação Sucessiva (SOR), Point Block Jacobi (PBJACOBI), Multigrid (MG), Jacobi (JACOBI), Fatoração Incompleta de Cholesky (ICC), Multigrid Algébrico Geométrico (GAMG), Eisenstat (EISENSTAT). Estes precondicionadores estão disponíveis no PETSc [2] e foram acessados empregando-se a opção `-pc` com os seus respectivos valores *default*. Geramos dez sistemas lineares aleatórios para cada matriz. Os vetores solução $x^\star$ foram extraídos de uma distribuição normal padrão e os vetores do lado direito foram calculados como $b = Ax^\star$. Em seguida, resolvemos cada sistema usando o método de Gradiente Conjugado Precondicionado (PCG) [14].

Para cada sistema linear e precondicionador, executamos o método PCG cinco vezes, calculamos o tempo mediano de resolução e somamos esses tempos medianos de todos os dez sistemas para cada precondicionador.

Algumas configurações não convergiram para soluções aceitáveis em tempo hábil. Para compatibilizar as medidas de performance, utilizamos o resíduo e o erro relativos. Seja $\bar{x}$ o vetor solução estimado para o sistema $Ax = b$. Neste caso, o resíduo e o erro relativos são, respectivamente:

$$r(\bar{x}) = \frac{\|A\bar{x} - b\|_2}{\|b\|_2} \quad \text{e} \quad e(\bar{x}) = \frac{\|\bar{x} - x^\star\|_2}{\|x^\star\|_2}, x^\star \neq 0. \tag{1}$$

Definimos um par (matriz, precondicionador) como *viável* se, em todos os 10 sistemas lineares, tivermos $r(\bar{x}) < 0,01$, $e(\bar{x}) < 0,1$ e o tempo total para obtenção das soluções for inferior a um minuto. Cada par que não atendeu a essas condições foi rotulado como *inviável*. Com esse limite de tempo (um minuto), todas as matrizes obtiveram pelo menos um resolvedor viável.

Para cada matriz $A$ e precondicionador $P$, diremos que $P$ é um *precondicionador ótimo* se $(A, P)$ for viável e seu tempo de execução for, no máximo, 10% mais lento que o precondicionador viável mais rápido para a mesma matriz. Esta definição resultou em 126 conjuntos de precondicionadores ótimos, um conjunto por matriz.

Definimos o problema de seleção de precondicionadores como uma tarefa de classificação multi-rótulo, onde a entrada são os atributos da matriz e a saída é um conjunto de precondicionadores ótimos. Consideramos dois tipos de dados de entrada: propriedades escalares e dados de imagem. Usamos as seguintes bibliotecas para implementar os modelos de AP: scikit-learn, TensorFlow e Keras [4, 11, 12].

Aplicamos os seguintes classificadores ao conjunto de dados de propriedades escalares: Logistic Regression, Support Vector Classifier (SVC), Decision Tree, Random Forest, Gradient Boosting, K-Nearest Neighbors, Multi-layer Perceptron (MLP) e Gaussian Naive Bayes (GNB) — a descrição

 

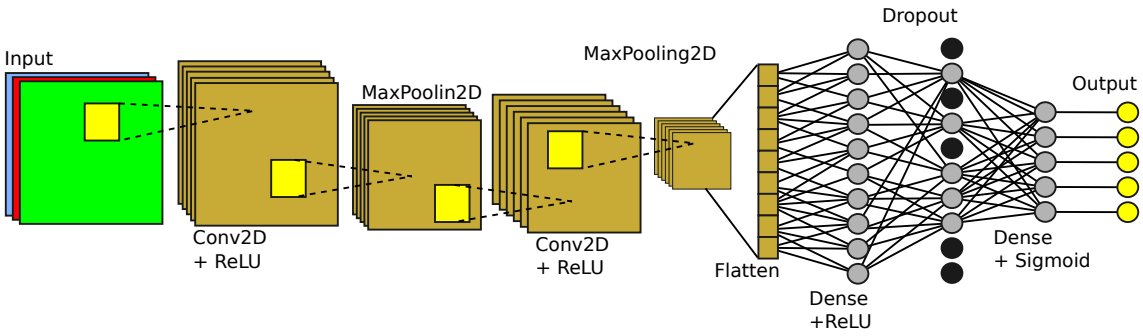


Figura 2: Arquitetura de Rede Neural Convolucional. Fonte: Os autores.

desses métodos está fora do escopo deste artigo, ver [9] para detalhes. Cada classificador foi utilizado com seus parâmetros *default*.

Para as imagens RGB, empregamos uma CNN [13] com a seguinte arquitetura: uma camada de entrada, que aceita os dados da imagem, aplica uma operação de convolução e uma função de ativação de Unidade Linear Retificada (ReLU) para introduzir não linearidade no modelo; a camada subsequente é um *pooling* máximo com uma operação $2 \times 2$ na saída da camada de entrada; agora temos outra convolução; uma outra camada de *pooling* máximo reduzindo ainda mais as dimensões do mapa de atributos; uma camada para o achatamento, o mapa de atributos resultante torna-se um vetor 1D; uma camada totalmente conectada consistindo em 128 neurônios e usando a função de ativação ReLU que aceita a saída achatada da camada anterior; uma camada de dropout para evitar *overfitting*, desconsiderando aleatoriamente 50% dos neurônios da camada anterior; finalmente, a camada de saída emprega uma função de ativação sigmoide para produzir a saída final do modelo, o número de neurônios nesta camada corresponde ao número de classes (precondicionadores) no conjunto de dados. A Figura 2 apresenta uma representação gráfica desta arquitetura CNN.

Algoritmos de classificação multi-rótulos podem produzir um conjunto vazio de rótulos (precondicionadores) como saída. Nesses casos, adotamos a alternativa de selecionar o precondicionador mais frequente no conjunto de dados, ou seja, ILU-1. Esta alternativa seria uma escolha natural para um usuário sem informações adicionais.

# 3 Resultados

O ambiente computacional foi um processador Intel(R) Xeon(R) CPU E5-2650 v2 @ 2,60GHz com 64GB de RAM, rodando Debian GNU/Linux 11 e C++ compilado com GNU g++ (Debian 10.2.1-6) com diretivas: -std=c++17 -O3 -lrt -fPIC -m64 -march=nativo -mtune=nativo -fopenmp-simd; PETSc compilado com mpicxx no GNU g++ (Debian 3.3.1-5). Testes executados sequencialmente com uma única thread e um processo MPI. TensorFlow (2.12.0) e scikit-learn (1.2.2) [11, 12] forneceram o ambiente de AP.

Avaliamos nossos modelos usando duas métricas: acurácia e desaceleração. A **acurácia** mede a proporção de precondicionadores ótimos previstos corretamente. Para a matriz $i$, onde $Y_i$ é o conjunto de precondicionadores ótimos e $\hat{Y}_{ij}$ os precondicionadores previstos pelo modelo $j$. A acurácia é dada por: $\frac{|Y_i \cap \hat{Y}_{ij}|}{|\hat{Y}_{ij}|}$. A **desaceleração** é a razão entre o tempo de execução do precondicionador mais rápido previsto pelo modelo e o tempo de execução do precondicionador ótimo mais rápido. Para a matriz $i$ e o modelo $j$, $t_{ijk}$ é o tempo de execução do precondicionador $k$ previsto pelo modelo $j$, e $t_i^*$ é o tempo de execução do precondicionador ótimo mais rápido,

DOI: 10.5540/03.2026.012.01.0262  

desaceleração$_{ij} = \frac{\min_k t_{ijk}}{t_i^*}$. Essas métricas são complementares: a desaceleração considera apenas o melhor precondicionador previsto, enquanto a acurácia penaliza os modelos que preveem muitos precondicionadores não ideais. Usamos uma divisão de treinamento-teste de 80%-20% e repetimos cada ciclo de treinamento-teste-avaliação 30 vezes objetivando resultados confiáveis e atenuando a tendência de qualquer partição de dados única.

A Tabela 1 apresenta a probabilidade de cada classificador obter acurácia perfeita ($P$(acurácia = 1)) e desaceleração aceitável ($P$(desaceleração $< 1,5$)). Também mostra a quantidade de precondicionadores ótimos previstos. O MLP obteve a maior probabilidade de acurácia perfeita, 0,57, enquanto o GNB apresentou a maior probabilidade de desaceleração aceitável, 0,79. O número médio de precondicionadores ideais por matriz é de 1,29. O GNB prevê uma média de 4,9 precondicionadores por matriz; esta previsão excessiva reduz sua acurácia, embora capture o precondicionador ideal. MLP , SVC, Random Forest e K-Nearest Neighbors demonstram um desempenho mais equilibrado, com probabilidades de acurácia acima de 0,55 e probabilidades de desaceleração acima de 0,6.

Tabela 1: Acurácia e a desaceleração para classificadores escalares. Precondicionadores previstos.

| Classificador | $P$(acur. = 1) | $P$(desac. < 1.5) | Previstos |
|---|---|---|---|
| MLP | **0,57** | 0,69 | 1,07 |
| SVC | 0,56 | 0,64 | 1,00 |
| Random Forest | 0,55 | 0,70 | 1,13 |
| K-Nearest Neighbors | 0,55 | 0,66 | 1,05 |
| Logistic Regression | 0,48 | 0,61 | 1,10 |
| Gradient Boosting | 0,47 | 0,68 | 1,29 |
| Decision Tree | 0,39 | 0,69 | 1,49 |
| Gaussian Naive Bayes | 0,05 | **0,79** | 4,91 |

Avaliamos CNNs em imagens matriciais com quatro resoluções diferentes: 32×32, 64×64, 128×128 e 256×256 pixels, denotados como `CNN_32`, `CNN_64`, `CNN_128` e `CNN_256`, respectivamente.

Conforme mostrado na Tabela 2, os modelos CNN têm desempenho excepcional. O `CNN_256` obteve a maior probabilidade de precisão perfeita com 0,89. Em termos de métricas de desaceleração, tanto o `CNN_256` quanto o `CNN_128` mantiveram uma probabilidade de 0,94 de atingir um fator de desaceleração abaixo de 1,5. Embora o desempenho geralmente melhore com o aumento da resolução da imagem, os ganhos diminuem além de 128×128 pixels, sugerindo que imagens de altíssima resolução podem não oferecer benefícios adicionais.

Tabela 2: Acurácia e desaceleração para classificadores baseados em imagem.

| Classificador | $P$(acurácia = 1) | $P$(desaceleração < 1.5) |
|---|---|---|
| CNN_256 | **0,89** | **0,94** |
| CNN_128 | 0,88 | **0,94** |
| CNN_64 | 0,86 | 0,93 |
| CNN_32 | 0,82 | 0,89 |

Finalmente, comparamos os modelos de melhor desempenho dos métodos escalares e baseados em imagem. Como linha de base, incluímos um classificador de referência que sempre seleciona o precondicionador que ocorre com mais frequência no conjunto de dados (ILU-1). Como a Tabela 3 indica, os modelos CNN superam o desempenho do modelo de referência e dos modelos baseados

 

em escalares. O modelo `CNN_256` obteve uma probabilidade 32% maior de precisão perfeita em comparação com o melhor modelo baseado em parâmetros escalares e uma melhoria de 24% em relação ao ILU-1 puro. Os modelos CNN apresentaram vantagem na manutenção de desaceleração aceitável. Esses resultados indicam, sobre este conjunto de matrizes, que as representações baseadas em imagem dos padrões de esparsidade da matriz capturam informações mais relevantes para a seleção do precondicionador do que os modelos baseados em parâmetros escalares tradicionais.

Tabela 3: Desempenho comparativo dos principais modelos de cada abordagem.

| Classificador | $P(\text{acurácia} = 1)$ | $P(\text{desaceleração} < 1.5)$ |
|---|---|---|
| Benchmark (ILU-1) | 0,49 | 0,58 |
| MLP | 0,57 | 0,69 |
| SVC | 0,56 | 0,64 |
| Random Forest | 0,55 | 0,70 |
| CNN_256 | **0,89** | **0,94** |
| CNN_128 | 0,88 | **0,94** |
| CNN_64 | 0,86 | 0,93 |

# 4 Considerações Finais

Nosso estudo empírico, utilizando 126 matrizes SPD da SuiteSparse Matrix Collection, revela resultados relevantes. A abordagem baseada em CNN demonstra melhorias em relação aos métodos tradicionais - um aumento de 32% na probabilidade de obter precisão perfeita e uma melhoria de 25% na manutenção de uma desaceleração computacional aceitável.

Neste estudo, os modelos baseados em parâmetros escalares foram usados com suas configurações padrão sem ajuste extensivo de parâmetros. Isso sugere que, com a otimização adequada, esses modelos podem alcançar um desempenho melhor do que o relatado aqui. No entanto, a grande diferença de desempenho entre os modelos escalares não ajustados e nossa abordagem CNN indica que o aprimoramento decorre principalmente da forma como as informações são representadas e não da sofisticação do modelo. A codificação de padrões de esparsidade de matriz como imagens captura informações estruturais que os parâmetros escalares perdem.

A abordagem baseada em imagens oferece uma alternativa para a seleção automatizada de precondicionadores em solucionadores iterativos. No entanto, uma comparação justa entre as versões totalmente otimizadas de ambas as abordagens forneceria informações adicionais sobre seus pontos fortes e possíveis aplicações.

No futuro, planejamos estender nossa abordagem CNN para otimizar parâmetros adicionais de métodos iterativos, incluindo a seleção do tipo de solver e parâmetros de tolerância, entre outros. Além disso, uma investigação abrangente sobre modelos baseados em parâmetros escalares ajustados de forma otimizada forneceria referências comparativas valiosas. Essa direção de pesquisa visa fornecer um ajuste mais abrangente e automatizado para aplicativos de computação de alto desempenho que exigem soluções eficientes de sistemas lineares esparsos.

# Referências

[1] J. Ackmann et al. **Machine-Learned Preconditioners for Linear Solvers in Geophysical Fluid Flows**. 2020. DOI: 10.48550/arXiv.2010.02866.

[2] S. Balay et al. **PETSc/TAO Users Manual**. Rel. técn. ANL-21/39 - Revision 3.20. Argonne National Laboratory, 2023. DOI: 10.2172/1968587.

 

[3] M. Barreda et al. "Performance modeling of the sparse matrix–vector product via convolutional neural networks". Em: **The Journal of Supercomputing** 76.11 (2020), pp. 8883–8900. DOI: 10.1007/s11227-020-03186-1.

[4] F. Chollet et al. **Keras**. Acessado em 01/09/2023, https://keras.io. 2015.

[5] L. Gasparini et al. "Hybrid parallel iterative sparse linear solver framework for reservoir geomechanical and flow simulation". Em: **Journal of Computational Science** 51 (2021), p. 101330. ISSN: 1877-7503. DOI: 10.1016/j.jocs.2021.101330.

[6] M. Götz et al. "Machine learning-aided numerical linear algebra". Em: **2018 IEEE/ACM 9th ScalA**. IEEE. 2018, pp. 49–56. DOI: 10.1109/ScalA.2018.00010.

[7] I. Guyon et al. "An introduction to feature extraction". Em: **Feature extraction: foundations and applications**. Springer, 2006, pp. 1–25. DOI: 10.1007/978-3-540-35488-8_1.

[8] S. P. Kolodziej et al. "The SuiteSparse Matrix Collection Website Interface". Em: **Journal of Open Source Software** 4.35 (2019), p. 1244. DOI: 10.21105/joss.01244.

[9] S. B. Kotsiantis et al. "Supervised Machine Learning". Em: **Emerging Artificial Intelligence Applications in Computer Engineering**. Ed. por I. Maglogiannis. Vol. 160. Frontiers in Artificial Intelligence and Applications. Amsterdam: IOS Press, 2007, pp. 3–24. ISBN: 9781586037802.

[10] Z. Li et al. "A survey of convolutional neural networks: analysis, applications, and prospects". Em: **IEEE Trans. on neural net. and learn. sys.** 33.12 (2022), pp. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827.

[11] M. Abadi et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. Acessado em 01/09/2023, https://www.tensorflow.org. 2015. DOI: 10.5555/1953048.2078195.

[12] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". Em: **Journal of Machine Learning Research** 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[13] W. Rawat et al. "Deep Convolutional Neural Networks for Image Classification". Em: **Neural Computation** 29.9 (2017), pp. 2352–2449. DOI: 10.1162/neco_a_00990.

[14] Y. Saad. **Iterative methods for sparse linear systems**. SIAM, 2003. ISBN: 978-0-89871-534-7.

[15] M. Souza et al. **A Comparison of Image and Scalar-Based Approaches in Preconditioner Selection**. 2023. DOI: 10.48550/arXiv.2312.15747.

[16] K. Yamada et al. "Preconditioner auto-tuning using deep learning for sparse iterative algorithms". Em: **2018 Sixth CANDARW**. IEEE. 2018, pp. 257–262. DOI: 10.1109/CANDARW.2018.00055.

 