Vinícius Couto Biermann

Predição de Características de Partículas Atmosféricas Utilizando Redes Neurais Convolucionais

Florianópolis

2019

Vinícius Couto Biermann

Predição de Características de Partículas Atmosféricas Utilizando Redes Neurais Convolucionais

Monografia submetida ao Programa de Graduação em Ciência da Computação para a obtenção do Grau de Bacharel.

Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Ciência da Computação

Orientador: Prof. Dr. Mauro Roisenberg

Coorientador: Me. Daniel Priori

Florianópolis

2019

Biermann, Vinícius Couto

Predição de Características de Partículas Atmosféricas Utilizando Redes Neurais Convolucionais / Vinícius Couto Biermann; Orientador: Prof. Dr. Mauro Roisenberg; Coorientador: Me. Daniel Priori – 2019.

91 p. : il. (algumas color.) ; 30 cm.

Trabalho de Conclusão de Curso de Graduação – Universidade Federal de Santa Catarina, Departamento de Informática e Estatística, Ciência da Computação, Florianópolis, 2019.

Inclui referências

1. Ciência da Computação. 2. Inteligência Artificial. 3. Deep Learning. I. Roisenberg, Mauro. II. Priori, Daniel. III. Universidade Federal de Santa Catarina. IV. Título

Vinícius Couto Biermann

Predição de Características de Partículas Atmosféricas Utilizando Redes Neurais Convolucionais

Esta Monografia foi julgada aprovada para a obtenção do Título de "Bacharel em Ciência da Computação", e aprovada em sua forma final pelo Programa de Graduação em Ciência da Computação.

Florianópolis, 8 de dezembro de 2019:

Prof. José Francisco D. de G. C. Fletes
Coordenador do Curso

Banca Examinadora:

Prof. Dr. Mauro Roisenberg
Orientador

Me. Daniel Priori
Coorientador

Drª Giseli de Sousa
University of Hertfordshire, UK

Prof. Dr. Elder Rizzon Santos
Universidade Federal de Santa Catarina

Florianópolis
2019

Agradecimentos

Primeiramente, agradeço aos meus pais, meu irmão e o restante da minha família, pelo apoio, amor e ensinamentos.

Agradeço ao professor Mauro Roisenberg e ao Daniel Priori pela orientação e dedicação, assim como aos demais membros da banca avaliadora. Também aos demais professores do departamento, pelas disciplinas ministradas e todos os servidores e funcionários que mantém a universidade.

Agradeço aos meus amigos e colegas de curso, que sempre estiveram ao meu lado e compartilharam do árduo processo de graduação e a inestimável amizade desde que nos conhecemos. Também aos meus amigos de fora do curso, que mesmo morando em outros estados ou países, sempre me deram apoio e trouxeram momentos de alegria.

Agradeço aos demais membros do Laboratório de Conexionismo e Ciências Cognitivas (L3C) do INE-UFSC, pela ajuda durante o desenvolvimento desse trabalho.

Agradeço aos professores(as) e colegas do Instituto Cultural Niten e da Associação Nipo-Catarinense, por tudo que me ensinam, pela compreensão e apoio, também a todos que participaram do meu processo de aprendizado até aqui.

Por fim, agradeço a todos que contribuíram para o avanço da humanidade e àqueles que permitiram meus estudos nesta universidade pública.

Resumo

A concentração de partículas prismáticas na atmosfera possui influências significativas em nosso planeta. Modelos climáticos são fortemente afetados por características de partículas como os cristais de gelo, presentes em nuvens, que alteram taxas de reflexão e absorção de radiação, alterando o clima terrestre. Uma forma de coletar amostras para o estudo de tais partículas é a utilização de sondas, que registram imagens de padrões bidimensionais de dispersão de luz de uma partícula capturada. Dessas imagens, podemos extrair informações úteis, como tamanho e formato. Para extrair estas informações de tamanho e formato, este relatório foi elaborado com o objetivo de implementar um ambiente de arquitetura *Deep Learning* com redes neurais convolucionais para um problema de regressão em um conjunto de pequenas partículas atmosféricas com a intenção de prever o tamanho projetado de tais partículas. O principal aspecto da implementação, foi deixar o sistema invariante à possíveis rotações existentes nos dados das partículas, para tanto, uma abordagem de tratamento de imagens com o uso de transformadas rápidas de Fourier e sua inversa foi utilizada. Os resultados obtidos do modelo desenvolvido foram satisfatórios, com um coeficiente de determinação $R^2 = 0.9901$ no melhor dos experimentos realizados. Também foi feita uma comparação com os resultados já obtidos em um trabalho anterior que usou o mesmo conjunto de dados sobre diferentes técnicas de aprendizado de máquina.

Palavras-chaves: Inteligência Artificial. Deep Learning. Redes Neurais Convolucionais. Partículas Atmosféricas. Padrões Bidimensionais de Dispersão de Luz.

Abstract

The concentration of prismatic particles on the atmosphere has significant influences on our planet. Climate models are strongly affected by characteristics of atmospheric particles such as ice crystals, present in clouds, which can modify the reflection and absorption rates of radiation, thus changing Earth's climate. One way of collecting samples for the study of such particles is the use of probes, which record the two-dimensional light scattering patterns of a captured particle. From those images, one can extract useful information, such as size and shape of particles. To extract those informations, the objective of this report is to implement a Deep Learning architecture environment with convolutional neural networks for a regression problem on a set of small atmospheric particles with the intention of predicting the projected size of these particles. The main aspect of the implementation is to make the system invariant to possible rotations which can exist in the particle data; thus, an approach of image processing using fast Fourier transforms and its inverse was used. The results obtained from the developed model were satisfactory, with a coefficient of determination $R^2 = 0.9901$ on the best of the experiments performed. These results were also compared with results obtained from a previous work that used the same dataset on different machine learning techniques.

Key-words: Artificial Intelligence. Deep Learning. Convolutional Neural Networks. Atmospheric Particles. Two-Dimensional Light Scattering Patterns.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

2DLS	*2-Dimensional Light Scattering* - Dispersão bidimensional de luz
SID	*Small Ice Detector*
ZM	*Zernike Moments* - Momentos de Zernike
SVM	*Support Vector Machine* - Máquina de vetores de suporte
SVR	*Support Vector Regression* - Regressão por vetores de suporte
SVC	*Support Vector Classifier* - Classificador por vetores de suporte
RFC	*Random Forests Classification* - Classificação por floresta aleatória
FFT	*Fast Fourier Transform* - Transformada rápida de Fourier
IFFT	*Inverse Fast Fourier Transform* - Transformada Inversa Rápida de Fourier
DNN	*Deep Neural Network* - Rede neural profunda
RNN	*Recurrent Neural Networks* - Redes neurais recorrentes
CNN	*Convolutional Neural Network* - Redes neurais convolucionais
ReLU	*Rectified Linear Unit* - Unidade linear retificada
GPU	*Graphic Processing Unit* - Unidade de processamento gráfico

Sumário

1 Introdução

A história das redes neurais começa com o trabalho de Mcculloch e Pitts (1943), como o primeiro modelo de representação de um esquema neural biológico em computadores. Ao longo de décadas foram feitos vários avanços para possibilitar a computação com esses modelos, como a criação do *Perceptron* de Rosenblatt (1958), o algoritmo de *Backpropagation* de Rumelhart, Hinton e Williams (1986) e outros. Um dos modelos de redes neurais artificiais é conhecido como redes neurais convolucionais, que evoluiu de um conjunto de ideias para representar a organização de um córtex visual de um animal e de outros modelos como o *Neocognitron*, de Fukushima (1980).

Uma rede neural é composta por unidades de processamento simples denominadas neurônios. Estas unidades estão distribuídas em diferentes camadas, que se interconectam através de saídas e entradas dos neurônios que as formam. Temos uma rede de camadas profundas, ou *Deep Learning*, quando o número de camadas aumenta, tendo como entrada de uma camada a saída da anterior, com o uso de operações não lineares no processamento de informações extraídas e transformadas. Durante muito tempo existiu uma barreira que impossibilitava o seu uso, pois quanto maior for o conjunto de dados e número de camadas, mais cresce a necessidade de processamento. Esse uso se tornou possível com os avanços no *hardware*, com processadores mais eficientes e, atualmente, com o uso do poder de processamento intenso de GPUs (*Graphics Processing Units*).

Diferentes classes de problemas, assim como o tipo de seus dados, exigem redes específicas para alcançar resultados satisfatórios. Assim, surgiram várias arquiteturas em *Deep Learning* que se especializam em solucionar problemas de uma determinada classe — *i.e.* cada tipo de problema possui uma arquitetura de rede mais adequada para sua resolução.

Redes neurais convolucionais tem sido usadas em aplicações desde jogos como Go, com a rede AlphaGo [1], até processamento de linguagens naturais e principalmente no reconhecimento de padrões em imagens, utilizando, por exemplo, bancos de dados como o ImageNet [2], que hoje conta com mais de 14 milhões de imagens diversas. Por terem o propósito de reconhecimento de padrões, tais redes foram utilizadas na resolução do problema proposto e, portanto, receberam um nível maior de detalhes nesta monografia.

O problema tratado neste relatório foi o de regressão de imagens de partículas atmosféricas com diâmetros pequenos, na ordem de alguns micrômetros (μm). Estas imagens são padrões bidimensionais de dispersão de luz (*2DLS* [3] *patterns*, em inglês). Espera-se

[1] <https://deepmind.com/research/alphago/>
[2] <http://www.image-net.org/>
[3] *Two-Dimensional Light Scattering*

que as redes neurais convolucionais se mostrem adequadas para a resolução do problema, obtendo, no mínimo, resultados semelhantes aos já registrados com o uso de outras técnicas de aprendizagem que foram aplicadas ao mesmo conjunto de dados na dissertação original do problema descrita por Salawu (2015) e no trabalho continuado por Priori (2017).

1.1 Objetivos

A proposta deste relatório é a pesquisa de *Deep Learning*, com uma introdução de seus fundamentos e uma breve apresentação aos principais métodos utilizados nessa ciência. Será então dado um enfoque maior na técnica de redes neurais convolucionais e a utilização dessas técnicas na resolução do problema de predição de partículas atmosféricas.

Os objetivos são, portanto, divididos em:

- Objetivo Geral
  - Utilizar redes neurais convolucionais inseridas em uma arquitetura de redes neurais de camada profunda para prever características relevantes, tais como tamanho projetado de partículas prismáticas cristalinas, a partir de imagens de seus padrões de difração sob a incidência de feixes de luz. O sistema de processamento de dados e treinamento da rede neural desenvolvida deve ser invariante à rotações das imagens provenientes das intensidades de difração.
- Objetivos Específicos
  - Estudar os conceitos básicos de *Deep Learning*.
  - Estudar redes convolucionais.
  - Estudar as imagens dos padrões de difração das partículas e que características poderiam ser extraídas delas.
  - Estudar o método de Transformadas Rápidas de Fourier (FFT) para garantir invariância à rotação das imagens no sistema desenvolvido.
  - Montar um modelo de redes neurais convolucionais.
  - Realizar experimentos e obter resultados.
  - Analisar os resultados obtidos.

1.2 Método de Pesquisa

A primeira fase desse trabalho englobou a pesquisa e estudo do estado da arte em *Deep Learning*, principalmente em redes neurais convolucionais, e trabalhos anteriores que utilizam o mesmo conjunto de dados. Para tal, foram utilizados livros recentes, como os de Goodfellow, Bengio e Courville (2016) e Nielsen (2015), disponibilizados *online* de maneira pública, assim como artigos e teses relacionados ao tema, principalmente os de Salawu (2015), Priori (2017) e Salawu et al. (2017).

Na fase posterior, o conhecimento teórico serviu de base para o desenvolvimento de um código em `MATLAB`®, onde uma rede convolucional teve seus parâmetros ajustados e foi treinada para prever o tamanho projetado das partículas através de imagens de seus padrões de difração de luz. Os dados de intensidade foram pré-processados para garantir à rede invariância à rotação. Resultados foram então comparados com os já obtidos em trabalhos anteriores para verificar a viabilidade do uso de redes convolucionais no problema proposto.

1.3 Organização do Texto

Esta monografia está estruturada nas seguintes partes:

- Introdução
- Revisão Bibliográfica
- Apresentação da Proposta
- Experimentos e Resultados
- Conclusões

2 Revisão Bibliográfica

Esse capítulo traz uma contextualização sobre partículas atmosféricas, explicando brevemente sua origem e influências, assim como experimentos atuais. O local de origem de partículas cristalinas onde o conjunto de dados utilizado se baseia. A seguir, é apresentado o *Small Ice Detector*, um tipo de sonda capaz de capturar e armazenar dados de dispersão de luz de pequenas partículas. Uma seção de trabalhos correlatos expõe pesquisas anteriores que utilizaram o mesmo conjunto de dados e buscaram, com diferentes técnicas de aprendizado de máquina, prever o tamanho de partículas.

Uma grande parte desse relatório é colocada na seção de fundamentos teóricos, onde são enunciados conhecimentos base de redes neurais, como *perceptron*, diferentes funções de ativação e *backpropagation*. O tema de *Deep Learning* é introduzido e um enfoque mais profundo é dado em redes neurais convolucionais, demonstrando cada elemento essencial para entender seu funcionamento, pois esse tipo de arquitetura será empregue como uma abordagem diferente de resolver o mesmo problema visto nos trabalhos correlatos.

2.1 Partículas Atmosféricas

O aerossol é definido como um conjunto de pequenas partículas sólidas ou líquidas que estão suspensas em um gás. Quando queremos nos referir à partícula em si, usamos o termo partícula atmosférica (particulado ou material particulado). Tais partículas se originam de produção humana ou natural, podendo ser orgânicas ou inorgânicas. Alguns exemplos são: pólen; esporos; bactérias (FRÖHLICH-NOWOISKY et al., 2016); poeira vulcânica; produtos de combustão (LANGLEY RESEARCH CENTER, 1996); etc.

Por possuírem tamanhos microscópicos, pode-se imaginar que o ar esteja repleto dessas partículas e que a presença de uma grande quantidade delas pode afetar de alguma maneira a vida no planeta. De fato, a existência de aerossóis na atmosfera tem significativas influências na Terra.

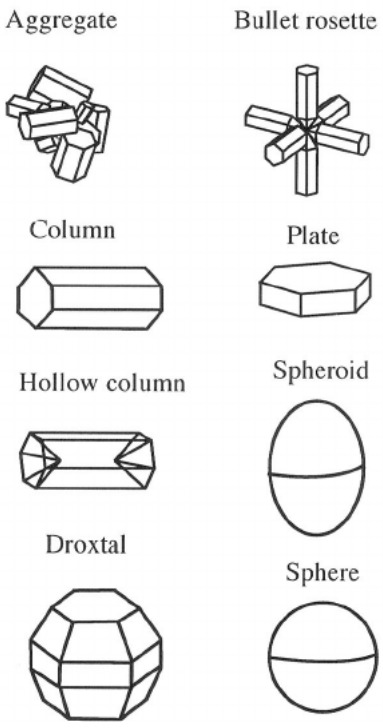
Diversos projetos de pesquisa tem como objetivo uma melhor compreensão de aerossóis em nuvens e os impactos climáticos causados por ambos. Um desses projetos é o experimento CLOUD (Cosmics Leaving Outdoor Droplets)[1], da CERN [2], que estuda as relações entre aerossóis e raios cósmicos. Com os novos dados obtidos, o entendimento sobre a formação de novas partículas na troposfera permite uma melhor precisão na projeção do aumento de temperatura na Terra (CHALMERS; JARLETT, 2016).

[1] Site oficial do experimento CLOUD: <https://home.cern/about/experiments/cloud>
[2] *Organisation Européenne pour la Recherche Nucléaire*

Outras partículas que influenciam no clima são os cristais de gelo presentes principalmente na composição de nuvens do tipo *cirrus*. Essas nuvens se formam na troposfera, em altitudes entre 5 km e 14 km, cobrindo cerca de 30% da superfície do planeta e se concentram, em maior parte, na região dos trópicos. As *cirrus* fazem parte do balanço de radiação da Terra, onde podem aumentar ou diminuir a temperatura abaixo delas através de processos físicos de reflexão e absorção de radiações solares e da superfície do planeta, agindo como um efeito estufa natural (BARAN, 2009).

Os parâmetros para as taxas de reflexão e absorção de radiação estão ligados às características dos cristais de gelo, como tamanho e formato, e a concentração de massa de cristais nas nuvens. A Figura 1 mostra alguns dos formatos conhecidos dos cristais. Para a extração de informações dessas partículas, foram criadas sondas passíveis de serem acopladas em asas de aviões que então podem coletar e analisar os cristais diretamente no seu local de origem.

Figura 1 – Alguns formatos de cristais de gelo.



Em linha: agregado, bullet rosette, coluna, platô, coluna oca, esferoide, droxtal e esfera.

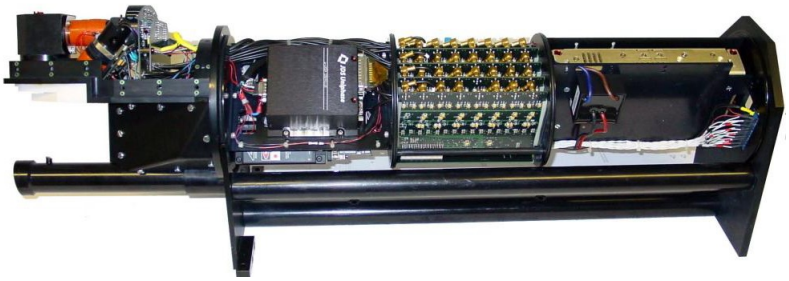
Fonte: Yang et al. (2005)

2.1.1 A sonda SID

O *Small Ice Detector* (SID) é uma série de instrumentos desenvolvidos pela universidade de *Hertfordshire*, no Reino Unido, com o objetivo de coletar dados bidimensionais de dispersão de luz (*Two-Dimensional Light Scattering*, ou 2DSL) de partículas.

Para a captura de dados de uma partícula, um feixe de luz é emitido entre os dois "braços"da sonda (lado esquerdo da Figura 2) que apontam para a frente enquanto a aeronave a qual foi acoplada passa por uma nuvem para coletas. Quando uma partícula entra em contato com o *laser*, dois sensores detectam um pico na dispersão de luz causada e ativam o detector principal para a captura dos dados (STOPFORD, 2010).

Figura 2 – Sonda SID-2 com parte eletrônica exposta.



Fonte: Adaptado de Stopford (2010)

A sonda SID-2 consegue observar partículas de 5 a 100μm operando em uma frequência de 10KHz, sendo possível coletar amostras de aproximadamente 9000 partículas por segundo (STOPFORD, 2010). A principal diferença entre a sonda original SID para a SID-2, é que SID possui apenas seis unidades de detectores para analisar uma partícula, o que resultava em uma avaliação bruta dos tamanhos e formatos coletados, enquanto o número de detectores em SID-2 foi elevado para trinta e dois, arranjados radialmente em torno do eixo do canhão de *laser* [3].

2.2 Trabalhos Correlatos

A dissertação de mestrado de Salawu (2015) é o trabalho original onde o conjunto de dados dos padrões 2DLS foram utilizados juntamente com algumas técnicas de aprendizado de máquina para tentar estimar características das partículas, como tamanho e *aspect ratio*.

Foram explorados métodos de aprendizado de máquina de regressão e classificação, assim como alguns métodos de *scaling* e normalização, como *min-max scaling, z-score scaling e log-scaling*. Como os modelos utilizados eram sensíveis à rotações, foi necessário também o uso de *Zernike Moments* (ZM) para torná-los invariantes à rotação.

Os melhores resultados foram obtidos com o uso de *Support Vector Machines* (SVM) em regressão para tamanho projetado de partículas, com *Support Vector Regression* (SVR) e classificação para classes de *aspect ratio*, com *Support Vector Classifier*

[3] <https://www.herts.ac.uk/research/centres/cacp/particle-instruments-and-diagnostics/sid-probes>

(SVC). Foi utilizado *natural-log scaling*, atingindo uma acurácia de 99%. Esse valor máximo de acurácia foi possível apenas em rotação de 20°, já que em outros níveis de rotação o desempenho degradava.

Outro trabalho que explorou o mesmo conjunto de dados foi a dissertação de Priori (2017), dando continuação ao trabalho anterior, que busca o valor previsto de tamanho projetado das partículas. Foram empregadas quatro diferentes modelos de aprendizado de máquina: dois utilizando *Feed Forward Multi-Layer Perceptron* com regularização Bayesiana, um terceiro com Função de Base Radial, e o quarto um arquitetura *Deep Learning* do tipo *Autoencoders*.

Assim como no trabalho de Salawu, diversas técnicas de *scaling* e normalização foram testadas e, de acordo com Priori, os melhores desempenhos no processo de teste foram obtidos com a aplicação de ambos *natural-log* e *z-score* em sequência.

O modelo de *autoencoders* obteve os melhores resultados, especialmente quanto à predição com as menores partículas, onde os outros modelos não foram tão acurados, atingindo uma performance (R^2) de 99%. Enquanto o problema com a rotação de partículas foi abordado com a utilização de ZM por Salawu, Priori utilizou Transformadas Rápidas de Fourier (FFT, do inglês *Fast Fourier Transform*), mantendo um resultado entre 98% e 99% em qualquer nível de rotação.

No artigo publicado por Salawu et al. (2017), tem-se uma continuação do trabalho de mestrado de Salawu. O novo sistema está dividido em três partes. Na primeira parte, *Zernike Moments* foram novamente utilizados para garantir invariância à rotação no sistema. Após, foi aplicado a técnica de *Random forests classification* (RFC) para prever a orientação de cada padrão 2DLS. O terceiro estágio contém implementações específicas de SVMs para cada uma das 133 orientações. Os modelos de SVM desenvolvidos foram utilizados para diferentes características como descrito a seguir:

- 133 SVRs para prever o tamanho de prismas hexagonais;
- 133 SVRs para prever o tamanho projetado de prismas hexagonais, descrito como o diâmetro médio de área projetada;
- 133 SVCs para prever o *aspect ratio* de prismas hexagonais.

Assim, com o resultado obtido na etapa de RFC, é determinado o preditor e classificador apropriado para o padrão 2DLS fornecido, e então ele é repassado ao estágio 3, onde os modelos treinados irão analisar o parão e dar como resultado os valores previstos das características. Os resultados atingiram acurácia de entre 98% e 99% para qualquer nível de rotação.

A Tabela 1 mostra uma comparação dos trabalhos correlatos focados na predição de tamanhos projetados das partículas.

Tabela 1 – Trabalhos correlatos

Autor	Dataset	Técnicas utilizadas	Resultado
Salawu (2015)	Partículas atmosféricas prismáticas	SVR com ZM	99%
Priori	Partículas atmosféricas prismáticas	Autoencoders com FFT	99%
Salawu (2017)	Partículas atmosféricas prismáticas	ZM, RFC, SVR e SVC	99%

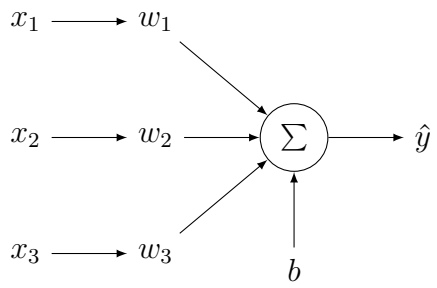
Fonte: Elaborada pelo autor.

2.3 Fundamentos Teóricos

2.3.1 *Perceptron*

Após o trabalho de Mcculloch e Pitts (1943), Frank Rosenblatt (1958) criou um modelo de neurônio artificial chamado de *perceptron*. Seu algoritmo é um classificador linear binário em sistemas com aprendizado supervisionado. Abaixo temos uma visão de um *perceptron* simples mostrado como um grafo.

Figura 3 – *Perceptron*



Fonte: Elaborada pelo autor.

Dado um conjunto de valores binários de entrada, denominado por $x_1, x_2, ..., x_n$, o *perceptron* produz uma saída binária $\hat{y}$. Outras duas entradas também utilizadas são os seguintes dois parâmetros: o conjunto de *weights* (pesos), $w_1, w_2, ..., w_n$ e um número real b, denominado *bias*. O valor de saída do algoritmo se dá por:

$$\hat{y} = \begin{cases} 1, & \text{se } W \cdot X + b > 0 \\ 0, & \text{se } W \cdot X + b \leq 0 \end{cases} \tag{2.1}$$

onde W e X são matrizes que contém os valores de w e x respectivamente e

$$W \cdot X = \sum_{i=1}^{n} w_i x_i \tag{2.2}$$

O valor de *bias* representa a inclinação de um neurônio ser ativado, ou seja, quanto maior for o seu valor, maior é a chance da saída de seu *perceptron* ser 1.

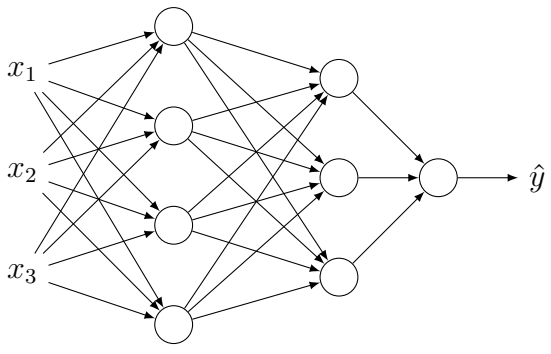
2.3.2 *Multilayer Perceptrons*

Uma rede com apenas uma camada só pode representar funções linearmente separáveis, portanto para que problemas mais complexos possam ser resolvidos, são utilizadas redes com múltiplas camadas. As camadas (*layers*) de uma rede *multilayer* são divididas nos seguintes grupos:

- *Input Layer* – Contém os dados de entrada da rede;
- *Hidden Layers* – Conjunto de camadas intermediárias;
- *Output Layer* – Camada com uma unidade que gera o valor da saída.

Para a contagem do número de camadas em uma rede, não é considerada a *input layer*. A Figura 4 mostra uma representação de uma rede com três camadas, sendo duas pertencentes ao conjunto das *hidden layers*, com 4 e 3 unidades em cada, e uma última de uma unidade pertencente à *output layer*. Para referências à camadas usamos $[n]$, onde n é o número da camada.

Figura 4 – Multilayer Perceptron



Fonte: Elaborada pelo autor.

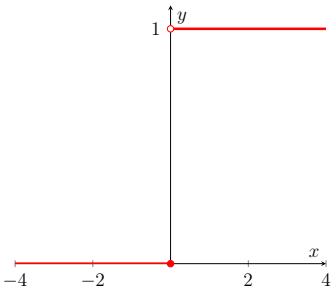
2.3.3 Funções de Ativação

Seja $z^{[n]} = W^{[n]} \cdot a^{[n-1]} + b^{[n]}$ um vetor com os valores de saída dos neurônios de uma camada n qualquer, uma função diferenciável f, geralmente não-linear, aplicada em

z é chamada de função de ativação ou de transferência. Portanto, uma ativação a em uma camada n é dada por $a^{[n]} = f(z^{[n]})$. No caso onde são calculados os valores na primeira camada, temos que $a^{[0]}$ é igual ao conjunto de entrada X.

O *Perceptron* original utilizava uma função degrau (*Step Function*) — *i.e.* os únicos dois valores possíveis são 0 e 1, como pode ser visto na equação 2.1 e na Figura 5. Por ser uma função com derivada sempre 0 para $x \neq 0$, pequenas mudanças em *weights* e *biases* podem não gerar nenhuma mudança na saída, ou mudá-la completamente, não permitindo um aprendizado mais controlado (NIELSEN, 2015).

Figura 5 – Função Degrau



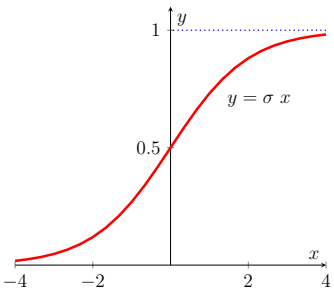
Fonte: Elaborada pelo autor.

Abaixo temos algumas das funções mais utilizadas atualmente.

- Função Sigmóide ou Logística: Seu formato é uma versão suave da função degrau, aceitando quaisquer valores entre 0 e 1 (NIELSEN, 2015).

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

Figura 6 – Sigmóide



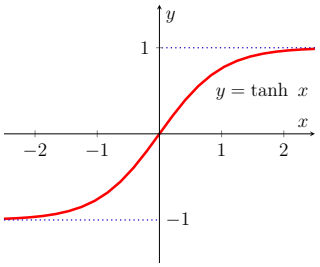
Fonte: Elaborada pelo autor.

- Função Tangente Hiperbólica: Muito similar à função sigmóide, pode ser definida como $tanh(z) = 2\sigma(2z) - 1$. Geralmente oferece um melhor desempenho durante

o treinamento da rede que a função sigmóide por ser mais próxima da identidade (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.4}$$

Figura 7 – Tangente Hiperbólica

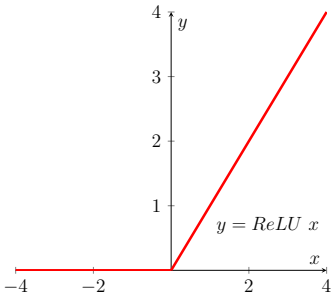


Fonte: Elaborada pelo autor.

- Rectified Linear Unit - ReLU: Função muito utilizada por ser formada com duas partes lineares, o que preserva propriedades de modelos lineares que são de fácil otimização com métodos baseados em gradiente (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$ReLU(z) = max(0, z) \tag{2.5}$$

Figura 8 – *Rectified Linear Unit*



Fonte: Elaborada pelo autor.

2.3.4 *Backpropagation*

Backpropagation é um procedimento criado para diminuir a diferença entre a saída da rede e o valor real desejado. A ideia é calcular a função de perda (ou custo) e então retro-propagar [4] derivações parciais na rede para atualizar valores dos parâmetros *weight*

[4] *i.e.* enviar valores da direita para a esquerda entre os nodos da rede

e *bias* de unidades presentes nas *hidden layers* (RUMELHART; HINTON; WILLIAMS, 1986).

Como a saída da função de ativação de uma unidade em uma camada l é utilizada por todas as unidades da camada $l+1$, temos que uma variação $\Delta a_n^{[l]}$ no n-ésimo neurônio da camada l será propagada no cálculo de todas as unidades das próximas camadas, finalmente resultando em um valor diferente na saída final da rede.

É com o uso desse algoritmo que uma rede neural consegue alterar seus parâmetros para que esses valores aplicados no processo de aprendizagem consigam gerar uma saída mais próxima da desejada (fornecida para o treinamento), alcançando melhores resultados finais. Assim, o desenvolvedor da rede não precisa se preocupar com o cálculo de difíceis parâmetros, como o *kernel*, que é apresentado nas seções seguintes.

2.4 *Deep Learning*

Quando começamos a adicionar *hidden layers* em um modelo, ele é classificado como uma *Deep Neural Network* (DNN), onde o uso do termo *deep* se dá ao fato de que quando acrescentamos uma nova camada à rede, dizemos que ela fica mais profunda. Existem muitas nomenclaturas para definir modelos que possuem múltiplas camadas, mas a mais utilizada atualmente é *Deep Learning*. Na literatura, não existe um acordo sobre quantas camadas uma DNN precisa ter, as definições geralmente se preocupam em explicitar abstração e complexidade, mas no estado da arte, é comum encontrar redes com dezenas de camadas.

Deep Learning permite modelos computacionais compostos por múltiplas camadas de processamento aprenderem representações de informações em múltiplos níveis de abstração. Podemos dizer que as camadas iniciais de uma rede aprendem funções menos complexas, enquanto as camadas posteriores, com o uso das transformações da informação obtidas das camadas iniciais, conseguem aprender funções mais complexas (LECUN; BENGIO; HINTON, 2015).

Tomando uma imagem de um rosto humano como exemplo, as primeiras camadas podem detectar bordas, que são padrões simples. As camadas intermediárias podem agrupar as informações dessas bordas para formar partes de um rosto (*e.g.* olhos, lábios). As últimas camadas então, se encarregam de agrupar essas informações de partes para reconhecer uma face completa (NG, 2017).

2.4.1 Diferentes Arquiteturas

Como dito anteriormente, *Deep Learning* abrange diferentes arquiteturas que são apropriadas para resolução de problemas distintos. Esta seção apresenta brevemente outro

tipo de arquitetura que, assim como a convolucional, teve um grande nível de desenvolvimento e é amplamente usada, as *Recurrent Neural Networks* (RNNs).

RNNs são redes que atuam muito bem quando temos que processar dados sequenciais (*e.g.* textos, falas) como entrada. RNNs tratam o conjunto de entrada como uma sequência de informações no tempo t, onde uma entrada $X^{[t]}$ pode influenciar o processamento de uma entrada $X^{[t+n]}$, com isso, em uma representação em grafo da rede, poderíamos inserir um ciclo em uma camada. Uma propriedade dessas redes é o compartilhamento de parâmetros, o que permite aplicar o mesmo modelo em entradas de tamanhos variados, ou produzir uma saída com tamanho diferente da entrada (GOODFELLOW; BENGIO; COURVILLE, 2016).

Com isso, essas redes são utilizadas para o processamento de textos, quando queremos, por exemplo, gerar uma tradução. Para isso, duas redes podem ser utilizadas, uma sobre o texto original, chamada de *encoder* e outra sobre a saída da primeira, chamada de *decoder*, que produzirá o texto traduzido em outra linguagem. Um exemplo dessa aplicação pode ser encontrado no sistema *Google Neural Machine Translation*, onde duas redes com 8 camadas cada foram utilizadas (WU et al., 2016).

2.4.2 Visão Computacional

O campo de visão computacional visa dar habilidades visuais aos computadores, podendo abstrair informações de uma imagem ou uma sequência de imagens. Com as arquiteturas de *Deep Learning*, vários avanços foram feitos, principalmente quando se quer reconhecer ou detectar objetos e padrões.

O YOLO [5](*You Only Look Once*) é um sistema de detecção e objetos construído com 53 camadas convolucionais e capaz de processar imagens em 30 *frames* por segundo com uma média de precisão de 57.9% (REDMON; FARHADI, 2018).

Outro exemplo recente do uso de visão computacional é novamente o Google Translate™, onde um ambiente une o uso de redes convolucionais, que extraem de uma imagem a informação das letras detectadas e as dá como entrada para uma RNN, efetivamente traduzindo o texto de uma imagem [6].

2.5 Redes Neurais Convolucionais

As redes neurais convolucionais, ou CNNs (do inglês *Convolutional Neural Networks*), representam uma das mais importantes arquiteturas presentes nas redes *Deep Learning*. De influência dos resultados de estudos neurocientíficos do cérebro e sistema visual no sé-

[5] Disponível em: <https://pjreddie.com/darknet/yolo/>
[6] <https://ai.googleblog.com/search/label/Google%20Translate>

culo XX, surgiram alguns dos princípios básicos utilizados nessas redes, como a resposta de neurônios a intensidades de luz ou padrões bem definidos.

Os conceitos que caracterizam uma CNN são explicados nas subseções a seguir, começando pela definição matemática de convolução, a operação que dá o nome a essas redes, e se estendendo em outros processos que auxiliam e corrigem problemas de uma convolução pura, dando mais ferramentas para o aumento da qualidade dos resultados obtidos em um treinamento.

2.5.1 Convoluções

A convolução é uma operação matemática que define uma regra de produção de uma função a partir de outras duas funções.

Uma convolução é definida como uma transformada integral:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(x-u)g(u)du \tag{2.6}$$

Aqui, o símbolo $*$ é o operador de convolução, não uma multiplicação usual. As funções f e g operam sobre o mesmo domínio de entradas. Uma função gerada pela convolução depende das outras duas em todo o intervalo $-\infty < x < \infty$ para calcular seu valor em um ponto x qualquer.

Na arte de processamento de sinais (*e.g.* som e imagem), uma operação denominada *cross-correlation* pode ser usada. Tal operação é similar à convolução, como veremos a seguir. Para funções contínuas, é definida como:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(u)g(u+x)du \tag{2.7}$$

Para funções discretas, é definida por:

$$(f * g)(x) = \sum_{u=-\infty}^{\infty} f(u)g(u+x) \tag{2.8}$$

Como queremos trabalhar com imagens, f e g são matrizes que representam, respectivamente, a *input* (imagem de entrada para a função) e o *kernel* (também comumente chamado de "filtro" ou "máscara"), utilizado para detectar padrões na imagem.

A simples diferença entre o uso de convolução e *cross-correlation*, é que na primeira o *kernel* é rotacionado 180° para realizar as operações, enquanto ele se mantém em operações de *cross-correlation*. A Figura 9 mostra a rotação de uma matriz.

Quando tratamos de redes neurais, o resultado não mudará se o *kernel* estiver ou não rotacionado, pois o algoritmo utilizado aprenderá de maneira igual. Como as duas formas podem gerar o mesmo resultado, o *cross-correlation* é geralmente o escolhido para ser implementado por simplificar o código escrito para uma operação. Existe também uma

Figura 9 – Rotação de uma matriz em 180°.

$$\begin{bmatrix} 2 & 5 & 1 \\ 9 & 6 & 4 \\ 8 & 7 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 7 & 8 \\ 4 & 6 & 9 \\ 1 & 5 & 2 \end{bmatrix}$$

Fonte: Elaborada pelo autor.

convenção da literatura, feita para facilitar o uso de termos, onde a operação feita por uma camada convolucional é chamada de convolução, mesmo que ela, efetivamente, seja de *cross-correlation* (NG, 2017).

O resultado de uma operação entre *input* e um *kernel* é denominado *feature map* e, para diferenciar do símbolo $*$ utilizado anteriormente na definição de convolução, o símbolo $\otimes$ será usado. Assim, a fórmula que representa uma operação de convolução em duas dimensões pode ser dada por:

$$featureMap(x, y) = (In \otimes Ker)(x, y) = \sum_{\alpha=0}^{lin-1} \left(\sum_{\beta=0}^{col-1} In(x + \alpha, y + \beta) \cdot Ker(\alpha, \beta) \right) \quad (2.9)$$

onde In é a *input*, Ker o *kernel*, lin e col são, respectivamente, número de colunas e linhas do *kernel*, que devido ao fato do *kernel* ser uma matriz quadrada, terão um mesmo valor t, representando o tamanho da matriz (GOODFELLOW; BENGIO; COURVILLE, 2016).

A operação pode ser vista simplesmente como um produto interno usual entre o *kernel* e uma região com as mesmas dimensões do *kernel* na *input*. É feita uma varredura na matriz input para gerar cada elemento do *feature map*. Por exemplo, dado um *kernel* de dimensões 3×3, o elemento na posição $(1, 2)$ de *feature map* é dado por:

$$featureMap(1, 2) = \sum_{\alpha=0}^{2} \left(\sum_{\beta=0}^{2} In(1 + \alpha, 2 + \beta) \cdot Ker(\alpha, \beta) \right) \quad (2.10)$$

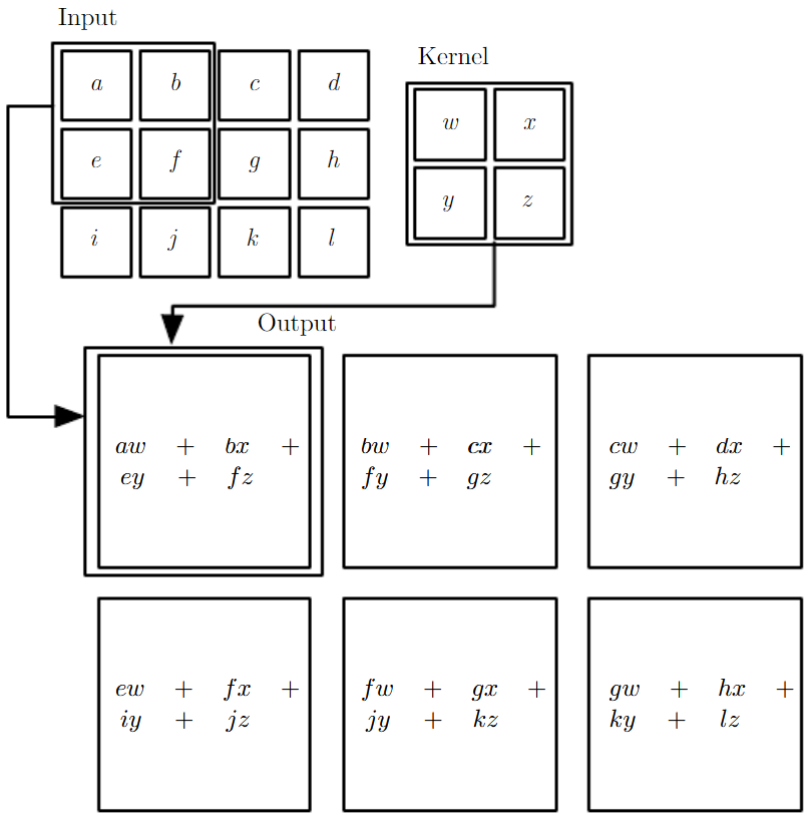
Um exemplo de formação completa pode ser visto na Figura 10, onde uma *input* dada por uma matriz $I(3 \times 4)$ e um *kernel* $K(2 \times 2)$ geram um *feature map* de dimensão 2×3.

Dado que uma *input* tenha dimensão $n \times m$ e um *kernel* de dimensão $f \times f$ seja usado, a dimensão de um *feature map* pode ser calculada da seguinte maneira:

$$(n - f + 1) \times (m - f + 1) \quad (2.11)$$

Veremos a seguir que essa fórmula ainda está muito ingênua e não se aplica realmente na prática, portanto será modificada para incluir outras variáveis importantes, *padding* e *stride*.

Figura 10 – Exemplo de formação de um *feature map*.



Fonte: Goodfellow, Bengio e Courville (2016)

É relevante notar que as fórmulas apresentadas aqui pertencem à um contexto onde estamos trabalhando com operações de convolução bidimensionais, e precisam ser alteradas para adição ou remoção de uma ou mais dimensões.

2.5.2 Detecção de Padrões

A detecção de padrões em uma imagem é feita com diferentes valorações para os elementos de um filtro (iremos utilizar essa nomenclatura para referências ao *kernel* a partir de agora).

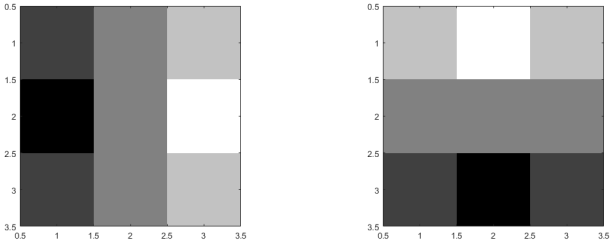
Para achar uma borda, por exemplo, um filtro Sobel (1968) pode ser utilizado. Esse tipo de filtro trabalha com uma matriz de intensidades de uma imagem, onde regiões com uma mudança de intensidade podem ser destacadas como uma borda encontrada. Existem na verdade, dois filtros diferentes, pois eles precisam detectar variações que estão em direções diferentes, um para mudanças horizontais, ou seja, paralelas ao eixo y, e outra para verticais, paralelas ao eixo x.

As seguintes matrizes apresentam as duas variações do filtro Sobel para detecção de bordas verticais(V) e horizontais(H).

$$V = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}; \; H = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Esses filtros geram valores positivos em uma variação de intensidades menores para maiores e valores negativos caso contrário. Se os valores forem altos, isso indica uma mudança brusca de intensidade, enquanto valores pequenos indicam mudanças suaves.

Figura 11 – Os dois filtros Sobel como matrizes em escala de cinza.

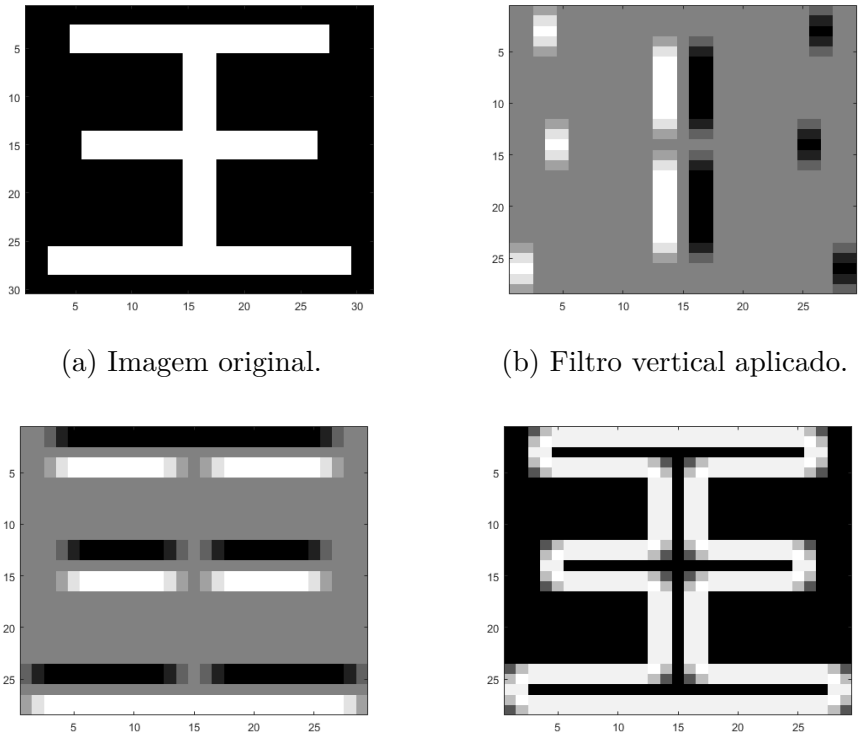


(a) Filtro vertical. (b) Filtro horizontal.

Fonte: Elaborada pelo autor.

A Figura 12 apresenta um exemplo de aplicação dos filtros Sobel em uma imagem simples em escala de cinza, apresentada em (a). Em (b) podemos ver a detecção de bordas verticais na imagem com o uso do filtro vertical. Observamos o mesmo em (c), as bordas horizontais com a aplicação do devido filtro. A imagem (d) foi obtida através da fórmula $G = \sqrt{{G_v}^2 + {G_h}^2}$, onde G_v e G_h são, respectivamente, o resultado da operação de convolução entre a imagem (a) com os filtros vertical e horizontal.

Figura 12 – Exemplo de aplicação dos filtros Sobel.



(a) Imagem original. (b) Filtro vertical aplicado.

(c) Filtro horizontal aplicado. (d) Convolução com ambos os filtros.

Fonte: Elaborada pelo autor.

2.5.3 *Striding*

Na operação de convolução vista até agora, percorremos a matriz da imagem de entrada movendo uma região de mesma dimensão do filtro com passos de tamanho 1, ou seja, a região pula apenas um *pixel* nas colunas e linhas da entrada para operar com o filtro e produzir o *feature map*. Tal operação pode ser vista na Figura 10.

Podemos alterar o tamanho desse passo com a utilização de *striding*.

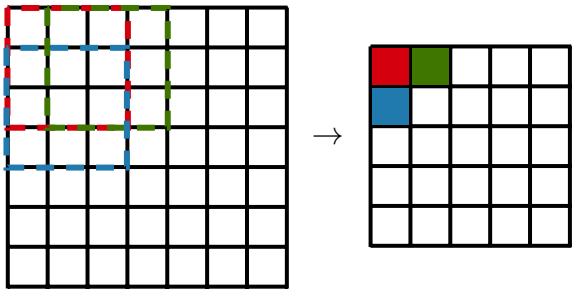
Com a adição do *striding*, o cálculo de dimensão de um *feature map* dado anteriormente em 2.11 pode ser atualizado para:

$$\left\lfloor \frac{n-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{m-f}{s} + 1 \right\rfloor \tag{2.12}$$

onde s é o valor de *striding*.

Tomemos imagem e filtro de dimensão 7×7 e 3×3, respectivamente. A Figura 13 mostra a operação de convolução com $s = 1$, gerando uma saída de dimensão 5×5, onde a formação de três elementos foi destacada para mostrar o passo $s = 1$ de *striding*.

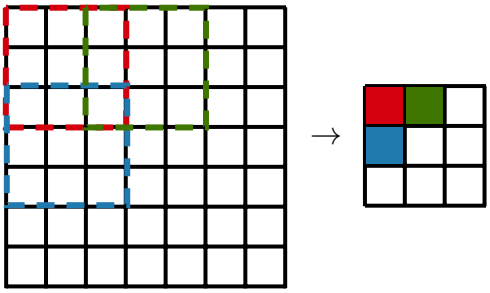
Figura 13 – Convolução com *stride* 1



Fonte: Elaborada pelo autor.

A Figura 14 nos dá a operação sobre mesma imagem e filtro, mas com $s = 2$, gerando uma saída de dimensão 3×3. Novamente, a formação de três elementos foi destacada.

Figura 14 – Convolução com *stride* 2



Fonte: Elaborada pelo autor.

2.5.4 *Padding*

Quando produzimos um *feature map*, temos como saída uma matriz um pouco menor que a imagem original de entrada. Utilizando a primeira fórmula fornecida, o resultado será uma matriz de dimensão $n - f + 1$, mas como vimos acima, podemos utilizar um valor de *striding* maior que 1, o que nos dá um *feature map* com dimensão ainda menor.

Isso acaba se tornando um problema, pois muitas vezes queremos implementar diversas camadas convolucionais em nossa rede, e não queremos que a imagem fique menor a cada produto de uma convolução, pois isso acarretaria em uma perda de informações que podem ser obtidas da imagem de entrada.

Um segundo problema que pode ser encontrado na operação de convolução, é o mal uso de informações presentes nas bordas de uma imagem, pois os *pixels* que estão nelas são utilizados por poucas operações de convolução, devido à maneira de varredura da matriz.

Note que os quatro *pixels* que ficam nas extremidades da imagem aparecem apenas em uma operação. Isso pode ser visto nas Figuras 13 e 14, onde o pixel na primeira posição da imagem aparece na formação apenas do primeiro elemento do resultado. O mesmo acontece com as outras três extremidades da borda. A Figura 14 também mostra que o aumento do *stride* faz outros *pixels* não serem utilizados em mais de uma operação de convolução.

Para a resolução dos problemas acima, uma nova borda pode ser adicionada à imagem de entrada. O nome dado a essa técnica é *zero padding*, ou simplesmente *padding*, pois os elementos da borda inserida são zerados. Com essa borda adicional, poderemos manter a dimensão do *feature map* igual à da imagem original, ou pelo menos desacelerar o processo de redução da imagem. Além disso, os *pixels* que anteriormente pertenciam ao conjunto da borda da imagem, não mais pertencem, e as informações contidas neles podem ser melhor exploradas, pois estarão presentes em mais operações de convolução.

Dada uma quantia de *padding* p e uma imagem de entrada com dimensão $n \times m$, a entrada com borda adicional a ser utilizada terá dimensão $n + 2p \times m + 2p$. O valor de p é multiplicado por 2 para contar ambos os opostos horizontais e verticais da imagem.

Uma convolução pode ser classificada quanto ao seu valor de *padding* como:

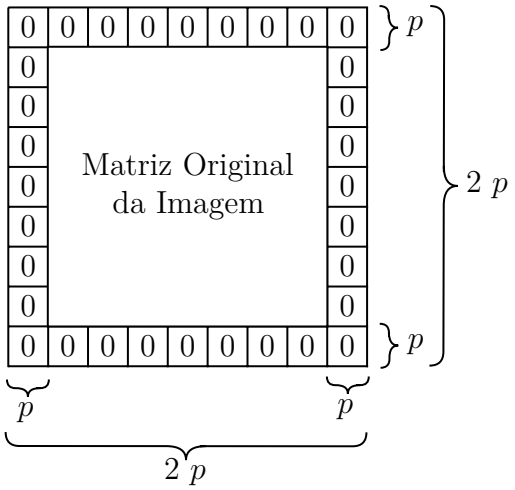
1. *Valid*: Quando nenhum *padding* é aplicado.
2. *Same*: Quando o *padding* aplicado resultará em um *feature map* que mantém a dimensão da imagem de entrada.

Agora podemos atualizar o cálculo da dimensão do *feature map* de 2.12 adicionando o valor de *padding*. Chegamos em:

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{m + 2p - f}{s} + 1 \right\rfloor \tag{2.13}$$

A Figura 15 mostra a adição de uma borda com valor $p = 1$ em uma imagem original de dimensão qualquer. Note como o valor p representa o número de linhas ou colunas que foram adicionadas, de maneira igual em cada lado da matriz, contendo todos os seus elementos zerados.

Figura 15 – Matriz imagem com *Padding* 1



Fonte: Elaborada pelo autor.

2.5.5 *Pooling*

Uma última modificação que pode ser feita na saída de uma camada convolucional é a operação de *pooling*. Sua função é gerar uma nova matriz de saída construída com uma seleção de valores de regiões retangulares da saída original, em uma operação de varredura similar à já vista com a convolução.

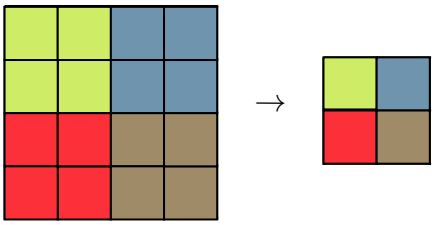
Existem variações na operação de *pooling*, mas a mais utilizada na prática, seleciona da matriz original o valor máximo presente em uma vizinhança de dimensão $f \times f$, onde f é o tamanho da nova matriz gerada para a saída, e damos o nome de *max pooling*.

Como geralmente queremos apenas detectar a existência de algumas características em uma imagem e não saber sua localização exata, com o *pooling* conseguimos um certo nível de invariância à translação, pois não precisamos saber onde em uma certa região o padrão se encontra, mas apenas que ele está lá. Note que ainda possuímos a posição em relação aos quadrantes utilizados na operação, portanto não há uma perda total de informações de posicionamento.

A dimensão da nova saída pode ser calculada com a mesma fórmula apresentada em 2.12, pois *zero padding* não é aplicado em uma camada de *pooling*. Além disso, os hyperparâmetros utilizados aqui são fixos, escolhidos pelo desenvolvedor da aplicação, e não precisam ser calculados pela rede com o uso de *backpropagation*, ou seja, não há nada que precise ser aprendido. Os valores mais utilizados são $f, s = 2$ e $f = 3, s = 2$ (NG, 2017).

A Figura 16 mostra a aplicação de um *pooling* utilizando os parâmetros $f = 2$ e $s = 2$.

Figura 16 – Operação de *Pooling*

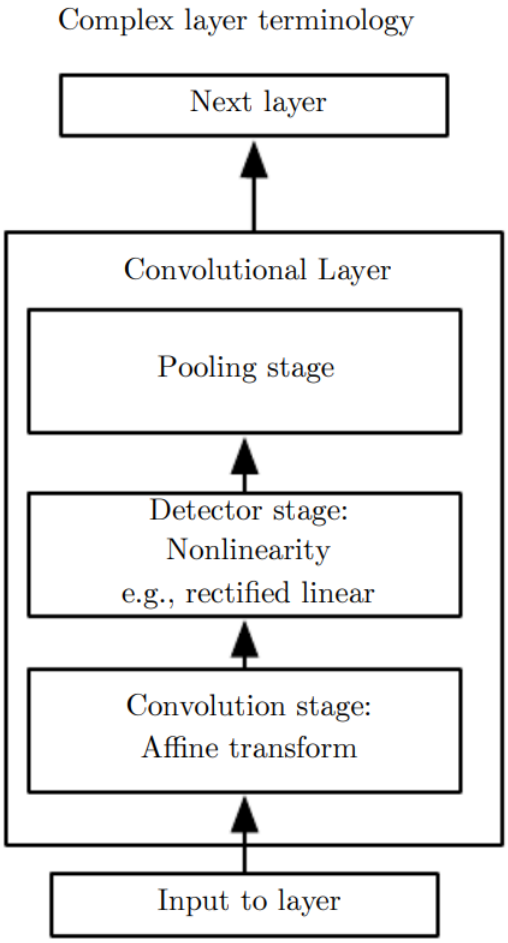


Fonte: Elaborada pelo autor.

2.5.6 Organização de uma CNN

Existem diferentes terminologias para quais passos fazem parte de uma denominada camada convolucional, a Figura 17 representa um esquema de uma possível estruturação da camada de uma rede neural convolucional em uma terminologia complexa daquilo que compõe uma camada convolucional, agrupando diferentes etapas em um conjunto.

Figura 17 – Terminologia de camada convolucional.



Fonte: Adaptado de Goodfellow, Bengio e Courville (2016)

Da figura, podemos descrever uma camada convolucional como o conjunto de três etapas que aplicam transformações nas informações que percorrem a rede. A primeira, *convolution stage*, é onde vemos o percorrimento da entrada e a aplicação da operação de convolução gerando um *feature map*. É também nessa etapa que temos a aplicação de *padding* na imagem de entrada e o uso do valor de *stride*, alterando ou mantendo o tamanho da imagem de saída da etapa em relação ao da tamanho da entrada.

Na segunda etapa, *Detector Stage: Nonlinearity*, é onde temos a presença das funções de ativação, que introduzem a não-linearidade na rede utilizando funções não lineares, como a ReLU apresentada anteriormente, assim a saída não é apenas uma transformação linear da entrada.

Para findar uma camada convolucional, o estágio de *pooling* existe para selecionar apenas as informações importantes enquanto reduz a dimensão do *feature map*, diminuindo o número de parâmetros necessários nas próximas camadas, reduzindo o custo computacional e mantendo uma certa invariância a pequenas translações.

Assim, podemos representar uma camada convolucional como um bloco composto de três estágios, que representam os processos de:

- Convolução
- Função de Ativação não Linear
- *Pooling*

Esse bloco pode se repetir diversas vezes na totalidade de uma rede, realizando as operações sobre a saída da camada anterior, exceto na primeira, onde é utilizada a imagem de entrada. Após todas as camadas convolucionais, uma camada totalmente conectada (*fully connected layer*) é adicionada, ou seja, todos os neurônios se conectam na saída da camada anterior. Ela combina as informações obtidas nas camadas anteriores para identificar padrões maiores nas imagens. Assim conseguimos então ter uma última seção para regressão linear, ou então uma função *softmax* para a aplicação de uma classificação.

2.6 Redes pré-treinadas

As vezes, o custo computacional e de tempo para o aprendizado de uma rede pode ser além do desejado (podendo levar dias para um treinamento) e o uso de uma técnica conhecida como *transfer learning*. Dizemos que é uma transferência de aprendizado, pois uma rede que já foi pré-treinada para um certo problema pode ser reutilizada para a solução de algum problema similar.

Em problemas onde os dados a serem tratados são imagens, redes pré-treinadas com a base de imagens do ImageNet podem ser utilizadas, como a AlexNet (KRIZHEVSKY;

SUTSKEVER; HINTON, 2012), GoogLeNet(Inception) (SZEGEDY et al., 2014) e VGG (SIMONYAN; ZISSERMAN, 2014).

2.7 Métricas de avaliação

Existem diversas métricas para avaliar a qualidade do treinamento de uma rede neural. Essas técnicas podem variar de acordo com a arquitetura e tipo de problema apresentado à rede. Por exemplo, em problemas de classificação, pode-se usar acurácia, precisão, etc. Para problemas de regressão, métricas como o coeficiente de determinação (or R^2) são utilizadas [7] [8].

O coeficiente de determinação pode ser determinado pela fórmula 2.14 [9];

$$R^2 = 1 - \frac{SQE}{STQ} \tag{2.14}$$

Onde SQE é a soma dos quadrados do erro, ou soma dos quadrados dos resíduos, e apresenta o desvio entre os valores reais e os preditos pelo modelo. SQT é a soma total dos quadrados e apresenta a variância presente nos dados. As fórmulas de SQE e SQT estão representadas nas equações 2.15 e 2.16, respectivamente.

$$SQE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.15}$$

onde y_i é o valor da variável a ser prevista, $\hat{y}_i$ é o valor previsto para y_i e n é o número de observações;

$$STQ = \sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{2.16}$$

onde y_i é a variável dependente, $\bar{y}$ é a média dos valores y e n é o número de observações.

Quanto mais próximo de 1 o valor obtido por R^2 for, melhor o modelo conseguiu descrever as variáveis a serem previstas.

[7] https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce

[8] <https://medium.com/usf-msds/choosing-the-right-metric-for-machine-learning-models-part-1-a99d7d7414e4>

[9] <https://www.mathworks.com/help/stats/coefficient-of-determination-r-squared.html>

3 Apresentação dos Dados e Modelo Proposto

Esse capítulo introduz o conjunto de dados e suas características, apresentando as medições de intensidade em elevação e azimutes diferentes que resultarão em matrizes individuais de orientação para cada padrão 2DLS. Nas seções seguintes é descrito como os dados foram normalizados e transformados em matrizes. A seção 3.4 e suas subseções tratam da importante característica de invariância à rotação necessária para a rede. Por fim, um diagrama de fluxo dos dados no sistema é revisado com explicações de cada passo.

3.1 O *Dataset*

O conjunto de dados cedido pela Universidade de *Hertfordshire* contém dados de partículas prismáticas hexagonais, geradas por um simulador, armazenados em arquivos no formato `HDF5` (*Hierarchical Data Format 5*) com informações de 162 partículas atmosféricas, cada uma com 133 orientações. Essas orientações são os três ângulos de Euler denominados α, β e γ, onde representam, respectivamente, inclinações de um corpo que rotaciona em relação aos eixos z, x e y.

Um padrão 2DLS é associado a cada orientação de uma partícula e a intensidade desses padrões de difração, declarada como o atributo *intensity*, é indexada por outros dois atributos: *elevation* e *azimuth*. Elevação (ou altitude) e azimute são ângulos pertencentes ao sistema horizontal de coordenadas, que utiliza o horizonte do observador como um plano que divide uma esfera em dois hemisférios.

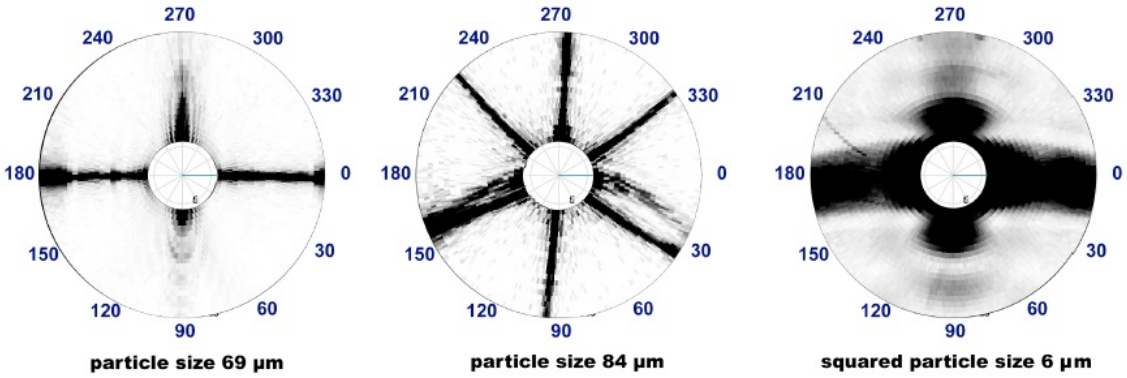
Elevation é dado como o ângulo formado pelo feixe de luz que incide na partícula e foram considerados relevantes apenas aqueles com valores entre 6° e 25°, enquanto em *azimuth*, todos os ângulos foram utilizados — *i.e.* de 0° a 360° (SALAWU, 2015). Assim, temos um total de 7.220 valores de intensidade obtidos pela multiplicação das vinte medições de elevação com as 361 de azimute, ou seja, de $((25 - 6) + 1) \times 361$.

Foi fornecido pelo professor Mauro Roisenberg, o arquivo `carregaImagens.m`, um código `MATLAB`® que lê os arquivos `.h5`, retira as informações necessárias para o cálculo de intensidades e do tamanho projetado de cada partícula, armazenado no atributo *size*. O código finda sua execução gerando um arquivo no formato `.csv` (*Comma-Separated Values*), uma matriz de $n \times 133$ linhas e 7.221 colunas, onde n é o número de partículas analisadas e como colunas são utilizados os valores de intensidade com adição do tamanho projetado no final. Portanto, para o conjunto completo das 162 partículas, a matriz tem

21.546 linhas e 7.221 colunas.

A Figura 18 apresenta exemplos de padrões de difração de luz de três partículas com tamanhos diferentes utilizando uma representação em coordenadas polares.

Figura 18 – Exemplos de padrões de difração.



Alguns exemplos de padrões de difração de partículas com tamanhos diferentes.

Fonte: Priori (2017).

3.2 Normalização dos dados

Para a normalização dos dados, foi utilizado um conjunto composto por dois procedimentos aplicados em sequência. A primeira operação processada sobre o conjunto de dados é a do logaritmo natural ($\ln(x)$ ou $\log_e(x)$), onde sua função é reduzir o intervalo de características para uma escala mais razoável, melhorando o desempenho de uma rede neural (PRIORI, 2017).

Uma uniformização dos dados após o logaritmo natural é feita com a operação de *z-score*, que desloca a média da amostra para zero e a deixa com um desvio padrão igual à um. Dado um vetor z, obtemos o vetor normalizado com a seguinte fórmula:

$$x' = \frac{x - \bar{x}}{s} \tag{3.1}$$

substituindo cada valor x do vetor z com um x', onde $\bar{x}$ e s são, respectivamente, a média e o desvio padrão da amostra. De acordo com Salawu (2015), essas duas técnicas aplicadas em sequência geraram os melhores resultados em seus experimentos.

As operações de normalização descritas acima foram aplicadas em cima da matriz composta por todas as orientações de cada partícula utilizada.

3.3 Gerando imagens de partículas e conjuntos de treino e teste

Após a normalização dos dados, cada linha da matriz, representando uma orientação diferente de uma partícula, é separada em uma matriz própria com duas dimensões e tamanho 20×361 — *i.e. elevation* $\times$ *azimuth*. Essa operação é dada por uma função de *reshape*, que seleciona, em agrupamentos de 20, os elementos de uma linha da matriz original para gerar cada coluna da nova matriz. Como $7220/20 = 361$, temos nossas 361 colunas de *azimuth*. Um exemplo em imagem de uma partícula nesse estado pode ser visto na Figura 19 da próxima seção, sobre Invariância à Rotação.

O conjunto de todas as imagens é então dividido em dois subconjuntos. O primeiro contém 70% dos padrões e é utilizado durante a etapa de treino da rede neural, enquanto o segundo engloba os 30% restantes dos dados e é utilizado para testes. As imagens presentes no conjunto de testes sofrerão uma operação de rotação para aproximar mais o conjunto com dados obtidos no mundo real, pois nem sempre a coleta de difração se dará na mesma posição espacial, e queremos verificar se a rede consegue prever o tamanho de uma partícula nessas condições. Não foram separados dados para uma etapa de validação.

3.4 Invariância à Rotação

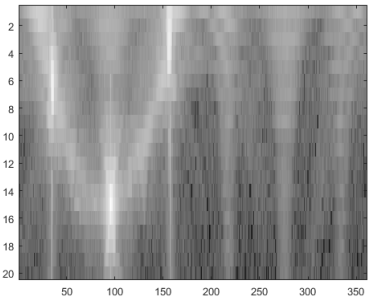
Durante o processo de coleta dos padrões 2DLS de uma partícula, ela pode estar rotacionada em qualquer um dos 360 ângulos possíveis em torno do eixo do *laser* disparado, gerando uma imagem de difração diferente para cada grau de medição. Como uma partícula, mesmo estando rotacionada, deve sempre possuir o mesmo tamanho, queremos que o treinamento da rede se dê de tal forma que ela seja invariante à rotações nas imagens.

Para reproduzir o efeito de λ graus de rotação em uma partícula, foi utilizado uma função de *circular shift* (deslocamento circular) em cima da imagem — *i.e.* todas as colunas serão deslocadas λ posições em um determinado sentido, caso chegue no final da imagem será deslocada para o lado oposto. Uma função de geração de números pseudo-aleatórios com distribuição normal foi utilizada para criar um vetor r, de tamanho igual ao número de partículas do conjunto de teste e com valores inteiros entre 1 e 360. Cada elemento pertencente ao vetor r representa um grau λ de rotação a ser aplicado na respectiva partícula de mesma posição no conjunto de teste.

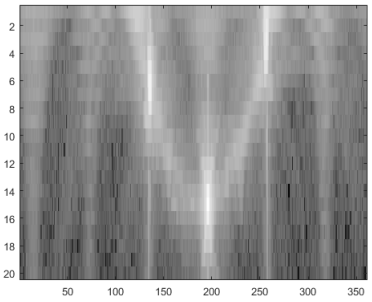
A Figura 19 apresenta três imagens do padrão 2DLS da partícula `l3.5_d6.9_flat` em sua sétima orientação, que possui um tamanho projetado $27.3350\mu m$. As primeiras duas imagens(a e b) mostram, respectivamente, o padrão de difração da partícula em sua forma original e o mesmo após sofrer uma rotação de 100°. A ilustração (c) é um gráfico dos valores da primeira elevação e todos os azimutes presentes nas figuras (a) e (b),

podemos visualmente notar o deslocamento de 100 unidades para a direita. Utilizaremos essa mesma partícula para a continuação dos exemplos sobre invariância à rotação no restante desse capítulo.

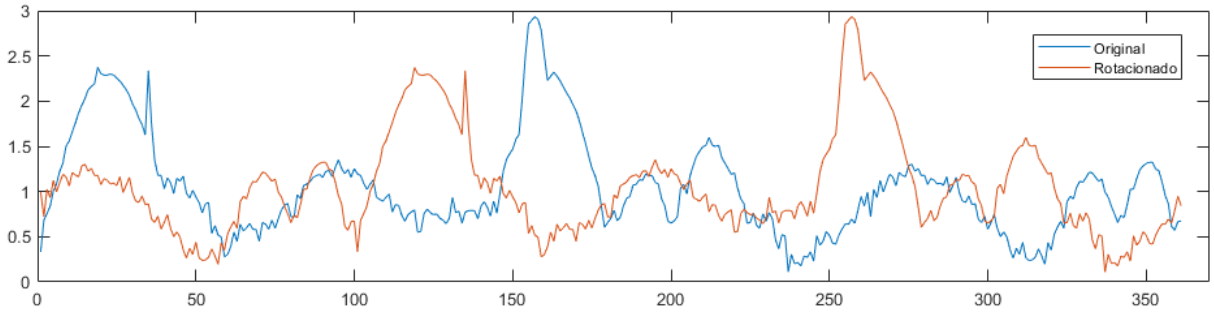
Figura 19 – Exemplos de rotação nos padrões de difração.



(a) Padrão original.



(b) Padrão rotacionado.



(c) Gráfico da primeira elevação.

Exemplo de rotação de 100° no padrão 2DLS de uma partícula.

Fonte: Elaborada pelo autor.

3.4.1 *Fast Fourier Transform*

Para conseguir um sistema que seja invariante à rotação, foi utilizado o método de *Fast Fourier Transform*(FFT) [1], assim como no trabalho de Priori (2017). Aplicando uma operação de FFT, saímos do domínio espacial na imagem e trabalhamos em cima do domínio de frequências.

A Figura 20 apresenta as imagens obtidas após a aplicação do método de FFT a partir dos dados anteriores da partícula (original e rotacionada). Para gerar as figuras (a) e (b), foram extraídos os valores absolutos para obter a magnitude do sinal FFT [2] e utilizada a função de *FFT shift* para deslocar os componentes de frequência zero para o centro da matriz, por questões de visibilidade. A imagem (c) expõe a diferença entre

[1] Em português, Transformada Rápida de Fourier

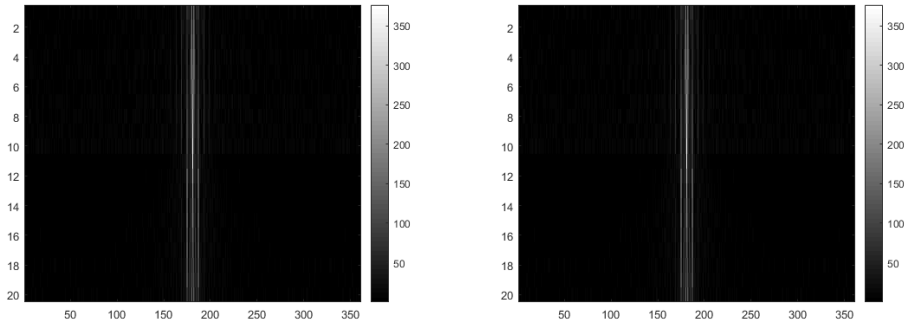
[2] <https://www.mathworks.com/help/signal/examples/practical-introduction-to-frequency-domain-analysis.html>

os valores de (a) e (b), podemos notar na barra de cores que os valores estão com uma precisão de $\times 10^{-14}$. De fato, se procurarmos pelos valores extremos de (c), encontramos:

- Mínimo: -5.6843×10^{-14}
- Máximo: 5.6843×10^{-14}

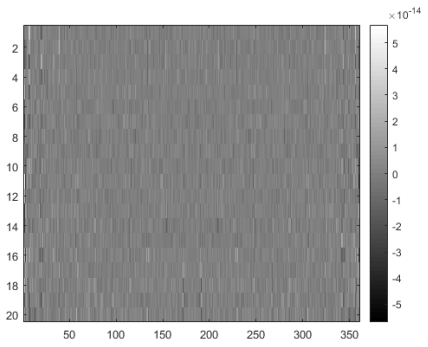
Podemos então classificar as duas imagens como sendo iguais, pois atingem uma precisão adequada para o modelo proposto.

Figura 20 – FFT.



(a) FFT do padrão original.

(b) FFT do padrão rotacionado.



(c) Diferença entre (a) e (b).

Exemplo de aplicação da função FFT em uma partícula.

Fonte: Elaborada pelo autor.

3.4.2 *Inverse Fast Fourier Transform*

Aplicamos a função de *Inverse Fast Fourier Transform*(IFFT) [3] para sair do domínio de frequências e voltar ao domínio espacial. Para isso, não pode-se simplesmente aplicar diretamente a função IFFT no resultado obtido via FFT, pois as imagens retornariam para os padrões 2DLS originais, perdendo a invariância à rotação fornecida pelo

[3] Em português, Transformada Inversa Rápida de Fourier

uso da função FFT. Aplicando o IFFT nos valores absolutos, ou de magnitude, gera-se imagens que são mais próximas do que seria um padrão normal de difração, porém ainda mantendo a invariância à rotação. Por esses resultados com as imagens, surgiu a ideia de realizar um experimento com IFFT, além do com apenas FFT, para observar se o uso de IFFT trará algum impacto positivo nos resultados.

A Figura 21 apresenta as imagens obtidas após a aplicação do método de IFFT a partir dos valores absolutos das matrizes geradas pelo FFT apresentadas na subseção anterior e na Figura 20, porém sem o uso da função de *FFT shift*. Em (a) e (b) podemos ver as imagens em escala de cinza resultantes da aplicação de IFFT como descrito acima. A imagem (c) expõe a diferença entre os valores de (a) e (b), podemos notar na barra de cores que os valores estão com uma precisão de $\times 10^{-15}$. De fato, se procurarmos pelos valores extremos de (c), encontramos:

- Mínimo: -1.7764×10^{-15}
- Máximo: 1.7764×10^{-15}

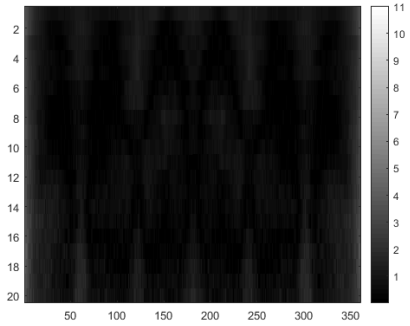
Podemos então classificar as duas imagens como sendo iguais, pois atingem uma precisão desejada.

3.4.3 Espelhamento de FFT e IFFT

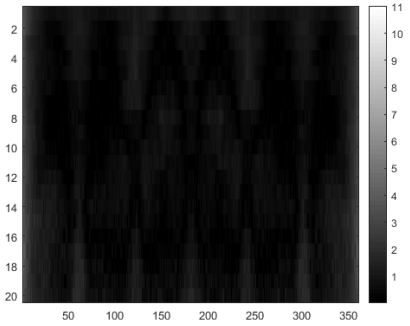
O resultado das operações de FFT e IFFT é simétrico em relação ao eixo y. Isso vem da natureza da Transformada Discreta de Fourier e suas propriedades, pois estamos aplicando a operação em cima de vetores com valores reais (*i.e.* com parte imaginária zerada), gerando uma função par (simétrica em relação ao eixo y) como sinal de magnitude e uma ímpar (simétrica em relação à origem) para o sinal de fase.

Realmente, nota-se que esse espelhamento está presente nas imagens (a) e (b) de ambas as Figuras 20 e 21. Portanto, foram utilizadas para treinamento e teste da CNN desenvolvida apenas metade de cada imagem, com isso reduzindo em cerca de 50% o tempo de treino da rede.

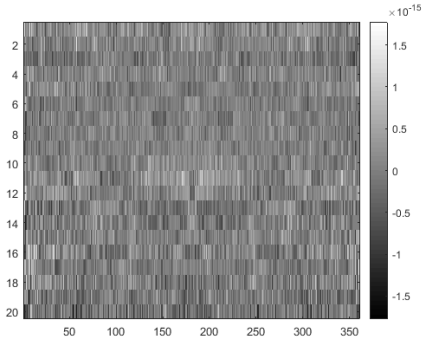
Figura 21 – IFFT.



(a) FFT do padrão original.



(b) FFT do padrão rotacionado.



(c) Diferença entre (a) e (b).

Exemplo de aplicação da função IFFT em uma partícula.

Fonte: Elaborada pelo autor.

3.5 Fluxograma do sistema desenvolvido

A Figura 22 apresenta um diagrama do fluxo geral de execução em 17 passos. É dividido em dois estágios [4], o primeiro (passos 1 à 11) com o pré-processamento dos dados e o segundo (passos 12 à 17) com o treinamento e teste da CNN.

Nos passos 1 à 3, têm-se o carregamento do *dataset*, uma matriz gerada pelo código fornecido mencionado anteriormente. Os dados passam por uma normalização e então uma matriz para cada padrão é gerada. Em 4, uma permutação aleatória é feita para dividir o conjunto de matrizes e seus respectivos valores de tamanho projetado em 70% dados de treino (5) e os restantes 30% para dados de teste (6). Os padrões selecionados para treino são então rotacionados no passo 7.

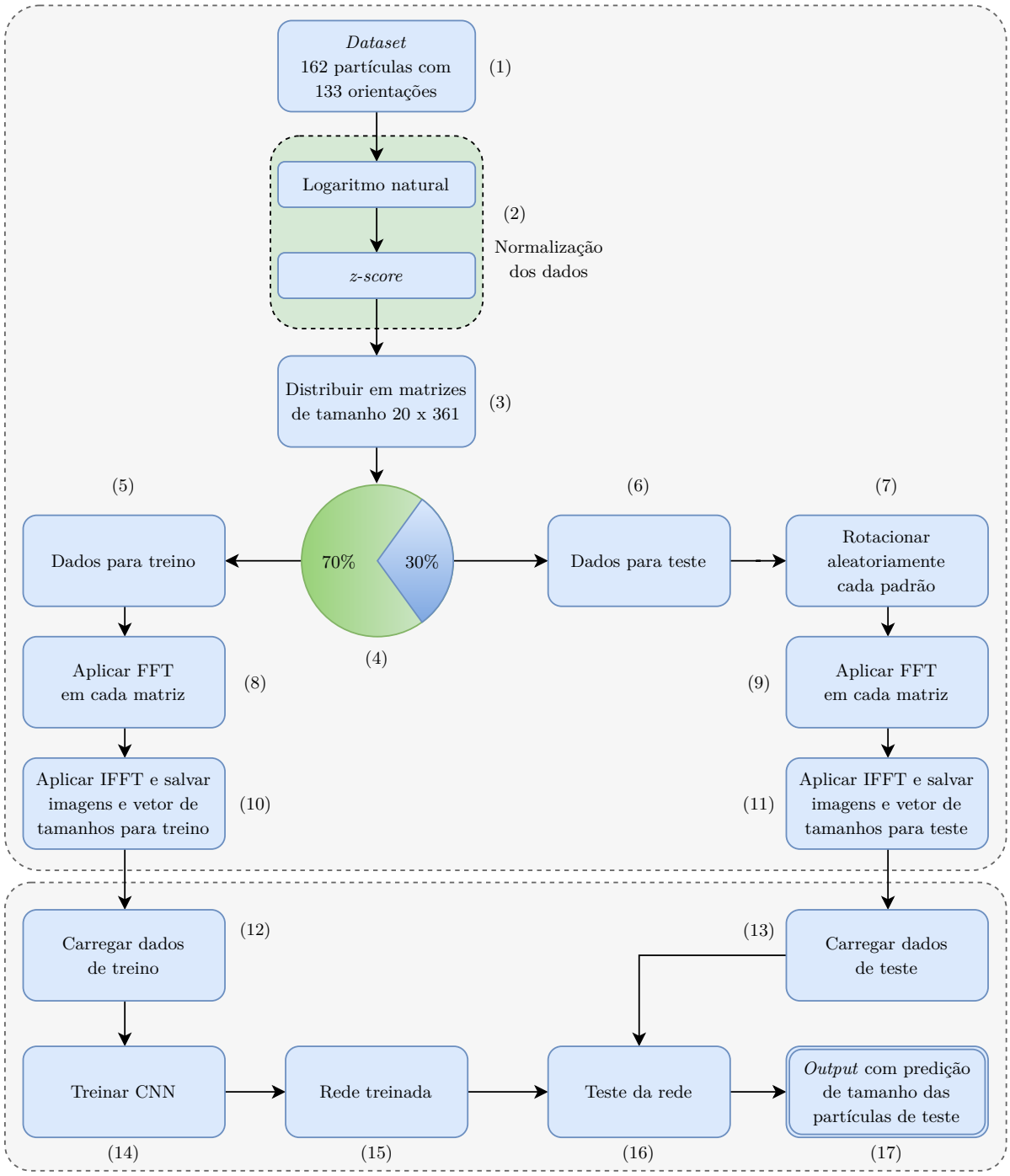
A aplicação da operação de FFT nos passos 8 e 9 é onde garantimos a invariância à rotação do sistema. Em 10 e 11 aplicamos a IFFT, voltando para o domínio espacial. As matrizes resultantes do IFFT são transformadas em imagens em escala de cinza e

[4] Cada estágio se encontra em um arquivo próprio no código desenvolvido.
Repositório com o código: <https://github.com/ViniciusBiermann/TCC/tree/master/code>

salvas em um diretório `FFTImages`, onde há a separação dos dados de treino e teste nos subdiretórios `TrainSet` e `TestSet`, respectivamente. Em `FFTImages` também são salvos dois arquivos de extensão *Comma-Separated Values*, `TrainSizes.csv` e `TestSizes.csv`, contendo os tamanhos projetados de cada orientação na mesma ordem em que foram salvas as imagens. Como descrito anteriormente, os resultados de FFT e IFFT são simétricos, portanto salvaremos apenas metade de cada imagem, isso deixará o treinamento da rede mais rápido.

O segundo estágio começa com os passos 12 e 13, onde são carregadas as imagens e vetores de tamanhos para treino e teste. Em 14, ocorre o treinamento da CNN com a arquitetura implementada, que será explicada no capítulo 4. Com o resultado do treinamento da rede (15), seu comportamento pode ser testado (16) com os dados de padrões rotacionados carregados para teste (13), sendo esses dados não apresentados à rede anteriormente. É, então, gerada uma saída (*output*) com a predição do tamanho projetado das partículas prismáticas hexagonais advindas dos padrões 2DLS do conjunto de teste, onde podemos calcular a qualidade do aprendizado da rede comparando com os dados reais de tamanho carregados em (13), através de métricas como coeficientes de correlação e determinação de regressão linear.

Figura 22 – Fluxograma do sistema.



Fonte: Elaborada pelo autor.

4 Experimentos e Resultados

Esse capítulo contém a organização de dois experimentos propostos para verificar se o uso de IFFT após FFT trará algum impacto na qualidade da predição de tamanho das partículas. O caso de uso de IFFT é tratado no experimento I, enquanto no experimento II modificamos o salvamento das imagens para usar apenas o que veio da função FFT.

A configuração de *hardware* da máquina onde os treinos e testes foram executados conta com uma placa de vídeo (GPU) NVIDIA GeForce® GTX 1080 Ti e 64GB de memória RAM. A rede de ambos os experimentos foi treinada com opção de ambiente de execução definido como GPU nas opções providas pelo MATLAB®.

Em cada modelo, são amostrados a qualidade dos resultados de treino e de teste, com uma representação gráfica da regressão linear. Na última seção desse capítulo, é feita uma amostragem com exemplos de partículas de diferentes tamanhos; e então, o desempenho da melhor abordagem de Priori (2017) é comparado com os resultados obtidos nos dois experimentos realizados nesse trabalho.

4.1 Experimento I

O primeiro modelo de CNN desenvolvido utiliza as imagens em escala de cinza obtidas a partir das matrizes de IFFT.

A arquitetura montada para o experimento I foi a seguinte:

1. *Image Input*: Camada de entrada onde define-se o tamanho da imagem, no caso definimos 20×181, e o número de canais. Como estamos lidando com imagens em escala de cinza, temos apenas um canal. A entrada também é normalizada subtraindo a média.

2. *Convolution*: Camada convolucional que consiste de 32 filtros 7×7 com *stride* $s = 1$ e *padding* $p = same$.

3. *ReLU*

4. *Batch Normalization*: Normalização do *batch* com 32 canais.

5. *Max Pooling*: Valores máximos em uma região de tamanho 1×3 com *stride* $s = [1\ 2]$ e *padding* $p = same$.

6. *Convolution*: Camada convolucional que consiste de 64 filtros 5×5 com *stride* $s = 1$ e *padding* $p = 0$.

7. *ReLU*
8. *Batch Normalization*: Normalização do *batch* com 64 canais.
9. *Max Pooling*: Valores máximos em uma região de tamanho 1×3 com *stride* $s = [1\ 2]$ e *padding* $p = same$.
10. *Convolution*: Camada convolucional que consiste de 64 filtros 3×3 com *stride* $s = 1$ e *padding* $p = same$.
11. *ReLU*
12. *Batch Normalization*: Normalização do *batch* com 64 canais.
13. *Max Pooling*: Valores máximos em uma região de tamanho 1×3 com *stride* $s = [1\ 2]$ e *padding* $p = same$.
14. *Fully Connected*: Camada totalmente conectada com 50 neurônios.
15. *ReLU*
16. *Fully Connected*: Camada totalmente conectada com 30 neurônios.
17. *ReLU*
18. *Fully Connected*: Camada totalmente conectada com 1 neurônio.
19. *ReLU*
20. *Regression Output*: Camada que calcula a perda via erro médio quadrático.

Nas camadas de *Max Pooling*, foram utilizadas regiões de tamanho 1×3 e valor de *stride* $s = [1\ 2]$ — *i.e.* deslocamentos em uma linha e duas colunas. Como as imagens possuem tamanho 20×181, não é desejável que ela seja reduzida igualmente em número de linhas e colunas, pois existem muito mais colunas que linhas, e a imagem ficaria pequena demais muito rápido, perdendo informação no processo.

Além da estruturação da rede em camadas, opções de treinamento devem ser escolhidas. As opções para treinamento foram definidas como:

- Número máximo de épocas: 100
- Taxa de aprendizado inicial: 1^{-4}
- Fator de diminuição da taxa de aprendizado: 0.1
- Período de diminuição da taxa de aprendizado: 40

- Ambiente de execução: GPU
- Embaralhar em cada época

Com essas opções, a taxa de aprendizado iniciará em 1^{-4} e a cada 40 épocas cairá em uma taxa de 0.1, ou seja, na época 40 ficará 1^{-5}, na 80 em 1^{-6} e encerrará o treinamento na época 100. Um embaralhamento dos dados de treino ocorre antes de cada época para evitar qualquer viés entre padrões nos lotes de treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

Foram utilizadas várias arquiteturas de camadas e opções de treinamento até chegar nos valores descritos acima. Foi testado, por exemplo, números diferentes de camadas convolucionais e um número máximo de épocas maior, porém foi reduzido por não apresentar melhoras nos resultados. Muitas configurações diferentes impactaram pouco nos resultados, mas algumas, como aumentar a taxa de aprendizado inicial para valor maior causava um crescimento intenso no erro médio quadrático.

Após treinar a rede, pode-se verificar o seu desempenho através de índices como o coeficiente de correlação (R) e o coeficiente de determinação (R^2). A Figura 23 apresenta a regressão linear com R da predição feita pela rede dos tamanhos das partículas de treino, ou seja, aquelas que ela usou para aprender os padrões comuns identificados entre as imagens de orientações que possuem tamanho projetado aproximado. Da imagem têm-se que $R = 0.9983$ e $R^2 = 0.9966$.

Figura 23 – Resultado de treino com IFFT

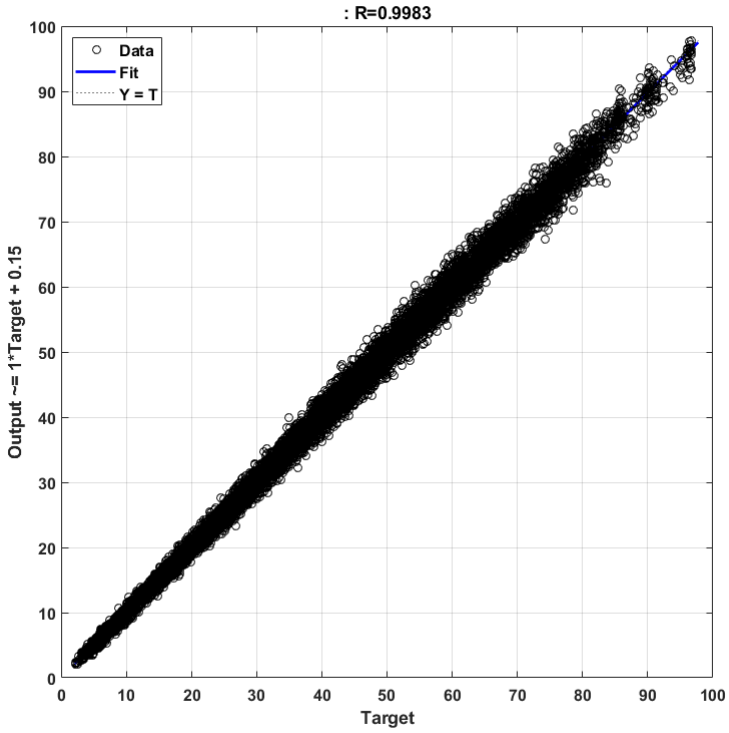


Gráfico com tamanhos previsto e real com conjunto de treino no experimento I. Cada elemento representa uma imagem de orientação de uma partícula. No eixo x estão os valores reais das orientações, enquanto no eixo y têm-se o valor predito pela rede.

Fonte: Elaborada pelo autor.

Mais interessante que os dados de treino, precisamos observar o comportamento da rede com os dados de teste, pois eles que revelarão a real qualidade do aprendizado e se o sistema conseguiu com sucesso ser invariante à rotação, visto que todos os elementos do conjunto de testes sofreram rotações em graus aleatórios. A Figura 24 apresenta a regressão linear com R da predição feita dos tamanhos das partículas do conjunto de teste.

Figura 24 – Resultado de teste com IFFT

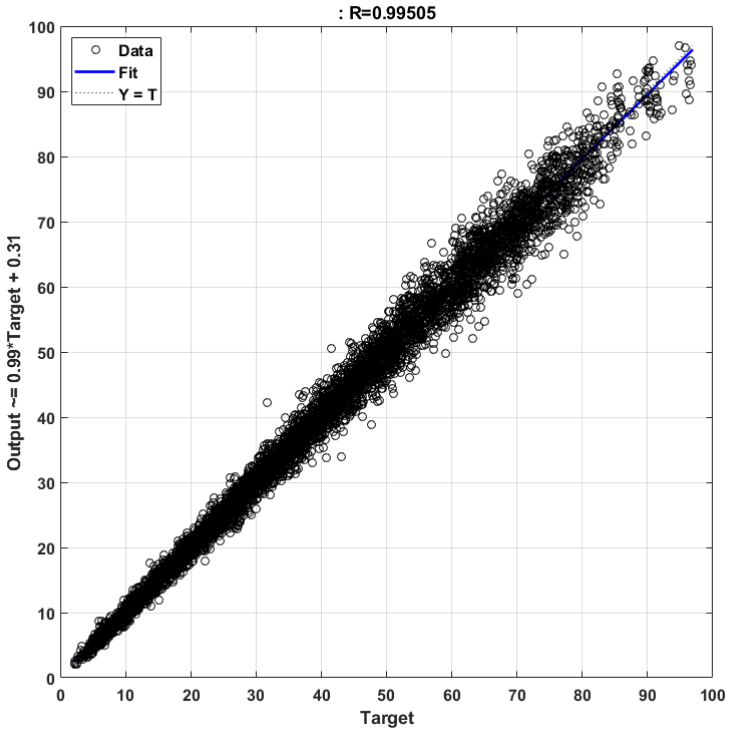


Gráfico com tamanhos predito e real com conjunto de teste no experimento I. Cada elemento representa uma imagem de orientação de uma partícula. No eixo x estão os valores reais das orientações, enquanto no eixo y têm-se o valor predito pela rede.

Fonte: Elaborada pelo autor.

Podemos notar que os resultados são satisfatórios, principalmente para partículas de menor tamanho e mesmo havendo um espalhamento maior para partículas maiores, ainda temos predições aceitáveis. O modelo utilizando imagens a partir do IFFT conseguiu um coeficiente de correlação $R = 0.99505$ e coeficiente de determinação $R^2 = 0.9901$.

No total, o experimento I levou 11 minutos e 59 segundos para concluir o treinamento, executou 117 iterações por época, resultando num total de 11.700 iterações.

4.2 Experimento II

O segundo modelo de CNN desenvolvido utiliza as imagens geradas a partir apenas da função FFT, sem a aplicação de sua inversa. Queremos saber se a CNN consegue resultados equivalentes. Em caso positivo, pode-se reduzir $n \times 133$ operações de IFFT durante o pré-processamento dos dados, onde n é o número total de partículas utilizadas, resultando em uma execução mais rápida.

A organização de camadas e opções de treinamento do experimento II são as mesmas utilizadas no experimento I (consulte seção 4.1).

A Figura 25 apresenta a regressão linear com R da predição feita pela rede dos tamanhos das partículas de treino, ou seja, aquelas que ela usou para aprender os padrões comuns identificados entre as imagens de orientações que possuem tamanho projetado aproximado. Da imagem tiramos que $R = 0.99728$ e $R^2 = 0.9945$.

Figura 25 – Resultado de treino com FFT

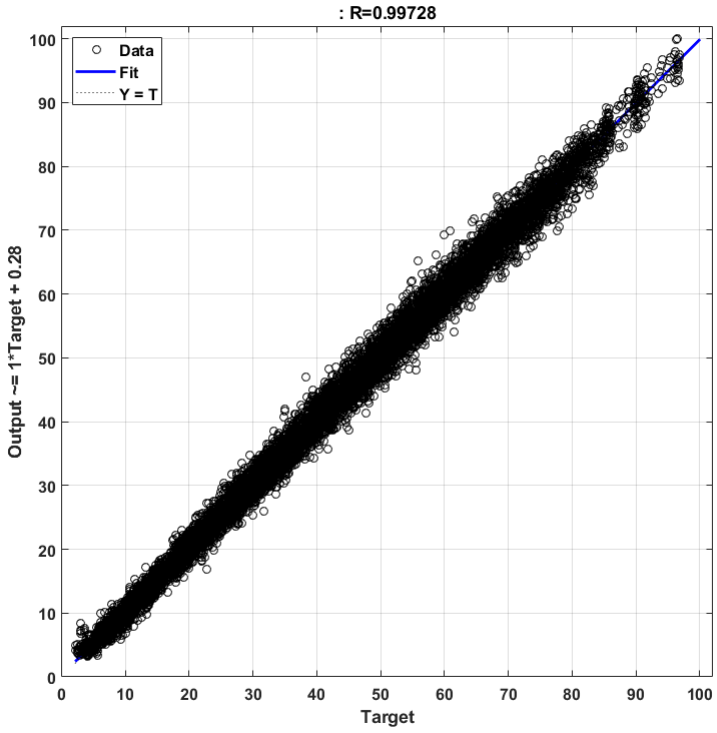


Gráfico com tamanhos predito e real com conjunto de treino no experimento II. Cada elemento representa uma imagem de orientação de uma partícula. No eixo x estão os valores reais das orientações, enquanto no eixo y têm-se o valor predito pela rede.

Fonte: Elaborada pelo autor.

Novamente, quer-se verificar a qualidade da predição de tamanho da rede com os dados rotacionados do conjunto de testes. A figura 26 mostra a regressão linear juntamente com o coeficiente de correlação.

Figura 26 – Resultado de teste com FFT

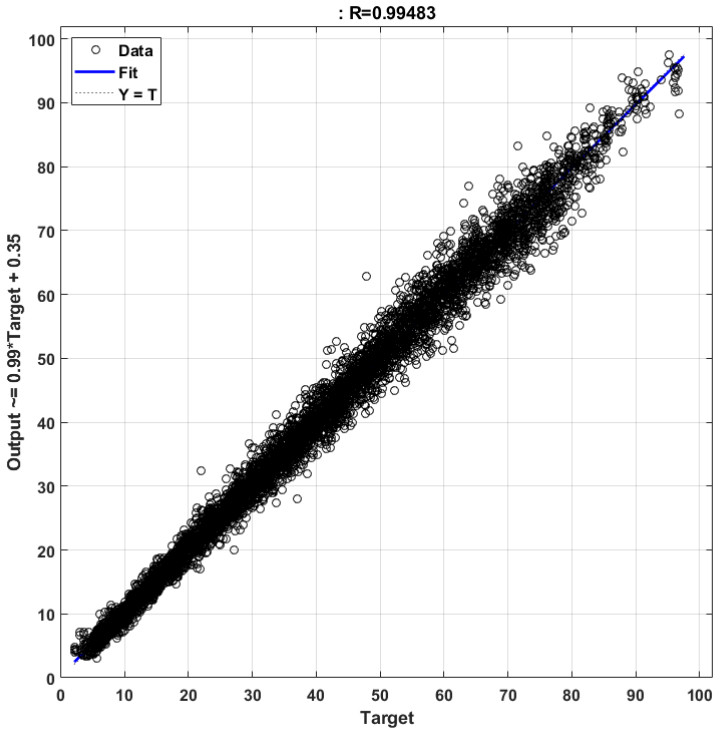


Gráfico com tamanhos predito e real com conjunto de teste no experimento II. Cada elemento representa uma imagem de orientação de uma partícula. No eixo x estão os valores reais das orientações, enquanto no eixo y têm-se o valor predito pela rede.

Fonte: Elaborada pelo autor.

Nota-se que os resultados, apesar de inferiores aos do experimento I, ainda conseguiram atingir um nível satisfatório de qualidade, com $R = 0.99483$ e $R^2 = 0.9897$. Porém, nota-se que a rede teve uma dificuldade maior para prever tamanhos de pequenas partículas, enquanto o espalhamento de dados em tamanhos maiores está mais similar com os do experimento I.

No total, o experimento II levou 12 minutos e 04 segundos para concluir o treinamento, executou 117 iterações por época, resultando num total de 11.700 iterações.

4.3 Comparações

A Tabela 2 compara os resultados do modelo 4 (*Autoencoders*) de Priori (2017), que atingiu a melhor média de performance, com os obtidos nos experimentos I e II. Foram escolhidas todas as quarenta e nove partículas de diferentes tamanhos projetados utilizadas por Priori para comparar seus modelos para testar o comportamento do sistema com apenas as 133 orientações de cada partícula. O desempenho dos modelos foi dado pelo seu coeficiente de determinação (R^2). Na primeira coluna da tabela, têm-se que P é o número da partícula, C é seu comprimento e D o diâmetro.

Tabela 2 – Comparação de resultados.

Partícula	Performance (R^2)		
	Priori	Experimento I	Experimento II
P:1; C:1.1; D:9.1	**0.956**	0.857	0.631
P:2; C:1.5; D:8.5	**0.959**	0.908	0.674
P:5; C:10.3; D:82.6	**0.987**	0.977	0.978
P:7; C:102.5; D:34.2	0.942	**0.953**	0.948
P:8; C:105.7; D:19.2	0.958	**0.964**	0.950
P:10; C:108.9; D:36.3	0.944	**0.946**	0.940
P:13; C:11.7; D:93.6	**0.981**	**0.981**	0.979
P:16; C:115.3; D:21.0	0.957	**0.972**	0.969
P:17; C:115.6; D:38.5	0.887	0.956	**0.957**
P:18; C:118.0; D:14.7	0.959	**0.974**	0.957
P:20; C:12.3; D:98.5	0.964	0.980	**0.983**
P:26; C:13.0; D:2.4	**0.976**	0.857	0.736
P:27; C:13.2; D:105.5	**0.987**	0.986	0.979
P:31; C:14.0; D:28.1	**0.805**	0.759	0.644
P:34; C:143.2; D:26.0	0.944	**0.979**	0.968
P:37; C:15.7; D:2.0	**0.974**	0.829	0.658
P:38; C:15.8; D:15.8	**0.896**	0.719	0.593
P:39; C:15.8; D:5.3	**0.969**	0.876	0.714
P:44; C:16.6; D:49.9	0.884	**0.917**	0.907
P:66; C:22.6; D:45.2	0.776	**0.794**	0.759
P:68; C:23.0; D:69.0	0.916	0.916	**0.934**
P:70; C:24.5; D:73.5	0.927	0.931	**0.933**
P:71; C:24.7; D:49.4	0.746	**0.778**	0.775
P:75; C:26.9; D:3.4	**0.989**	0.946	0.863
P:77; C:28.3; D:14.2	**0.957**	0.901	0.759

Continua na próxima página

Tabela 2 – Continuação da página anterior

Partícula	Performance (R^2)		
	Priori	Experimento I	Experimento II
P:79; C:29.7; D:9.7	**0.970**	0.948	0.870
P:83; C:3.5; D:19.2	**0.974**	0.927	0.881
P:85; C:3.9; D:31.6	**0.987**	0.959	0.951
P:87; C:31.6; D:5.7	**0.981**	0.934	0.928
P:98; C:39.7; D:39.7	0.761	**0.860**	0.794
P:99; C:4.1; D:12.3	**0.932**	0.786	0.631
P:101; C:4.6; D:37.2	**0.981**	0.974	0.966
P:104; C:43.2; D:43.2	**0.830**	0.827	0.744
P:106; C:46.5; D:46.5	0.749	**0.851**	0.794
P:112; C:5.3; D:5.3	**0.881**	0.368	0.474
P:114; C:5.6; D:11.1	**0.882**	0.528	0.515
P:115; C:5.6; D:16.9	**0.926**	0.860	0.732
P:118; C:54.6; D:27.3	0.874	**0.927**	0.897
P:119; C:55.8; D:18.6	**0.954**	0.952	0.905
P:121; C:59.3; D:10.8	**0.986**	0.966	0.941
P:124; C:6.7; D:53.9	0.969	0.973	**0.974**
P:131; C:68.5; D:12.5	0.964	**0.967**	0.946
P:134; C:7.4; D:40.5	**0.980**	0.938	0.938
P:146; C:8.9; D:26.6	**0.964**	0.880	0.804
P:147; C:8.9; D:8.9	**0.776**	0.579	0.288
P:148; C:81.1; D:40.5	0.877	**0.950**	0.930
P:150; C:83.6; D:10.4	**0.968**	0.953	0.943
P:159; C:94.9; D:11.9	**0.982**	0.977	0.948
P:160; C:95.6; D:31.9	0.923	**0.963**	0.952
Média de performance	0.924	0.888	0.835

Fonte: Elaborada pelo autor.

Nota-se que o experimento I atingiu resultados superiores ao experimento II em quase todas as partículas presentes na tabela, exibindo uma média nos valores de coeficiente de determinação do conjunto de amostra de 0.888, enquanto o experimento II alcançou média de 0.835. A média obtida no experimento I também está mais próxima do resultado de Priori, que adquiriu 0.924.

Os melhores resultados para cada partícula podem ser vistos em negrito. Em geral, o modelo de Priori conseguiu melhores resultados, sendo superiores aos outros dois em vinte e oito casos e com mesmo valor em um caso com o experimento I. Em quinze casos, o

experimento I atingiu resultados melhores. O experimento II foi superior em cinco casos. As partículas 112, 114 e 147 foram as que tiveram piores predições, sendo inferiores ao modelo de Priori por uma grande margem.

A Figura 27 apresenta a regressão linear com duas partículas diferentes utilizando a rede treinada no experimento I. À esquerda, pode-se ver o resultado de uma partícula não quadrada, atingindo $R^2 = 0.981$, enquanto à direita têm-se uma partícula quadrada com um resultado inferior de $R^2 = 0.579$.

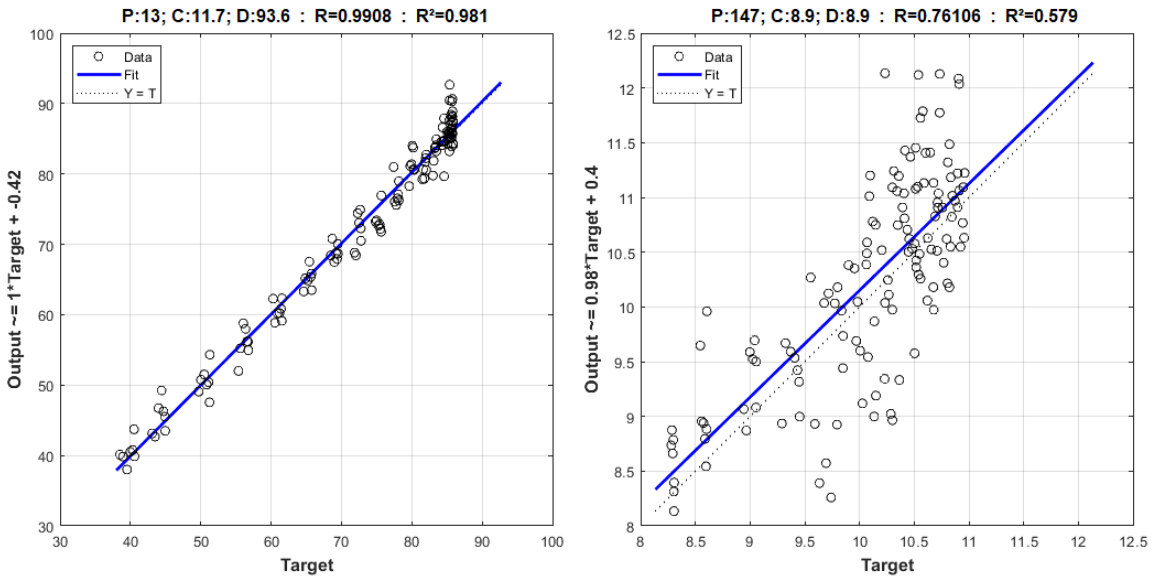


Figura 27 – Resultados individuais de duas partículas com o Experimento I.

Em geral, nos dois experimentos, e em todos os modelos de Priori, as partículas quadradas — *i.e.* aquelas com mesmo valor de comprimento e diâmetro, obtiveram predições piores, alcançando valores muito baixos no coeficiente de determinação. De acordo com Priori (2017), resultados ruins provavelmente se dão pela similaridade encontrada em padrões de partículas de tamanhos diferentes, mas que possuem mesmos valores de *aspect ratio* e orientação. Esse comportamento fica evidente nos resultados amostrados na Tabela 2, pode-se notar que em geral partículas menores, e especialmente as quadradas, atingiram resultados inferiores às maiores, enquanto as maiores atingiram resultados satisfatórios (*e.g.* a partícula quadrada 98).

5 Conclusões

Nesse trabalho foram realizados dois experimentos utilizando ambientes de redes neurais convolucionais (CNN) para predizer o tamanho projetado de um grupo com simulações de pequenas partículas cristalinas hexagonais presentes na atmosfera da Terra. O conjunto de dados cedido pela universidade de *Hertfordshire* conta com simulações dos padrões 2DLS de 162 partículas, cada uma com 133 orientações diferentes.

Como os dados de partículas reais coletadas podem estar rotacionados, o código desenvolvido precisou ser invariante a tais rotações. Para tanto, foi empregado o uso de Transformadas Rápidas de Fourier (FFT) durante o pré-processamento, levando as imagens do domínio espacial para o domínio de frequências. Assim, como demonstrado na seção 3.4.1, os padrões normal e rotacionado de uma partícula resultam em matrizes com diferença de valor na ordem de $\times 10^{-14}$.

O uso da função inversa da FFT, denominado IFFT, foi uma proposta inicial para o treino da rede, pois uma imagem gerada a partir do IFFT está no domínio espacial e tem características que lembram mais as imagens originais dos padrões se comparado com aquelas geradas apenas pela função FFT. Após a aplicação de IFFT nos resultados de padrão original e rotacionado, a diferença de valor entre os dois resultados ficou na ordem de $\times 10^{-15}$, possibilitando também o uso de IFFT para garantir a invariância à rotação no sistema.

Foi observado que devido à natureza da FFT (e IFFT), garantiu-se que as matrizes geradas a partir dos valores reais de intensidade tivessem a propriedade de uma função par, possuindo uma simetria em relação ao eixo y. Como o espelhamento faz com que os padrões da primeira metade da imagem apenas se repitam na segunda, optou-se por utilizar apenas uma parte [1].

Os dois experimentos elaborados visaram treinar e testar uma rede CNN com e sem a aplicação da função de IFFT após a de FFT. O experimento I, com IFFT, atingiu um coeficiente de determinação $R^2 = 0.9901$ durante a fase de testes, enquanto o experimento II com o uso direto da imagem de frequências conseguiu um resultado de $R^2 = 0.9897$. Ambos modelos precisaram de cerca de doze minutos para a etapa de treinamento, um valor baixo que foi atingido pelo uso de processamento matricial de uma GPU, juntamente com imagens reduzidas à metade, como evidenciado acima.

Os resultados obtidos foram satisfatórios e demonstram que um ambiente de CNN juntamente com o pré-processamento conseguiu predizer o tamanho projetado de partículas com invariância à rotação aplicada em seus padrões de intensidade.

[1] 181 das 361 colunas, portanto a metade mais um.

5.1 Trabalhos Futuros

Esse relatório focou apenas na predição do tamanho projetado das partículas de cristais de gelo atmosféricos proveniente de padrões 2DLS. Como trabalhos futuros, propõe-se uma continuação de estudos nos padrões 2DLS das partículas e características próprias que poderiam ser extraídas para novos conjuntos de treino e teste com a intenção de treinar um modelo para prever outros traços representantes de uma partícula (*e.g. aspect ratio*).

Outra proposta é a combinação do modelo estrutural de sistema definido por Salawu et al. (2017) com as técnicas de FFT e IFFT aplicadas em redes neurais convolucionais desenvolvidas nesse trabalho.

Referências

BARAN, A. A review of the light scattering properties of cirrus. *Journal of Quantitative Spectroscopy & Radiative Transfer - J QUANT SPECTROSC RADIAT*, v. 110, p. 1239–1260, Set 2009.

CHALMERS, M.; JARLETT, H. K. CLOUD experiment sharpens climate predictions. Out 2016. Disponível em: <http://cds.cern.ch/record/2229058>. Acesso em: 06.10.2017.

FRÖHLICH-NOWOISKY, J. et al. Bioaerosols in the earth system: Climate, health, and ecosystem interactions. *Atmospheric Research*, v. 182, n. Supplement C, p. 346 – 376, 2016.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, v. 36, p. 193–202, 1980.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. 800 p. Disponível em: <http://www.deeplearningbook.org>. Acesso em: 10.5.2017.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems 25*. [S.l.]: Curran Associates, Inc., 2012. p. 1097–1105.

LANGLEY RESEARCH CENTER. *Atmospheric Aerosols: What are They and why are They So Important?*. Langley Research Center, 1996. (NASA facts on line). Disponível em: <http://www.nasa.gov/centers/langley/news/factsheets/Aerosols.html>. Acesso em: 07.10.2017.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, p. 436–44, 05 2015.

MCCULLOCH, W. S.; PITTS, W. H. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 115–133, 1943.

NG, A. *Deep Learning Specialization*. Coursera, 2017. Disponível em: <https://www.deeplearning.ai/>. Acesso em: 21.06.2018.

NIELSEN, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015. Disponível em: <http://neuralnetworksanddeeplearning.com/>. Acesso em: 10.5.2017.

PRIORI, D. *Comparison of Neural Network Models Applied to Size Prediction of Atmospheric Particles Based on Their Two-Dimensional Light Scattering Patters*. 98 p. Dissertação (Mestrado) — Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2017.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv*, 2018.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, p. 65–386, 1958.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back propagating errors. *Nature*, v. 323, p. 533–536, 10 1986.

SALAWU, E. O. *Development of Computational Models for Characterizing Small Particles Based on their Two-Dimensional Light Scattering Patterns*. 84 p. Dissertação (Mestrado) — School of Computer Science, University of Hertfordshire, UK, 2015.

SALAWU, E. O. et al. Applying machine learning methods for characterization of hexagonal prisms from their 2d scattering patterns – an investigation using modelled scattering data. *Journal of Quantitative Spectroscopy and Radiative Transfer*, Elsevier Limited, v. 201, p. 115–127, 11 2017. ISSN 0022-4073.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

SOBEL, I. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 1968.

STOPFORD, C. *Ice crystal classification using two dimensional light scattering patterns*. 88 p. Tese (Doutorado) — University of Hertfordshire, UK, 2010.

SZEGEDY, C. et al. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

WU, Y. et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. Disponível em: <http://arxiv.org/abs/1609.08144>.

YANG, P. et al. Scattering and absorption property database for nonspherical ice particles in the near- through far-infrared spectral region. *Appl. Opt.*, v. 44, p. 5512–23, Set 2005.

Apêndices

APÊNDICE A – Código fonte desenvolvido

Código A.1 – tcc_dataNorm

```
%--//--
clear

% Creating destination folders
if (exist('FFTImages/','dir')==0)
    mkdir('FFTImages/TrainSet/');
    mkdir('FFTImages/TestSet/');
end

% Number of rows. 133*n: 133 different orientations for each particle. n is
% the total number of particles.
n = 162;
Row_Number = 133*n;
% Number of columns. 7220 intensity values for each particle + 1 projected
% size value.
Column_Number = 7221;

% Reading the csv file.
M = csvread('DadosIntensidade.csv', 0, 0, [0, 0, Row_Number-1, Column_Number-2]);
Y = csvread('DadosIntensidade.csv', 0, Column_Number-1, [0, Column_Number-1, Row_Number-1, Column_Number-1]);

% Normalisation of Data.
% Natural log.
NLog = log(M);
% Z-Score.
ZScr = zscore(NLog,[],2);
% Standard deviation
[mn,ps1] = mapstd(NLog);
```

```
29
30 % Normalised patters distributed in matrices of 20x361
31 C = cell([133 n]);
32 for i = 1:1:Row_Number
33     %D = reshape(mn(i,:),20,361); % with log and standard
       deviation
34     D = reshape(ZScr(i,:),20,361); % with log and zscore
35     %D = reshape(NLog(i,:),20,361); % with only log
36     %use reshape for this purpose
37     C{i} = D;
38 end
39
40 % Print example of particle matrix
41 %figure();imagesc(C{index}); colormap(gray);
42
43 % Splitting the Dataset.
44 % 70% for training set and 30% for testing set.
45 ind=randperm(Row_Number);
46 Split_Nbr = round(Row_Number*0.7);
47 Train_Set = cell([Split_Nbr 1]);
48 Test_Set = cell([(Row_Number-Split_Nbr) 1]);
49 YTrain = zeros(Split_Nbr, 1);
50 YTest = zeros((Row_Number-Split_Nbr), 1);
51 for i = 1:1:Split_Nbr
52     Train_Set{i} = C{ind(i)};
53     YTrain(i,1) = Y(ind(i));
54 end
55 for i = 1:1:(Row_Number-Split_Nbr)
56     Test_Set{i} = C{ind(Split_Nbr+i)};
57     YTest(i,1) = Y(ind(Split_Nbr+i));
58 end
59
60 % Use of Fast Fourier Transformation for rotation invariance
     on Train_Set
61 for i = 1:1:Split_Nbr
62     Train_Set{i} = fft(Train_Set{i}, [], 2);
63 end
64
65 % Creation of an array of random numbers used for rotation
```

```
66 test_size = Row_Number - Split_Nbr;
67 r = randi([1 360],test_size,1);
68
69 % Use of function circshift(A,K,2) to inflict azimuthal
      rotation on images,
70 % where A is the image matrix, K is degrees of rotation and 2
       to exchange
71 % columns
72 for i = 1:1:test_size
73     Test_Set{i} = circshift(Test_Set{i},r(i),2);
74 end
75
76 % Use of Fast Fourier Transformation for rotation invariance
      on Test_Set
77 for i = 1:1:test_size
78     Test_Set{i} = fft(Test_Set{i}, [], 2);
79 end
80
81 % Storing the FFT image of each particle fot train set
82 %Train_Images = cell([20 361]);
83 for i = 1:1:Split_Nbr
84
85     % Using IFFT images.
86 %     S = ifft(abs(Train_Set{i}), [], 2);
87
88     % Using FFT images.
89     S2 = abs(Train_Set{i}/360);
90     S = S2(1:end, (360/2)+1:end);
91     S(1:end, 2:end-1) = 2*S(1:end, 2:end-1);
92
93     if i < 10000
94         if i < 10
95             s = strcat('0000', num2str(i));
96         else
97             if i < 100
98                 s = strcat('000', num2str(i));
99             else
100                 if i < 1000
101                     s = strcat('00', num2str(i));
```

```
102                 else
103                     s = strcat('0', num2str(i));
104                 end
105             end
106         end
107     else
108         s = num2str(i);
109     end
110     imwrite(mat2gray(S(:,1:181)),['FFTImages/TrainSet/' s '.
        png'])
111 end
112
113 % Storing the FFT image of each particle fot test set
114 %Test_Images = cell([20 361]);
115 for i = 1:1:test_size
116
117     % Using IFFT images.
118 %     S = ifft(abs(Test_Set{i}), [], 2);
119
120     % Using FFT images.
121     S2 = abs(Test_Set{i}/360);
122     S = S2(1:end, (360/2)+1:end);
123     S(1:end, 2:end-1) = 2*S(1:end, 2:end-1);
124
125     if i < 1000
126         if i < 10
127             s = strcat('000', num2str(i));
128         else
129             if i < 100
130                 s = strcat('00', num2str(i));
131             else
132                 s = strcat('0', num2str(i));
133             end
134         end
135     else
136         s = num2str(i);
137     end
138     imwrite(mat2gray(S(:,1:181)),['FFTImages/TestSet/' s '.
        png'])
```

```
139 end
140
141 csvwrite('FFTImages/TrainSizes.csv',YTrain);
142 csvwrite('FFTImages/TestSizes.csv',YTest);
143
144 % ---------------------------------- //
    ----------------------------------
```

Código A.2 – tcc_convnet

```
1  clear
2  testDataPath = fullfile('FFTImages/TestSet/');
3  YTest = csvread('FFTImages/TestSizes.csv');
4  TestData = imageDatastore(testDataPath, 'Labels', YTest);
5
6  trainDataPath = fullfile('FFTImages/TrainSet/');
7  YTrain = csvread('FFTImages/TrainSizes.csv');
8  TrainData = imageDatastore(trainDataPath, 'Labels', YTrain);
9
10 img = readimage(TestData,1);
11 size(img)
12
13 tab = table(TrainData.readall, YTrain);
14
15 % Specifying the network layers.
16 layers = [imageInputLayer([20 181 1])
17
18           convolution2dLayer(7,32, 'Padding','same')
19           reluLayer
20           batchNormalizationLayer
21           maxPooling2dLayer([1 3],'Stride',[1 2])
22
23           convolution2dLayer(5,64, 'Padding','same')
24           reluLayer
25           batchNormalizationLayer
26           maxPooling2dLayer([1 3],'Stride',[1 2])
27
28           convolution2dLayer(3,64, 'Padding','same')
29           reluLayer
30           batchNormalizationLayer
```

```
31          maxPooling2dLayer([1 3],'Stride',[1 2])
32
33 %           dropoutLayer(0.2)
34
35          fullyConnectedLayer(50)
36          reluLayer
37 %           %dropoutLayer(0.2)
38          fullyConnectedLayer(30)
39          reluLayer
40          fullyConnectedLayer(1)
41          reluLayer
42          regressionLayer];
43
44 % Specifying the network options.
45 % Use Graphics Processing Unit(GPU) as execution environment.
46 options = trainingOptions('sgdm','MaxEpochs',100, ...
47     'Momentum',0.9,...
48     'InitialLearnRate',0.0001,...
49     'LearnRateSchedule', 'piecewise',...
50     'LearnRateDropFactor', 0.1,...
51     'LearnRateDropPeriod', 40, ...
52     'ExecutionEnvironment','gpu',...
53     'Shuffle','every-epoch', ...
54     'Plots','training-progress');
55
56 % Train the network.
57 convnet = trainNetwork(tab,layers,options);
58
59 % Test accuracy
60 testImages = table(TestData.readall);
61 predicted = predict(convnet,testImages);
62 trainImages = table(TrainData.readall);
63 predict2 = predict(convnet, trainImages);
64
65
66 mdl = fitlm(YTest, predicted);
67 r2 = mdl.Rsquared.Ordinary
68 r2Adj = mdl.Rsquared.Adjusted
69
```

```
70
71 figure();plotregression(YTrain, predict2);
72 figure();plotregression(YTest, predicted);
```

APÊNDICE B – Artigo SBC

Predição de Características de Partículas Atmosféricas Utilizando Redes Neurais Convolucionais

Vinícius C. Biermann[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)

vinicius.couto.biermann@grad.ufsc.br

Abstract. *Climate models are affected by the concentration and characteristics of atmospheric particles such as ice crystals, present in clouds. Particle data can be recorded as two-dimensional light scattering patterns from which we can extract useful information, such as size and shape. With the intention of predicting the projected size of these particles, the implementation of a Deep Learning architecture environment with convolutional neural networks for regression was made. The system used fast Fourier transforms to be invariant to possible rotations in the patterns. The results obtained were satisfactory, with a coefficient of determination* $R^2 = 0.9901$. *These results were also compared with results obtained from a previous work that used the same dataset on different machine learning techniques.*

Resumo. *Modelos climáticos são afetados pela concentração e características de partículas atmosféricas como os cristais de gelo, presentes em nuvens. Dados de partículas podem ser armazenados como padrõesbidimensionais de dispersão de luz, dos quais podemos extrair informações úteis, como tamanho e formato. Com a intenção de prever o tamanho projetado de tais partículas, a implementação de um ambiente de arquitetura Deep Learning com redes neurais convolucionais para regressão foi feita. O sistema utilizou de transformadas rápidas de Fourier para ser invariante à possíveis rotações nos padrões. Os resultados obtidos foram satisfatórios, com coeficiente de determinação* $R^2 = 0.9901$. *Também foi feita uma comparação com os resultados já obtidos em um trabalho anterior que usou o mesmo conjunto de dados sobre diferentes técnicas de aprendizado de máquina.*

1. Introdução

O aerossol é definido como um conjunto de pequenas partículas sólidas ou líquidas que estão suspensas em um gás. Tais partículas se originam de produção humana ou natural, podendo ser orgânicas ou inorgânicas. Alguns exemplos são: pólen; esporos; bactérias [Fröhlich-Nowoisky et al. 2016]; poeira vulcânica; produtos de combustão [1]; etc.

Diversos projetos visam uma melhor compreensão de aerossóis em nuvens e os impactos climáticos causados por ambos, como o estudo das relações entre aerossóis e raios cósmicos feito pelo experimento CLOUD (Cosmics Leaving Outdoor Droplets)[2],

[1]https://www.nasa.gov/centers/langley/news/factsheets/Aerosols.html
[2]Site oficial do experimento CLOUD: https://home.cern/about/experiments/cloud

da CERN [3]. Com os novos dados obtidos, o entendimento sobre a formação de novas partículas na troposfera permite uma melhor precisão na projeção do aumento de temperatura na Terra [Chalmers and Jarlett 2016].

Um tipo abundante de partícula são os cristais de gelo, que se formam nas nuvens do tipo *cirrus*, presentes na troposfera em altitudes entre 5 e 14km. Essas nuvens cobrem cerca de 30% da superfície terrestre e são parte do efeito estufa natural, pois fazem um balanço de radiação através da reflexão e absorção de raios solares, que intensificam ou diminuem de acordo com as características físicas das partículas de gelo, como tamanho e formato [Baran 2009].

2. Conjunto de dados

O conjunto de dados cedido pela Universidade de *Hertfordshire* contém dados de partículas prismáticas hexagonais, geradas por um simulador, armazenados em arquivos no formato `HDF5` (*Hierarchical Data Format 5*) com informações de 162 partículas atmosféricas, cada uma com 133 orientações. Essas orientações são os três ângulos de Euler denominados α, β e γ, onde representam, respectivamente, inclinações de um corpo que rotaciona em relação aos eixos z, x e y. Os dados coletados a partir das orientações estão armazenados como padrões bidimensionais de dispersão de luz (2DLS, *Two-Dimensional Light Scattering*).

Para a coleta de tais dados, uma sonda como a SID-2 é utilizada. A sonda possui um canhão de *laser* e, quando uma partícula entra em contato com o *laser*, sensores são ativados e capturam a difração causada na luz. A Figura 1 mostra o trajeto de uma partícula com representação da luz do *laser* e detecção dos sensores.

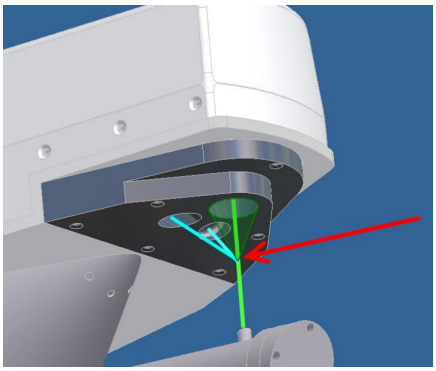


Figura 1. SID-2.

Um padrão 2DLS é associado a cada orientação de uma partícula e a intensidade desses padrões de difração, declarada como o atributo *intensity*, é indexada por outros dois atributos: *elevation* e *azimuth*. Elevação (ou altitude) e azimute são ângulos pertencentes ao sistema horizontal de coordenadas, que utiliza o horizonte do observador como um plano que divide uma esfera em dois hemisférios.

Elevation é dado como o ângulo formado pelo feixe de luz que incide na partícula e foram considerados relevantes apenas aqueles com valores entre $6°$ e $25°$, enquanto

[3] *Organisation Européenne pour la Recherche Nucléaire*

em *azimuth*, todos os ângulos foram utilizados — *i.e.* de $0°$ a $360°$ [Salawu 2015]. Assim, temos um total de 7.220 valores de intensidade obtidos pela multiplicação das vinte medições de elevação com as 361 de azimute, ou seja, de $((25 - 6) + 1) \times 361$.

A Figura 2 apresenta exemplos de padrões de difração de luz de três partículas com tamanhos diferentes utilizando uma representação em coordenadas polares.

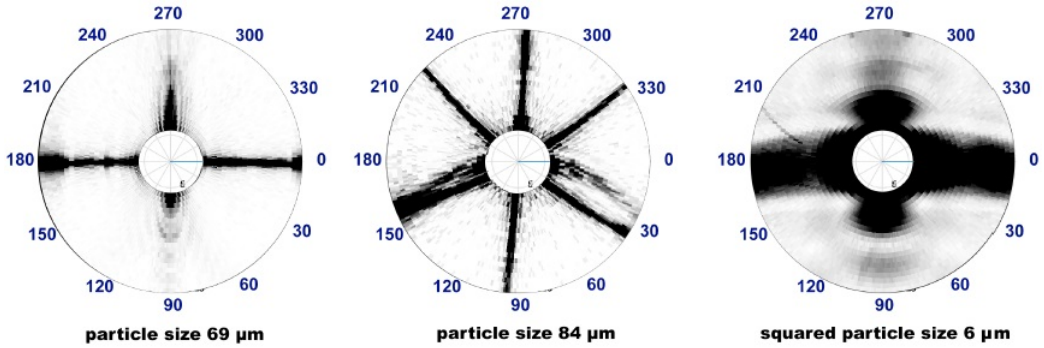


Figura 2. Exemplos de padrões de difração.

Uma matriz pode ser gerada para cada orientação, utilizando elevação como linhas e azimute como colunas, gerando uma matriz de 20 linhas por 361 colunas.

Como em um cenário real a captura dos padrões das partículas não ocorrerá sempre da mesma forma, isto é, ela pode estar rotacionada, simulações de possíveis rotações tiveram que ser aplicadas às imagens. Isso foi feito utilizando um *shift* circular nas colunas da imagem. A figura 3 mostra um exemplo de rotação de 100 graus em um padrão, realizando um *shift* de 100 posições para a direita.

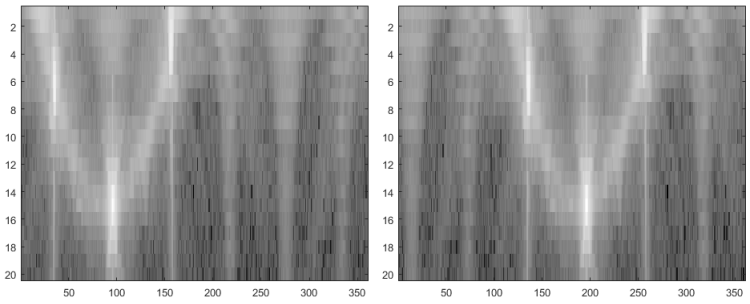


Figura 3. Exemplo de rotação em um padrão 2DLS.

3. Invariância à rotação

Foi necessário não apenas simular a rotação no padrão, como também fazer com que a rede neural desenvolvida fosse invariante à rotações — *i.e.* uma partícula deve possuir o mesmo tamanho, mesmo estando rotacionada.

3.1. *Fast Fourier Transform*

O método de Transformadas Rápidas de Fourier (FFT) foi utilizado por [Priori 2017] para garantir invariância à rotação no mesmo conjunto de dados. Com FFT, saímos do domínio

espacial de uma imagem e trabalhamos apenas com frequências. A ideia é que temos as mesmas frequências em uma imagem, não importando suas posições.

O FFT é aplicado em cada linha da matriz, gerando resultados muito similares tanto no padrão original como no rotacionado. A figura 4 apresenta, em ordem, as imagens resultantes após a aplicação da FFT no padrão original da figura 3, no padrão rotacionado e uma diferença entre os dois resultados. Os valores extremos da matriz de diferença são os seguintes:

- Máximo: 5.6843×10^{-14}
- Mínimo: -5.6843×10^{-14}

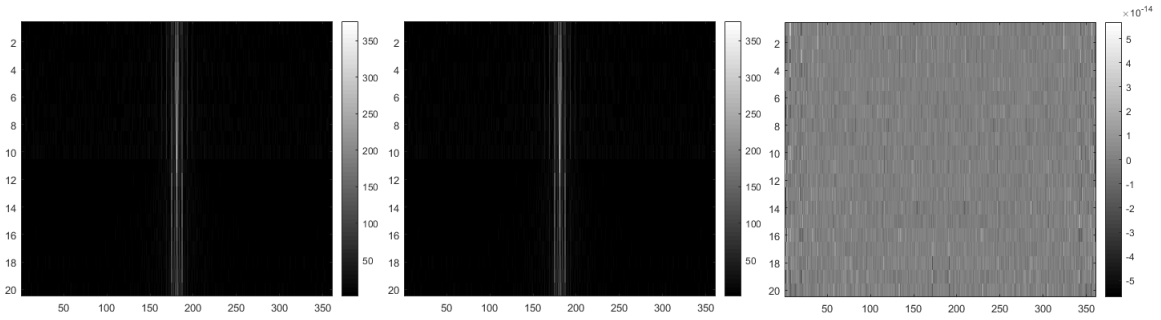


Figura 4. Exemplo de FFT em um padrão de uma partícula.

Notamos que os valores máximos e mínimos da diferença são muito pequenos, atingindo uma precisão adequada para os modelos, e podemos considerar as duas imagens como iguais. Ou seja, temos o mesmo resultado para um padrão com e sem rotação.

3.2. *Inverse Fast Fourier Transform*

Aplicamos a função de Transformada Inversa Rápida de Fourier (IFFT) para sair do domínio de frequências e voltar ao domínio espacial. Para isso, não pode-se simplesmente aplicar diretamente a função IFFT no resultado obtido via FFT, pois as imagens retornariam para os padrões 2DLS originais, perdendo a invariância à rotação fornecida pelo uso da função FFT. Aplicando o IFFT nos valores absolutos, ou de magnitude, gera-se imagens que são mais próximas do que seria um padrão normal de difração, porém ainda mantendo a invariância à rotação. Por esses resultados com as imagens, surgiu a ideia de realizar um experimento com IFFT, além do com apenas FFT, para observar se o uso de IFFT trará algum impacto positivo nos resultados.

Assim como no FFT, o IFFT também é aplicado em cada linha da matriz. A figura 5 apresenta, em ordem, as imagens resultantes após a aplicação da IFFT nas primeiras duas imagens da figura 4. A terceira imagem da figura, novamente, é uma diferença das duas à sua esquerda. Os valores extremos da matriz de diferença são os seguintes:

- Máximo: 1.7764×10^{-15}
- Mínimo: -1.7764×10^{-15}

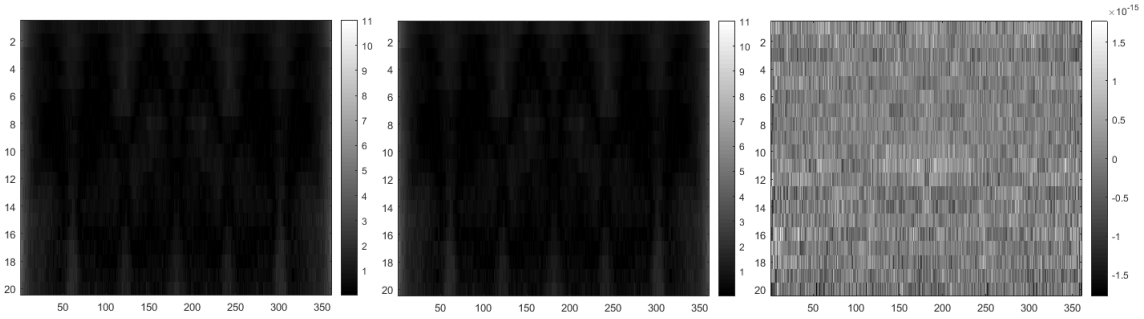


Figura 5. Exemplo de IFFT em um padrão de uma partícula.

3.3. Simetria

O resultado das operações de FFT e IFFT é simétrico em relação ao eixo y. Isso vem da natureza da Transformada Discreta de Fourier e suas propriedades, pois estamos aplicando a operação em cima de vetores com valores reais (*i.e.* com parte imaginária zerada), gerando uma função par (simétrica em relação ao eixo y) como sinal de magnitude e uma ímpar (simétrica em relação à origem) para o sinal de fase.

Realmente, nota-se que esse espelhamento está presente nas imagens de ambas as Figuras 4 e 5. Portanto, foram utilizadas para treinamento e teste da rede neural convolucional desenvolvida apenas metade de cada imagem, com isso reduzindo em cerca de 50% o tempo de treino da rede.

4. Experimentos

Dois experimentos utilizando redes neurais convolucionais foram definidos para prever o tamanho projetado de partículas. No primeiro, foi aplicado a sequência de FFT e IFFT nos dados de padrões 2DLS, enquanto o segundo se deu apenas com o uso de FFT. Nos dois experimentos, os dados passam por um processo de normalização, aplicação de funções, operações de salvamento e carregamento e treinamento e teste da rede. Pode-se dividir em dois estágios, descritas a seguir.

No primeiro estágio, as partículas inicialmente se encontram em uma mesma matriz, onde cada linha representa uma orientação, portanto, para 162 partículas temos 21.546 linhas por 7.220 colunas de intensidade. Um vetor possui o tamanho projetado de cada orientação, na mesma ordem. A matriz passa por um processo de normalização em duas etapas. A primeira de logaritmo natural serve para reduzir o intervalo de características para uma escala mais razoável, o que melhora o desempenho de uma rede neural. A segunda, de *z-score*, tem a função de deslocar a média para zero e deixar um desvio padrão de um.

O padrão é então distribuído em matrizes de 20 por 361, como explicado anteriormente. Uma permutação aleatória é feita para dividir o conjunto de orientações e respectivos tamanhos projetados em 70% dados de treino e 30% dados de teste. Apenas os dados do conjunto de teste sofrerão rotação.

À seguir, garantimos a invariância à rotação do sistema com a aplicação de FFT. IFFT é aplicado e as matrizes resultantes são transformadas em imagens em escala de cinza e salvas juntamente com um vetor de tamanhos em um diretório criado. Como dito anteriormente, apenas metade das imagens serão salvas.

O segundo estágio começa com o carregamento dos dados para treino e teste da rede. A rede é treinada e seu resultado passa por um teste utilizando os dados de padrões rotacionados. Um *output* é gerado com a predição dos tamanhos projetados de cada orientação presente no conjunto de testes. A Figura 6 mostra um esquemático da arquitetura da rede.

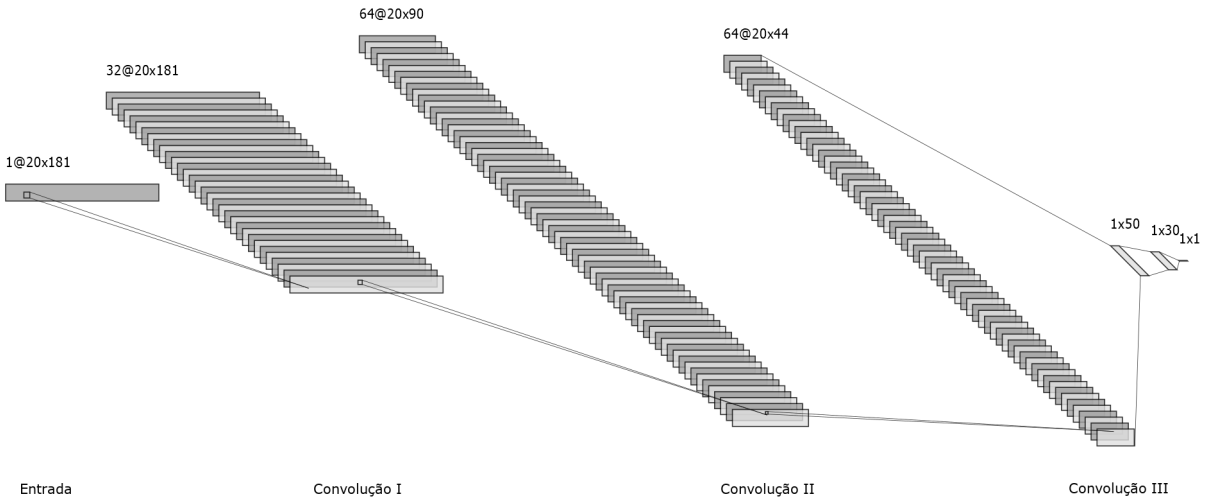


Figura 6. Diagrama da rede.

A arquitetura da rede convolucional desenvolvida conta com uma camada de entrada para imagens de tamanho 20 por 181 por 1, onde o 1 significa que a imagem está em escala de cinza. Foram utilizadas três camadas convolucionais, com filtros 7×7, 5×5 e 3×3. Todas foram seguidas por uma função de ativação *ReLU*, normalização e *max pooling* com região 1×3 e *stride* 1×2. Três camadas totalmente conectadas com 50, 30 e 1 neurônios darão o único valor final que representa o tamanho predito da partícula. Uma camada de saída de regressão calcula a perda da rede. Foram definidas 100 épocas com taxa de aprendizado inicial de 1^{-4} e decaimento de 0.1 a cada 40 épocas.

4.1. Experimento I

O primeiro modelo de CNN desenvolvido utiliza as imagens em escala de cinza obtidas a partir das matrizes de IFFT. Durante a fase de treinamento alcançou-se um coeficiente de determinação de $R^2 = 0.9966$. A Figura 7 apresenta a regressão linear com o conjunto de partículas de teste.

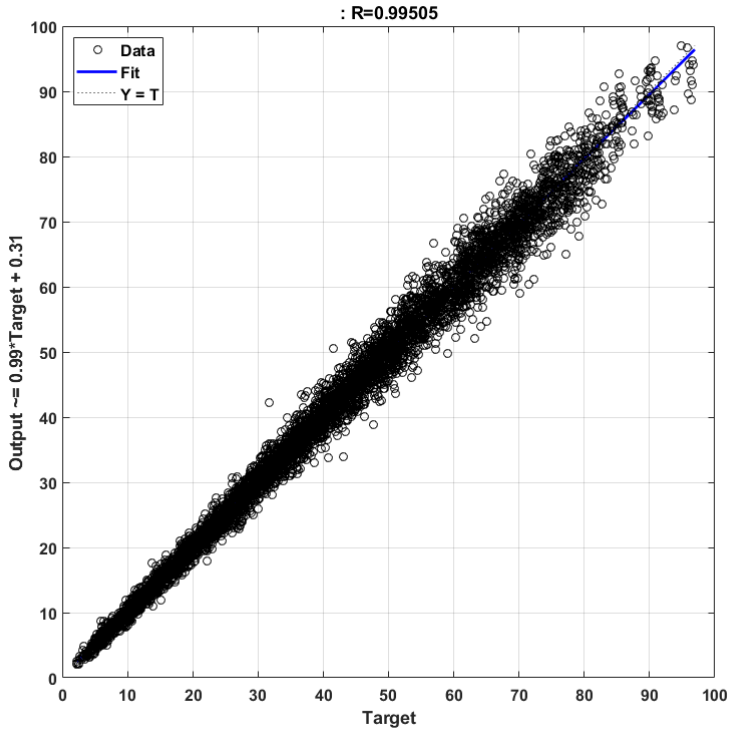


Figura 7. Resultado de teste no experimento I.

Podemos notar que os resultados são satisfatórios, principalmente para partículas de menor tamanho e mesmo havendo um espalhamento maior para partículas maiores, ainda temos predições aceitáveis. O modelo utilizando imagens a partir do IFFT conseguiu um coeficiente de correlação $R = 0.99505$ e coeficiente de determinação $R^2 = 0.9901$.

4.2. Experimento II

O segundo modelo de CNN desenvolvido utiliza as imagens geradas a partir apenas da função FFT, sem a aplicação de sua inversa. Durante a fase de treinamento alcançou-se um coeficiente de determinação de $R^2 = 0.9945$. A Figura 8 apresenta a regressão linear com o conjunto de partículas de teste.

Nota-se que os resultados, apesar de inferiores aos do experimento I, ainda conseguiram atingir um nível satisfatório de qualidade, com $R = 0.99483$ e $R^2 = 0.9897$. Porém, nota-se que a rede teve uma dificuldade maior para prever tamanhos de pequenas partículas, enquanto o espalhamento de dados em tamanhos maiores está mais similar com os do experimento I.

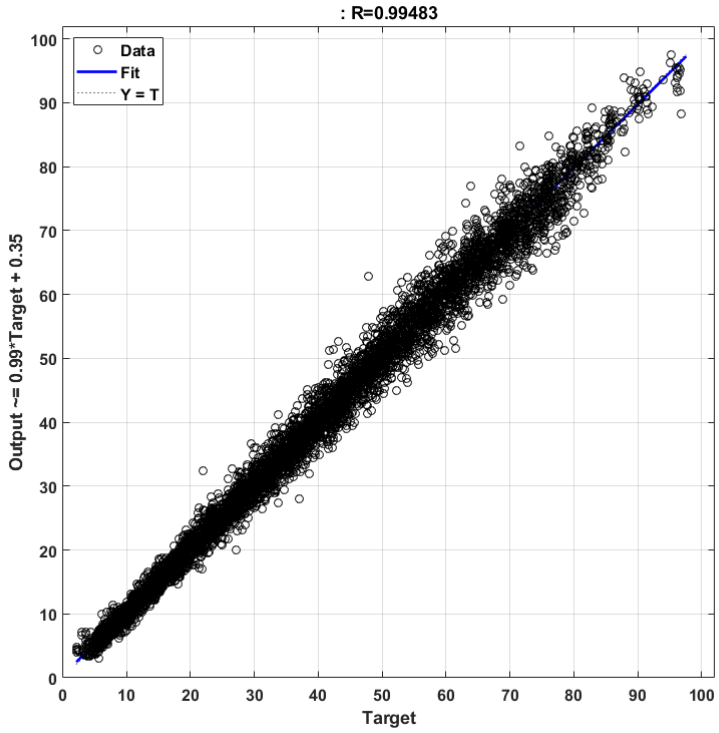


Figura 8. Resultado de teste no experimento II.

5. Comparação de Resultados

A Tabela 1 compara os resultados do modelo 4 (*Autoencoders*) de [Priori 2017], que atingiu a melhor média de performance, com os obtidos nos experimentos I e II. Foram escolhidas todas as quarenta e nove partículas de diferentes tamanhos projetados utilizadas por Priori para comparar seus modelos para testar o comportamento do sistema com apenas as 133 orientações de cada partícula. O desempenho dos modelos foi dado pelo seu coeficiente de determinação (R^2). Na primeira coluna da tabela, têm-se que P é o número da partícula, C é seu comprimento e D o diâmetro.

Tabela 1. Comparação de resultados.

Partícula	Performance (R^2)		
	Priori	Experimento I	Experimento II
P:1; C:1.1; D:9.1	**0.956**	0.857	0.631
P:2; C:1.5; D:8.5	**0.959**	0.908	0.674
P:5; C:10.3; D:82.6	**0.987**	0.977	0.978
P:7; C:102.5; D:34.2	0.942	**0.953**	0.948
P:8; C:105.7; D:19.2	0.958	**0.964**	0.950
P:10; C:108.9; D:36.3	0.944	**0.946**	0.940
P:13; C:11.7; D:93.6	**0.981**	**0.981**	0.979
P:16; C:115.3; D:21.0	0.957	**0.972**	0.969
P:17; C:115.6; D:38.5	0.887	0.956	**0.957**
Continua na próxima página			

Tabela 1 – Continuação da página anterior

Partícula	Performance (R^2)		
	Priori	Experimento I	Experimento II
P:18; C:118.0; D:14.7	0.959	**0.974**	0.957
P:20; C:12.3; D:98.5	0.964	0.980	**0.983**
P:26; C:13.0; D:2.4	**0.976**	0.857	0.736
P:27; C:13.2; D:105.5	**0.987**	0.986	0.979
P:31; C:14.0; D:28.1	**0.805**	0.759	0.644
P:34; C:143.2; D:26.0	0.944	**0.979**	0.968
P:37; C:15.7; D:2.0	**0.974**	0.829	0.658
P:38; C:15.8; D:15.8	**0.896**	0.719	0.593
P:39; C:15.8; D:5.3	**0.969**	0.876	0.714
P:44; C:16.6; D:49.9	0.884	**0.917**	0.907
P:66; C:22.6; D:45.2	0.776	**0.794**	0.759
P:68; C:23.0; D:69.0	0.916	0.916	**0.934**
P:70; C:24.5; D:73.5	0.927	0.931	**0.933**
P:71; C:24.7; D:49.4	0.746	**0.778**	0.775
P:75; C:26.9; D:3.4	**0.989**	0.946	0.863
P:77; C:28.3; D:14.2	**0.957**	0.901	0.759
P:79; C:29.7; D:9.7	**0.970**	0.948	0.870
P:83; C:3.5; D:19.2	**0.974**	0.927	0.881
P:85; C:3.9; D:31.6	**0.987**	0.959	0.951
P:87; C:31.6; D:5.7	**0.981**	0.934	0.928
P:98; C:39.7; D:39.7	0.761	**0.860**	0.794
P:99; C:4.1; D:12.3	**0.932**	0.786	0.631
P:101; C:4.6; D:37.2	**0.981**	0.974	0.966
P:104; C:43.2; D:43.2	**0.830**	0.827	0.744
P:106; C:46.5; D:46.5	0.749	**0.851**	0.794
P:112; C:5.3; D:5.3	**0.881**	0.368	0.474
P:114; C:5.6; D:11.1	**0.882**	0.528	0.515
P:115; C:5.6; D:16.9	**0.926**	0.860	0.732
P:118; C:54.6; D:27.3	0.874	**0.927**	0.897
P:119; C:55.8; D:18.6	**0.954**	0.952	0.905
P:121; C:59.3; D:10.8	**0.986**	0.966	0.941
P:124; C:6.7; D:53.9	0.969	0.973	**0.974**
P:131; C:68.5; D:12.5	0.964	**0.967**	0.946
P:134; C:7.4; D:40.5	**0.980**	0.938	0.938
P:146; C:8.9; D:26.6	**0.964**	0.880	0.804
P:147; C:8.9; D:8.9	**0.776**	0.579	0.288
P:148; C:81.1; D:40.5	0.877	**0.950**	0.930
P:150; C:83.6; D:10.4	**0.968**	0.953	0.943
P:159; C:94.9; D:11.9	**0.982**	0.977	0.948
P:160; C:95.6; D:31.9	0.923	**0.963**	0.952
Média de performance	0.924	0.888	0.835

Nota-se que o experimento I atingiu resultados superiores ao experimento II em quase todas as partículas presentes na tabela, exibindo uma média nos valores de coeficiente de determinação do conjunto de amostra de 0.888, enquanto o experimento II alcançou média de 0.835. A média obtida no experimento I também está mais próxima do resultado de Priori, que adquiriu 0.924.

Os melhores resultados para cada partícula podem ser vistos em negrito. Em geral, o modelo de Priori conseguiu melhores resultados, sendo superiores aos outros dois em vinte e oito casos e com mesmo valor em um caso com o experimento I. Em quinze casos, o experimento I atingiu resultados melhores. O experimento II foi superior em cinco casos. As partículas 112, 114 e 147 foram as que tiveram piores predições, sendo inferiores ao modelo de Priori por uma grande margem.

A Figura 9 apresenta a regressão linear com duas partículas diferentes utilizando a rede treinada no experimento I. À esquerda, pode-se ver o resultado de uma partícula não quadrada, atingindo $R^2 = 0.981$, enquanto à direita têm-se uma partícula quadrada com um resultado inferior de $R^2 = 0.579$.

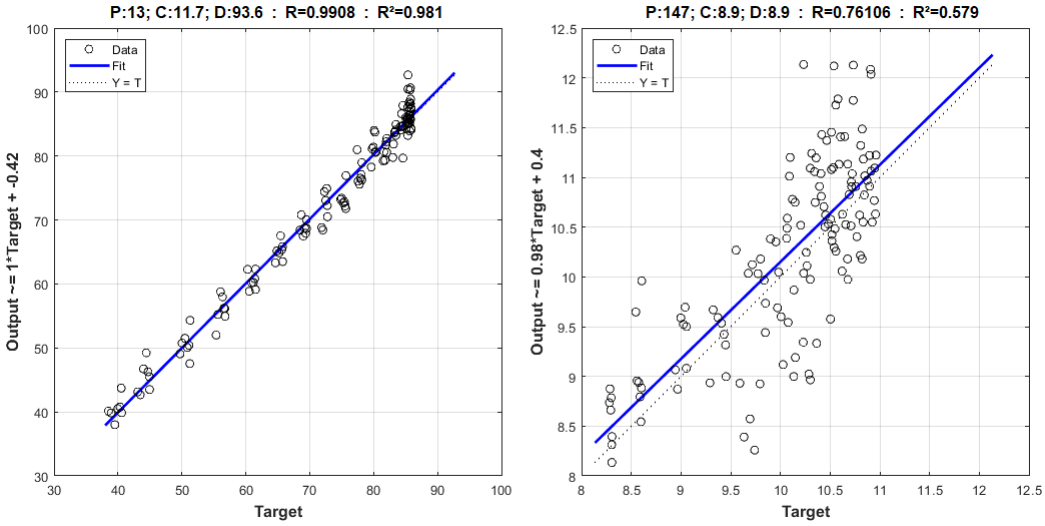


Figura 9. Resultados individuais de duas partículas com o Experimento I.

Em geral, nos dois experimentos, e em todos os modelos de Priori, as partículas quadradas — *i.e.* aquelas com mesmo valor de comprimento e diâmetro, obtiveram predições piores, alcançando valores muito baixos no coeficiente de determinação. De acordo com [Priori 2017], resultados ruins provavelmente se dão pela similaridade encontrada em padrões de partículas de tamanhos diferentes, mas que possuem mesmos valores de *aspect ratio* e orientação. Esse comportamento fica evidente nos resultados amostrados na Tabela 1, pode-se notar que em geral partículas menores, e especialmente as quadradas, atingiram resultados inferiores às maiores, enquanto as maiores atingiram resultados satisfatórios (*e.g.* a partícula quadrada 98).

6. Conclusão

Nesse trabalho foram realizados dois experimentos utilizando ambientes de redes neurais convolucionais (CNN) para predizer o tamanho projetado de um grupo com simulações de

pequenas partículas cristalinas hexagonais presentes na atmosfera da Terra. O conjunto de dados cedido pela universidade de *Hertfordshire* conta com simulações dos padrões 2DLS de 162 partículas, cada uma com 133 orientações diferentes.

Como os dados de partículas reais coletadas podem estar rotacionados, o código desenvolvido precisou ser invariante a tais rotações. Para tanto, foi empregado o uso de Transformadas Rápidas de Fourier (FFT) durante o pré-processamento, levando as imagens do domínio espacial para o domínio de frequências. Assim, como demonstrado na seção 3.1, os padrões normal e rotacionado de uma partícula resultam em matrizes com diferença de valor na ordem de $\times 10^{-14}$.

O uso da função inversa da FFT, denominado IFFT, foi uma proposta inicial para o treino da rede, pois uma imagem gerada a partir do IFFT está no domínio espacial e tem características que lembram mais as imagens originais dos padrões se comparado com aquelas geradas apenas pela função FFT. Após a aplicação de IFFT nos resultados de padrão original e rotacionado, a diferença de valor entre os dois resultados ficou na ordem de $\times 10^{-15}$, possibilitando também o uso de IFFT para garantir a invariância à rotação no sistema.

Foi observado que devido à natureza da FFT (e IFFT), garantiu-se que as matrizes geradas a partir dos valores reais de intensidade tivessem a propriedade de uma função par, possuindo uma simetria em relação ao eixo y. Como o espelhamento faz com que os padrões da primeira metade da imagem apenas se repitam na segunda, optou-se por utilizar apenas uma parte [4].

Os dois experimentos elaborados visaram treinar e testar uma rede CNN com e sem a aplicação da função de IFFT após a de FFT. O experimento I, com IFFT, atingiu um coeficiente de determinação $R^2 = 0.9901$ durante a fase de testes, enquanto o experimento II com o uso direto da imagem de frequências conseguiu um resultado de $R^2 = 0.9897$. Ambos modelos precisaram de cerca de doze minutos para a etapa de treinamento, um valor baixo que foi atingido pelo uso de processamento matricial de uma GPU, juntamente com imagens reduzidas à metade, como evidenciado acima.

Os resultados obtidos foram satisfatórios e demonstram que um ambiente de CNN juntamente com o pré-processamento conseguiu predizer o tamanho projetado de partículas com invariância à rotação aplicada em seus padrões de intensidade.

6.1. Trabalhos futuros

Esse relatório focou apenas na predição do tamanho projetado das partículas de cristais de gelo atmosféricos proveniente de padrões 2DLS. Como trabalhos futuros, propõe-se uma continuação de estudos nos padrões 2DLS das partículas e características próprias que poderiam ser extraídas para novos conjuntos de treino e teste com a intenção de treinar um modelo para prever outros traços representantes de uma partícula (*e.g. aspect ratio*).

Outra proposta é a combinação do modelo estrutural de sistema definido por [Salawu et al. 2017] com as técnicas de FFT e IFFT aplicadas em redes neurais convolucionais desenvolvidas nesse trabalho.

[4] 181 das 361 colunas, portanto a metade mais um.

Referências

Baran, A. (2009). A review of the light scattering properties of cirrus. *Journal of Quantitative Spectroscopy & Radiative Transfer - J QUANT SPECTROSC RADIAT*, 110:1239–1260.

Chalmers, M. and Jarlett, H. K. (2016). CLOUD experiment sharpens climate predictions.

Fröhlich-Nowoisky, J., Kampf, C. J., Weber, B., Huffman, J. A., Pöhlker, C., Andreae, M. O., Lang-Yona, N., Burrows, S. M., Gunthe, S. S., Elbert, W., Su, H., Hoor, P., Thines, E., Hoffmann, T., Després, V. R., and Pöschl, U. (2016). Bioaerosols in the earth system: Climate, health, and ecosystem interactions. *Atmospheric Research*, 182(Supplement C):346 – 376.

Priori, D. (2017). Comparison of neural network models applied to size prediction of atmospheric particles based on their two-dimensional light scattering patters. Master's thesis, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis.

Salawu, E. O. (2015). Development of computational models for characterizing small particles based on their two-dimensional light scattering patterns. Master's thesis, School of Computer Science, University of Hertfordshire, UK.

Salawu, E. O., Hesse, E., Stopford, C., Davey, N., and Sun, Y. (2017). Applying machine learning methods for characterization of hexagonal prisms from their 2d scattering patterns – an investigation using modelled scattering data. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 201:115–127.