Trabalho Final 1° Bimestre: Predição do Tamanho de Partículas de Gelo

Frederico Schardong

1 Enunciado do trabalho

O enunciado do trabalho propõem a criação de uma rede neural com objetivo de prever o tamanho de partículas de cristais de gelo a partir da dispersão da luz medida pelo *Small Ice Detector* (SID) [4]. O SID é um instrumento desenvolvido pela Universidade de Hertfordshire que gera dados bidimensionais consequentes da dispersão de luz produzida por partículas de gelo (*Two-Dimensional Light Scattering* (2DSL)). Este instrumento é acoplado ao lado externo de uma aeronave e, ao voar através de uma núvem, emite feixes de luz que são refletidos por partículas de gelo que são captadas por sensores. A partir dos dados destes sensores é possível estimar o tamanho das partículas de gelo.

O enunciado do trabalho também aponta para um repositório online com 162 arquivos no formato `hdf5` [8], cada um contendo resultados de medições de uma partícula de gelo. Cada partícula possui 133 orientações (medições em rotações distintas), e assim como em [9], apenas as elevações entre 6° e 25° (inclusivo) foram consideradas. Mais especificamente, cada partícula possui 133 orientações, cada uma com 20 elevações e cada elevação possui 361 valores (de azimuth) distintos, configurando a entrada do problema como uma matriz de $162 * 133$ linhas e $20 * 361$ colunas. Os tamanhos das partículas (valores a serem previstos pela rede neural) estão presentes em cada arquivo `hdf5` para cada uma das 133 orientações.

2 Resolução do Trabalho

Nesta seção é apresentado como o problema de prever o tamanho das partículas de gelo com base nas leituras de intensidade foi resolvido. O fluxograma exibido na Figura 1 explica, em alto nível, a execução do programa. As subseções a seguir explicam em detalhes como o problema foi atacado.

2.1 Normalização dos dados

A distribuição da intensidade das partículas é assimétrica, conforme pode ser visto na Figura 2a. Para normalizar a entrada do problema, a função escolhida foi o logarítmo de base natural, apontado por [9] como a de melhor resultado. O resultado da normalização é a transformação dos dados em uma distribuição próxima da normal, como mostrado na Figura 2b. O valor mínimo, máximo, média e desvio padrão antes e depois da normalização são listados na Tabela 1. Diferente de [10], a aplicação de *z-score* após a normalização logarítmica não apresentou melhoria nos experimentos conduzidos e consequentemente não foi utilizada.

	Mínimo	Máximo	Média	Desvio Padrão
Sem normalização	0	7580.1948	4.5862	29.2503
Com normalização logarítmica	-24.8460	8.9332	-1.1204	2.6368

Table 1: Características da distribuição antes e depois da normalização.

2.2 Invariânte a rotação

Assim como nos trabalhos [9, 10], neste relatório as intensidades de cada orientação foram tratadas para serem invariântes a rotação. Assim como em [9], polinômios de Zernike [1] de grau 20 foram calculados para

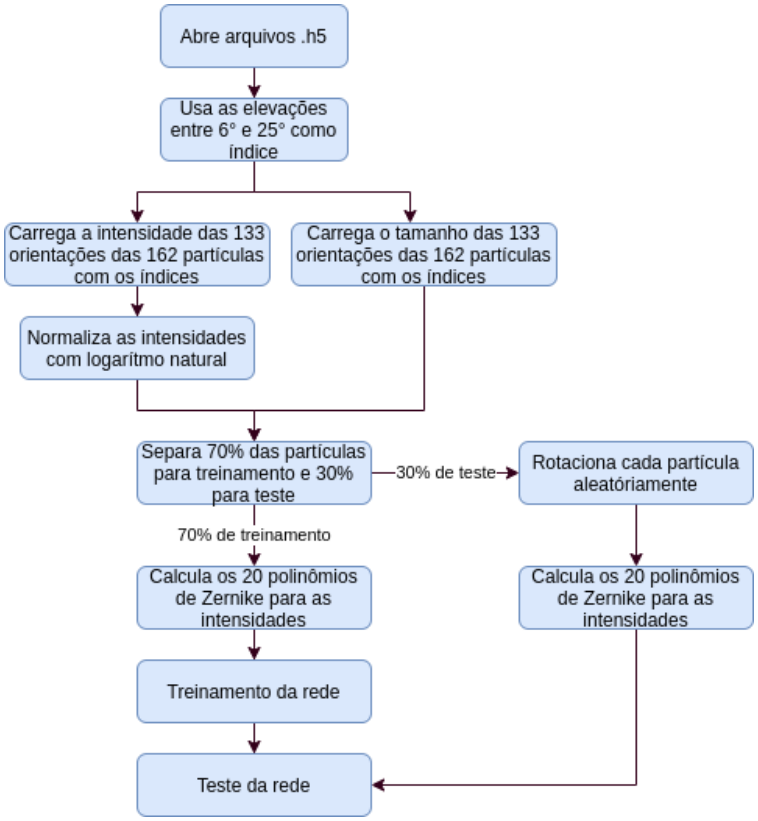


Figure 1: Fluxograma do algoritmo.

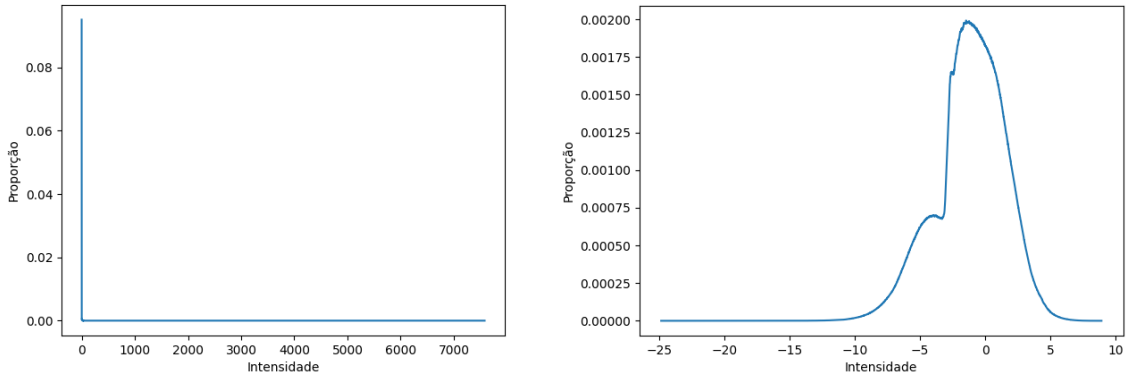
cada orientação de cada partícula de treinamento. Em relação ao conjunto de teste, antes de calcular os polinômios de Zernike, a matriz $20 * 361$ representando as leituras de intensidade de cada orientação de cada partícula foi rotacionada aleatoriamente.

2.3 Redução de dimensões

A utilização de grau 20 nos polinômios de Zernike resulta em um vetor de comprimento 121, reduzindo consideravelmente o vetor original de tamanho 7220 (ou matriz de 20 x 361). Ou seja, já existe uma redução de 7220 dimensões para 121, o que reduz consideravelmente o tempo necessário para treinar a rede neural. Para reduzir o número de dimensões ainda mais, [9] sugere a aplicação de análise de componentes principais (*Principal Component Analysis* (PCA)) [2], técnica capaz de encontrar dimensões não co-relacionadas através de transformações ortogonais e, consequentemente, reduzir o número de dimensões de um conjunto de dados. Em [9] PCA foi aplicado nas 121 dimensões retornadas pelo método de Zernike e foram selecionadas as 30 dimensões que possuem menor co-relação. Neste trabalho a utilização de PCA piorou o desempenho da rede, e consequentemente não foi utilizado.

2.4 Modelo Proposto

Apesar do enunciado do trabalho não especificar que o modelo de rede neural deva ser *feed forward*, optei por este modelo pois é o que foi discutido em maior profundidade na disciplina. Para encontrar a configuração que produz os melhores resultados, dezenas de configurações diferentes foram testadas. A biblioteca `scikit-learn` [5], utilizada para implementar a rede, fornece a classe `GridSearchCV` que recebe como parâmetro as possíveis configurações de uma rede neural e executa cada possível combinação de configuração, retornando o coeficiente de determinação (R^2) de cada configuração testada.



(a) Antes da normalização.

(b) Depois da normalização.

Figure 2: Normalização logarítmica dos dados de entrada geram uma distribuição próxima da normal.

Todos os parâmetros testados estão detalhados na Tabela 2. Os valores em negrito são os que apresentaram o maior coeficiente de determinação.

Parâmetro	**Possíveis Valores**
Função de ativação	Tangente Hiperbólica, **Unidade Linear Retificada (ReLU)**, Sigmóide
Solucionador	**Gradiente Descendente**, Adam, lbfgs
Taxa de aprendizado	**0.0001**, 0.001, 0.01
Tipo de taxa de aprendizado	Constante, **Adaptativo**
Camadas escondidas	(10), (10, 10), (10, 10, 10), (50), (50, 50), (50, 50, 50), (100), (100, 100), (100, 100, 100), (10, 50, 100), (100, 50, 10), (30, 60), (30, 60, 90), (30, 60, 90, 120), (60, 30), **(90, 60, 30)**, (120, 90, 60, 30)

Table 2: Todas as configurações da rede *feed forward* testados. Em negrito os parâmetros que produziram a rede com o maior coeficiente de determinação.

2.5 Detalhes de implementação

O trabalho foi implementado em Python 3, e as seguintes bibliotecas foram utilizadas:

- `h5py` [8] para carregar as medições e o tamanho correto das 162 partículas;
- `scikit-learn` [5] para criar a rede neural e treiná-la;
- `mahotas` [7] para calcular os polinômios de Zernike;
- `numpy` [6] para noramalizar os dados de entrada;
- `matplotlib` [3] para geração de gráficos.

Todas as bibliotecas podem ser instaladas através do gerenciador de pacotes padrão do Python, o `pip`, através do comando:

```
$ pip install h5py mahotas scikit-learn numpy matplotlib
```

O código fonte desta tarefa é publico[1] e possui a licensa MIT. Ele foi submetido junto com este relatório. Para reproduzir as imagens listas neste relatório e presentes na pasta resultados, basta executa o comando abaixo[2].

[1] Código fonte disponível em `https://github.com/fredericoschardong/FFNET-Two-Dimensional-Light-Scattering`.
[2] O script Python espera que os arquivos `h5` estejam no diretório `data`, no mesmo nível que o arquivo `main.py`.

```
$ python3 main.py
```

3 Resultados

Da mesma forma que os trabalhos [9, 10], neste relatório os resultados são mostrados em um gráfico onde o valor real e o valor previsto são confrontados. Na Figura 3 é possível verificar que os valores previstos pela rede neural (eixo X) se aproximam dos valores reais (eixo Y) para cada teste. O coeficiente de determinação é $R^2 = 0.966$, que é um valor alto considerando que as imagens testadas foram rotacionadas aleatoriamente. A linha laranja representa a solução ótima do problema.

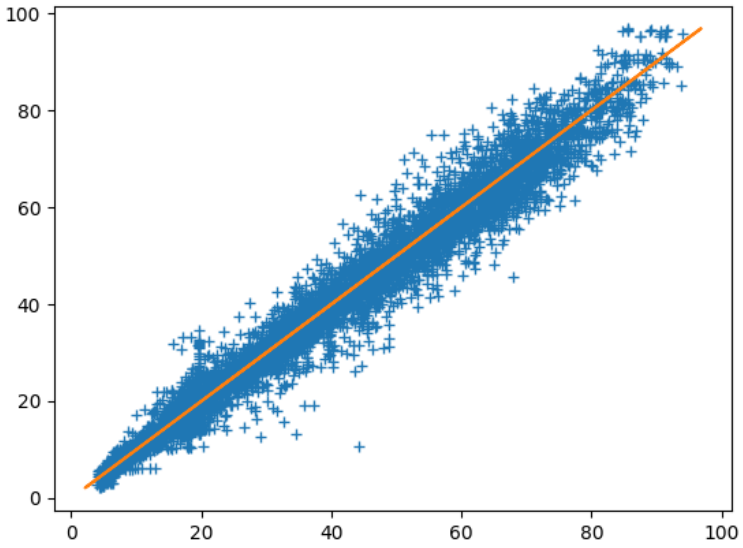


Figure 3: Resultado do teste da rede neural. Valores previstos no eixo X e valores reais no eixo Y.

4 Conclusão

É possível concluir que o objetivo proposto para este trabalho de prever o tamanho das partículas foi atingidos, bem como o desafio de prever o tamanho de partículas de forma invariânte a rotação. Por fim, o coeficiente de determinação encontrado, $R^2 = 0.966$, é muito próximo dos dois trabalhos encontrados na literatura $R^2 = 0.990$ [10] e $R^2 = 0.962$ [9].

References

[1] von F Zernike. "Beugungstheorie des schneidenver-fahrens und seiner verbesserten form, der phasenkontrastmethode". In: *physica* 1 (1934), pp. 689–704.

[2] Svante Wold, Kim Esbensen, and Paul Geladi. "Principal component analysis". In: *Chemometrics and intelligent laboratory systems* 2.1-3 (1987), pp. 37–52.

[3] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[4] C Stopford. "Ice crystal classification using two dimensional light scattering patterns". PhD thesis. 2010.

[5] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[6] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. “The NumPy array: a structure for efficient numerical computation”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[7] Luis Pedro Coelho. “Mahotas: Open source software for scriptable computer vision”. In: *arXiv preprint arXiv:1211.4907* (2012).

[8] Andrew Collette. *Python and HDF5: Unlocking Scientific Data.* ” O’Reilly Media, Inc.”, 2013.

[9] E O Salawu. “Development of Computational Models for Characterizing Small Particles Based on their Two-Dimensional Light Scattering Patterns”. MA thesis. UK: University of Hertfordshire, 2015.

[10] Vinıécius Couto Biermann et al. “Predição de Caracterıésticas de Partıéculas Atmosféricas Utilizando Redes Neurais Convolucionais”. Bachelor’s Thesis. 2019.