UNIVERSIDADE FEDERAL DE SANTA CATARINA
FLORIANÓPOLIS

Lorenzo Freitas da Cunha Varaschin

PREVISÃO DA IRRADIÂNCIA GLOBAL HORIZONTAL DE CURTO PRAZO COM MODELOS DE DEEP LEARNING

FLORIANÓPOLIS
2024

LORENZO FREITAS DA CUNHA VARASCHIN

PREVISÃO DA IRRADIÂNCIA GLOBAL HORIZONTAL DE CURTO PRAZO COM MODELOS DE DEEP LEARNING

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina, como requisito necessário para obtenção do grau de Bacharel em Engenharia Elétrica

Florianópolis, 12 julho de 2024

LORENZO FREITAS DA CUNHA VARASCHIN

Previsão da Irradiância Global Horizontal de Curto Prazo com Modelos de Deep Learning

Essa Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Engenharia Elétrica e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Elétrica.

Documento assinado digitalmente
Miguel Moreto
Data: 05/08/2024 09:14:48-0300
CPF: ***.850.100-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Miguel Moreto, Dr.
Coordenador do Curso
Universidade Federal de Santa Catarina

Banca Examinadora:

Documento assinado digitalmente
Danilo Silva
Data: 30/07/2024 07:44:33-0300
CPF: ***.557.924-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Danilo Silva, Dr.
Orientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Eduardo Luiz Ortiz Batista
Data: 07/08/2024 10:47:24-0300
CPF: ***.521.889-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Luiz Ortiz Batista, Dr.
Avaliador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Mauro Augusto da Rosa
Data: 25/07/2024 19:29:00-0300
CPF: ***.652.400-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Mauro Augusto Da Rosa, Dr.
Avaliador
Universidade Federal de Santa Catarina

Florianópolis, 12 de julho de 2024

Agradecimentos

Primeiramente gostaria de agradecer todos os amigos e colegas pelos momentos de alegria e felicidade que compartilharam comigo ao longo dessa jornada.

Também gostaria de agradeçer a todos os professores que tive na UFSC, que proporcionaram ensinamentos fundamentais para minha vida profissional e pessoal através de seus esforços e sua dedicação. Um agradecimento especial ao Prof. Danilo Silva, que sempre esteve disposto para tirar minhas dúvidas e abriu portas para mim na realização de estágio e orientação deste trabalho.

Por último, gostaria de agradecer à minha família: pais, irmão, primos e primas, tios e tias, avô e avó, que estiveram comigo desde sempre e proporcionaram experiências e aprendizados que ficarão para sempre. Em especial gostaria de agradecer àqueles que sempre estiveram e estarão ao meu lado, meus pais, Vitorio e Márcia.

Resumo

A utilização de fontes renováveis e sustentáveis de energia na rede elétrica está se tornando cada vez mais importante para enfrentar a atual crise climática. Dentre elas, a energia solar, em particular a energia solar fotovoltaica, se destaca por ser a mais barata, robusta e abundante ao redor do mundo. Contudo, a alta integração de uma fonte não despachável e incontrolável representa um desafio para os operadores do sistema elétrico, pois aumenta o grau de incerteza associado a suas tomadas de decisão. Uma possível solução para esse problema seria a previsão precisa da Irradiância Global Horizontal (GHI) de curto prazo, que forneceria aos operadores a informação da disponibilidade do recurso solar com minutos de antecedência, permitindo-lhes tempo para agir de acordo. Portanto, o objetivo deste trabalho consiste em treinar arquiteturas de aprendizado profundo, como a ResNet18 a as redes Long Short-Term Memory, para gerar previsões da GHI atual (*nowcasting*) e 5 minutos no futuro (*forecasting*) com o uso apenas de imagens do céu como dado de entrada. O conjunto de dados utilizado para o treinamento foi o de Folsom, CA [Pedro, Larson e Coimbra 2019], que contêm mais de um milhão de medidas de GHI e aroximadamente 800,000 imagens do céu correspondentes. Nos experimentos de *nowcasting*, o uso do índice de céu claro k_t como a variável alvo dos modelos gerou melhores resultados do que a GHI, ao mesmo tempo que mostrou-se mais robusto às diferenças de distribuição sinteticamente aplicadas no conjuto de teste. O melhor modelo de *nowcasting*, que foi usado tanto para o *fine tuning* quanto para a extração de *embeddings* nos experimentos de *forecasting*, alcançou um RMSE de 35,68 W/m^2 no conjunto de teste. No primeiro experimento de *forecasting*, os *embeddings* extraídos do *nowcasting* foram usados para treinar uma rede LSTM+MLP, enquanto que no segundo experimento é realizado um *fine tuning* no modelo de *nowcasting* e ele é treinado simultaneamente com a rede LSTM+MLP. Ambos os experimentos alcançaram resultados compatíveis com o atual estado da arte, com *forecasting skills* de 10,31% e 9,81%, respectivamente.

Palavras-chave:Previsão de Irradiância. Aprendizado Profundo. Visão Computacional. Regressão.

Abstract

The use of renewable and sustainable energy sources in the electrical grid is becoming increasingly important in order to address the world's ongoing climate crisis. Among these, solar energy, in particular solar photovoltaic energy, stands out as being the cheapest, most robust and abundant across the globe. However, the high integration of such a volatile and uncontrollable energy source poses a challenge to the system operators, as it increases the uncertainty in their decision making. One possible solution to this problem comes in the form of accurate short-term Global Horizontal Irradiance (GHI) forecasts, providing the grid operators the information about the availability of solar power generation several minutes ahead, which allows them time to act accordingly. Thus, the goal of this work is to train deep learning architectures, such as the popular ResNet18 and Long Short-Term Memory Networks, to provide GHI nowcasts and 5 minutes ahead forecasts based solely on the use of sky images as input data. The dataset used for training was the Folsom, CA dataset [Pedro, Larson e Coimbra 2019], which contains over one million GHI measurements, as well as approximately 800,000 corresponding sky images. In the nowcasting experiments, the use of the clear sky index k_t as the model's target variable yielded better results than the GHI, while also proving to be more robust to the synthetic distribution shift applied to the test set. The best nowcasting model, which was used for both fine tuning and embedding extraction in the forecasting task, achieved an RMSE of 35,68 W/m^2 on the test set. In the first forecasting task, the extracted nowcasting embeddings were used to train a LSTM+MLP network, while in the second task the nowcasting model is fine tuned and trained simultaneously with the LSTM+MLP network. Both experiments achieved state of the art results, with forecasting skills of 10,31% and 9,81%, respectively.

Keywords:Irradiance Forecasting. Deep Learning. Computer Vision. Regression.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Com o crescimento populacional e o desenvolvimento econômico, surge o aumento na demanda de energia elétrica. Até o século passado, esse aumento era suprido majoritariamente pela queima de combustíveis fosséis, como carvão, pétroleo e gás natural, o que emitiu quantidades desproporcionais de CO_2 na atmosfera e tornou o impacto do ser humano no meio ambiente preocupante. Uma das principais consequências desse impacto é a intensificação do efeito estufa e o aquecimento global.

A fim de se evitar uma catástrofe ambiental, a sociedade tem realizado diversos esforços internacionais nos últimos anos para promover o desenvolvimento sustentável, tentando melhorar a qualidade de vida da população atual sem comprometer a das gerações futuras. Em 2016 diversos países assinaram o acordo de Paris, cuja meta é reduzir drasticamente as emissões de gases poluentes até 2030 de forma que, ao final do século, a temperatura média do planeta esteja no máximo 1,5 °C acima do que era antes da Revolução Industrial [Masson-Delmotte et al. 2022]. Do ponto de vista de engenharia, isso significa que os combustíveis fósseis devem ceder cada vez mais espaço para fontes renováveis e sustentáveis de energia no abastecimento da demanda.

Dentro desse contexto, a energia solar, em particular a solar fotovoltaica, surge como uma alternativa promissora, uma vez que esse tipo de energia:

- É sustentável e renovável;
- Vem apresentando um declínio acentuado no seu preço de implementação e operação, o que atrai investidores e torna dessa fonte uma forte competidora nos leilões de energia [Pereira e Ruther 2021];
- Pode ser instalada próxima aos imóveis comerciais e residenciais, diminuindo as perdas de transmissão e distribuição;
- Tem sua disponibilidade energética coincidente com o horário comercial, onde se concentra a maior parte da demanda;
- Possui um enorme potencial de geração ao redor do mundo.

No Brasil a média anual desse potencial pode chegar até 6,25 kWh/m^2.dia em áreas da região Nordeste [Pereira et al. 2017]. Impulsionada por esse potencial, a capacidade de energia solar instalada no Sistema Interligado Nacional (SIN) cresce exponencialmente. Segundo dados do Operador Nacional do Sistema Elétrico (ONS), 28,49% do crescimento nos próximos 4 anos (2024-2028) será devido à energia solar e 38,11% devido à Micro e

Mini Geração Distribuída (MMGD) (Figura 1), composta marjoritariamente por geradores fotovoltaicos.

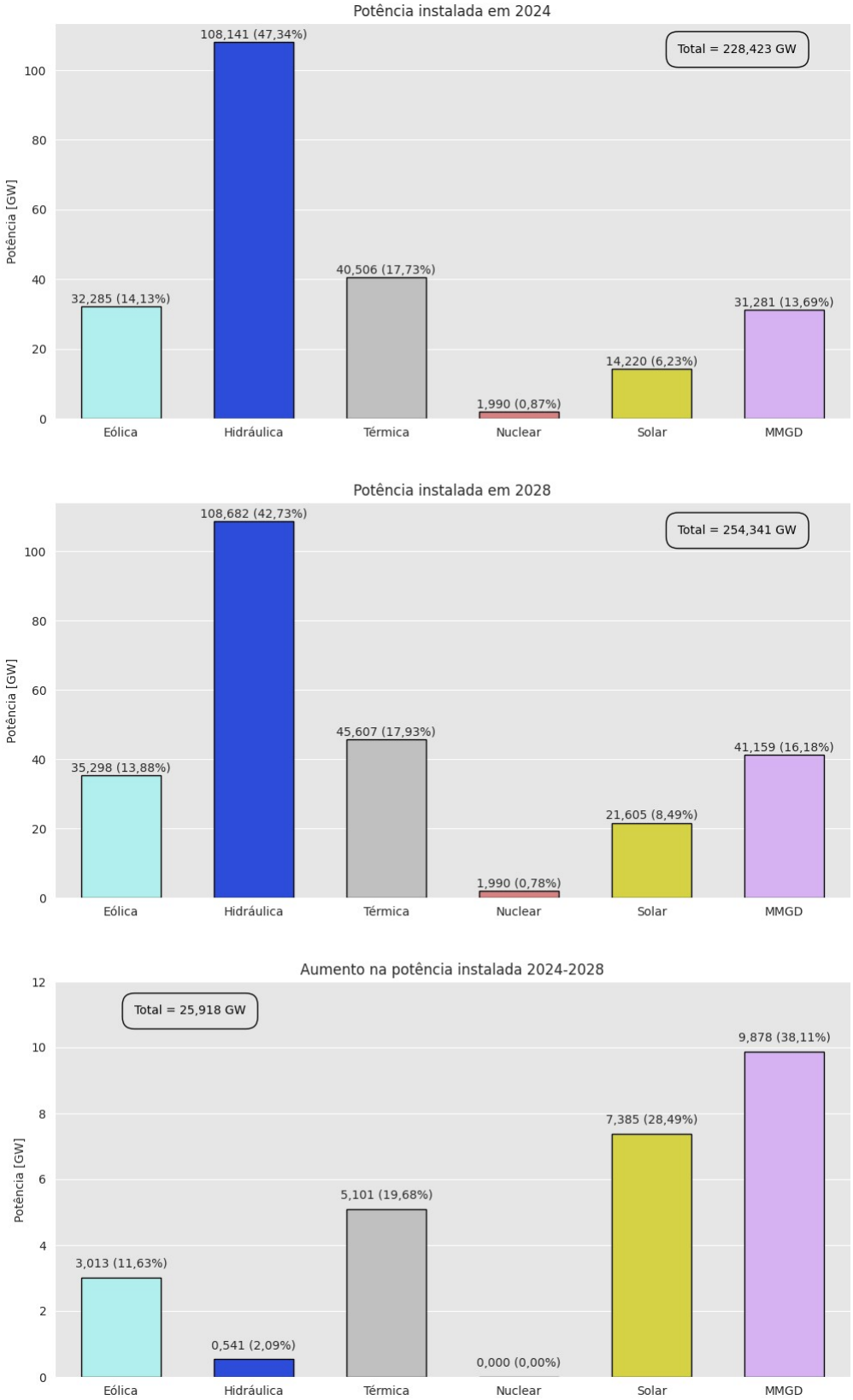
Esses números indicam que o Brasil está fortemente comprometido em trilhar um caminho em direção a um sistema com mais fontes sustentáveis/renováveis e em aumentar a participação da energia solar fotovoltaica na rede elétrica. Todavia, tal caminho está repleto de desafios, já que essa é uma fonte **não despachável**, ou seja, sua disponibilidade depende de fênomenos naturais sobre os quais o operador do sistema (ONS) não tem controle, dificultando seu planejamento de longo, médio e curto prazo. Surge, portanto, a motivação para este trabalho, que consiste em criar modelos para gerar previsões de irradiância de curto prazo e fornecer informações sobre a disponibilidade do recurso solar com antecedência ao operador, auxiliando-o: na decisão para o escalonamento das unidades produtoras; no dimensionamento dos requisitos de reserva operativa; na redução do grau de incerteza associado ao despacho econômico. Além disso, essas previsões também são de grande interesse dos proprietários do ativo solar, que buscam maximizar a eficiência e a produção solar para cumprir seus contratos de mercado.

1.1 Problemática

A geração solar fotovoltaica é, por natureza, não despachável e converte a energia solar diretamente em energia elétrica, por meio da conexão em série e/ou paralelo das células fotovoltaicas, usualmente compostas por silício monocristalino ou policristalino.

O silício é um material semicondutor cujos átomos possuem quatro elétrons na camada de valência. Ao ser dopado com átomos trivalentes (três elétrons na camada de valência) obtém-se um cristal com excesso de portadores de carga positiva (lacunas), ao qual se da o nome de silício tipo p. Ao ser dopado com átomos pentavalentes (cinco elétrons na camada de valência) obtém-se um cristal com excesso de portadores de carga negativa (elétrons), ao qual se da o nome de silício tipo n. Ambos os cristais são eletricamente neutros pois sua carga líquida (soma dos elétrons com prótons) é nula. Porém, ao se formar uma junção p-n, as lacunas em excesso do lado p fluirão para o lado n e os elétrons em excesso do lado n fluirão para o lado p, gerando uma região de carga não nula nas proximidades da junção, que é chamada de camada de depleção. Com isso, surge-se um campo elétrico $\overrightarrow{E}$ (Figura 2) que impede o fluxo de novos portadores de carga, em ambos os lados. Contudo, se as extremidades dos dois cristais forem conectadas, como é feito nas células fotovoltaicas, os elétrons do lado n, na presença do campo elétrico $\overrightarrow{E}$, irão se recombinar com as lacunas do lado p até que não sobre mais nenhum par elétron/lacuna. Ou seja, o campo elétrico criado pela camada de depleção gera uma diferença de potencial entre o cristal do tipo p e n, bem como uma corrente elétrica na presença de um circuito externo. Havendo corrente elétrica e tensão, há potência elétrica, e esse fenômeno recebe

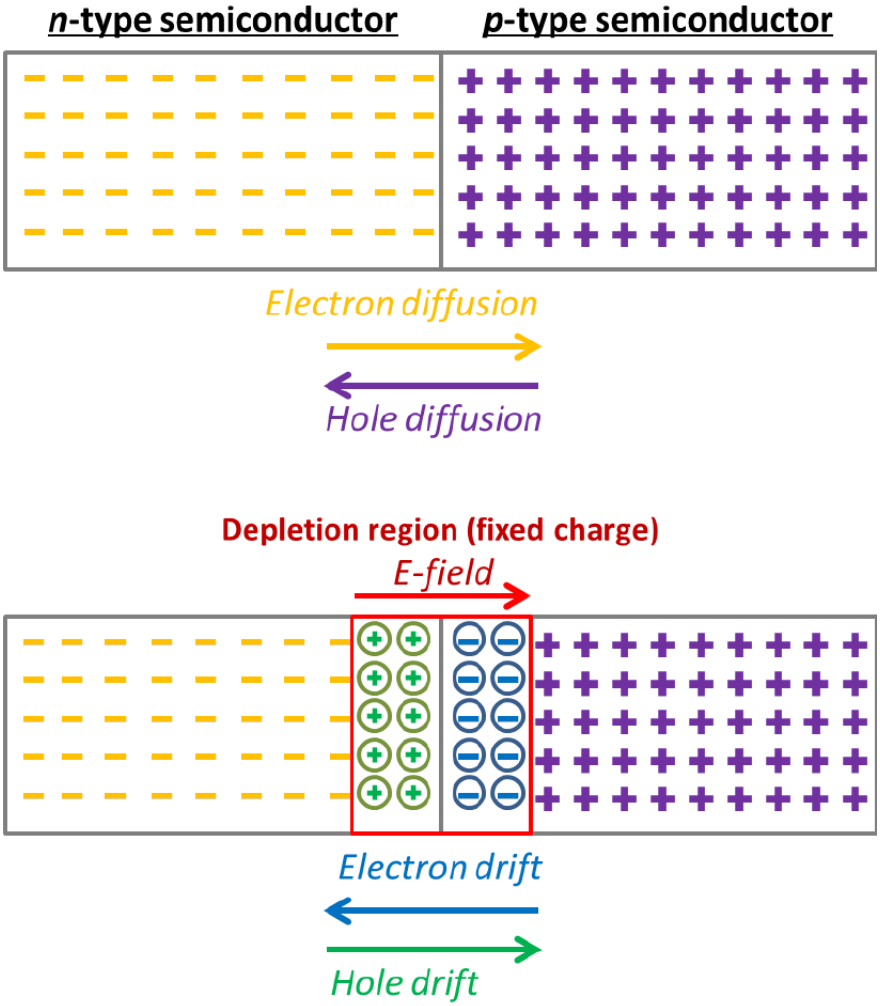
Figura 1: Capacidade instalada no SIN em 2024, simulada para 2028 e a diferença entre as duas.



Fonte: Adaptado de https://www.ons.org.br/paginas/sobre-o-sin/o-sistema-em-numeros

o nome de efeito fotovoltaico [Coelho, Schmitz e Martins 2022]. Se essa célula é exposta ao Sol, os fótons suficientemente energéticos advindos da radiação solar irão gerar novos pares de elétrons/lacunas, prolongando o efeito fotovoltaico e gerando mais energia.

Figura 2: Junção p-n.



Fonte: [Needleman 2014]

A geração fotovoltaica está intrinsecamente ligada à disponibilidade desses fótons, quanto mais disponíveis, maior a energia gerada e vice-versa. No curto prazo (poucos minutos até uma hora) essa disponibilidade é afetada principalmente pela presença de nuvens esparsas que bloqueiam/desbloqueiam o Sol, resultando em mudanças muito rápidas na geração fotovoltaica, que elevam o grau de incerteza associado as tomadas de decisão do operador e prejudicam a gestão do sistema. Um elevado grau de incerteza acarreta em um custo de operação muito alto, já que ele dificulta um equilíbrio entre as unidades produtoras que fornecem potência (serviços de sistema) e as unidades produtoras que fornecem energia (eficiência energética). Portanto, para acomodar uma participação cada vez maior da geração fotovoltaica, é preciso que o operador adote soluções que tornem o

sistema mais robusto à intermitência dessa geração e que reduzam os custos de operação. Uma solução bastante promissora é a utilização de modelos de previsão de irradiância de curto prazo, pois isso aumentará o controle do operador sobre a geração fotovoltaica e irá auxiliá-lo nas suas tomadas de decisão, diminuindo o grau de incerteza e aumentando a eficiência no despacho das unidades produtoras. Todavia, a criação de bons modelos, com previsões confiáveis, é uma tarefa difícil pois exige uma modelagem matemática das dinâmicas altamente complexas das nuvens e de como elas afetam a irradiância.

Até aqui utilizou-se os termos radiação solar e irradiância solar como sendo a mesma coisa, porém, a radiação solar se refere a energia expelida pelo Sol, proveniente das reações químicas que ocorrem em seu núcleo, enquanto que a irradiância solar é a potência contida na radiação que incide em uma dada área. Essa energia, após viajar milhões de quilômetros pelo vácuo espacial, chega até o topo da atmosfera terrestre com uma irradiância média de 1367 W/m^2, denominada de constante solar. Ao adentrar na atmosfera essa radiação interage com os gases atmosféricos, aerossóis e nuvens, parcelando-se em diferentes componentes e alterando significativamente o seu perfil na superfície terrestre em comparação com o perfil no topo da atmosfera. As principais componentes de irradiância, apresentadas na Figura 3, são:

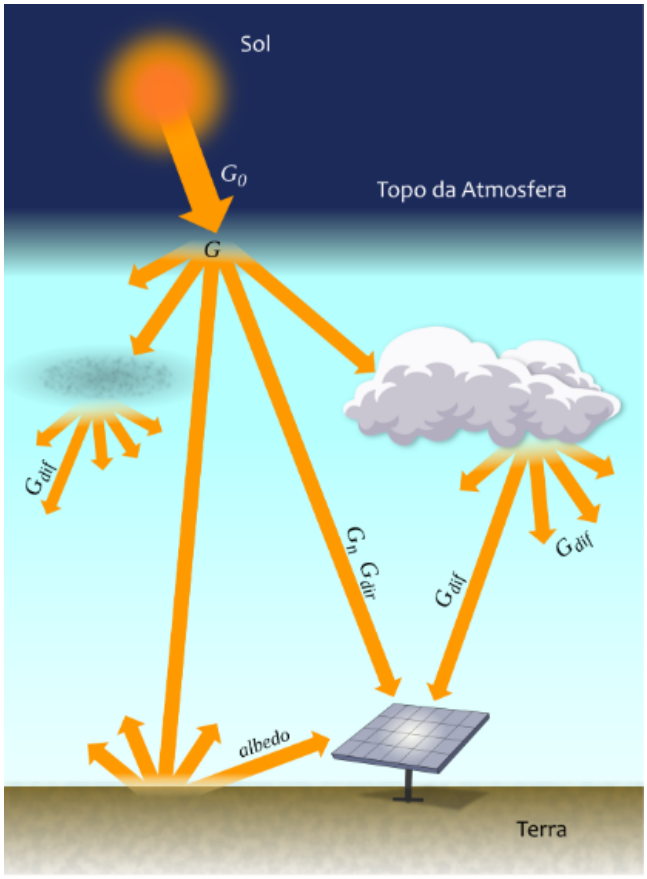
- Irradiância Direta Normal (DNI): Denominada por G_n na figura, é parcela de irradiância que chega diretamente à superfície e que não sofre difusão pelos gases atmoféricos, aerossóis ou nuvens;

- Irradiância Difusa Horizontal (DHI): Denominada por G_{dif} na figura, é a parcela da irradiância proveniente da difusão dos gases atmosféricos, aerossóis e nuvens;

- Irradiância Global Horizontal (GHI): Denominada por G na figura, é a soma das duas parcelas anteriores, dada por $GHI = DHI + DNI * cos(\theta)$ onde θ é o ângulo zenital.

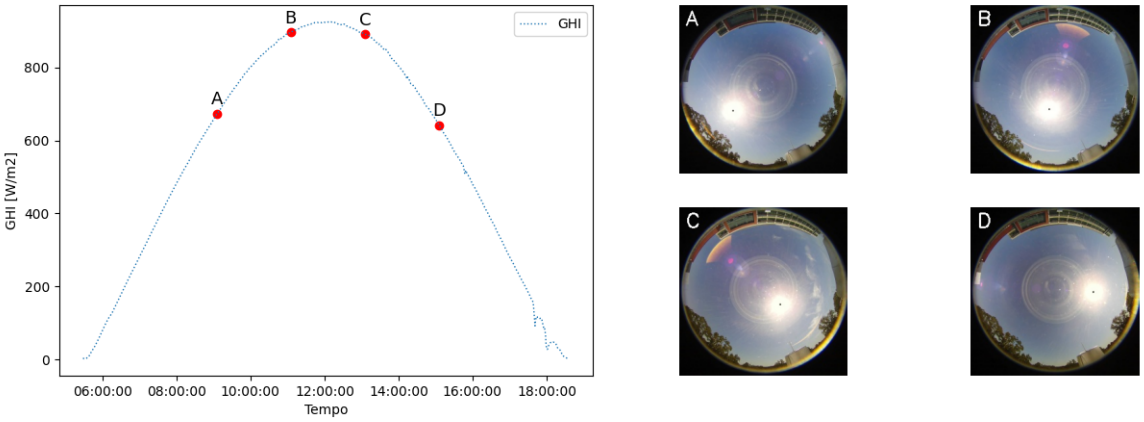
Neste trabalho, o foco será na componente GHI, uma vez que essa componente engloba todas as outras e possui uma relação linear com a potência fotovoltaica gerada. Nas Figuras 4, 5 e 6 pode-se ver o comportamento desse sinal para diferentes tipos de dias: totalmente ensolarados, parcialmente nublados e totalmente nublados. Percebe-se que a previsão de curto prazo é significativamente mais desafiadora nos dias parcialmente nublados, haja vista a grande variação de amplitude que a GHI pode atingir em um curto intervalo de tempo.

Figura 3: Componentes da irradiância solar.



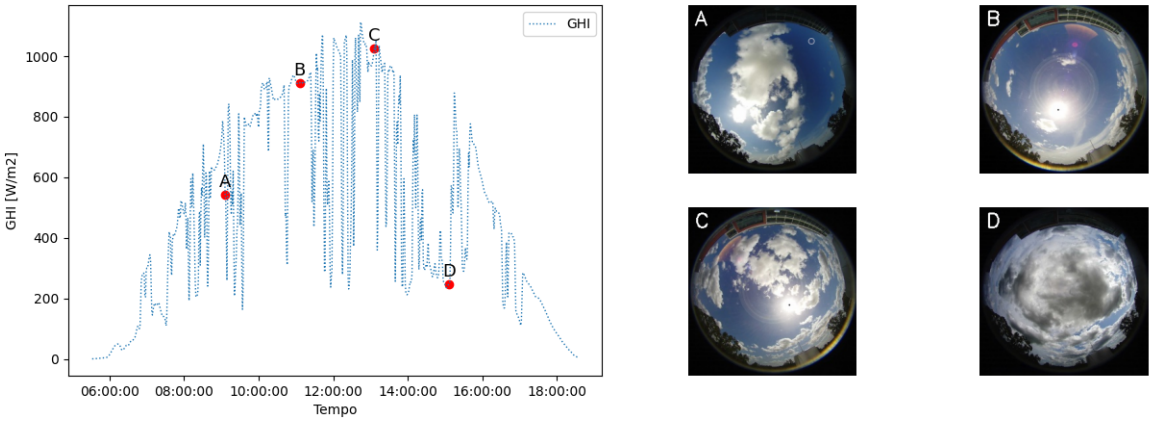
Fonte: Atlas Brasileiro de Energia Solar - 2ª edição

Figura 4: Sinal de GHI em um dia ensolarado.



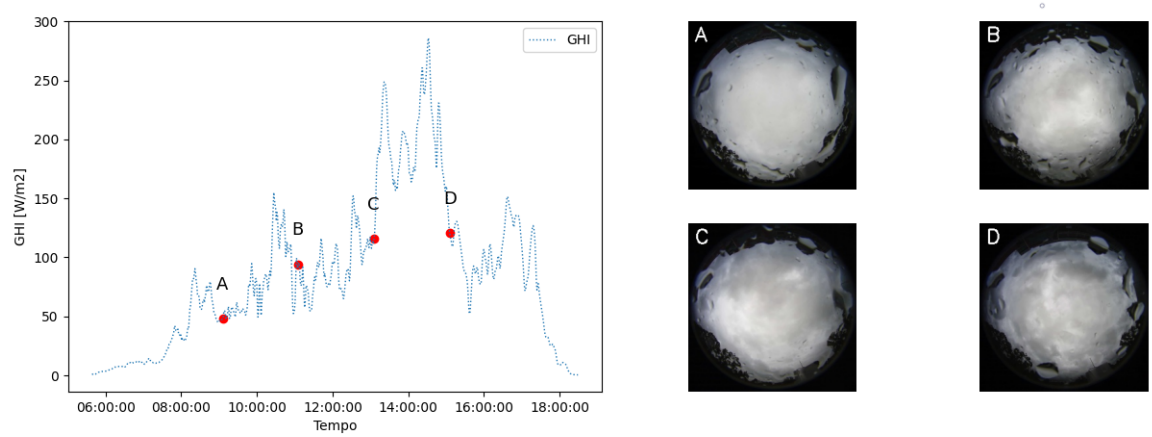
Fonte: O autor

Figura 5: Sinal de GHI em um dia com nuvens esparsas.



Fonte: O autor

Figura 6: Sinal de GHI em um dia totalmente nublado.



Fonte: O autor

1.2 Objetivo

1.2.1 Objetivo Geral

Desenvolver modelos baseados em técnicas de *Deep Learning* para prever a GHI com 5 minutos de antecedência **utilizando apenas imagens do céu.** Com isso, espera-se:

- Criar uma ferramenta de apoio a operação do sistema em tempo real.
- Aumentar a eficiência no despacho das unidades produtoras e reduzir os custos de operação.

1.2.2 Objetivos Específicos

- Revisar conceitos fundamentais de previsão da GHI e *deep learning* como: modelos de céu claro, modelo persistente, modelo do *perceptron*, redes neurais recorrentes e redes convolucionais.
- Avaliar a qualidade e a confiabilidade do conjunto de dados utilizado através de uma ampla análise exploratória.
- Utilizar técnicas de aprendizado supervisionado para treinar modelos de previsão da GHI com o uso exclusivamente de imagens do céu.
- Analisar e discutir parâmetros que foram relavantes para o treinamento dos modelos, bem como destacar suas possíveis limitações.

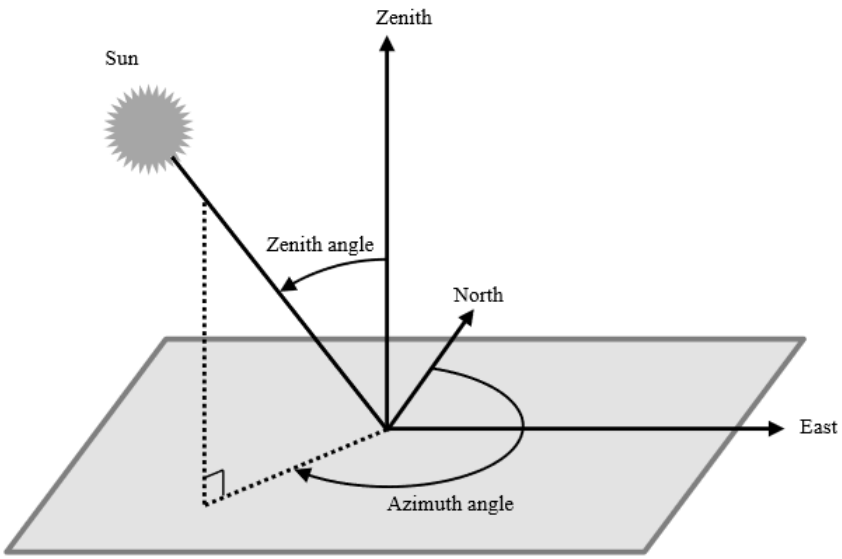
2 Fundamentação Teórica

2.1 Previsão de GHI

2.1.1 Modelos de Céu Claro

Em dias de céu claro a modelagem matemática da GHI se torna mais simples (Figura 4), uma vez que não há a presença de nuvens para atenuar a irradiância. Nesse caso o principal fator que altera a GHI é o ângulo zenital, que é o ângulo entre a radiação solar e a reta vertical ao observador (Figura 7). Quanto maior for esse ângulo, maior será a distância que a radiação solar deverá percorrer pela atmosfera e, portanto, menor a irradiância que chega ao observador, e vice-versa. Outros fatores que também afetam a GHI são: aerossóis, altitude, pressão, vapor d'água, entre outros [Reno, Hansen e Stein 2012].

Figura 7: Ângulos zenital (reta vertical ao observador) e azimutal (reta horizontal ao observador).



Fonte: [Nou et al. 2016]

Modelos que consideram apenas o ângulo zenital para determinar a irradiância de céu claro são os mais simples, como o modelo de Haurwitz [Haurwitz 1945]. À medida que outros fatores vão sendo considerados, os modelos vão se tornando mais complexos, como os modelos de Ineichen [Ineichen e Perez 2002] e o Simplified Solis [Ineichen 2008], que utilizam valores climatológicos como vapor d'água, aerossóis e turbidez de Linke para auxiliar na previsão. A forma mais simples de se implementar esses modelos é através de

bibliotecas de código aberto, como o pvlib[1], que já possui métodos integrados para gerar previsões de GHI de céu claro e valores padrão dos dados climatológicos necessários para cada modelo. Nesse caso, os únicos dados obrigatórios para obter uma previsão de GHI de céu claro são: a localização (latitude, longitude e altitude), data (mês, dia, hora e minuto) e o modelo de céu claro de interesse.

A utilidade desses modelos vai além da previsão da GHI em dias de céu claro, uma vez que com eles é possível definir um índice bastante relevante na aplicação deste trabalho, que é o índice de céu claro $k_{t,model}$, definido como:

$$k_{t,model} = \frac{GHI}{GHI_{cs,model}} \tag{2.1}$$

onde GHI é o valor da GHI medido e $GHI_{cs,model}$ é o valor de GHI previsto pelo modelo de céu claro escolhido. O valor de $k_{t,model}$ é adimensional e, teoricamente, varia de 0 (céu completamente nublado) à 1 (céu claro). Porém, na prática, podem haver casos em que a presença de nuvens aumenta a irradiância, resultando no fênomeno conhecido como *overirradiance* e em valores de $k_{t,model}$ maiores que 1. No Brasil, o maior valor de *overirradiance* registrado foi de 1845 W/m^2 em Caucaia-CE [Nascimento et al. 2019], quando as nuvens acabaram refletindo a irradiância para um mesmo ponto, aumentando drasticamente a componente DHI da irradiância global.

2.1.2 Modelo Persistente

Ao construir um modelo que gera previsões de um sinal que varia no tempo, o objetivo é minimizar o erro entre o valor previsto y_{pred} e o valor real medido y_{true}. É lógico pensar que o desempenho do modelo será tão pior quanto maior for o erro, porém, sem uma referência, torna-se difícil avaliar se o erro gerado pelo modelo está acima ou abaixo do esperado. O modelo persistence é um modelo que assume, ingenuamente, que o valor do sinal $y(t)$ se manterá constante durante todo o intervalo de previsão Δt, de forma que:

$$\mathrm{Y}(t + \Delta t) = \mathrm{Y}(t) \tag{2.2}$$

No caso da GHI, é possível formular um outro modelo persistente, chamado de *smart persistence*, que possui um desempenho melhor do que o persistente tradicional. Ao invés de assumir que a GHI se manterá constante durante todo o intervalo Δt, o modelo *smart persistence* assume que o índice de céu claro $k_{t,model}$ é que se manterá constante, e a previsão da GHI é então gerada pelo índice de céu claro atual e a irradiância de céu claro futura

$$\mathrm{GHI}(t + \Delta t) = k_{t,\mathrm{model}}(t) \cdot \mathrm{GHI}_{cs,\mathrm{model}}(t + \Delta t) \tag{2.3}$$

[1] https://pvlib-python.readthedocs.io/en/stable/index.html

onde:

$$\mathrm{GHI}(t+\Delta t) = \text{valor futuro da GHI previsto para o instante t + } \Delta\text{t;}$$
$$k_{t,\mathrm{model}}(t) = \text{valor atual do índice de céu claro;}$$
$$\mathrm{GHI}_{cs,\mathrm{model}}(t+\Delta t) = \text{valor futuro da GHI de céu claro previsto para o instante t + } \Delta\text{t.}$$

Os modelos persistentes são considerados modelos de referência pois são os modelos de previsão de GHI mais simples possíveis, já que o único dado de entrada necessário é a medida de GHI/k_t atual. Sendo assim, qualquer outro modelo deverá obter um desempenho melhor que o *smart persistence* para ser considerado um bom modelo e, através de métricas avaliativas, é possível medir o quão melhor o seu desempenho está em relação ao modelo persistente.

2.1.3 Métricas de Avaliação

Existem diversas métricas adequadas para se avaliar o desempenho de um modelo de regressão, sendo que muitas delas se baseiam em algum tipo de média do erro obtido. Neste trabalho serão usadas duas métricas de avaliação. A primeira delas será a raiz do erro médio quadrático (RMSE), uma vez que esse métrica pune mais erros que são muito grandes, que são mais problemáticos para o sistema elétrico do que vários erros pequenos. O RMSE pode ser definido da seguinte forma:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathrm{GHI}_{\mathrm{pred}} - \mathrm{GHI}_{\mathrm{true}})^2}. \tag{2.4}$$

A outra métrica de avaliação será o *Forecasting Skill* (FS), que mede o quão melhor ou pior o modelo está em relação ao modelo de referência (*smart persistence*). Valores negativos indicam que o modelo está pior e, para valores positivos, quanto mais próximo de 1 mais correta está a previsão. O FS depende da métrica de erro escolhida, nesse caso do RMSE:

$$\mathrm{FS} = 1 - \frac{\mathrm{RMSE}_{\mathrm{model}}}{\mathrm{RMSE}_{\mathrm{persistence}}} \tag{2.5}$$

onde $\mathrm{RMSE}_{\mathrm{model}}$ é o RMSE das previsões do modelo obtido e $\mathrm{RMSE}_{\mathrm{persistence}}$ é o RMSE das previsões do modelo *smart persistence*. Por ser uma métrica normalizada pelas previsões do *smart persistence*, ela facilita muito a comparação entre diferentes estudos, já que ela será independente do conjunto de dados utilizado.

2.2 Aprendizado Profundo

O aprendizado de máquina (*machine learning* - ML) é o ramo da inteligência artificial cujos modelos aprendem a realizar uma tarefa sem terem sido explicitamente

programados para isso [Géron 2019]. Uma boa analogia seria, por exemplo, obter um modelo capaz de assar um bolo. Na inteligência artificial tradicional seria necessário primeiro a confecção de uma receita e depois a programação explícita dela em código para reproduzi-lo. Por outro lado, se o bolo que se deseja reproduzir já existe, então é possível ensinar à máquina a combinar os ingredientes de tal forma que, na tentativa e erro, uma versão aproximada do bolo possa ser reproduzida, indefinidamente, sem nunca ter precisado de uma receita explícita.

O aprendizado desses modelos pode ser supervisionado, não-supervisionado, auto-supervisionado ou por reforço. No aprendizado supervisionado, que é o foco deste trabalho, os modelos aprendem a partir de dados rotulados, onde para cada dado de entrada tem-se um rótulo que representa a saída desejada daquela entrada. Esses rótulos podem ser tanto medidas automáticas de sensores, como anotações manuais de especialistas na área, dependendo da aplicação do modelo. Já as entradas dependem do ramo de *machine learning* em questão. No ramo tradicional, essas entradas são atributos numéricos e considerados relevantes para a aplicação. Por exemplo, se um modelo tem o objetivo de prever o preço de um imóvel, alguns atributos interessantes poderiam ser: número de cômodos, área do imóvel, número de estabelecimentos (restaurantes, escolas, hospitais) próximos, entre outros. Já para outros tipos de dados, como imagens, textos ou áudios, a transformação em atributos numéricos se torna bastante complexa, limitando a abordagem tradicional do aprendizado de máquina. Com isso, surge o aprendizado profundo (*deep learning*), que é o ramo de ML fundamentado em redes neurais artificiais (*artificial neural network* - ANN) capazes de codificar esses dados mais complexos em vetores, comumente chamados de *embeddings*, que representam aquele dado em um espaço latente, onde cada dimensão desse espaço corresponde ao atributo numérico automaticamente extraído pela rede neural [Zhang et al. 2020].

O conceito de ANNs foi introduzido há muito tempo por [McCulloch e Pitts 1943], porém levaram-se décadas para ele ser considerado relevante e apenas mais recentemente que essas arquiteturas têm ganhado destaque. Isso porque modelos baseados em ANNs requerem uma enorme quantidade de dados de treinamento para aprender e somente nos últimos anos que conjuntos de dados como o ImageNet [Deng et al. 2009], com milhões de imagens, estão disponíveis para treinar esses modelos. Outro fator determinante para a popularização das ANNs nos últimos anos foi o grande avanço no poder computacional das GPUs, que permitem paralelizar e reduzir o tempo de treinamento dessas redes em múltiplas ordens de grandeza. Por último, tem-se também a criação de bibliotecas de código aberto especializadas em *deep learning*, como Pytorch[2] e TensorFlow[3], que abstraem as partes mais complicadas do treinamento dos modelos, facilitando seu desenvolvimento e impulsionando avanços na comunidade científica.

[2] https://pytorch.org/
[3] https://www.tensorflow.org/?hl=pt-br

2.2.1 Perceptron

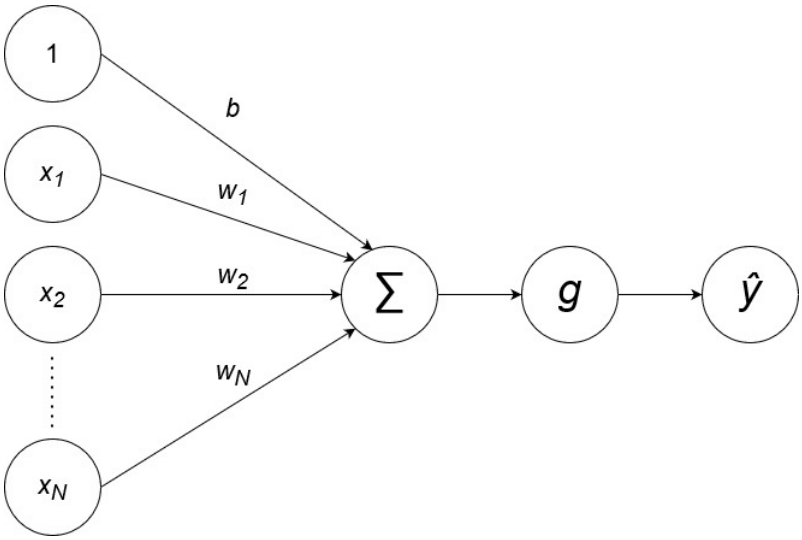
O perceptron é a arquitetura mais simples possível de uma rede neural e consiste basicamente em uma única unidade de processamento de informação [Watt, Borhani e Katsaggelos 2016]. Matematicamente, esse processamento pode ser descrito como a soma ponderada de suas entradas que, somada à um termo de deslocamento (*bias*), passará por uma função de ativação não-linear, conforme (2.6) e ilustrado na Figura 8.

$$\hat{y} = g\left(b + \sum_{i=1}^{N} w_i x_i\right) = g\left(z\right) \tag{2.6}$$

onde:

$\hat{y}$ = saída do perceptron;
g = função de ativação não-linear (ver Figura 9);
b = termo de *bias*;
N = número de entradas do perceptron;
x_i = i-ésima entrada do perceptron;
w_i = peso associado à i-ésima entrada do perceptron.

Figura 8: Diagrama do perceptron.

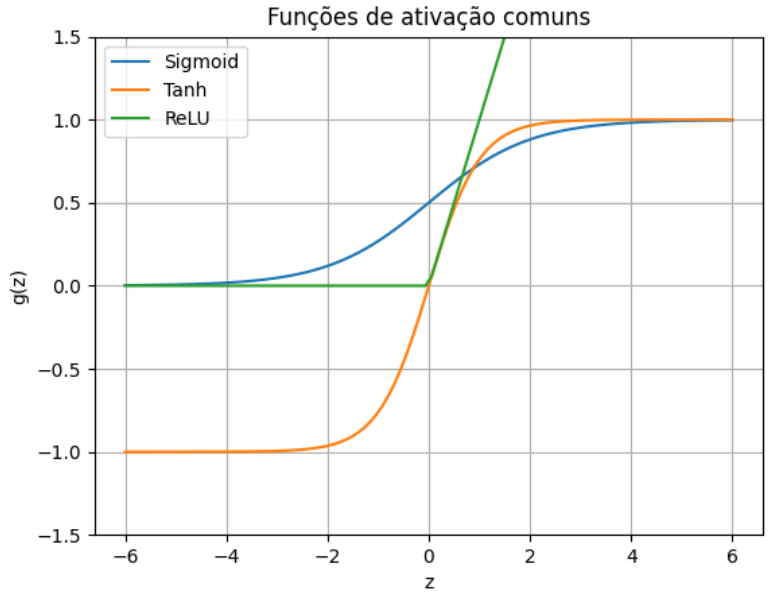


Fonte: O autor

Também é possível reescrever (2.6) em termos de vetores e matrizes com o auxílio da álgebra linear, resultando em (2.7).

$$\hat{y} = g\left(b + \mathbf{w}^{\mathbf{T}}\mathbf{x}\right) = g\left(z\right) \tag{2.7}$$

Figura 9: Funções de ativação mais comuns: sigmóide, tangente hiperbólica e ReLU.



Fonte: O autor

onde:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

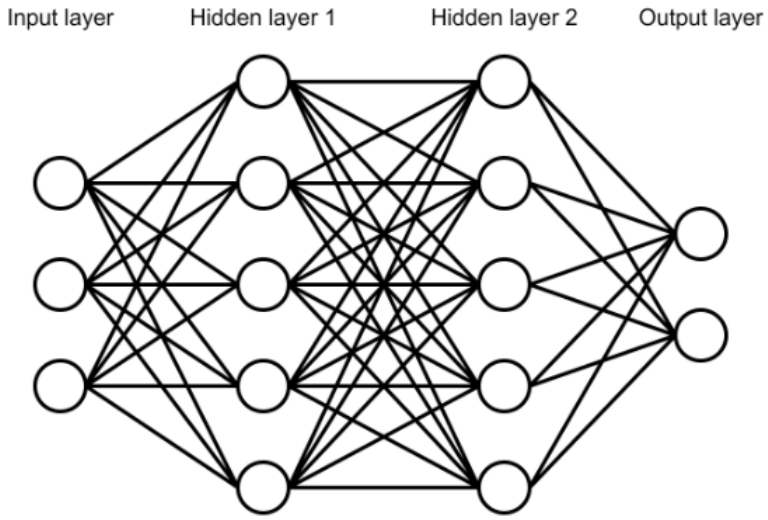
Devido à sua simplicidade, uma rede neural de um único perceptron é extremamente limitada em sua capacidade de aprendizado. Todavia, a adição de mais perceptrons à rede aumenta sua capacidade e permite a resolução de problemas mais complexos. Isso pode ser feito tanto dentro de uma mesma camada, quanto em camadas distintas, resultando em uma rede *multi layer perceptron* (MLP) [Zhang et al. 2020], conforme a Figura 10. Ao adicionar mais perceptrons dentro de uma mesma camada o vetor de pesos $\mathbf{w}$ se torna uma matriz $\mathbf{W}$ e o termo de *bias* se torna um vetor. Supondo que a primeira camada da rede tenha K perceptrons, então a saída dela pode ser determinada por (2.8).

$$\hat{y} = g\left(\mathbf{z}\right) \tag{2.8}$$

onde:

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_K \end{bmatrix}^{K\times 1} = \begin{bmatrix} \dots \mathbf{w}_1^{[1]} \dots \\ \dots \mathbf{w}_2^{[1]} \dots \\ \vdots \\ \dots \mathbf{w}_K^{[1]} \dots \end{bmatrix}^{K\times N} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}^{N\times 1} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_K^{[1]} \end{bmatrix}^{K\times 1}$$

Figura 10: Arquitetura *multi layer perceptron*



Fonte: [Gustineli 2022]

Em uma rede MLP com L camadas, a saída da última camada pode ser obtida conforme (2.9).

$$\begin{aligned} \mathbf{a}^{[1]} &= g\left(\mathbf{b}^{[1]} + \mathbf{W}^{[1]}\mathbf{x}\right) \\ \mathbf{a}^{[2]} &= g\left(\mathbf{b}^{[2]} + \mathbf{W}^{[2]}\mathbf{a}^{[1]}\right) \\ &\vdots \\ \mathbf{a}^{[\ell]} &= g\left(\mathbf{b}^{[\ell]} + \mathbf{W}^{[\ell]}\mathbf{a}^{[\ell-1]}\right) \end{aligned} \tag{2.9}$$

onde:

K_ℓ é o número de perceptrons da camada ℓ

$\mathbf{a}^{[\ell]}$ é o vetor de saída da camada ℓ e com dimensão $K_\ell \times 1$

$\mathbf{b}^{[\ell]}$ é o vetor de *bias* da camada ℓ e com dimensão $K_\ell \times 1$

$\mathbf{W}^{[\ell]}$ é a matriz de pesos da camada ℓ e com dimensão $K_{\ell-1} \times K_\ell$

2.2.1.1 Treinamento de uma Rede Neural

Para que um modelo consiga apresentar um desempenho satisafatório, é necessário otimizar as matrizes de pesos associadas à cada uma de suas camadas. Para isso, é utilizado um conjunto de dados de treino e uma função de perda com o intuito de minimizar o erro entre as previsões do modelo e as respostas anotadas do conjunto. Em problemas de regressão essas previsões pertencem à um espaço contínuo de possibilidades, como na previsão de GHI, que pode ser qualquer número real positivo, até um certo limite. Nesses

tipos de problema uma função perda bastante comum é o erro quadrático, dada por (2.10).

$$L(\hat{y}^{(i)}, y^{(i)}) = L\left(f(x^{(i)}, \mathbf{W}) - y^{(i)}\right)^2 \tag{2.10}$$

O objetivo de uma rotina de treino é, portanto, encontrar o conjunto de parâmetros ótimo $\mathbf{W}^*$ que minimize a função custo $J(\mathbf{W}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$, através do algoritmo de descida do gradiente.

Algorithm 1 Descida do Gradiente

Input: Inicialização dos pesos de forma aleatória
1: **while** convergência não atingida **do**
2: Cálculo do gradiente: $\frac{\partial J}{\partial \mathbf{W}}$
3: Atualização dos pesos: $\mathbf{W} = \mathbf{W} - \alpha\frac{\partial J}{\partial \mathbf{W}}$
4: **end while**
Output: Pesos otimizados

Para que o gradiente $\frac{\partial J}{\partial \mathbf{W}}$ possa ser calculado de forma eficiente utiliza-se o algoritmo de *backpropagation* [Watt, Borhani e Katsaggelos 2016]. Primeiro a regra da cadeia é aplicada de trás para frente na rede para determinar o gradiente da função custo com respeito à todas as ativações de cada perceptron (que pode ser interpretada como a contribuição para o erro de cada perceptron), obetendo-se $\frac{\partial J}{\partial \mathbf{z}}$. Em seguida, obtém-se o gradiente com respeito aos pesos:

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial \mathbf{z}}{\partial \mathbf{W}} \cdot \frac{\partial J}{\partial \mathbf{z}} \tag{2.11}$$

Embora a aplicação da regra da cadeia permita o cálculo eficiente do gradiente, ainda assim o tempo de treinamento pode ser excessivamente longo, principalmente com conjuntos de dados cada vez maiores. Não só isso, mas armazenar todos os gradientes do conjunto em memória é simplesmente inviável com a computação atual. Por esses motivos, utiliza-se a Descida do Gradiente Estocástica na prática, mais popularmente conhecida como *Stochastic Gradient Descent* (SGD), em inglês. O termo "estocástico" surge pois, nesse algoritmo, o conjunto de treinamento é dividido em *batches* de tamanho N (geralmente na ordem de dezenas à centenas) e a descida do gradiente é realizada apenas com as observações de um *batch*. Com isso, é possível reduzir o número de gradientes que precisam ser armazenados em memória e acelerar drasticamente o tempo de treinamento, uma vez que os pesos serão atualizados diversas vezes dentro de uma mesma época de treinamento.

Também existem outras variações que podem ser aplicadas ao algoritmo da descida do gradiente, que tendem a melhorar a otimização dos pesos através de uma taxa de aprendizado α adaptativa. Uma delas é aplicar a noção de momento à descida do gradiente de forma que, se a atualização dos pesos está sempre caminhando na mesma direção, então

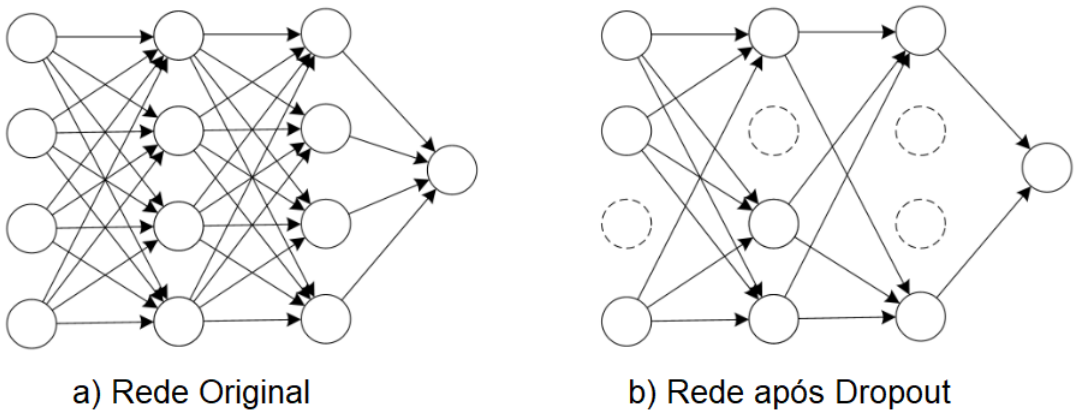
aumenta-se a "velocidade" com que os pesos são atualizados nessa direção. O otimizador SGD com momento utiliza essa metodologia [Watt, Borhani e Katsaggelos 2016]. Outro otimizador bastante utilizado é o Adam [Kingma e Ba 2014], que usa momentos de primeira e segunda ordem para adaptar taxas de aprendizado diferentes para cada um dos pesos.

2.2.1.2 Overfitting

Independente da escolha do otimizador, a otimização dos pesos do modelo será sempre feita com base no conjunto de treino. Todavia, na prática, o interesse real é obter um bom desempenho no conjunto de teste, já que ele representa a capacidade de generalização das previsões do modelo para dados não vistos durante o seu treinamento. Por esse motivo, deve-se ter em mente que o modelo com o melhor desempenho possível no treino não necessariamente implica em um melhor modelo no geral, já que ele pode simplesmente estar "decorando" as respostas do treino sem aprender algo de fato útil, também conhecido como *overfitting* no conjunto de treino. Uma técnica bastante utilizada para se evitar *overfitting* é a regularização L2 ou *weight decay* [Watt, Borhani e Katsaggelos 2016], que adiciona um termo a mais na função custo para penalizar parâmetros muito elevados através de um hiperparâmetro λ, dada por (2.12). Também é possível evitar *overfitting* através da técnica de *dropout*, que consiste em aleatoriamente desativar alguns perceptrons de cada camada, conforme ilustrado na Figura 11, evitando que as previsões de um único perceptron domine sobre as demais e forçando o aprendizado de todos os perceptrons [Hinton et al. 2012].

$$J(\mathbf{W}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \lambda \mathbf{W}^T\mathbf{W} \tag{2.12}$$

Figura 11: Técnica de *dropout* em uma rede neural.



Fonte: Adaptado de [Khalifa e Frigui 2016]

2.2.1.3 Otimização de Hiperparâmetros

Hiperparâmetros são os parâmetros que não são aprendidos, porém possuem um grande impacto durante o treinamento do modelo. Na prática, há uma enorme quantidade de fatores que podem ser considerados hiperparâmetros, sendo que alguns podem ser específicos de cada problema, enquanto outros são mais gerais e fazem sentido para qualquer aplicação, como por exemplo:

- Taxa de aprendizado: determina o quanto que os pesos serão atualizados pelo cálculo do gradiente. Valores muito altos podem explodir a otimização e valores muito baixos podem demorar demais o aprendizado, logo deve ser ajustada com cautela.

- Batch size: determina o número de amostras que serão observadas para estimar os gradientes. Valores altos exigem um maior poder computacional, bem como um maior número de épocas de aprendizado, porém tornam o processo de otimização mais determinístico. Em contrapartida, valores baixos vão requerer menos computação e um número menor de épocas, porém a otimização será mais estocástica.

- Otimizador: a escolha de otimizadores como SGD com momento, Adam e AdamW (Adam levemente modificado com o *weight decay*) também pode impactar o desempenho do modelo, porém não existe uma regra prática para definir qual é melhor para cada caso e a escolha geralmente é feita com base na tentativa e erro.

- Weight decay: determina o valor de λ na regularização L2 que força o modelo a diminuir o valor de seus parâmetros. Valores muito elevados podem causar *underfitting* (modelo não acerta nem o conjunto de treino) e valores muito baixos podem causar *overfitting* (modelo simplesmente "decora" o conjunto de treino).

- Dropout: determina a probabilidade de um perceptron ser desativado durante o treino. Assim como o λ, valores muito altos podem causar *underfitting* e muito baixos *overfitting*.

- Número de perceptrons/Número de camadas: o número de perceptrons por camada e o número de camadas também pode ser considerado um hiperparâmetro. Quanto mais, maior sera a capacidade de modelo e maior o risco de *overfitting*. Quanto menos, maior o risco de *underfitting*.

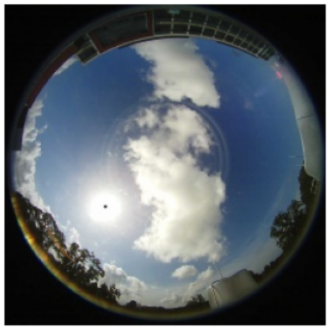
Esses são apenas alguns dos inúmeros hiperparâmetros que devem ser levados em consideração **antes** de treinar uma rede neural. Como os hiperparâmetros não são aprendidos pela rede, a busca pela melhor combinação possível não é uma tarefa trivial e geralmente é feita com base em um de três algoritmos: *grid search*, *random search* e *bayesian search*.

O *grid search* consiste em exaustivamente testar **todas** as combinações possíveis, o que sempre irá gerar a solução ótima porém não escala nem um pouco bem para um espaço de hiperparâmetros muito grande [Géron 2019]. Nesse caso, o *random search* é a melhor alternativa pois ele irá testar um número N de combinações diferentes, onde o valor de cada hiperparâmetro é escolhido com base em uma distribuição de probabilidade. Isso permite que um leque maior de valores sejam testados para cada hiperparâmetro e que o tempo máximo de busca seja definido pelo usuário através do N [Géron 2019]. Por último, tem-se a busca bayesiana que, ao contrário das duas anteriores, utiliza as iterações passadas para tomar decisões mais informadas sobre quais combinações testar nas próximas iterações, através de processos Gaussianos e funções de aquisição [Frazier 2018]. Para um número pequeno de hiperparâmetros, a busca bayesiana é a que consegue encontrar a solução ótima no menor número de iterações, mas o *random search* tende a ser melhor à medida que o número de hiperparâmetros aumenta.

2.2.2 Redes Neurais Recorrentes

Uma das maiores limitações das redes MLP é que a sua entrada deverá sempre ser um vetor de tamanho fixo. Isso representa um problema para o treinamento de dados sequenciais, onde a ordem de cada dado de entrada é um fator determinante para a previsão do modelo. Por exemplo, foi visto no Capítulo 1 que a movimentação das nuvens é o fator mais relevante para a previsão da GHI, porém, sem uma modelagem sequencial desse dado, qualquer previsão do modelo seria uma previsão aleatória já que é impossível saber em qual direção as nuvens estão se movendo, conforme ilustrado na Figura 12. Já se a previsão é feita em cima de uma sequência de dados de entrada, essa informação será dada para o modelo e poderá ser aprendida, conforme ilustrado na Figura 13.

Figura 12: Imagem do céu estática - Impossível de identificar a movimentação das nuvens



Fonte: O autor.

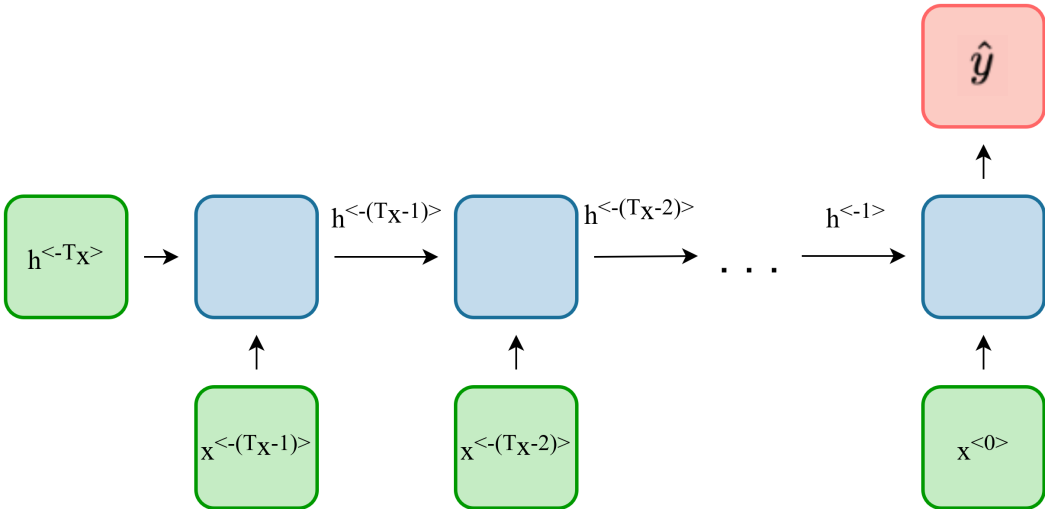
Surge, portanto, a necessidade de uma arquitetura de rede neural diferente para resolver esse problema, como as rede neurais recorrentes (RNN - *recurrent neural network*). As RNNs são redes projetadas especificamente para o aprendizado de dados sequenciais (texto, áudio, séries temporais, entre outros) uma vez que o estado da rede, após o

Figura 13: Sequência de imagens do céu demonstrando a movimentação das núvens

Fonte: O autor.

processamento de um elemento da sequência, é propagado ao longo dela [Zhang et al. 2020]. Esse processo está ilustrado na Figura 14, onde o primeiro vetor da sequência $\mathbf{x}^{<-(T_x-1)>}$ é processado, em conjunto com o vetor de estado (*hidden state*) inicial $\mathbf{h}^{<-T_x>}$, pela RNN (ilustrada pelos quadrados azuis), gerando o próximo vetor de *hidden state* que será processado em conjunto com o próximo vetor da sequência de entrada. Esse processo se repete T_x vezes, que representa o tamanho da sequência de entrada $\mathbf{x}$ e, no caso *many to one*, a previsão $\hat{y}$ é obtida apenas na última iteração (ou *time step*) de processamento. Os *hidden states* de cada *time step* podem ser obtidos a partir de (2.13). É importante ressaltar que as matrizes $\mathbf{W_{hh}}$ e $\mathbf{W_{xh}}$ serão as mesmas em todos os *time steps*.

Figura 14: Arquitetura de uma RNN *many to one*



Fonte: Adaptado de [Amidi e Amidi]

$$\mathbf{h}^{<t>} = tanh\left(\mathbf{W}_{\mathbf{hh}}^{\mathbf{T}}\mathbf{h}^{<t-1>} + \mathbf{W}_{\mathbf{xh}}^{\mathbf{T}}\mathbf{x}^{<t>}\right) \tag{2.13}$$

2.2.2.1 Long Short Term Memory - LSTM

O desaparecimento e a explosão do gradiente durante o *backpropagation* são dois desafios comuns que devem ser enfrentados durante o treinamento de uma rede neural. Isso é especialmente verdade para as redes neurais recorrentes, já que a multiplicação de valores muito pequenos ou muito grandes ao longo de toda a sequência pode rapidamente sair do controle. Para combater esse problema utiliza-se a rede *Long Short Term Memory* (LSTM), que são RNNs aprimoradas para evitar problemas com o gradiente no *backpropagation*, melhorando o rastreamento das dependências temporais dentro de uma sequência. A LSTM [Hochreiter e Schmidhuber 1997] é composta por 4 componentes principais: *forget gate*, *input gate*, *output gate* e *cell state* (ver Figura 15). O *forget gate* é responsável por determinar a quantidade de informação passada que será propagada adiante, através da aplicação da função sigmóide, conforme mostrado em (2.14). Em seguida o *input gate* determina a quantidade de informação nova que deverá ser armazenada, por meio das equações 2.15 e 2.16. O *cell state* da célula atual é então atualizado com base nesses dois *gates* conforme (2.17). Por último, o *output gate* utiliza o *cell state* atual para determinar o quanto de informação que será propagada adiante, conforme mostrado nas equações 2.18 e 2.19.

$$\mathbf{f}^{<t>} = \sigma\left(\mathbf{W_{fh}}\mathbf{h}^{<t-1>} + \mathbf{W_{fx}}\mathbf{x}^{<t>} + \mathbf{b_f}\right) \tag{2.14}$$

$$\mathbf{i}^{<t>} = \sigma\left(\mathbf{W_{ih}}\mathbf{h}^{<t-1>} + \mathbf{W_{ix}}\mathbf{x}^{<t>} + \mathbf{b_i}\right) \tag{2.15}$$

$$\mathbf{S}^{<t>} = tanh\left(\mathbf{W_{Sh}}\mathbf{h}^{<t-1>} + \mathbf{W_{Sx}}\mathbf{x}^{<t>} + \mathbf{b_S}\right) \tag{2.16}$$

$$\mathbf{C}^{<t>} = \mathbf{C}^{<t-1>} \odot \mathbf{f}^{<t>} + \mathbf{S}^{<t>} \odot \mathbf{i}^{<t>} \tag{2.17}$$

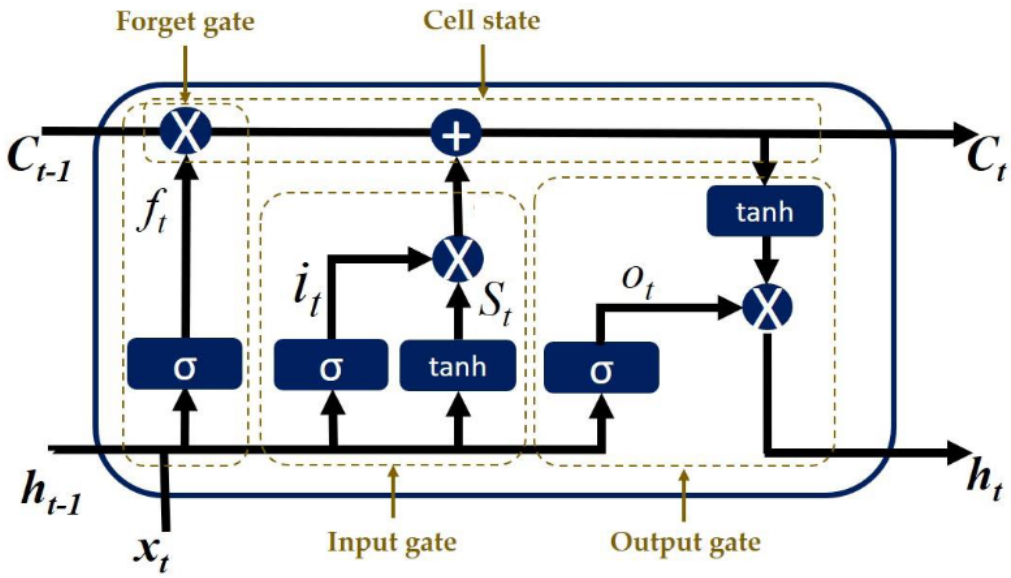
$$\mathbf{o}^{<t>} = \sigma\left(\mathbf{W_{oh}}\mathbf{h}^{<t-1>} + \mathbf{W_{ox}}\mathbf{x}^{<t>} + \mathbf{b_o}\right) \tag{2.18}$$

$$\mathbf{h}^{<t>} = \mathbf{o}^{<t>} \odot tanh\left(\mathbf{C}^{<t>}\right) \tag{2.19}$$

2.2.3 Redes Neurais Convolucionais

O campo de visão computacional é o campo de *machine learning* que processa dados visuais de imagens e vídeos e está cada vez mais presente no cotidiano da sociedade. No entanto, o termo "visão" é um pouco enganoso, uma vez que um computador não "enxerga" que nem um ser humano enxerga. Para um computador, tudo que uma imagem realmente é é apenas uma matriz de números onde cada elemento representa a intensidade de cor do

Figura 15: Arquitetura de uma LSTM.

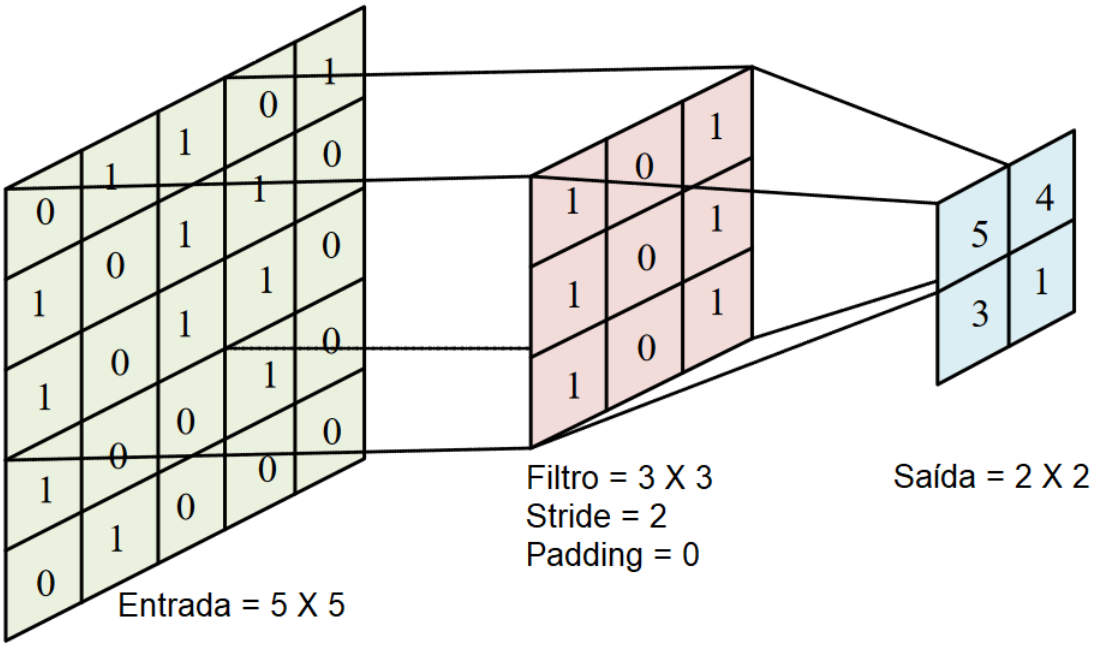


Fonte: Adaptado de [Rajagukguk, Kamil e Lee 2021]

canal correspondente (cor branca para imagens tons de cinza, cores RGB para imagens coloridas) daquele pixel. Como as redes MLP e LSTM são destinadas para processar vetores unidimensionais de entrada, a utilização de tais modelos para o processamento de imagens está longe de ser ideal uma vez que exigiria que as imagens fossem achatadas em vetores, perdendo completamente a informação espacial contida na matriz e rapidamente aumentando o tamanho desses vetores com o aumento da resolução da imagem. Por esses motivos, a principal arquitetura utilizada no processamento de imagens em *deep learning* são as redes convolucionais (*convolutional neural network* - CNN), uma vez que elas são capazes de extrair os atributos das imagens sem destruir a informação espacial e o número de parâmetros da rede é independente da resolução da imagem de entrada [Zhang et al. 2020].

O primeiro componente de uma CNN é o filtro (ou *kernel*) convolucional. Esse filtro é uma matriz quadrada (usualmente 3x5 ou 5x5) que aplica a operação de convolução na matriz de entrada para obter a matriz de saída, conforme ilustrado na Figura 16. Cada pixel de saída (equivalente ao *perceptron*) é obtido pela multiplicação elemento a elemento do filtro e de seu campo receptivo e, após passar por toda a entrada, cada filtro irá gerar sua própria saída, chamadas de *feature maps*. Essas *feature maps* contêm informações semânticas das imagens (como borda, formato, tamanho de objetos, entre outros) e o treinamento de uma CNN consiste justamente em otimizar os pesos de cada filtro de forma que as informações mais relevantes sejam extraídas. Com isso, torna-se possível para a rede aprender as informações espacias das imagens e, ao mesmo tempo, ter um número fixo de parâmetros, independente da resolução da imagem de entrada.

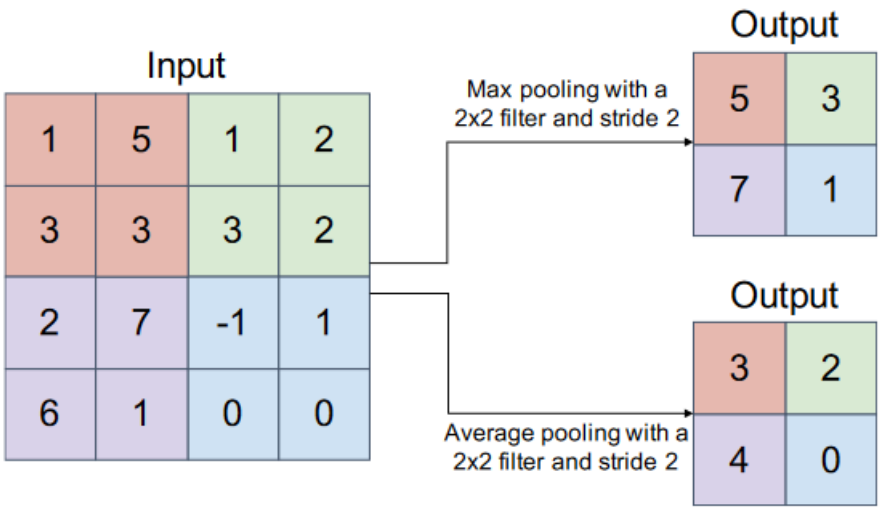
Figura 16: Filtro de Convolução em uma CNN.



Fonte: Adaptado de [Song 2023]

O outro componente importante de uma CNN é o filtro de *pooling* que podem aplicar a média (*average pooling*) ou o máximo (*max pooling*) nos seus campos receptivos (ver Figura 17). Esses filtros são úteis pois são a forma mais simples de reduzir o resolução das *feature maps*, o que essencialmente aumenta o campo receptivo dos filtros convolucionais das camadas subsequentes, permitindo-lhes capturar padrões mais complexos nas imagens sem ter que aumentar drasticamente seu número de parâmetros.

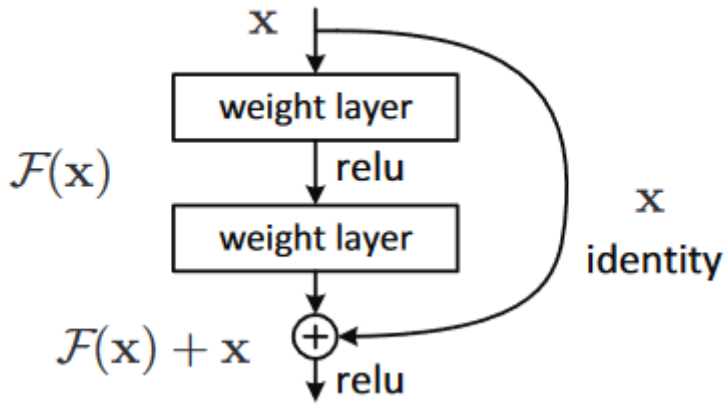
Figura 17: Filtro de *pooling* em uma CNN.



Fonte: [Alapatt et al. 2020]

Embora seja intuitivo pensar que, quanto mais profunda a rede, mais complexos serão os padrões aprendidos e melhor será o desempenho, o aumento excessivo da profundidade da rede dificulta o seu treinamento, podendo na realidade degradar o seu desempenho. Os autores [He et al. 2016] demonstram que, para o mesmo conjunto de dados, o erro de treinamento para uma CNN de 56 camadas é maior do que para uma de 20 camadas. Para contornar esse problema, eles introduziram os blocos residuais, que permitem que o modelo aprenda o mapeamento ótimo $\mathcal{H}(x)$ indiretamente através do mapeamento residual, que eles definem como $\mathcal{F}(x) = \mathcal{H}(x) - x$. Na sequência, o mapeamento $\mathcal{H}(x) = \mathcal{F}(x) + x$ é obtido por meio de uma *skip connection* que geralmente aplica apenas a função identidade, ilustrado na Figura 18. Os autores demonstram que é muito mais fácil para o modelo aprender $\mathcal{F}(x)$ do que $\mathcal{H}(x)$ e citam, por exemplo, um caso hipotético em que o mapeamento ideal fosse a própria identidade. Nesse caso, seria muito mais fácil para o modelo aprender que o $\mathcal{F}(x)$ deve zerar a saída do que aprendar que o $\mathcal{H}(x)$ deve aplicar a função identidade. Modelos que utilizam esses blocos residuais, popularmente conhecidos como as ResNets, já ganharam diveras competições de visão computacional e chegam a ter centenas de camadas, sem que isso prejudique o treinamento.

Figura 18: Blocos residuais.



Fonte: [He et al. 2016]

2.3 Revisão do Estado da Arte

Previsões precisas de irradiância vêm se tornando cada vez mais importantes com o aumento da geração fotovoltaica pelo mundo, impulsionando cada vez mais estudos sobre o tema. Dependendo do objetivo de cada estudo, os modelos podem ser treinados com dados de entrada endógenos, exógenos ou uma combinação de ambos [Rajagukguk, Ramadhan e Lee 2020]. Dados de entrada endógenos são aqueles que são iguais aos dados de saída do modelo, ou seja, seriam a GHI, DNI, DHI, k_t ou a própria potência de geração (quando se deseja prever a geração diretamente com o modelo). Por outro lado, os

dados do tipo exógenos serão diferentes, podendo ser dados de: temperatura, humidade, velocidade/direção do vento, pressão, precipitação, imagem, previsão numérica do tempo, entre outros. Para modelos com o objetivo de gerar previsões de curto prazo (poucos minutos à uma hora) os dados de imagem mais relevantes são as imagens do céu, devido à sua alta resolução espaço-temporal. Já para previsões com horizonte de tempo de uma hora ou mais, as imagens do céu se tornam pouco relevantes e as imagens de satélite são mais comumente empregadas.

Dentro do contexto dos modelos que usam imagens do céu como entrada, que é o foco deste trabalho, existem diversas abordagens. Os autores [Dissawa et al. 2021] utilizaram técnicas de processamento de imagens para rastrear as nuvens e eventualmente determinar a irradiância futura. Através desse processamento, eles conseguem separar os pixels de céu azul, nuvens brancas e nuvens acinzentadas, bem como como determinar a velocidade dessas nuvens com o auxílio de duas câmeras posicionadas em locais distintos. O estudo de [Rajagukguk, Kamil e Lee 2021] converte a imagem de 3 canais (RGB) original em uma imagem de um único canal através da proporção vermelho-azul (RBR) de cada pixel. Após esse processamento é aplicado um limiar de RBR para segmentar a imagem em pixels correspondentes à nuvens e à céu azul e, em seguida, é calculado um índice que indica o percentual de cobertura do céu. Com uma sequência temporal desses índices e com uma rede LSTM o índice de cobertura futuro é previsto e utilizado para determinar a GHI futura. Os autores [Zuo et al. 2022] também utilizam as imagens para obter o índice de cobertura, porém de uma outra forma. Primeiro eles identificam o centro do Sol na imagem a partir do ângulo zenital, campo de visão da câmera, distorção da lente da câmera, entre outros. Com isso, eles usam a intensidade dos pixels ao redor do centro do Sol para determinar se ele está exposto ou não e, em seguida, calculam o índice de cobertura com o RBR de cada pixel.

Todos esses artigos realizam técnicas de processamento de imagens para extrair manualmente seus atributos, porém, estudos que utilizam modelos de *deep learning* para extrair os atributos das imagens automaticamente estão mais relacionados à proposta deste trabalho. Dentre esses destaca-se, por exemplo, o estudo de [Wen et al. 2020] que utiliza redes convolucionais (em particular a ResNet-18) para extrair os atributos das imagens e uma última camada *fully connected* (FC) para prever a GHI de 5 a 10 minutos no futuro. Para cada horizonte de previsão, é treinada uma ResNet separada e é utilizado um *stack* de imagens como entrada, para fornercer informação sobre a movimentação das nuvens à ResNet. Cada *stack* é composto pelos canais vermelhos de 3 imagens, $R_{t-2\tau}$, $R_{t-\tau}$ e R_t onde τ é o horizonte de previsão do modelo. Por exemplo, no modelo de 6 minutos, um *stack* seria composto pelos canais vermelos das imagens: R_{t-12}, R_{t-6} e R_t. Com isso eles conseguem um modelo com desempenho melhor que o modelo persistente, porém, como o modelo persistente será tão melhor quanto menor for o horizonte de previsão, o modelo de 5 minutos é o que tem o FS mais baixo, sendo apenas 5,6% melhor que o persistente. Já o

modelo de 10 minutos consegue ser 17,7% melhor que o persistente.

Outro estudo interessante é o de [Papatheofanous et al. 2022]. Nele os autores primeiro localizam o centro do Sol na imagem, de forma similar à [Zuo et al. 2022], para então criar uma máscara binária do Sol e adicioná-la como um quarto canal à imagem RGB original. Com isso eles conseguem forncer à rede convolucional a informação de onde o Sol está na imagem, que é muito importante para a determinação de GHI e especialmente útil quando ele estiver encoberto por nuvens. A principal CNN utilizada nesse estudo é a ResNet-50, que é utilizada tanto para o *forecasting* quanto para o *nowcasting*. No *nowcasting* a CNN é treinada para prever a GHI correspondente à imagem de entrada e eles demonstram que, adicionando a máscara do Sol, o RMSE é reduzido em aproximadamente 6,5% em comparação com o uso somente da imagem RGB. No *forecasting* os autores treinam a ResNet-50 para prever a GHI de 5, 10 e 15 minutos a frente da imagem de entrada, que é composta por um *stack* de 3 imagens RGB e a máscara. Nos modelos de 5, 10 e 15 minutos, é demonstrado uma melhora de 8,04%, 15,80% e 19,83% no *forecasting skill*, respectivamente.

Os autores [Paletta, Arbod e Lasenby 2021] realizam uma análise bastante interessante sobre o tema. Primeiro eles propõem o uso da métrica de rampa, que é uma métrica de avaliação mais específica para o problema da GHI. Rampas de GHI ocorrem justamente quando as nuvens cobrem o Sol e conseguir prever tal evento é tão importante quanto conhecer o erro médio das previsões. Essa métrica, inicialmente proposta por [Vallance et al. 2017], consiste em realizar uma aproximação linear por partes das previsões e comparar essas rampas lineares com os valores verdadeiros. Além da métrica, os autores treinam diversas arquiteturas para gerar previsões de GHI de 10 minutos. A primeira consiste apenas em utilizar uma CNN para codificar os atributos das imagens, que serão concatenados com os atributos de outros dados (exógenos e endógenos) pertinentes. Esse é o modelo mais simples e não codifica qualquer informação temporal contida em uma sequência de dados. A segunda arquitetura resolve esse problema adicionando uma camada LSTM após a concatenação dos atributos. A terceira arquitetura dispensa o uso da LSTM pois eles substituem a CNN tradicional (2D) por uma 3D-CNN, que é capaz de codificar um vídeo (sequência de imagens) por meio de convoluções 3D, levando em consideração a ocorrência de cada imagem (*frame*) no tempo. A última arquitetura utilizada consiste em uma combinação de 2D-CNN com 2D-LSTM para codificar a sequência das imagens. As 4 arquiteturas alcançam, respectivamente, um *forecasting skill* de 18,1%, 19,2%, 19,7% e 20,4%.

3 Metodologia

3.1 Dataset Folsom

Um dos maiores problemas da previsão de GHI de curto prazo está na definição de um conjunto de dados de referência, em cima do qual diferentes abordagens possam ser comparadas. Para tentar preencher essa lacuna, os autores [Pedro, Larson e Coimbra 2019] publicaram um conjunto de dados publicamente disponível com dados capturados nos anos de 2014, 2015 e 2016, sendo eles:

- Imagens do céu capturadas com uma câmera com lente tipo olho de peixe com uma resolução de 1536x1536 pixels. As imagens são capturadas apenas durante o dia e a cada intervalo de 1 minuto, totalizando mais 760.000 imagens.
- Medidas de GHI, DNI e DHI. Essas medidas são amostradas a cada segundo, porém os autores disponibilizam apenas a média *backward* de um minuto dessas medidas, de forma que a medida em 1:00 corresponde a média das 60 medidas de 0:01 à 1:00.
- Outros dados atmosféricos, como temperatura, pressão, humidade relativa, velocidade do vento e precipitação.

Todos esses dados foram capturados em uma estação solarimétrica situada na cidade de Folsom, Califórnia, com coordenadas $38,642°$ Norte e $121,148°$ Oeste. Por ser um conjunto de dados publicamente disponível e com uma grande quantidade de dados, todas as análises deste trabalho foram feitas em cima dele.

3.2 Processamento dos Dados

3.2.1 Rotulando as imagens com as medidas de GHI

Na Tabela 1 está sendo mostrado como os autores estruturaram os seus arquivos de imagens e irradiância. Dado que o nome dos arquivos das imagens segue o formato ANO/MÊS/DIA HORA:MINUTO:SEGUNDO, é natural rotular cada imagem com a GHI medida no mesmo instante. Contudo, ao fazer isso percebeu-se um comportamento estranho quando, por exemplo, uma imagem que possuía o Sol completamente exposto estava associada à uma GHI menor do que uma imagem que possuía o Sol completamente encoberto, sendo que ambas as imagens foram capturadas em instantes próximos. Esse fenômeno foi amplamente estudado neste trabalho e percebeu-se que ele está associado a

um erro na nomenclatura dos arquivos das imagens. De fato, pode-se ver na Figura 19 que parece haver uma diferença de 11 minutos e 31 segundos entre o horário indicado pelo nome e o horário indicado pelo *date modified* do arquivo. Como o horizonte de previsão é de 5 minutos, essa discrepância têm impactos bastante significativos no desemepenho dos modelos.

Tabela 1 – Arquivos do dataset de Folsom.

Arquivo	**Descrição**
Folsom_irradiance.csv	Medidas de GHI, DNI e DHI com resolução de 1 minuto para os anos de 2014, 2015 e 2016. Todas as medidas estão sincronizadas em :00 segundos.
Folsom_sky_images_YEAR.tar.bz2	Imagens do céu de Folsom no ano YEAR com resolução de aproximadamente 1 minuto. Cada mês do ano posssui sua própria pasta, bem como cada dia do mês. Portanto, cada pasta contém de 500 à 800 imagens, dependendo da estação do ano. Esses arquivos são nomeados no seguinte formato YEARMMDD_HHMMSS.jpg. Nem todas as imagens estão sincronizadas em :00 segundos.

Figura 19: Código de comparação entre o *date modified* e o nome do arquivo.

```
from datetime import datetime
import os
img_path = "2016/10/12/"
img_file = "20161012_224059.jpg"
modified_time = os.path.getmtime(img_path + img_file)
modified_utcdatetime = datetime.utcfromtimestamp(modified_time)
print(f"UNIX Epoch: {modified_time}\nImage File: {img_file}\nUTC timestamp: {modified_utcdatetime}")
```

```
UNIX Epoch: 1476311368.0
Image File: 20161012_224059.jpg
UTC timestamp: 2016-10-12 22:29:28
```

Fonte: O autor

Nas Figuras 20 e 21 também pode ser observado esse fenômeno, onde tem-se uma sequência de imagens na linha superior que corresponde às imagens rotuladas de acordo com o *date modified*, uma sequência de imagens na linha inferior que corresponde às imagens rotuladas de acordo com o *file name* e uma sequência de 5 minutos de medidas de GHI na linha do meio. Além disso, os títulos de cada imagem (Img0, Img1, Img2, etc...) correspondem à ordem das imagens no intervalo que está sendo analisado e estão referenciados com respeito ao *file name*, ou seja, a Img0 corresponde à primeira imagem do *file name* no intervalo, Img1 à segunda e assim por diante. As medidas de GHI foram linearmente interpoladas para facilitar a visualização. Com isso, pode-se perceber que:

- Dia 20/05/2016 13:09:00 - 13:14:00 (Figura 20):

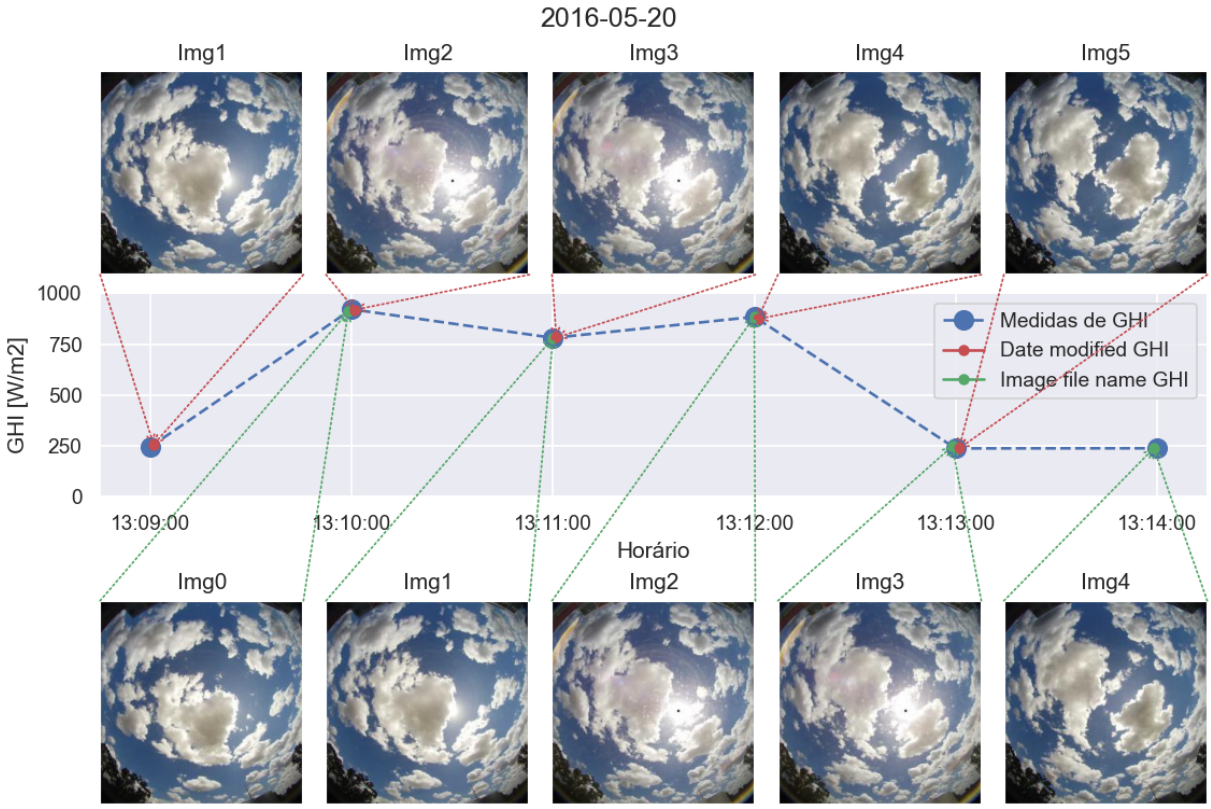
  - O *date modified* da Img1 indica que ela foi capturada às 13:09:00, enquanto que o *file name* indica que ela foi capturada às 13:11:00, o que significa que o *file name* está 2 minutos adiantado em relação ao *date modified*.
  - Há um grande salto de GHI entre 13:09:00 e 13:10:00, o que indica que o Sol foi exposto nesse intervalo. Na sequência do *date modified* isso realmente ocorre, entre a Img1 e Img2, porém, na sequência do *file name* a Img0 é rotulada com uma GHI extremamente alta para uma imagem com o Sol completamente encoberto.
  - Às 13:13:00 a GHI está aproximadamente 800 W/m^2 mais baixa que às 13:10:00, porém, na sequência do *file name*, o Sol está encoberto às 13:10:00 (Img0) e exposto às 13:13:00 (Img3), o que não faz sentido. Já na sequência do *date modified* os resultados são mais coerentes, já que às 13:10:00 o Sol está exposto (Img2) e às 13:13:00 o Sol está encoberto (Img5).
  - Às 13:12:00 a GHI está em aproximadamente 900 W/m^2 e depois despenca para 200 W/m^2, porém a imagem do *date modified* (Img4) está com o Sol encoberto enquanto que a imagem do *file name* (Img2) está com o Sol exposto, o que a princípio indica que o *date modified* está errado nesse caso. Esse fenômeno foi verificado sempre no instante antes de haver uma variação brusca na GHI (para cima ou para baixo) e levanta-se a hipótese de que ele está mais relacionado ao atraso introduzido pelo cálculo da média na GHI do que por um erro no *date modified*.

- Dia 23/04/2016 12:40:00 - 12:45:00 (Figura 21):

  - O *date modified* da Img3 indica que ela foi capturada às 12:40:56, enquanto que o *file name* indica que ela foi capturada às 12:43:00, o que significa que o *file name* está 2 minutos e 4 segundos adiantando em relação ao *date modified*.
  - No horário 12:44:00 a GHI está alta, porém a Img4, que é a imagem correspondente do *file name* nesse horário, está com o Sol encoberto. Já a Img6, correspondente do *date modified*, está com o Sol exposto, o que indica que o *date modified* está mais correto do que o *file name*
  - Poucos segundos antes do salto brusco de GHI às 12:43:00 a imagem do *date modified* parace estar mais errada do que a do *file name*, porém, novamente, esse erro muito provavelmente se deve ao atraso introduzido pela média no cálculo da GHI.

Na Figura 22 é apresentado a diferença média diária, em segundos, entre o *date modified* e o *file name*, que é calculada subtraindo-se o horário indicado no nome do arquivo pelo horário indicado na sua data de modificação, ou seja, um valor positivo indica que o

Figura 20: Comparação entre o *date modified* e o *file name*.

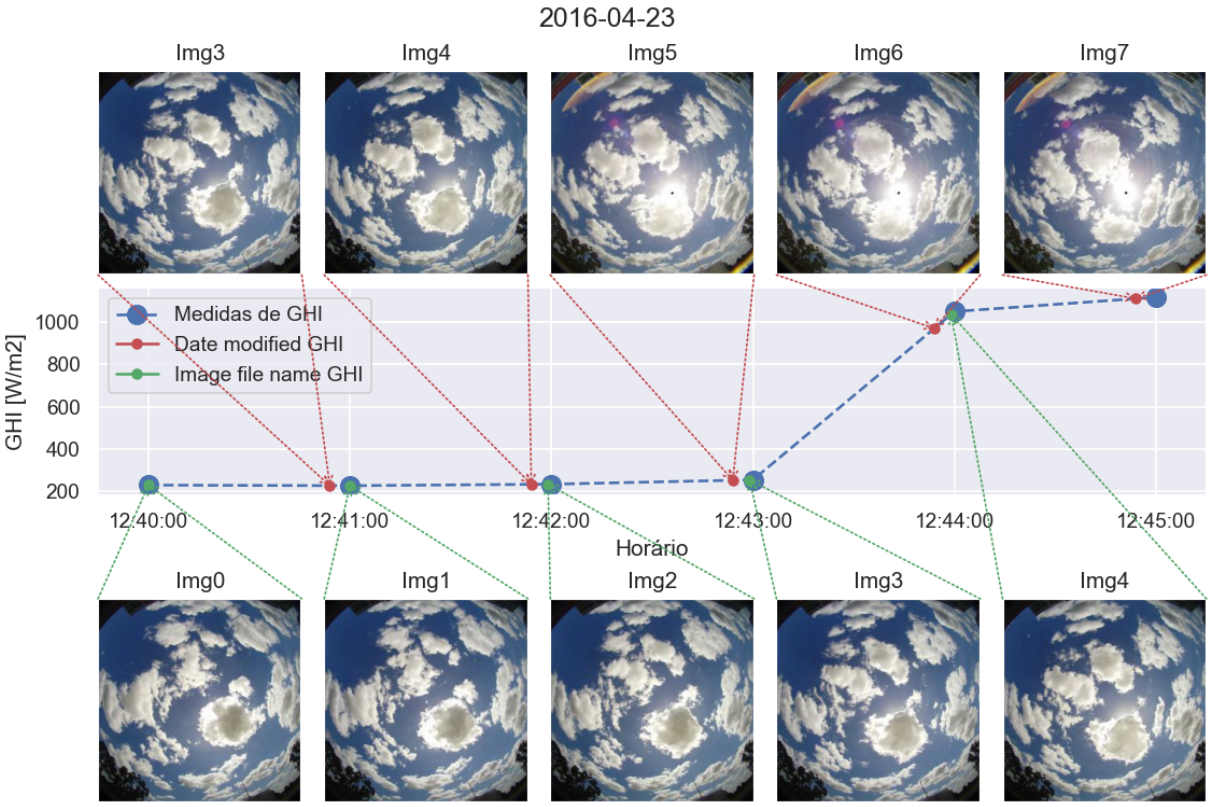


Fonte: O autor

file name está adiantado e um valor negativo indica que está atrasado. Percebe-se que quando iniciaram-se as medidas, em 2014, essa diferença era pequena, porém, perto do mês de Junho, houve um salto que aumentou a diferença. Esse mesmo salto ocorreu no começo de 2015 e no final de 2016, que piorou ainda mais, chegando à 700 segundos de diferença. Ainda é um mistério o porquê disso, mas supõe-se que há uma defasagem entre os *clocks* internos da câmera e do *datalogger* e que esses saltos sejam alguma tentativa de correção automática que na verdade piora a situação.

Ao rotular as imagens com a GHI, também é preciso levar em consideração o fato de que as medidas de GHI estão regularmente espaçadas (sincronizadas em :00 segundos) porém as imagens não estão, especialmente quando se assume o *date modified* como o instante verdadeiro da imagem, onde o sincronismo nos segundos está uniformemente espaçado (ver Figura 23). Para contornar isso, foram testadas duas abordagens: arredondar o *date modified*/*file name* da imagem ao minuto mais próximo (1:30 → 1:00 ou 1:31 → 2:00) ou realizar um *upsampling* nas medidas de GHI através das interpolações linear, spline cúbica ou sinc. A interpolação linear é a mais adequada para os instantes em que a GHI está variando lentamente, porém, para variações extremamente rápidas, fica difícil dizer qual a melhor interpolação uma vez que a GHI é amostrada à uma taxa de 1 min/amostra. Para investigar isso foram analisadas janelas de 7 minutos onde o ΔGHI

Figura 21: Comparação entre o *date modified* e o *file name*.

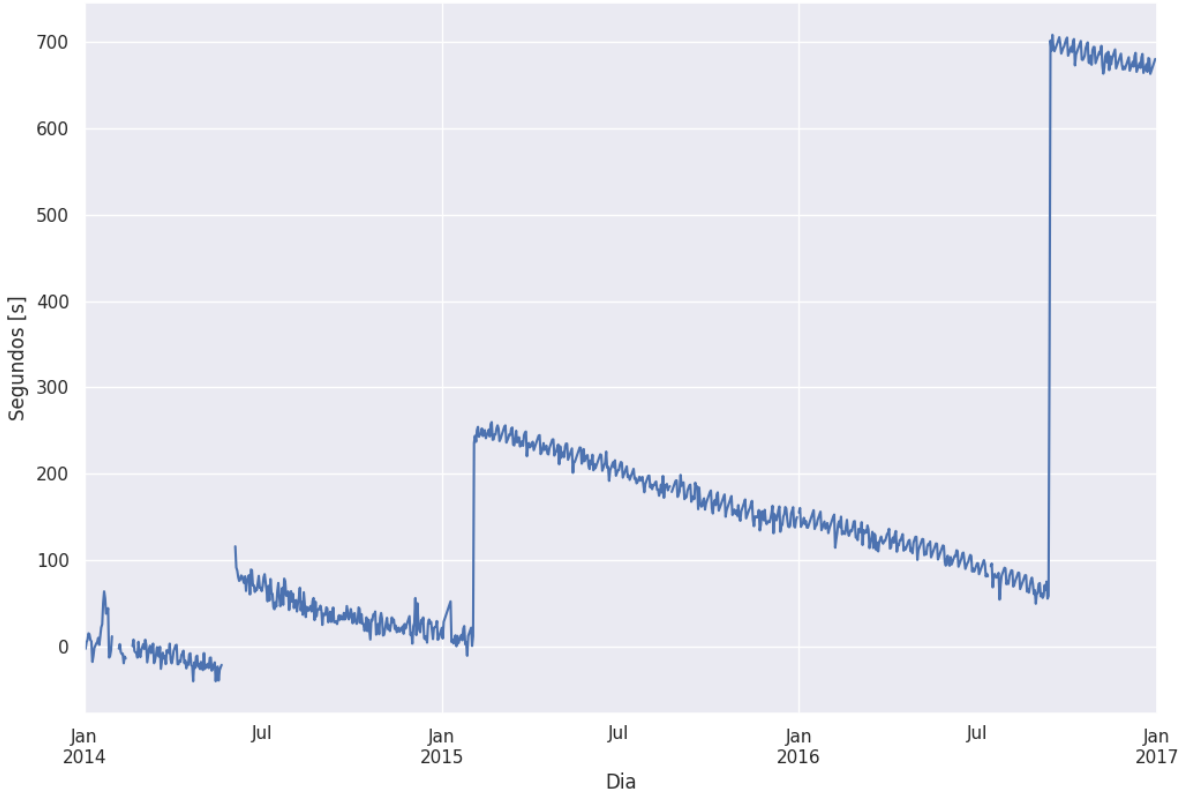


Fonte: O autor

era grande, porém essa variação ocorria lentamente entre a primeira a última amostra da janela. Essas janelas foram então *downsampled* em taxas mais lentas (2 à 7 min/amostra) e o erro entre cada interpolação e as amostras originais foi calculado para determinar qual das três é a mais adequada para reconstruir a GHI nos instantes de variação rápida. Por exemplo, na Figura 24 está sendo apresentada uma janela onde o ΔGHI entre a primeira e última amostra é de aproximadamente 900 W/m^2 em termos absolutos, porém ocorre lentemante ao longo de todo o intervalo. Pode-se observar que o erro médio de cada interpolação é semelhante, indicando que caso a movimentação das nuvens fosse a mesma só que mais rápida, o desempenho de cada interpolação seria muito parecido. Na Figura 25 tem-se o resumo do desempenho das 3 interpolações em diversas janelas, onde cada gráfico representa um *downsampling* diferente (indicado pelo título) e cada ponto mostra o RMSE de 100 janelas onde o ΔGHI era maior ou igual ao valor indicado no eixo x. Pode-se dizer que a interpolação sinc foi a pior das 3, porém por muito pouco, o que indica que o erro é pouco afetado pelo tipo de interpolação e mais pelo ΔGHI mínimo da janela e pela redução na taxa de amostragem.

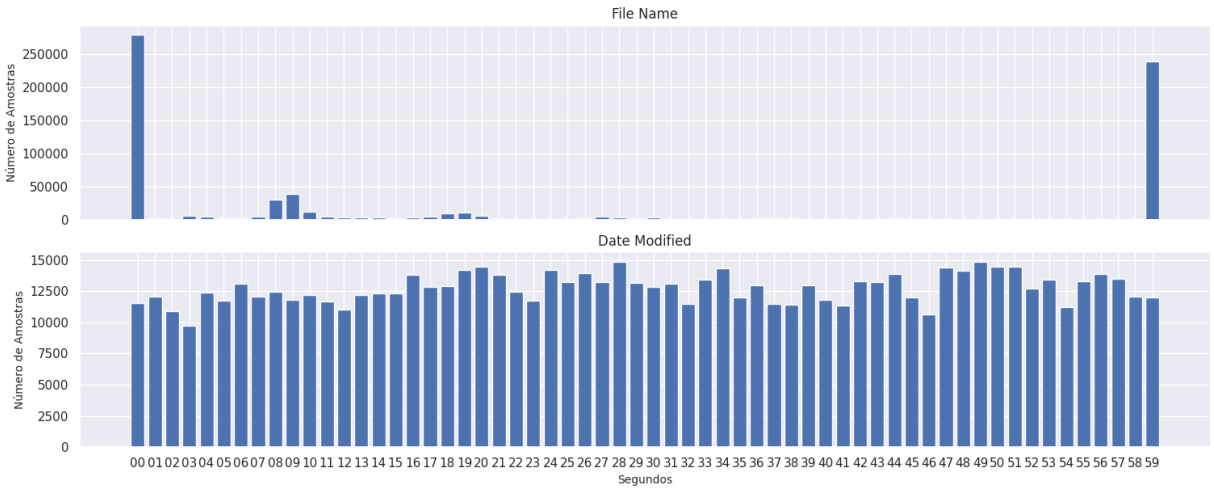
Outro ponto de consideração está no atraso introduzido pelo cálculo da média da GHI. Para mitigiar esse problema, foram aplicados diferentes *timedeltas* nas medidas de GHI conforme ilustrado na Figura 26. Como as medidas estão atrasadas então um

Figura 22: Diferença média diária entre o *file name* e o *date modified*. Calcula-se a diferença entre os dois de todas as amostras do dia e tira-se a média.



Fonte: O autor

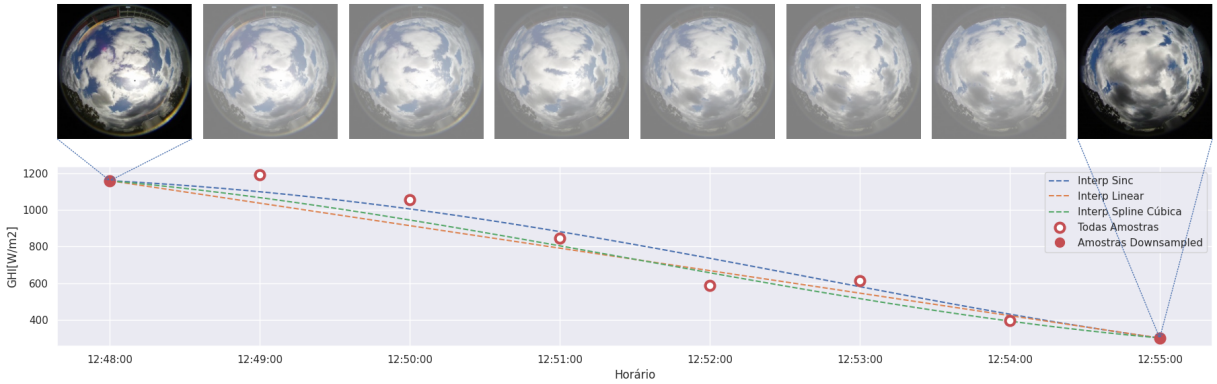
Figura 23: Histograma demonstrando os segundos de captura da imagem de acordo com o *file name* (gráfico de cima) e com o *date modified* (gráfico de baixo).



Fonte: O autor

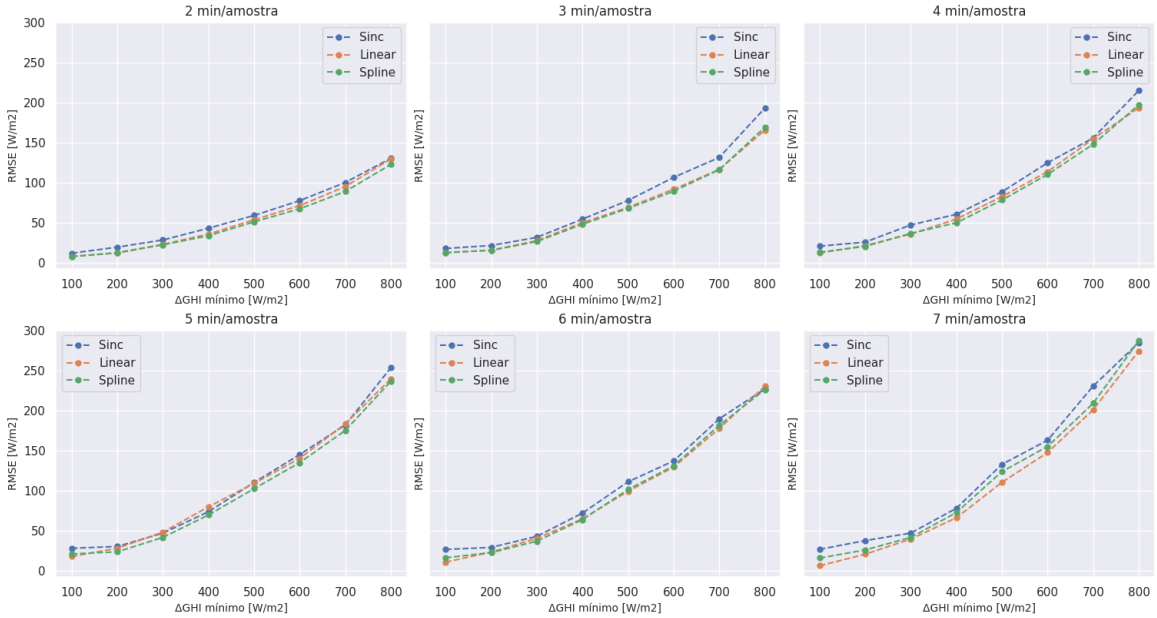
timedelta < 0 seria o mais adequado para mitigar esse atraso, o que foi verificado na Seção 4.2.2.

Figura 24: Janela de 7 minutos com *downsampling* de 7 min/amostra.



Fonte: O autor

Figura 25: Resumo do desempenho das 3 interpolações (linear, spline cúbica e sinc).



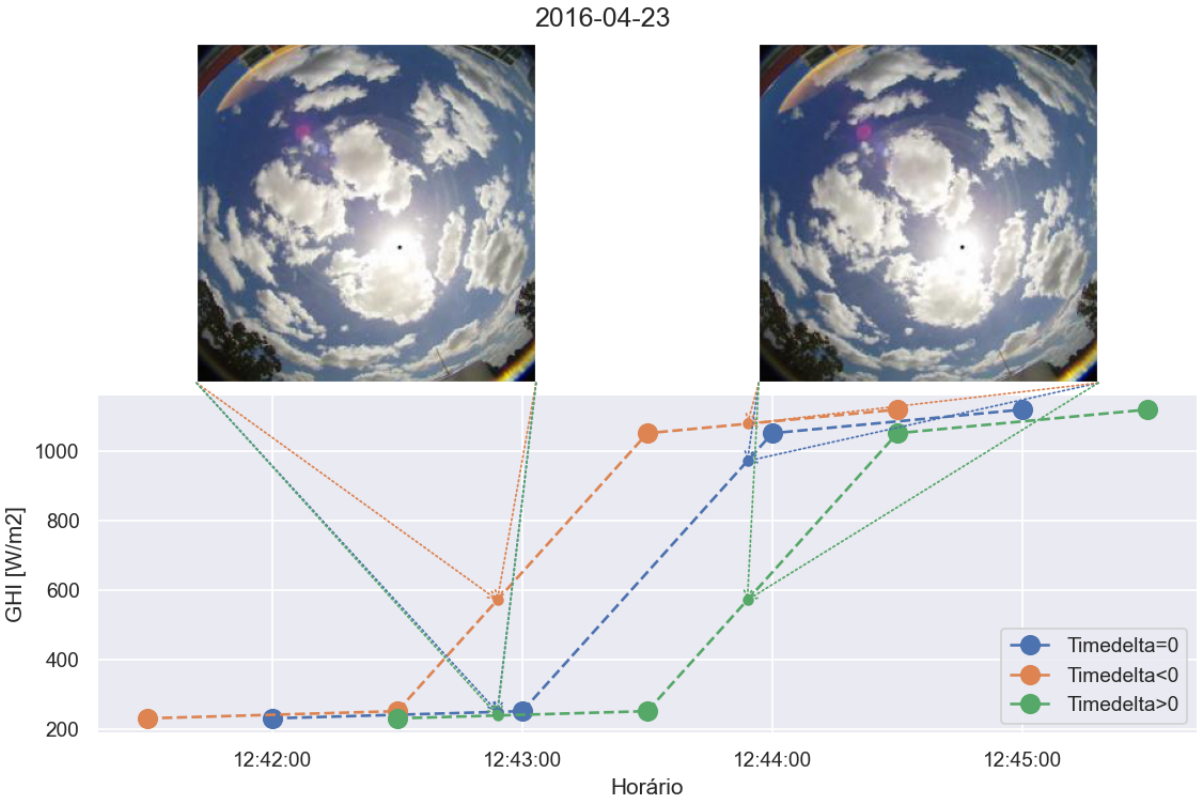
Fonte: O autor

3.2.2 Obtenção da Máscara do Sol

A máscara solar utilzada neste trabalho consiste em um círculo centrado no centro do Sol com valores de pixel igual à 255 dentro do círculo e 0 fora, sendo ela adicionada como um quarto canal às imagens de entrada do modelo. Isso fornece ao modelo a informação sobre a localização do Sol na imagem, que é particularmente útil em instantes onde o Sol está encoberto por muitas nuvens. Para obter essa máscara, primeiro é necessário descobrir o centro do Sol em cada imagem e depois, com o auxílio da biblioteca OpenCV[1], cria-se um círculo centrado nesse ponto. A localização desse ponto foi realizada da mesma forma que em [Papatheofanous et al. 2022], com base nos ângulos azimutal φ_s e zenital θ_s (Figura

[1] https://opencv.org/

Figura 26: Rótulo das imagens para *timedeltas* nulos, positivos e negativos.



Fonte: O autor

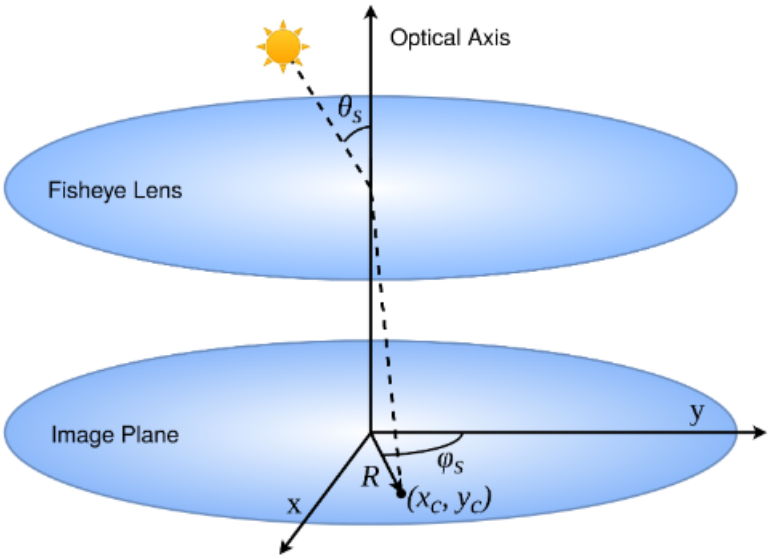
27) da estação solarimétrica de Folsom, que foram obtidos através da biblioteca pvlib[2]. Com esses ângulos, já seria possível projetar o centro do Sol na imagem, porém, devido às distorções introduzidas pela lente *fisheye* da câmera, os resultados estariam incorretos. Para resolver isso, a distância R entre o centro da imagem e o centro do Sol é mapeada com base no tipo de projeção da lente que, embora não tenha sido disponibilizada pelo *dataset*, os autores descobriram na tentativa e erro que a projeção era do tipo estereográfica e que a lente possui uma distância focal $f = 0,48$. Com isso, as coordenadas cartesianas do centro do Sol podem ser obtidas:

$$\begin{aligned} R &= 2f \tan\left(\frac{\theta_s}{2}\right) \\ x_c &= R \sin\left(\varphi_s - 165^\circ\right) \\ y_c &= R \cos\left(\varphi_s - 165^\circ\right) \end{aligned} \tag{3.1}$$

Os 165° em (3.1) são para compensar a orientação da câmera e, para obter o centro do Sol nas coordenadas dos pixels, basta multiplicar x_c e y_c pela metade da largura da imagem. Com isso obtém-se a máscara do Sol sendo mostrada na Figura 28.

[2] https://pvlib-python.readthedocs.io/en/stable/

Figura 27: Localização do centro do Sol na imagem.



Fonte: [Papatheofanous et al. 2022]

3.2.3 Redimensionamento e Corte nas Imagens

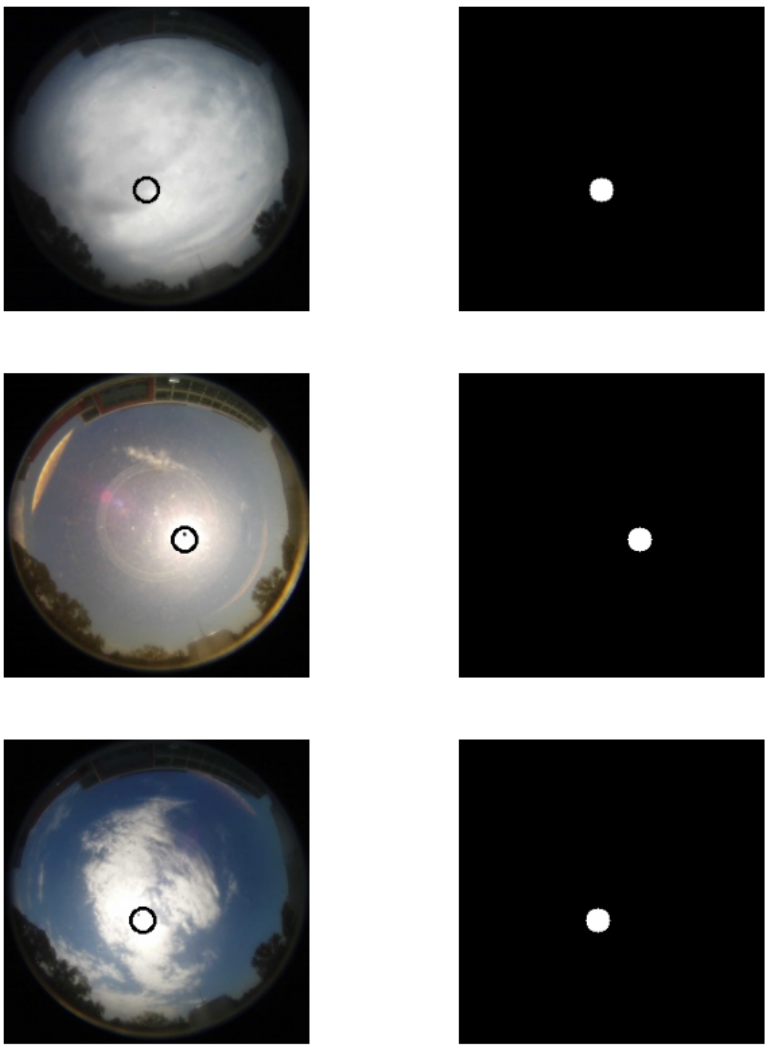
O primeiro redimensionamento das imagens consiste em reduzir sua resolução original de 1536x1536 para 256x256 pixels e salvá-las em um novo arquivo para que o *dataloader* economize tempo na hora de carregar as imagens durante o treinamento dos modelos, já o segundo redimensionamento é realizado durante o treino e os impactos de diferentes resoluções são investigados na Seção 4.2.3. Outro processamento padrão das imagens é o recorte de suas bordas escuras (Figura 29), que é realizado antes do redimensionamento.

3.2.4 Seleção dos Conjuntos de Treino, Teste e Validação

Imagens correspondentes aos anos 2014 e 2015 foram utilizadas como imagens de treino e as de 2016 foram utilizadas como teste, sendo que instantes com um ângulo zenital maior que 80° foram retirados de ambos os conjuntos uma vez que o Sol está muito pouco visível nesses casos. O conjunto validação foi construído a partir do conjunto de treino, com um dia de cada mês sendo aleatóriamente retirado do treino e adicionado à validação até que o tamanho dos dois conjuntos fosse, respectivamente, 90% e 10% do tamanho original do treino. O total de amostras de treino, teste e validação foi de 383.713, 220.755 e 42.802, respectivamente. Além disso, no treinamento de modelos de *forecasting*, foram utilizadas janelas de tempo de 15 minutos para o treino/teste/validação. Para mantê-las consistentes entre si, foram impostas as seguintes condições:

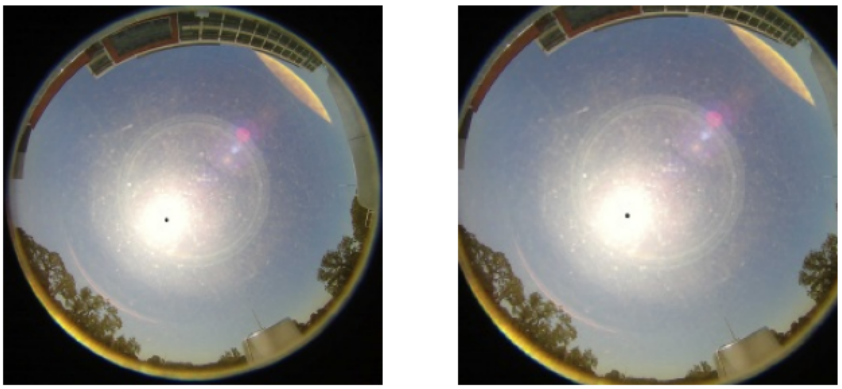
- Toda janela deve, obrigatoriamente, possuir 15 imagens (t-14, t-13, ..., t);

Figura 28: Exemplos da máscara do Sol.



Fonte: O autor

Figura 29: Recorte realizado nas imagens.

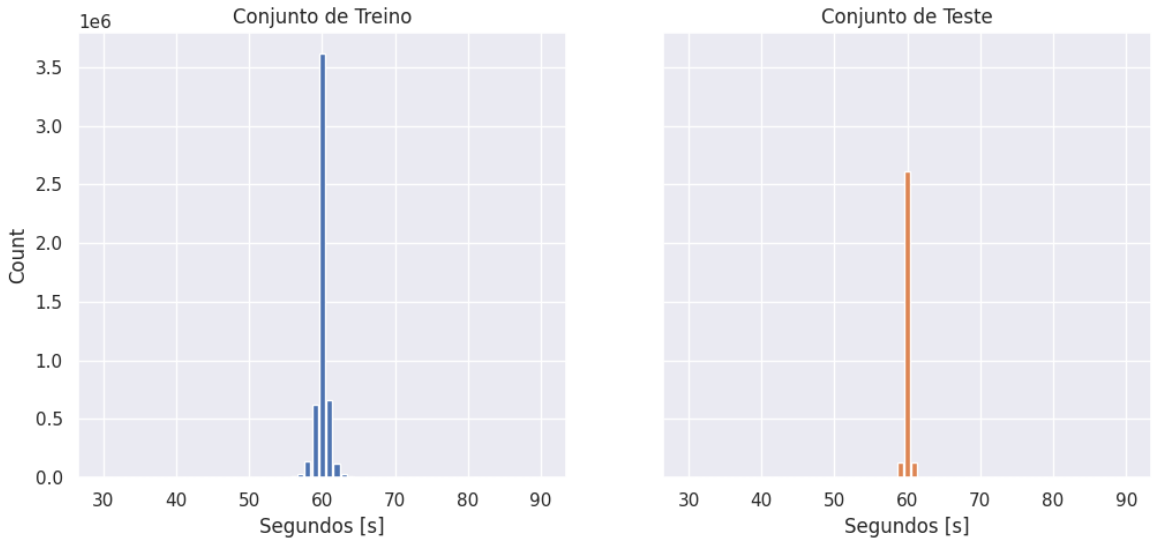


Fonte: O autor

- O intervalo de tempo entre cada imagem subsequente de uma janela deverá ser no máximo 90 segundos e no mínimo 30 segundos (ver Figura 30);

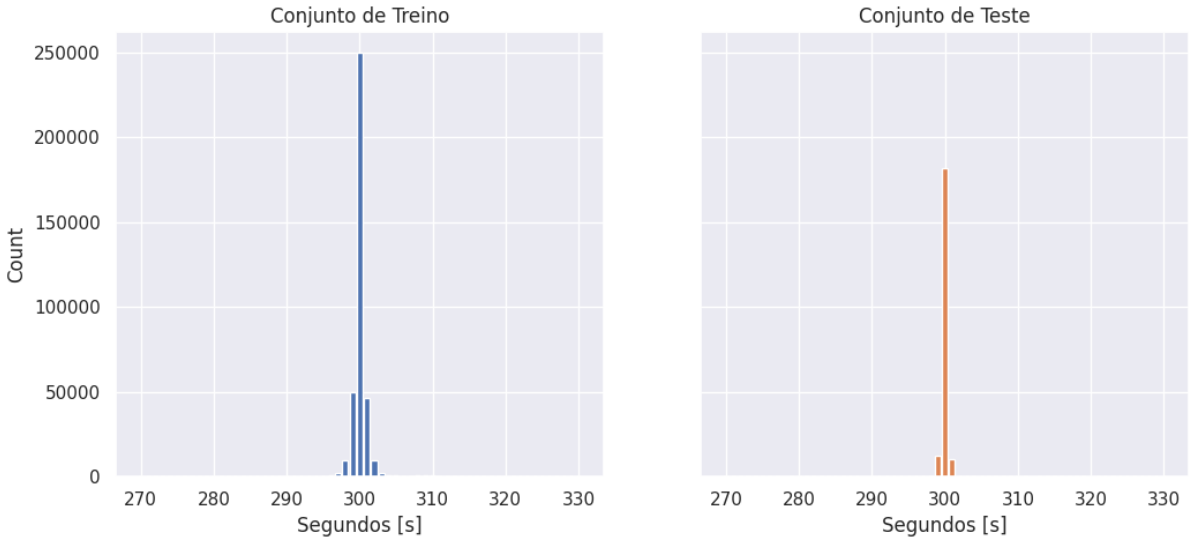
- Deverá haver uma imagem no mínimo 4:30 minutos e no máximo 5:30 minutos à frente da última imagem de cada janela (ver Figura 31).

Figura 30: Distribuição da diferença de tempo entre todas as imagens subsequentes de cada janela.



Fonte: O autor

Figura 31: Distribuição da diferença de tempo entre a última imagem de cada janela a uma imagem aproximadamente 5 minutos à frente.



Fonte: O autor

3.3 Modelos Utilizados

Todos os modelos e experimentos deste trabalho foram criados e treinados em PyTorch[3] no ambiente de execução do Google Colab[4] com uma GPU Tesla T4. Outras bibliotecas padrão de Pyhton também foram utilizadas, como Pandas[5], NumPy[6] e Matplotlib[7].

3.3.1 Modelos de Nowcasting

Na literatura existem diferentes definições para o *nowcasting* de GHI. Neste trabalho será utilizado a mesma definição utilizada pelos autores [Nie et al. 2023] que consiste no uso das imagens para prever a GHI atual. A utilidade do *nowcasting* vai muito além da predição atual de GHI sendo que, neste trabalho, foi útil para:

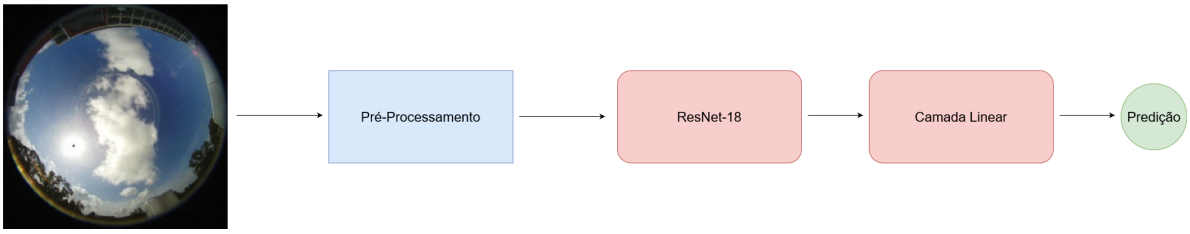
- Descobrir qual a melhor variável alvo para as imagens (a própria GHI ou o índice de céu claro k_t);
- Diagnosticar os problemas encontrados no *dataset* e discutidos na Seção 3.2.1.
- Ganhar *insights* sobre o impacto que a resolução das imagens e a máscara do Sol têm no desempenho do modelo.
- Obter bons *embeddings* que representem as imagens em forma de vetores, que poderão ser processados depois por uma LSTM para realizar o *forecasting*.

A estrutura geral desses experimentos está ilustrada na Figura 32, onde cada bloco representa:

- Pré-Processamento: Redimensionamento e corte nas imagens, conforme descrito na Seção 3.2.3.
- ResNet-18: ResNet18 com a última camada de classificação removida (Figura 33). É treinada *end-to-end* em todos os experimentos de *nowcasting* e inicializada com os parâmetros vencedores da competição ImageNet.
- Camada Linear: Camada linear com 512 entradas e 1 saída que substituí a camada de 512 entradas e 1000 saídas da ResNet-18 padrão (Figura 34).
- Predição: Índice de céu claro k_t atual ou a própria GHI atual.

[3] https://pytorch.org/
[4] https://colab.research.google.com/
[5] https://pandas.pydata.org/
[6] https://numpy.org/
[7] https://matplotlib.org/

Figura 32: Estrutura dos modelos de *nowcasting*.



Fonte: O autor

3.3.2 Modelos de Forecasting

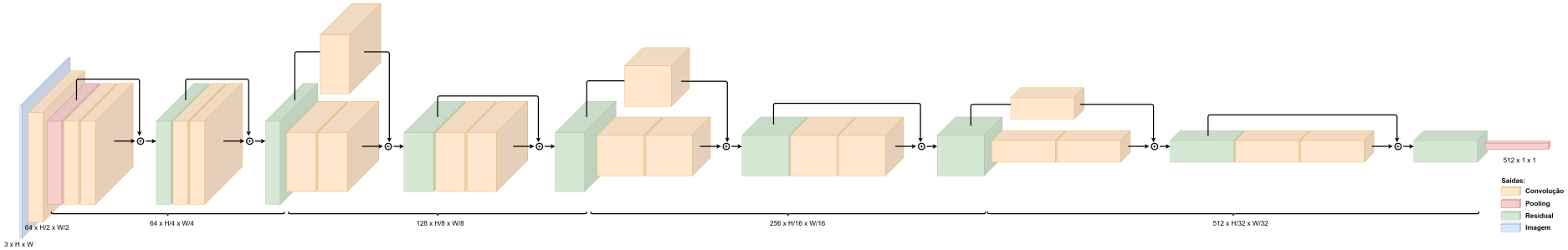
O primeiro modelo de *forecasting* consiste em utilizar a ResNet18 treinada para o *nowcasting* para extrair os *embeddings* das imagens de treino/teste/validação de forma *offline*. Ou seja, os melhores pesos de *nowcasting* da ResNet18 são **congelados** e os *embeddings* são extraídos e salvos em um arquivo com formato .parquet. Em seguida, esses *embeddings* são processados para formarem sequências equivalentes à uma janela de tempo de 15 minutos, onde cada sequência corresponde à uma amostra que será utilizada para treinar uma rede LSTM+MLP. Todas as sequências obtidas deverão obedecer às 3 condições descritas na Seção 3.2.4, ou seja:

- Serão formadas por 15 *embeddings* que, em média, estarão espaçados de 1 minuto (t-14, t-13, ..., t).
- Serão rotuladas com uma variável alvo que, em média, estará 5 minutos à frente do último *embedding* da sequência (t+5).

Essa variável alvo só será prevista pelo último *hidden state* da LSTM, correspondente à saída do último *time step* (t). Porém, em vez de prever o rótulo com uma camada linear, adiciona-se uma camada oculta de 512 *perceptrons*, conforme ilustrado na Figura 35.

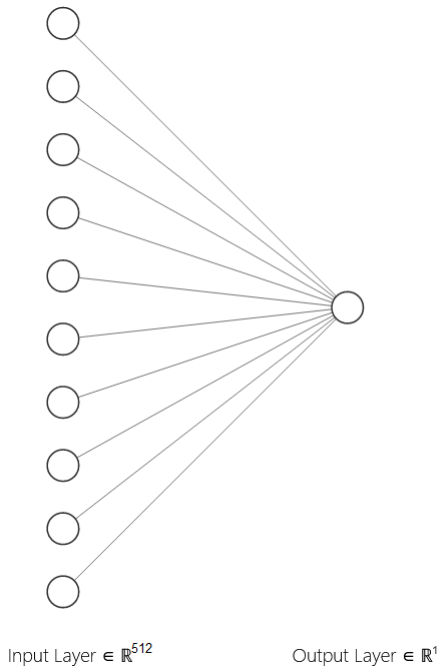
O segundo modelo de *forecasting* é bastante similar ao primeiro e consiste apenas em descongelar os pesos da ResNet18. Dessa forma, os *embeddings* serão extraídos e atualizados à cada época, e a amostra de treino/teste/validação será composta por uma sequência de imagens que serão processadas pela ResNet18, uma a uma, para gerar uma sequência de *embeddings* de saída, que por sua vez serão processados por uma LSTM+MLP da mesma forma que no primeiro modelo.

Figura 33: ResNet-18 com a última camada removida.



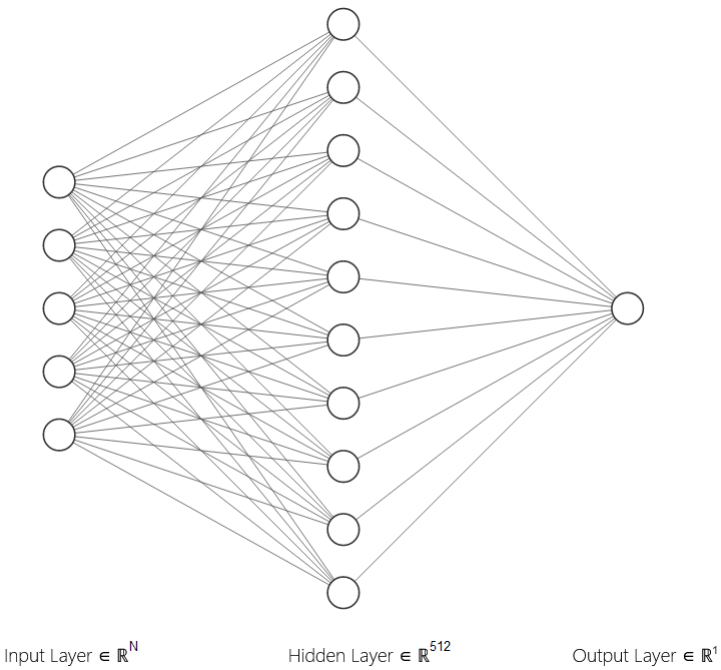
Fonte: O autor

Figura 34: Última camada linear do modelo de *nowcasting*.



Fonte: O autor

Figura 35: Últimas camadas do modelo de *forecasting*. N representa o tamanho do *hidden state* da LSTM.



Fonte: O autor

4 Experimentos e Resultados

4.1 Modelo Persistente

Conforme discutido na Seção 2.1.2, o modelo persistente é o modelo mais simples possível, pois gera suas previsões apenas com as medidas de GHI/k_t atuais. Sendo assim, o resultado de qualquer experimento/modelo de *forecasting* deve sempre ser comparado ao desempenho do modelo persistente, para que seja possível averiguar o quão bom o novo modelo realmente está. No contexto de previsão de GHI, existem dois possíveis modelos persistentes: o tradicional (2.2) e o *smart persistence* (2.3). O modelo tradicional apenas prevê que a GHI futura será igual à GHI atual. Já o modelo *smart persistence* prevê que o k_t futuro será igual ao k_t atual para que a GHI futura seja obtida a partir dele e da GHI de céu claro futura, que dependem do modelo de céu claro escolhido. Neste trabalho, foram investigados os 3 modelos de céu claro disponíveis na biblioteca pvlib[1]: Haurwitz [Haurwitz 1945], Ineichen [Ineichen e Perez 2002] e Simplified Solis [Ineichen 2008]. A Tabela 2 apresenta o desempenho dos 4 modelos persistentes para a previsão da GHI de 5 minutos no futuro.

Tabela 2 – Comparação do desempenho no conjunto de teste dos modelos persistentes na previsão de 5 minutos.

Modelo	***RMSE [W/m²]***
Persistence	72,519
Smart Persistence (Haurwitz)	71,678
Smart Persistence (Ineichen)	71,675
Smart Persistence (Simplified Solis)	71,668

Percebe-se que o modelo com o melhor desempenho é o *smart persistence* utilizando o Simplified Solis. Todavia, a diferença de desempenho é bastante pequena, principalmente entre os 3 modelos *smart persistence*. Isso faz sentido, já que (2.3) pode ser re-escrita da seguinte forma

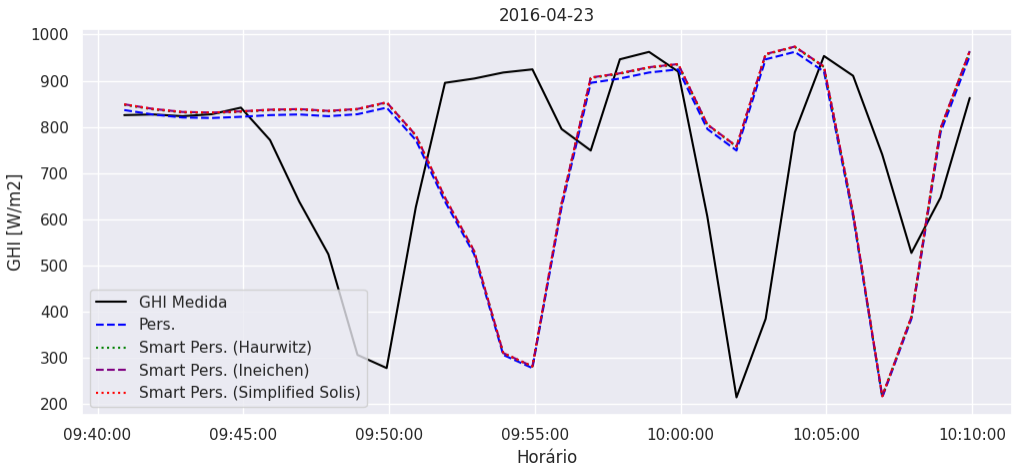
$$\mathrm{GHI}(t+\Delta t) = \mathrm{GHI}(t) \cdot \frac{\mathrm{GHI}_{cs,\mathrm{model}}(t+\Delta t)}{\mathrm{GHI}_{cs,\mathrm{model}}(t)} \tag{4.1}$$

onde $\mathrm{GHI}(t)$ é a previsão do modelo persistente tradicional e $\frac{\mathrm{GHI}_{cs,\mathrm{model}}(t+\Delta t)}{\mathrm{GHI}_{cs,\mathrm{model}}(t)}$ é um número muito próximo de 1 para $\Delta t = 5$ minutos, independente do modelo de céu claro utilizado. Além disso, se t ocorre antes do pico da GHI de céu claro (perto do meio dia),

[1] https://pvlib-python.readthedocs.io/en/stable/

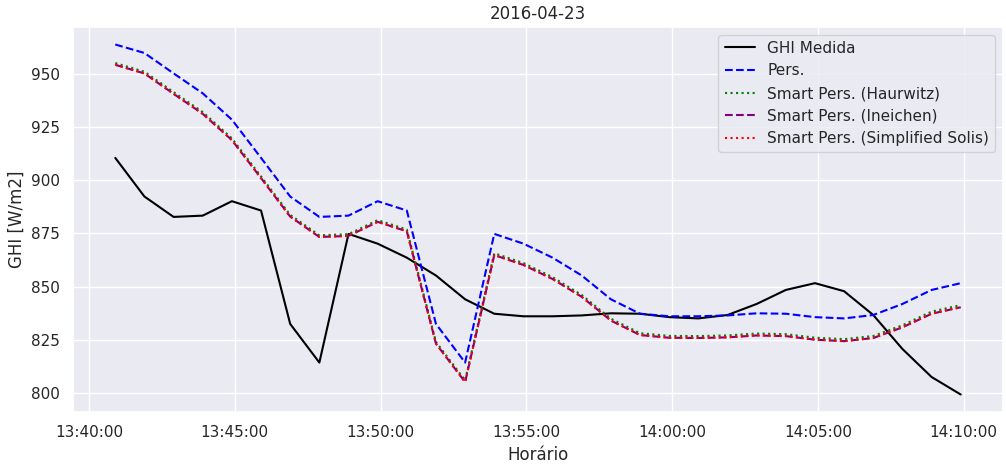
então a fração será um número levemente maior que 1 e levemente menor se t ocorre depois do pico. Isso fica ainda mais evidente com as Figuras 36 e 37, onde as previsões do *smart persistence* dos 3 modelos de céu claro estão, respectivamente, maiores e menores que as do persistente tradicional. Embora a diferença seja pequena, o modelo de céu claro utilizado nos experimentos que seguem será o Simplified Solis, já que ele apresentou o menor RMSE.

Figura 36: Previsões de 5 minutos dos modelos persistentes em um horário antes do pico da GHI de céu claro. Percebe-se que as previsões dos modelos *smart persistence* estão maiores que do persistente tradicional.



Fonte: O autor

Figura 37: Previsões de 5 minutos dos modelos persistentes em um horário depois do pico da GHI de céu claro. Percebe-se que as previsões dos modelos *smart persistence* estão menores que do persistente tradicional.



Fonte: O autor

4.2 Modelos de Nowcasting

Os modelos de *nowcasting*, treinados de acordo com o que foi descrito na Seção 3.3.1, tiveram dois objetivos principais:

- Descobrir quais os hiperparâmetros/configurações ideais de treinamento;
- Extrair bons *embeddings* das imagens para serem utilizados no treinamento dos modelos de *forecasting*.

Ao todo, foram realizados 5 experimentos. Em cada um deles, foram testados os diferentes hiperparâmetros apresentados na Tabela 3, **em que os valores são testados apenas no experimento indicado, caso contrário é utilizado o valor padrão**. Por exemplo, no primeiro experimento, foram treinados dois modelos: um com variável alvo GHI e outro com o k_t. Os outros hiperparâmetros foram mantidos em seus valores padrão em ambos os modelos.

Além disso, a taxa de aprendizado é reduzida para 0,0001 a partir da vigésima época nos 4 primeiros experimentos.

Após o experimento 5, é treinado um **último** modelo de *nowcasting* para extraír os *embeddings* das imagens, com os hiperparâmetros descritos na Seção 4.2.5.

4.2.1 Experimento 1 - Impacto da Variável Alvo

Este experimento tem como objetivo definir qual é a variável alvo mais adequada para as imagens, o índice de céu claro k_t ou a própria GHI. Pela Figura 38 percebe-se que o desempenho no conjunto de teste é significativamente melhor no modelo que prevê o k_t (foi feita a conversão para a escala de GHI na figura), mesmo com um desempenho similar no treino. A principal hipótese para isso é que modelos treinados para prever a GHI diretamente precisam aprender à codificar a posição do Sol com muita precisão para conseguir generalizar suas previsões. Por exemplo, na Figura 39 são mostradas as curvas da GHI e do k_t para um dia de cada estação do ano e as imagens correspondentes ao pico da GHI nesses dias. Percebe-se que a única diferença entre as imagens é um pequeno deslocamento na posição do Sol, que provoca mudanças radicais na curva da GHI, forçando o modelo a aprender algo que, embora seja fácil de identificar em dias de céu claro, pode ser bem complicado em dias nublados ou parcialmente nublados. Já na curva do k_t esse pequeno deslocamento pouco afetou o valor medido, o que facilita no aprendizado do modelo.

Também foi verificado o impacto que diferenças em relação à distribuição dos dados de treino tiveram nos dois modelos. Para isso, foram aplicadas diversas transformações nas imagens do conjunto de teste, ilustradas na Figura 40, para que ambos os modelos

pudessem gerar suas previsões nesse conjunto transformado. Os resultados apresentados na Tabela 4 demonstram que o modelo treinado com o k_t é muito mais robusto à mudanças no conjunto de dados e, em casos como o *horizontal flip*, chega a ter um desempenho melhor que o modelo treinado com a GHI tem no conjunto de teste sem nenhuma transformação.

Tabela 3 – Hiperparâmetros utilizados nos dados de treino/teste/validação. Os valores são testados apenas no experimento indicado, caso contrário é utilizado o valor padrão.

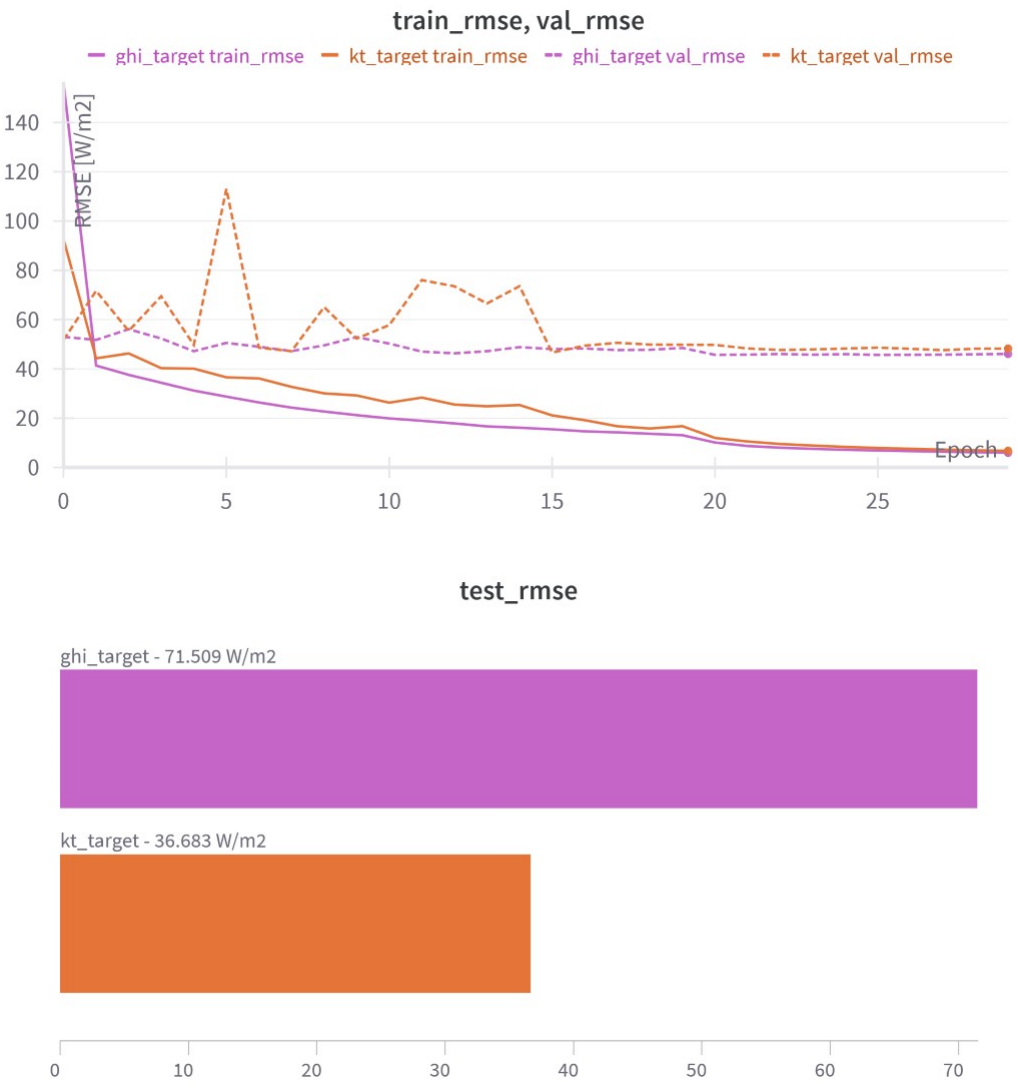
Hiperparâmetro	***Valores Testados***	***Valor Padrão***
(Exp. 1) Variável alvo	k_t GHI	k_t
(Exp. 2) Data das imagens	Date Modified File Name	Date Modified
(Exp. 2) Interpolação das medidas de GHI	Linear Spline Cúbica Arredondamento	Linear
(Exp. 2) Timedelta em segundos nas medidas de GHI	[-60: 60: 5]	-30
(Exp. 3) Resolução da imagem	2^nx2^n, $n \subset [0: 8: 1]$	64x64
(Exp. 4) Máscara do Sol	Não Sim	Não
(Exp. 5) Taxa de aprendizado	Figura 45	0,001
(Exp. 5) Batch size	Figura 45	128
(Exp. 5) Número de épocas	Figura 45	30
(Exp. 5) Otimizador	Figura 45	Adam
(Exp. 5) Inicialização de pesos pré-treinados no ImageNet	Figura 45	Sim
(Exp. 5) Weight decay	Figura 45	0

4.2.2 Experimento 2 - Impacto do Timestamp das Imagens

O segundo experimento consiste em diagnosticar os problemas presentes no dataset e verificar se as hipóteses levantadas na Seção 3.2.1 fazem sentido. Foram testados os impactos que as seguintes configurações tiveram no desempenho do modelo:

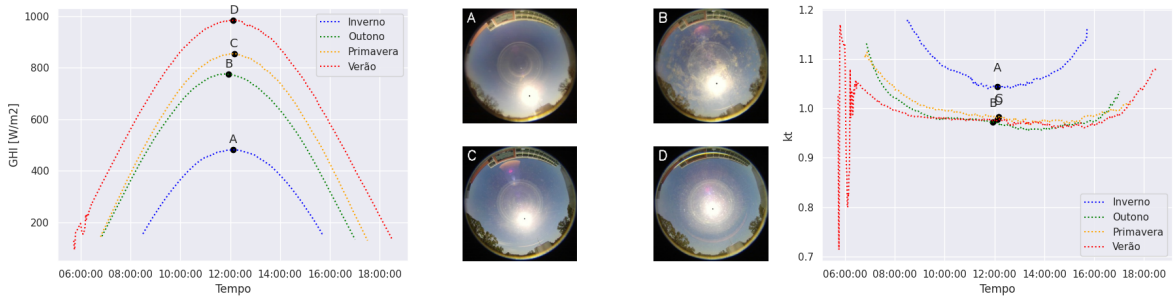
- Rotular com o *date modified* vs rotular com o *file name*.
- Deslocar as medidas de GHI por um *timedelta*.
- Aplicar uma interpolação linear, *spline* cúbica ou arredondar para a medida de GHI mais próxima na hora de rotular as imagens.

Figura 38: Curvas de treino, teste e validação.



Fonte: O autor

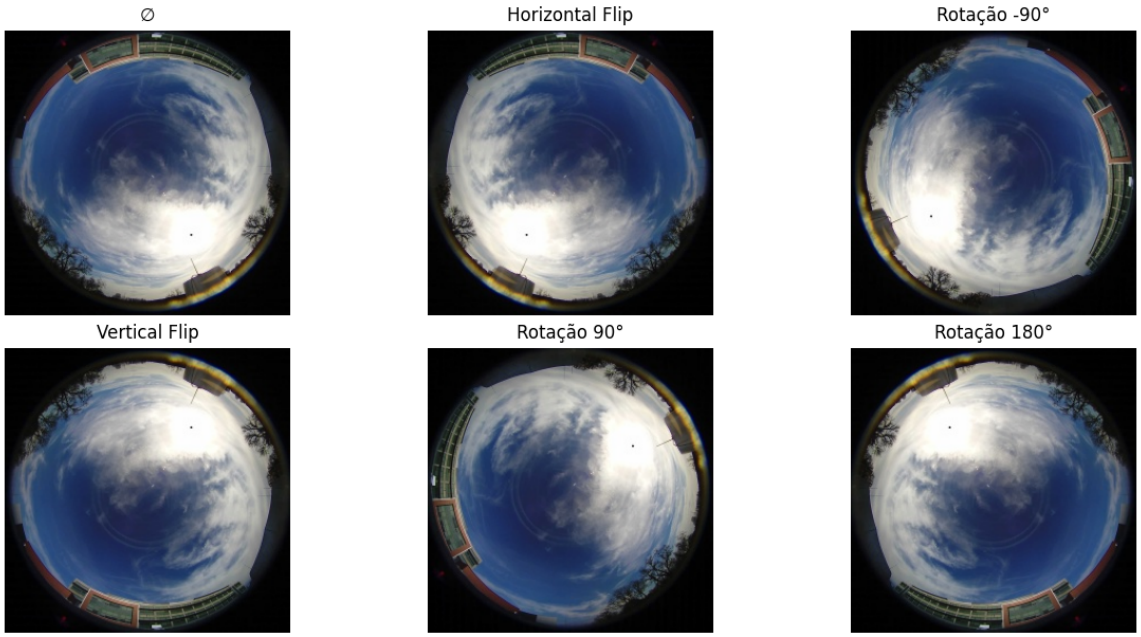
Figura 39: Curvas da GHI (esquerda), imagens do céu (meio) e curvas do k_t (direita) para um dia de cada estação do ano.



Fonte: O autor

Inicialmente, foi investigado qual combinação que mais reduziria o RMSE no conjunto de teste a partir dessas 3 configurações. Para isso, foi treinado e avaliado um

Figura 40: Transformações aplicadas no conjunto de teste.



Fonte: O autor

Tabela 4 – Desempenho no conjunto de teste dos dois modelos para diferentes transformações nas imagens.

	Transformação	***RMSE [W/m^2]***	***Aumento***
GHI	∅	71,51	-
	hflip	195,94	↑ 174,00%
	$-90°$	232,17	↑ 224,67%
	vflip	249,46	↑ 248,84%
	90°	249,96	↑ 249,54%
	180°	254,85	↑ 256,38%
k_t	∅	36,68	-
	hflip	63,13	↑ 72,11%
	$-90°$	77,68	↑ 111,77%
	vflip	90,87	↑ 147,73%
	90°	94,53	↑ 157,71%
	180°	95,80	↑ 161,17%

modelo para cada combinação possível, **que era aplicada tanto no conjunto de treino quanto no conjunto de teste**. Os resultados apresentados na Figura 41 mostram que a melhor combinação possível é: *date modified*; interpolação linear; *timedelta* de -30 segundos. Esses resultados indicam que há, de fato, um erro na nomenclatura automática dos arquivos e que o atraso introduzido pelo cálculo da média nos dados de GHI são mitigados com um $timedelta < 0$, confirmando as hipóteses levantadas na Seção 3.2.1.

Das 3 configurações, a que mais parece afetar o desempenho é o *date modified*

vs *file name*. Para investigar ainda mais o impacto dessa configuração, foram treinados modelos com 4 combinações de treino/teste diferentes (mantendo as outras configurações/hiperparâmetros nos seus valores padrão):

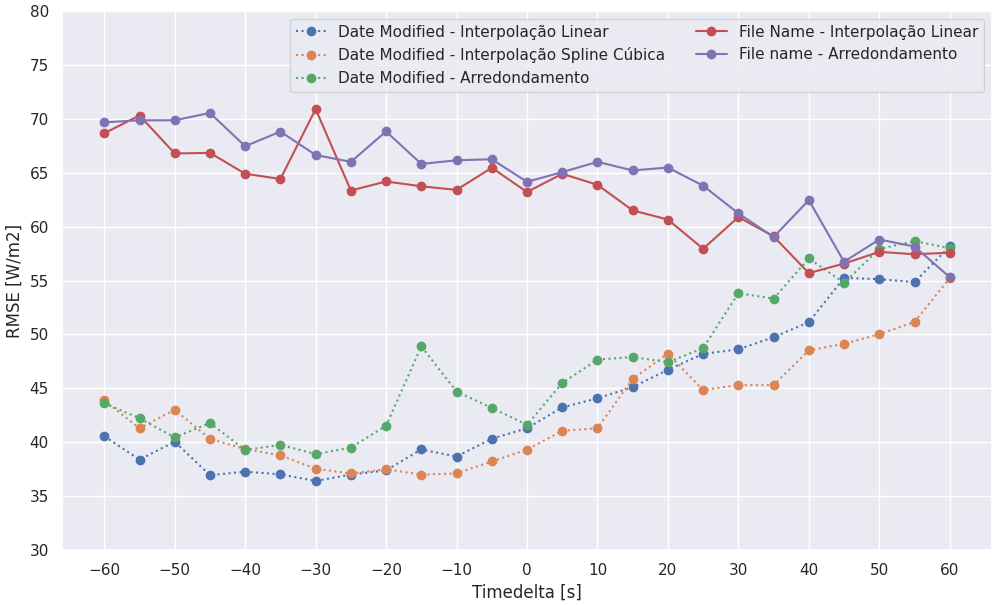
- File name / File name: o conjunto de treino e de teste são rotulados com o *file name*;
- File name / Date modified: o conjunto de treino é rotulado com o *file name* e o de teste com o *date modified*;
- Date modified / File name: o conjutno de treino é rotulado com o *date modified* e o de teste com o *file name*;
- Date modified / Date modified: conjunto de treino e de teste são rotulados com o *date modified*.

Pela Figura 42, percebe-se que o desempenho de teste é sempre melhor quando ele é rotulado com o *date modified* e quando o conjunto de treino é rotulado com o *date modified*. Essa melhora no desempenho é ainda mais significativa quando ambos os conjuntos são anotados com o *date modified*, indicando mais uma vez que as imagens foram incorretamente nomeadas. Outro ponto interessante de se observar é que, a princípio, esperaria-se que o desempenho no teste seria sempre melhor quando o conjunto de treino fosse corretamente rotulado, porém, nota-se que o desempenho de teste foi melhor na combinação File name / Date modified do que na combinação Date modified / File name. Uma possível razão para isso seria que a discrepância entre o *date modified* e *file name* no conjunto de treino (2014 e 2015) é menor que no conjunto de teste (2016), conforme mostrado na Figura 22. Isso significa que, quando os conjuntos de treino/teste são anotados com o *file name*/*date modified*, respectivamente, o modelo ainda consegue aprender alguns atributos relevantes sobre as imagens que serão corretamente avaliados no teste. Já quando os conjuntos de treino/teste são anotados com o *date modified*/*file name*, respectivamente, o modelo irá aprender atributos mais relevantes sobre as imagens, porém eles serão incorretamente avaliados no teste, dando a falsa impressão de que o modelo está ruim.

4.2.3 Experimento 3 - Impacto da Resolução da Imagem

Esse terceiro experimento consiste em verificar qual o desempenho do modelo treinado com diferentes resoluções: 1x1, 2x2, 4x4, 8x8, 16x16, 32x32, 64x64, 128x128 e 256x256. A Figura 43 mostra que o RMSE no conjunto de teste decresce exponencialmente até os 64x64 pixels, após o qual o desempenho estabiliza.

Figura 41: RMSE de teste para diferentes configurações de *timedeltas* aplicados nas medidas de GHI, data da imagem e interpolação/arredondamento das medidas de GHI. As configurações utilizadas no treino e no teste foram as mesmas para a obtenção desse gráfico.



Fonte: O autor

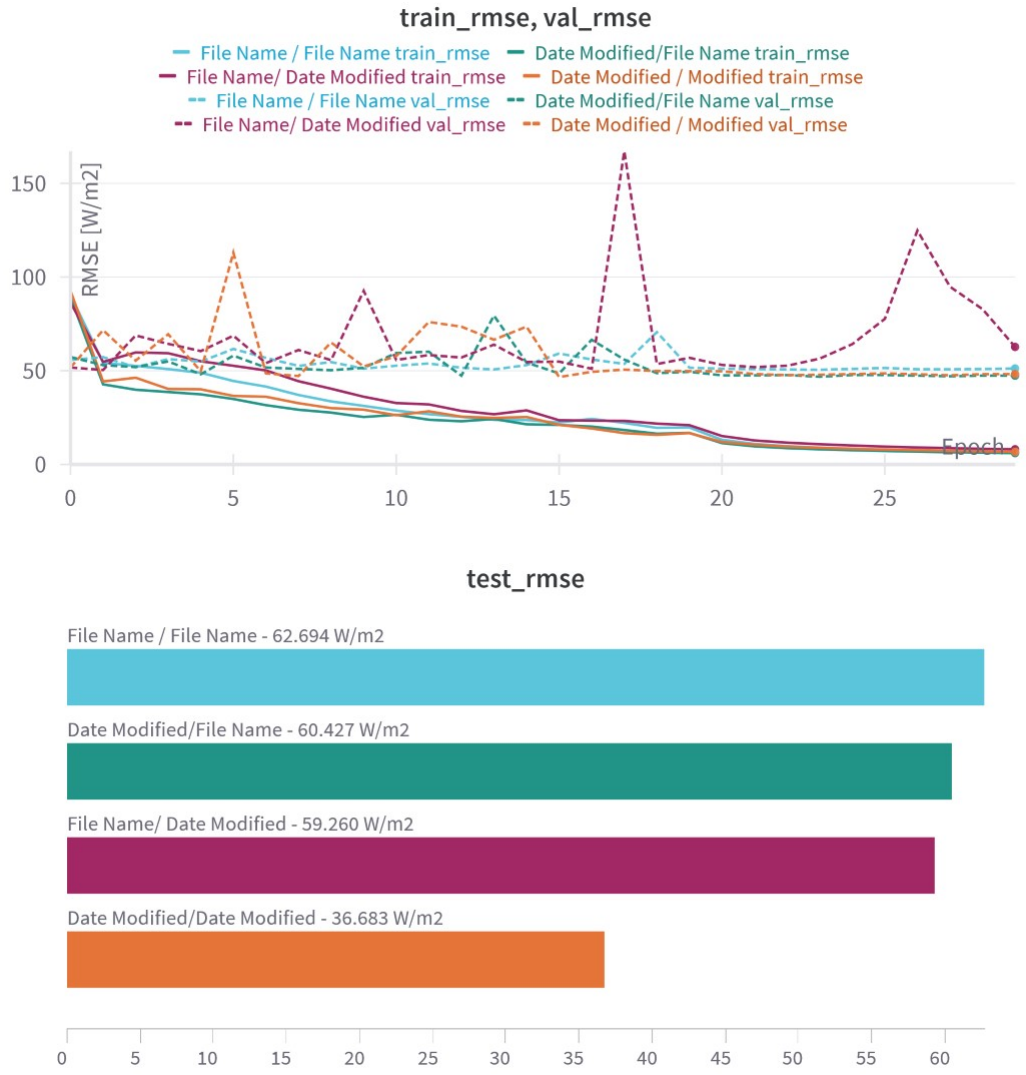
4.2.4 Experimento 4 - Impacto da Máscara do Sol

A máscara do Sol utilizada nesse experimento consiste em um círculo branco centrado no centro do Sol, que foi determinado de mesma forma que descrito na Seção 3.2.2. Essa máscara é então adicionada como um quarto canal à imagem, porém, como a ResNet18 foi projetada para um *input* de três canais, a primeira camada convolucional foi alterada para processar um *input* de quatro canais. Todas as outras camadas foram mantidas iguais à da ResNet18 original, inicializadas com os pesos vencedores da competição ImageNet. Dessa forma, foram treinados dois modelos, um que prevê a GHI e outro o k_t, com o desempenho dos dois sendo apresentado na Figura 44. Percebe-se que, quando a máscara é adicionada, há uma grande diminuição no RMSE do modelo que prevê a GHI, enquanto que o impacto no modelo que prevê o k_t é quase irrelevante. Isto está coerente com o que foi observado no Experimento 1, já que os modelos treinados com a GHI são muito mais dependentes da posição do Sol na imagem do que modelos treinados com o k_t.

4.2.5 Experimento 5 - Otimização dos Hiperparâmetros

Nesse experimento foram otimizados os seguintes hiperparâmetros: *batch size*, taxa de aprendizado, otimizador, *weight decay* e a inicialização de pesos pré-treinados (ImageNet) ou aleatórios. No total, foram realizadas 50 buscas bayesianas com o objetivo de minimzar o RMSE no conjunto de validação após 5 épocas de treinamento, no espaço

Figura 42: Curvas de treino/teste/validação para os modelos treinados com 4 configurações diferentes (*nowcasting*). A configuração do conjunto de validação é a mesma do conjunto de treino. Os hiperparâmetros e as outras configurações são mantidos nos seus valores padrão.

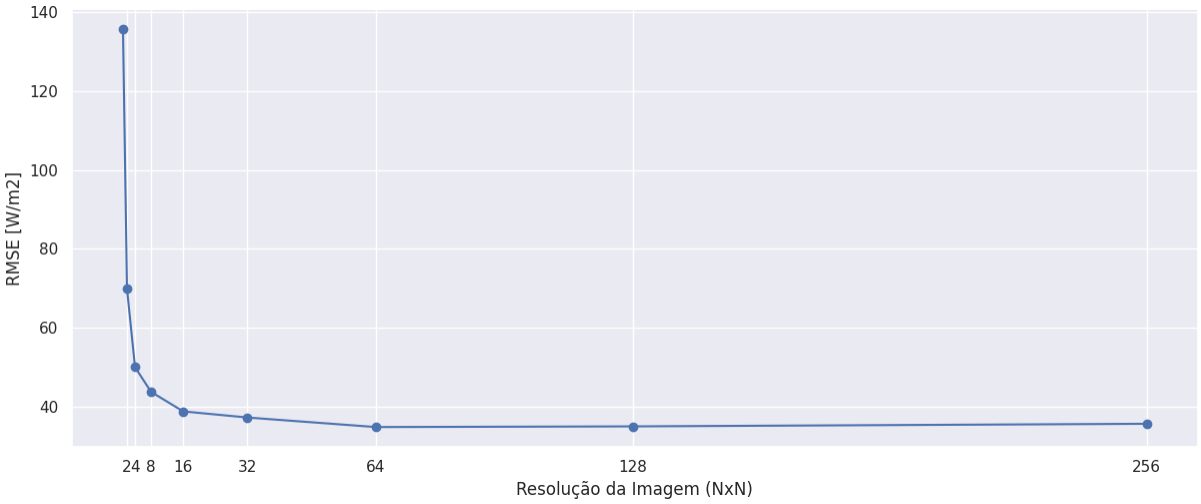


Fonte: O autor

de hiperparamêtros mostrado na Figura 45. O resultado dessas 50 buscas está resumido na Figura 46, da qual pode-se deduzir que:

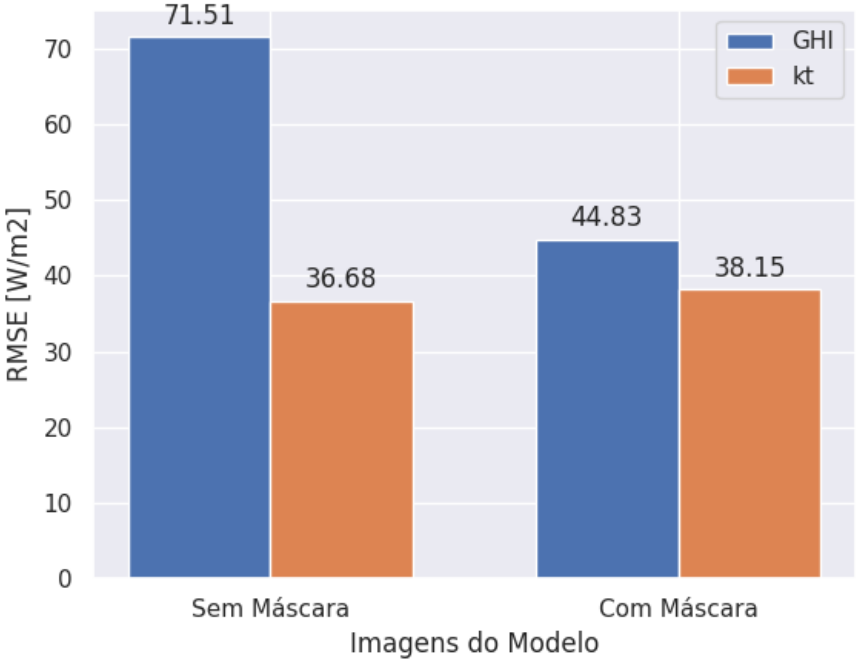
- *Batch sizes* maiores são melhores.
- Taxas de aprendizado menores são melhores.
- O otimizador AdamW é o melhor.
- A inicialização com pesos pré-treinados é melhor.

Figura 43: RMSE de teste para diferentes resoluções.



Fonte: O autor

Figura 44: RMSE de teste para os modelos de GHI e k_t, treinados com e sem a máscara do Sol.



Fonte: O autor

A busca com o melhor resultado conseguiu um RMSE de 48,93 W/m^2 no conjunto de validação, com os seguintes hiperparâmetros: batch_size=144; learning_rate=0,00037; optimizer=adamw; pretrained=true; weight_decay=0,00020. Na sequência, foi treinado um **último** modelo de *nowcasting* com esses valores e os valores padrão da Tabela 3 para os outros hiperparâmetros. Para tentar extrair o melhor desempenho possível desse último

modelo, foi utilizado um *model checkpoint* a partir da 5^a época de treinamento, onde o *checkpoint* era a época com a menor perda de validação. Ou seja, da 5^a época em diante, o modelo era inicializado com os pesos da época com a menor perda de validação. O pseudo-código apresentado na Figura 47 ilustra melhor como isso foi aplicado.

Pela Figura 48, percebe-se que o modelo obeteve a menor perda de validação na época de número 9, com um RMSE de 43,56 W/m^2. Isso siginifica que todas as épocas subsequentes foram inicializadas com os pesos dessa época, porém nenhuma conseguiu um desempenho melhor. Sendo assim, os pesos do **último** modelo de *nowcasting* são os pesos da época 9, que geram um RMSE no conjunto de teste de **35,68 W/m^2**.

Figura 45: Espaço de busca de hiperparâmetros para o modelo de *nowcasting*.

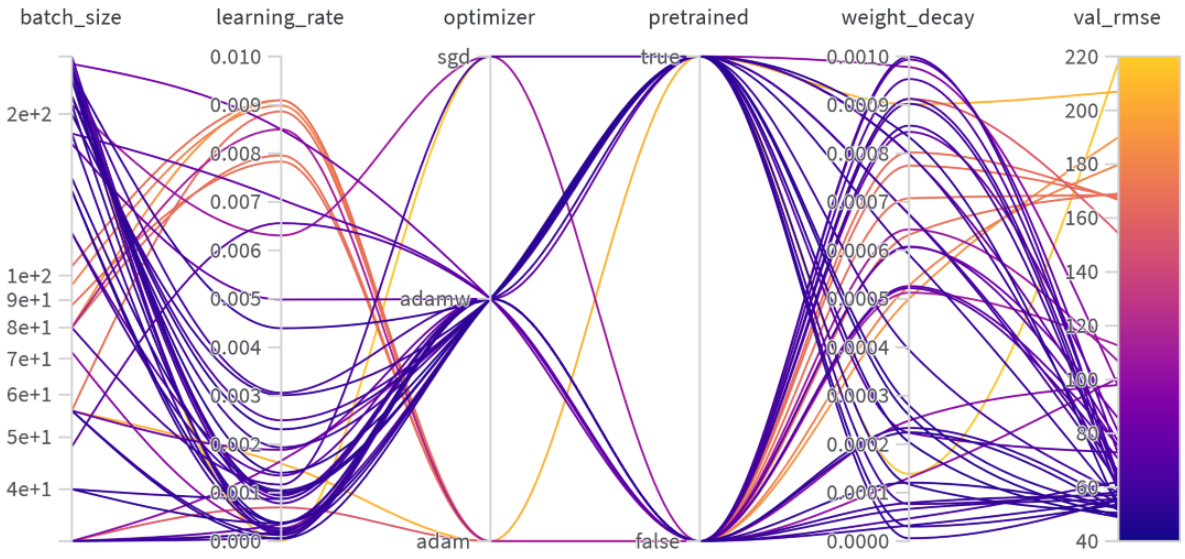
```
method: bayes
metric:
  goal: minimize
  name: val_rmse
parameters:
  batch_size:
    distribution: q_log_uniform_values
    max: 256
    min: 32
    q: 8
  epochs:
    value: 5
  learning_rate:
    distribution: uniform
    max: 0.01
    min: 1e-06
  optimizer:
    values:
      - adam
      - sgd
      - adamw
  pretrained:
    values:
      - false
      - true
  weight_decay:
    distribution: uniform
    max: 0.001
    min: 0
```

Fonte: O autor

4.3 Modelos de Forecasting

Os dois modelos de *forecasting* obtidos foram treinados para gerar previsões de GHI de 5 minutos à frente. Para isso, foram utilizadas **apenas sequências de imagens** na entrada dos modelos, que são processados e treinados de acordo com o que foi descrito na Seção 3.3.2.

Figura 46: Resultados das 50 buscas bayesianas com o objetivo de reduzir a perda/RMSE de validação. Cada linha é uma busca e cada eixo é um hiperparâmetro (*nowcasting*).



Fonte: O autor

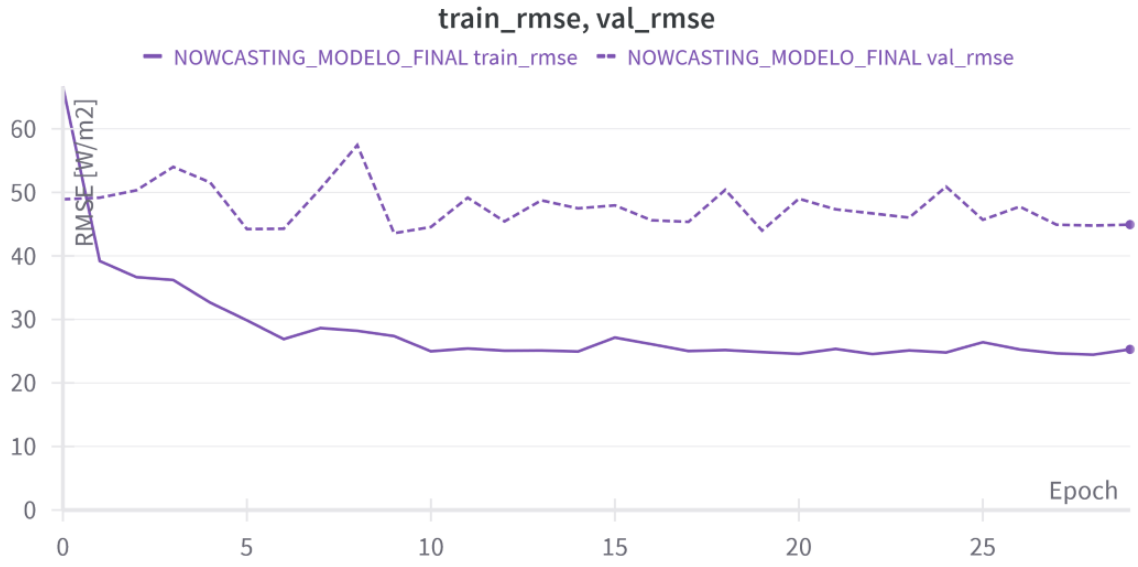
Figura 47: Pseudo-código para aplicar o *model checkpoint* durante o treinamento.

```
VAL_LOSS = 10000000000
PATH = 'MODEL_CHECKPOINT.pth'
for epoch in range(epochs):
    if epoch > 4:
        #Load the saved checkopoint
        model_checkpoint = torch.load(PATH)
        model.load_state_dict(model_checkpoint['model_state_dict'])
        optimizer.load_state_dict(model_checkpoint['optimizer_state_dict'])
        VAL_LOSS = model_checkpoint['val_loss']

    #Train model...

    #Evaluate model
    val_loss = model(val_dataloader)

    if (epoch > 3) & (val_loss < VAL_LOSS):
        #Save the new checkpoint
        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'val_loss': val_loss
            }, PATH)
```

Fonte: O autor

No primeiro modelo, as sequências de *embeddings* das imagens, extraídos a partir da ResNet18 do *nowcasting* (congelada), são utilizadas para treinar uma rede LSTM+MLP. O segundo modelo é parecido com o primeiro, com a diferença de que a ResNet18 é desconge-

Figura 48: Curvas do RMSE de treino e validação para o último modelo de *nowcasting*.



Fonte: O autor

lada e treinada em conjunto com a rede LSTM+MLP. No *forecasting*, são introduzidos dois novos hiperparâmetros: hidden_dim e lstm_layers. O primeiro corresponde ao tamanho do vetor de *hidden state* da LSTM, que também será o tamanho da primeira camada da MLP (Figura 35). O segundo indica o número de camadas da rede LSTM. Os outros hiperparâmetros/configurações são mantidos os **mesmos** que os valores padrão da Tabela 3, a menos que dito o contrário.

4.3.1 Modelo 1 - ResNet18 Congelada

Nessa seção será realizada a previsão em cima dos embeddings obtidos a partir do *model checkpoint* da ResNet18 utilizada no *nowcasting*, que são processados para formarem uma sequência de 15 elementos que obedece às condições apresentadas na Seção 3.3.2. As sequências obtidas são utilizadas para treinar uma rede LSTM em conjunto com uma MLP e, para maximizar o desempenho, são realizadas 50 buscas bayesianas no espaço de hiperparâmetros apresentado na Figura 49, com o objetivo de minimizar a perda de validação após 5 épocas. Os resultados de todas as buscas está apresentado na Figura 50 e, diferente do que ocorreu no *nowcasting*, os hiperparâmetros testados não parecem ter tanta importância, dado que o Δval_rmse está pequeno. De qualquer maneira, a busca com o melhor resultado alcançou um RMSE de 70,82 W/m^2 na validação e foi obtida com: batch_size=56; learning_rate=0,00129; optimizer=adam; hidden_dim=32; lstm_layers=5. Com isso, foi treinado um modelo por 15 épocas (curvas de treino e validação apresentadas na Figura 51) que obteve um RMSE de **64,28 W/m^2** no teste ao final da última época. Como o desempenho do modelo *smart persistence* é de **71,67**

$\mathbf{W/m^2}$ no mesmo conjunto, o *forecasting skill* resultante é de **10,31%**.

Da mesma forma como foi feito no *nowcasting* (Seção 4.2.2), foram verificados os impactos das 4 possíveis combinações de *file name* vs *date modified* nos conjuntos de treino e teste. Para isso, foram mantidos constantes os hiperparâmetros citados acima e os outros valores padrão da Tabela 3. Também vale ressaltar que os *embeddings* utilizados para treinamento **são os *embeddings* extraídos do modelo de *nowcasting* treinado com o rótulo *date modified***. Pela Figura 52 nota-se que a principal diferença em relação ao *nowcasting* é que a melhora no desempenho quando ambos os conjuntos são rotulados com o *date modified* não é tão grande assim. Um possível motivo para isso é que, ao aumentar o horizonte de previsão, a discrepância entre o *file name* e *date modified* vai se tornando menos relevante para a previsão.

Figura 49: Espaço de busca de hiperparâmetros para o modelo de *forecasting*.

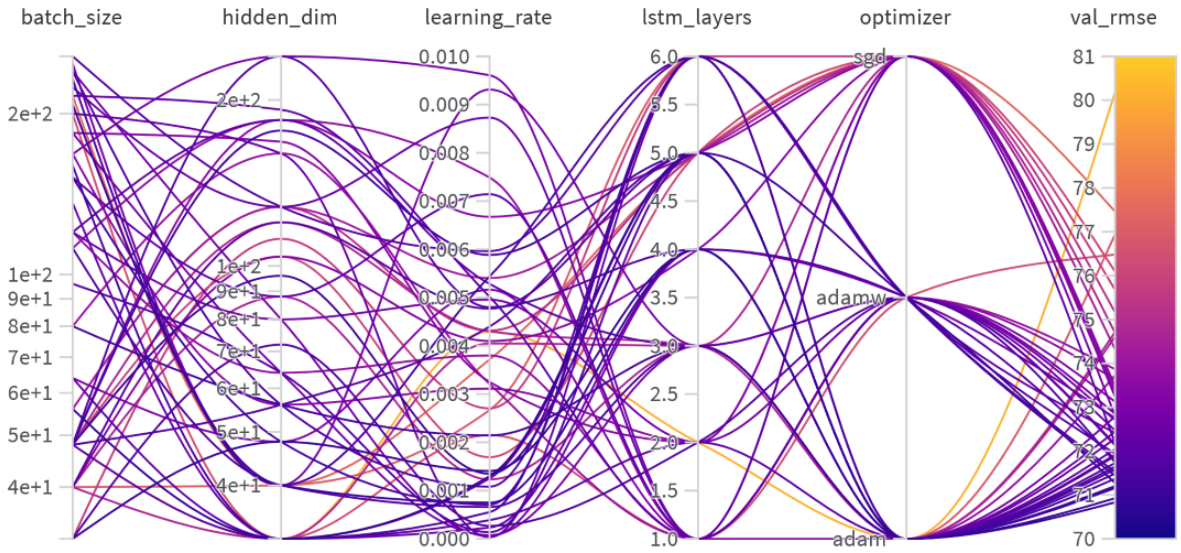
```
method: bayes
metric:
  goal: minimize
  name: val_loss
parameters:
  batch_size:
    distribution: q_log_uniform_values
    max: 256
    min: 32
    q: 8
  epochs:
    value: 5
  hidden_dim:
    distribution: q_log_uniform_values
    max: 256
    min: 32
    q: 8
  learning_rate:
    distribution: uniform
    max: 0.01
    min: 1e-06
  lstm_layers:
    distribution: int_uniform
    max: 6
    min: 1
  optimizer:
    values:
      - adam
      - sgd
      - adamw
```

Fonte: O autor

4.3.2 Modelo 2 - ResNet18 Descongelada

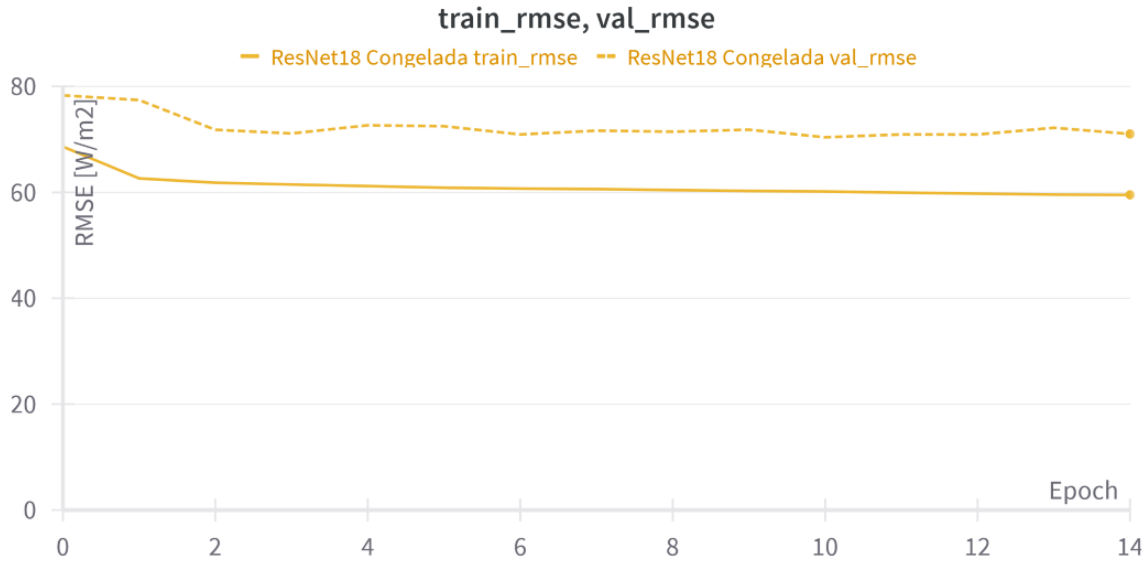
O segundo modelo utiliza a mesma ResNet18 e rede LSTM+MLP que o primeiro, porém a ResNet18 é **descongelada** e treinada em conjunto com a LSTM+MLP, que é inicializada com os pesos do modelo 1. É importante ressaltar que o fato da ResNet18 ser

Figura 50: Resultados das 50 buscas bayesianas com o objetivo de reduzir a perda/RMSE de validação. Cada linha é uma busca e cada eixo é um hiperparâmetro (*forecasting*).



Fonte: O autor

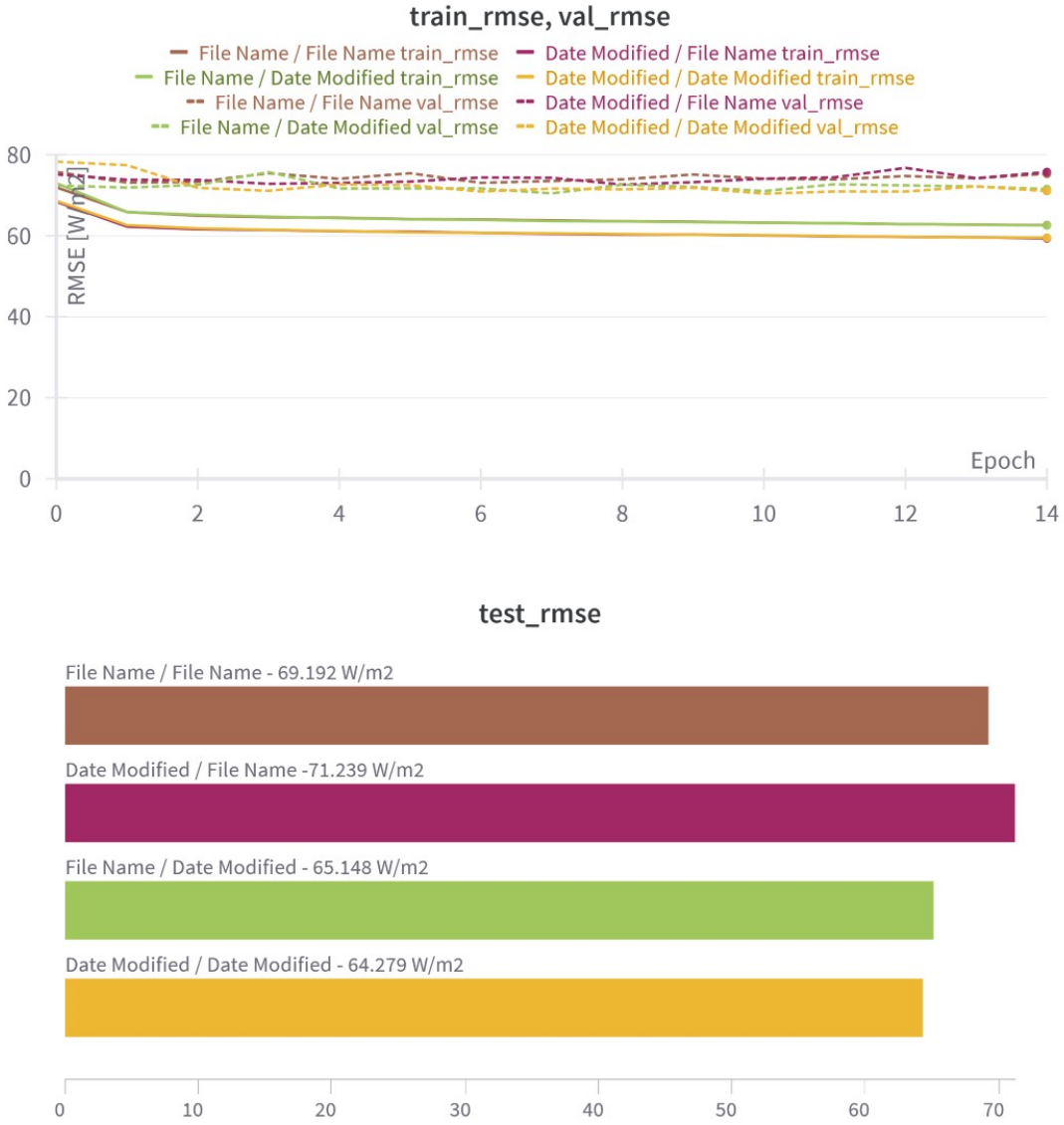
Figura 51: Curvas do RMSE de treino e validação para o modelo 1 de *forecasting*.



Fonte: O autor

descongelada em nada muda o processamento feito para a formação das sequências de treino/teste/validação. A única diferença é que agora essas sequências são sequências de imagens em vez de *embeddings*, que serão processadas uma a uma pela ResNet18 à cada *forward pass*. Ambos esses processos (formação das sequências e *forward pass* na ResNet18) aumentam bastante o tempo de treinamento. Por esse motivo, não foi feita a busca pelos melhores hiperparâmetros do modelo 2 e, inicialmente, foram mantidos os mesmos do

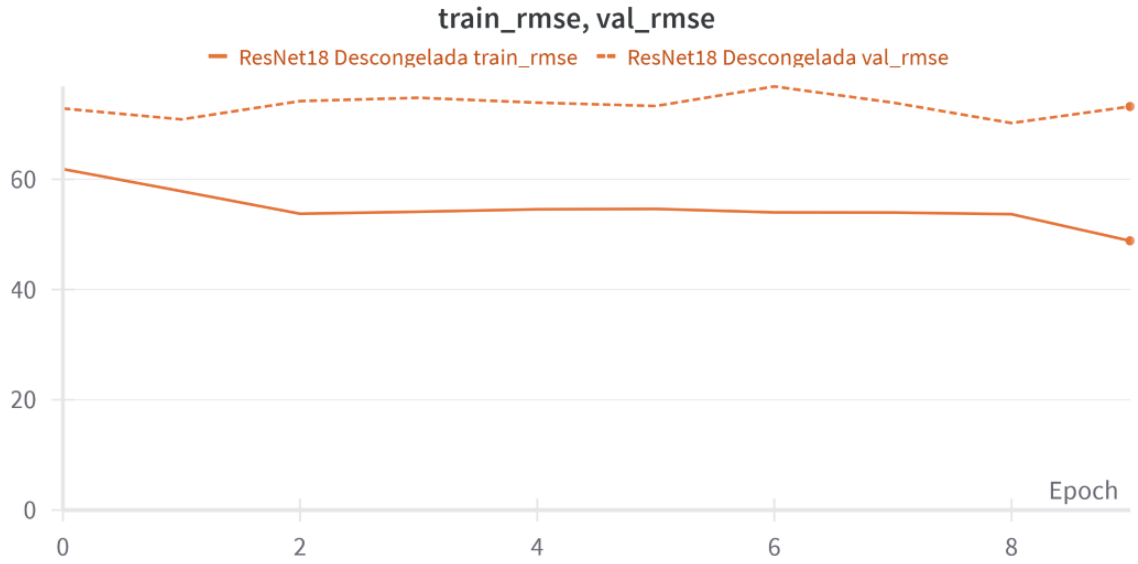
Figura 52: Curvas de treino/teste/validação para os modelos treinados com 4 configurações diferentes (*forecasting*). A configuração do conjunto de validação é a mesma do conjunto de teste.



Fonte: O autor

modelo 1. Porém, notou-se um grande *overfitting* nesse caso, o que faz sentido já que com a ResNet18 descongelada o número de parâmetros do modelo aumenta significativamente. Para contornar esse problema, a *learning rate* foi dividida por 2, foi adicionado um *weight decay* de 0,0001 e um *model checkpoint* com a melhor época de validação. Esse modelo foi treinado por 10 épocas (Figura 53) e, na melhor época de validação, alcançou um RMSE de **64,64 W/m^2** no teste, o que equivale à um *forecasting skill* de **9,81%**.

Figura 53: Curvas do RMSE de treino e validação para o modelo 2 de *forecasting*.



Fonte: O autor

4.3.3 Análise dos Resultados

A avaliação dos resultados é realizada primeiro em cima de todo o conjunto de teste, para os modelos 1 e 2. Na Tabela 5 está resumido o desempenho desses dois modelos, bem como de dois artigos que utilizaram o *dataset* de Folsom com metodologias similares de treino. Embora a conjunto de teste utilizado por esses dois artigos seja diferente (por isso a discrepância no RMSE) o *forecasting skill* indica que o desempenho dos 4 modelos estão relativamente próximos.

Além disso, os custos computacionais dos 4 modelos também são próximos, já que todos são baseados na ResNet18, que possui aproximadamente 11 milhões de parâmetros. A primeira vista, isso indicaria que o *forecasting skill* desses modelos está baixo demais, dado que os modelos persistentes são extremamente simples e não possuem nenhum parâmetro. Todavia, há de se lembrar que, quanto pior for a previsão, maior será a incerteza vinculada às decisões do operador e, consequentemente, maiores os custos de operação. Sendo assim, um ganho de 10,31% em cima do modelo persistente (modelo com as ResNet18 congelada) mais do que compensa o custo computacional de um modelo com 11 milhões de parâmetros.

As piores previsões da GHI do modelo[2] estão apresentadas na Figura 54, que mostra que os maiores erros ocorrem quando o Sol está próximo de ser bloqueado/desbloqueado por nuvens. Embora esse resultado já fosse esperado, deve-se levar em consideração que o modelo foi treinado com a variável alvo sendo o k_t. Isso é relevante pois as piores previsões

[2] As previsões nas três figuras são referentes ao modelo 2, porém conclusões similares foram obtidas para o modelo 1.

do k_t, apresentadas na Figura 55, ocorrem em instantes muito diferentes. De fato, a Figura 56 mostra que os maiores erros da GHI estão mais uniformemente espalhados ao longo do dia, enquanto que os maiores erros do k_t ocorrem em instantes em que o ângulo zenital está alto, o que pode indicar que:

- A distorção da lente *fisheye* é muito severa nas bordas, podendo prejudicar o aprendizado do modelo nos instantes em que o Sol está perto da borda. Nesse caso, soluções que envolvam algum processamento para remover a distorção da lente podem ser interessantes.
- Os modelos de céu claro não são bem calibrados para um ângulo zenital muito elevado, o que representa uma desvantagem de se prever o k_t, já que nesse caso um erro muito grande implica em uma punição muito forte no modelo, que não condiz com grandes erros de GHI, de real interesse.

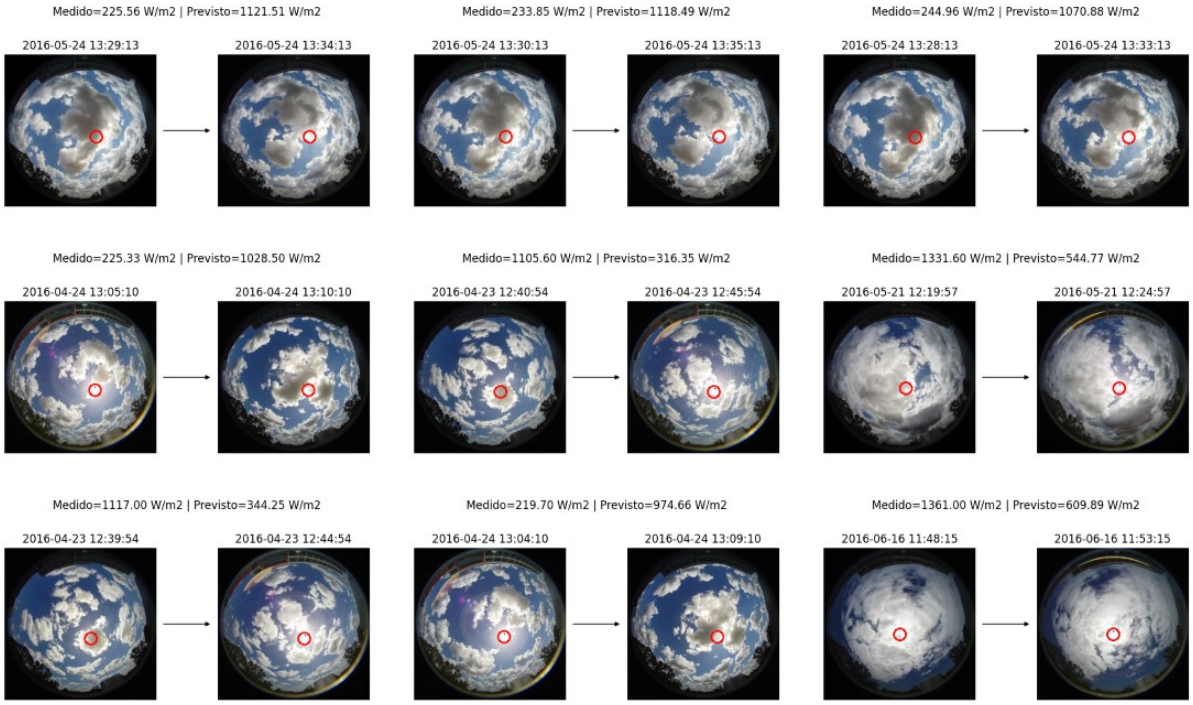
Também foram avaliadas as previsões dos modelos em instantes de alta variação, onde notou-se que ambos os modelos funcionam, essencialmente, como um *smart persistence*, porém com previsões mais conservadoras, observação que também foi levantada por [Paletta, Arbod e Lasenby 2021]. Por exemplo, na Figura 57, pode-se observar que, embora o RMSE dos modelos 1 e 2 estejam menores que o *smart persistence*, os três modelos geram previsões atrasadas do *ground truth*. Nesse caso, a única diferença é que as previsões dos modelos de *deep learning* são mais moderadas, o que evita potencializar o erro quando há oposição de fase entre o *ground truth* e o *smart persistence* entre 12:20:00-12:30:00 e 12:45:00-13:00:00. Na Figura 58 também pode ser observado um fenômeno parecido, em que há um decréscimo nas previsões às 09:50:00 e um acréscimo às 10:41:00, ambos 5 minutos após o *ground truth*.

Tabela 5 – Comparação do desempenho no conjunto de teste dos modelos de *forecasting* de 5 minutos e de dois artigos similares. Ambos os artigos também só utilizaram imagens como dado de entrada de seus modelos.

Modelo	***RMSE [W/m²]***	***FS$_{smart\ persis.}$ [%]***	***FS$_{persis.}$ [%]***
Smart Persistence	71,67	0	-
Persistence	72,52	-	0
ResNet18 Congelada	64,28	10,31	11,36
ResNet18 Descongelada	64,64	9,81	10,86
[Wen et al. 2020]	105,50*	-	5,60
[Papatheofanous et al. 2022]	66,79*	-	8,04

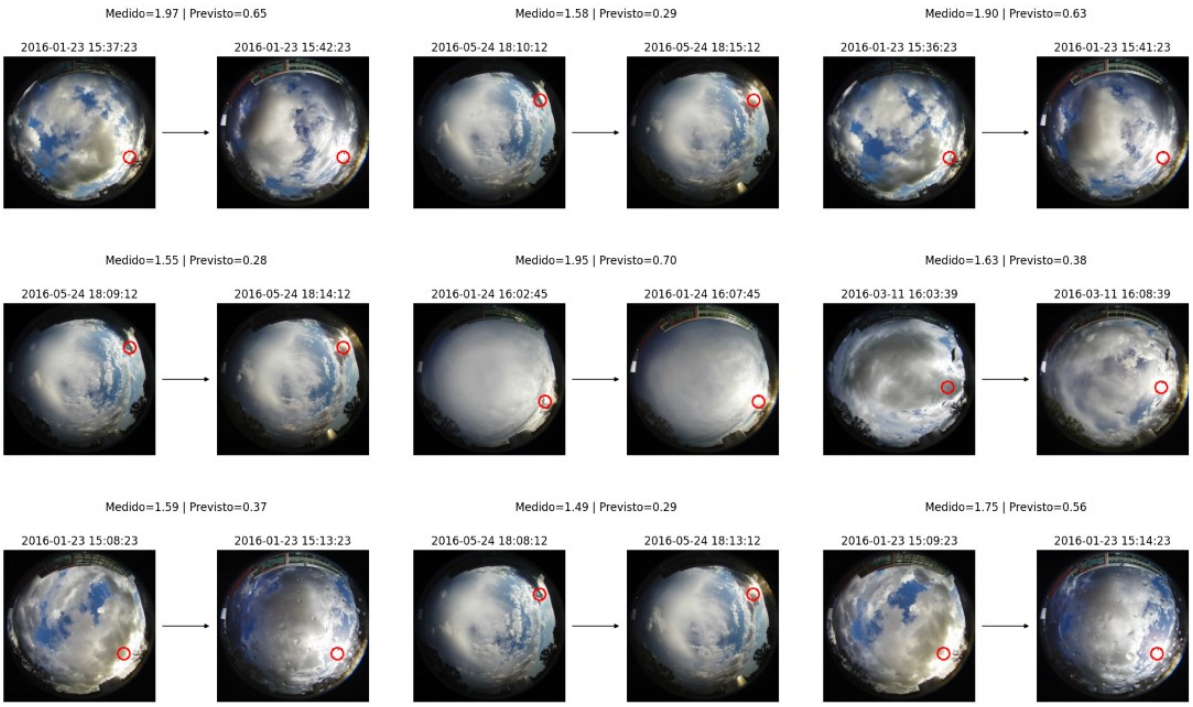
*Os conjuntos de teste utilizados pelos dois artigos são diferentes.

Figura 54: Piores previsões da GHI do modelo. Os *timestamps* correspondem ao *date modified* das imagens.



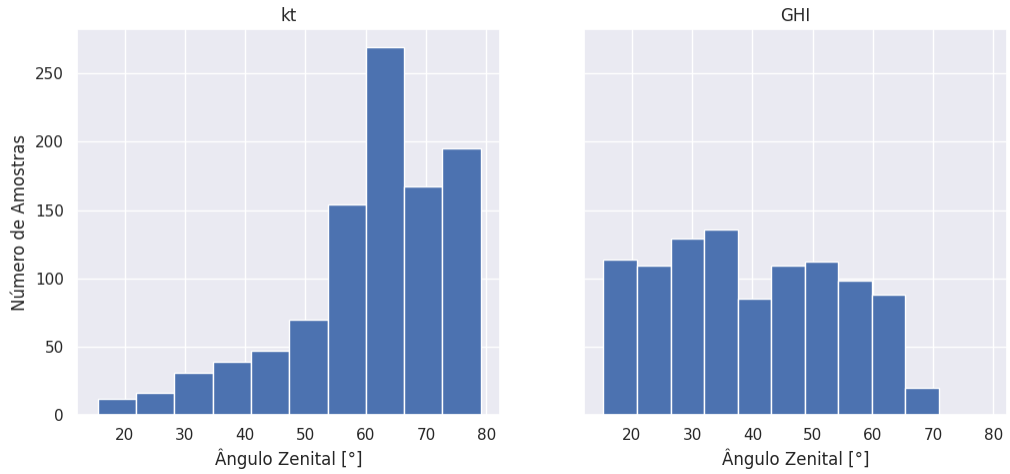
Fonte: O autor

Figura 55: Piores previsões do k_t do modelo. Os *timestamps* correspondem ao *date modified* das imagens.



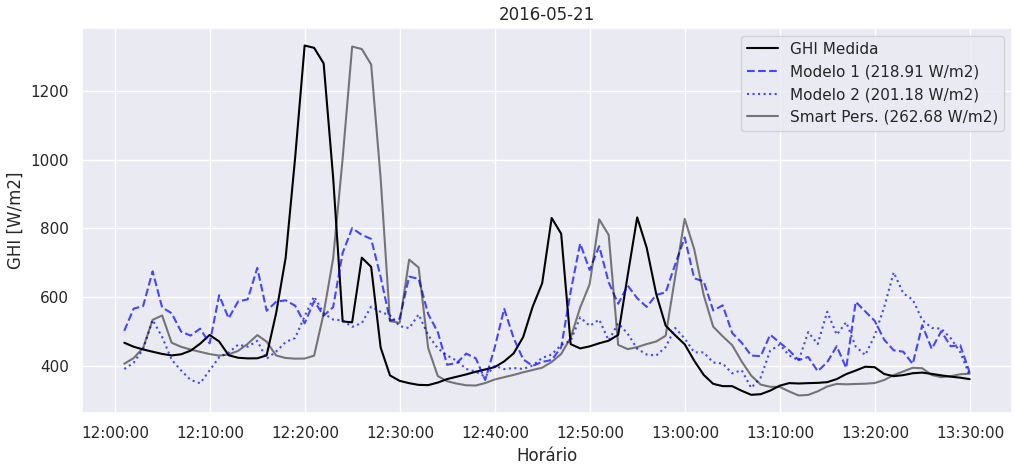
Fonte: O autor

Figura 56: Distribuição do ângulo zenital para as 1000 piores previões do k_t e da GHI.



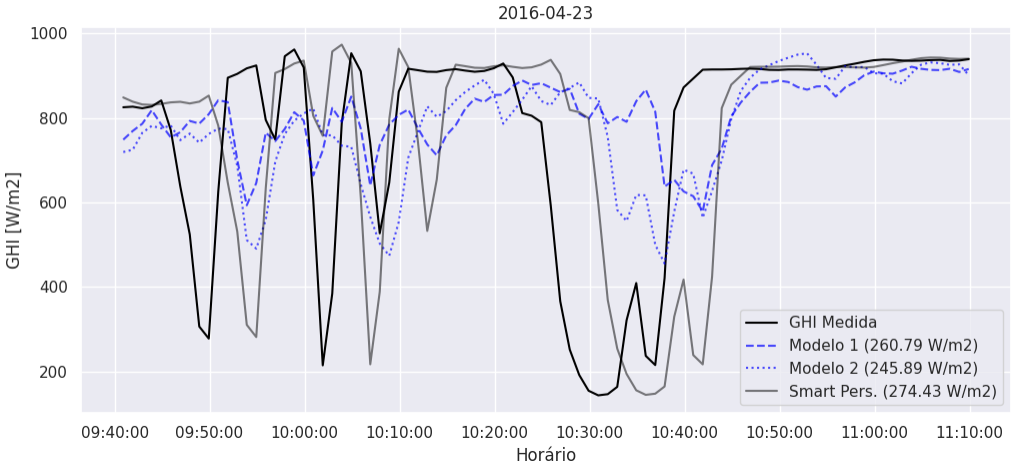
Fonte: O autor

Figura 57: Previsões de 5 minutos dos modelos de *deep learning* e *smart persistence*, com o RMSE correspondente à janela entre parênteses.



Fonte: O autor

Figura 58: Previsões de 5 minutos dos modelos de *deep learning* e *smart persistence*, com o RMSE correspondente à janela entre parênteses.



Fonte: O autor

5 Conclusão

O impacto das mudanças climáticas estão cada vez mais perceptíveis no dia-a-dia da sociedade sendo que, na área de geração de energia, o grande culpado é a queima de combustíveis fosséis. Com isso, a substituição desse tipo de fonte por fontes mais sustentáveis, como a energia solar fotovoltaica, se torna cada vez mais urgente. Contudo, a implementação da energia fotovoltaica no sistema elétrico deve ser feita com cautela, já que a sua intermitência de geração pode desestabilizar e colocar em risco os limites de segurança do sistema. Uma boa maneira para controlar isso seria se o operador do sistema tivesse acesso à previsões confiáveis de GHI de curto prazo que o auxiliariam em suas tomadas de decisão, porém a obtenção de modelos capazes de gerar tais previsões é ainda um problema em aberto.

Dentro desse contexto, modelos baseados em *deep learning* são bastante promissores, haja visto a sua forte capacidade de aprendizado nas mais diversas áreas. Neste trabalho, essas arquiteturas foram treinadas para prever a GHI a partir de imagens do céu disponibilizadas pelo conjunto de dados de Folsom e observou-se que:

- Há um erro na nomenclatura dos arquivos de imagens, que indicam os instantes em que elas foram capturadas.
- A interpolação linear parece ser a mais adequada para resolver a irregularidade na captura das imagens.
- A aplicação de um *timedelta* negativo nas medidas de GHI tende a amenizar o atraso introduzido pelo cálculo da média.
- O índice de céu claro k_t é mais adequado para ser utilizado como rótulo dos modelos, uma vez que não exigem uma codificação tão precisa da posição do Sol na imagem, tornando esses modelos mais robustos à possiveis alterações em relação à distribuição dos dados de treino. Todavia, as maiores punições no treinamento desses modelos ocorrem quando o Sol está nas bordas da imagem, que são instantes que não condizem com grandes erros de GHI, já que ela estará naturalmente mais baixa.
- Após um certo limite, o aumento na resolução das imagens parece ter pouco impacto no desempenho dos modelos.

Com isso, foram obtidos modelos com *forecasting skill* de 10,31% e 9,81%, o que já representa uma boa melhora em relação ao modelo *smart persistence* e está coerente com o atual estado da arte. Por outro lado, observou-se que essa melhora ocorre pois as

previsões dos modelos de *deep learning* produzem um valor médio, evitando grandes erros ao custo de perder a previsão de grandes picos de GHI, que são críticos para a estabilidade do sistema elétrico. Dessa forma, conclui-se que a utilização apenas de uma perda MSE no treinamento desses modelos é insuficiente para garantir a qualidade das previsões.

Dito isso, exerga-se algumas melhorias para trabalhos futuros. A primeira seria melhorar a instrumentação eletrônica utilizada pelos autores do conjunto de dados, já que o erro de nomenclatura das imagens está, muito provavelmente, relacionado a um erro de instrumentação. Melhorar a instrumentação não é uma tarefa trivial, já que envolve sincronizar e processar as medidas de diversos sensores em tempo real, porém acredita-se que isso contribuiria para o desempenho dos modelos.

A segunda consiste em adicionar outras perdas no treinamento, que forçem o modelo a aprender fatores como: previsão de grandes picos de GHI, classificação e segmentação de tipos de nuvens, previsão da velocidade das nuvens, entre outros. Todos essas são informações relavantes para a previsão de GHI, porém exigem um grande processamento nos dados e uma melhoria ainda maior na instrumentação para conseguir gerar todos esses rótulos.

A última melhoria consiste em um treinamento mais auto-supervisionado, que trocaria a necessidade de obter todos esses rótulos pela necessidade de criar uma boa tarefa para ensinar ao modelo. No Apêndice A é apresentado uma abordagem nesse sentido, porém os resultados ruins obtidos indicam que a definição de uma boa tarefa de aprendizado não é tão simples.

De qualquer forma, conclui-se esse trabalho indicando que, independente da abordagem/metodologia de aprendizado, a definição de um conjunto de dados de referência e publicamente disponível é ainda um problema em aberto na área de previsão de GHI, o que dificulta a comparação/reprodução de diferentes estudos e a união de esforços coletivos por parte da comunidade científica.

Referências

ALAPATT, D. et al. Artificial intelligence in surgery: Neural networks and deep learning. 2020. Disponível em: <https://doi.org/10.48550/arXiv.2009.13411>. Citado na página 41.

AMIDI, A.; AMIDI, S. Cs230 - deep learning - recurrent neural networks. Disponível em: <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>. Citado na página 38.

COELHO, R. F.; SCHMITZ, L.; MARTINS, D. C. Energia solar fotovoltaica. 2022. Citado na página 22.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. 2009. Citado na página 30.

DISSAWA, L. H. et al. Sky image-based localized, short-term solar irradiance forecasting for multiple pv sites via cloud motion tracking. 2021. Disponível em: <https://doi.org/10.1155/2021/9973010>. Citado na página 43.

FRAZIER, P. I. A tutorial on bayesian optimization. 2018. Disponível em: <https://doi.org/10.48550/arXiv.1807.02811>. Citado na página 37.

GUSTINELI, M. A survey on recently proposed activation functions for deep learning. 2022. Disponível em: <https://doi.org/10.48550/arXiv.2204.02921>. Citado na página 33.

GéRON, A. Hands-on machine learning with scikit-learn, keras, and tensorflow: concepts, tools, and techniques to build intelligent systems. 2019. Citado 2 vezes nas páginas 30 e 37.

HAURWITZ, B. Insolation in relation to cloudiness and cloud density. *Journal of Meteorology*, v. 2, p. 154–166, 1945. Disponível em: <https://doi.org/10.1175/1520-0469(1945)002<0154:IIRTCA>2.0.CO;2>. Citado 2 vezes nas páginas 27 e 61.

HE, K. et al. Deep residual learning for image recognition. 2016. Disponível em: <10.1109/CVPR.2016.90>. Citado na página 42.

HINTON, G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors. 2012. Disponível em: <https://doi.org/10.48550/arXiv.1207.0580>. Citado na página 35.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Computation*, v. 9, p. 1735 – 1780, 1997. Disponível em: <10.1162/neco.1997.9.8.1735>. Citado na página 39.

INEICHEN, P. A broadband simplified version of the solis clear sky model. *Solar Energy*, v. 82, p. 758–762, 2008. Disponível em: <https://doi.org/10.1016/j.solener.2008.02.009>. Citado 2 vezes nas páginas 27 e 61.

INEICHEN, P.; PEREZ, R. A new airmass independent formulation for the linke turbidity coefficient. *Solar Energy*, v. 73, p. 151–157, 2002. Disponível em: <https://doi.org/10.1016/S0038-092X(02)00045-2>. Citado 2 vezes nas páginas 27 e 61.

KHALIFA, A. B.; FRIGUI, H. Multiple instance fuzzy inference neural networks. 2016. Disponível em: <https://www.researchgate.net/publication/309206911_Multiple_Instance_Fuzzy_Inference_Neural_Networks>. Citado na página 35.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. 2014. Disponível em: <https://doi.org/10.48550/arXiv.1412.6980>. Citado na página 35.

MASSON-DELMOTTE, V. et al. Summary for policymakers. *Global Warming of 1.5C IPCC Special Report on Impacts of Global Warming of 1.5C above Pre-industrial Levels in Context of Strengthening Response to Climate Change, Sustainable Development, and Efforts to Eradicate Poverty*, Cambridge University Press, p. 3–24, 2022. Disponível em: <https://doi.org/10.1017/9781009157940.001>. Citado na página 19.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 115–133, 1943. Citado na página 30.

NASCIMENTO, L. R. et al. Extreme solar overirradiance events: Occurrence and impacts on utility-scale photovoltaic power plants in brazil. *Solar Energy*, v. 186, p. 370–381, 2019. Disponível em: <https://doi.org/10.1016/j.solener.2019.05.008>. Citado na página 28.

NEEDLEMAN, D. B. Optical design guidelines for spectral splitting photovoltaic systems : a sensitivity analysis approach. 2014. Disponível em: <https://www.researchgate.net/publication/279810779_Optical_design_guidelines_for_spectral_splitting_photovoltaic_systems_a_sensitivity_analysis_approach>. Citado na página 22.

NIE, Y. et al. Skipp'd: A sky images and photovoltaic power generation dataset for short-term solar forecasting. *Solar Energy*, v. 255, p. 171–179, 2023. Disponível em: <https://doi.org/10.1016/j.solener.2023.03.043>. Citado na página 56.

NOU, J. et al. A new approach to the real-time assessment of the clear-sky direct normal irradiance. *Applied Mathematical Modelling*, v. 40, p. 7245–7264, 2016. Disponível em: <https://doi.org/10.1016/j.apm.2016.03.022>. Citado na página 27.

PALETTA, Q.; ARBOD, G.; LASENBY, J. Benchmarking of deep learning irradiance forecasting models from sky images – an in-depth analysis. *Solar Energy*, v. 224, p. 855–867, 2021. Citado 2 vezes nas páginas 44 e 78.

PAPATHEOFANOUS, E. A. et al. Deep learning-based image regression for short-term solar irradiance forecasting on the edge. 2022. Citado 4 vezes nas páginas 44, 51, 53 e 78.

PEDRO, H. T. C.; LARSON, D. P.; COIMBRA, C. F. M. A comprehensive dataset for the accelerated development and benchmarking of solar forecasting methods. 2019. Disponível em: <10.1063/1.5094494>. Citado 3 vezes nas páginas 7, 9 e 45.

PEREIRA, E. B. et al. *Atlas Brasileiro de Energia Solar*, 2017. Citado na página 19.

PEREIRA, O. S.; RUTHER, P. Energia solar fotovoltaica. *Revista Brasileira de Energia*, v. 27, n. 3, 2021. Disponível em: <https://doi.org/10.47168/rbe.v27i3.642>. Citado na página 19.

RAJAGUKGUK, R. A.; KAMIL, R.; LEE, H. J. A deep learning model to forecast solar irradiance using a sky camera. 2021. Disponível em: <https://doi.org/10.3390/app11115049>. Citado 2 vezes nas páginas 40 e 43.

RAJAGUKGUK, R. A.; RAMADHAN, R. A. A.; LEE, H. A review on deep learning models for forecasting time series data of solar irradiance and photovoltaic power. 2020. Disponível em: <https://doi.org/10.3390/en13246623>. Citado na página 42.

RENO, M. J.; HANSEN, C. W.; STEIN, J. S. Global horizontal irradiance clear sky models: Implementation and analysis. 2012. Disponível em: <https://doi.org/10.2172/1039404>. Citado na página 27.

SERMANET, P. et al. Time-contrastive networks: Self-supervised learning from video. 2018. Disponível em: <https://doi.org/10.48550/arXiv.1704.06888>. Citado na página 89.

SONG, S. The use of color elements in graphic design based on convolutional neural network model. 2023. Disponível em: <10.2478/amns.2023.2.00536>. Citado na página 41.

VALLANCE, L. et al. Towards a standardized procedure to assess solar forecast accuracy: A new ramp and time alignment metric. *Solar Energy*, v. 150, p. 408–422, 2017. Citado na página 44.

WATT, J.; BORHANI, R.; KATSAGGELOS, A. K. Machine learning refined: foundations, algorithms, and applications. 2016. Citado 3 vezes nas páginas 31, 34 e 35.

WEN, H. et al. Deep learning-based multi-step solar forecasting for pv ramp-rate control using sky images. *IEEE Transactions on Industrial Informatics*, v. 17, p. 1397–1406, 2020. Citado 2 vezes nas páginas 43 e 78.

ZHANG, A. et al. Dive into deep learning. 2020. Citado 4 vezes nas páginas 30, 32, 38 e 40.

ZUO, H. M. et al. Ten-minute prediction of solar irradiance based on cloud detection and a long short-term memory (lstm) model. *Energy Reports*, v. 8, p. 5146–5157, 2022. Disponível em: <https://doi.org/10.1016/j.egyr.2022.03.182>. Citado 2 vezes nas páginas 43 e 44.

APÊNDICE A – Aprendizado Contrastivo no Tempo

Nessa abordagem auto-supervisionada, a ResNet18 foi treinada para aproximar os *embeddings* de imagens próximas no tempo, ao mesmo tempo que distanciar o das imagens distantes no tempo, similar o que foi feito na área de robótica em [Sermanet et al. 2018]. Para cada imagem de referência, denominada âncora, foram obtidas 10 imagens positivas entre -5 e +5 minutos da âncora, que eram aleatóriamente escolhidas a cada época para formar o par positivo e forçar a **proximidade** dos *embeddings*. O par negativo, forçado a **distanciar** os *embeddings*, era obtido aleatóriamente a partir de 10 imagens negativas entre -15→ -10 e +10→ +15 minutos da âncora. Isso foi feito com a Triplet Margin Loss do PyTorch que pune o modelo quando a distância euclidiana entre a âncora a a amostra negativa é menor do que a distância entre a âncora e a amostra positiva através da equação A.1:

$$L\left(a,p,n\right) = max\left\{d(a_i,p_i) - d(a_i,n_i) + margem, 0\right\} \tag{A.1}$$

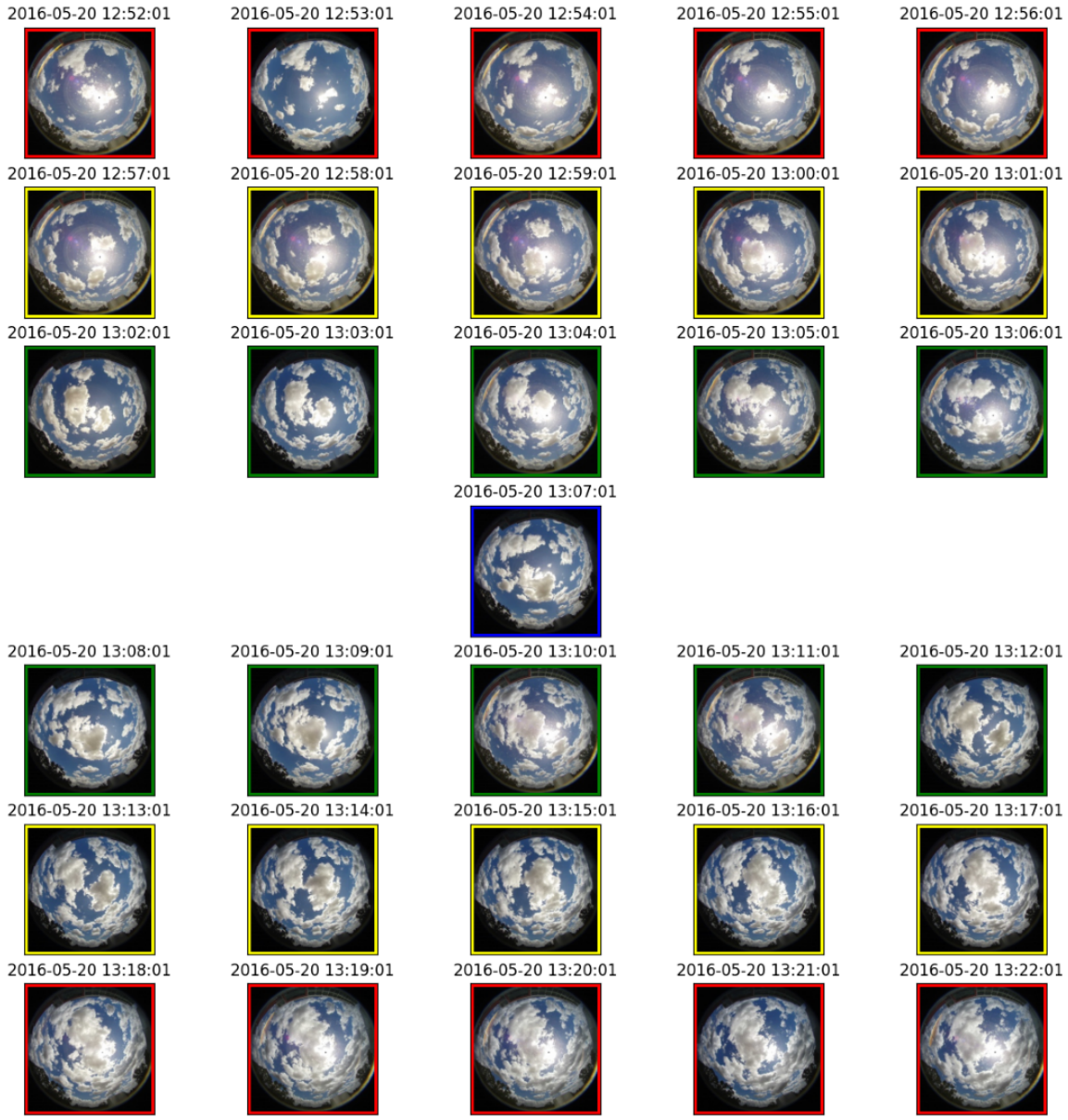
Onde:

- a_i é o *embedding* da âncora.
- p_i é o *embedding* positivo.
- n_i é o *embedding* negativo.
- margem é um hiperparâmetro não negativo que pune o modelo quando a distância do par negativo não é grande o suficiente em relação à distância do par positivo.
- d(x_i, y_i) é a distância euclidiana entre x_i e y_i.

Com a ResNet18 treinada dessa forma, os seus pesos eram então congelados para extrair os *embeddings* das imagens e que foram utilizados para treinar uma LSTM, similar ao que foi feito no Experimento 6. Embora essa tarefa auto-supervisionada pareça fazer sentido à primeira vista, podem haver casos em que as imagens negativas são mais parecidas com a âncora do que as imagens positivas. Por exemplo, na Figura 59 percebe-se que a âncora possui o Sol encoberto por nuvens, porém em várias das imagens positivas os Sol está exposto, o que forçaria o modelo a aproximar duas imagens com irradiâncias totalmente diferentes. De fato, o melhor RMSE obtido no conjunto de teste com essa

abordagem foi de 73,45 W/m^2, indicando um modelo pior do que o *smart persistence*, com um *forecasting skill* de -2,48%.

Figura 59: Exemplos das imagens positivas (borda verde) e negativas (borda vermelha) que serão amostradas a cada época para uma dada âncora (borda azul). As imagens com borda em amarelo determina um intervalo de tempo de no mínimo 5 minutos entre as imagens positivas e negativas



Fonte: O autor