UNIVERSIDADE FEDERAL DO PAMPA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

WENDELL GASPARONI DA LUZ ALVES

APLICAÇÃO DE REDES NEURAIS CONVOLUCIONAIS NA ESTIMATIVA DA ALTURA DE PASTAGENS NATIVAS E CULTIVADAS

Bagé
2025

WENDELL GASPARONI DA LUZ ALVES

APLICAÇÃO DE REDES NEURAIS CONVOLUCIONAIS NA ESTIMATIVA DA ALTURA DE PASTAGENS NATIVAS E CULTIVADAS

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Orientadora: Ana Paula Lüdtke Ferreira

Coorientadora: Teresa Cristina Moraes Genro

Bagé
2025

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

A474a Gasparoni da Luz Alves, Wendell

Aplicação de redes neurais convolucionais na estimativa da altura de pastagens nativas e cultivadas / Wendell Gasparoni da Luz Alves. – 2025.

169 f.: il.

Orientadora: Ana Paula Lüdtke Ferreira
Coorientadora: Teresa Cristina Moraes Genro
Dissertação (Mestrado) – Universidade Federal do Pampa, Campus Bagé, Programa de Pós-Graduação em Computação Aplicada, 2025.

1. Aprendizado de máquina. 2. CNN. 3. Forragem. 4. Massa seca. 5. Pastagem. I. Ana Paula Lüdtke Ferreira. II. Título.

WENDELL GASPARONI DA LUZ ALVES

Aplicação de redes neurais convolucionais na estimativa da altura de pastagens nativas e cultivadas

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Computação Aplicada.

Dissertação defendida e aprovada em: 17 de dezembro de 2024.

Banca examinadora:

_____

Prof.ª Dr.ª Ana Paula Lüdtke Ferreira
Orientadora

(UNIPAMPA)

_____

Prof.ª Dr.ª Márcia Cristina Teixeira da Silveira
(Embrapa)

_____

Prof.ª Dr.ª Carolina Bremm
(DDPA)

_____

Prof. Dr. Edson Prestes e Silva Jr.
(UFRGS)

Assinado eletronicamente por **ANA PAULA LUDTKE FERREIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 17/12/2024, às 12:08, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Edson Prestes e Silva Junior**, **Usuário Externo**, em 17/12/2024, às 16:09, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Carolina Bremm**, **Usuário Externo**, em 18/12/2024, às 00:47, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Márcia Cristina Teixeira da Silveira**, **Usuário Externo**, em 19/12/2024, às 14:29, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1614017** e o código CRC **DF46B37C**.

RESUMO

A produção animal em pasto exige uma boa estimativa da disponibilidade de forragem para que o manejo dos animais seja realizado com precisão, garantindo bons resultados no processo produtivo e evitando problemas de sobre- ou subpastejo. A literatura apresenta que a altura da vegetação pode ser utilizada como métrica para estimar a disponibilidade de alimento no pasto, pois está correlacionada com a quantidade de matéria seca disponível. Como vantagem, a altura é uma medida fácil de ser obtida no campo e requer pouca orientação sobre procedimentos, embora exija trabalho braçal se mensurada de forma direta. Este trabalho propõe uma solução para a medição da altura da vegetação em pastagens nativas e cultivadas, fazendo uso de técnicas de visão computacional. Particularmente, o sistema de inferência recebe uma imagem da vegetação e a processa por meio de redes neurais convolucionais, cada uma treinada especificamente para um tipo de vegetação, a fim de estimar sua altura média. As espécies de pasto usadas nos modelos apresentados neste trabalho são: vegetação nativa dos campos sul-brasileiros, azevém e capim-sudão, embora a técnica possa ser usada para qualquer outro tipo de vegetação. O uso de redes neurais convolucionais não é novo, mas sua aplicação específica para quantificar a altura da vegetação em pastagens representa uma abordagem pioneira. Os benefícios dessa abordagem incluem o desenvolvimento de uma aplicação para coleta de dados e sua integração com o aplicativo H-Pasture, que faz recomendação de entrada e saída de animais com base na altura média do pasto, possibilitando uma tomada de decisão mais acertada no manejo da pastagem. A automação desse processo pode contribuir para a redução de custos operacionais e aumentar a eficiência na produção, promovendo uma gestão mais sustentável dos recursos naturais. Os resultados obtidos apresentaram os seguintes coeficientes de correlação e erros médios absolutos: para pastagens nativas, 0,8949 e 1,5236 cm; para azevém, 0,96 e 1,3791 cm; e para capim-sudão, 0,81 e 7,9795 cm, respectivamente. Esses dados ressaltam a aplicabilidade da abordagem em atividades baseadas na produção animal em pasto, especialmente no manejo de pastagens nativas e azevém.

Palavras-chave: Aprendizado de máquina; CNN; Forragem; Massa seca; Pastagem.

ABSTRACT

Pasture-based livestock production requires an accurate estimation of forage availability to enable precise animal management, ensuring good results in the production process and preventing issues such as overgrazing or undergrazing. The literature shows the use of vegetation height to estimate forage availability in pastures, as it correlates with the amount of dry matter available. As an advantage, height is easily measured in the field and requires minimal procedural guidance, although it demands physical labor when measured directly. This work proposes a solution for measuring vegetation height in native and cultivated pastures using computer vision techniques. Specifically, the inference system processes an image of the vegetation through convolutional neural networks, each trained for a specific vegetation type, to estimate its average height. The pasture species used in the models presented in this work are native vegetation from the southern Brazilian grasslands, ryegrass, and Sudan grass. However, the same technique applies to any other type of vegetation. While convolutional neural networks are not new, their specific application for quantifying vegetation height in pastures represents a novel approach as far as our literature review shows. The benefits of this approach include the development of a data collection application and its integration with the H-Pasture app, which provides recommendations for animal entry and removal based on the average pasture height, enabling more accurate decision-making in pasture management. Automating this process can help reduce operational costs and increase production efficiency, promoting more sustainable management of natural resources. The results obtained the following correlation coefficients and mean absolute errors: 0.8949 and 1.5236 cm for native pastures, 0.96 and 1.3791 cm for ryegrass, and 0.81 and 7.9795 cm for Sudan grass, respectively. These data highlight the applicability of the approach in pasture-based livestock production activities, especially in the management of native pastures or ryegrass.

Keywords: CNN; Dry Fodder; Forage; Machine Learning; Pasture.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANN	*artificial neural network*
API	*application programming interface*
BPCC	*bayesian point cloud classification*
CHM	*canopy height model*
CNN	*convolutional neural network*
EVI	*enhanced vegetation index*
FLOPS	*floating point operations per second*
FNN	*feedforward neural network*
IA	*inteligência artificial*
ILSVRC	*ImageNet Large Scale Visual Recognition Challenge*
LiDAR	*light detection and ranging*
MAE	*mean absolute error*
MBConv	*inverted residual block*
MLR	*multiple linear regression*
MSE	*mean squared error*
MVVM	*Model-View-ViewModel*
NDVI	*normalized difference vegetation index*
PF	*particle filter*
PolInSAR	*polarimetric interferometric synthetic aperture radar*
ReLU	*rectified linear units*
RGB	*red, green, and blue*
RNN	*recurrent neural network*
RMSE	*root mean squared error*
RVoG	*random volume over ground*

SAR	*synthetic aperture radar*
SAVI	*soil-adjusted vegetation index*
SfM	*structure from motion*
SLR	*simple linear regression*
SPAD	*soil plant analysis development*
SWCM	*simplified water cloud model*
TDNN	*time-delay neural network*
UAV	*unmanned aerial vehicle*
ViT	*vision transformer*
WCM	*water cloud model*
XGBoost	*eXtreme gradient boosting*

LISTA DE SÍMBOLOS

μ	valor médio
r	coeficiente de correlação
R^2	coeficiente de determinação
σ	desvio padrão
σ^2	variância

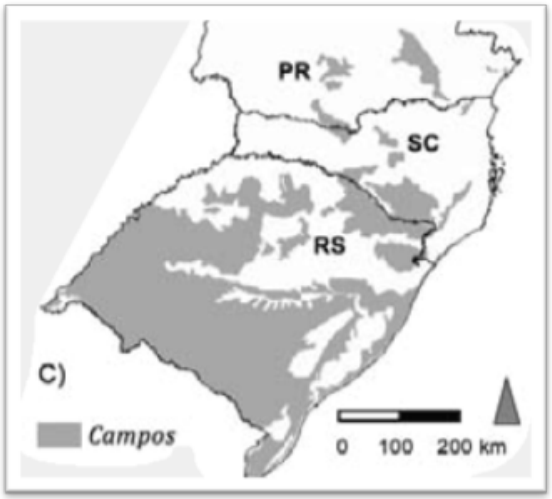
SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto e motivação

Os Campos Sulinos são um ecossistema que se estende pela Região Sul do Brasil, Uruguai, Argentina e uma parte do Paraguai, apresentando uma grande biodiversidade florística (TRINDADE et al., 2012; BENCKE; CHOMENKO; SANT'ANNA, 2016). No Brasil (Figura 1), estes campos estão distribuídos em dois biomas: bioma Pampa – correspondente à metade sul do estado do Rio Grande do Sul – e bioma Mata Atlântica – que inclui áreas no Planalto Sul-Brasileiro, formando mosaicos com as florestas na metade norte do Rio Grande do Sul, Santa Catarina e Paraná (PILLAR, 2009). Devido às características da vegetação desse ecossistema, os Campos Sulinos favorecem atividades baseadas na produção animal em pasto (PILLAR, 2009; NABINGER et al., 2020).

Figura 1 – Campos Sulinos no Brasil



Fonte: (PILLAR, 2009)

A produção animal em pasto exige uma boa estimativa da disponibilidade de forragem, para que a movimentação de animais entre potreiros possa ser realizada com precisão, garantindo bons resultados no processo produtivo (GENRO; SILVEIRA, 2018).

As pastagens são um ecossistema dinâmico e o manejo do pasto é apontado por produtores e técnicos que utilizam esse ecossistema para a produção animal como desafiador, pois as orientações e as definições relativas à lotação dos piquetes e à definição do momento de entrada e saída dos animais nas áreas são pouco precisas. Frequentemente, são observadas situações de sobre- ou subpastejo, ambas acarretando em

baixo desempenho animal, seja por produção vegetal aquém do potencial e degradação da pastagem, no primeiro caso, ou desperdício de forragem, no segundo (HODGSON, 1990).

As pesquisas na área de manejo de pastagens têm trabalhado, principalmente, com dois conceitos de utilização do pasto: o primeiro diz respeito à oferta de forragem e à intensidade de pastejo, propondo o cálculo da oferta da quantidade de pasto existente na área em função de peso vivo animal (quilos de matéria seca ofertados para cada 100 quilos de peso vivo, ou % PV) por um determinado período de tempo (MARASCHIN, 2004); o segundo é a interceptação luminosa, assumindo que o momento ideal para pastejo é quando o pasto está interceptando, em média, 95% da radiação que chega da atmosfera (CARNEVALLI et al., 2006; BARBOSA et al., 2007). Ambos os conceitos exigem treinamento apropriado das pessoas que vão conduzir as avaliações e equipamentos específicos, os quais dificilmente serão encontrados em propriedades rurais de pequeno ou médio portes.

Com o objetivo de construir um sistema de apoio ao manejo das pastagens, a literatura evidencia a busca por um dos componentes da estrutura do pasto que apresente uma boa relação com a massa de forragem disponível para pastejo e que, ao mesmo tempo, seja de mensuração simples. Diversos estudos com produção animal em pastagens cultivadas ou nativas têm apontado que a altura do pasto apresenta uma relação linear positiva (i.e., correlação) com a massa de forragem (ABRAMIDES et al., 1982; TRINDADE et al., 2007; CARVALHO et al., 2010; ROSA; BREMM; MACHADO, 2017; GENRO; SILVEIRA, 2018; FILHO et al., 2019). Por exemplo, Genro e Silveira (2018) encontraram um coeficiente de determinação de 88,2734 para a relação entre altura e massa de forragem em pastagens nativas. Essa relação entre a altura e os componentes da estrutura do pasto, principalmente com a massa de forragem, viabiliza a utilização da altura como critério prático para definir o momento ideal de pastejo, bem como permite identificar a necessidade ou não de realização de ajustes de carga animal, visando estabelecer condições ótimas de utilização do pasto mediante os principais processos envolvidos no crescimento e utilização das plantas forrageiras sob pastejo.

Dados de pesquisas com forrageiras utilizadas nos sistemas de produção do Estado do Rio Grande do Sul têm reforçado a possibilidade do uso da altura como um método seguro para auxiliar o produtor no manejo do pasto da propriedade. Segundo Martins et al. (2015), a relação entre altura do pasto e a massa de forragem é uma importante ferramenta de manejo pois, ao controlar a altura, maneja-se indiretamente diferentes

componentes do sistema. Adicionalmente, dentro da lógica de produção sustentável, tem sido possível observar que o manejo adequado do pasto contribui com serviços ambientais como a mitigação das emissões de gases de efeito estufa (GEE), por meio da redução das intensidades de emissão de metano por quilo de peso vivo produzido por área. Isso porque, quando se usa a meta de altura adequada para manejo do pasto, os desempenhos individuais dos animais em pastejo são mais elevados (SAVIAN et al., 2018) e mais carbono é sequestrado no solo (FRANZLUEBBERS; STUEDEMANN, 2009).

Filho et al. (2019) avalia o efeito de diferentes intensidades de pastejo por novilhos em relação ao desempenho animal, ao consumo de forragem e às emissões de metano (CH_4) no período de utilização da pastagem de um sistema integrado soja-gado de corte no Sul do Brasil. Os tratamentos foram diferentes, definidos pelas alturas-alvo de pastejo (10, 20, 30 e 40 cm) em pastagem consorciada de aveia preta (*Avena strigosa Schreb.*) e azevém (*Lolium multiflorum Lam.*), sob lotação contínua. O manejo do pastejo afetou a utilização da forragem e, consequentemente, o desempenho animal e a emissão de metano. Individualmente, a emissão de metano e o desempenho dos animais apresentaram valores ótimos quando a altura do pasto foi manejada dentro de uma faixa de 23 a 30 cm. Os dados agrupados mostraram um efeito linear positivo da intensidade do pastejo no ganho de peso vivo do animal por hectare e nos custos ambientais associados ao uso da terra. O ganho de peso vivo aumentou em 90 g/ha/dia e as emissões de metano aumentaram em 500 g CO_2 eq/ha/dia para cada centímetro de redução da altura do pasto alvo. Esses resultados mostram o impacto ambiental, em termos de emissões de GEE, que pode ser gerado sem otimização do manejo do pasto.

Assim sendo, o manejo da altura da vegetação em pastagens é uma tarefa da pecuária com impacto direto na qualidade da forragem, no desempenho do gado e na sustentabilidade ambiental. No entanto, é importante destacar que, no cenário prático, o manejo da altura do pasto ainda se baseia predominantemente em métodos manuais, nos quais os pecuaristas utilizam réguas e discos para medir a altura da vegetação. Essa abordagem manual é notoriamente trabalhosa, consumidora de tempo e também está sujeita a erros humanos – visto que a estimativa da média da altura envolve diversas medições que sejam representativas da altura e tipo de vegetação encontrada, podendo prejudicar o desempenho geral do sistema produtivo. A complexidade do ambiente agrícola, com suas variáveis, como diferentes espécies de forragem, características variáveis de crescimento e variações nas condições do solo e topografia, aumenta a dificuldade e a imprecisão dessas medições manuais (VILJANEN et al., 2018).

Com o intuito de superar os desafios enfrentados na medição da altura da vegetação em pastagens nativas e cultivadas, particularmente nas pastagens da Região Sul do Brasil, este trabalho desenvolveu uma abordagem voltada para a automatização desse processo por meio da aplicação de técnicas de visão computacional. A estratégia adotada envolveu o uso de redes neurais convolucionais (CNN), um tipo de rede neural artificial utilizada para processar dados com topologia em grade, como imagens e séries temporais (GOODFELLOW; BENGIO; COURVILLE, 2016). As CNN foram treinadas individualmente para cada espécie de vegetação, permitindo estimar de forma automática a altura média do pasto a partir do processamento de fotos da vegetação na área fotografada.

Embora a utilização de redes neurais convolucionais não seja, por si só, uma abordagem inovadora nas áreas de visão computacional ou reconhecimento de imagens, seu emprego específico na quantificação da altura da vegetação em pastagens nativas e cultivadas da Região Sul do Brasil representou uma aplicação pioneira, até onde a revisão da literatura mostrou. A complexidade e a diversidade da vegetação nessa região exigem adaptações específicas, tornando essa abordagem uma contribuição relevante para a eficácia do manejo sustentável de pastagens e o desenvolvimento de práticas agrícolas mais eficientes nessa região.

Por fim, essa abordagem demonstra vantagens, incluindo a capacidade de coletar dados de forma mais rápida e prática, permitindo um processo de tomada de decisão mais eficiente no manejo da pastagem. A automação do processo de medição da altura por meio de fotos obtidas de celulares contribui para a redução dos custos operacionais e o aumento da eficiência na produção. Além disso, ao contribuir com a otimização do manejo da forragem, essa tecnologia ajuda a evitar práticas que possam causar degradação do solo, como o sobrepastejo, promovendo uma gestão mais sustentável dos recursos naturais.

1.2 Objetivos e resultados esperados

O objetivo deste trabalho foi desenvolver o projeto e a implementação de um sistema de inteligência artificial (IA) especializado na estimativa da altura da forragem em pastagens por meio da análise de imagens capturadas por *smartphones*.

A IA empregada foi baseada em redes neurais convolucionais (CNN) e seu desenvolvimento visou aprimorar a capacidade de monitoramento e gestão de pastagens no âmbito dos sistemas produtivos agropecuários.

Os objetivos específicos deste trabalho são:

- Produzir uma revisão da literatura sobre o estado-da-arte na área da estimativa da altura da vegetação por meio da análise de imagens fotográficas.
- Estudar e discutir diferentes arquiteturas de redes neurais convolucionais, permitindo boas escolhas relacionadas às características do problema.
- Analisar os resultados de estimativa de altura da vegetação por meio das CNN construídas no processo de solução do problema.
- Aprimorar o aplicativo H-Pasture (GASPARONI et al., 2021) para capturar a altura da vegetação por meio de fotos.

Os resultados esperados a partir do desenvolvimento deste trabalho são:

- Disseminação da estratégia de estimativa de massa de forragem por altura, incentivando novos trabalhos sobre o tema, relacionados a diversas forrageiras.
- Simplificação do processo de medição da altura do pasto para pecuaristas e pessoal de campo.
- Contribuição para a disseminação de artefatos e técnicas de Agricultura Digital nos sistemas produtivos agropecuários na região de abrangência do PPGCAP.
- Fortalecimento da parceria entre a Universidade Federal do Pampa e a Embrapa Pecuária Sul, na área de Tecnologias para Produção Agropecuária.

1.3 Organização do trabalho

O restante do texto deste trabalho está estruturado conforme descrito nos parágrafos seguintes.

O Capítulo 2 detalha os procedimentos adotados para a condução deste estudo. Nesse capítulo, são descritos os passos seguidos para planejar e executar a revisão da literatura, o desenvolvimento do aplicativo utilizado para coleta de dados, destacando as etapas do processo, as ferramentas e linguagens de programação empregadas, bem como as interfaces de usuário desenvolvidas. Também são discutidas as decisões relacionadas à escolha do tipo de rede neural utilizada, os métodos de validação adotados para garantir a precisão e eficácia do modelo, assim como a configuração do ambiente de desenvolvimento utilizado para a criação e implementação da rede neural.

Os trabalhos correlatos são abordados no Capítulo 3, apresentando uma análise

crítica da revisão da literatura, identificando as principais técnicas usadas para determinação da altura da vegetação por meio de imagens e as principais aplicações descritas.

O Capítulo 4 aborda o referencial teórico sobre CNN, fornecendo a base conceitual necessária para compreender as abordagens empregadas neste trabalho. Naturalmente que o referencial é resumido, mas está referenciado para que o leitor interessado possa buscar mais informações.

No Capítulo 5, é apresentado o desenvolvimento do módulo de inteligência artificial integrado ao H-Pasture, detalhando todas as etapas envolvidas. Também são apresentadas as redes finais escolhidas, explicando o processo iterativo de criação e os critérios utilizados para otimização.

O Capítulo 6 apresenta uma análise do desempenho das redes desenvolvidas, incluindo os testes estatísticos realizados para identificar a arquitetura mais adequada para cada tipo de vegetação.

Por fim, o Capítulo 7 sintetiza os resultados e conclusões deste trabalho, destacando as contribuições feitas e sugerindo possíveis direções para estudos futuros, que possam melhorar os resultados obtidos.

2 MATERIAL E MÉTODOS

2.1 Caracterização e fases do trabalho

O trabalho desenvolvido pode ser caracterizado como uma pesquisa experimental e aplicada, com procedimentos de natureza bibliográfica, empírica, projetual e de análise de dados (WAZLAWICK, 2009).

As fases do trabalho são apresentadas como se segue:

Revisão da literatura : essa fase teve como objetivo conhecer e analisar o escopo do conhecimento sobre estimativa da altura da vegetação a partir de imagens fotográficas. A revisão bibliográfica permitiu a identificação de técnicas, metodologias e melhores práticas estabelecidas na literatura. O detalhamento do protocolo e dos procedimentos da revisão é apresentado na Seção 2.2. O resultado do processo é discutido no Capítulo 3.

Aquisição de imagens : nessa fase, um aplicativo dedicado à coleta de dados foi desenvolvido, visto que não havia disponibilidade prévia de uma base de dados de imagens de vegetação rotuladas com dados de altura. O aplicativo simplificou o processo de coleta em campo, facilitando a extração e organização das informações. Com um *back end* na nuvem, os dados coletados foram centralizados, estabelecendo um fluxo de processo para o posterior treinamento da rede neural. Essa integração entre a coleta de dados em campo e o armazenamento na nuvem proporcionou uma solução completa e eficiente para adquirir e gerenciar dados. O detalhamento da construção da solução é apresentado na Seção 2.3. A descrição dos procedimentos de coleta é feita no Capítulo 5.

Pré-processamento de dados : nessa fase os dados coletados passaram por um processo de pré-processamento para prepará-los para o treinamento da rede neural. O processo envolveu vários procedimentos, incluindo a aplicação de técnicas de aumento de dados (*data augmentation*) para enriquecer o conjunto de treinamento. Além disso, as imagens foram submetidas a um redimensionamento para garantir que todas fossem entradas compatíveis com a arquitetura da rede neural. Mais detalhes sobre as técnicas de pré-processamento são descritos na Seção 5.3.

Implementação das redes neurais : essa fase envolveu a definição das arquiteturas e a implementação das redes neurais. Foram desenvolvidas diferentes arquiteturas para cada espécie de vegetação, utilizando as CNN consideradas na literatura como

as mais adequadas para a análise de fotos e extração de recursos relevantes para a predição da altura da vegetação. A implementação das redes foi conduzida de forma iterativa, envolvendo ajustes de hiperparâmetros e refinamentos na arquitetura das redes para aprimorar o desempenho dos modelos. Informações detalhadas sobre a implementação, incluindo a configuração específica e as camadas utilizadas, podem ser encontradas nos Capítulos 5 e 6.

Análise dos resultados : nessa fase do trabalho foram realizadas análises e discussões dos resultados obtidos com a implementação das redes neurais para a predição da altura da vegetação, a partir da base de dados coletada. O desempenho dos modelos foi avaliado, bem como sua capacidade de prever as alturas da vegetação em diferentes cenários e as implicações práticas para o pecuarista. Os detalhes dessa fase são encontrados no Capítulo 6.

Integração do módulo de IA no aplicativo H-Pasture : nessa fase foi implementada a funcionalidade de medição da altura média da vegetação por meio de fotos. As redes desenvolvidas durante o trabalho foram integradas ao aplicativo, permitindo a análise automatizada das imagens e o cálculo da altura da vegetação, necessárias para produzir a estimativa da quantidade de alimento no pasto e correspondente recomendação ao produtor de entrada e saída de animais dos potreiros. A arquitetura do aplicativo H-Pasture, com o módulo de IA integrado, pode ser consultada em Seção 5.1.

2.2 Protocolo de revisão da literatura

A revisão bibliográfica é o processo inicial do desenvolvimento de projetos de pesquisa, permitindo a construção de conceitos sólidos sobre um tema de interesse (RIBEIRO; JUNIOR, 2022). A abordagem escolhida para o método de revisão é a de uma revisão de escopo (ARKSEY; O'MALLEY, 2005), que é mais adequada para a descoberta do estado-da-arte sobre um problema de pesquisa bem definido do que uma revisão sistemática (KITCHENHAM, 2004), que é mais adequada para a descoberta de efetividade de métodos usados em processos empíricos. Note-se que os procedimentos de ambos são bastante similares, embora haja diferenças no sentido de que a comparação de resultados de métodos não são o foco da revisão de escopo, que é centrada na descoberta de quais métodos são usados para resolver um determinado problema. Para que a revisão

possa ser reproduzível, cada etapa metodológica deve ser projetada e controlada por meio de um protocolo ou etapas formais de pesquisa.

A elaboração do protocolo e a condução da revisão da literatura usou como suporte a ferramenta Parsifal (<https://parsif.al/>), uma ferramenta online projetada para dar suporte aos pesquisadores na realização de revisões sistemáticas no contexto da Engenharia de Software. O Parsifal oferece um espaço de trabalho compartilhado e colaborativo para pesquisadores geograficamente dispersos, auxiliando em todas as fases da revisão.

Durante a fase de planejamento, o Parsifal deu suporte à definição dos objetivos da revisão, à elaboração de questões de pesquisa, à criação de *strings* de busca, à seleção de fontes relevantes e à estipulação de critérios de inclusão e exclusão dos trabalhos encontrados. Além disso, a ferramenta auxiliou na construção de uma lista de verificação de avaliação de qualidade e formulários de extração de dados, assegurando que a revisão fosse conduzida de forma uniforme, rigorosa e completa.

Por fim, na fase de execução, o Parsifal possibilitou a importação de referências bibliográficas no formato BibTeX, também usadas na construção deste texto. A ferramenta Parsifal permite que a seleção de estudos possa ser feita dentro da ferramenta, identificando os critérios de inclusão ou de exclusão que determinaram a decisão. A identificação de duplicatas entre várias fontes é feita automaticamente, bem como o cálculo da avaliação de qualidade e a extração de dados dos artigos selecionados, a partir do formulário de extração de dados construído. Assim, a ferramenta desempenhou um papel importante na organização e na manutenção de um processo de revisão da literatura estruturado e eficiente.

Na fase de planejamento da revisão da literatura, foram estabelecidas as bases que direcionaram esta revisão. Para a condução desta fase, foram delineadas as seguintes etapas:

1. **Objetivo da revisão**: delimitou-se de maneira precisa o objetivo do trabalho de revisão.
2. **Questões de pesquisa**: formularam-se questões de pesquisa específicas que direcionaram as palavras-chave, as fontes de pesquisa do trabalho de revisão e os critérios de inclusão e exclusão de trabalhos encontrados.
3. **Palavras-chave e sinônimos**: identificaram-se palavras-chave relevantes para o tópico, juntamente com seus sinônimos, que serviram como base para a busca de literatura.

4. ***Strings* de busca**: utilizou-se as palavras-chave e seus sinônimos, bem como operadores lógicos, para criar *strings* de pesquisa criteriosas, que orientaram a identificação de estudos relevantes.
5. **Fontes de pesquisa**: foram determinados os repositórios acadêmicos que seriam explorados durante esta revisão, garantindo que a busca incluísse fontes confiáveis e abrangentes em relação ao objetivo do trabalho.
6. **Critérios de seleção**: estipularam-se critérios de inclusão e exclusão que permitiram determinar quais estudos seriam considerados com base em sua relevância para o objetivo desta revisão.
7. **Avaliação de qualidade e extração de dados**: nesta etapa os estudos identificados foram submetidos a uma análise crítica e os dados relevantes foram extraídos automaticamente a partir das análises feitas sobre os trabalhos selecionados.

Assim, o planejamento dessas etapas da revisão assegurou que a pesquisa fosse conduzida de maneira estruturada e eficaz. Também deve ser observado que ocorreu mais de uma iteração entre as etapas 2 e 7, para refinamento do protocolo e obtenção de trabalhos mais relevantes em relação ao objetivo do trabalho de revisão.

Na etapa 1, foi definido como o objetivo desta revisão: “adquirir um conhecimento abrangente acerca do estado-da-arte na área da estimativa da altura da vegetação, por meio da análise de imagens fotográficas, com ênfase na identificação e compreensão detalhada das técnicas e ferramentas tecnológicas atualmente empregadas”.

Após a definição do objetivo na etapa 1, seguiu-se para a etapa 2. Nesta etapa foram formuladas as seguintes questões de pesquisa da revisão bibliográfica, que norteiam todos os demais elementos do protocolo:

Q1 Quais são os trabalhos existentes na literatura que tem como objetivo usar imagens para medir a altura da vegetação?

Q2 Quais trabalhos utilizam inteligência artificial para a mensuração de altura ou de outros atributos relacionados à distância entre pontos?

As etapas 3, 4 e 6 contemplaram a definição das palavras-chave (e seus sinônimos), a criação de *strings* de busca e a seleção dos repositórios. A Tabela 1 apresenta as palavras-chave que foram definidas, juntamente com seus respectivos sinônimos. A definição dos sinônimos permite que a pesquisa abranja variações terminológicas e, consequentemente, encontre um maior número de artigos relacionados.

Cada repositório pesquisado possui particularidades na sintaxe da *string* de busca.

Tabela 1 – Palavras-chave e sinônimos

Palavra-Chave	**Sinônimos**
height	length, size, weight, width
image	photograph, picture
prediction	calculation, estimation
vegetation	forage, pasture, sward

Como resultado, foram desenvolvidas *strings* específicas para cada repositório, garantindo que a busca atendesse à sintaxe requerida em cada plataforma. Também ocorreram adaptações na semântica das *strings* de busca para atender às nuances de cada repositório, assegurando assim que a revisão abrangesse uma variedade de artigos dentro do contexto da pesquisa, sem gerar sobrecargas com uma grande quantidade de artigos fora do contexto. A Tabela 2 apresenta as *strings* criadas por repositório. Na primeira linha dessa tabela está a *string* base, que serviu de partida para a criação das *strings* individuas para cada repositório.

Tabela 2 – *Strings* de busca

Repositório	***String***
***String* Base**	**(“vegetation” OR “forage” OR “pasture” OR “sward”) AND (“height” OR “dimention” OR “length” OR “size” OR “weight”) AND (“image” OR “photograph” OR “picture”) AND (“prediction” OR “calculation” OR “estimation”)**
IEEE Digital Library	((“All Metadata”:“vegetation” OR “All Metadata”:“forage” OR “All Metadata”:“pasture” OR “All Metadata”:“sward”) AND (“All Metadata”:“height” OR “All Metadata”:“dimention” OR “All Metadata”:“lenght” OR “All Metadata”:“size” OR “All Metadata”:“width”) AND (“All Metadata”:“image” OR “All Metadata”:“photograph” OR “All Metadata”:“picture”) AND (“All Metadata”:“prediction” OR “All Metadata”:“calculation” OR “All Metadata”:“estimation”) AND (NOT (“All Metadata”:“forest” OR “All Metadata”:“forests” OR “All Metadata”:“tree” OR “All Metadata”:“trees”)))
Science@Direct	(“vegetation” OR “forage” OR “pasture”) AND (“height”) AND (“image” OR “photograph” OR “picture”) AND (“prediction” OR “estimation”) -forest -tree -forests -trees
Scopus	(TITLE-ABS-KEY (vegetation OR forage OR pasture OR sward) AND TITLE-ABS-KEY (height) AND TITLE-ABS-KEY (image OR photograph OR picture) AND TITLE-ABS-KEY (prediction OR estimation) AND NOT TITLE-ABS-KEY (forest OR forests OR tree OR trees)) AND PUBYEAR >2018

A etapa 6 contemplou a elaboração dos critérios de inclusão e exclusão que seriam aplicados para identificar os estudos relevantes durante a revisão. Os critérios foram definidos da seguinte maneira:

Critérios de inclusão :

1. Ano de publicação a partir de 2019.
2. Os estudos devem estar disponíveis em língua inglesa ou portuguesa.
3. Os estudos devem responder a pelo menos uma das questões de pesquisa definidas.

Critérios de exclusão :

1. Trabalhos anteriores dos mesmos autores e com o mesmo tema.
2. Trabalhos que estão no contexto de florestas ou árvores, uma vez que não estão diretamente relacionados ao escopo desta pesquisa.
3. Trabalhos que são repetidos, evitando assim duplicações na análise dos resultados.

Esses critérios de seleção foram estabelecidos para garantir a inclusão de estudos relevantes e a exclusão de trabalhos que não atendem aos objetivos e ao foco da revisão.

Por fim, na etapa 7 foram elaboradas questões para avaliar a qualidade dos artigos e foi desenvolvido um formulário para facilitar a extração dos dados dos trabalhos, facilitando as análises. Ficaram definidas as seguintes questões para avaliar a qualidade dos artigos:

1. O artigo descreve o método a ponto de ser reproduzível?
2. O artigo valida os resultados com algum tipo de técnica de validação estatística?

Para a resposta de cada uma das questões foi atribuído um peso, sendo 1 para SIM, 0.5 para PARCIALMENTE e 0 para NÃO. Para a extração dos dados, visando uma padronização, desenvolveu-se um "Formulário de Extração de Dados" que descreve os dados a serem coletados. Esse formulário (Tabela 3) é uma ferramenta para garantir a consistência e a eficácia do processo de revisão. A seguir, explica-se as colunas do formulário:

- **Coluna 1 - Nome do campo:** Nesta coluna, está descrito o nome do dado extraído, como: "Tipo de medição", "Técnica utilizada", "Tipo de vegetação" e outros.
- **Coluna 2 - Tipo de campo:** Essa coluna especifica o tipo de campo. Pode

Tabela 3 – Formulário de extração de dados

Nome do campo	**Tipo de campo**	**Valores possíveis**
Tipo de medição	Seleção única	Comprimento/altura/largura, Outra, Volume
Técnica utilizada	Seleção múltipla	*EXtreme gradient boosting (XGBoost)*, Fotogrametria, Inversão modifica do *polarimetric interferometric synthetic aperture radar (PolInSAR)*, Inversão do *simplified water cloud model (SWCM)*, Inversão do *canopy height model (WCM)*, *Light detection and ranging (LiDAR)*, *Bayesian point cloud classification (BPCC)*, PROTOLIDAR, *Particle filter (PF)*, Random forest, *Artificial neural network (ANN)*, *Convolutional neural network (CNN)*, *Multiple linear regression (MLR)*, *Simple linear regressionl (SLR)*, Regressão polinomial, *Support vector machine* (SVN), *Structure from motion (SfM)*, *K-nn trees*
Tipo de vegetação	Seleção múltipla	Pastagens, Plantações, Savana
Ambiente de coleta	Seleção múltipla	Aéreo, Espacial, Terrestre
Técnica de coleta	Seleção múltipla	*Global navigation satellite system* (GNSS), *Light detection and ranging (LiDAR)*, Multiespectral, *Polarimetric interferometric synthetic aperture radar (PolInSAR)*, *red, green, and blue (RGB)*, RGB-D, *Synthetic-aperture radar* (SAR), Térmica
Tipo de dados	Seleção única	Real, Descrição
Resultados de desempenho	Descritivo	N/A

ser “Seleção única”, “Seleção múltipla” ou “Descritivo”. “Seleção única” significa que apenas um valor pode ser selecionado ou aplicado para esse campo. Por exemplo, no caso de “Tipo de medição”, deve-se escolher entre “Comprimento/altura/largura”, “Outra” ou “Volume”. “Seleção múltipla” indica que vários valores podem ser selecionados, como em “Técnica utilizada”, onde várias técnicas, como “CNN” e “Fotogrametria”, podem ser aplicadas.

- **Coluna 3 - Valores possíveis:** Nesta coluna, estão listados os valores ou opções possíveis para cada campo, dependendo do tipo de campo. Por exemplo, no campo “Tipo de medição”, os valores possíveis incluem “Comprimento/altura/largura”, “Outra” e “Volume”. Para “Técnica utilizada”, há várias técnicas, como “CNN”, “*multiple linear regression (MLR)*” e “Fotogrametria”. No campo “Resultados

de desempenho" é utilizado "N/A" (Não aplicável) para indicar que o campo é descritivo e não possui valores específicos associados a ele.

O "Formulário de Extração de Dados" não apenas facilita a categorização dos dados extraídos dos artigos, mas também promove a consistência ao longo de todo o processo de análise dos trabalhos. Ao fornecer critérios claros e estruturados para a extração de informações, o formulário ajuda a garantir que todos os dados relevantes sejam coletados e que todos os trabalhos encontrados sejam avaliados pelos mesmos critérios.

Após a conclusão das iterações de planejamento da revisão da literatura, as *strings* de busca foram aplicadas nos repositórios previamente selecionados. Como resultado, a Tabela 4 apresenta o número de artigos encontrados e incluídos por repositório.

Tabela 4 – Artigos por repositório

Repositório	**Artigos encontrados**	**Artigos incluídos**
IEEE Digital Library	44	3
Science@Direct	42	2
Scopus	129	9

As *strings* de busca permitiram que muitos artigos fossem retornados. No entanto, ao analisar o conteúdo desses artigos, tornou-se evidente que grande parte deles não respondia satisfatoriamente a pelo menos uma das questões de pesquisa estabelecidas. Dessa forma, visando excluir da pesquisa aqueles que claramente estavam fora do contexto do estudo, foi empreendida a leitura dos resumos de todos os artigos. Essa etapa de triagem primária assegurou que apenas os artigos mais alinhados com os objetivos da pesquisa fossem mantidos para análise posterior.

Após a primeira triagem, foi empregada uma análise mais detalhada dos artigos restantes. Nessa segunda fase de análise, além da leitura das seções de introdução, materiais e métodos e conclusão de cada artigo, foi preenchido o questionário de qualidade e o formulário para a extração de dados. Essa etapa permitiu confirmar a pertinência de cada artigo à pesquisa e também contribuiu para a compreensão do estado da arte na área de predição da altura da vegetação.

Ao final da fase de execução, foram identificados um total de 14 artigos que se mostraram pertinentes com o propósito da pesquisa (ou seja, que atenderam a todos os critérios de inclusão e a nenhum dos critérios de exclusão). O conteúdo desses artigos é discutido na Seção 3.1, onde será apresentada suas contribuições e resultados em relação ao estado da arte na área de predição da altura da vegetação.

2.3 Construção da base de imagens

O processo de aquisição de imagens para construção da base rotulada de dados foi conduzido a partir do desenvolvimento de um aplicativo móvel fazendo uso do *framework* Flutter (Google, 2023) e um *back end* programado em ASP.NET 6 (Microsoft, 2023), conectado a um banco de dados relacional PostgreSQL (PostgreSQL Global Development Group, 2023).

O aplicativo de coleta de dados (denominado *H-Pasture Collector*) foi desenvolvido com o objetivo de facilitar a coleta e a rotulação de informações de campo por meio de um *smartphone*. O aplicativo também foi incorporado como um módulo da arquitetura do H-Pasture, tendo sido projetado para permitir sua utilização em formato colaborativo, ou seja, com múltiplos usuários contribuindo para a coleta de dados em campo. A função do aplicativo abrange a captura, a organização e gestão das fotos coletadas, facilitando o processo de construção da base de dados de imagens.

O aplicativo *H-Pasture Collector* possui a capacidade de operar em modo *offline*, permitindo a coleta de fotos em locais remotos ou com conectividade limitada. A funcionalidade *offline* oferece a flexibilidade necessária para coletar dados de campo de maneira contínua, independentemente das condições de conectividade à internet. Após a conclusão do processo de captura, os dados são armazenados localmente no dispositivo, aguardando a sincronização com a nuvem.

A Figura 2 apresenta a tela inicial do aplicativo, podendo-se observar todas as coletas já realizadas. Uma coleta corresponde a uma ou mais medições de altura e uma ou mais capturas de fotos de determinado ponto da área de pastagem sendo estudada. O botão na parte superior da tela inicial realiza a sincronização de todas as coletas ainda não sincronizadas.

A Figura 3 apresenta a funcionalidade de registro de coleta de imagens do aplicativo, que é realizada em duas etapas: primeiro, ocorre a inserção de uma ou mais medições da altura de determinado ponto no pasto, seguida pela captura das fotos no local. Na tela A da Figura 3, são inseridas as medições da altura do pasto, e o aplicativo permite especificar o tipo de pastagem associado a cada medição. Em seguida, na tela B, realiza-se a captura de uma ou mais fotos do mesmo ponto de medição.

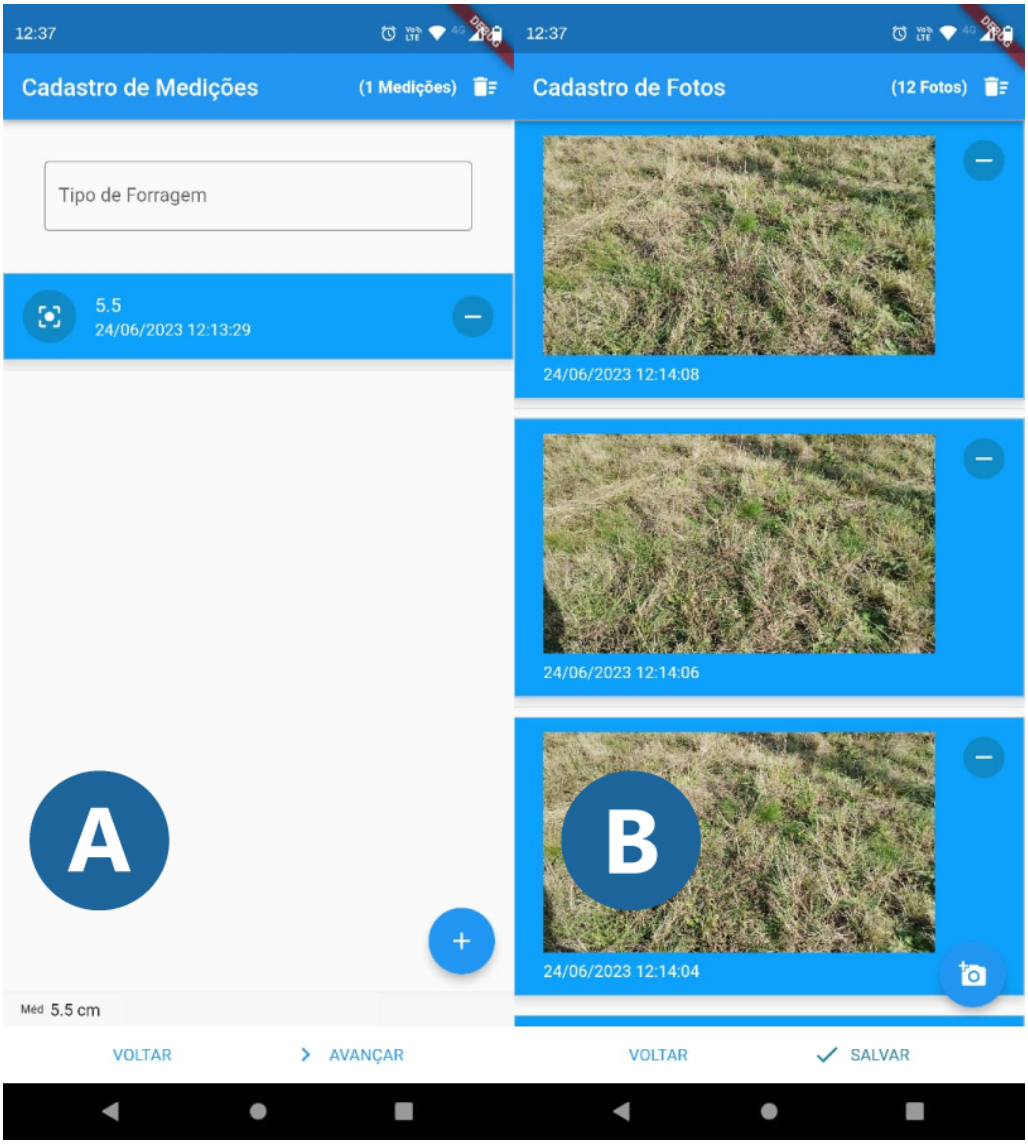
O aplicativo H-Pasture Collector padronizou a captura de fotos em uma resolução de 1280x720 *pixels*, buscando um equilíbrio entre a qualidade das imagens e o uso de armazenamento. Essa escolha considera que alguns dispositivos móveis podem ter

Figura 2 – H-Pasture Collector – tela inicial

Fonte: Autor (2024)

limitações de armazenamento (BEDI; SINGH; GUPTA, 2016), o que poderia impactar na usabilidade do aplicativo durante o uso em campo, quando uma grande quantidade de fotos devem ser capturadas em sequência e armazenadas localmente. Além disso, a literatura indica que CNN podem alcançar resultados satisfatórios com resoluções menores, como mostram os estudos de Simonyan e Zisserman (2015) com a rede VGG (que utiliza resoluções de 224×224 *pixels*) e de Chollet (2017) com a rede Xception (que explora resoluções de até 299×299 *pixels*).

Figura 3 – H-Pasture Collector – funcionalidade de cadastro de coletas



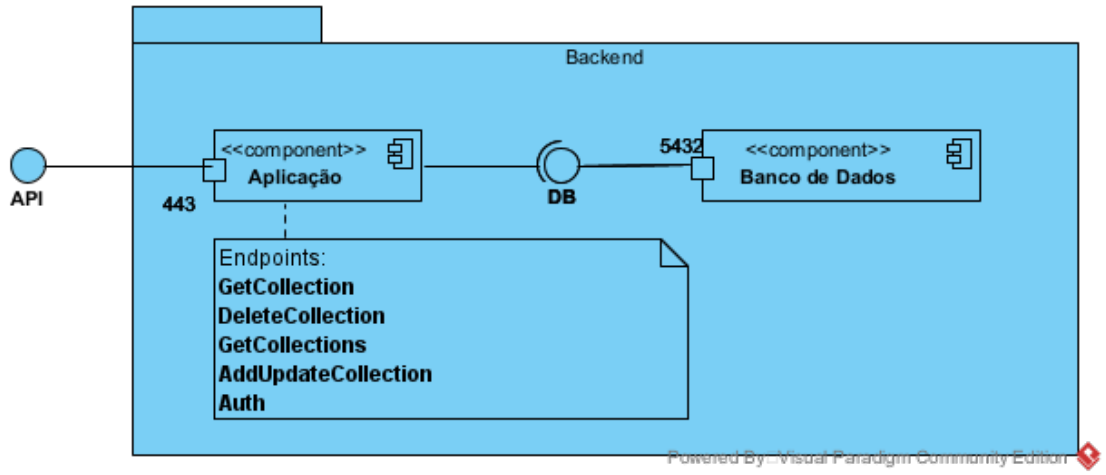
Fonte: Autor (2024)

A integração entre medições e fotografias, aliada à possibilidade de utilização em modo *offline*, fornece uma solução completa para a coleta de dados em campo. Isso simplifica o processo e, ao mesmo tempo, aprimora a qualidade e organização dos dados, pois associa informações visuais ao contexto das medições. Por fim, essa estrutura oferece todos os dados necessários para o treinamento das redes neurais.

Para a centralização da coleta de dados e a facilitação da extração desses dados para o subsequente treinamento das redes neurais, foi desenvolvido um *back end* na nuvem. Esse *back end* consiste em uma *application programming interface (API)* desenvolvida utilizando o *framework* ASP.NET 6 e um banco de dados relacional baseado no PostgreSQL.

A API foi projetada para oferecer um conjunto de *endpoints* que atendem às necessidades específicas do aplicativo H-Pasture Collector instalado no *smartphone* do usuário (nesse caso, da pessoa que está coletando as fotos para construção da base). Também foi incluído um *endpoint* dedicado à autenticação do usuário, que faz uso do mecanismo de autenticação Basic Auth (RESCHKE, 2015), no qual o usuário fornece suas credenciais (nome de usuário e senha) durante as requisições. Vale ressaltar que não foi incorporado um *endpoint* para o cadastro de novos usuários; em vez disso, todos os usuários coletores são cadastrados diretamente no banco de dados. A Figura 4 apresenta em maior detalhes a arquitetura do *back end* e a nota vinculada ao componente "Aplicação" resume os *endpoints* implementados.

Figura 4 – H-Pasture Collector – arquitetura do *back end*

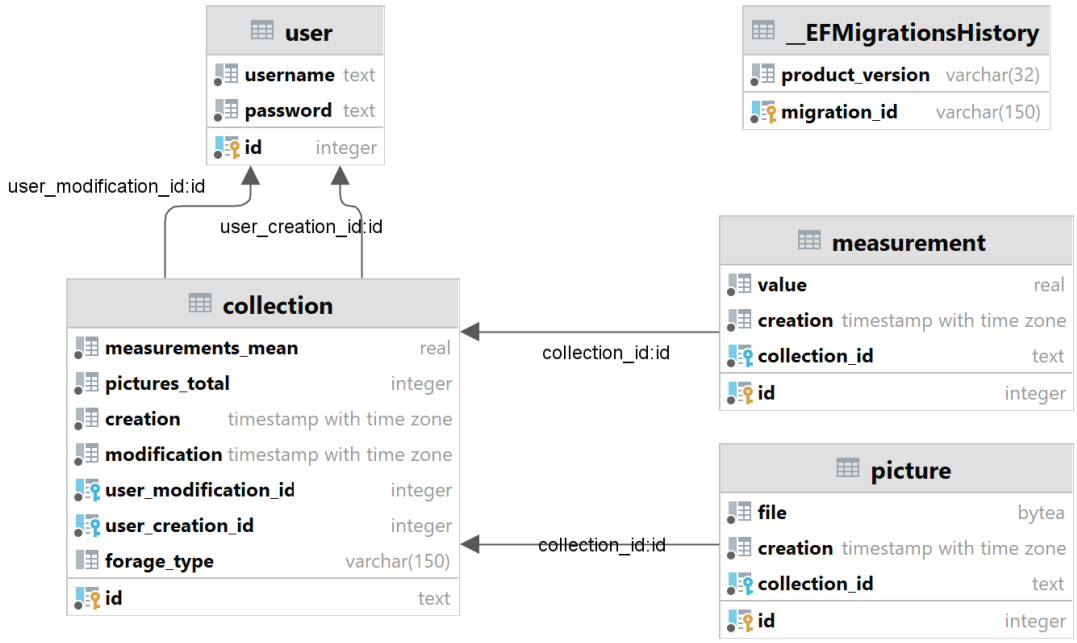


Fonte: Autor (2024)

Para o armazenamento dos dados foi desenvolvido um banco de dados relacional baseado no PostgreSQL, que é um sistema de gerenciamento de bancos de dados (SGBD) robusto e *open source*. A Figura 5 apresenta o modelo relacional do banco de dados, fornecendo uma visão de sua estrutura. Nesse modelo, uma coleta pode conter uma ou mais medições e uma ou mais fotos. Essa estrutura permite que os dados sejam extraídos com facilidade para o treinamento da rede neural.

A hospedagem do *back end* deste projeto empregou serviços da Amazon Web Services (AWS). A API é executada em uma instância EC2 do tipo t2.micro, enquanto o banco de dados PostgreSQL é gerenciado por meio do Amazon RDS. Ambas as instâncias foram configuradas dentro do AWS Free Tier, permitindo a implantação econômica durante o desenvolvimento. A escolha da AWS garante escalabilidade e confiabilidade

Figura 5 – H-Pasture Collector – modelo relacional



Fonte: Autor (2024)

à infraestrutura do sistema.

2.4 Implementação, treinamento e validação das redes neurais

O treinamento e a validação das redes neurais implementadas neste trabalho foram realizados na plataforma Google Colab (Google Research, 2017), que oferece uma infraestrutura de desenvolvimento na nuvem com suporte à linguagem Python (Python Software Foundation, 2024) e a diversas bibliotecas de aprendizado de máquina. Essa escolha se deu pela flexibilidade e escalabilidade que o Google Colab proporciona, permitindo o acesso a recursos computacionais de alto desempenho, sem a necessidade de configurar servidores locais. Adicionalmente, a integração com o Google Drive facilitou o armazenamento e compartilhamento de dados e modelos, otimizando o fluxo de trabalho.

A biblioteca Keras (CHOLLET et al., 2015) foi empregada na implementação dos modelos. Keras é uma biblioteca de código aberto para construção, treinamento e avaliação de redes neurais, operando sobre a plataforma TensorFlow (ABADI et al., 2015). A escolha do Keras se deve a sua facilidade de uso e capacidade de criar modelos de redes neurais com hiperparâmetros otimizados.

Este trabalho buscou desenvolver modelos de redes neurais convolucionais para estimar a altura média de determinada área para as vegetações: campo nativo, azevém e capim-sudão. Para isso, foram testados três modelos para cada espécie, utilizando a base convolucional das arquiteturas VGG16 (SIMONYAN; ZISSERMAN, 2015), Xception (CHOLLET, 2017) e EfficientNetV2 (TAN; LE, 2021). Cada modelo foi inicializado com pesos pré-treinados na ImageNet (DENG et al., 2009), conforme a configuração padrão da biblioteca Keras. No entanto, as camadas convolucionais foram mantidas como treináveis, o que permitiu que as redes se adaptassem às características do problema.

A biblioteca Keras Tuner (O'MALLEY et al., 2019) foi utilizada para otimizar os hiperparâmetros das redes neurais, explorando configurações como o número de camadas totalmente conectadas, quantidade de neurônios e taxa de aprendizado. A ferramenta aplicou otimização bayesiana para conduzir a busca por combinações de hiperparâmetros para cada modelo. No total, foram realizadas 9 execuções de otimização, considerando os 3 tipos de vegetação e as 3 arquiteturas base. Em cada execução, foram testadas até 20 combinações de hiperparâmetros. O *early stopping* foi configurado com *patience* de 5, interrompendo o treinamento caso o desempenho não melhorasse. Cada tentativa teve um máximo de 30 épocas de treinamento. Maiores detalhes sobre a implementação e o processo de otimização das redes, incluindo a definição do espaço de busca de hiperparâmetros, são apresentados na Seção 5.4

O método *hold-out* foi empregado para treinamento e validação das redes, que é uma das formas mais simples e amplamente utilizadas de validação de modelos em aprendizado de máquina (AGGARWAL, 2018; NURHAYATI et al., 2014; AWWALU; OGWUELEKA, 2019). Nesse método, o conjunto de dados é dividido em duas partes principais: um conjunto de treinamento e um conjunto de teste. A divisão é feita de modo que uma parte dos dados seja usada para treinar o modelo e a outra parte seja usada para avaliar o desempenho do modelo.

Normalmente, a divisão padrão envolve a alocação de uma porcentagem dos dados ao conjunto de teste e o restante aos dados de treinamento, sendo comum uma divisão entre 70% e 80% dos dados para treinamento e 30% a 20% dos dados para teste. A proporção costuma variar dependendo do tamanho do conjunto de dados e dos requisitos do projeto.

A ideia principal por trás do *hold-out* é que o conjunto de treinamento é usado para treinar o modelo, enquanto o conjunto de teste é usado para avaliar o quão bem o modelo generaliza para dados não vistos. Isso ajuda a estimar o desempenho do modelo

em situações da vida real, onde ele será usado para fazer estimativas em novos dados. Neste trabalho, os dados foram divididos em 80% para treinamento e 20% para teste.

Para a avaliação individual do desempenho de cada modelo, foram empregadas as seguintes métricas: *mean absolute error (MAE)* (ou erro médio absoluto), *mean squared error (MSE)* (ou erro médio quadrático) e a média dos resíduos. O coeficiente de correlação foi calculado para examinar a relação entre as estimativas do modelo e os valores reais, permitindo avaliar a qualidade das estimativas em termos de direção e intensidade.

O MAE mensura a média dos valores absolutos das diferenças entre as os valores reais e as estimativas do modelo neural na medição da altura média da vegetação (como é o caso neste trabalho). A expressão do cálculo do valor do MAE é apresentada na Eq. (1).

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{1}$$

onde n é o número de amostras, y_i representa o valor real e $\hat{y}_i$ é a estimativa feita pelo modelo. O MAE foi utilizado nas avaliações finais dos modelos por possuir uma interpretação mais direta e intuitiva.

O MSE expressa a média das diferenças quadráticas entre os valores reais e as estimativas do modelo. A Eq. (2) descreve o cálculo do MSE.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2}$$

O MSE foi utilizado nos treinamentos para incentivar os modelos a se ajustarem melhor aos erros maiores, devido à natureza quadrática dessa métrica.

O resíduo (Eq. (3)) é definido como a diferença entre o valor real (y_i) e a estimativa do modelo ($\hat{y}_i$) para uma determinada amostra, representando o erro individual da estimativa (BLUMAN, 2018). A média dos resíduos, por sua vez, calcula o valor médio desses erros ao longo de todas as n amostras avaliadas. Essa métrica foi utilizada para verificar a tendência dos modelos em superestimar ou subestimar as estimativas de altura média da vegetação. A expressão para o cálculo da média dos resíduos é apresentada na Eq. (4).

$$r_i = y_i - \hat{y}_i \tag{3}$$

$$\text{média dos resíduos} = \frac{1}{n}\sum_{i=1}^{n} r_i \tag{4}$$

Para comparar o desempenho dos modelos desenvolvidos para cada espécie de vegetação, foi realizada uma análise estatística utilizando análise de variância (ANOVA) com um nível de significância de 5%. A ANOVA é uma técnica estatística que permite avaliar se há diferenças significativas entre as médias de três ou mais grupos (HERZOG; FRANCIS; CLARKE, 2019). Neste estudo, cada grupo corresponde a um modelo de rede neural aplicado a uma das espécies de vegetação (nativa, azevém ou capim-sudão). A ANOVA foi aplicada sobre o módulo dos resíduos extraídos das estimativas realizadas no conjunto de teste. Assim, considerando que o MAE é a média do módulo dos resíduos, a análise buscou verificar se as diferenças no MAE entre os modelos são estatisticamente significativas.

Quando a ANOVA indicou diferenças significativas entre os modelos, foi aplicado o teste de Tukey como uma análise adicional para comparar diretamente os modelos entre si. O teste de Tukey permite identificar quais pares de modelos apresentam diferenças significativas (MONTGOMERY, 2019). Entretanto, quando os dados não atenderam ao pressuposto de normalidade, foi utilizado o teste de Kruskal-Wallis como alternativa não paramétrica para avaliar diferenças entre os grupos. Nesse caso, o teste de Nemenyi foi empregado como análise pós-hoc para identificar quais pares de modelos apresentaram diferenças significativas. Portanto, os testes estatísticos permitiram identificar os modelos que se destacaram para cada espécie, considerando as variações observadas nos erros absolutos.

3 TRABALHOS RELACIONADOS

3.1 Análise dos achados da literatura

A evolução das tecnologias de sensoriamento remoto e aprendizado de máquina tem proporcionado avanços no manejo de pastagens e na agricultura de precisão. O foco deste trabalho é o uso de visão computacional para estimativa da altura da vegetação no pasto. Dessa forma, o protocolo de revisão bibliográfica apresentado na Seção 2.2 teve o foco na busca de trabalhos centrados na medição da altura da vegetação por meio de imagens. A aplicação do protocolo revelou um total de 14 (quatorze) artigos relacionados.

A análise da literatura consultada é apresentada nesta seção. Ao explorar essas pesquisas, uma variedade de abordagens foi identificada, abrangendo desde métodos baseados em visão computacional até técnicas baseadas em sensoriamento remoto. Embora apenas dois desses estudos tenham incorporado o uso da IA para medir a altura, esses exemplos destacam o potencial da IA na obtenção dessas medidas em ambientes vegetativos. A síntese dos achados na literatura é apresentada na próxima seção e sumarizada na Tabela 5.

O trabalho de Silveira et al. (2022), intitulado "Aprendizado de máquina com base na resposta espectral de imagens aéreas obtidas por VANTs e aplicado no manejo de pastagens", utiliza *unmanned aerial vehicle (UAV)* para capturar imagens aéreas de pastagens e emprega o algoritmo *Random Forest* para classificar quatro classes de manejo do pasto, a saber: pré-pastejo, em pastejo, pós-pastejo e solo exposto. As imagens capturadas foram analisadas para identificar a resposta espectral associada a cada classe de manejo. O objetivo era monitorar e estimar a cobertura e a altura do pasto, proporcionando uma ferramenta para os produtores rurais. Os resultados mostraram que a técnica utilizada alcançou classificações satisfatórias para o manejo do capim-piatã sob pastejo rotacionado.

Embora este trabalho utilize uma metodologia diferente, focando na determinação direta da altura média do pasto por meio de CNN em um problema de regressão, o estudo de Silveira et al. (2022) forneceu percepções sobre a aplicação de técnicas de aprendizado de máquina em contextos agrícolas. Assim, sublinhamos a importância de pesquisas anteriores que pavimentaram o caminho para inovações. Este trabalho busca expandir o conhecimento existente, aplicando CNN para determinar a altura média do pasto diretamente, demonstrando assim como diferentes abordagens de aprendizado de

máquina podem ser adaptadas e aplicadas para resolver desafios no manejo de pastagens.

Uma técnica que se destaca como amplamente utilizada nos trabalhos encontrados é o *structure from motion (SfM)*. SfM é uma técnica de fotogrametria que utiliza imagens aéreas sobrepostas para criar um modelo tridimensional da superfície do terreno e da vegetação (NYIMBILI et al., 2016). Neste contexto, cinco artigos foram destacados, e todos eles exploram o uso do SfM em combinação com UAV para medir a altura da vegetação.

O primeiro artigo, intitulado "Brazilian Savanna Height Estimation Using UAV Photogrammetry" (DUTRA et al., 2021), concentra-se na aplicação dessa técnica na região do Cerrado brasileiro, explorando a altura da copa das árvores em savanas e campos. Os resultados demonstram um potencial para o uso da fotogrametria de UAV na monitorização da estrutura da vegetação e na implementação de práticas de manejo ambiental. O artigo também discute as vantagens e desvantagens da fotogrametria de UAV em relação aos métodos tradicionais de estimativa de altura da copa. Neste artigo não é apresentada análise quantitativa dos resultados.

O segundo artigo, "Estimating pasture biomass and canopy height in Brazilian Savanna using UAV photogrammetry" (BATISTOTI et al., 2019), explora a estimativa da biomassa e da altura das pastagens na savana brasileira. Mais uma vez, a fotogrametria por UAV é empregada para medir a vegetação em áreas remotas, com uma comparação entre os resultados obtidos e os métodos tradicionais de medição de pastagens. O artigo também discute as implicações práticas dessas medições para a gestão das pastagens na savana brasileira. Em resultados, é apresentado um coeficiente de determinação de 0,80 entre a altura medida por régua e por UAV.

O terceiro artigo, "Estimation of the Plant Height for Energy Crops Using UAV-SfM Method" (HASEGAWA et al., 2022), se concentra na medição da altura de culturas energéticas, como a cana-de-açúcar (*Saccharum officinarum*). O estudo avalia a precisão dessa técnica em comparação com medições reais de altura e oferece informações sobre a altitude ideal para capturar imagens das culturas. O artigo apresenta coeficiente de determinação de 0,95 para estimar a altura da vegetação.

O quarto artigo, "Using UAV Borne, Multi-Spectral Imaging for the Field Phenotyping of Shoot Biomass, Leaf Area Index and Height of West African Sorghum Varieties under Two Contrasted Water Conditions" (GANO et al., 2021) se concentra no uso de UAV e imagens multiespectrais para estimar a altura das plantas de sorgo. No estudo, é apresentado um coeficiente de determinação de 0,83 para as estimativas de

altura baseadas em UAV. Em geral, o estudo demonstra o potencial de imagens baseadas em UAV para a fenotipagem de culturas, incluindo a estimativa de altura das plantas.

Por fim, no quinto artigo, intitulado “Rice Height Monitoring between Different Estimation Models Using UAV Photogrammetry and Multispectral Technology” (LU; OKAYAMA; KOMATSUZAKI, 2022), os autores introduziram uma abordagem que incorporou não apenas SfM, mas também outras técnicas complementares para estimar a altura da vegetação de arroz irrigado por inundação. Em particular, os pesquisadores exploraram três métodos. O primeiro método empregou uma câmera RGB de alta resolução para adquirir imagens e estabelecer o *canopy height model (CHM)*. O segundo método adotou um sensor batimétrico conectado a uma nuvem de dados para detectar o nível da água, além do uso de imagens RGB de alta resolução na criação do CHM. No terceiro método, uma câmera multiespectral montada em um UAV foi utilizada para capturar imagens multiespectrais. Essas imagens foram então combinadas com os valores *soil plant analysis development (SPAD)*, que indicam o conteúdo de clorofila nas folhas. Essa fusão de dados serviu para estabelecer o MLR. O terceiro método apresentou coeficiente de determinação de 0,838 para estimativa geral de altura medida.

Além da técnica de SfM, três artigos aplicaram especificamente modelos de regressão para a predição da altura da vegetação. Um desses artigos, “Evaluation of Multiorbital SAR and Multisensor Optical Data for Empirical Estimation of Rapeseed Biophysical Parameters” (ALLIES et al., 2021), demonstra a utilidade dos dados de *synthetic aperture radar (SAR)* na estimativa da altura (entre outros parâmetros biofísicos) da cultura da colza (*Brassica napus*). Os autores introduziram novos índices baseados na soma cumulativa de cada índice de sensoriamento remoto, o que se demonstrou eficaz para aprimorar a estimativa da altura da colza. Os melhores resultados para predição de altura foram obtidos com $\eta\sigma VH$ (um novo índice proposto no estudo), com coeficiente de determinação de 0,87 e *root mean squared error (RMSE)* de 21,19 cm.

Outro artigo, “Normalizing the Local Incidence Angle in Sentinel-1 Imagery to Improve Leaf Area Index, Vegetation Height, and Crop Coefficient Estimations” (KAPLAN et al., 2021), foca em melhorar a precisão da estimativa na altura (entre outros parâmetros biofísicos) da cultura do trigo, tomate para processamento e algodão. Isso é feito por meio da normalização do ângulo de incidência local em imagens da missão Sentinel-1, resultando em modelos mais precisos em comparação com os índices de vegetação tradicionais. Seguindo as transformações sugeridas pelo artigo, coeficiente de determinação aumentou de 0,0172 a 0,668 e o RMSE melhorou de 5% a 52%. Por

exemplo, para a altura do algodão, utilizando o método de normalização β_0, o coeficiente de determinação melhorou de 0,278 para 0,9467, enquanto o RMSE foi reduzido para 5 cm, representando uma melhora de 38%.

O último artigo que emprega exclusivamente modelos de regressão, intitulado "Using Sentinel-1 and Sentinel-2 imagery for estimating cotton crop coefficient, height, and Leaf Area Index" (KAPLAN et al., 2023), apresenta vários modelos que estimam a altura da vegetação a partir de imagens de satélite, utilizando índices de vegetação como *normalized difference vegetation index (NDVI)*, *enhanced vegetation index (EVI)* e *soil-adjusted vegetation index (SAVI)*. Esses modelos são calibrados com dados de campo para estimar a altura da vegetação em diferentes estágios de crescimento da cultura do algodão, embora a precisão possa variar devido a fatores como densidade da vegetação, sombras e qualidade das imagens de satélite. O modelo mais robusto para estimar a altura, baseado na missão Sentinel-2, atingiu um coeficiente de determinação de 0,8883 e RMSE de 10 cm.

"An algorithm to automate the filtering and classifying of 2D LiDAR data for site-specific estimations of canopy height and width in vineyards" (CHERAïET et al., 2020) se destaca como o único a fazer uso de dados *light detection and ranging (LiDAR)*. Este trabalho apresenta um novo algoritmo denominado *bayesian point cloud classification (BPCC)* para processar os dados e estimar a altura e largura do dossel em vinhedos. O algoritmo utiliza uma combinação de filtragem automática e classificação baseada em agrupamento para estimar as dimensões do dossel. Os resultados mostraram que o BPCC é comparável ao método manual e semi-automático, sendo uma opção promissora para o processamento LiDAR em tempo real. No entanto, aprimoramentos são necessários, especialmente na determinação de limiares e seleção de classes para diferentes estágios fenológicos e tamanhos de copas. O algoritmo apresentado no estudo mostrou forte correlação com o método PROTOLIDAR nas estimativas de altura e largura da vegetação, com coeficiente de determinação de 0,94 e 0,89, respectivamente.

A Tabela 5 ainda inclui cinco artigos que exploram o uso de dados de retroespalhamento de radar obtidos por satélites com o propósito de estimar a altura da vegetação na superfície terrestre. Estes artigos utilizam técnicas diferentes das descritas anteriormente, sendo que dois utilizam redes neurais. O primeiro deles, intitulado "Application of Sentinel-1 SAR-derived Vegetation Descriptors for Soil Moisture Retrieval and Plant Height Prediction During the Wheat Growth Cycle" (DAVE et al., 2023), concentra-se no uso de dados SAR da missão Sentinel-1 para prever a

umidade do solo e a altura das plantas em campos de trigo. Esse estudo emprega dados de campo coletados em oito datas, sincronizadas com as passagens do satélite Sentinel-1A, para medir a umidade do solo e a altura das plantas. A partir dos dados SAR, são derivados descritores de vegetação e parâmetros de umidade do solo para realizar essas previsões. Os resultados destacam que os índices de desempenho mais significativos na previsão da altura são um coeficiente de correlação de 0,763 e um RMSE de 0,214 m.

A Tabela 5 resume os dados extraídos dos artigos apresentados nesta revisão bibliográfica. Na primeira coluna é apresentado o nome do artigo, na segunda coluna a técnica utilizada e, na terceira coluna, o tipo de vegetação estudada no artigo.

O segundo artigo, "Application of the Trace Coherence to HH-VV PolInSAR TanDEM-X Data for Vegetation Height Estimation" (ROMERO-PUIG; MARINO; LOPEZ-SANCHEZ, 2022), investiga, pela primeira vez, a inclusão do operador de Coerência de Traço (TrCoh) em metodologias SAR para a estimativa de parâmetros biofísicos da vegetação. Ele apresenta um algoritmo de inversão modificado com base no modelo de *random volume over ground (RVoG)*, que utiliza o TrCoh e soluções analíticas para melhorar a precisão. A validação é realizada com dados do satélite TanDEM-X sobre uma área de arrozal na Espanha, comparando os resultados com o algoritmo *polarimetric interferometric synthetic aperture radar (PolInSAR)*. Com o método convencional PolInSAR, foi obtido RMSE de 23 cm e um coeficiente de determinação de 0,48. Por outro lado, o método proposto alcançou um RMSE de 16 cm e um coeficiente de determinação de 0,58. É importante destacar que esses números se referem a plantas de arroz com altura superior a 25 cm.

No terceiro artigo, "Spatio-Temporal Estimation of Rice Height Using Time Series Sentinel-1 Images" (YANG et al., 2022), foram utilizadas imagens de séries temporais do satélite Sentinel-1A para estimar a distribuição espaço-temporal da altura do arroz, um atributo biofísico utilizado na estimativa de fenologia de culturas e produtividade. Este estudo comparou a aplicação de um *particle filter (PF)* em tempo real com um *simplified water cloud model (SWCM)* com base no mapeamento e data de transplante de arroz. Os resultados indicam que a estimativa da altura do arroz pelo PF obteve um RMSE de 7,36 cm e um coeficiente de determinação de 0,95, superando o SWCM (RMSE = 12,59 cm e coeficiente de determinação = 0,86).

Os últimos dois artigos estão dentro do contexto de redes neurais. Em "A comparative analysis of SLR, MLR, ANN, XGBoost and CNN for crop height estimation of sunflower using Sentinel-1 and Sentinel-2" (ABDIKAN et al., 2023), o estudo explora

a estimativa da altura das culturas de girassol utilizando dados de sensoriamento remoto das missões Sentinel-1 e Sentinel-2. A pesquisa é composta por três partes principais: preparação de dados e modelos, análise de dados e comparação de resultados. Diversos métodos, incluindo *simple linear regressionl (SLR)*, MLR, *artificial neural network (ANN)*, *eXtreme gradient boosting (XGBoost)* e CNN foram usados para a estimativa da altura das culturas. Diferentes recursos dos dados Sentinel-1 e Sentinel-2 foram combinados e cenários foram compostos para análise dos dados. O estudo não detalhou os aspectos arquiteturais da ANN e CNN, no entanto, é mencionada a utilização de uma CNN unidimensional. A CNN utiliza filtros convolucionais unidimensionais e consiste em camadas de entrada, convolução, achatamento (*flatten*), camadas densas e uma camada de saída. Quanto aos parâmetros da CNN, a função de ativação utilizada foi o *rectified linear units (ReLU)*, o algoritmo de otimização foi o Adam, o número de épocas de treinamento foi 50, e o tamanho do lote (*batch size*) foi definido como 2048. Nos resultados do estudo, a ANN obteve o menor RMSE de 3,083 cm no período de alongamento do caule. A CNN teve o menor RMSE nas etapas de desenvolvimento da inflorescência e floração, com valores de 19,223 cm e 8,731 cm, respectivamente. Para o período de amadurecimento, o XGBoost alcançou o menor RMSE de 8,731 cm.

Finalmente, "Crop Height Estimation Using RISAT-1 Hybrid-Polarized Synthetic Aperture Radar Data" (CHAUHAN; SRIVASTAVA; PATEL, 2019) explora o potencial de usar dados de radar de abertura sintética híbrido-polarizado para estimar a altura de culturas de trigo em países do sul da Ásia. Para isso, foi utilizado *canopy height model (WCM)* modificado para gerar dados de retroespalhamento da vegetação, que foram então usados como entrada para duas redes neurais artificiais para estimar a altura da cultura. Neste estudo, embora não tenham sido fornecidos detalhes sobre a arquitetura específica da rede neural, os resultados demonstraram um coeficiente de determinação de 0,68.

Tabela 5 – Síntese de trabalhos relacionados

Artigo	**Técnica utilizada**	**Vegetação**
Brazilian Savanna Height Estimation Using UAV Photogrammetry (DUTRA et al., 2021)	*Structure from motion (SfM)*	Savana
Estimating pasture biomass and canopy height in Brazilian Savanna using UAV photogrammetry (BATISTOTI et al., 2019)	*Structure from motion (SfM)*	Savana

continua na próxima página

Tabela 5 – Síntese de trabalhos relacionados (cont.)

Artigo	Técnica utilizada	Vegetação
Estimation of the Plant Height for Energy Crops Using UAV-SfM Method (HASEGAWA et al., 2022)	*Structure from motion (SfM)*	Plantações
Using uav borne, multi-spectral imaging for the field phenotyping of shoot biomass, leaf area index and height of west african sorghum varieties under two contrasted water conditions (GANO et al., 2021)	*Structure from motion (SfM)*	Plantações
Rice Height Monitoring between Different Estimation Models Using UAV Photogrammetry and Multispectral Technology (LU; OKAYAMA; KOMATSUZAKI, 2022)	*Multiple linear regression (MLR)*, *Structure from motion (SfM)*	Plantações
Evaluation of Multiorbital SAR and Multisensor Optical Data for Empirical Estimation of Rapeseed Biophysical Parameters (ALLIES et al., 2021)	Regressão Polinomial	Plantações
Normalizing the local incidence angle in sentinel-1 imagery to improve leaf area index, vegetation height, and crop coefficient estimations (KAPLAN et al., 2021)	Regressão polinomial	Plantações
Using Sentinel-1 and Sentinel-2 imagery for estimating cotton crop coefficient, height, and Leaf Area Index (KAPLAN et al., 2023)	Regressão polinomial	Plantações
An algorithm to automate the filtering and classifying of 2D LiDAR data for site-specific estimations of canopy height and width in vineyards (CHERAïET et al., 2020)	*Bayesian point cloud classification (BPCC)*, PROTOLIDAR	Plantações
Application of sentinel-1 SAR-derived vegetation descriptors for soil moisture retrieval and plant height prediction during the wheat growth cycle (DAVE et al., 2023)	Inversão modificada do *canopy height model (WCM)*	Plantações
Application of the Trace Coherence to HH-VV PolInSAR TanDEM-X Data for Vegetation Height Estimation (ROMERO-PUIG; MARINO; LOPEZ-SANCHEZ, 2022)	Inversão do *random volume over ground (RVoG)*, *Particle filter (PF)*	Plantações

continua na próxima página

Tabela 5 – Síntese de trabalhos relacionados (cont.)

Artigo	Técnica utilizada	Vegetação
Spatio-Temporal Estimation of Rice Height Using Time Series Sentinel-1 Images (YANG et al., 2022)	Inversão do *simplified water cloud model (SWCM)*	Plantações
A comparative analysis of SLR, MLR, ANN, XGBoost and CNN for crop height estimation of sunflower using Sentinel-1 and Sentinel-2 (ABDIKAN et al., 2023)	*EXtreme gradient boosting (XGBoost)*, *Artificial neural network (ANN)*, *Convolutional neural network (CNN)*, *Multiple linear regression (MLR)*, *Simple linear regressionl (SLR)*	Plantações
Crop Height Estimation Using RISAT-1 Hybrid-Polarized Synthetic Aperture Radar Data (CHAUHAN; SRIVASTAVA; PATEL, 2019)	*Artificial neural network (ANN)*	Plantações

3.2 Síntese dos resultados

A análise dos artigos encontrados na revisão evidenciou que a literatura apresenta tendências e lacunas no campo da estimativa da altura da vegetação com base na análise de imagens fotográficas. Vários artigos (BATISTOTI et al., 2019; DUTRA et al., 2021; GANO et al., 2021; HASEGAWA et al., 2022; LU; OKAYAMA; KOMATSUZAKI, 2022) destacam a técnica de SfM em combinação com UAV para medir a altura da vegetação em cenários de savana e plantações. Alguns desses estudos também forneceram análises quantitativas dos resultados, com coeficiente de determinação variando de 0,80 a 0,95, caracterizando a precisão dessas abordagens para o cenário estudado.

A aplicação de modelos de regressão para a estimativa da altura da vegetação também foi abordada nos artigos (ALLIES et al., 2021; KAPLAN et al., 2021; LU; OKAYAMA; KOMATSUZAKI, 2022; KAPLAN et al., 2023). Esses modelos mostraram resultados promissores para os cenários estudados (com coeficiente de determinação

variando de 0,838 a 0,9467, especialmente quando combinados com dados de sensores remotos, como SAR e imagens multiespectrais. Os resultados quantitativos obtidos em diversos estudos destacam a capacidade desses modelos em prever a altura da vegetação em plantações.

(YANG et al., 2022; ROMERO-PUIG; MARINO; LOPEZ-SANCHEZ, 2022; DAVE et al., 2023) implementam inversão customizada de modelos físicos, como WCM e RVoG. Nestes trabalhos o coeficiente de determinação varia entre 0,58 e 0,86. Em (ROMERO-PUIG; MARINO; LOPEZ-SANCHEZ, 2022) é aplicado o método PF, alcançando coeficiente de determinação de 0,95.

Em (CHERAïET et al., 2020), o estudo introduz um algoritmo BPCC, que usa filtragem automática e classificação por agrupamento para analisar dados LiDAR. Este algoritmo foi testado contra métodos manuais e semi-automáticos (PROTOLIDAR). Mostrou forte correlação com o método PROTOLIDAR nas estimativas de altura e largura da vegetação, com coeficiente de determinação de 0,94 e 0,89, respectivamente.

O uso de aprendizado de maquina como método para estimar a altura da vegetação é destacado em dois trabalhos. (ABDIKAN et al., 2023) compara diferentes técnicas, incluindo ANN e CNN, para estimar a altura de culturas de girassol utilizando dados dos satélites Sentinel-1 e Sentinel-2. O menor RMSE para o período de alongamento do caule foi alcançado pela ANN (3,083 cm), enquanto para CNN, os menores RMSE foram registrados durante as fases de desenvolvimento da inflorescência e florescimento (19,223 cm e 8,731 cm, respectivamente).

Em outro estudo (CHAUHAN; SRIVASTAVA; PATEL, 2019), a integração de dados SAR híbrido-polarizados da RISAT-1 com WCM modificado e redes neurais foi utilizada para a estimativa da altura da cultura de trigo. Essa metodologia alcançou um coeficiente de determinação de 0,68.

A maioria dos artigos revisados concentra-se em plantações, que apresentam topografia e distribuição de vegetação mais uniformes. O escopo deste trabalho (assim como em (SILVEIRA et al., 2022)) abrange a aplicação em pastagens. O pastejo é um processo dinâmico, que modifica a estrutura das plantas que estão na área, afetando a medida da altura, ao contrário das plantas cultivadas para grão, que não sofrem remoção de tecido. Essa dinâmica adicional, com a presença de animais que interagem com a vegetação, impõe desafios na estimativa da altura da vegetação. Este trabalho busca apresentar uma nova contribuição para o campo, uma vez que se propõe a preencher algumas dessas lacunas ao explorar a aplicação de CNN em conjunto com imagens

fotográficas para a predição da altura da vegetação em pastagens típicas dos Campos Sulinos, levando em consideração a maior heterogeneidade na topografia do terreno e na altura da vegetação. Espera-se que essa abordagem ajude no desenvolvimento de práticas agrícolas mais eficientes e no manejo sustentável de pastagens nesta região.

4 REDES NEURAIS CONVOLUCIONAIS

4.1 Redes Neurais Artificiais

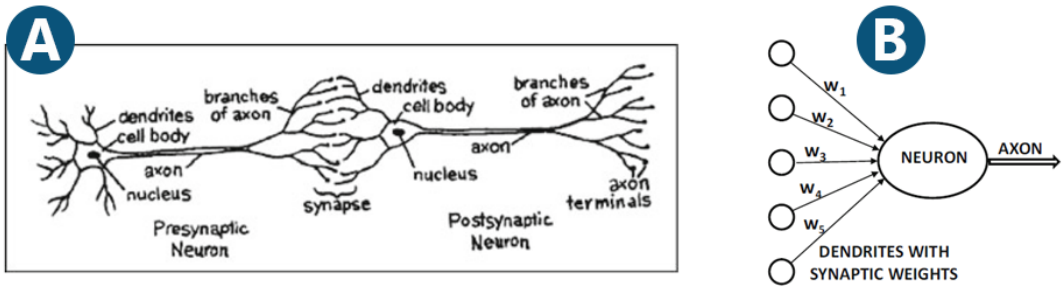
As redes neurais artificiais (RNA, ou ANN, do Inglês *artificial neural networks*) consistem em uma técnica de aprendizado de máquina que busca reproduzir, de maneira sintética e simplificada, os processos de aprendizagem que ocorrem em organismos biológicos.

No sistema nervoso humano, os neurônios são as células especializadas responsáveis por esse processo. Os neurônios conectam-se uns aos outros por meio de estruturas chamadas axônios e dendritos, formando uma rede complexa que opera a comunicação de informação por meio de estímulos elétricos. As regiões onde as conexões entre axônios e dendritos ocorrem são chamadas de sinapses. As sinapses formam o meio de ajuste à força das conexões, que produzem resposta a estímulos, modulando a transmissão de sinais. Conforme ilustrado na Figura 6(a), as sinapses são pontos de contato onde a força das conexões pode ser alterada, permitindo que o aprendizado ocorra nos organismos vivos (AGGARWAL, 2018).

A unidade de processamento de informação base para a operação de uma rede neural é o neurônio artificial, Figura 6(b), que, tal como seu análogo biológico, recebe entradas, processa essas informações e gera uma saída. O modelo de neurônio artificial produz uma combinação linear de suas entradas (originárias de outros neurônios ou de uma entrada independente de dados) com os pesos das conexões entre os neurônios, inspirados nas sinapses dos modelos biológicos. O disparo do sinal da conexão, em sistemas biológicos, não é linear com a entrada, mas exige um certo limiar para sua ativação. Esse processo é modelado por meio de uma função não linear que tem como argumento a saída do neurônio (usualmente funções diferenciáveis que mimetizam a função degrau matemática, como a função sigmóide ou a função arcotangente).

Uma *feedforward neural network (FNN)*, retratada pela Figura 7, segue uma organização em que as unidades básicas de aprendizado (neurônios artificiais) estão organizadas em uma sequência de camadas, em que todos os neurônios de uma camada estão conectados a todos os neurônios da camada seguinte. Os sinais recebidos pela rede fluem em uma única direção através das camadas, sem retroalimentação – daí a designação *feed forward*, ou alimentada para a frente. A estrutura de uma FNN é composta por três tipos de camadas: a camada de entrada (nodos de cor azul), que recebe

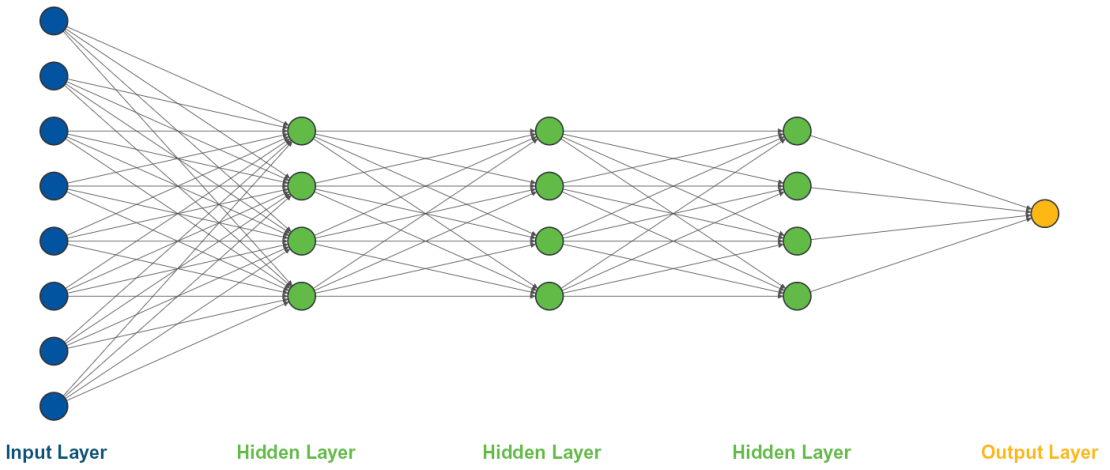
Figura 6 – Organização conceitual de neurônios biológicos e artificiais



Fonte: Aggarwal (2018)

os dados de entrada; as camadas ocultas (nodos de cor verde), formadas por neurônios que processam e transformam os sinais de entrada; e, por fim, a camada de saída (nodo de cor laranja), que gera as respostas da rede com base no processamento realizado nas camadas anteriores (HAYKIN, 1999).

Figura 7 – *Feedforward neural network*



Fonte: Autor (2024)

DeVore, Hanin e Petrova (2021) traz uma apresentação formal da arquitetura de uma FNN $\mathcal{N}$ que, em sua formulação mais geral, é associada a um grafo direcionado acíclico $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, chamado de arquitetura da rede. Esse grafo é composto por um conjunto finito $\mathcal{V}$ de vértices e um conjunto finito de arestas direcionadas $\mathcal{E}$, onde cada vértice $v \in \mathcal{V}$ pertence a pelo menos uma aresta $e \in \mathcal{E}$. Os vértices $\mathcal{V}$ são divididos em três subconjuntos distintos:

- O conjunto $\mathcal{I}$ de vértices de entrada, que não possuem arestas de entrada e representam as variáveis independentes (ou seja, as entradas da rede).

- O conjunto $\mathcal{O}$ de vértices de saída, que não possuem arestas de saída e armazenam os valores das variáveis dependentes correspondentes às entradas fornecidas (ou seja, as saídas da rede).
- O conjunto de vértices ocultos $\mathcal{H} = \mathcal{V} \setminus \{\mathcal{I}, \mathcal{O}\}$, que armazenam valores intermediários usados para calcular as saídas da rede.

Com cada $v \in \mathcal{V} \setminus \mathcal{I}$, está associada uma função de ativação $\sigma_v : \mathbb{R} \to \mathbb{R}$ e um escalar $b_v \in \mathbb{R}$, chamado de viés. Para cada aresta $e \in \mathcal{E}$, existe um escalar $w_e \in \mathbb{R}$, chamado de peso. Os pesos e vieses são os parâmetros treináveis de $\mathcal{N}$. Para uma determinada arquitetura de rede definida e fixa, a variação de valores dos parâmetros treináveis produz uma família de funções de saída. Para descrever como essas funções são construídas, associa-se uma unidade computacional chamada neurônio para cada vértice $v \in \mathcal{V} \setminus \mathcal{I}$. O neurônio recebe como entradas os valores escalares $x_{v'}$ dos vértices $v' \in \mathcal{V} \setminus \mathcal{O}$, com uma aresta $e = (v', v) \in \mathcal{E}$ terminando em v, e gera como saída o valor escalar definido pela Eq. (5).

$$x_v = \sigma_v \left(b_v + \sum_{e=(v',v)\in\mathcal{E}} w_e x_{v'} \right) \tag{5}$$

Um neurônio associado a um vértice $v \in \mathcal{V} \setminus \mathcal{I}$ recebe os sinais $x_{v'}$ calculados por neurônios anteriores associados a v', combina esses sinais por meio dos pesos sinápticos w_e, onde $e = (v', v)$, e gera a saída x_v, que é então transmitida aos neurônios subsequentes. Para todos os neurônios associados a vértices $v \in \mathcal{O}$, a função de ativação σ_v é a função identidade. O neurônio associado ao i-ésimo vértice de entrada $v \in \mathcal{I}$, onde $i = 1, \ldots, d$ e $d = |\mathcal{I}|$, recebe um sinal escalar de entrada (externamente fornecido) x_i e o transmite como saída para os neurônios subsequentes.

Os sinais de entrada escalares x_i, $i = 1, \ldots, d$, são considerados variáveis independentes $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, e a função de saída da rede $\mathcal{N}$ é definida como:

$$S_{\mathcal{N}}(x) = (x_v, v \in \mathcal{O}), \quad d' = |\mathcal{O}|. \tag{6}$$

Assim, $S_{\mathcal{N}}$ é uma função que mapeia $\mathbb{R}^d$ para $\mathbb{R}^{d'}$, chamada de saída da rede $\mathcal{N}$. Para uma arquitetura de rede fixa $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, as saídas $S_{\mathcal{N}}$ formam uma família de funções, determinadas pelos parâmetros treináveis $\{w_e, b_v\}$, onde $e \in \mathcal{E}$ e $v \in \mathcal{V} \setminus \mathcal{I}$.

Em redes neurais, o processo de aprendizado pode ser definido como um procedimento no qual os parâmetros treináveis de uma rede são ajustados por meio de

estímulos do ambiente no qual a rede está inserida. O processo envolve a seguinte sequência de eventos:

1. A rede neural é estimulada pelo ambiente.
2. A rede neural sofre alterações em seus parâmetros treináveis em resposta ao estímulo.
3. A rede neural responde de uma nova maneira ao ambiente devido às mudanças ocorridas em sua estrutura interna.

Um conjunto de regras bem definidas para solucionar um problema de aprendizado é chamado de *algoritmo de aprendizado*. O algoritmo de aprendizado *backpropagation* se tornou padrão em FNN (HAYKIN, 1999). O *backpropagation* pode ser separado em dois estágios distintos. No primeiro, o algoritmo calcula o gradiente de uma função de perda E em relação aos parâmetros treináveis da rede w, isto é, $\nabla E(w)$. Esse gradiente é utilizado na segunda etapa que emprega um algoritmo de otimização, como o *stochastic gradient descent* (SGD), para atualizar os parâmetros treináveis e minimizar o erro da rede (BISHOP, 2006).

O processo de aprendizado é iterativo e envolve atualizações sucessivas dos parâmetros treináveis, sendo que uma época corresponde a uma passagem completa por todo o conjunto de dados de treinamento. O número de épocas é um hiperparâmetro[1] que controla quantas vezes a rede é exposta aos dados de treinamento. Um número insuficiente de épocas pode resultar em *underfitting*, enquanto um número excessivo pode levar ao *overfitting*, quando o modelo se ajusta demasiadamente aos dados de treinamento, comprometendo sua capacidade de generalização. A escolha do número adequado de épocas depende da complexidade do modelo e das características do conjunto de dados (GOODFELLOW; BENGIO; COURVILLE, 2016).

Este trabalho se baseia em redes neurais convolucionais (do Inglês *convolutional neural network*, ou CNN), uma classe de FNN projetada para lidar com dados que possuem uma topologia estruturada em grade, como séries temporais ou imagens, em que os elementos de uma vizinhança estejam relacionados entre si. As CNN, possuem capacidade de invariância em relação a transformações como translação, escala e distorção nos dados de entrada. Isso significa que a rede consegue identificar padrões ou objetos, independentemente de mudanças na posição, no tamanho ou em deformações,

[1] Hiperparâmetros são variáveis definidas antes do treinamento, como taxa de aprendizado, número de épocas e tamanho do *mini-batch*, que influenciam o desempenho do modelo, mas não são ajustadas automaticamente durante o processo de otimização.

o que as torna particularmente eficazes em tarefas de processamento e reconhecimento de imagens. Em suma, uma CNN se diferencia de uma FNN tradicional por possuir pelo menos uma camada convolucional, isto é, uma camada que aplica a operação matemática de convolução (GOODFELLOW; BENGIO; COURVILLE, 2016; HAYKIN, 1999).

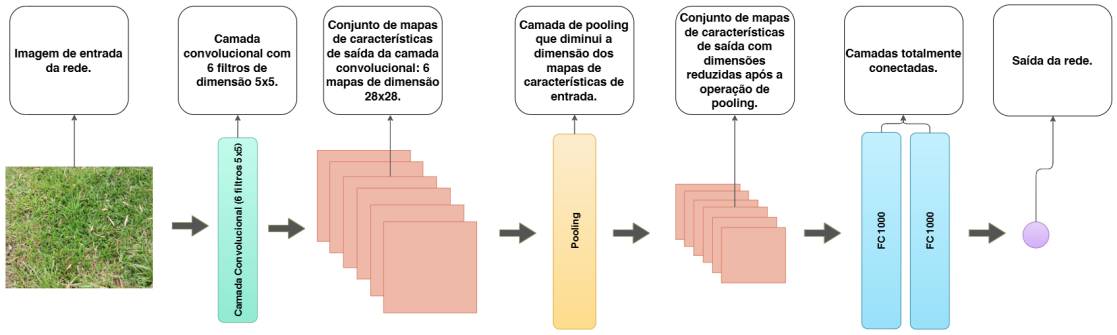
A Figura 8 apresenta uma arquitetura básica de CNN com o intuito de explicar os diferentes tipos de camadas e o funcionamento básico desse tipo de rede. A finalidade da rede de exemplo, assim como neste trabalho, é estimar a altura média da pastagem em determinada foto. A CNN começa recebendo uma imagem de entrada, que será processada pelas camadas seguintes para extrair e combinar características. Primeiramente, a imagem passa por uma camada convolucional, que aplica filtros, representados por matrizes de pesos que a rede aprende durante o treinamento. Esses filtros detectam padrões específicos, como bordas e texturas, gerando um conjunto de mapas de características. Um mapa de características é uma matriz que armazena o resultado da operação de convolução (definida mais a frente) entre um filtro e a imagem de entrada (ou saída de uma camada anterior). Assim, esses mapas refletem a resposta do filtro a cada região de entrada. Na camada convolucional do exemplo, com 6 filtros de dimensão 5×5, são gerados 6 mapas de características, cada um com dimensão 28×28.

Após a camada convolucional, a CNN aplica uma camada de *pooling*, que reduz a dimensão dos mapas de características, trazendo benefícios como a diminuição do custo computacional. Em seguida, as características extraídas são passadas para camadas totalmente conectadas, onde cada neurônio se conecta a todos os neurônios da camada anterior. Essas camadas totalmente conectadas processam as características para gerar a estimativa final da rede. No exemplo, a rede possui duas camadas totalmente conectadas com 1.000 neurônios cada, seguidas por uma camada de saída responsável pela estimativa de um valor escalar, uma vez que essa rede está configurada para resolver um problema de regressão.

Em redes convolucionais mais complexas, camadas de convolução e *pooling* podem ser organizadas em diferentes combinações para ampliar a capacidade de aprendizado e a profundidade da rede. Diferentes arquiteturas de CNN variam nos tipos de camadas, tamanhos de filtros, funções de ativação e níveis de profundidade, adaptando-se às características e demandas específicas de cada problema.

A operação de convolução é utilizada em CNN para extrair características locais em dados com estrutura espacial, temporal ou espaço-temporal. Essas características locais podem incluir, por exemplo, bordas, texturas e contornos presentes em imagens.

Figura 8 – Arquitetura de exemplo de uma *convolutional neural network*



Fonte: Autor (2024)

A convolução de uma entrada bidimensional I com um filtro K pode ser descrita pela equação (GOODFELLOW; BENGIO; COURVILLE, 2016):

$$S(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{7}$$

onde:

- $S(i,j)$: Representa o valor do mapa de características resultante na posição (i,j),
- $I(i+m, j+n)$: Refere-se à entrada bidimensional (por exemplo, uma imagem), deslocada para corresponder à posição do filtro,
- $K(m,n)$: Representa o filtro convolucional aplicado à entrada,
- m e n: São os índices que percorrem as dimensões do filtro.

O cálculo da convolução envolve múltiplas etapas, resultando em um mapa de características que será utilizado nas camadas subsequentes da CNN. As etapas do cálculo da convolução podem ser resumidas em:

1. **Posicionamento do filtro**: Para cada $S(i,j)$, o filtro K é aplicado a uma submatriz da entrada I com o mesmo tamanho que o filtro. Os valores de $I(i+m, j+n)$ correspondam aos elementos da submatriz, onde m e n percorrem as dimensões do filtro.
2. **Multiplicação e soma dos elementos correspondentes**: Com o filtro K posicionado sobre a submatriz da entrada, cada elemento $K(m,n)$ do filtro é multiplicado pelo elemento correspondente $I(i+m, j+n)$ da entrada. Em seguida, todos os produtos resultantes são somados para calcular o valor final de $S(i,j)$, que representa o resultado do filtro aplicado à submatriz atual da entrada.

3. **Geração do mapa de características**: Esse processo de deslocamento do filtro sobre diferentes submatrizes, multiplicação e soma é repetido para cada posição válida (i, j). O conjunto de valores $S(i, j)$ forma o mapa de características.

Embora a Eq. (7) represente a operação de *cross-correlation*, muitas bibliotecas de aprendizado de máquina utilizam o termo "convolução" de forma intercambiável. Em termos práticos, a *cross-correlation* é "igual" à convolução no sentido de que realiza o mesmo cálculo, mas sem o passo adicional de espelhamento do filtro. Esse espelhamento não é necessário no contexto de redes neurais convolucionais, pois os valores do filtro são ajustados automaticamente durante o treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

A Figura 9 apresenta um exemplo de convolução entre uma entrada de dimensões $7 \times 7 \times 1$ (uma imagem com um único canal, como uma imagem em escala de cinza) e um filtro de $3 \times 3 \times 1$. Nessa operação, o filtro percorre a matriz de entrada, multiplicando cada elemento do filtro pelos elementos correspondentes da submatriz na entrada e somando os resultados para formar um único valor. Esse valor é inserido na posição correspondente da matriz de saída (*output*). O filtro se desloca até cobrir cada possível submatriz 3×3 da entrada, repetindo o processo até que toda a matriz tenha sido percorrida. O resultado final é um mapa de características extraído da entrada.

No exemplo da Figura 9, é utilizado um *stride* de 1. *Stride* é o parâmetro que define o número de posições que o filtro avança sobre a entrada durante a operação de convolução. A finalidade do *stride* é controlar a resolução do mapa de características gerado (AGGARWAL, 2018).

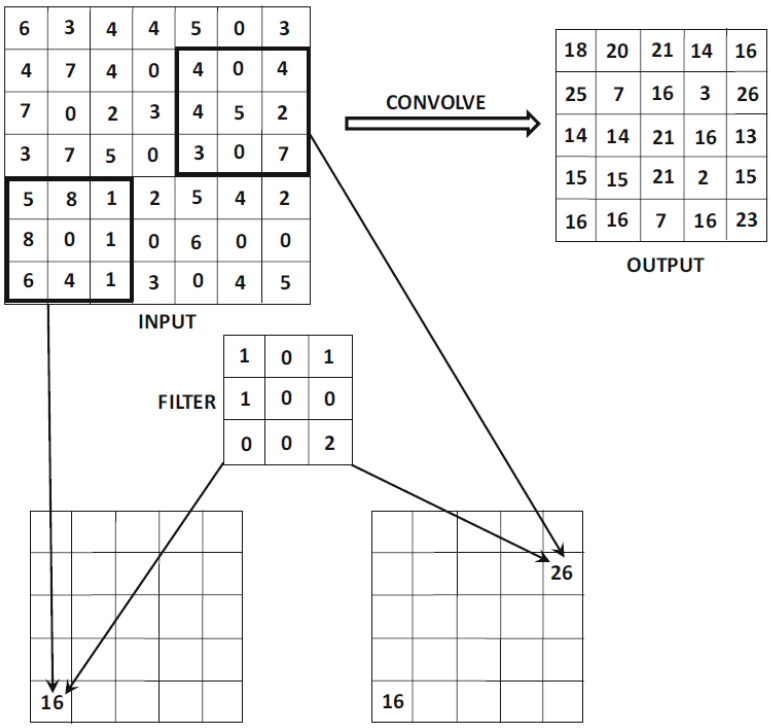
Quando lidamos com entradas que têm múltiplos canais (como imagens coloridas com 3 canais), a convolução envolve filtros multidimensionais que processam todos os canais simultaneamente. O filtro também terá profundidade d, correspondendo ao número de canais da entrada. Nesse caso, a operação de convolução é realizada em todos os canais, e o resultado é somado para gerar um único valor no mapa de características. Para entradas com múltiplos canais, a convolução pode ser descrita pela seguinte equação:

$$S(i, j) = \sum_{d=1}^{D} \sum_{m} \sum_{n} I_d(i + m, j + n) K_d(m, n) \tag{8}$$

onde:

- D é o número de canais da entrada,
- $I_d(i + m, j + n)$ representa o valor do canal d da entrada na posição $(i + m, j + n)$,

Figura 9 – Um exemplo de convolução entre uma entrada de dimensões $7 \times 7 \times 1$ e um filtro de $3 \times 3 \times 1$, com *stride* de 1



Fonte: Aggarwal (2018)

- $K_d(m, n)$ representa o valor do filtro para o canal d na posição (m, n).

Independentemente da profundidade do filtro, um único filtro cria um único mapa de características. Quando utilizamos múltiplos filtros, cada um gera seu próprio mapa de características, e a saída final será um conjunto de mapas, um para cada filtro.

A operação de convolução reduz o tamanho da camada $(q+1)$ em relação à camada q, o que pode resultar na perda de informações nas bordas da imagem ou dos mapas de características. Em geral, essa redução de tamanho não é desejável e, para evitá-la, é utilizada a técnica de *padding*. O *padding* adiciona $\frac{(F_q-1)}{2}$ "pixels" ao redor das bordas do mapa de características, preservando a dimensão da entrada. É comum que esses "pixels" adicionados tenham valor 0, de modo que as regiões preenchidas não contribuam para o produto escalar final. Dessa forma, o *padding* permite que a operação de convolução mantenha a integridade da estrutura espacial da entrada (AGGARWAL, 2018).

Nos últimos anos, diferentes arquiteturas de CNN (CHOLLET, 2016; HOWARD et al., 2017; SANDLER et al., 2018) foram propostas com variações na operação de convolução tradicional, visando melhorar a eficiência. Em Howard et al. (2017), por exemplo, é utilizada a variante *depthwise separable convolution*, que divide a convolução

em duas etapas: primeiro, uma convolução *depthwise*, que aplica um filtro a cada canal de entrada de forma independente; em seguida, uma convolução *pointwise* 1×1, que combina os canais resultantes para produzir a saída final.

Na convolução *depthwise*, é aplicado um filtro individualmente a cada canal de entrada, resultando em um custo computacional em termos de operações de multiplicação de:

$$D_K^2 \cdot M \cdot D_F^2$$

onde:

- D_K^2 é o tamanho do filtro,
- M é o número de canais de entrada,
- D_F^2 é a dimensão espacial do mapa de características de entrada.

Embora a convolução *depthwise* reduza o custo computacional em relação à convolução padrão, ela apenas aplica um filtro para cada canal da entrada individualmente, sem combinar as informações entre os canais. Para realizar essa combinação e gerar novas características, é aplicada uma convolução *pointwise* 1×1, com um custo computacional em termos de operação de multiplicação de:

$$M \cdot N \cdot D_F^2$$

onde:

- M é o número de canais de entrada,
- N é o número de canais de saída,
- D_F^2 é a dimensão espacial do mapa de características de entrada.

O custo total da *depthwise separable convolution* é, portanto, a soma dos custos das convoluções *depthwise* e *pointwise*:

$$D_K^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2$$

Em comparação, o custo de uma convolução padrão, que combina aplicação de filtro e combinação de canais em uma única operação, é:

$$D_K^2 \cdot M \cdot N \cdot D_F^2$$

Portanto, a redução computacional proporcionada pela *depthwise separable convolution* é dada por:

$$\frac{D_K^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2}{D_K^2 \cdot M \cdot N \cdot D_F^2} = \frac{1}{N} + \frac{1}{D_K^2}$$

Para ilustrar, considere um filtro de 3×3 (ou seja, $D_K = 3$) e $N = 32$ canais de saída. Nesse caso, a *depthwise separable convolution* utiliza aproximadamente $14,2\%$ do custo computacional de uma convolução padrão, resultando em uma economia de cerca de $85,8\%$.

Sandler et al. (2018) introduziram o *inverted residual block (MBConv)* como uma variante da convolução tradicional, baseada em *depthwise separable convolution*. No MBConv, em vez de aplicar diretamente as convoluções *depthwise* e *pointwise*, uma camada inicial de expansão com convolução 1×1 aumenta temporariamente o número de canais de entrada por um fator t, ampliando a capacidade de captura de características antes da convolução *depthwise*. Com os canais expandidos, o MBConv aplica uma convolução *depthwise* 3×3, processando cada canal de forma independente. Finalmente, uma camada de projeção, também com convolução 1×1, reduz o número de canais para N, garantindo que o mapa de características final tenha a dimensão desejada. O custo computacional (em termos de operações de multiplicação) total do MBConv, considerando todas essas etapas, é dado por:

$$D_F^2 \cdot t \cdot C_{\text{in}} \cdot \left(C_{\text{in}} + D_K^2 + C_{\text{out}}\right)$$

onde:

- D_F: Tamanho de um dos lados (altura ou largura) do mapa de características quadrado,
- C_{in}: Número de canais de entrada,
- C_{out}: Número de canais de saída,
- t: Fator de expansão que aumenta temporariamente o número de canais de entrada,
- D_K: Tamanho de um dos lados (altura ou largura) do filtro quadrado na convolução *depthwise*.

Em uma CNN, também podemos encontrar a operação de *pooling*. Essa operação é aplicada após a convolução e tem a capacidade de reduzir as dimensões espaciais dos mapas de características gerados pela camada de convolução anterior.

Essa redução permite, por exemplo, diminuir a quantidade de dados processados nas camadas seguintes e aumentar a invariância a translações. Quando utilizada, a operação de *pooling* é executada em cada mapa de características separadamente, mantendo, assim, a profundidade da camada de entrada. No caso do *max-pooling* (um tipo de operação de *pooling*), para cada região quadrada de tamanho $P_q \times P_q$ em cada um dos d_q mapas de características de entrada, é retornado o valor máximo dessa região.

A nova dimensão espacial da camada após o *pooling* depende do tamanho da região $P_q \times P_q$ e do *stride* S_q. Se o *stride* for $S_q = 1$, o *pooling* produz uma nova camada de tamanho $(L_q - P_q + 1) \times (B_q - P_q + 1) \times d_q$, onde L_q e B_q são as dimensões espaciais da camada de entrada. No entanto, para um *stride* $S_q > 1$, as dimensões da nova camada são dadas por (AGGARWAL, 2018):

$$\left(\frac{L_q - P_q}{S_q} + 1\right) \times \left(\frac{B_q - P_q}{S_q} + 1\right) \times d_q \tag{9}$$

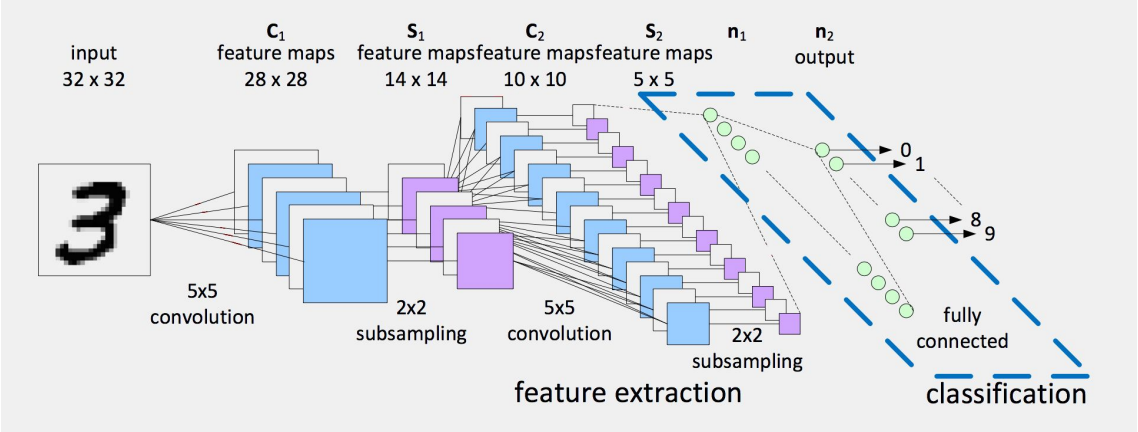
A Figura 10 apresenta um exemplo de funcionamento de uma CNN. Nesta arquitetura, a rede é alimentada com uma imagem de 32×32 *pixels* contendo um único canal de cor. O primeiro processamento é realizado pela camada convolucional C_1, que possui 6 filtros de tamanho 5×5. Cada filtro convoluciona a imagem de entrada, gerando um mapa de características. A dimensão resultante de cada mapa de características é reduzida para 28×28, dada pela fórmula (AGGARWAL, 2018):

$$\begin{aligned} \text{Dimensão de saída} &= \text{Dimensão de entrada} - \text{Tamanho do filtro} + 1 \\ &= 32 - 5 + 1 \\ &= 28 \end{aligned}$$

Essa operação resulta em 6 mapas de características de 28×28, cada um correspondente a um filtro da camada C_1. Após essa etapa, uma camada de *pooling*, chamada S_1, é aplicada com uma região 2×2 e *stride* 2, o que reduz a dimensão de cada mapa de características para 14×14, calculada pela Eq. (9). Em seguida, outra camada convolucional, C_2, com novos filtros, e uma camada de *pooling*, S_2, são aplicadas, produzindo mapas de características com tamanho 5×5 ao final dessas etapas.

Por fim, as camadas totalmente conectadas são ativadas, resultando na saída da rede. Vale ressaltar que, ao contrário do presente trabalho, o exemplo abordado refere-se a um problema de classificação, e não de regressão. No entanto, a mesma arquitetura

Figura 10 – Exemplo de funcionamento de uma CNN



Fonte: Lab (2017)

poderia ser mantida para ambos os tipos de problemas, sendo necessário apenas ajustar a última camada para se adequar à natureza do problema de regressão.

4.2 Arquiteturas de CNN

Os experimentos de David Hubel e Torsten Wiesel, realizados na década de 1960, inspiraram diretamente o desenvolvimento das CNN. Estudando o córtex visual de gatos, eles descobriram que, ao estimular áreas específicas do campo visual, determinadas células no córtex visual eram ativadas. Essas células respondiam de acordo com a forma e a orientação dos objetos no campo visual; por exemplo, bordas verticais ativavam certos neurônios, enquanto bordas horizontais ativavam outros. A organização em camadas dessas células sugeriu que os mamíferos utilizam essas camadas para construir representações visuais em diferentes níveis de abstração. Do ponto de vista do aprendizado de máquina, esse princípio corresponde à extração hierárquica de características nas CNN, onde camadas sucessivas capturam características visuais que vão de formas simples a padrões mais complexos (AGGARWAL, 2018).

Fukushima (1980) apresenta a rede Neocognitron, uma rede inspirada diretamente pelos trabalhos de David Hubel e Torsten Wiesel e que usava uma arquitetura de camadas hierárquicas para o reconhecimento de padrões visuais. Embora a Neocognitron incorporasse alguns elementos de design das redes convolucionais modernas, esta arquitetura não utilizava o algoritmo de *backpropagation*, empregando em vez disso um método de treinamento não supervisionado. Posteriormente, em 1988, Lang e Hinton

introduziram o uso de *backpropagation* em redes temporais conhecidas como *time-delay neural network (TDNN)*, que são redes convolucionais unidimensionais aplicadas a dados temporais. Em LeCun et al. (1989) é introduzida a LeNet-1, o que é considerada a primeira rede convolucional moderna ao aplicar o algoritmo de *backpropagation* em convoluções bidimensionais para imagens (GOODFELLOW; BENGIO; COURVILLE, 2016).

A LeNet-1 consiste em uma camada de entrada 16×16 seguida por duas camadas convolucionais ($H1$ e $H2$) com 12 filtros 5×5. Após $H2$, a rede possui duas camadas totalmente conectadas, sendo uma com 30 unidades e uma de saída com 10 unidades para classificação (LECUN et al., 1989). Já a LeNet-5, uma versão aprimorada introduzida em 1998, expandiu essa estrutura, aumentando a dimensão da imagem de entrada para 32×32 *pixels*, o número de filtros nas camadas convolucionais e o número de camadas totalmente conectadas. A LeNet-5 possui duas camadas convolucionais: a primeira camada aplica 6 filtros 5×5, enquanto a segunda camada convolucional usa 16 filtros 5×5. Cada camada convolucional é seguida por uma camada de *pooling*. Após as camadas convolucionais e de *pooling*, a rede inclui três camadas totalmente conectadas. A LeNet-5 foi implementada como o núcleo de sistemas de reconhecimento de escrita manual em dispositivos de entrada de caneta, proporcionando *feedback* em tempo real enquanto o usuário escrevia (LECUN et al., 1998).

Nos últimos anos, foram propostas diferentes arquiteturas de CNN. Em 2012, a AlexNet venceu o desafio *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* com uma taxa de erro top-5 de 15,4%, superando o segundo colocado, que obteve 26%. A métrica top-5 avalia se a resposta correta está entre as cinco principais predições do modelo. Inspirada na LeNet-5, a arquitetura da AlexNet possui cinco camadas convolucionais seguidas por três camadas totalmente conectadas. A AlexNet introduziu escolhas arquiteturais que se tornariam comuns em CNN futuras, como a função de ativação ReLU, *dropout* para descartar neurônios aleatoriamente e reduzir o *overfitting*, além do uso de unidades de processamento gráfico para o treinamento. A rede também aplicou técnicas de aumento de dados para ampliar a variabilidade do conjunto de treinamento (AGGARWAL, 2018; LI et al., 2022).

A GoogLeNet, proposta em 2014, introduziu um novo conceito referido como arquitetura Inception. O início da GoogLeNet segue a estrutura de uma rede convolucional tradicional, mas o diferencial está nos módulos Inception, aplicados nas camadas intermediárias. Os módulos Inception funcionam aplicando filtros de diferentes

tamanhos em paralelo sobre a mesma entrada. Cada filtro captura detalhes em uma escala diferente: os filtros menores focam em detalhes finos, enquanto os maiores capturam padrões mais amplos. Esses filtros processam a entrada simultaneamente, permitindo que a rede escolha automaticamente quais tamanhos são mais úteis para cada parte da entrada. Dessa forma, o módulo oferece flexibilidade para a rede "ver" diferentes escalas. A GoogLeNet venceu o ILSVRC de 2014 com um erro top-5 de 6,7% (AGGARWAL, 2018).

Ainda em 2014, foi proposta a VGG, uma arquitetura com design simples e sequencial. A VGG é composta de camadas convolucionais sequenciais com filtros 3×3. Após as camadas convolucionais são seguidas camadas de *pooling*. Os autores testaram variantes da arquitetura com diferentes profundidades, como VGG-11, VGG-16 e VGG-19, e observaram que o aumento da profundidade melhorava o desempenho da rede até certo ponto. A VGG introduziu escolhas arquiteturais que se tornariam comuns em redes futuras, como a redução do tamanho dos filtros e o aumento da profundidade da rede, permitindo uma representação mais detalhada das características com uma economia de parâmetros. Por exemplo, três camadas convolucionais com filtros 3×3 em sequência produzem um campo receptivo equivalente a de um filtro 7×7 aplicado diretamente na entrada, porém com uma quantidade menor de parâmetros (27 em vez de 49) e com mais variações não lineares. Essa abordagem permite que a VGG capture padrões complexos com menos parâmetros. O campo receptivo em CNN refere-se à região da imagem de entrada que influencia diretamente a ativação de um neurônio em uma camada específica. A VGG obteve segundo lugar com um erro top-5 de 7,3% no ILSVRC de 2014 (AGGARWAL, 2018; ARAUJO; NORRIS; SIM, 2019).

A ResNet (Residual Network), introduzida em He et al. (2015), permitiu a criação de arquiteturas com centenas de camadas, mitigando o problema da degradação, onde o desempenho da rede tende a cair conforme a profundidade aumenta. A ResNet aborda o problema da degradação utilizando conexões residuais, que permitem que a informação "salte" camadas. Uma conexão residual pode ser representada pela equação:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

onde $\mathbf{x}$ e $\mathbf{y}$ são, respectivamente, a entrada e a saída do bloco de camadas, e $\mathcal{F}(\mathbf{x}, \{W_i\})$ representa a função residual a ser aprendida, que pode consistir em múltiplas camadas convolucionais e operações de ativação. Os autores da ResNet propõem a hipótese de que é mais fácil otimizar uma função residual do que otimizar diretamente

a função de mapeamento original. Como exemplo, se a função identidade fosse a ideal, seria mais simples ajustar a função residual para zero do que fazer com que uma pilha de camadas não lineares aprenda diretamente uma função identidade. A ResNet venceu o ILSVRC de 2015 com um erro top-5 de 3,6% (AGGARWAL, 2018).

Em Tan e Le (2019), a EfficientNet foi introduzida como uma família de modelos de CNN desenvolvida para otimizar a precisão e a eficiência computacional. Após um estudo sistemático sobre escalonamento de modelos, os autores identificaram que balancear a profundidade, a largura e a resolução das redes neurais pode melhorar o desempenho. Com base nessa observação, eles propuseram um método de escalonamento composto e depois aplicaram o método em uma arquitetura base denominada Effitientet-B0 para gerar as variações da família. O escalonamento composto ajusta todas as dimensões da rede de maneira sistemática e uniforme, conforme a fórmula:

$$\begin{aligned} \text{profundidade:} \quad & d = \alpha^{\phi} \\ \text{largura:} \quad & w = \beta^{\phi} \\ \text{resolução:} \quad & r = \gamma^{\phi} \\ \text{s.a.} \quad & \alpha \times \beta^2 \times \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{10}$$

O coeficiente ϕ, definido pelo usuário, controla a quantidade de recursos disponíveis para o escalonamento da rede, enquanto α, β e γ determinam como esses recursos são distribuídos entre as três dimensões (profundidade, largura e resolução). Como as operações de convolução geralmente dominam o custo computacional em redes convolucionais, e o número de operações de *floating point operations per second (FLOPS)* é proporcional a d, w^2 e r^2, o escalonamento pela Eq. (10) aumenta o total de FLOPS aproximadamente por um fator de 2^{ϕ}, mantendo a condição $\alpha \times \beta^2 \times \gamma^2 \approx 2$. A EfficientNet-B7 atingiu uma erro top-5 de aproximadamente 3,03% no ImageNet (*dataset* utilizado no ILSVRC).

Neste trabalho, ao acolher o desafio de explorar o vasto território das CNN, a atenção foi inicialmente direcionada para arquiteturas já consolidadas, reconhecendo o valor de construir sobre os alicerces de conhecimento preexistentes. A ideia foi utilizar a base convolucional de arquiteturas existentes para compor a arquitetura final de cada vegetação. Uma base convolucional é o conjunto de camadas convolucionais de uma CNN

responsável por extrair características da entrada de dados. Assim, foram selecionadas arquiteturas para fins de exploração e comparação, sendo que a escolha dessas arquiteturas foi motivada pela necessidade de avaliar o impacto de diferentes níveis de complexidade (em termos de profundidade) no contexto deste trabalho. A profundidade da rede, examinada em estudos como He et al. (2015), influencia diretamente a capacidade da rede em aprender representações hierárquicas e complexas. Ao aumentar a profundidade, a rede é capaz de extrair características mais abstratas e discriminativas dos dados.

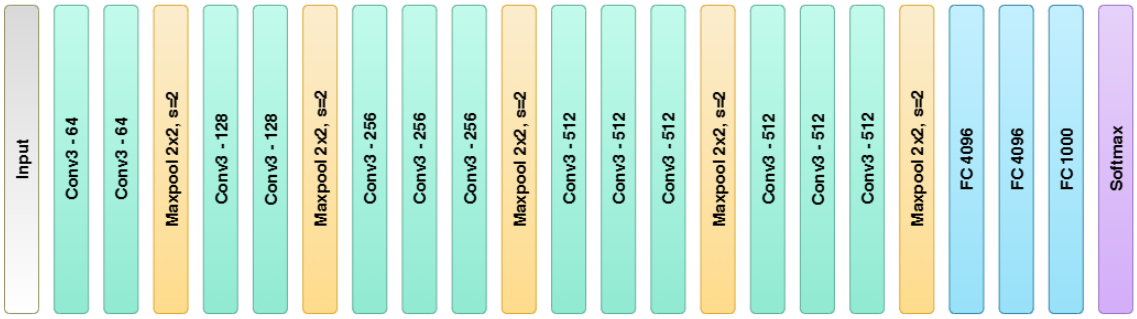
As arquiteturas VGG16 (SIMONYAN; ZISSERMAN, 2015), Xception (CHOLLET, 2017) e EfficientNetV2S (TAN; LE, 2020) foram selecionadas para exploração e comparação, com o objetivo de incluir uma boa variabilidade de complexidades. Essa escolha permite avaliar como diferentes profundidades e tamanhos de rede impactam a capacidade de generalização dos modelos no contexto deste trabalho. Além disso, estudos prévios evidenciam a aplicabilidade das bases convolucionais das redes VGG16, Xception e EfficientNetV2S em problemas de regressão (LATHUILIèRE et al., 2017; HU, 2020; LAAZOUFI; HASSOUNI; CHERIFI, 2023; WILIE; CAHYAWIJAYA; ADIPRAWITA, 2018; DREISBACH et al., 2023; GE; FROM; XIONG, 2024; SIACHOS et al., 2024), reforçando a relevância dessas arquiteturas para o presente trabalho.

A arquitetura VGG16 apresentada na Figura 11, consiste em uma sequência de 13 camadas convolucionais, seguidas por três camadas totalmente conectadas: as duas primeiras têm 4.096 neurônios cada, e a terceira realiza a classificação de 1.000 classes, contendo assim, 1.000 neurônios. A função *softmax* (SINGH et al., 2023) é utilizada na última camada para fornecer as probabilidades de classificação. Todas as camadas convolucionais usam filtros 3×3, permitindo capturar detalhes finos nas imagens (KHAN et al., 2020). A rede possui 16 camadas de pesos, totalizando cerca de 138 milhões de parâmetros. Camadas de *max-pooling* são aplicadas após as camadas convolucionais de índices 2, 4, 7, 10 e 13 para reduzir a dimensionalidade. A VGG16 obteve um erro top-5 de 7,3% no ILSVRC.

Detalhada pela Figura 12, a arquitetura Xception é inspirada nos módulos Inception, mas aprimorada com o uso de *depthwise separable convolutions*. A Xception possui 36 camadas convolucionais organizadas em 14 blocos, divididos em três fluxos: *Entry Flow*, *Middle Flow* e *Exit Flow*.

O *Entry Flow* processa as entradas iniciais e inclui camadas convolucionais seguidas por operações de *max pooling*, que reduzem progressivamente a resolução dos

Figura 11 – Arquitetura VGG16



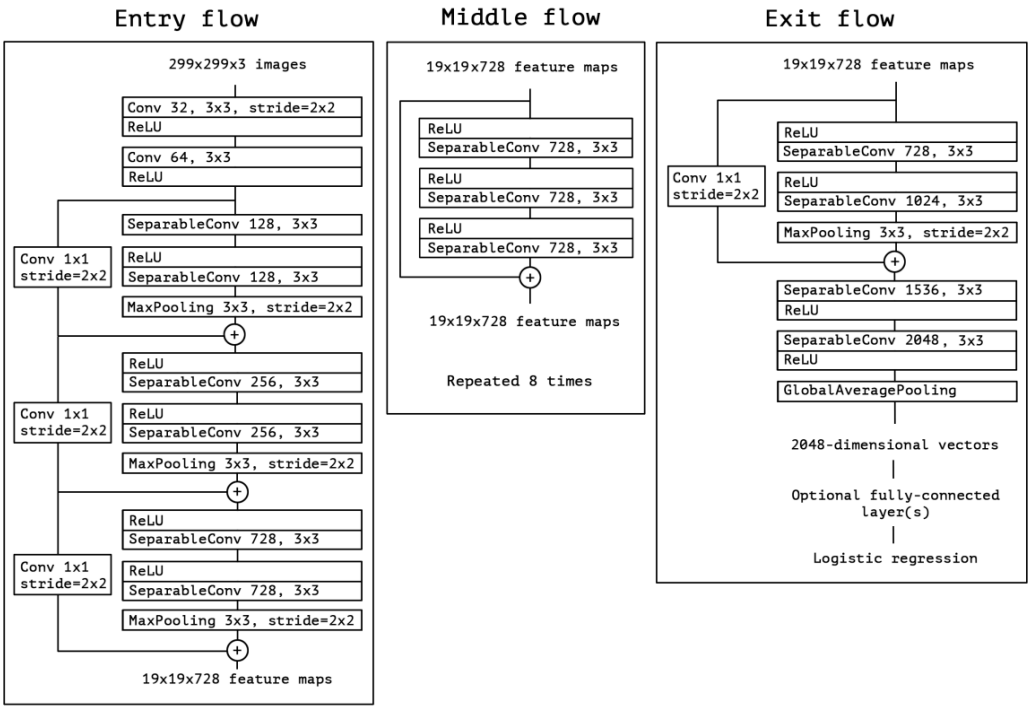
Fonte: Autor (2024)

mapas de características. No *Middle Flow*, a rede possui uma estrutura repetitiva, onde o mesmo módulo convolucional é repetido oito vezes. O *Exit Flow* finaliza a extração de características com uma série de convoluções mais profundas e termina com uma camada de *Global Average Pooling* para reduzir o mapa de características a um vetor de 2.048 dimensões, que é, então, passado para uma camada totalmente conectada opcional e uma camada de regressão logística para a classificação final. Todos os módulos da Xception empregam conexões residuais lineares, com exceção do primeiro módulo no *Entry Flow* e do último módulo no *Exit Flow*. Segundo o autor, a adição dessas conexões residuais levou a uma melhora na velocidade de convergência e no desempenho final da rede. Com aproximadamente 23 milhões de parâmetros, a Xception obteve um erro top-5 de 5,5% no conjunto de dados ImageNet (CHOLLET, 2016).

A família EfficientNetV2, introduzida em Tan e Le (2021), foi projetada para melhorar a eficiência em termos de parâmetros e velocidade de treinamento. Os autores estudaram os gargalos da família anterior (EfficientNet) — que incluíam lentidão no treinamento e limitações no escalonamento da arquitetura — para propor os modelos da EfficientNetV2. Para isso, eles ajustaram a busca de arquitetura com o objetivo de otimizar três aspectos: a precisão do modelo, a eficiência dos parâmetros e a velocidade de treinamento em aceleradores modernos. A EfficientNetV2S é o resultado dessa busca de arquitetura.

A arquitetura da EfficientNetV2S, ilustrada na Figura 13, é composta por uma camada convolucional inicial, seguida de 10 camadas Fused-MBConv e 30 camadas MBConv. Ao final, uma camada convolucional conecta-se a uma camada de pooling, que, por sua vez, se liga a uma camada totalmente conectada responsável pela saída. Essa estrutura resulta em um total de 42 camadas de convolução e cerca de 22 milhões de parâmetros. No conjunto de dados ImageNet, a EfficientNetV2S alcança um erro top-5

Figura 12 – Arquitetura Xception

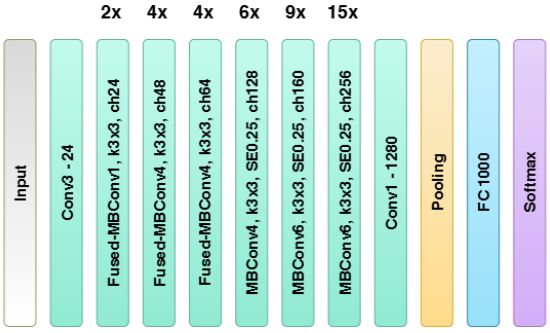


Fonte: Chollet (2017)

de aproximadamente 3,3%.

Finalmente, a escolha das arquiteturas VGG16, Xception e EfficientNetV2S também levou em conta a facilidade de utilização, pois todas possuem implementações disponíveis no Keras. Além disso, a viabilidade de execução em um ambiente com restrições computacionais, como o Google Colab, foi outro fator relevante na seleção dessas arquiteturas.

Figura 13 – Arquitetura EfficientNetV2S



Fonte: Autor (2024)

4.3 Keras Tuner

A otimização dos hiperparâmetros é uma das etapas do treinamento de redes neurais, sendo que esses valores influenciam diretamente o desempenho final do modelo. Ajustar os hiperparâmetros de forma arbitrária é um processo demorado e complexo, especialmente porque o espaço de estados dos hiperparâmetros é infinito e os recursos de tempo e de computação são limitados. Para simplificar e otimizar essa tarefa, surgiram bibliotecas especializadas em otimização de hiperparâmetros (CHOLLET, 2018), que usam heurísticas para guiar o processo de definição dos hiperparâmetros.

A ferramenta Keras Tuner faz uso de uma técnica conhecida como *define-by-run* configurar o espaço de valores dos hiperparâmetros (O'MALLEY et al., 2019). A técnica consiste na construção dinâmica do espaço de busca durante a definição do modelo, ao invés de ser fixado antecipadamente. Como exemplo, um modelo pode começar com um conjunto básico de hiperparâmetros, como a taxa de aprendizado ou o número de camadas, e, conforme o modelo se desenvolve, o espaço de busca pode se expandir ou se adaptar para incluir outras variáveis relevantes.

Para iniciar o processo de otimização de hiperparâmetros no Keras Tuner, inicialmente é definido um espaço de busca inicial, contendo todos os hiperparâmetros que devem ser otimizados (taxa de aprendizado, número de camadas, quantidade de unidades por camada, tipo de ativação, etc.) e que influenciam o desempenho do modelo. Cada hiperparâmetro possui um intervalo de valores ou um conjunto de opções a serem testadas.

Após a definição do espaço de busca, o próximo passo é escolher o otimizador de busca de hiperparâmetros. O otimizador é um algoritmo que determina como explorar o espaço de busca para encontrar as combinações de hiperparâmetros que produzem os melhores resultados para o modelo, usando uma heurística específica. O Keras Tuner possui diversos algoritmos de otimização.

Neste trabalho fez-se uso da otimização Bayesiana, que utiliza o modelo de probabilidade condicional para orientar a busca de hiperparâmetros, contruindo uma função de aquisição que estima a probabilidade de sucesso de cada configuração, considerando as tentativas anteriores. Essa função de aquisição ajuda a balancear entre a exploração (testar novas configurações) e a exploração direcionada (refinar as melhores configurações conhecidas). A cada nova tentativa, o modelo probabilístico é atualizado com os resultados obtidos, aprimorando suas previsões e focando em configurações que

apresentam maior potencial. A otimização Bayesiana é especialmente útil em espaços de busca amplos e em cenários com recursos limitados, pois reduz o número de tentativas necessárias para encontrar uma boa configuração (FRAZIER, 2018).

Após a execução da busca, o Keras Tuner gera uma saída que inclui as melhores configurações de hiperparâmetros encontradas, classificadas com base na métrica definida pelo usuário (como perda ou precisão). O Keras Tuner permite, ainda, recuperar o modelo otimizado pronto para uso, juntamente com uma lista das principais configurações testadas e suas respectivas métricas de desempenho, facilitando a análise comparativa e a escolha do modelo final.

5 PROJETO DO MÓDULO DE IA DO H-PASTURE 2.0

5.1 Arquitetura do sistema H-Pasture

O objetivo deste trabalho é construir um módulo inteligente, baseado em CNN, para ser integrado no sistema H-Pasture. O aplicativo H-Pasture (GASPARONI et al., 2021) tem como objetivo auxiliar a tomada de decisão dos produtores rurais com relação à movimentação de animais nos potreiros, evitando situações de sobre- ou subpastejo. A estimativa de disponibilidade de alimento é gerada a partir da altura média do pasto, conforme as correlações e recomendações definidas em (GENRO; SILVEIRA, 2018). O aplicativo foi lançado com a funcionalidade de inserção manual das medições de altura pelo usuário. Este adiciona a funcionalidade de inserir fotos tiradas do próprio celular do usuário da aplicação, usando o valor estimado da altura média da pastagem para cálculo da massa de forragem disponível na área.

Entre outras funcionalidades do H-Pasture estão:

- Cadastro de propriedades rurais, permitindo registrar informações como nome e localização.
- Cadastro de áreas de pastejo, permitindo especificar o tipo de vegetação (nativo, azevém ou capim-sudão), a área total em hectares e o sistema de manejo adotado (pastejo rotativo ou contínuo).
- Registro periódico de medições da altura das áreas de pastejo monitoradas.
- Geração de recomendações práticas, como a necessidade de remover os animais da área de pastejo.
- Cálculo da quantidade estimada de forragem disponível, utilizando modelos baseados nos estudos de Genro e Silveira (2018).

A arquitetura do aplicativo H-Pasture foi projetada seguindo o padrão de projeto *Model-View-ViewModel (MVVM)*, buscando uma separação entre a lógica de negócios, a apresentação dos dados e a interface do usuário. Nesse padrão, o *Model* gerencia os dados e a lógica de acesso, como a consulta de informações sobre as áreas de pastagem e a inserção de medições no banco de dados local. O *ViewModel* atua como uma camada intermediária, expondo os dados necessários para a interface de forma reativa, além de processar as interações dos usuários, como a seleção de áreas ou a geração de relatórios. Por fim, a *View* é responsável por apresentar as informações e capturar as interações dos

usuários, mantendo-se desacoplada da lógica do sistema. Essa abordagem permite maior testabilidade, modularidade e escalabilidade.

A Figura 14 apresenta a arquitetura do aplicativo H-Pasture por meio de um diagrama de componentes. A visão foi simplificada, com o objetivo de destacar a integração do módulo de IA (*AI Module*) ao restante da aplicação. Nesta representação, foram ocultados componentes relacionados às camadas *ViewModel* e *Model*, além do banco de dados local, buscando uma visão mais clara da interação entre os elementos responsáveis pelo fluxo de cadastro de medições.

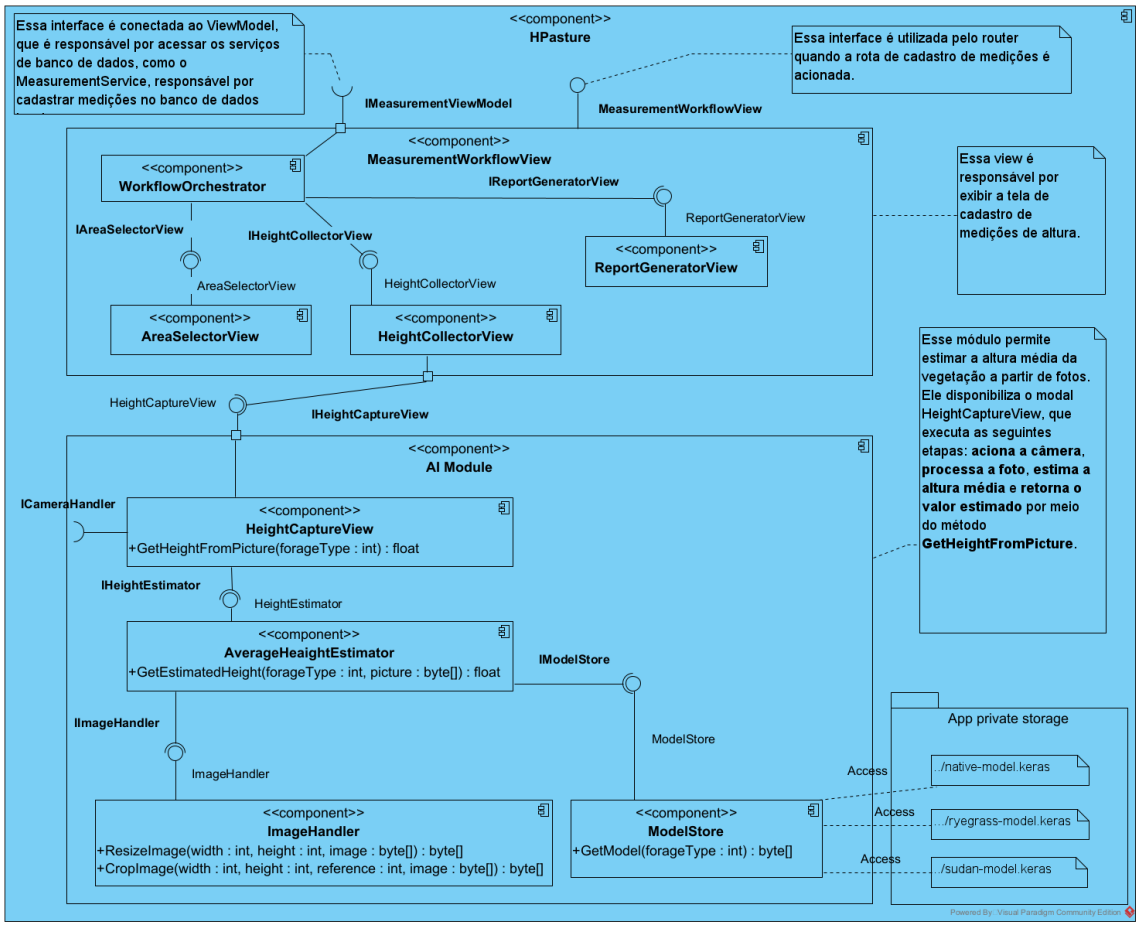
O *Workflow Orchestrator* é o componente central responsável por orquestrar o fluxo de cadastro de medições. Esse processo é dividido em três etapas: (1) a seleção da área de pastagem, realizada pelo componente *AreaSelectorView*; (2) a coleta das alturas das pastagens, por meio do componente *HeightCollectorView*; e (3) a geração de relatórios de recomendações, realizada pelo componente *ReportGeneratorView*, após o salvamento das medições.

O módulo de IA (*AI Module*) fornece a funcionalidade de estimativa de alturas médias a partir de imagens capturadas. Ele disponibiliza a *modal HeightCaptureView*, que é responsável por acionar a câmera para a captura de fotos e, em seguida, chamar o método *GetEstimatedHeight* do componente *AverageHeightEstimator*. Este método realiza o tratamento da imagem para ajustá-la ao formato esperado pelas redes neurais, como redimensionamento e recorte, utilizando o componente *ImageHandler*. Após o pré-processamento, o método *GetEstimatedHeight* busca o modelo correspondente ao tipo de pastagem (*forageType*) no componente *ModelStore*, que, por sua vez, acessa os modelos armazenados no *App private storage*. O *App private storage* é uma área de armazenamento dedicada e isolada para cada aplicativo no sistema operacional Android. Por fim, a estimativa é realizada e o valor é retornado em um tipo *float*.

O módulo de IA foi integrado ao componente *HeightCollectorView*, expandindo suas funcionalidades para incluir a inserção de alturas a partir de fotos. Inicialmente, o *HeightCollectorView* apenas permitia ao usuário digitar os valores das alturas diretamente. Com a integração do módulo de IA, esse componente agora oferece a inserção por meio de fotos. Para isso, o *HeightCollectorView* aciona a modal HeightCaptureView, que processa a foto capturada pelo usuário e retorna o valor estimado. Esse valor é então adicionado à lista de alturas coletadas para salvamento futuro.

A Figura 15 apresenta três telas sequenciais que exemplificam o processo de

Figura 14 – Arquitetura do H-Pasture



Fonte: Autor (2024)

cadastro de uma medição de altura no aplicativo H-Pasture. Para simplificar, a tela de seleção da área de pastagem foi omitida, mas ela é parte integral do fluxo de cadastro. A sequência das telas ilustra como o usuário pode registrar as medições de altura de diferentes pontos de uma área de pastagem monitorada.

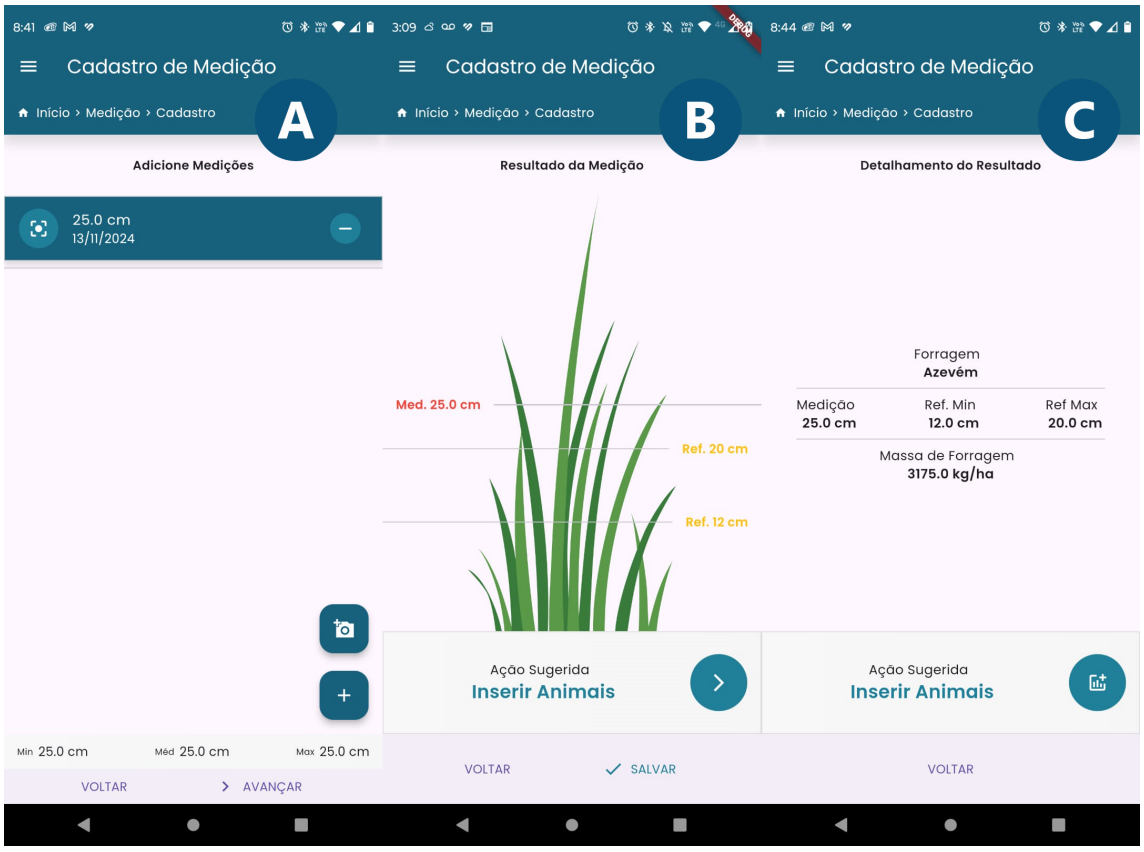
- Tela A: nesta primeira tela, o usuário adiciona as medições de altura em diferentes pontos da área de pastagem. A interface oferece duas opções de inserção, representadas por botões no canto inferior direito: o botão com o símbolo "+" permite a adição de medições manuais, enquanto o botão de câmera permite o registro de medições por meio de fotos, acionando o módulo de IA. Ao clicar no botão de câmera, o usuário é redirecionado para capturar uma foto. Com a imagem capturada, o módulo de IA processa a foto e retorna a altura média estimada do pasto na foto. No exemplo da figura, o módulo estimou uma altura de 25 cm.
- Tela B: nesta segunda tela, o aplicativo apresenta a ação sugerida com base na

medição realizada. A ação é gerada considerando a altura registrada e as diretrizes de manejo para o tipo de vegetação na área.

- Tela C: a terceira tela fornece um detalhamento dos resultados exibidos na Tela B, com informações adicionais sobre a medição. Além da altura registrada (25 cm), o aplicativo apresenta a massa de forragem disponível na área, calculada com base nas alturas medidas e no tipo da vegetação (neste caso, azevém). Também são mostradas as referências mínima e máxima de altura para auxiliar o usuário na avaliação do estado da pastagem.

Esse fluxo permite que o usuário registre e visualize de forma prática as medições de altura da pastagem, além de receber orientações para o manejo adequado da área monitorada.

Figura 15 – Telas do H-Pasture



Fonte: Autor (2024)

O restante deste capítulo apresenta o método e as decisões de projeto tomadas ao longo do trabalho. Particularmente, a Seção 5.2 apresenta os métodos e processos de aquisição dos dados de imagem da vegetação, a Seção 5.3 descreve as transformações dos dados de imagens para alimentação das CNN e a Seção 5.4 descreve as arquiteturas

implementadas.

5.2 Coleta de imagens

A coleta de dados ocorreu nas dependências da Embrapa Pecuária Sul, em 20 de outubro de 2023, 20 de fevereiro de 2024 e em 21 de outubro de 2024, com captura de 1120 fotos de pastagens com vegetação nativa, 1028 fotos com azevém (*Lolium multiflorum L.*) e 504 fotos de capim-sudão (*Sorghum sudanense L.*), totalizando 2652 fotos. As coletas ocorreram em diferentes áreas de pastejo, e os locais de captura não foram repetidos entre os dias, visando maximizar a variabilidade das condições de vegetação. As fotos visavam abranger uma variedade de alturas da vegetação em diferentes ambientes, no entanto, a sua amplitude foi limitada pela configuração da vegetação nos dias das coletas. Assim, as alturas da vegetação variaram entre 2 a 43 centímetros para a vegetação nativa, 3 a 39 centímetros para o azevém e 27 a 130 centímetros para o capim-sudão.

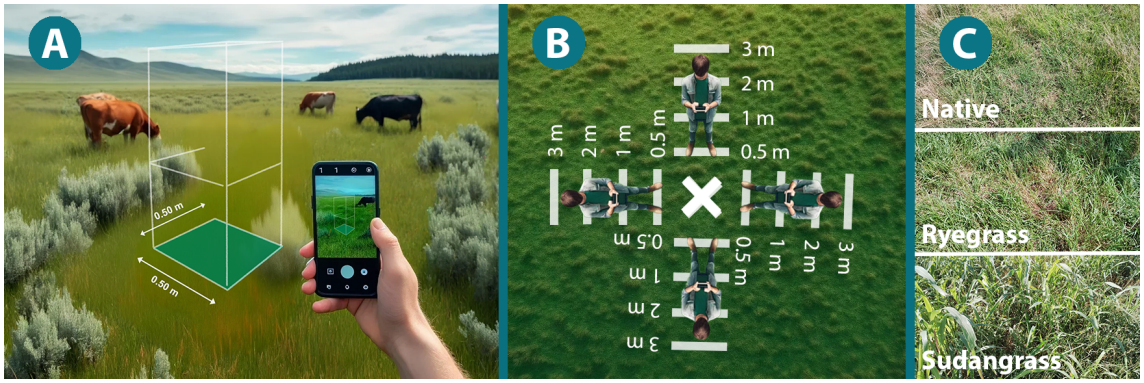
O procedimento de coleta das fotos envolveu etapas padronizadas. Para cada ponto de amostragem, um quadro de 0,25 m² foi posicionado na vegetação para servir como ponto de referência (Figura 16-A). Em seguida, foram realizadas cinco medições de altura a partir do interior desse quadro. Após anotar o valor médio das medições realizadas no aplicativo H-Pasture Collector, o quadro foi removido e as fotos foram tiradas, garantindo que a região central do quadro estivesse sempre centralizada nas imagens. A captura das fotos, retratada pela Figura 16-B, ocorreu em diferentes distâncias entre a câmera e a vegetação (0,5; 1; 2 e 3 metros) e em diferentes ângulos (0°, 90°, 180° e 270°) em relação ao ponto de referência, resultando em aproximadamente 16 fotos coletadas para cada ponto de amostragem. Esse procedimento foi repetido em todos os pontos de amostragem para garantir uma representação abrangente da vegetação em diferentes condições. Exemplos de fotos capturadas para cada tipo de vegetação são apresentados na Figura 16-C.

As imagens foram capturadas por meio do aplicativo de coleta de dados H-Pasture Collector, apresentado na Seção 2.3.

Os histogramas apresentados na Figura 17 exibem a distribuição das alturas medidas para os três tipos de vegetação: nativa (em azul), azevém (em verde) e sudão (em roxo). Assim, permitem a análise da variabilidade nos dados coletados.

Para a vegetação nativa, as alturas concentram-se principalmente entre 5 e 15 cm,

Figura 16 – Método para coleta de fotos



Fonte: Autor (2024)

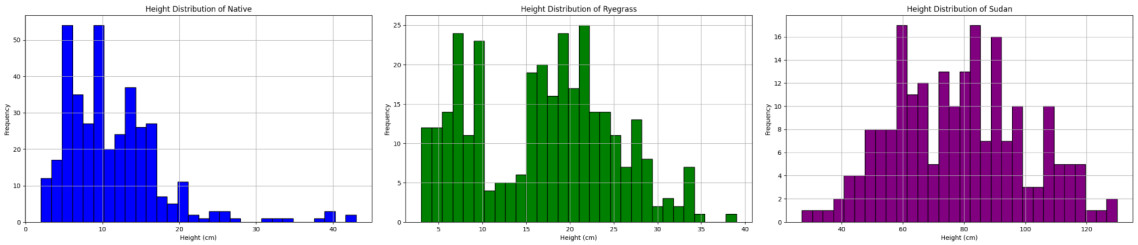
com um pico de frequência em torno de 5 a 10 cm. O azevém, por sua vez, apresenta uma distribuição de altura mais concentrada entre 5 a 10 cm e 15 a 25 cm, com alguns picos dentro destas faixas. Por fim, a vegetação de sudão apresenta uma maior variabilidade, com alturas que vão de 27 a 130 cm, e picos de frequência em torno de 60 e 80 cm. Esses histogramas foram gerados a partir de dados coletados em campo, refletindo a distribuição da vegetação nos dias das coletas.

Ainda com base nas observações dos histogramas da Figura 17, as faixas dominantes de altura favorecem o treinamento das redes neurais para detectar alturas nessas mesmas faixas. Assim, podemos notar que, para a vegetação nativa, as alturas entre 8 a 12 cm e entre 9 a 13 cm estão dentro da faixa dominante de alturas coletadas. Os intervalos 8 a 12 cm e entre 9 a 13 cm são relevantes, pois correspondem às alturas recomendadas para o manejo de pastagens nativas (campanha e depressão central, respectivamente) em sistemas de pastejo contínuo com taxa variável. Da mesma forma, para o azevém, o intervalo de altura entre 15 a 20 cm está dentro de uma faixa dominante nos dados coletados e também é uma faixa recomendada para o manejo dessa vegetação em pastejo contínuo com taxa variável (GENRO; SILVEIRA, 2018). A presença dessas faixas dominantes nos dados coletados permite que os modelos treinados estejam alinhados às práticas de manejo recomendadas.

Entretanto, o mesmo comportamento não foi observado no histograma do capim-sudão. Embora o intervalo de alturas entre 30 e 40 cm, recomendado para o manejo em pastejo contínuo com taxa variável (GENRO; SILVEIRA, 2018), esteja presente no histograma, ele não aparece com a mesma representatividade observada nas outras espécies. Esse resultado se deve ao fato de que, devido ao cronograma do trabalho e à sazonalidade do capim-sudão, a coleta pôde ser realizada em apenas um dia, limitando

a representatividade da faixa de alturas e a quantidade de dados disponíveis para essa espécie. Mesmo buscando incluir uma variedade de alturas, incluindo amostras de capim-sudão pastejado, os dados coletados não apresentaram alta variabilidade nas faixas de altura mais baixas para essa espécie.

Figura 17 – Distribuição das alturas coletadas em campo nativo, azevém e capim-sudão



Fonte: Autor (2024)

5.3 Pré-processamento de dados

A fase de pré-processamento de dados é uma das etapas no ciclo de desenvolvimento de sistemas de aprendizagem de máquina. O pré-processamento tem como fim a limpeza, organização, formatação e divisão dos dados coletados, tornando-os adequados para serem usados como entrada do software de treinamento e avaliação da rede neural. Para este trabalho, o pré-processamento de dados foi realizado utilizando *scripts* dedicados a esse fim. Assim sendo, foram desenvolvidos dois conjuntos de *scripts*, um em C# (MICROSOFT, 2023) para a primeira etapa e um *notebook* em Python (Python Software Foundation, 2024) para a segunda etapa do pré-processamento das fotos. As etapas de pré-processamento descritas abaixo foram realizadas individualmente para cada tipo de vegetação.

A primeira etapa foi dedicada à descarga das fotos coletadas do *back end* na nuvem. Após a conclusão deste processo, um arquivo JSON foi gerado, tendo como finalidade mapear cada foto descarregada com a altura média coletada no contexto da foto. O processo proporcionou um mecanismo para rotulagem dos dados, com vistas a um treinamento supervisionado, associando as imagens às suas respectivas medições.

A Figura 18 apresenta o JSON resultante desta primeira etapa e descreve os metadados associados às coletas realizadas. A chave Mean representa a altura média da vegetação na coleta, enquanto Id corresponde ao identificador único da coleta. A chave ForageType indica o tipo de vegetação coletada, como azevém, nativo ou sudão.

As chaves StandardDeviation e Distance estão presentes no formato do arquivo, mas não foram utilizadas neste trabalho. Cada coleta pode conter até 16 fotos, que são armazenadas em arquivos com o nome IDCOLETA_NSEQUENCIAL. Nesse formato, IDCOLETA refere-se ao identificador único da coleta (chave Id) e NSEQUENCIAL é um número sequencial de 0 a 15, que diferencia as imagens associadas à mesma coleta. Esse esquema permite mapear cada foto diretamente à altura média registrada para sua respectiva coleta. No JSON da Figura 18 estão sendo apresentadas apenas duas coletas para fins de exemplificação.

Figura 18 – Arquivo JSON gerado após descarga das fotos para azevém


```
[
  {
    "Id": 1,
    "ForageType": "azevem",
    "Mean": 22.4,
    "StandardDeviation": 0.0,
    "Distance": 0.0
  },
  {
    "Id": 2,
    "ForageType": "azevem",
    "Mean": 20.4,
    "StandardDeviation": 0.0,
    "Distance": 0.0
  }
]
```


Fonte: Autor (2024)

Detalhes mais específicos sobre essa etapa, incluindo a geração do arquivo JSON resultante, podem ser encontrados no Apêndice A. Nesse apêndice, está disponível o *script* em C# utilizado, o qual especifica o processo de descarga das fotos e detalha a geração do arquivo JSON.

Na segunda etapa, as fotos (originalmente capturadas na resolução 1280×720) extraídas na etapa anterior foram ajustadas para uma resolução de 720×720 *pixels*. Este ajuste incluiu um corte das imagens a partir do centro. Todas as fotografias preservaram seus três canais RGB de cores padrão.

Posteriormente, as fotos foram distribuídas em dois conjuntos distintos: um conjunto de treinamento com 80% das imagens e um conjunto de teste, que abrange as 20% imagens restantes. O conjunto de treinamento é o conjunto de exemplos que serão apresentados à rede neural, permitindo que ela identifique e aprenda padrões e características nas imagens. O conjunto de teste, por sua vez, é reservado para avaliar o desempenho do modelo e validar sua capacidade de fazer predições. A divisão em dados de teste e de treinamento foi realizada de forma aleatória.

Além da divisão dos dados, uma técnica conhecida como *data augmentation* (AGGARWAL, 2018) foi aplicada exclusivamente ao conjunto de treinamento. Essa técnica permite enriquecer o conjunto de treinamento, tornando-o mais diversificado. No contexto deste trabalho, *data augmentation* consistiu no espelhamento horizontal das imagens. Esse processo duplicou efetivamente o tamanho do conjunto de treinamento, garantindo que a rede neural fosse exposta a uma gama mais ampla de cenários. O trabalho seminal de Krizhevsky, Sutskever e Hinton (2012) com AlexNet destacou o papel do espelhamento horizontal como uma técnica simples e eficaz para melhorar o desempenho em tarefas de visão computacional. Para detalhes específicos sobre a implementação desta segunda etapa, incluindo o algoritmo empregado no processo de *data augmentation*, consulte o Apêndice B.

5.4 Arquiteturas de CNN

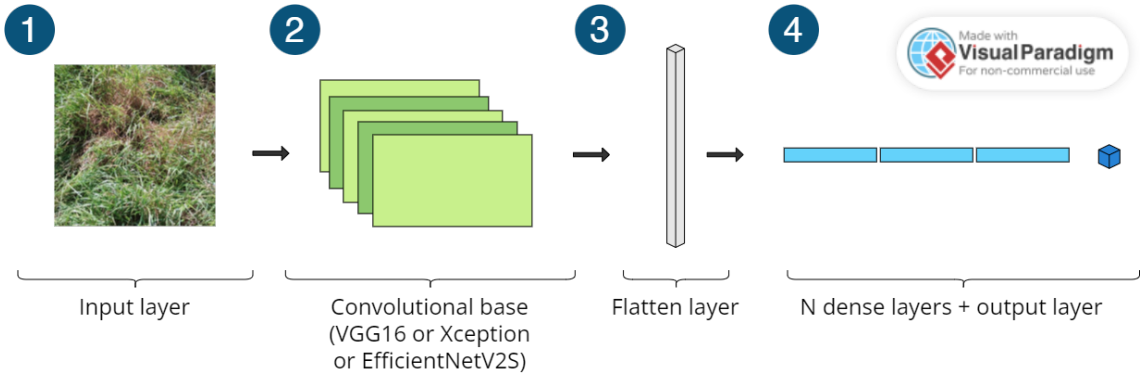
Lathuilière et al. (2020) apresenta o conceito de *vanilla deep regression*, que consiste na adaptação de CNN originalmente projetadas para tarefas de classificação para problemas de regressão. Essa abordagem substitui a função de ativação limitada da camada de saída (softmax, por exemplo) por uma camada com ativação linear, eliminando o limite de valores de saída , permitindo que a rede apresente saídas sem limites de valores.

Motivado pelo potencial apresentado em Lathuilière et al. (2020), este trabalho buscou o desenvolvimento de redes que seguem os princípios de *vanilla deep regression* – ou seja, usando funções de ativação de saída sem limitação, mesmo com arquiteturas tradicionalmente usadas para problemas de classificação.

A Figura 19 apresenta uma visão abstrata das arquiteturas desenvolvidas neste trabalho, destacando as principais etapas do fluxo de processamento das redes neurais. O processo inicia com uma imagem de entrada (1), que representa uma área de pastagem capturada em campo. Essa imagem é processada pela base convolucional (2), responsável por extrair características relevantes da vegetação, como texturas e padrões visuais associados à altura média da pastagem. As informações extraídas são então sintetizadas em uma ou mais camadas totalmente conectadas (3), que agregam e interpretam os dados das camadas anteriores. Por fim, a saída da rede é gerada por uma camada de regressão (4), configurada com ativação linear, que retorna um valor contínuo representando a altura média da vegetação.

Conforme discutido na Seção 4.2, ao escolher as CNN como método de solução

Figura 19 – Representação abstrata das arquiteturas



Fonte: Autor (2024)

do problema da estimativa da altura da vegetação por meio de imagens, a atenção foi inicialmente direcionada para arquiteturas já consolidadas, reconhecendo o valor de construir sobre os alicerces de conhecimento preexistentes. A ideia foi utilizar a base convolucional de arquiteturas existentes para compor a arquitetura final de cada vegetação. Assim, foram selecionadas as bases convolucionais das arquiteturas VGG16, Xception e EfficientNetV2S para exploração e comparação. A razão para essa escolha consistiu nas descrição de aplicações bem-sucedidas dessas arquiteturas, tanto para problemas de regressão como de classificação. Adicionalmente, queríamos comparar os resultados obtidos por meio de uma abordagem estatística, para mensurar qual a diferença existente no uso das diversas redes.

Com base nesse direcionamento, foram desenvolvidas 9 (nove) arquiteturas no total. Para a vegetação nativa, foram criadas três arquiteturas: uma utilizando a base convolucional VGG16, outra baseada na Xception e uma última com a EfficientNetV2S. O mesmo processo foi aplicado também às vegetações azevém e capim-sudão, totalizando três arquiteturas distintas para cada tipo de vegetação. Após a análise estatística realizada em cada grupo de arquiteturas (nativa, azevém e sudão), foram selecionadas três arquiteturas finais, uma para cada vegetação, que acabaram por integrar o aplicativo H-Pasture, descrito em Seção 5.1. Essa abordagem permitiu explorar o impacto de diferentes arquiteturas na tarefa proposta, assim como possibilitou o desenvolvimento de modelos que considerassem as particularidades de cada tipo de vegetação.

O desenvolvimento das arquiteturas ocorreu de forma iterativa e em etapas. Na primeira etapa, foram realizados testes empíricos em diferentes redes para compreender o comportamento das bases convolucionais no contexto do problema deste trabalho. Esses testes iniciais permitiram avaliar as capacidades de extração de características das bases

convolucionais (VGG16, Xception e EfficientNetV2S) selecionadas para exploração.

A resolução das imagens nos experimentos iniciais foi fixada em 360×360 *pixels*, uma redução pela metade da resolução original das imagens (720×720 *pixels*) geradas nas etapas de pré-processamento (Seção 5.3). Essa escolha foi motivada pelas limitações de memória identificadas durante os testes. No entanto, estudos como (HE et al., 2015), (CHOLLET, 2017) e (SIMONYAN; ZISSERMAN, 2015) empregaram resoluções entre 224×224 a 299×299, e alcançaram resultados competitivos em redes convolucionais, com erros *top-5* de 6,71% para a ResNet-50 (224×224), 4,56% para a Xception (299×299) e 7,3% para a VGG-16 (224×224) em tarefas de classificação no conjunto de dados ImageNet. Dessa forma, consideramos a resolução de 360×360 adequada para as experimentações, buscando equilibrar o uso de recursos computacionais e qualidade dos resultados.

O tamanho do lote (*batch size*), que define a quantidade de exemplos de treinamento processados simultaneamente pela rede antes da atualização dos pesos, foi fixado em 32. Essa decisão foi fundamentada no estudo (MASTERS; LUSCHI, 2018), que analisou o impacto do tamanho do lote em *datasets* como CIFAR-10, CIFAR-100 e ImageNet. Os resultados desse estudo mostram que o aumento progressivo do tamanho do lote reduz a faixa de taxas de aprendizado que proporcionam convergência estável e desempenho aceitável nos testes. Por outro lado, tamanhos de lote menores fornecem cálculos de gradientes mais atualizados, resultando em um treinamento mais estável. O estudo concluiu que o melhor desempenho foi consistentemente alcançado para tamanhos de lote entre 2 e 32.

Para a taxa de aprendizado (*learning rate*), inicialmente foi utilizado o valor padrão da biblioteca Keras 10^{-3}. No entanto, durante os experimentos, verificou-se que taxas menores (10^{-4} e 10^{-5}) promoviam maior estabilidade no treinamento, melhorando a convergência.

O número de camadas totalmente conectadas foi escolhido com base nos estudos originais das bases convolucionais utilizadas neste trabalho, que utilizam de 1 a 3 camadas. Embora a organização do número de neurônios tenha se inspirado nas recomendações de (BENGIO, 2012), que sugerem que usar o mesmo número de neurônios em todas as camadas totalmente conectadas pode apresentar resultados iguais ou melhores do que arquiteturas piramidais (crescentes ou decrescentes), essa abordagem não foi seguida estritamente, uma vez que os autores observam que a eficácia dessa configuração pode depender das características do conjunto de dados

utilizado. Durante os experimentos, foram realizadas variações no número de camadas totalmente conectadas e na quantidade de neurônios, com o objetivo de avaliar como essas configurações influenciam o desempenho do modelo.

Ainda nas experimentações, foram utilizadas duas funções de ativação na camada de saída: a linear e a softplus. A função linear é comumente utilizada em tarefas de regressão, pois permite que a saída do modelo seja qualquer valor contínuo, positivo ou negativo. Por outro lado, a função softplus, definida como $f(x) = \ln(1 + e^x)$, é uma alternativa suavizada à ReLU, garantindo que todas as saídas sejam estritamente positivas. O comportamento da função softplus é relevante para este trabalho, uma vez que os modelos desenvolvidos precisam estimar apenas valores positivos, correspondentes às alturas médias da vegetação. A inclusão da softplus no conjunto de funções avaliadas buscou explorar como uma ativação que reflete melhor a natureza dos dados influencia o desempenho das redes treinadas.

A primeira etapa do desenvolvimento das redes identificou combinações promissoras de camadas e hiperparâmetros, como o número de neurônios e funções de ativação. Com esses resultados, a segunda etapa foi dedicada ao refinamento das redes por meio de um espaço de busca de hiperparâmetros. Esse espaço incluiu hiperparâmetros como o número de camadas totalmente conectadas, o número de neurônios por camada, as funções de ativação e a taxa de aprendizado. Para realizar a otimização, foi utilizada a otimização bayesiana por meio da biblioteca Keras Tuner. Esse método permitiu explorar o espaço de busca de forma direcionada, priorizando configurações com maior probabilidade de melhorar o desempenho e reduzindo o número de experimentos necessários.

A Tabela 6 apresenta a definição do espaço de busca utilizado na segunda etapa, que é composto por quatro dimensões. Cada dimensão representa um hiperparâmetro ajustável que contribui para determinar a configuração final da rede. O hiperparâmetro *num_layers* define o número de camadas totalmente conectadas na rede, variando entre 1 e 3. O hiperparâmetro *units_*n especifica o número de neurônios na camada n, com valores entre 32 e 512, em passos de 128. Já para a função de ativação (*activation*), foram consideradas duas opções: *softplus*, que é adequada para estimar apenas valores positivos, e *linear*, que permite estimar valores contínuos sem restrições. Por último, a taxa de aprendizado (*lr*) foi configurada como uma escolha entre 10^{-4} e 10^{-5}.

No total, foram realizadas 9 execuções de otimização, considerando os 3 tipos de vegetação e as 3 arquiteturas por tipo. Em cada execução, foram testadas até 20

Tabela 6 – Espaço de busca de hiperparâmetros

Hiperparâmetro	**Tipo**	**Intervalo/Valores**
num_layers	Inteiro	1 a 3 (passo = 1)
*units_*n	Inteiro	32 a 512 (passo = 128)
activation	Escolha	{softplus, linear}
lr	Escolha	{0.0001, 0.00001}

Fonte: Autor (2024)

combinações de hiperparâmetros, um número definido com base nas limitações de custo computacional, já que cada tentativa envolvia até 30 épocas de treinamento. O número de épocas foi determinado durante as experimentações da etapa 1, com base na análise dos gráficos de convergência. Adicionalmente, o *early stopping* foi configurado com um *patience* de 5, interrompendo o treinamento caso o desempenho não apresentasse melhorias consecutivas por 5 épocas.

A Tabela 7 apresenta as arquiteturas finais selecionadas para cada tipo de vegetação – nativa, azevém e capim-sudão – após análises estatísticas aplicadas nos resultados da etapa de otimização (detalhes da análise estatística podem ser consultados em Seção 6.2). Cada arquitetura segue uma estrutura semelhante, começando com uma camada de entrada que processa imagens com resolução de $360 \times 360 \times 3$, representando imagens RGB. A base convolucional varia entre as vegetações: EfficientNetV2S foi definida para as vegetações nativa e capim-sudão, enquanto a Xception foi utilizada para o azevém. Essas bases convolucionais são responsáveis pela extração das características relevantes das imagens.

Após a extração de características, os dados passam por uma camada de *Flatten*, que transforma a saída da base convolucional em um vetor unidimensional. Em seguida, as redes incluem camadas totalmente conectadas. Para nativo e azevém, foram utilizadas três camadas totalmente conectadas, contendo 288, 32 e 416 neurônios, enquanto, para capim-sudão, as camadas possuem 32, 160 e 288 neurônios. Todas as camadas totalmente conectadas utilizam a função de ativação ReLU. As camadas de saída possuem um único neurônio com a função de ativação softplus (comum a todas as arquiteturas).

As arquiteturas possuem diferentes quantidades de parâmetros (treináveis), refletindo a complexidade de cada modelo. Para a vegetação nativa, a arquitetura contém 73.285.329 parâmetros (279,56 MB), para azevém, 105.765.289 parâmetros (403,46 MB), e para capim-sudão, 26.127.697 parâmetros (99,67 MB). O desempenho foi avaliado por meio do erro médio absoluto (MAE) e do coeficiente de correlação. Os resultados

Tabela 7 – Arquiteturas finais por tipo de vegetação

Arquiteturas								
Nativo			**Azevém**			**Capim-sudão**		
Camada	**Formato da saída**	**Ativação**	**Camada**	**Formato da saída**	**Ativação**	**Camada**	**Formato da saída**	**Ativação**
input	(360, 360, 3)	-	input	(360, 360, 3)	-	input	(360, 360, 3)	-
efficientnetv2s	(12, 12, 1280)	-	xception	(12, 12, 2048)	-	efficientnetv2s	(12, 12, 1280)	-
flatten	(184320)	-	flatten	(294912)	-	flatten	(184320)	-
dense_1	(288)	ReLU	dense_1	(288)	ReLU	dense_1	(32)	ReLU
dense_2	(32)	ReLU	dense_2	(32)	ReLU	dense_2	(160)	ReLU
dense_3	(416)	ReLU	dense_3	(416)	ReLU	dense_3	(288)	ReLU
output	(1)	softplus	output	(1)	softplus	output	(1)	softplus
Params: 73.285.329 (279,56 MB)			**Params: 105.765.289 (403,46 MB)**			**Params: 26.127.697 (99,67 MB)**		

Fonte: Autor (2024)

mostraram que as redes para nativo (MAE de 1,5236 cm e correlação de 0,8949) e azevém (MAE de 1,3791 cm e correlação de 0,96) apresentaram melhor desempenho, enquanto o capim-sudão apresentou maior variabilidade nas estimativas (MAE de 7,9795 cm e correlação de 0,81). Essas diferenças podem ser atribuídas às particularidades de cada vegetação, como suas características estruturais (por exemplo, altura e densidade), e à representatividade dos dados coletados.

As três redes finais foram integradas ao aplicativo H-Pasture, permitindo a estimativa automatizada da altura média das pastagens diretamente a partir de fotos capturadas pelos usuários.

Por fim, para o desenvolvimento das redes, foram criados diferentes *notebooks* Jupyter para as etapas de treinamento, otimização e validação estatística das redes. Esses notebooks foram executados no Google Colab. O notebook utilizado para os treinamentos e otimizações pode ser consultado no Apêndice C, enquanto o *notebook* dedicado à análise estatística está detalhado no Apêndice D. Adicionalmente, o Apêndice E apresenta o código-fonte do *data generator* desenvolvido, que foi utilizado para a alimentação das redes durante os experimentos.

6 RESULTADOS E DISCUSSÃO

6.1 Modelos, processo de treinamento e otimização

Conforme descrito na Seção 5.4, o desenvolvimento das redes ocorreu em duas etapas: uma etapa experimental e uma etapa de refinamento. Durante a etapa experimental, diferentes arquiteturas foram testadas para entender melhor o comportamento das redes frente aos dados do problema, buscando identificar arquiteturas promissoras. Na etapa de refinamento, foi definido um espaço de busca de hiperparâmetros e utilizada a otimização bayesiana para ajustar as arquiteturas de maneira direcionada.

Foram desenvolvidas 9 redes neurais, baseadas nas arquiteturas VGG16, Xception e EfficientNetV2S, aplicadas a cada tipo de vegetação: nativa, azevém e capim-sudão. A otimização bayesiana foi realizada individualmente para cada rede e foi executada por até 20 tentativas, onde cada tentativa correspondia a uma combinação de hiperparâmetros. Em cada tentativa, o treinamento de determinada rede era realizado por até 30 épocas, com possibilidade de interrupção antecipada por meio de *early stopping*, configurado com uma *patience* de 5 épocas. Isso significa que o treinamento era encerrado caso o desempenho não apresentasse melhorias consecutivas por 5 épocas.

Os resultados de cada tentativa eram utilizados para direcionar a busca por configurações mais promissoras de hiperparâmetros, focando em regiões do espaço de hiperparâmetros com maior potencial. Ao final das 20 tentativas, o Keras Tuner selecionava e retornava o melhor modelo já treinado, classificado com base na métrica MSE.

A Tabela 8 apresenta as arquiteturas resultantes da etapa de otimização para a vegetação nativa. Essas arquiteturas incluem as bases convolucionais VGG16, Xception e EfficientNetV2S, cada uma seguida por camadas totalmente conectadas. Na arquitetura baseada na VGG16, observa-se um total de 16.702.625 parâmetros, com camadas totalmente conectadas configuradas para 32 e 160 neurônios, utilizando a função de ativação softplus na camada de saída. Na arquitetura com base na Xception, há 30.378.281 parâmetros, sendo as camadas totalmente conectadas configuradas com 32, 416 e 288 neurônios, e a função de ativação utilizada na saída é a linear. Por fim, na arquitetura baseada na EfficientNetV2S, foram configurados 73.285.329 parâmetros, com as camadas totalmente conectadas ajustadas para 288, 32 e 416 neurônios, também

utilizando a função de ativação softplus na saída.

Tabela 8 – Arquiteturas otimizadas para vegetação nativa

Arquiteturas								
VGG16			**Xception**			**EfficientNetV2S**		
Camada	**Formato da saída**	**Ativação**	**Camada**	**Formato da saída**	**Ativação**	**Camada**	**Formato da saída**	**Ativação**
input	(360, 360, 3)	-	input	(360, 360, 3)	-	input	(360, 360, 3)	-
vgg16	(11, 11, 512)	-	xception	(12, 12, 2048)	-	efficientnetv2s	(12, 12, 1280)	-
flatten	(61952)	-	flatten_1	(294912)	-	flatten_1	(184320)	-
dense_1	(32)	ReLU	dense_1	(32)	ReLU	dense_1	(288)	ReLU
dense_2	(160)	ReLU	dense_2	(416)	ReLU	dense_2	(32)	ReLU
output	(1)	softplus	dense_3	(288)	ReLU	dense_3	(416)	ReLU
			output	(1)	linear	output	(1)	softplus
Params: 16.702.625 (63,72 MB)			**Params: 30.378.281 (115,88 MB)**			**Params: 73.285.329 (279,56 MB)**		

Fonte: Autor (2024)

A Tabela 9 detalha as arquiteturas resultantes para a vegetação azevém. Na arquitetura baseada na VGG16, foram configurados 16.769.857 parâmetros, com as camadas totalmente conectadas compostas por 32, 160 e 416 neurônios, sendo a saída configurada com a função linear. Na Xception, com 105.765.289 parâmetros, as camadas totalmente conectadas seguem o padrão de 288, 32 e 416 neurônios, e a função de ativação softplus foi aplicada na camada de saída. Por sua vez, a arquitetura baseada na EfficientNetV2S apresentou 49.749.201 parâmetros, com camadas totalmente conectadas configuradas com 160, 416 e 32 neurônios, utilizando também a função softplus na saída.

Tabela 9 – Arquiteturas otimizadas para azevém

Arquiteturas								
VGG16			**Xception**			**EfficientNetV2S**		
Camada	**Formato da saída**	**Ativação**	**Camada**	**Formato da saída**	**Ativação**	**Camada**	**Formato da saída**	**Ativação**
input	(360, 360, 3)	-	input	(360, 360, 3)	-	input	(360, 360, 3)	-
vgg16	(11, 11, 512)	-	xception	(12, 12, 2048)	-	efficientnetv2s	(12, 12, 1280)	-
flatten	(61952)	-	flatten_1	(294912)	-	flatten_1	(184320)	-
dense_1	(32)	ReLU	dense_1	(288)	ReLU	dense_1	(160)	ReLU
dense_2	(160)	ReLU	dense_2	(32)	ReLU	dense_2	(416)	ReLU
dense_3	(416)	ReLU	dense_3	(416)	ReLU	dense_3	(32)	ReLU
output	(1)	linear	output	(1)	softplus	output	(1)	softplus
Params 16.769.857 (63,97 MB)			**Params: 105.765.289 (403,46 MB)**			**Params 49.749.201 (189,78 MB)**		

Fonte: Autor (2024)

Para o capim-sudão, as arquiteturas resultantes da otimização são apresentadas na Tabela 9. Na arquitetura baseada na VGG16, foram configurados 32.603.553 parâmetros, com camadas totalmente conectadas compostas por 288 e 160, sendo a saída configurada com a função linear. Já na arquitetura baseada na Xception, com 68.073.385 parâmetros, as camadas totalmente conectadas seguem o padrão de 160, 416 e 32 neurônios, e a função de ativação softplus foi aplicada na camada de saída. Por fim, a arquitetura baseada na EfficientNetV2S apresentou 26.127.697 parâmetros, com camadas totalmente conectadas configuradas com 32, 160 e 288 neurônios, utilizando também a função softplus na saída.

Tabela 10 – Arquiteturas otimizadas para capim-sudão

Arquiteturas								
VGG16			**Xception**			**EfficientNetV2S**		
Camada	**Formato da saída**	**Ativação**	**Camada**	**Formato da saída**	**Ativação**	**Camada**	**Formato da saída**	**Ativação**
input	(360, 360, 3)	-	input	(360, 360, 3)	-	input	(360, 360, 3)	-
vgg16	(11, 11, 512)	-	xception	(12, 12, 2048)	-	efficientnetv2s	(12, 12, 1280)	-
flatten	(61952)	-	flatten_1	(294912)	-	flatten_1	(184320)	-
dense_1	(288)	ReLU	dense_1	(160)	ReLU	dense_1	(32)	ReLU
dense_2	(160)	ReLU	dense_2	(416)	ReLU	dense_2	(160)	ReLU
output	(1)	linear	dense_3	(32)	ReLU	dense_3	(288)	ReLU
			output	(1)	softplus	output	(1)	softplus
Params: 32.603.553 (124,37 MB)			**Params: 68.073.385 (259,68 MB)**			**Params: 26.127.697 (99,67 MB)**		

Fonte: Autor (2024)

Os resultados das otimizações indicaram a função de ativação softplus como particularmente adequada, sendo selecionada em 6 das 9 arquiteturas. Essa preferência aparenta refletir sua capacidade de gerar saídas positivas e contínuas, características alinhadas ao problema de estimar alturas médias da vegetação, que possuem valores sempre não negativos. Quanto à taxa de aprendizado, 10^{-4} foi o valor predominante na maioria das arquiteturas, enquanto 10^{-5} foi otimizado para todas as arquiteturas com a base convolucional VGG16, sugerindo a necessidade de ajustes mais graduais nos pesos dos modelos baseados nessa arquitetura.

Os tempos de otimização registrados na Tabela 11 reforçam a influência das características das arquiteturas e do tamanho dos conjuntos de dados no custo computacional. Esses tempos, obtidos em uma GPU NVIDIA A100-SXM4-40GB no ambiente do Google Colab, mostraram que a EfficientNetV2S apresentou os maiores tempos em todos os cenários, com destaque para o conjunto azevém, que levou aproximadamente 5 horas e 12 minutos para ser otimizado. Já a VGG16 e a Xception apresentaram tempos mais próximos entre si. O tempo reduzido observado para o capim-sudão pode ser explicado pelo menor tamanho deste conjunto de dados, que impactou diretamente no custo computacional, reduzindo a duração das otimizações.

Tabela 11 – Tempos de otimização por arquitetura

Tempo de otimização			
	VGG16	**Xception**	**EfficientNetV2S**
Nativo	03h 18m 46s	03h 12m 43s	04h 49m 09s
Azevém	03h 21m 08s	03h 02m 07s	05h 12m 40s
Capim-sudão	01h 37m 40s	01h 49m 33s	03h 56m 12s

Fonte: Autor (2024)

Por fim, a Figura 20 apresenta a evolução da função de perda (*loss function*) ao longo das tentativas de otimização realizadas pelo Keras Tuner para as diferentes

combinações de arquiteturas e tipos de vegetação. A função de perda mede a discrepância entre os valores estimados pela rede e os valores reais. O objetivo do treinamento é minimizar essa função, ajustando os pesos da rede para reduzir os erros de estimativa. Nesse trabalho, a função de perda utilizada foi o MSE, conforme definido na Seção 2.4.

Cada gráfico da Figura 20 reflete o processo de otimização para uma combinação específica de $tipo\_vegetacao \times arquitetura$, com o mecanismo de *early stopping* interrompendo alguns treinamentos que não apresentaram melhorias por mais de cinco épocas consecutivas.

O comportamento de convergência é observado em diferentes tentativas na Figura 20, sendo que as redes baseadas na Xception e na EfficientNetV2S apresentam maior estabilidade. A convergência refere-se a um comportamento onde a função de perda diminui progressivamente ao longo das épocas de treinamento até atingir uma estabilização. Esse comportamento indica que o modelo está aprendendo os padrões dos dados de maneira consistente, e que os ajustes nos pesos da rede neural estão resultando na redução dos erros de estimativa.

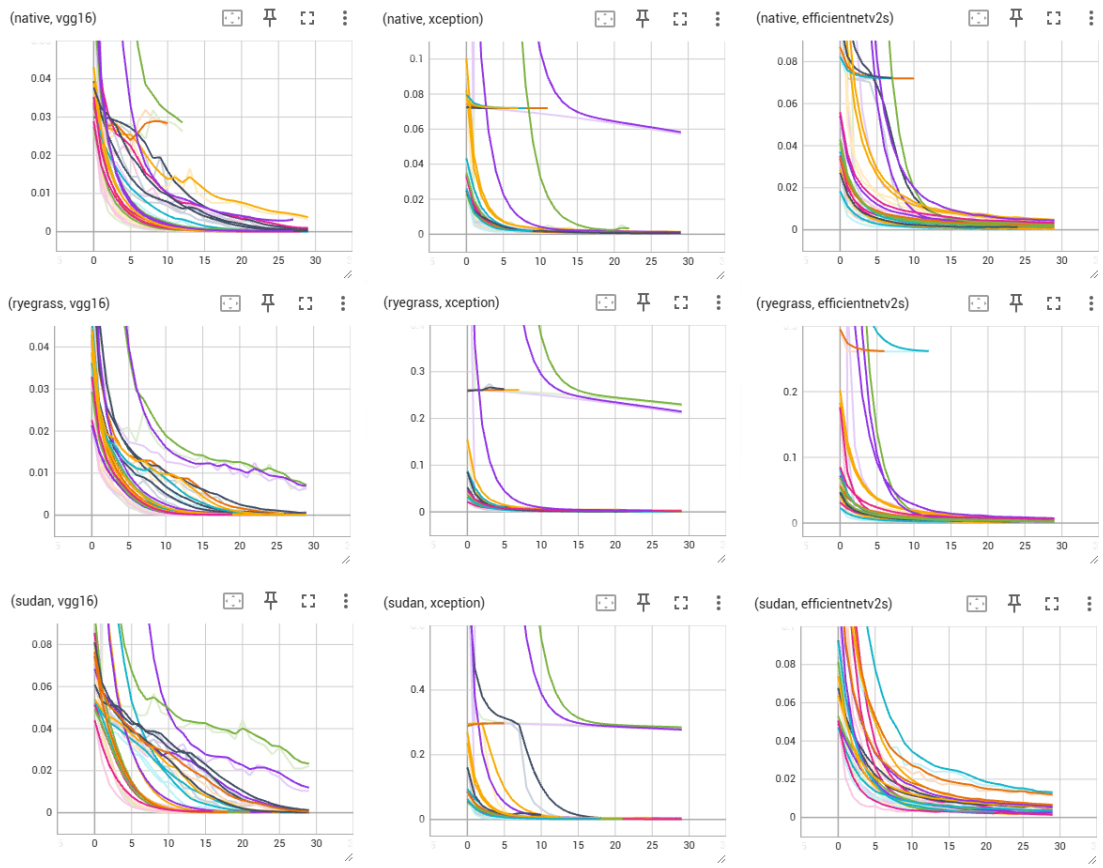
6.2 Avaliação dos modelos

Com o término do processo de otimização, a validação das redes foi realizada, iniciando pela análise de desempenho por meio das métricas MAE, coeficiente de correlação e a média dos resíduos para identificar tendências em subestimar ou superestimar as alturas médias. Em seguida, foi conduzida uma análise estatística para comparar o desempenho entre os diferentes modelos para tipos de vegetação.

A Figura 21 apresenta o desempenho das redes para a vegetação nativa. São apresentados os gráficos de correlação entre os valores estimados e os valores reais de altura média da vegetação, utilizando as arquiteturas base VGG16, Xception e EfficientNetV2S. Cada gráfico relaciona os valores reais de altura (no eixo x) com as estimativas realizadas pelas redes (no eixo y). A linha pontilhada indica a referência ideal onde os valores estimados seriam exatamente iguais aos valores reais ($y = x$).

No gráfico à esquerda da Figura 21, referente à arquitetura VGG16, observa-se um coeficiente de correlação de 0,8324, indicando um desempenho inferior em comparação com as outras arquiteturas. Há maior dispersão dos pontos em relação à linha de referência, especialmente para valores de altura acima de 15 cm, onde a rede tende a subestimar os valores reais. No entanto, esta rede apresenta uma média de resíduos de

Figura 20 – Gráfico da evolução da função de perda durante as tentativas de otimização

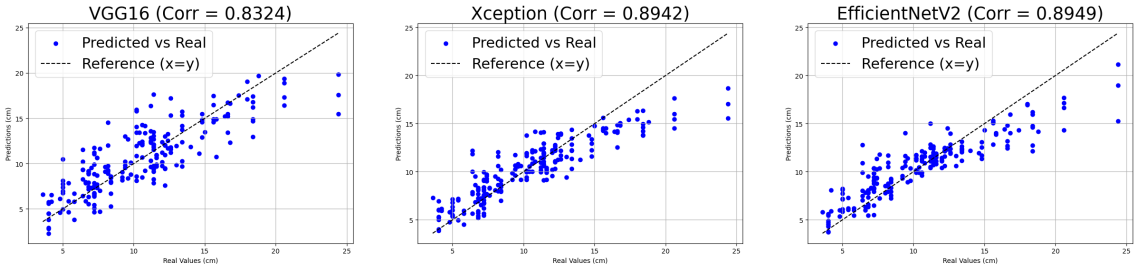


Fonte: Autor (2024)

-0,2090 cm, o que indica que, em geral, a tendência dessa arquitetura é superestimar as alturas médias.

No gráfico central, correspondente à arquitetura baseada na Xception, observa-se uma melhora no desempenho, com um coeficiente de correlação de 0,8942. Os pontos

Figura 21 – Gráfico de desempenho das redes para vegetação nativa



Fonte: Autor (2024)

estão mais alinhados à linha de referência, sugerindo estimativas mais consistentes com os valores reais. Apesar disso, também se nota uma maior dispersão para alturas acima de 15 cm, o que se reflete na média dos resíduos de 0,1198 cm, indicando uma tendência de subestimar as alturas médias.

O gráfico à direita apresenta o desempenho da arquitetura EfficientNetV2S, que obteve o maior coeficiente de correlação, com um valor de 0,8949. Embora o MAE de 1,5236 cm seja ligeiramente maior em relação ao da arquitetura Xception (1,5203 cm). Em todas as arquiteturas para a vegetação nativa, observa-se uma dispersão maior após os 15 cm. Este comportamento reflete a distribuição das alturas coletadas para essa vegetação, conforme discutido em Seção 5.2.

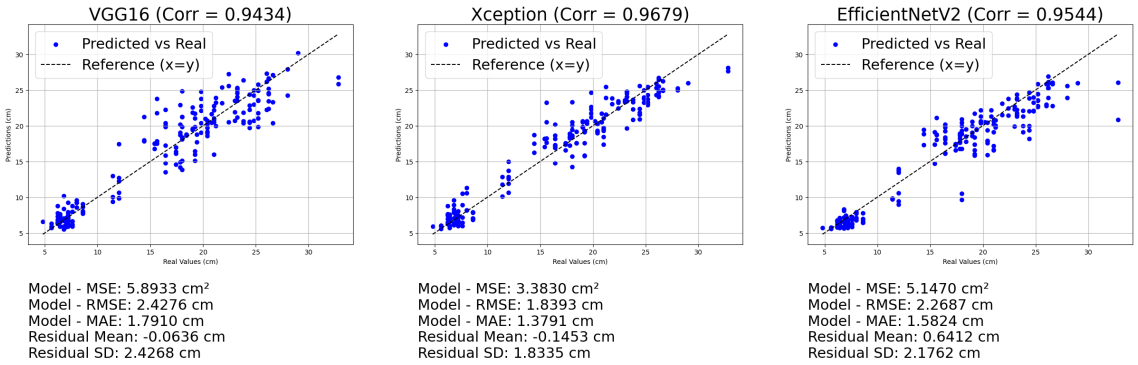
O desempenho para o azevém é apresentado na Figura 22. No gráfico à esquerda, referente à arquitetura VGG16, observa-se um coeficiente de correlação de 0,9434, indicando uma correlação forte entre as alturas estimadas e as reais, mas com maior dispersão em comparação às outras arquiteturas. O MAE foi de 1,7910 cm e a média dos resíduos foi de -0,0636 cm, sugerindo uma tendência em superestimar as alturas médias.

No gráfico central, referente à arquitetura baseada na Xception, o desempenho melhora, com coeficiente de correlação de 0,9679, o maior entre as três arquiteturas. Esta arquitetura também apresentou o desempenho em termos de MAE superior às outras duas arquiteturas, o MAE foi reduzido para 1,3791 cm. A média dos resíduos foi de -0,1453 cm, o que indica uma tendência de superestimar as alturas médias. A dispersão dos pontos acima de 15 cm é reduzida em comparação com a VGG16, sugerindo maior estabilidade para alturas maiores.

Por fim, o gráfico à direita apresenta o desempenho da EfficientNetV2S, com um coeficiente de correlação de 0,9544, ligeiramente inferior ao da Xception (com coeficiente de correlação de 0,9679), mas ainda superior ao da VGG16. O MAE foi de 1,5824 cm, intermediário entre as arquiteturas analisadas. A média dos resíduos foi de 0,6412 cm, indicando que esta arquitetura tende a subestimar as alturas médias de maneira mais acentuada em relação às outras redes.

A Figura 23 apresenta o desempenho para o capim-sudão. Para a arquitetura baseada na VGG16, observa-se o menor coeficiente de correlação entre os modelos, com valor de 0,5820, indicando uma relação moderada entre as estimativas e os valores reais. A dispersão dos pontos é considerável ao longo das estimativas, o que reflete nas métricas de desempenho, como o MAE de 11,2809 cm. A média dos resíduos foi de 1,4402 cm, indicando uma tendência de subestimação.

Figura 22 – Gráfico de desempenho das redes para azevém



Fonte: Autor (2024)

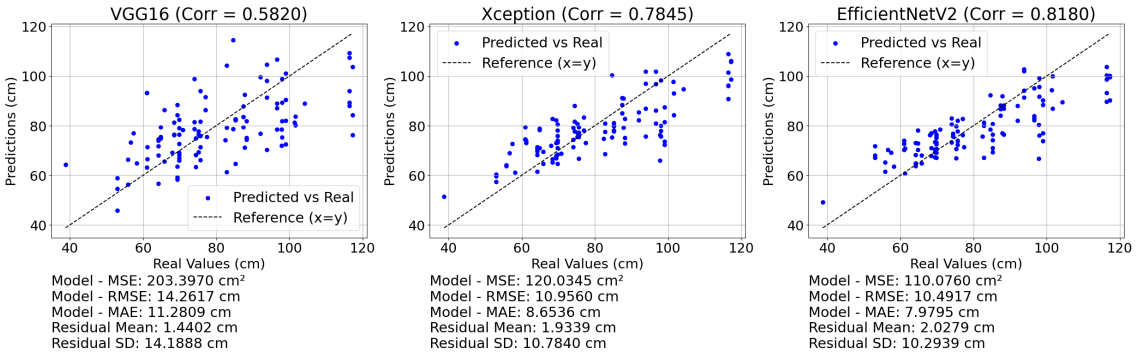
O gráfico central, referente à arquitetura baseada na Xception, demonstra um aumento no desempenho, com coeficiente de correlação de 0.7845. O MAE foi reduzido para 8,6530 cm, refletindo uma maior precisão em relação à VGG16. A média dos resíduos foi de 1,9339 cm, indicando uma tendência de subestimação. A dispersão dos pontos é menor, especialmente em alturas mais baixas, mas ainda visível para valores acima de 90 cm.

O desempenho da EfficientNetV2S para estimar a altura média obteve o maior coeficiente de correlação entre as arquiteturas para essa espécia, com valor de 0,8180. Para esta rede, o MAE foi de 7,9795 cm (o menor entre as três redes). A média dos resíduos foi de 2,0279 cm, indicando uma tendência de subestimação das alturas médias.

O desempenho das redes na estimativa da altura do capim-sudão foi inferior em comparação com os outros tipos de vegetação. Esse resultado pode ser explicado, em parte, por um conjunto de dados menor disponível para essa espécie, o que pode ter limitado a capacidade das redes em aprender padrões específicos do capim-sudão. Além disso, as características intrínsecas dessa espécie, como sua estrutura e alturas geralmente mais elevadas, podem ter tornado a tarefa de estimativa mais desafiadora.

A fim de avaliar se as diferenças observadas entre os modelos criados para cada espécie são estatisticamente significativas ou podem ser atribuídas ao acaso, foram realizados testes estatísticos. Inicialmente, foi aplicado o teste de normalidade aos módulos dos resíduos de cada rede. Essa análise revelou que, em geral, os dados não seguiam uma distribuição normal. Diante disso, para cada tipo de vegetação, foi utilizado o teste de Kruskal-Wallis para verificar a existência de diferenças significativas entre os grupos. Quando essas diferenças foram confirmadas, o teste de Nemenyi foi aplicado para identificar quais grupos apresentavam diferenças significativas. No contexto deste

Figura 23 – Gráfico de desempenho das redes para capim-sudão



Fonte: Autor (2024)

trabalho, cada grupo corresponde a uma arquitetura aplicada a um determinado tipo de vegetação.

O teste de Kruskal-Wallis identificou significância estatística entre os grupos da vegetação nativa, e a Tabela 12 apresenta os resultados do teste de Nemenyi. Os resultados indicam que há uma diferença estatisticamente significativa entre as arquiteturas VGG16 e EfficientNetV2S ($p = 0.0488$). Assim, a EfficientNetV2S apresentou desempenho estatisticamente superior à VGG16 em termos de MAE, com um valor de 1,5236 cm, enquanto a VGG16 obteve 1,8441 cm, indicando maior precisão na estimativa das alturas da vegetação nativa. Por outro lado, não foram encontradas diferenças significativas entre VGG16 e Xception ($p = 0.0985$), nem entre Xception e EfficientNetV2S ($p = 0.9534$). Com base nesses resultados, a EfficientNetV2S foi selecionada para integrar o aplicativo H-Pasture, pois apresentou diferenças significativas em relação à VGG16, o mesmo não ocorreu para a Xception.

Tabela 12 – Teste de Nemenyi para arquiteturas da vegetação nativa

Teste de Nemenyi (*p-value*)			
	VGG16	**Xception**	**EfficientNetV2S**
VGG16	1.0000	0.0985	0.0488*
Xception	0.0985	1.0000	0.9534
EfficientNetV2S	0.0488*	0.9534	1.0000

Fonte: Autor (2024)

Para o azevém, o teste de Kruskal-Wallis não identificou significância estatística entre os grupos, indicando que não há diferenças significativas em termos de MAE para as arquiteturas avaliadas. Diante disso, a escolha da arquitetura para integrar o aplicativo H-Pasture foi baseada diretamente no valor de MAE. Nesse contexto, a rede baseada na

Xception foi selecionada, pois apresentou o menor MAE entre as arquiteturas avaliadas, com um valor de 1,3791 cm.

Por fim, o teste de Kruskal-Wallis identificou significância estatística entre os grupos do capim-sudão. O teste de Nemenyi, apresentado na Tabela 13, revelou uma diferença significativa entre as arquiteturas VGG16 e EfficientNetV2S ($p = 0.0013$), evidenciando que a EfficientNetV2S possui um desempenho superior em relação à VGG16 em termos de MAE. Por outro lado, não foram observadas diferenças significativas entre VGG16 e Xception ($p = 0.0787$) ou entre Xception e EfficientNetV2S ($p = 0.3684$). Com base nessa análise, a EfficientNetV2S foi selecionada para integrar o aplicativo H-Pasture, considerando seu desempenho, em termos de MAE, superior estatisticamente em relação à VGG16.

Tabela 13 – Teste de Nemenyi para arquiteturas do capim-sudão

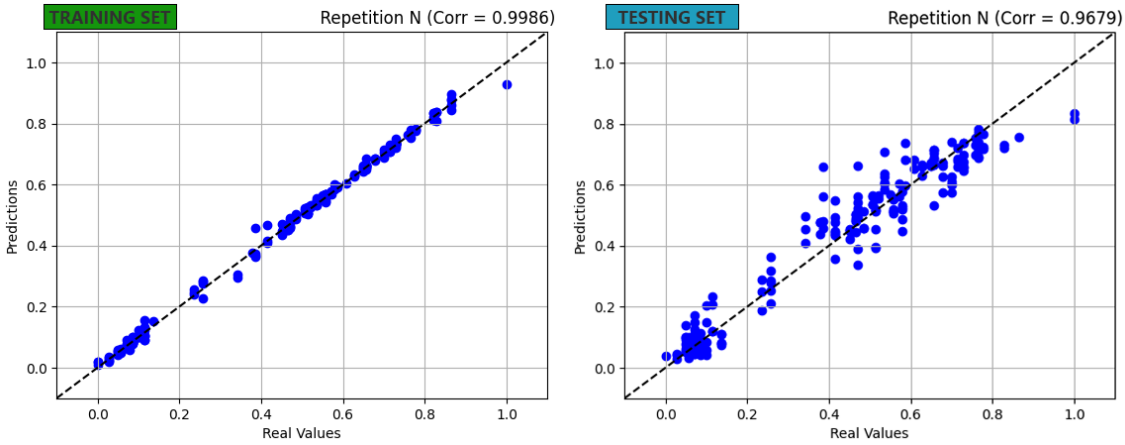
Teste de Nemenyi			
	VGG16	**Xception**	**EfficientNetV2S**
VGG16	1.0000	0.0787	0.0013**
Xception	0.0787	1.0000	0.3684
EfficientNetV2S	0.0013**	0.3684	1.0000

Fonte: Autor (2024)

Na Figura 24 é apresentado o gráfico de dispersão da rede baseada na arquitetura Xception para a vegetação azevém. Essa figura busca esclarecer um comportamento que reflete diretamente a metodologia utilizada na coleta das fotos. Para cada ponto de coleta, foram tiradas 16 fotos, todas associadas à mesma altura média da vegetação medida naquele local. Essa abordagem metodológica resulta em conjuntos de treinamento e teste que compartilham alturas médias idênticas para várias imagens, o que pode criar a impressão de similaridade entre o conjunto de treinamento e o conjunto de teste. Apesar disso, a separação entre os conjuntos foi realizada de forma criteriosa, assegurando que nenhuma foto utilizada no treinamento estivesse presente no conjunto de teste. Esse comportamento observado é uma característica da metodologia empregada e não compromete a validade dos resultados obtidos.

Como avaliação adicional, foram gerados os *heatmaps* das estimativas utilizando o método Grad-CAM (Gradient-weighted Class Activation Mapping), conforme descrito em Selvaraju et al. (2017). Esse método permite visualizar as regiões das imagens que mais contribuíram para as estimativas das redes, fornecendo uma análise qualitativa dos padrões aprendidos por cada arquitetura. A Figura 25 apresenta os *heatmaps* para a

Figura 24 – Gráficos de dispersão para os conjuntos de treinamento e teste da rede baseada na arquitetura Xception aplicada à vegetação azevém



Fonte: Autor (2024)

vegetação azevém, considerando as arquiteturas VGG16, Xception e EfficientNetV2S. O comportamento das três redes foi analisado para o mesmo conjunto de quatro imagens.

Os *heatmaps* da VGG16 mostram ativações dispersas em várias áreas da imagem, sem um padrão claro de foco, indicando menor seletividade em identificar as regiões mais relevantes para a estimativa da altura média. Por outro lado, as arquiteturas Xception e EfficientNetV2S apresentam *heatmaps* mais focados, com ativação concentrada em áreas mais específicas da imagem. Esse comportamento mais focado nas redes Xception e EfficientNetV2S pode ser reflexo da maior profundidade dessas arquiteturas, o que lhes confere maior capacidade de extrair detalhes finos e aprender padrões mais complexos nas imagens.

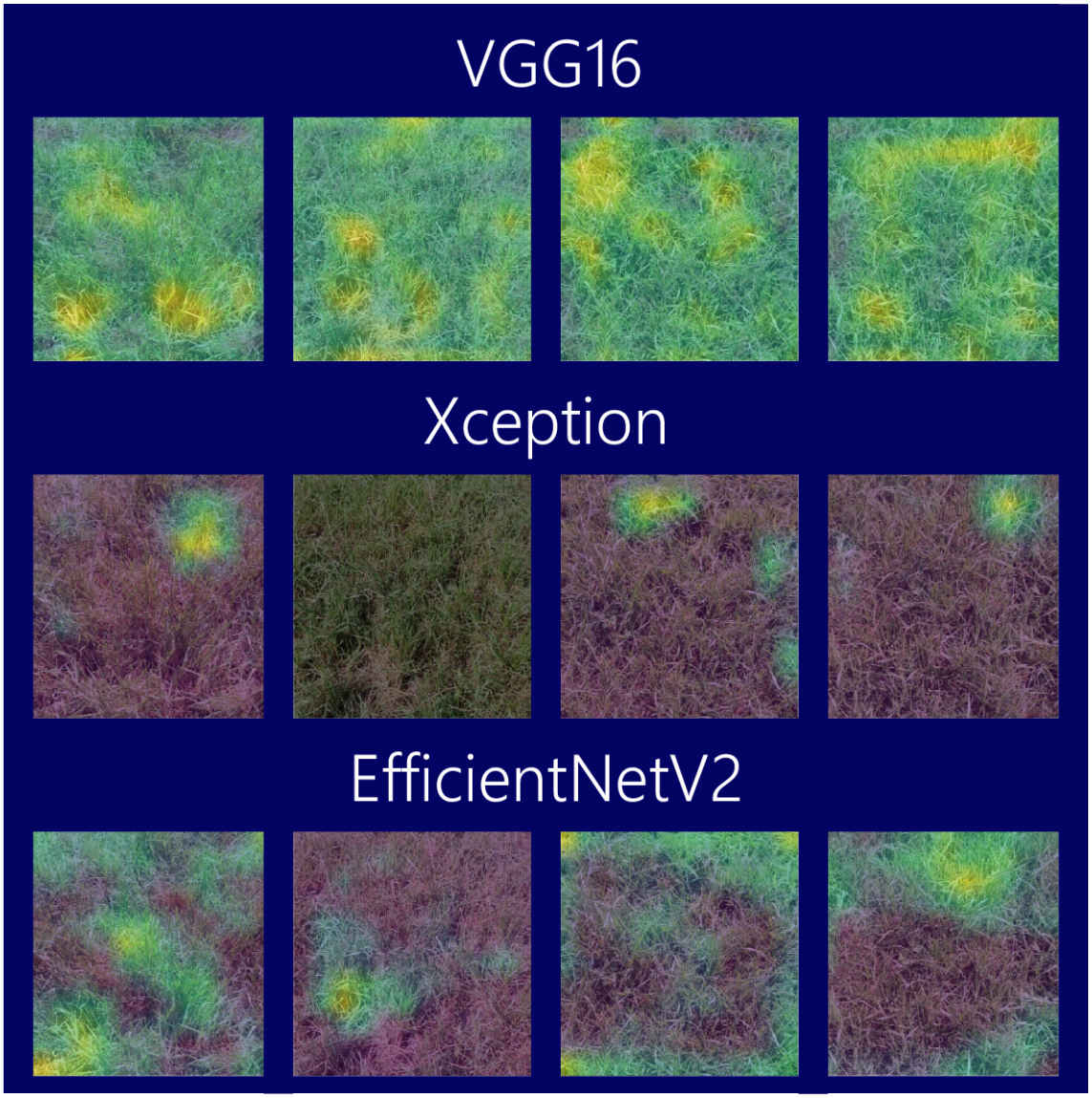
Para o azevém, os *heatmaps* da Xception, baseados na última camada convolucional da rede, foram os que mais se aproximaram da metodologia empregada neste trabalho para a medição das alturas durante a coleta dos dados, que consiste em realizar medições centrais na imagem, conforme descrito na Seção 5.2.

Por fim, a imagem da coluna 2, linha 2, na arquitetura Xception, reflete um comportamento particular desta rede para determinadas fotos. Nesse caso, a última camada não contribuiu positivamente para o valor estimado da altura, assim, as contribuições para a estimativa vieram das camadas anteriores da rede. Apesar desse comportamento particular, a rede apresentou um resultado interessante para essa imagem específica, estimando a altura em 19,17 cm, próximo do valor real de 20,4 cm.

No Apêndice F são apresentados os *heatmaps* gerados para os outros tipos de vegetação analisados neste trabalho, incluindo vegetação nativa e capim-sudão. Esses

heatmaps também utilizam o método Grad-CAM para destacar as regiões das imagens que mais contribuíram para as estimativas das redes.

Figura 25 – Comparação dos *heatmaps* das redes neurais para o azevém



Fonte: Autor (2024)

6.3 Avaliação do impacto da resolução nos resultados

Conforme apresentado na Seção 5.4, as redes desenvolvidas neste trabalho utilizaram uma resolução fixa de 360×360. Contudo, foram realizados testes com resoluções maiores (460×460, 560×560 e 660×660), para avaliar se o aumento no nível de detalhe das imagens poderia melhorar o desempenho das redes. A resolução original

de 720×720 não foi testada em virtude das limitações de memória no ambiente do Google Colab.

Devido a restrições de unidade de computação no Google Colab, as experimentações foram realizadas com base nas configurações das redes resultantes do processo de otimização para o azevém. As redes resultantes para o azevém apresentaram desempenho semelhante, não sendo identificada significância estatística entre as três arquiteturas. Assim, essa escolha buscou isolar o efeito da variável investigada (neste caso, a resolução).

Para as resoluções maiores, foram criadas novas redes a partir das arquiteturas otimizadas, ajustando a resolução da camada de entrada para refletir as diferentes dimensões das imagens. Por fim, os treinamentos também ocorreram em trinta épocas e a taxa de aprendizado foi extraída do processo de otimização.

Os resultados apresentados na Tabela 14 mostram que, em geral, houve uma tendência de piora no desempenho, tanto em termos de MAE quanto de coeficiente de correlação, com o aumento da resolução. Para a rede Xception, o modelo não convergiu para a resolução de 660×660, enquanto para a EfficientNetV2S, a falta de convergência ocorreu nas resoluções de 560×560 e 660×660. Nos três cenários de não convergência, os treinamentos foram interrompidos antes da 10ª época pelo mecanismo de *early stopping*, indicando que as redes não foram capazes de aprender adequadamente os padrões para essas resoluções dentro das configurações estabelecidas.

Tabela 14 – Desempenho das redes neurais por resolução para espécie azevém

Desempenho por resolução de imagem			
	VGG16	**Xception**	**EfficientNetV2S**
360x360	MAE = 1,7910, Corr = 0,9434	MAE = 1,3791, Corr = 0,9679	MAE = 1,5824, Corr = 0,9544
460x460	MAE = 1,8235, Corr = 0,9430	MAE = 1,6339, Corr = 0,9506	MAE = 1,6072, Corr = 0,9497
560x560	MAE = 1,8567, Corr = 0,9373	MAE = 1,7902, Corr = 0,9449	MAE = 11,5095, Corr = 0
660x660	MAE = 1,8220, Corr = 0,9378	MAE = 11,5095, Corr = 0	MAE = 11,5095, Corr = 0

Fonte: Autor (2024)

Além disso, foram realizados testes estatísticos dentro de cada rede para avaliar a significância das variações de desempenho, em termos de MAE, entre as diferentes resoluções (desconsiderando os casos em que não houve convergência). Os resultados não indicaram diferenças estatisticamente significativas, sugerindo que o impacto da resolução, dentro do intervalo analisado, não é suficiente para alterar significativamente o desempenho das redes.

Em resumo, os resultados indicam que, para o azevém, o aumento da resolução

das imagens não trouxe benefícios claros no desempenho das redes, reforçando a escolha de 360×360. Esses resultados corroboram, em parte, as observações feitas em Tan e Le (2020), onde os autores destacam que, embora resoluções mais altas possam melhorar a precisão das redes, os ganhos se tornam marginais em resoluções muito altas (como aquelas acima de 560×560 no referido estudo).

7 CONCLUSÃO

7.1 Síntese dos resultados

Este trabalho teve como ponto central a proposição e análise de desempenhos de redes neurais convolucionais como ferramenta para estimar a altura média da vegetação em pastagens nativas e cultivadas. O método fez uso de arquiteturas consolidadas de redes neurais convolucionais, nomeadamente VGG16, Xception e EfficientNetV2, com funções de ativação não limitadas na última camada. Ainda que as CNN sejam largamente utilizadas para reconhecimento de imagens, aplicações referentes à estimativa de altura de pastagens não foram encontradas na literatura. Note-se que as pastagens possuem uma característica dinâmica, com atividade de pastejo que altera a estrutura da vegetação – diferente de lavouras, que não são modificadas até a fase da colheita.

Os resultados indicam a viabilidade do uso da abordagem proposta, evidenciada por desempenhos promissores nas vegetações de campo nativo (MAE de 1,5236 cm, correlação de 0,8949) e azevém (MAE de 1,3791 cm, correlação de 0,96). Esses desempenhos destacam o potencial dos modelos para fornecer estimativas em diferentes cenários de pastagens. Entretanto, o capim-sudão apresentou maior variabilidade nas estimativas (MAE de 7,9795 cm, correlação de 0,81), refletindo que mais experimentos devem ser conduzidos para que a origem do problema possa ser delineada, seja por limitações no conjunto de dados disponíveis para essa espécie, já que as coletas ocorreram em um único dia devido à sazonalidade e restrições do cronograma, ou por outra característica da vegetação que faz com que as arquiteturas usadas não tenham sido capazes de realizar boas estimativas.

As redes neurais treinadas que obtiveram os melhores resultados neste trabalho foram incorporadas ao aplicativo H-Pasture. Agora, além da possibilidade de inserir medições de altura manualmente, é possível capturar imagens por meio da câmera do *smartphone*, que tem a altura inferida pela rede, a depender da pastagem fotografada. A partir da altura (ou das alturas) inseridas, o aplicativo faz a recomendação sobre inserção ou retirada de animais da área (com base na altura, como é a programação feita até agora).

A inserção da funcionalidade de medição da altura por fotos não apenas simplifica o processo operacional, mas também permite ao pecuarista uma tomada de decisão mais rápida. Diante da informação atualizada sobre a altura da vegetação, o pecuarista pode ajustar estratégias de pastagem, otimizar a distribuição de recursos, e até antecipar

a necessidade de suplementação alimentar para o gado. Essa agilidade na resposta a mudanças na vegetação não só economiza tempo, mas também pode ter impactos diretos na eficiência do manejo do rebanho e na produtividade da fazenda. Além disso, a integração entre a medição por fotos e a funcionalidade de recomendação de manejo do H-Pasture ajuda a manter as pastagens dentro da faixa recomendada de utilização. Ao garantir a altura ideal das pastagens, o manejo se torna mais eficiente, prevenindo problemas como sobrepastejo e subpastejo, otimizando os recursos disponíveis e promovendo um solo mais saudável. Pastagens bem manejadas também contribuem para a fixação de carbono no solo, reduzindo emissões de gases de efeito estufa e promovendo a sustentabilidade do sistema produtivo. Assim, a implementação dessa tecnologia representa não apenas uma inovação técnica, mas uma evolução prática na rotina do pecuarista. Ao possibilitar a obtenção de dados em tempo real sobre a altura da vegetação por meio de um *smartphone*, o H-Pasture simplifica os processos e transforma a maneira como o pecuarista interage e toma decisões no contexto dinâmico da pecuária.

Uma contribuição adicional desse trabalho será a disponibilização dos metadados e das imagens usadas nesse trabalho de forma aberta. Não foi possível encontrar bases similares – até pela falta de trabalhos relacionados diretamente com a medição de altura de pastagens. Essa disponibilização vai permitir outros trabalhos que podem ser diretamente comparados a este e também formar um conjunto de dados independentes para testes de estimativas de altura de pastagem por imagens em outros cenários/técnicas de aplicação.

7.2 Trabalhos futuros

Apesar do avanço conseguido em relação à estimativa de altura da vegetação, este trabalho também revela a necessidade de futuros aprimoramentos. A seguir, são citadas as principais questões que, no nosso entendimento, podem guiar a qualificação do aplicativo H-Pasture e seu uso por produtores:

1. Coletas adicionais de dados para todos os tipos de vegetação e em diferentes cenários. Os histogramas referentes às alturas coletadas mostram que existem lacunas de alturas nos pontos amostrais. A sazonalidade de pastagens cultivadas e a variabilidade de configurações do campo nativo requerem mais dados, exigência para que as CNN produzam bons resultados.

2. Investigar os problemas relacionados ao aprendizado das redes em relação ao capim-sudão, que consistentemente apresentou os piores resultados. Embora a representatividade limitada nas faixas de altura seja um fator relevante, ela não explica completamente a fragilidade dos resultados. Um ponto a ser destacado é que as alturas coletadas para essa vegetação abrangem uma amplitude expressivamente maior do que a recomendada para o manejo adequado, incluindo valores superiores a 100 cm. No manejo ideal, as alturas recomendadas variam conforme o sistema: em pastejo contínuo com taxa variável, situam-se entre 30 e 40 cm; em pastejo rotativo, entre 60 cm na entrada e 15 a 20 cm na saída. A grande amplitude dos dados coletados pode ter dificultado a capacidade de generalização dos modelos. Um aprofundamento nos resultados, aliado a coletas adicionais mais próximas das faixas recomendadas de manejo, pode ajudar a esclarecer essas questões e fortalecer a robustez dos modelos.
3. Estudos futuros podem explorar novas bases convolucionais, investigando como diferentes arquiteturas, com características distintas, impactam o desempenho na estimativa das alturas.
4. O aplicativo H-Pasture também pode ter suas funcionalidades aumentadas, especialmente em termos de recomendações. Hoje, a recomendação de entrada e saída é feita por altura, estimada a quantidade de massa seca disponível por hectare. A inserção de dados climáticos e de sazonalidade pode aumentar a precisão das estimativas em diferentes condições ambientais, possibilitando predizer a disponibilidade estimada futura e não somente presente. A estimativa do consumo também pode ser inserida no aplicativo, buscando a informação sobre a quantidade de animais e quilos de peso vivo por hectare para predição da velocidade do consumo de alimento na área.

REFERÊNCIAS

ABADI, M. et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Disponível em: https://www.tensorflow.org/.

ABDIKAN, S. et al. A comparative analysis of slr, mlr, ann, xgboost and cnn for crop height estimation of sunflower using sentinel-1 and sentinel-2. **Advances in Space Research**, v. 71, n. 7, p. 3045–3059, 2023. ISSN 0273-1177. Recent Advances in Space Research in Monitoring Sustainable Development Goals. Disponível em: https://www.sciencedirect.com/science/article/pii/S027311772201078X.

ABRAMIDES, P. et al. Estimativa da quantidade de forragem em pastagens de capins prostrados tropicais, através da medida da altura média da vegetação. v. 20, n. 1, p. 17–41, 1982.

AGGARWAL, C. C. **Neural Networks and Deep Learning: A Textbook**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2018. ISBN 3319944622.

ALLIES, A. et al. Evaluation of multiorbital sar and multisensor optical data for empirical estimation of rapeseed biophysical parameters. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, v. 14, p. 7268–7283, 2021.

ARAUJO, A.; NORRIS, W.; SIM, J. Computing receptive fields of convolutional neural networks. **Distill**, 2019. Https://distill.pub/2019/computing-receptive-fields.

ARKSEY, H.; O'MALLEY, L. Scoping studies: Towards a methodological framework. **Int. J. Social Research Methodology**, Routledge, v. 8, n. 1, p. 19–32, 2005.

AWWALU, J.; OGWUELEKA, F. On holdout and cross validation: A comparison between neural network and support vector machine. v. 6, p. 2394–9333, 04 2019.

BARBOSA, R. A. et al. Capim-tanzânia submetido a combinações entre intensidade e freqüência de pastejo. **Pesquisa Agropecuária Brasileira**, v. 42, n. 3, p. 329–340, 2007.

BATISTOTI, J. et al. Estimating pasture biomass and canopy height in brazilian savanna using uav photogrammetry. **Remote Sensing**, v. 11, p. 2447, 10 2019.

BEDI, R. K.; SINGH, J.; GUPTA, S. K. Current trends in cloud storage for resource constrained mobile devices. In: **2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)**. [S.l.: s.n.], 2016. p. 1144–1149.

BENCKE, G. A.; CHOMENKO, L.; SANT'ANNA, D. M. **O que é o Pampa?** [S.l.]: Fundação Zoobotânica do Rio Grande do Sul, 2016. 17-17 p.

BENGIO, Y. Practical recommendations for gradient-based training of deep architectures. **CoRR**, abs/1206.5533, 2012. Disponível em: http://arxiv.org/abs/1206.5533.

BISHOP, C. M. **Pattern Recognition and Machine Learning (Information Science and Statistics)**. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

BLUMAN, A. G. **Elementary Statistics: A Step by Step Approach**. 8th. ed. New York: McGraw-Hill, 2018. ISBN 978-1-260-02859-1.

CARNEVALLI, R. A. et al. Herbage production and grazing losses in panicum maximum cv. mombaça under four grazing managements. **Tropical Grasslands**, v. 40, p. 165–176, 2006.

CARVALHO, P. C. F. et al. Característica produtiva e estrutural de pastos mistos de aveia e azevém manejados em quatro alturas sob lotação continua. **Revista Brasileira de Zootecnia**, v. 39, n. 9, p. 1857–1865, 2010.

CHAUHAN, S.; SRIVASTAVA, H. S.; PATEL, P. Crop height estimation using risat-1 hybrid-polarized synthetic aperture radar data. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, v. 12, n. 8, p. 2928–2933, 2019.

CHERAïET, A. et al. An algorithm to automate the filtering and classifying of 2d lidar data for site-specific estimations of canopy height and width in vineyards. **Biosystems Engineering**, v. 200, p. 450–465, 2020. ISSN 1537-5110. Disponível em: https://www.sciencedirect.com/science/article/pii/S1537511020302944.

CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. **CoRR**, abs/1610.02357, 2016. Disponível em: http://arxiv.org/abs/1610.02357.

CHOLLET, F. **Xception: Deep Learning with Depthwise Separable Convolutions**. 2017.

CHOLLET, F. **Deep Learning with Python**. [S.l.]: Manning Publications, 2018.

CHOLLET, F. et al. **Keras**. 2015. <https://keras.io>.

DAVE, R. et al. Application of sentinel-1 sar-derived vegetation descriptors for soil moisture retrieval and plant height prediction during the wheat growth cycle. **International Journal of Remote Sensing**, v. 44, 01 2023.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: **2009 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2009. p. 248–255.

DEVORE, R.; HANIN, B.; PETROVA, G. Neural network approximation. **Acta Numerica**, v. 30, p. 327–444, 2021.

DREISBACH, C. et al. Using simulated data to predict birthweight from prenatal ultrasound images. In: **2023 IEEE Western New York Image and Signal Processing Workshop (WNYISPW)**. [S.l.: s.n.], 2023. p. 1–4.

DUTRA, A. C. et al. Brazilian savanna height estimation using uav photogrammetry. In: **2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS**. [S.l.: s.n.], 2021. p. 5945–5948.

FILHO, W. S. et al. Mitigation of enteric methane emissions through pasture management in integrated crop-livestock systems: trade-offs between animal performance and environmental impacts. **J. Clean. Prod.**, v. 213, p. 968–975, 2019.

FRANZLUEBBERS, A. J.; STUEDEMANN, J. A. Soil-profile organic carbon and total nitrogen during 12 years of pasture management in the southern piedmont usa. **Agric. Ecosyst. Environ.**, v. 129, p. 28–36, 2009.

FRAZIER, P. I. **A Tutorial on Bayesian Optimization**. 2018. Disponível em: https://arxiv.org/abs/1807.02811.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. **Biological Cybernetics**, Springer, v. 36, n. 4, p. 193–202, 1980.

GANO, B. et al. Using uav borne, multi-spectral imaging for the field phenotyping of shoot biomass, leaf area index and height of west african sorghum varieties under two contrasted water conditions. **Agronomy**, v. 11, n. 5, 2021. ISSN 2073-4395. Disponível em: https://www.mdpi.com/2073-4395/11/5/850.

GASPARONI, W. et al. H-pasture: uma aplicação móvel para estimativa da disponibilidade de forragem e ajuste de carga em pastagens sob lotação contínua ou rotativa. In: **Anais do XIII Congresso Brasileiro de Agroinformática**. Porto Alegre, RS, Brasil: SBC, 2021. p. 154–163. ISSN 2177-9724. Disponível em: https://sol.sbc.org.br/index.php/sbiagro/article/view/18386.

GE, Y.; FROM, P. J.; XIONG, Y. Multi-view gripper internal sensing for the regression of strawberry ripeness using a mini-convolutional neural network for robotic harvesting. **Computers and Electronics in Agriculture**, v. 216, p. 108474, 2024. ISSN 0168-1699. Disponível em: https://www.sciencedirect.com/science/article/pii/S0168169923008621.

GENRO, T. C. M.; SILVEIRA, M. C. T. da. **Uso da altura para ajuste de carga em pastagens**. [S.l.], 2018. 17 p.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016.

Google. **Flutter Framework**. Version x.y.z. Internet, 2023. Acessado em 26 de outubro de 2023. Disponível em: https://flutter.dev/.

Google Research. **Colaboratory: Jupyter Notebooks in the Cloud**. 2017. <https://colab.research.google.com>.

HASEGAWA, F. et al. Estimation of the plant height for energy crops using uav-sfm method. **Journal of the Japan Institute of Energy**, v. 101, n. 12, p. 265–269, 2022.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation 2Nd Ed.** Prentice-Hall Of India Pvt. Limited, 1999. ISBN 9788120323735. Disponível em: https://books.google.com.br/books?id=5YO3jwEACAAJ.

HE, K. et al. Deep residual learning for image recognition. **CoRR**, abs/1512.03385, 2015. Disponível em: http://arxiv.org/abs/1512.03385.

HERZOG, M. H.; FRANCIS, G.; CLARKE, A. **Understanding Statistics and Experimental Design: How to Not Lie with Statistics**. Cham: Springer, 2019. (Learning Materials in Biosciences). ISBN 978-3-030-03498-6. Disponível em: https://library.oapen.org/handle/20.500.12657/23029.

HODGSON, J. **Grazing management: science into practice**. New York: John Willey, Longman Scientific and Technical, 1990. 200 p.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **CoRR**, abs/1704.04861, 2017. Disponível em: http://arxiv.org/abs/1704.04861.

HU, L. Image noise estimation with using pre-trained conventional neural network. In: **2020 IEEE 3rd International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)**. [S.l.: s.n.], 2020. p. 492–495.

KAPLAN, G. et al. Normalizing the local incidence angle in sentinel-1 imagery to improve leaf area index, vegetation height, and crop coefficient estimations. **Land**, v. 10, p. 680, 06 2021.

KAPLAN, G. et al. Using sentinel-1 and sentinel-2 imagery for estimating cotton crop coefficient, height, and leaf area index. **Agricultural Water Management**, v. 276, p. 108056, 2023. ISSN 0378-3774. Disponível em: https://www.sciencedirect.com/science/article/pii/S0378377422006035.

KHAN, A. et al. A survey of the recent architectures of deep convolutional neural networks. **Artificial Intelligence Review**, Springer Science and Business Media LLC, v. 53, n. 8, p. 5455–5516, abr. 2020. ISSN 1573-7462. Disponível em: http://dx.doi.org/10.1007/s10462-020-09825-6.

KITCHENHAM, B. **Procedures for Performing Systematic Reviews**. Keele, 2004.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in Neural Information Processing Systems (NIPS)**. [S.l.: s.n.], 2012. p. 1097–1105.

LAAZOUFI, A.; HASSOUNI, M. E.; CHERIFI, H. Point cloud quality assessment using 1d vgg16 based transfer learning model. In: **2023 17th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)**. [S.l.: s.n.], 2023. p. 381–387.

LAB, K. **A Toy Convolutional Neural Network for Image Classification with Keras**. 2017. Publicado em 09/02/2017. Disponível em: https://www.kernix.com/article/a-toy-convolutional-neural-network-for-image-classification-with-keras/.

LATHUILIèRE, S. et al. Deep mixture of linear inverse regressions applied to head-pose estimation. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2017. p. 7149–7157.

LATHUILIèRE, S. et al. A comprehensive analysis of deep regression. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 42, n. 9, p. 2065–2081, 2020.

LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. **Neural Computation**, v. 1, n. 4, p. 541–551, 1989.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998.

LI, Z. et al. A survey of convolutional neural networks: Analysis, applications, and prospects. **IEEE Transactions on Neural Networks and Learning Systems**, v. 33, n. 12, p. 6999–7019, 2022.

LU, W.; OKAYAMA, T.; KOMATSUZAKI, M. Rice height monitoring between different estimation models using uav photogrammetry and multispectral technology. **Remote Sensing**, v. 14, n. 1, 2022. ISSN 2072-4292. Disponível em: https://www.mdpi.com/2072-4292/14/1/78.

MARASCHIN, G. E. Estratégias para valorizar sistemas pastoris sob a ótica de políticas de segurança alimentar, bem estar animal e social. In: ____. **REUNION DEL GRUPO TÉCNICO REGIONAL DEL CONO SUR EN MEJORIAMENTO Y UTILIZACIÓN DE LOS RECURSOS FORRAJEROS DEL ÁREA TROPICAL Y SUBTROPICAL**. Salto: GRUPO CAMPOS, 2004. p. 67–83.

MARTINS, A. P. et al. **Integração Soja-Bovinos de Corte no Sul do Brasil**. Porto Alegre: Grupo de Pesquisa em Sistema Integrado de Produção Agropecuária, UFRGS, 2015. 102 p.

MASTERS, D.; LUSCHI, C. Revisiting small batch training for deep neural networks. **CoRR**, abs/1804.07612, 2018. Disponível em: http://arxiv.org/abs/1804.07612.

Microsoft. **ASP.NET 6 Framework**. Version x.y.z. Internet, 2023. Acessado em 26 de outubro de 2023. Disponível em: https://dotnet.microsoft.com/apps/aspnet.

MICROSOFT. **C# Language**. 2023. Disponível em: https://docs.microsoft.com/en-us/dotnet/csharp/.

MONTGOMERY, D. C. **Design and Analysis of Experiments**. 10th. ed. [S.l.]: Wiley, 2019. ISBN 978-1119113478.

NABINGER, C. et al. **Anais do I Congresso sobre o Bioma Pampa - Reunindo Saberes**. [S.l.]: UFPel, 2020. 42-42 p.

NURHAYATI et al. A study of hold-out and k-fold cross validation for accuracy of groundwater modeling in tidal lowland reclamation using extreme learning machine. In: **2014 2nd International Conference on Technology, Informatics, Management, Engineering and Environment**. [S.l.: s.n.], 2014. p. 228–233.

NYIMBILI, P. et al. Structure from motion (sfm) - approaches and applications. In: . [S.l.: s.n.], 2016.

O'MALLEY, T. et al. **KerasTuner**. 2019. <https://github.com/keras-team/keras-tuner>.

PILLAR, V. **Campos Sulinos: Conservação e Uso Sustentável da Biodiversidade**. [S.l.: s.n.], 2009. ISBN 978-85-7738-117-3.

PostgreSQL Global Development Group. **PostgreSQL Database Management System**. Version 13.4. Internet, 2023. Acessado em 26 de outubro de 2023. Disponível em: https://www.postgresql.org/.

Python Software Foundation. **Python Language Reference, version 3.8**. 2024. <https://www.python.org>. Accessed: 2024-11-13.

RESCHKE, J. **The 'Basic' HTTP Authentication Scheme**. RFC Editor, 2015. RFC 7617. (Request for Comments, 7617). Disponível em: https://www.rfc-editor.org/info/rfc7617.

RIBEIRO, G. F.; JUNIOR, A. B. **Roteiro para o desenvolvimento e condução de uma revisão bibliográfica sistemática**. Atena Editora, 2022. Disponível em: https://doi.org/10.22533/at.ed.209221504.

ROMERO-PUIG, N.; MARINO, A.; LOPEZ-SANCHEZ, J. M. Application of the trace coherence to hh-vv polinsar tandem-x data for vegetation height estimation. **IEEE Transactions on Geoscience and Remote Sensing**, v. 60, p. 1–10, 2022.

ROSA, F. Q.; BREMM, C.; MACHADO, D. R. Efeito da oferta de forragem na estrutura do pasto. In: ____. **Nativão: 30 anos de Pesquisa em Campo Nativo**. Porto Alegre: UFRGS, 2017. p. 23–25.

SANDLER, M. et al. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. **CoRR**, abs/1801.04381, 2018. Disponível em: http://arxiv.org/abs/1801.04381.

SAVIAN, J. V. et al. Rotatinuous stocking: a grazing management innovation that has high potential to mitigate methane emissions by sheep. **J. Clean. Prod.**, v. 186, p. 602–608, 2018.

SELVARAJU, R. R. et al. Grad-cam: Visual explanations from deep networks via gradient-based localization. In: **Proceedings of the IEEE international conference on computer vision (ICCV)**. [S.l.: s.n.], 2017. p. 618–626.

SIACHOS, N. et al. Development and validation of a fully automated 2-dimensional imaging system generating body condition scores for dairy cows using machine learning. **Journal of Dairy Science**, v. 107, n. 4, p. 2499–2511, 2024. ISSN 0022-0302. Disponível em: https://www.sciencedirect.com/science/article/pii/S0022030223008044.

SILVEIRA, M. C. T. da et al. **Aprendizado de máquina com base na resposta espectral de imagens aéreas obtidas por VANTs e aplicado no manejo de pastagens**. Bagé, RS, 2022. 59 p. (Boletim de Pesquisa e Desenvolvimento, 51).

SIMONYAN, K.; ZISSERMAN, A. **Very Deep Convolutional Networks for Large-Scale Image Recognition**. 2015.

SINGH, B. et al. Analyzing the impact of activation functions on the performance of the data-driven gait model. **Results in Engineering**, v. 18, p. 101029, 2023. ISSN 2590-1230. Disponível em: https://www.sciencedirect.com/science/article/pii/S2590123023001561.

TAN, M.; LE, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. **CoRR**, abs/1905.11946, 2019. Disponível em: http://arxiv.org/abs/1905.11946.

TAN, M.; LE, Q. V. **EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks**. 2020. Disponível em: https://arxiv.org/abs/1905.11946.

TAN, M.; LE, Q. V. **EfficientNetV2: Smaller Models and Faster Training**. 2021.

TRINDADE, J. K. et al. Composição morfológica da forragem consumida por bovinos de corte durante o rebaixamento do capim-marandu submetido a estratégias de pastejo rotativo. **Pesquisa Agropecuária Brasileira**, v. 42, n. 6, p. 883–890, jun 2007.

TRINDADE, J. K. et al. Forage allowance as a target of grazing management: Implications on grazing time and forage searching. **Rangeland Ecology and Management**, v. 65, n. 4, p. 382–393, 2012. ISSN 1550-7424. Disponível em: https://www.sciencedirect.com/science/article/pii/S1550742412500631.

VILJANEN, N. et al. A novel machine learning method for estimating biomass of grass swards using a photogrammetric canopy height model, images and vegetation indices captured by a drone. **Agriculture**, v. 8, p. 70, 05 2018.

WAZLAWICK, R. S. **Metodologia de Pesquisa para Ciência da Computação**. [S.l.]: Elsevier Brasil, 2009. 122 p.

WILIE, B.; CAHYAWIJAYA, S.; ADIPRAWITA, W. Countnet: End to end deep learning for crowd counting. In: **2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)**. [S.l.: s.n.], 2018. p. 128–132.

YANG, H. et al. Spatio-temporal estimation of rice height using time series sentinel-1 images. **Remote Sensing**, v. 14, n. 3, 2022. ISSN 2072-4292. Disponível em: https://www.mdpi.com/2072-4292/14/3/546.

APÊNDICE A – *SCRIPT* - PRÉ-PROCESSAMENTO DE DADOS - ETAPA 1

Este apêndice contém o *script* C# utilizado durante a etapa 1 do pré-processamento de dados.

```
 1 using HPastureDataExtractor.Dtos;
 2 using Newtonsoft.Json;
 3 using System;
 4 using System.Collections.Generic;
 5 using System.IO;
 6 using System.Net.Http;
 7 using System.Net.Http.Headers;
 8 using System.Text;
 9 using System.Threading.Tasks;
10 using System.Drawing;
11
12 namespace HPastureDataExtractor
13 {
14     class Resume
15     {
16         public int Id { get; set; }
17         public string ForageType { get; set; }
18         public float Mean { get; set; }
19         public float StandardDeviation { get; set; }
20         public float Distance { get; set; }
21     }
22
23     public class DownloaderService
24     {
25         private HttpClient _http = new HttpClient();
26
27         private IList<Resume> _experiment = new List<Resume>();
28
29         public DownloaderService()
30         {
31             _http.DefaultRequestHeaders.Authorization = AuthHeader();
32         }
33
34         public async Task Download(Action<float> downloadCallback)
35         {
36             CheckCreateDownloadFolder();
37
38             _experiment = new List<Resume>();
39
40             var ids = await GetCollectionsIdAsync();
41
42             int idAtual = 1;
43             int total = ids.Count;
44
45             foreach (var id in ids)
46             {
47                 float progress = ((float)idAtual / total * 100);
48                 downloadCallback(progress);
49
```

```
50                  var collection = await GetCollectionAsync(id);
51
52                  if (collection.Measurements.Count == 0)
53                      continue;
54
55                  if(collection.ForageType != "azevem" &&
56                      collection.ForageType != "azevem florescência trevo"
                    &&
57                      collection.ForageType != "azevem e nativo")
58                      continue;
59
60                  HandleExportCollection(idAtual, collection);
61
62                  idAtual++;
63              }
64
65              SaveExperimentoJson();
66
67              downloadCallback(100);
68          }
69
70          private void CheckCreateDownloadFolder()
71          {
72              if(System.IO.Directory.Exists("Download"))
73                  System.IO.Directory.Delete("Download", true);
74
75              System.IO.Directory.CreateDirectory("Download");
76          }
77
78          private async Task<List<string>> GetCollectionsIdAsync()
79          {
80              var response = await _http.GetAsync("http://ec2-52-67-2-31.sa-
                  east-1.compute.amazonaws.com/collections/ids");
81
82              if(!response.IsSuccessStatusCode)
83                  throw new Exception("Http error. StatusCode = " + (int)
                      response.StatusCode + " " + response.StatusCode + " - "
                      + await response.Content.ReadAsStringAsync());
84
85              string json = await response.Content.ReadAsStringAsync();
86              List<string> ids = JsonConvert.DeserializeObject<List<string>>
                  (json) ?? new List<string>();
87
88              return ids;
89          }
90
91          private async Task<CollectionDto> GetCollectionAsync(string id)
92          {
93              var response = await _http.GetAsync("http://ec2-52-67-2-31.sa-
```

```
                east-1.compute.amazonaws.com/collections/" + id);
 94
 95             if (!response.IsSuccessStatusCode)
 96                 throw new Exception("Http error. StatusCode = " + (int)
                      response.StatusCode + " " + response.StatusCode + " - "
                      + await response.Content.ReadAsStringAsync());
 97
 98             string json = await response.Content.ReadAsStringAsync();
 99             var collection = JsonConvert.DeserializeObject<CollectionDto>
                  (json);
100
101             if (collection == null)
102                 throw new Exception("Collection deserialization is
                      null.");
103
104             return collection;
105         }
106
107         private void HandleExportCollection(int idAtual, CollectionDto
              collection)
108         {
109             string id = (idAtual + "").PadLeft(6, '0');
110
111             int index = 0;
112
113             _experiment.Add(new Resume()
114             {
115                 Id = idAtual,
116                 ForageType = collection.ForageType,
117                 Distance = 0,
118                 StandardDeviation = 0,
119                 Mean = collection.MeasurementsMean == 0 ? 0.1f :
                      collection.MeasurementsMean
120
121             });
122
123             foreach (var picture in collection.Pictures)
124             {
125                 ConvertBase64ToImage(picture.Picture, "./Download/" + id +
                      "_" + index + ".jpg");
126                 index++;
127             }
128         }
129
130         private void SaveExperimentoJson()
131         {
132             string json = JsonConvert.SerializeObject(_experiment,
                  Formatting.Indented);
133
```

```
134             // Caminho e nome do arquivo onde o JSON será salvo
135             string arquivoJson = "./Download/experiment.json";
136 
137             // Salva o JSON no arquivo
138             File.WriteAllText(arquivoJson, json);
139         }
140 
141         public static void ConvertBase64ToImage(string base64String,
              string outputPath)
142         {
143             // Remova o prefixo "" se estiver
                presente
144             if (base64String.Contains(","))
145             {
146                 base64String = base64String.Split(',')[1];
147             }
148 
149             // Converta a string Base64 em bytes
150             byte[] imageBytes = Convert.FromBase64String(base64String);
151 
152             // Crie um fluxo de memória para a imagem
153             using (MemoryStream ms = new MemoryStream(imageBytes))
154             {
155                 // Carregue a imagem a partir do fluxo de memória
156                 Image image = Image.FromStream(ms);
157 
158                 // Salve a imagem em um arquivo JPG
159                 image.Save(outputPath,
                      System.Drawing.Imaging.ImageFormat.Jpeg);
160             }
161         }
162 
163         private AuthenticationHeaderValue AuthHeader()
164         {
165             string username = "admin";
166             string password = "Admin2016*";
167 
168             // Codifique as credenciais para Base64
169             string base64Credentials = Convert.ToBase64String
                (Encoding.ASCII.GetBytes($"{username}:{password}"));
170 
171             return new AuthenticationHeaderValue("Basic",
                base64Credentials);
172         }
173     }
174 }
175 
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6
 7  namespace HPastureDataExtractor.Dtos
 8  {
 9      public class CollectionDto
10      {
11          public string Id { get; set; }
12          public string Creator { get; set; }
13          public string ForageType { get; set; }
14          public float MeasurementsMean { get; set; }
15          public int PicturesTotal { get; set; }
16          public List<MeasurementDto> Measurements { get; set; } = new
              List<MeasurementDto>();
17          public List<PictureDto> Pictures { get; set; } = new
              List<PictureDto>();
18      }
19
20      public class MeasurementDto
21      {
22          public string Id { get; set; }
23          public string Creation { get; set; }
24          public string Value { get; set; }
25      }
26
27      public class PictureDto
28      {
29          public string Id { get; set; }
30          public string Creation { get; set; }
31          public string Picture { get; set; }
32      }
33  }
34
```

APÊNDICE B – *NOTEBOOK* - PRÉ-PROCESSAMENTO DE DADOS - ETAPA 2

Este apêndice contém o *notebook* Jupyter utilizado durante a etapa 2 do pré-processamento de dados.

notebook-data-preprocessing-step2

November 21, 2024

Script para aplicar data augmentation no dataset de imagens em .exp-data. Após o processo os dados de treino estarão em .exp-data/training/ e os para teste em ./exp-data/test/.

[ ]:
```
import pandas as pd
import numpy as np
import os
from random import random, shuffle
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import label_binarize
from sklearn.metrics import confusion_matrix, classification_report,␣
↪ConfusionMatrixDisplay
import cv2
import matplotlib.image as mpimg
import json
import shutil
```

[ ]:
```
specie = 'sudao'
specie_out = 'sudao'
```

[ ]:
```
def create_dataset(folder):
    imgs = []
    heights = []

    resumes = load_resumes(folder)

    ids = []

    files = sorted(os.listdir(folder))

    print(files)

    for file in files:
        image_path = os.path.join(folder, file)
        file_name, file_extension = os.path.splitext(file)
```

```
        if file_extension != '.jpg':
            continue

        id = get_file_id(file_name)

        ids.append(id)

        resume = find_resume(int(id), resumes)

        heights.append(float(resume['Mean']))

        #image = cv2.imread(image_path, cv2.COLOR_RGBA2BGRA)#.astype(np.float32)
        image = tf.keras.utils.load_img(image_path)
        input_arr = tf.keras.utils.img_to_array(image)
        npimage = np.array(input_arr)
        #npimage /= 255
        imgs.append(npimage)

    scaler = MinMaxScaler()
    heights = np.array(heights).reshape(-1, 1)
    heights = scaler.fit_transform(heights)

    #scaler = MinMaxScaler()
    #scaler.fit(heights)
    #heights = scaler.transform(heights)
    print(ids)

    return ids, imgs, heights

def load_resumes(folder):
    experiment = ''
    with open(os.path.join(folder, 'experiment.json'), 'r') as f:
        experiment = f.read()

    return json.loads(experiment)

def find_resume(id, resumes):
    return list(filter(lambda resume: resume['Id'] == id, resumes))[0]

def get_file_id(file_name):
    return file_name.split('_')[0]
```

[ ]:
```
ids, imgs, heights = create_dataset('.exp_data/' + specie + '/')
```

[ ]:
```
np.random.seed(1)
image_size = 720 # width and length
resize_size = 720
```

```
image_pixels = image_size * image_size
```

[ ]:
```
def crop_to_square(image):
    height, width, _ = image.shape

    # Determinar o tamanho do corte
    size = min(height, width)

    # Calcular as coordenadas do corte
    start_x = (width - size) // 2

    # Realizar o corte utilizando o TensorFlow
    cropped_image = tf.image.crop_to_bounding_box(image, 0, start_x, size, size)

    return cropped_image

def save_image_without_augmentation(path, id, image, counter):
    if not os.path.isdir(path):
        os.makedirs(path)

    img = (tf.keras.utils.array_to_img(crop_to_square(image)))
    keras.preprocessing.image.save_img(path + id + '_' + str(counter) + '.jpg',␣
 ↪img)

def save_image_with_augmentation(path, resume_id, image, counter):
    if not os.path.isdir(path):
        os.makedirs(path)

    img = (tf.keras.utils.array_to_img(crop_to_square(image)))

    id = resume_id + '_' + str(counter)

    # 0°
    ref0b0 = tf.image.resize(img, [resize_size, resize_size])
    keras.preprocessing.image.save_img(path + id + '_r0_b0.jpg', ref0b0)

    ref0b0flip = tf.image.flip_left_right(img)
    keras.preprocessing.image.save_img(path + id + '_r0_b0_flip.jpg',␣
 ↪ref0b0flip)
```

[ ]:
```
np.random.seed(3) # Random numbers will be ever the same
rnd = np.random.rand(len(imgs)) < 0.8 # Training set will contain 80% of the␣
 ↪data

train_x = []
train_y = []
test_x = []
```

```
test_y = []

print('imgs[0] dimensions are: ', imgs[0].shape)

counter = 0
tests_n = 0

for x in range(0, len(ids)):
    # Creating the training dataset (80%)
    rnd = random()
    counter = counter + 1

    if(rnd <= 0.8):
        save_image_with_augmentation('data/' + specie_out + '/training/',␣
 ↪ids[x], imgs[x], counter)
        continue

    save_image_without_augmentation('data/' + specie_out + '/test/', ids[x],␣
 ↪imgs[x], counter)
    tests_n = tests_n + 1
```

[ ]:
```
if not os.path.isdir('data/' + specie_out + '/test_with_training'):
    os.makedirs('data/' + specie_out + '/test_with_training')

files = os.listdir('data/' + specie_out + '/training')
shuffle(files)

files = files[:tests_n]

for file in files:
    shutil.copyfile('data/' + specie_out + '/training/' + file, 'data/' +␣
 ↪specie_out + '/test_with_training/' + file)
```

APÊNDICE C – *NOTEBOOK* - TREINAMENTOS E OTIMIZAÇÕES

Este apêndice apresenta o *notebook* Jupyter utilizado durante a série de experimentos para treinamento e otimização das arquiteturas base. O *notebook* inclui todo o código utilizado para o treinamento e ajuste dos hiperparâmetros das redes.

notebook-optimization

November 21, 2024

Este notebook realiza a otimização de hiperparâmetros, treinamento e a validação da CNN.

CNN Base: **Xception**

Dataset: **Azevem**

[ ]:
```
import sys
import os

global_root = os.path.join('E:/', 'mestrado-ai')

if 'google.colab' in sys.modules:
    from google.colab import drive
    drive.mount(os.path.join('/', 'content', 'drive'))
    global_root = os.path.join('/', 'content', 'drive', 'MyDrive',␣
 ↪'mestrado-ai')

!pip install keras-tuner -q

sys.path.insert(0, global_root)
```

```
Mounted at /content/drive
                         129.1/129.1 kB
2.9 MB/s eta 0:00:00
```

[ ]:
```
global_config_base_folder = global_root # Pasta raiz do projeto.

global_config_base_choice = 'Xception' # VGG16, Xception ou EfficientNetV2S.
global_config_dataset_name = 'azevem' # Nome do dataset a ser utilizado.
global_config_out_folder = 'a100-xception-azevem-hflip' # Nome da pasta onde␣
 ↪serão salvos os resultados.

global_config_keras_search = True # True se deve realizar a busca por␣
 ↪hiperparâmetros.
global_config_training = True # True se deve realizar o treinamento.
global_config_from_last_training = False # Se True, continua treinamento a␣
 ↪partir da última repetição.
```

```
global_config_plot_images = True # Se True, plota imagens de exemplo da␣
 ↪predição.
global_config_debug_training_data = True # Se True, plota correlação de dados␣
 ↪de treinamento.

global_config_total_repetitions = 1 # Total de repetições treinamento +␣
 ↪validação.
global_config_keras_max_trials = 20 # Total de tentativas de hiperparâmetros.
```

[ ]:
```
# Bibliotecas padrão.
import json
import csv
import math
import shutil
import time
import random

# Bibliotecas científicas e de dados.
import numpy as np
import pandas as pd
from numpy.random import randn
from scipy.stats import shapiro
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Bibliotecas de visualização.
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns

# Bibliotecas de machine learning e deep learning.
import tensorflow as tf
import tensorflow.compat.v2 as tfv2
from tensorflow import keras
from keras import activations
from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping
import keras_tuner
import keras.applications as apps

# Gerador de dados para treinamento/teste.
from data_generators_5 import DataGenerator

tf.keras.utils.set_random_seed(536)

print(tf.__version__)
```

```
2.17.0
```

[ ]:
```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Wed Nov 13 11:33:30 2024
+---------------------------------------------------------------------------------
--------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version:
12.2     |
|-----------------------------------------+----------------------+-------------
--------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util
Compute M. |
|                                         |                      |
MIG M. |
|=========================================+======================+==============
========|
|   0  NVIDIA A100-SXM4-40GB          Off | 00000000:00:04.0 Off |
0 |
| N/A   30C    P0              46W / 400W |      2MiB / 40960MiB |      0%
Default |
|                                         |                      |
Disabled |
+-----------------------------------------+----------------------+-------------
--------+

+---------------------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                            GPU
Memory |
|        ID   ID
Usage      |
|=================================================================================
========|
|  No running processes found
|
+---------------------------------------------------------------------------------
--------+
```

[ ]:
```
batch_size = 32
epochs = 30

training_path = os.path.join(global_config_base_folder, 'dataset',␣
 ↪global_config_dataset_name, 'training')
test_path = os.path.join(global_config_base_folder, 'dataset',␣
 ↪global_config_dataset_name, 'test')
test_with_training_path = os.path.join(global_config_base_folder, 'dataset',␣
 ↪global_config_dataset_name, 'test_with_training')
experiment_path = os.path.join(global_config_base_folder, 'dataset',␣
 ↪global_config_dataset_name)

tuner_path = os.path.join(global_config_base_folder, global_config_out_folder,␣
 ↪'tuner')
training_csv_path = os.path.join(global_config_base_folder,␣
 ↪global_config_out_folder)
model_path = os.path.join(global_config_base_folder, global_config_out_folder,␣
 ↪'model')
best_model_path = os.path.join(global_config_base_folder,␣
 ↪global_config_out_folder, 'best-model')
model_checkpoint_path = os.path.join(global_config_base_folder,␣
 ↪global_config_out_folder, 'model-checkpoint')
min_max_json_file = os.path.join(global_config_base_folder,␣
 ↪global_config_out_folder, 'min-max.json')
repetitions_path = os.path.join(global_config_base_folder,␣
 ↪global_config_out_folder, 'repetitions')
board_path = os.path.join(global_config_base_folder, global_config_out_folder,␣
 ↪'board')

print(f"Training Path: {training_path}")
print(f"Test Path: {test_path}")
print(f"Test with Training Path: {test_with_training_path}")
print(f"Experiment Path: {experiment_path}")
print(f"Tuner Path: {tuner_path}")
print(f"Training CSV Path: {training_csv_path}")
print(f"Model Path: {model_path}")
print(f"Best Model Path: {best_model_path}")
print(f"Model Checkpoint Path: {model_checkpoint_path}")
print(f"Min-Max JSON File: {min_max_json_file}")
```

```
Training Path: /content/drive/MyDrive/mestrado-ai/dataset/azevem/training
Test Path: /content/drive/MyDrive/mestrado-ai/dataset/azevem/test
Test with Training Path: /content/drive/MyDrive/mestrado-
ai/dataset/azevem/test_with_training
Experiment Path: /content/drive/MyDrive/mestrado-ai/dataset/azevem
Tuner Path: /content/drive/MyDrive/mestrado-ai/a100-xception-azevem-hflip/tuner
Training CSV Path: /content/drive/MyDrive/mestrado-ai/a100-xception-azevem-hflip
```

```
Model Path: /content/drive/MyDrive/mestrado-ai/a100-xception-azevem-hflip/model
Best Model Path: /content/drive/MyDrive/mestrado-ai/a100-xception-azevem-
hflip/best-model
Model Checkpoint Path: /content/drive/MyDrive/mestrado-ai/a100-xception-azevem-
hflip/model-checkpoint
Min-Max JSON File: /content/drive/MyDrive/mestrado-ai/a100-xception-azevem-
hflip/min-max.json
```

[ ]:
```python
def remove_and_recreate_directory(directory_path):
    try:
        # Remove o diretório se existir
        shutil.rmtree(directory_path)
        print(f'Diretório {directory_path} removido com sucesso.')
    except FileNotFoundError:
        print(f'O diretório {directory_path} não existe.')

    # Cria o diretório novamente
    os.makedirs(directory_path)
    print(f'Diretório {directory_path} recriado com sucesso.')

def save_min_max(data_gen, path):
  min_max = { 'min': data_gen.data_min, 'max': data_gen.data_max }

  with open(path, 'w') as f:
    json.dump(min_max, f, ensure_ascii=False)

def softplus_activation(x):
  return activations.softplus(x)

def end_colab():
  from google.colab import runtime
  runtime.unassign()

# Função para desnormalizar os valores
def denormalize(values, data_min, data_max):
    return values * (data_max - data_min) + data_min
```

0.1 A próxima seção explora a otimização de hiperparâmetros usando o Keras Tuner, empregando a otimização bayesiana para encontrar a melhor configuração do modelo.

[ ]:
```python
size = 360
conv_input_size = size

def build_model_x():
    #with strategy.scope():  # Garantir que o modelo seja criado com TPU
```

```
    if global_config_base_choice == 'VGG16':
        conv_base = apps.VGG16(weights='imagenet', include_top=False,␣
↪input_shape=(conv_input_size, conv_input_size, 3))
        #preprocess_input = tf.keras.applications.vgg16.preprocess_input
    elif global_config_base_choice == 'Xception':
        conv_base = apps.Xception(weights='imagenet', include_top=False,␣
↪input_shape=(conv_input_size, conv_input_size, 3))
        #preprocess_input = tf.keras.applications.xception.preprocess_input
    elif global_config_base_choice == 'EfficientNetV2S':
        conv_base = apps.EfficientNetV2S(weights='imagenet', include_top=False,␣
↪input_shape=(conv_input_size, conv_input_size, 3))
        #preprocess_input = tf.keras.applications.efficientnet_v2.
↪preprocess_input

    '''model = keras.models.Sequential()
    # Ajustar a entrada para ser compatível com TPU (menor resolução)
    model.add(keras.Input(shape=(size, size, 3)))
    #model.add(tf.keras.layers.Resizing(size, size))
    #model.add(tf.keras.layers.Lambda(preprocess_input))
    model.add(conv_base)

    model.add(keras.layers.Flatten())

    # Adiciona camadas densas fixas com 288 unidades e ativação 'relu'
    model.add(keras.layers.Dense(288, activation='relu'))
    model.add(keras.layers.Dense(288, activation='relu'))

    # Camada de saída com ativação 'softplus'
    model.add(keras.layers.Dense(1, activation='softplus'))'''

    # Modelo usando Functional API para controle de fluxo de dados
    inputs = keras.layers.Input(shape=(size, size, 3))
    x = conv_base(inputs)
    x = keras.layers.Flatten()(x) #166 416 416
    #x = keras.layers.Dense(288, activation='relu')(x)
    x = keras.layers.Dense(288, activation='relu')(x)
    x = keras.layers.Dense(288, activation='relu')(x)
    #outputs = keras.layers.Dense(1, activation='softplus')(x)
    outputs = keras.layers.Dense(1, activation='linear')(x)

    model = keras.Model(inputs, outputs)

    # Taxa de aprendizado fixa de 1e-5
    learning_rate = 1e-5

    conv_base.trainable = True
```

```
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
        loss=keras.losses.MeanSquaredError(),
        metrics=[keras.metrics.MeanSquaredError(), keras.metrics.
 ↪MeanAbsoluteError()]
    )

    return model

# Exemplo de uso
build_model_x()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256             4s
0us/step
```

[ ]:
```
<Functional name=functional, built=True>
```

[ ]:
```
size = 360
conv_input_size = size

def build_model(hp):
    #with strategy.scope():  # Garantir que o modelo seja criado com TPU
    if global_config_base_choice == 'VGG16':
        conv_base = apps.VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(conv_input_size, conv_input_size, 3))
    elif global_config_base_choice == 'Xception':
        conv_base = apps.Xception(weights='imagenet', include_top=False,␣
 ↪input_shape=(conv_input_size, conv_input_size, 3))
    elif global_config_base_choice == 'EfficientNetV2S':
        conv_base = apps.EfficientNetV2S(weights='imagenet', include_top=False,␣
 ↪input_shape=(conv_input_size, conv_input_size, 3))

    model = tf.keras.models.Sequential()
    model.add(keras.Input(shape=(size, size, 3)))
    model.add(conv_base)
    model.add(tf.keras.layers.Flatten())

    # Tune the number of layers.
    for i in range(hp.Int("num_layers", 1, 3)):
      model.add(tf.keras.layers.Dense(units=hp.Int(f"units_{i}", min_value=32,␣
 ↪max_value=512, step=128), activation='relu'))

    activation_choice = hp.Choice('activation', ['softplus', 'linear'])

    if(activation_choice == 'softplus'):
```

```
      model.add(tf.keras.layers.Dense(1, activation='softplus'))
    elif(activation_choice == 'linear'):
      model.add(tf.keras.layers.Dense(1, activation='linear'))

    conv_base.trainable = True

    learning_rate_choice = hp.Choice('lr', [0.0001, 0.00001])

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate_choice),
        loss=keras.losses.MeanSquaredError(),
        metrics=[keras.metrics.MeanSquaredError(), keras.metrics.
↪MeanAbsoluteError()]
    )

    return model
```

[ ]:
```
def get_tuner(reset = False):
  return keras_tuner.BayesianOptimization(
    hypermodel=build_model,
    objective='loss',
    max_trials=global_config_keras_max_trials,##25,####15,
    #max_epochs=32,
    #factor=3,
    overwrite=False,#reset,
    directory=tuner_path,
    project_name='hpasture-kt')


def get_hb_tuner(reset=False):
    return keras_tuner.Hyperband(
        hypermodel=build_model,
        objective='loss',
        max_epochs=32,
        factor=3,
        overwrite=False,
        directory=tuner_path,
        project_name='hpasture-kt')

tuner = get_tuner(global_config_keras_search)
```

```
Reloading Tuner from /content/drive/MyDrive/mestrado-ai/a100-xception-azevem-
hflip/tuner/hpasture-kt/tuner0.json
```

[ ]:
```
if(not global_config_keras_search):
  raise Exception('Keras search is disabled.')
```

```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 4
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 3, 'step': 1,
'sampling': 'linear'}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step':
128, 'sampling': 'linear'}
activation (Choice)
{'default': 'softplus', 'conditions': [], 'values': ['softplus', 'linear'],
'ordered': False}
lr (Choice)
{'default': 0.0001, 'conditions': [], 'values': [0.0001, 1e-05], 'ordered':
True}
```

[ ]:
```
if(not global_config_keras_search):
  raise Exception('')

try:

  kt_early_stopping = EarlyStopping(monitor='loss', patience=5, verbose=1,␣
 ↪mode='min')

  data_gen = DataGenerator(training_path, experiment_path, batch_size)

  tuner.search(x=data_gen, epochs=epochs, callbacks=[kt_early_stopping, keras.
 ↪callbacks.TensorBoard(board_path)])

except Exception as e:
    print("Exception:", e)
    time.sleep(20)
    end_colab()
```

```
Trial 20 Complete [00h 01m 02s]

Best loss So Far: 0.00037461804458871484
Total elapsed time: 03h 02m 07s
```

[ ]:
```
if(not global_config_keras_search):
  raise Exception('')

tuner.results_summary()
```

```
Results summary
Results in /content/drive/MyDrive/mestrado-ai/a100-xception-azevem-
hflip/tuner/hpasture-kt
Showing 10 best trials
```

```
Objective(name="loss", direction="min")

Trial 14 summary
Hyperparameters:
num_layers: 3
units_0: 288
activation: softplus
lr: 0.0001
units_1: 32
units_2: 416
Score: 0.00037461804458871484

Trial 02 summary
Hyperparameters:
num_layers: 2
units_0: 32
activation: softplus
lr: 1e-05
units_1: 416
units_2: 160
Score: 0.0004954331088811159

Trial 00 summary
Hyperparameters:
num_layers: 3
units_0: 416
activation: softplus
lr: 1e-05
units_1: 32
units_2: 32
Score: 0.000572710414417088

Trial 17 summary
Hyperparameters:
num_layers: 2
units_0: 160
activation: softplus
lr: 0.0001
units_1: 32
units_2: 160
Score: 0.0006659436621703207

Trial 13 summary
Hyperparameters:
num_layers: 3
units_0: 32
activation: linear
lr: 1e-05
```

```
units_1: 160
units_2: 416
Score: 0.0011536559322848916

Trial 03 summary
Hyperparameters:
num_layers: 3
units_0: 32
activation: linear
lr: 0.0001
units_1: 416
units_2: 288
Score: 0.001258053001947701

Trial 04 summary
Hyperparameters:
num_layers: 3
units_0: 288
activation: softplus
lr: 1e-05
units_1: 416
units_2: 288
Score: 0.001334807020612061

Trial 11 summary
Hyperparameters:
num_layers: 2
units_0: 32
activation: linear
lr: 1e-05
units_1: 160
units_2: 416
Score: 0.0016394995618611574

Trial 15 summary
Hyperparameters:
num_layers: 2
units_0: 288
activation: linear
lr: 1e-05
units_1: 160
units_2: 288
Score: 0.0018469954375177622

Trial 09 summary
Hyperparameters:
num_layers: 3
units_0: 288
```

```
activation: linear
lr: 1e-05
units_1: 32
units_2: 160
Score: 0.001977980602532625
```

0.2 As próximas seções lidam com validação de acordo com os retornos do Keras Tuner.

[ ]:
```
# Arquitetura final.
best_model = tuner.get_best_models()[0]
best_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744              0s
0us/step

/usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:576:
UserWarning: Skipping variable loading for optimizer 'adam', because it has 2
variables whereas the saved optimizer has 326 variables.
  saveable.load_own_variables(weights_store.get(inner_path))

Model: "sequential"
```

Layer (type)	Output Shape	Param #
xception (Functional)	(None, 12, 12, 2048)	20,861,480
flatten (Flatten)	(None, 294912)	0
dense (Dense)	(None, 288)	84,934,944
dense_1 (Dense)	(None, 32)	9,248
dense_2 (Dense)	(None, 416)	13,728
dense_3 (Dense)	(None, 1)	417

Total params: 105,819,817 (403.67 MB)

Trainable params: 105,765,289 (403.46 MB)

Non-trainable params: 54,528 (213.00 KB)

[ ]:
```
# Executar código abaixo antes dos testes.

data_gen = DataGenerator(test_path, experiment_path, batch_size)

data_min = data_gen.data_min;
data_max = data_gen.data_max;

# Cria o dataset com tf.data para predição
##predict_dataset = data_gen.create_dataset(for_predict=True)

# Faz as predições
predictions = best_model.predict(x=data_gen, verbose=1)

# Obter valores reais
real_values = np.array(data_gen.get_real_values())

print(f"Tamanho das predições: {len(predictions)}")
print(f"Tamanho dos valores reais: {len(real_values)}")

# Garantir que as predições sejam achatadas em uma array 1D
predictions_flat = predictions.ravel()

# Calcular o coeficiente de correlação entre predições e valores reais
correlation = np.corrcoef(predictions_flat, real_values)[0, 1]

# Calcular MSE e MAE
mse = mean_squared_error(real_values, predictions_flat)
mae = mean_absolute_error(real_values, predictions_flat)

# Desnormalizar predições e valores reais
predictions_rescaled = predictions_flat * (data_max - data_min) + data_min
real_values_rescaled = real_values * (data_max - data_min) + data_min

# Calcular MAE e MSE nos valores desnormalizados
mse_rescaled = mean_squared_error(real_values_rescaled, predictions_rescaled)
mae_rescaled = mean_absolute_error(real_values_rescaled, predictions_rescaled)
```

```
# Salvar os resultados no arquivo CSV
#save_results(results_file, repetition, correlation, mse, mae, mse_rescaled,␣
 ↪mae_rescaled)

# Salvar predições e valores reais para esta repetição no JSON
#save_predictions_json(predictions_file, repetition, predictions_flat,␣
 ↪real_values)

print(f"Repetição N - Correlation: {correlation:.4f}, MSE: {mse:.4f}, MAE: {mae:
 ↪.4f}          |         MSE_: {mse_rescaled:.4f}, MAE_: {mae_rescaled:.4f}  ␣
 ↪    \n\n")
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
7/7             24s 2s/step
Tamanho das predições: 201
Tamanho dos valores reais: 201
Repetição N - Correlation: 0.9679, MSE: 0.0043, MAE: 0.0493          |
MSE_: 3.3864, MAE_: 1.3800
```

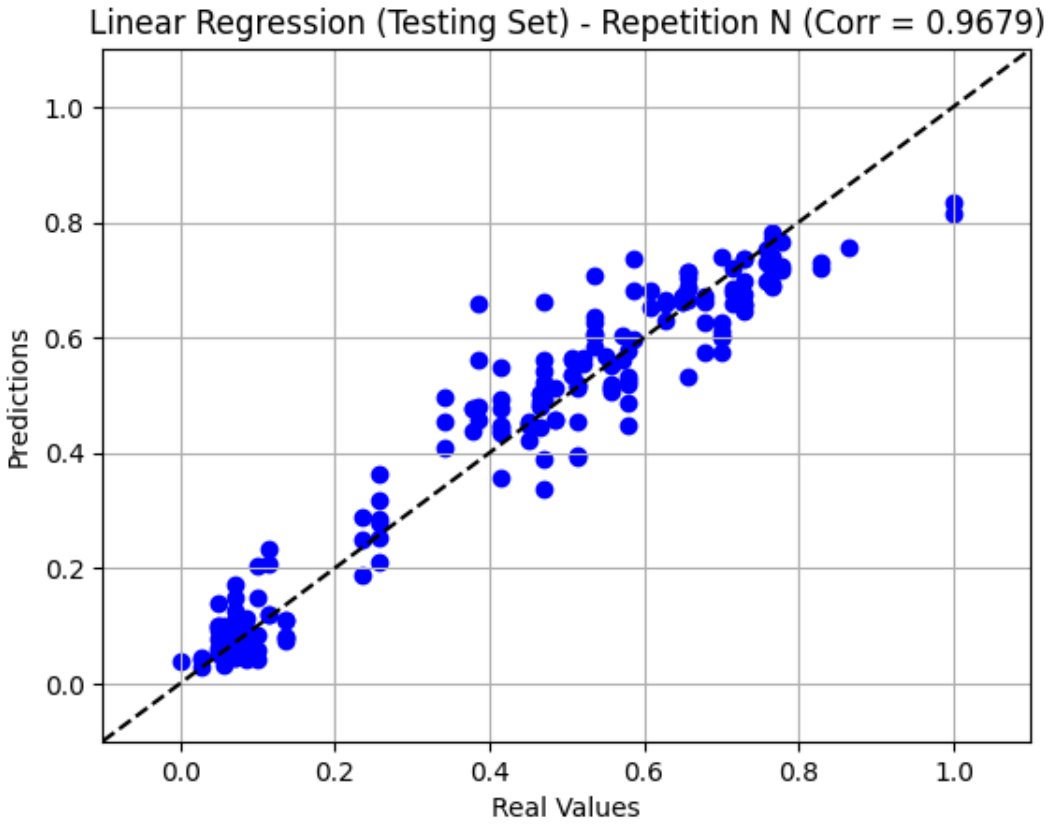
0.3 T1) Implementa analise a partir da saída do Keras.

[ ]:
```
# Carregar predições e valores reais da melhor repetição
predictions = predictions_flat

# Calcular o coeficiente de correlação de Pearson
correlation = np.corrcoef(real_values, predictions)[0, 1]
print(f"Correlation Coefficient - Repetition N: {correlation:.4f}\n")

# Gerar um gráfico de regressão (valores reais vs predições)
plt.scatter(real_values, predictions, marker = 'o', c = 'blue')
plt.plot([-0.1, 1.1], [-0.1, 1.1], color = 'black', ls = '--')  # Linha de␣
 ↪referência (x = y)
plt.ylabel('Predictions')
plt.xlabel('Real Values')
plt.title(f'Linear Regression (Testing Set) - Repetition N (Corr = {correlation:
 ↪.4f})')
plt.ylim(-0.1, 1.1)
```

```
plt.xlim(-0.1, 1.1)
plt.grid(True)
plt.show()

# Desnormalizar predições e valores reais
predictions_cm = predictions_rescaled
real_values_cm = real_values_rescaled

# Calcular os resíduos em cm
residuals = real_values_cm - predictions_cm
residuals_abs = np.abs(residuals)

# Calcular métricas
mse = mean_squared_error(real_values_cm, predictions_cm)
rmse = math.sqrt(mse)
mae = np.mean(residuals_abs)
mean_residual = np.mean(residuals)
std_residual = np.std(residuals)
shap_stat, shap_p_value = shapiro(residuals)

# Exibir os resultados da análise dos resíduos
print('\n\n\nResidual Analysis (Test Set) \n')
print('Model - MSE: {:.4f} cm²'.format(mse))
print('Model - RMSE: {:.4f} cm'.format(rmse))
print('Model - MAE: {:.4f} cm'.format(mae))
print('Residual Distribution - Mean: {:.4f} cm'.format(mean_residual))
print('Residual Distribution - SD: {:.4f} cm'.format(std_residual))
print('Residual Distribution - Shapiro-Wilk Test: Statistic = {:.4f}, p-value =␣
 ↪{:.8f}'.format(shap_stat, shap_p_value))

# Check normality based on p-value
if shap_p_value > 0.05:
    print("Residuals follow a normal distribution (p-value > 0.05)\n")
else:
    print("Residuals do NOT follow a normal distribution (p-value <= 0.05)\n")

# Plotar a distribuição dos resíduos
sns.displot(residuals, color="dodgerblue", label="Residuals", kind="kde")
plt.title('Residual Distribution (Testing Set)')
plt.xlabel('Residuals (cm)')
plt.ylabel('Density')
plt.show()

# Plota treinamento.
print('\n\nTraining\n')

'''history = pd.read_csv(training_csv, header=0, sep=",")
```

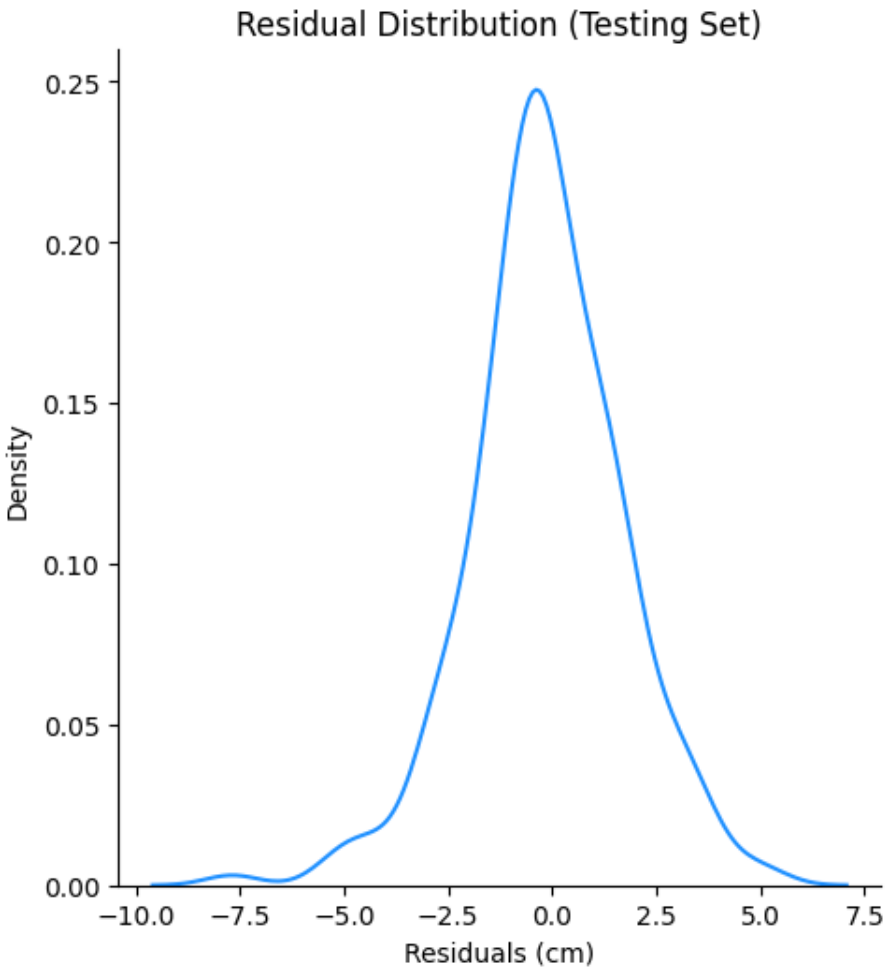
```
plt.plot(history['mean_squared_error'])
plt.title('Training MSE')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Error'], loc='upper right')
plt.show()'''
```

```
Correlation Coefficient - Repetition N: 0.9679
```




```
Residual Analysis (Test Set)

Model - MSE: 3.3864 cm²
Model - RMSE: 1.8402 cm
Model - MAE: 1.3800 cm
Residual Distribution - Mean: -0.1488 cm
Residual Distribution - SD: 1.8342 cm
```

```
Residual Distribution - Shapiro-Wilk Test: Statistic = 0.9812, p-value =
0.00850464
Residuals do NOT follow a normal distribution (p-value <= 0.05)
```



Training

```
[ ]: 'history = pd.read_csv(training_csv, header=0,
sep=",")\n\nplt.plot(history[\'mean_squared_error\'])\nplt.title(\'Training
MSE\')\nplt.ylabel(\'MSE\')\nplt.xlabel(\'Epoch\')\nplt.legend([\'Error\'],
loc=\'upper right\')\nplt.show()'
```

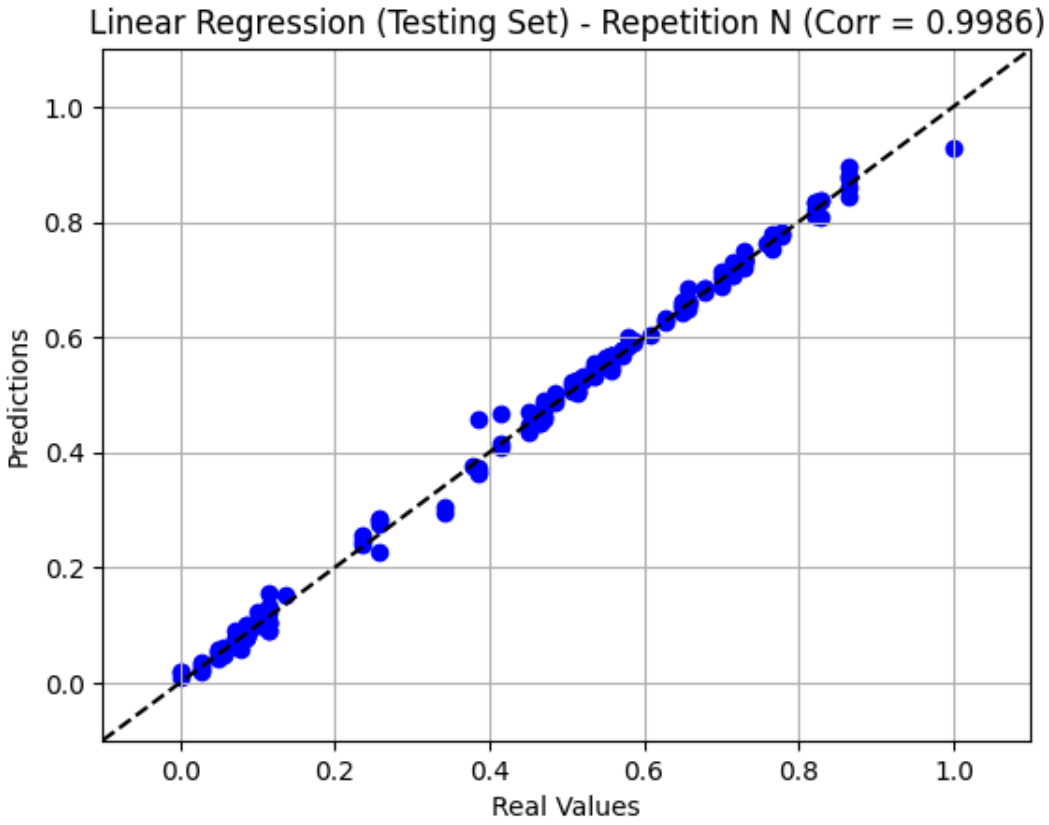
0.4 T2) Teste nos dados de treinamento.

[ ]:
```
# Carregar predições e valores reais da melhor repetição
data_gen = DataGenerator(test_with_training_path, experiment_path, batch_size)

# Cria o dataset com tf.data para predição
#predict_dataset = data_gen.create_dataset(for_predict=True)

# Faz as predições
predictions = best_model.predict(x=data_gen, verbose=1)

predictions_flat = predictions.ravel()

real_values = data_gen.get_real_values()

# Calcular o coeficiente de correlação de Pearson
correlation = np.corrcoef(real_values, predictions_flat)[0, 1]
print(f"Correlation Coefficient - Repetition N: {correlation:.4f}\n")

# Gerar um gráfico de regressão (valores reais vs predições)
plt.scatter(real_values, predictions, marker = 'o', c = 'blue')
plt.plot([-0.1, 1.1], [-0.1, 1.1], color = 'black', ls = '--')  # Linha de␣
 ↪referência (x = y)
plt.ylabel('Predictions')
plt.xlabel('Real Values')
plt.title(f'Linear Regression (Testing Set) - Repetition N (Corr = {correlation:
 ↪.4f})')
plt.ylim(-0.1, 1.1)
plt.xlim(-0.1, 1.1)
plt.grid(True)
plt.show()

# Desnormalizar predições e valores reais
predictions_cm = predictions_rescaled#denormalize(predictions, data_min,␣
 ↪data_max)
real_values_cm = real_values_rescaled#denormalize(real_values, data_min,␣
 ↪data_max)

# Calcular os resíduos em cm
residuals = real_values_cm - predictions_cm
residuals_abs = np.abs(residuals)

# Calcular métricas
mse = mean_squared_error(real_values_cm, predictions_cm)
rmse = math.sqrt(mse)
mae = np.mean(residuals_abs)
mean_residual = np.mean(residuals)
```

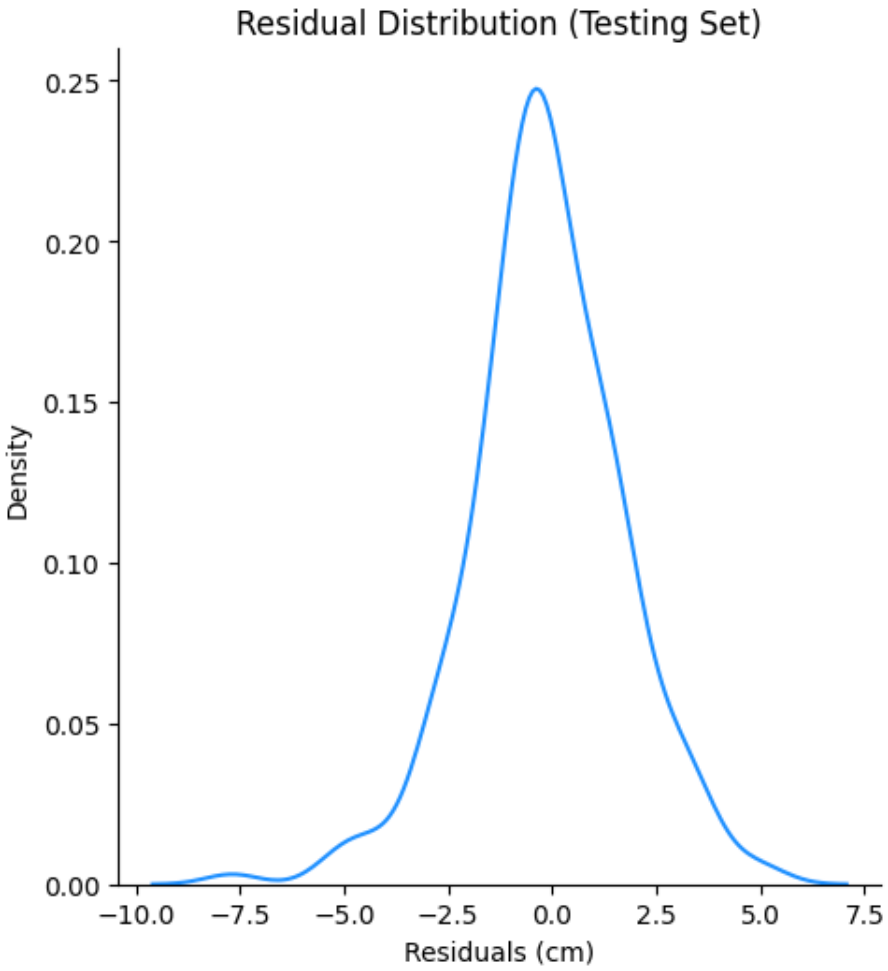
```
std_residual = np.std(residuals)
shap_stat, shap_p_value = shapiro(residuals)

# Exibir os resultados da análise dos resíduos
print('\n\n\nResidual Analysis (Test Set) \n')
print('Model - MSE: {:.4f} cm²'.format(mse))
print('Model - RMSE: {:.4f} cm'.format(rmse))
print('Model - MAE: {:.4f} cm'.format(mae))
print('Residual Distribution - Mean: {:.4f} cm'.format(mean_residual))
print('Residual Distribution - SD: {:.4f} cm'.format(std_residual))
print('Residual Distribution - Shapiro-Wilk Test: Statistic = {:.4f}, p-value =␣
 ↪{:.8f}'.format(shap_stat, shap_p_value))

# Check normality based on p-value
if shap_p_value > 0.05:
    print("Residuals follow a normal distribution (p-value > 0.05)\n")
else:
    print("Residuals do NOT follow a normal distribution (p-value <= 0.05)\n")

# Plotar a distribuição dos resíduos
sns.displot(residuals, color="dodgerblue", label="Residuals", kind="kde")
plt.title('Residual Distribution (Testing Set)')
plt.xlabel('Residuals (cm)')
plt.ylabel('Density')
plt.show()

# Plota treinamento.
print('\n\nTraining\n')

'''history = pd.read_csv(training_csv, header=0, sep=",")

plt.plot(history['mean_squared_error'])
plt.title('Training MSE')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Error'], loc='upper right')
plt.show()'''
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
7/7             2s 294ms/step
Correlation Coefficient - Repetition N: 0.9986
```




```
Residual Analysis (Test Set)

Model - MSE: 3.3864 cm²
Model - RMSE: 1.8402 cm
Model - MAE: 1.3800 cm
Residual Distribution - Mean: -0.1488 cm
Residual Distribution - SD: 1.8342 cm
Residual Distribution - Shapiro-Wilk Test: Statistic = 0.9812, p-value =
0.00850464
Residuals do NOT follow a normal distribution (p-value <= 0.05)
```



Training

[ ]:
```
'history = pd.read_csv(training_csv, header=0,
sep=",")\n\nplt.plot(history[\'mean_squared_error\'])\nplt.title(\'Training
MSE\')\nplt.ylabel(\'MSE\')\nplt.xlabel(\'Epoch\')\nplt.legend([\'Error\'],
loc=\'upper right\')\nplt.show()'
```

0.5 T3) Teste em imagens.

[ ]:
```
if(not global_config_plot_images):
  raise Exception('Plot images is disabled.')

# Validação e cálculo do coeficiente de correlação
```

```
val_model = best_model

image_data_gen = DataGenerator(test_path, experiment_path, batch_size)

# Supondo que já tenha o modelo carregado e o image_data_gen instanciado
real_values, images = image_data_gen.get_real_values_and_images()

# Configurações para a grid de imagens
rows, cols = 3, 3
f, axarr = plt.subplots(nrows=rows, ncols=cols, figsize=(30, 30))

for row in range(0, rows):
    for col in range(0, cols):
        idx0 = random.randint(0, len(real_values) - 1)  # Pegar um índice␣
↪aleatório
        true_y = real_values[idx0]  # Pega o valor real
        image = images[idx0]

        # Fazer a predição usando o modelo
        predictions = val_model.predict(tf.expand_dims(images[idx0], axis=0)) ␣
↪# Redimensionar para (1, height, width, 3)
        pred_cm = denormalize(predictions[0][0], data_min, data_max)  #␣
↪Converter a predição para cm

        # Mostrar a imagem na grid
        axarr[row, col].imshow(image)
        axarr[row, col].set_title('Pred = {:.2f} cm   Real = {:.2f} cm'.
↪format(pred_cm, denormalize(true_y, image_data_gen.data_min, image_data_gen.
↪data_max)))
        axarr[row, col].axis('off')  # Ocultar os eixos

plt.show()
```

```
1/1             3s 3s/step
1/1             0s 25ms/step
1/1             0s 26ms/step
1/1             0s 33ms/step
1/1             0s 27ms/step
1/1             0s 25ms/step
1/1             0s 24ms/step
1/1             0s 25ms/step
1/1             0s 25ms/step
```

0.6 TensorBoard

[ ]:
```
%load_ext tensorboard
```

[ ]:
```
%tensorboard --logdir /content/drive/MyDrive/mestrado-ai/
↪a100-xception-azevem-hflip/board/
```

```
<IPython.core.display.Javascript object>
```

0.7 Shutdown

[ ]:
```
# Terminate Colab session...
time.sleep(30)
end_colab()
```

APÊNDICE D – *NOTEBOOK* - VALIDAÇÃO ESTATÍSTICA

Este apêndice apresenta um *notebook* Jupyter contendo o código completo utilizado para a validação estatística das redes neurais desenvolvidas.

notebook-statistical-test

November 21, 2024

Este notebook aplica teste estatístico nos modelos desenvolvidos.

[ ]:
```
import sys
import os

global_root = os.path.join('E:/', 'mestrado-ai')

if 'google.colab' in sys.modules:
    from google.colab import drive
    drive.mount(os.path.join('/', 'content', 'drive'))
    global_root = os.path.join('/', 'content', 'drive', 'MyDrive',␣
 ↪'mestrado-ai')

sys.path.insert(0, global_root)

!pip install statsmodels
!pip install keras-tuner -q
!pip install scikit-posthocs

global_config_base_folder = global_root # Pasta raiz do projeto.
```

```
Mounted at /content/drive
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-
packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (1.13.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (2.2.2)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-
packages (from statsmodels) (1.0.1)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.2)
```

```
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels)
(1.16.0)
                         129.1/129.1 kB
4.7 MB/s eta 0:00:00
Collecting scikit-posthocs
  Downloading scikit_posthocs-0.10.0-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from scikit-posthocs) (1.26.4)
Requirement already satisfied: scipy>=1.9.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-posthocs) (1.13.1)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-
packages (from scikit-posthocs) (0.14.4)
Requirement already satisfied: pandas>=0.20.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-posthocs) (2.2.2)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-
packages (from scikit-posthocs) (0.13.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from scikit-posthocs) (3.8.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->scikit-posthocs)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.20.0->scikit-posthocs) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.20.0->scikit-posthocs) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-posthocs)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->scikit-posthocs) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-posthocs)
(4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-posthocs)
(1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-posthocs)
(24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->scikit-posthocs) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-posthocs)
(3.2.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-
```

```
packages (from statsmodels->scikit-posthocs) (1.0.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas>=0.20.0->scikit-posthocs) (1.16.0)
Downloading scikit_posthocs-0.10.0-py3-none-any.whl (33 kB)
Installing collected packages: scikit-posthocs
Successfully installed scikit-posthocs-0.10.0
```

[ ]:
```python
# Bibliotecas padrão.
import json
import csv
import math
import shutil
import time
import random

# Bibliotecas científicas e de dados.
import numpy as np
import pandas as pd
from numpy.random import randn
import scipy.stats as stats
from sklearn.metrics import mean_squared_error, mean_absolute_error
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from scikit_posthocs import posthoc_dunn, posthoc_nemenyi_friedman

# Bibliotecas de visualização.
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib as mtp
import seaborn as sns

# Bibliotecas de machine learning e deep learning.
import tensorflow as tf
import tensorflow.compat.v2 as tfv2
from tensorflow import keras
from keras import activations
from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping
import keras_tuner
import keras.applications as apps
from tensorflow.keras.models import Model

# Gerador de dados para treinamento/teste.
from data_generators_5 import DataGenerator

tf.keras.utils.set_random_seed(536)
```

[ ]:
```python
size = 360
conv_input_size = size
```

```
def build_model(hp, base_model_name):
    # Escolher a base da rede de acordo com o parâmetro fornecido.
    if base_model_name == 'VGG16':
        conv_base = apps.VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(conv_input_size, conv_input_size, 3))
    elif base_model_name == 'Xception':
        conv_base = apps.Xception(weights='imagenet', include_top=False,␣
 ↪input_shape=(conv_input_size, conv_input_size, 3))
    elif base_model_name == 'EfficientNetV2S':
        conv_base = apps.EfficientNetV2S(weights='imagenet', include_top=False,␣
 ↪input_shape=(conv_input_size, conv_input_size, 3))
    else:
        raise ValueError("Invalid base_model_name. Choose from 'VGG16',␣
 ↪'Xception', or 'EfficientNetV2S'.")

    model = tf.keras.models.Sequential()
    model.add(keras.Input(shape=(size, size, 3)))
    model.add(conv_base)
    model.add(tf.keras.layers.Flatten())

    for i in range(hp.Int("num_layers", 1, 3)):
        model.add(tf.keras.layers.Dense(units=hp.Int(f"units_{i}",␣
 ↪min_value=32, max_value=512, step=128), activation='relu'))

    activation_choice = hp.Choice('activation', ['softplus', 'linear'])
    model.add(tf.keras.layers.Dense(1, activation=activation_choice))

    conv_base.trainable = True

    learning_rate_choice = hp.Choice('lr', [0.0001, 0.00001])

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate_choice),
        loss=keras.losses.MeanSquaredError(),
        metrics=[keras.metrics.MeanSquaredError(), keras.metrics.
 ↪MeanAbsoluteError()]
    )

    return model

def build_model_func(hp, base_model_name, best_model, submodel):
    # Entrada do modelo.
    inputs = keras.layers.Input(shape=(size, size, 3))

    # Rede base.
```

```
    # Escolher a base da rede de acordo com o parâmetro fornecido.
    if base_model_name == 'VGG16':
        conv_base = apps.VGG16(input_tensor=inputs, weights='imagenet',␣
↪include_top=False, input_shape=(conv_input_size, conv_input_size, 3))
    elif base_model_name == 'Xception':
        conv_base = apps.Xception(input_tensor=inputs, weights='imagenet',␣
↪include_top=False, input_shape=(conv_input_size, conv_input_size, 3))
    elif base_model_name == 'EfficientNetV2S':
        conv_base = apps.EfficientNetV2S(input_tensor=inputs,␣
↪weights='imagenet', include_top=False, input_shape=(conv_input_size,␣
↪conv_input_size, 3))
    else:
        raise ValueError("Invalid base_model_name. Choose from 'VGG16',␣
↪'Xception', or 'EfficientNetV2S'.")

    # Flatten.
    x = keras.layers.Flatten()(conv_base.output)

    for i in range(hp.Int("num_layers", 1, 3)):
        units = hp.Int(f"units_{i}", min_value=32, max_value=512, step=128)
        x = keras.layers.Dense(units=units, activation='relu')(x)

    activation_choice = hp.Choice('activation', ['softplus', 'linear'])

    outputs = keras.layers.Dense(1, activation=activation_choice)(x)

    # Criação do modelo.
    model = Model(inputs=inputs, outputs=outputs)

    conv_base.trainable = True

    learning_rate_choice = hp.Choice('lr', [0.0001, 0.00001])

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate_choice),
        loss=keras.losses.MeanSquaredError(),
        metrics=[keras.metrics.MeanSquaredError(), keras.metrics.
↪MeanAbsoluteError()]
    )

    # To functional.
    for layer_1, layer_2 in zip(model.layers, submodel.layers):
        if len(layer_1.get_weights()) > 0 and len(layer_2.get_weights()) > 0:
            layer_1.set_weights(layer_2.get_weights())

    # To denses.
```

```
    for layer_1, layer_2 in zip(best_model.layers[1:], model.
 ↪layers[len(submodel.layers):]):
        if len(layer_1.get_weights()) > 0 and len(layer_2.get_weights()) > 0:
            layer_2.set_weights(layer_1.get_weights())

    return model

def get_tuner(path, base_model_name):
    return keras_tuner.BayesianOptimization(
        hypermodel=lambda hp: build_model(hp, base_model_name),
        objective='loss',
        overwrite=False,
        directory=path,
        project_name='hpasture-kt'
    )

def get_tuner_folder(base_cnn_out_folder):
    return os.path.join(global_config_base_folder, base_cnn_out_folder, 'tuner')

def get_heatmap_folder(base_cnn_out_folder):
    return os.path.join(global_config_base_folder, base_cnn_out_folder,␣
 ↪'heatmap')

def get_test_folder(vegetation_name):
    return os.path.join(global_config_base_folder, 'dataset', vegetation_name,␣
 ↪'test')

def get_experiment_folder(vegetation_name):
    return os.path.join(global_config_base_folder, 'dataset', vegetation_name)

def remove_and_recreate_directory(directory_path):
    try:
        # Remove o diretório se existir
        shutil.rmtree(directory_path)
        print(f'Diretório {directory_path} removido com sucesso.')
    except FileNotFoundError:
        print(f'O diretório {directory_path} não existe.')

    # Cria o diretório novamente
    os.makedirs(directory_path)
    print(f'Diretório {directory_path} recriado com sucesso.')

def denormalize(values, data_min, data_max):
    return values * (data_max - data_min) + data_min
```

[ ]:
```
vgg16_folder = 'tpu-vgg16-nativox-hflip'
xception_folder = 'tpu-xception-nativox-hflip'
```

```
effv2_folder = 'tpu-efficientnetv2s-nativox-hflip'
vegetation_name = 'nativo'

batch_size = 32
```

[ ]:
```
#data_gen = DataGenerator(get_test_folder(vegetation_name),␣
↪get_experiment_folder(vegetation_name), batch_size)
tuner_vgg16 = get_tuner(get_tuner_folder(vgg16_folder), 'VGG16')
tuner_xception = get_tuner(get_tuner_folder(xception_folder), 'Xception')
tuner_effv2 = get_tuner(get_tuner_folder(effv2_folder), 'EfficientNetV2S')
```

```
Reloading Tuner from /content/drive/MyDrive/mestrado-ai/tpu-vgg16-nativox-
hflip/tuner/hpasture-kt/tuner0.json
Reloading Tuner from /content/drive/MyDrive/mestrado-ai/tpu-xception-nativox-
hflip/tuner/hpasture-kt/tuner0.json
Reloading Tuner from /content/drive/MyDrive/mestrado-ai/tpu-
efficientnetv2s-nativox-hflip/tuner/hpasture-kt/tuner0.json
```

[ ]:
```
best_hp_vgg16 = tuner_vgg16.get_best_hyperparameters()[0]
print(best_hp_vgg16.values)

best_hp_xception = tuner_xception.get_best_hyperparameters()[0]
print(best_hp_xception.values)

best_hp_effv2 = tuner_effv2.get_best_hyperparameters()[0]
print(best_hp_effv2.values)
```

```
{'num_layers': 2, 'units_0': 32, 'activation': 'linear', 'lr': 1e-05, 'units_1':
160, 'units_2': 416}
{'num_layers': 3, 'units_0': 32, 'activation': 'linear', 'lr': 0.0001,
'units_1': 416, 'units_2': 288}
{'num_layers': 3, 'units_0': 288, 'activation': 'softplus', 'lr': 0.0001,
'units_1': 32, 'units_2': 416}
```

[ ]:
```
best_model_vgg16 = tuner_vgg16.get_best_models()[0]
best_model_vgg16.summary()
print(best_model_vgg16.layers[-1].activation)

best_model_xception = tuner_xception.get_best_models()[0]
best_model_xception.summary()
print(best_model_xception.layers[-1].activation)

best_model_effv2 = tuner_effv2.get_best_models()[0]
best_model_effv2.summary()
print(best_model_effv2.layers[-1].activation)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256             0s
```

```
0us/step
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:713:
UserWarning: Skipping variable loading for optimizer 'adam', because it has 2
variables whereas the saved optimizer has 66 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
```

```
Model: "sequential"
```

```
 Layer (type)                        Output Shape                      ␣
↪Param #

 vgg16 (Functional)                  (None, 11, 11, 512)               ␣
↪14,714,688

 flatten (Flatten)                   (None, 61952)                          ␣
↪  0

 dense (Dense)                       (None, 32)                        ␣
↪1,982,496

 dense_1 (Dense)                     (None, 160)                            ␣
↪5,280

 dense_2 (Dense)                     (None, 1)                              ␣
↪161


 Total params: 16,702,625 (63.72 MB)

 Trainable params: 16,702,625 (63.72 MB)

 Non-trainable params: 0 (0.00 B)
```

```
<function linear at 0x788ce828a680>
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744             0s
0us/step
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:713:
UserWarning: Skipping variable loading for optimizer 'adam', because it has 2
variables whereas the saved optimizer has 326 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
xception (Functional)	(None, 12, 12, 2048)	20,861,480
flatten (Flatten)	(None, 294912)	0
dense (Dense)	(None, 32)	9,437,216
dense_1 (Dense)	(None, 416)	13,728
dense_2 (Dense)	(None, 288)	120,096
dense_3 (Dense)	(None, 1)	289

```
Total params: 30,432,809 (116.09 MB)

Trainable params: 30,378,281 (115.88 MB)

Non-trainable params: 54,528 (213.00 KB)

<function linear at 0x788ce828a680>
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/efficientnet_v2/efficientnetv2-s_notop.h5
82420632/82420632             1s
0us/step

/usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:713:
UserWarning: Skipping variable loading for optimizer 'adam', because it has 2
variables whereas the saved optimizer has 918 variables.
  saveable.load_own_variables(weights_store.get(inner_path))

Model: "sequential"
```

```
Layer (type)                                Output Shape                                ␣
↪Param #

efficientnetv2-s (Functional)               (None, 12, 12, 1280)                        ␣
↪20,331,360

flatten (Flatten)                           (None, 184320)                                 ␣
↪  0

dense (Dense)                               (None, 288)                                 ␣
↪53,084,448

dense_1 (Dense)                             (None, 32)                                     ␣
↪9,248

dense_2 (Dense)                             (None, 416)                                 ␣
↪13,728

dense_3 (Dense)                             (None, 1)                                      ␣
↪417


Total params: 73,439,201 (280.15 MB)

Trainable params: 73,285,329 (279.56 MB)

Non-trainable params: 153,872 (601.06 KB)

<function softplus at 0x788ce828a170>
```

[ ]:
```
func_model_vgg16 = build_model_func(tuner_vgg16.get_best_hyperparameters()[0],␣
↪'VGG16', best_model_vgg16, best_model_vgg16.get_layer("vgg16"))
func_model_xception = build_model_func(tuner_xception.
↪get_best_hyperparameters()[0], 'Xception', best_model_xception,␣
↪best_model_xception.get_layer("xception"))
func_model_effv2 = build_model_func(tuner_effv2.get_best_hyperparameters()[0],␣
↪'EfficientNetV2S', best_model_effv2, best_model_effv2.
↪get_layer("efficientnetv2-s"))
```

[ ]:
```
# Executar código abaixo antes dos testes.

data_gen = DataGenerator(get_test_folder(vegetation_name),␣
↪get_experiment_folder(vegetation_name), batch_size)
data_gen2 = DataGenerator(get_test_folder(vegetation_name),␣
↪get_experiment_folder(vegetation_name), batch_size)
```

```
data_gen3 = DataGenerator(get_test_folder(vegetation_name),␣
 ↪get_experiment_folder(vegetation_name), batch_size)

data_min = data_gen.data_min;
data_max = data_gen.data_max;

real_values = data_gen.get_real_values()

# Faz as predições.
predictions_vgg16 = func_model_vgg16.predict(x=data_gen, verbose=1)
predictions_xception = func_model_xception.predict(x=data_gen2, verbose=1)
predictions_effv2 = func_model_effv2.predict(x=data_gen3, verbose=1)

predictions_vgg16 = predictions_vgg16.ravel()
predictions_xception = predictions_xception.ravel()
predictions_effv2 = predictions_effv2.ravel()

real_values = np.array(real_values).ravel()

residuals1 = real_values - predictions_vgg16
residuals2 = real_values - predictions_xception
residuals3 = real_values - predictions_effv2
abs_errors1 = np.abs(real_values - predictions_vgg16)
abs_errors2 = np.abs(real_values - predictions_xception)
abs_errors3 = np.abs(real_values - predictions_effv2)
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
7/7             54s 4s/step
7/7             50s 3s/step
```

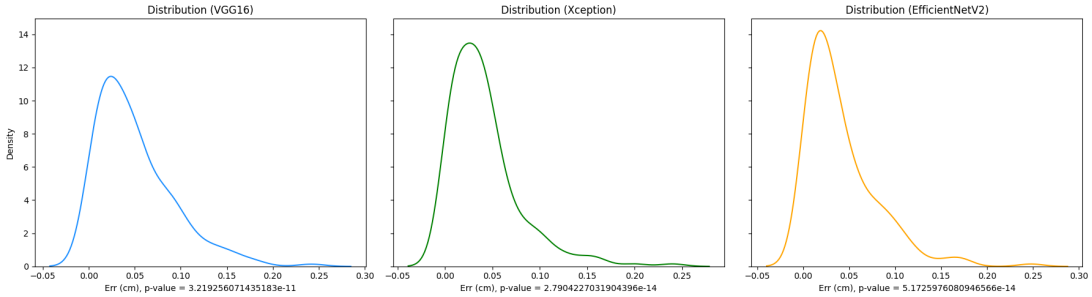
```
WARNING:tensorflow:5 out of the last 15 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x788c50260820> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
7/7             33s 3s/step
```

1 Normality Analysis

[ ]:
```
# Teste normalidade.

print('\nNormality Analysis (Test Set) \n')

shap_stat, shap_p_value1 = stats.shapiro(abs_errors1)
print('VGG16 - Distribution - Shapiro-Wilk Test: Statistic = {:.4f}, p-value =␣
 ↪{:.12f}'.format(shap_stat, shap_p_value1))
print("Residuals follow a normal distribution (p-value > 0.05)\n" if␣
 ↪shap_p_value1 > 0.05 else "Residuals do NOT follow a normal distribution␣
 ↪(p-value <= 0.05)\n")

shap_stat, shap_p_value2 = stats.shapiro(abs_errors2)
print('Xception - Distribution - Shapiro-Wilk Test: Statistic = {:.4f}, p-value␣
 ↪= {:.12f}'.format(shap_stat, shap_p_value2))
print("Err follow a normal distribution (p-value > 0.05)\n" if shap_p_value2 >␣
 ↪0.05 else "Err do NOT follow a normal distribution (p-value <= 0.05)\n")

shap_stat, shap_p_value3 = stats.shapiro(abs_errors3)
print('EfficientNetV2S - Distribution - Shapiro-Wilk Test: Statistic = {:.4f},␣
 ↪p-value = {:.12f}'.format(shap_stat, shap_p_value3))
print("Err follow a normal distribution (p-value > 0.05)\n" if shap_p_value3 >␣
 ↪0.05 else "Err do NOT follow a normal distribution (p-value <= 0.05)\n")

# Configuração da figura com 3 subplots lado a lado
fig, axes = plt.subplots(1, 3, figsize=(18, 5), sharey=True)

# Gráfico para abs_errors1
sns.kdeplot(abs_errors1, ax=axes[0], color="dodgerblue", label="VGG16")
axes[0].set_title('Distribution (VGG16)')
axes[0].set_xlabel('Err (cm), p-value = ' + str(shap_p_value1))
axes[0].set_ylabel('Density')

# Gráfico para abs_errors2
sns.kdeplot(abs_errors2, ax=axes[1], color="green", label="Xception")
axes[1].set_title('Distribution (Xception)')
axes[1].set_xlabel('Err (cm), p-value = ' + str(shap_p_value2))

# Gráfico para abs_errors3
sns.kdeplot(abs_errors3, ax=axes[2], color="orange", label="EfficientNetV2")
axes[2].set_title('Distribution (EfficientNetV2)')
axes[2].set_xlabel('Err (cm), p-value = ' + str(shap_p_value3))

# Ajuste de layout e exibição
plt.tight_layout()
plt.show()
```

```
Normality Analysis (Test Set)

VGG16 - Distribution - Shapiro-Wilk Test: Statistic = 0.8898, p-value =
0.000000000032
Residuals do NOT follow a normal distribution (p-value <= 0.05)

Xception - Distribution - Shapiro-Wilk Test: Statistic = 0.8311, p-value =
0.000000000000
Err do NOT follow a normal distribution (p-value <= 0.05)

EfficientNetV2S - Distribution - Shapiro-Wilk Test: Statistic = 0.8370, p-value
= 0.000000000000
Err do NOT follow a normal distribution (p-value <= 0.05)
```



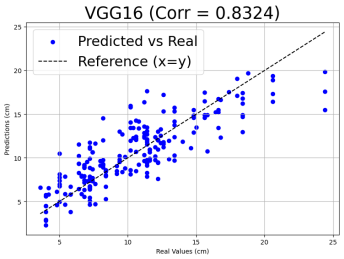
2 Analysis

[ ]:
```
# Desnormalizar predições e valores reais
predictions_vgg16_cm = denormalize(predictions_vgg16, data_min, data_max)
predictions_xception_cm = denormalize(predictions_xception, data_min, data_max)
predictions_efficientnetv2_cm = denormalize(predictions_effv2, data_min,␣
↪data_max)
real_values_cm = denormalize(real_values, data_min, data_max)

# Calcular os resíduos em cm
residuals_vgg16_cm = real_values_cm - predictions_vgg16_cm
residuals_abs_vgg16_cm = np.abs(residuals_vgg16_cm)
residuals_xception_cm = real_values_cm - predictions_xception_cm
residuals_abs_xception_cm = np.abs(residuals_xception_cm)
residuals_efficientnetv2_cm = real_values_cm - predictions_efficientnetv2_cm
residuals_abs_efficientnetv2_cm = np.abs(residuals_efficientnetv2_cm)
```

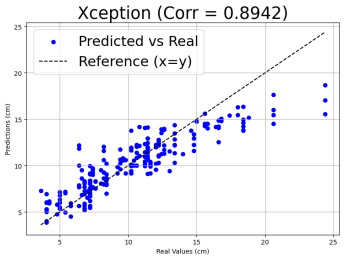
```
# Configuração da figura com 3 subplots lado a lado
fig, axes = plt.subplots(1, 3, figsize=(30, 6), gridspec_kw={'width_ratios':␣
 ↪[1, 1, 1]})

# Métricas para VGG16
mse_vgg16 = mean_squared_error(real_values_cm, predictions_vgg16_cm)
rmse_vgg16 = math.sqrt(mse_vgg16)
mae_vgg16 = np.mean(residuals_abs_vgg16_cm)
mean_residual_vgg16 = np.mean(residuals_vgg16_cm)
std_residual_vgg16 = np.std(residuals_vgg16_cm)
correlation_vgg16 = np.corrcoef(real_values, predictions_vgg16)[0, 1]

plt.rcParams.update({'font.size': 22})  # Aumenta a fonte globalmente

# Gráfico para VGG16
axes[0].scatter(real_values_cm, predictions_vgg16_cm, marker='o', c='blue',␣
 ↪label='Predicted vs Real')
axes[0].plot([real_values_cm.min(), real_values_cm.max()],
             [real_values_cm.min(), real_values_cm.max()],
             color='black', linestyle='--', label='Reference (x=y)')
axes[0].set_title(f'VGG16 (Corr = {correlation_vgg16:.4f})')
axes[0].set_xlabel('Real Values (cm)')
axes[0].set_ylabel('Predictions (cm)')
axes[0].grid(True)
axes[0].legend()
axes[0].text(0, -0.35, f"Model - MSE: {mse_vgg16:.4f} cm²\n"
                       f"Model - RMSE: {rmse_vgg16:.4f} cm\n"
                       f"Model - MAE: {mae_vgg16:.4f} cm\n"
                       f"Residual Mean: {mean_residual_vgg16:.4f} cm\n"
                       f"Residual SD: {std_residual_vgg16:.4f} cm",␣
 ↪fontsize=22, va='center', transform=axes[0].transAxes)


# Métricas para Xception
mse_xception = mean_squared_error(real_values_cm, predictions_xception_cm)
rmse_xception = math.sqrt(mse_xception)
mae_xception = np.mean(residuals_abs_xception_cm)
mean_residual_xception = np.mean(residuals_xception_cm)
std_residual_xception = np.std(residuals_xception_cm)
correlation_xception = np.corrcoef(real_values, predictions_xception)[0, 1]

# Gráfico para xception
axes[1].scatter(real_values_cm, predictions_xception_cm, marker='o', c='blue',␣
 ↪label='Predicted vs Real')
axes[1].plot([real_values_cm.min(), real_values_cm.max()],
             [real_values_cm.min(), real_values_cm.max()],
             color='black', linestyle='--', label='Reference (x=y)')
```

```
axes[1].set_title(f'Xception (Corr = {correlation_xception:.4f})')
axes[1].set_xlabel('Real Values (cm)')
axes[1].set_ylabel('Predictions (cm)')
axes[1].grid(True)
axes[1].legend()
axes[1].text(0, -0.35, f"Model - MSE: {mse_xception:.4f} cm²\n"
                       f"Model - RMSE: {rmse_xception:.4f} cm\n"
                       f"Model - MAE: {mae_xception:.4f} cm\n"
                       f"Residual Mean: {mean_residual_xception:.4f} cm\n"
                       f"Residual SD: {std_residual_xception:.4f} cm",␣
 ↪fontsize=22, va='center', transform=axes[1].transAxes)


# Métricas para EfficientNetV2
mse_efficientnetv2 = mean_squared_error(real_values_cm,␣
 ↪predictions_efficientnetv2_cm)
rmse_efficientnetv2 = math.sqrt(mse_efficientnetv2)
mae_efficientnetv2 = np.mean(residuals_abs_efficientnetv2_cm)
mean_residual_efficientnetv2 = np.mean(residuals_efficientnetv2_cm)
std_residual_efficientnetv2 = np.std(residuals_efficientnetv2_cm)
correlation_efficientnetv2 = np.corrcoef(real_values, predictions_effv2)[0, 1]

# Gráfico para EfficientNetV2
axes[2].scatter(real_values_cm, predictions_efficientnetv2_cm, marker='o',␣
 ↪c='blue', label='Predicted vs Real')
axes[2].plot([real_values_cm.min(), real_values_cm.max()],
             [real_values_cm.min(), real_values_cm.max()],
             color='black', linestyle='--', label='Reference (x=y)')
axes[2].set_title(f'EfficientNetV2 (Corr = {correlation_efficientnetv2:.4f})')
axes[2].set_xlabel('Real Values (cm)')
axes[2].set_ylabel('Predictions (cm)')
axes[2].grid(True)
axes[2].legend()
axes[2].text(0, -0.35, f"Model - MSE: {mse_efficientnetv2:.4f} cm²\n"
                       f"Model - RMSE: {rmse_efficientnetv2:.4f} cm\n"
                       f"Model - MAE: {mae_efficientnetv2:.4f} cm\n"
                       f"Residual Mean: {mean_residual_efficientnetv2:.4f}␣
 ↪cm\n"
                       f"Residual SD: {std_residual_efficientnetv2:.4f} cm",␣
 ↪fontsize=22, va='center', transform=axes[2].transAxes)
```

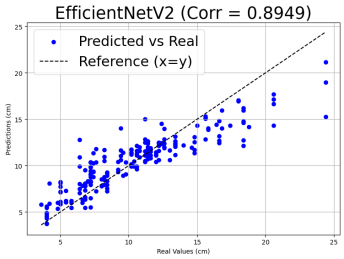
[ ]: Text(0, -0.35, 'Model - MSE: 4.3447 cm²\nModel - RMSE: 2.0844 cm\nModel - MAE: 1.5236 cm\nResidual Mean: -0.1046 cm\nResidual SD: 2.0818 cm')



Model - MSE: 5.7298 cm²
Model - RMSE: 2.3937 cm
Model - MAE: 1.8441 cm
Residual Mean: -0.2090 cm
Residual SD: 2.3846 cm



Model - MSE: 4.2179 cm²
Model - RMSE: 2.0538 cm
Model - MAE: 1.5203 cm
Residual Mean: 0.1198 cm
Residual SD: 2.0503 cm



Model - MSE: 4.3447 cm²
Model - RMSE: 2.0844 cm
Model - MAE: 1.5236 cm
Residual Mean: -0.1046 cm
Residual SD: 2.0818 cm

2.0.1 Test

[ ]:
```
# Calcular a média dos resíduos para cada modelo.
mean_residuals_vgg16 = np.mean(residuals1)
mean_residuals_xception = np.mean(residuals2)
mean_residuals_effic = np.mean(residuals3)

correlation_vgg16 = np.corrcoef(predictions_vgg16, real_values)[0, 1]
correlation_xception = np.corrcoef(predictions_xception, real_values)[0, 1]
correlation_effv2 = np.corrcoef(predictions_effv2, real_values)[0, 1]

# Exibir as médias dos resíduos.
print('-------------------------------------------')
print("Média dos resíduos para VGG16:", mean_residuals_vgg16)
print("Média dos resíduos para Xception:", mean_residuals_xception)
print("Média dos resíduos para EfficientNetV2S:", mean_residuals_effic)

# Aplicar ANOVA.
f_statistic, p_value = stats.f_oneway(abs_errors1, abs_errors2, abs_errors3)

print('-------------------------------------------f_oneway')
print("F-Statistic:", f_statistic)
print("P-Value:", p_value)

f_statistic, p_value = stats.kruskal(abs_errors1, abs_errors2, abs_errors3)

# Resultados da Kruskal.
print('-------------------------------------------kruskal')
print("F-Statistic:", f_statistic)
print("P-Value:", p_value)

print('-------------------------------------------')

# Interpretação do resultado.
```

```
# Preparar os dados para o teste de Tukey.
'''errs = np.concatenate([abs_errors1, abs_errors2, abs_errors3])
labels = ['VGG16'] * len(abs_errors1) + ['Xception'] * len(abs_errors2) +␣
 ↪['EfficientNetV2S'] * len(abs_errors3)

# Aplicar o teste de Tukey.
tukey_result = pairwise_tukeyhsd(errs, labels, alpha=0.05)

# Exibir os resultados do teste de Tukey.
print(tukey_result)'''

data = pd.DataFrame({
  'VGG16': abs_errors1,
  'Xception': abs_errors2,
  'EfficientNetV2S': abs_errors3
})

# Aplicar o teste de Nemenyi.
nemenyi_result = posthoc_nemenyi_friedman(data)

# Adicionar asteriscos para significância.
def format_with_stars(value):
    if value < 0.001:
        return f"{value:.4f}***"
    elif value < 0.01:
        return f"{value:.4f}**"
    elif value < 0.05:
        return f"{value:.4f}*"
    else:
        return f"{value:.4f}"

nemenyi_with_stars = nemenyi_result.map(format_with_stars)

if p_value < 0.05:
  print("\nHá uma diferença estatisticamente significativa entre as médias dos␣
 ↪|resíduos| dos três modelos para esta vegetação.")
else:
  print("\nNão há uma diferença estatisticamente significativa entre as médias␣
 ↪dos |resíduos| dos três modelos para esta vegetação!!!!!!!!!")

# Exibir a tabela estilizada com asteriscos
print("\nResultados do Teste de Nemenyi (com Asteriscos):\n")
styled_table = nemenyi_with_stars.style.set_properties(**{'text-align':␣
 ↪'center'}).set_caption("Tabela de Resultados do Teste de Nemenyi")
display(styled_table)
```

```
-------------------------------------------
Média dos resíduos para VGG16: -0.0056781267625027314
Média dos resíduos para Xception: 0.0032562901262804533
Média dos resíduos para EfficientNetV2S: -0.0028428339579373786
-------------------------------------------f_oneway
F-Statistic: 3.4326566997217944
P-Value: 0.03291519256922311
-------------------------------------------kruskal
F-Statistic: 7.647855917159859
P-Value: 0.021841838339801998
-------------------------------------------

Há uma diferença estatisticamente significativa entre as médias dos |resíduos|
dos três modelos para esta vegetação.

Resultados do Teste de Nemenyi (com Asteriscos):


<pandas.io.formats.style.Styler at 0x7d9539e1b9d0>
```

2.1 Heatmap

```
[ ]: def overlay_heatmap_on_image(image, heatmap, alpha=0.5, colormap='viridis'):
        """
        Sobrepõe um heatmap em uma imagem com controle de transparência.

        Args:
            image (numpy.ndarray or tensorflow.Tensor): A imagem original com␣
    ↪valores normalizados (0 a 1).
            heatmap (numpy.ndarray or tensorflow.Tensor): O heatmap (2D) gerado,␣
    ↪normalizado entre 0 e 1.
            alpha (float): Transparência do heatmap sobreposto. Valores entre 0␣
    ↪(transparente) e 1 (opaco).
            colormap (str): Nome do mapa de cores para o heatmap. Ex: 'viridis',␣
    ↪'jet'.

        Returns:
            numpy.ndarray: A imagem sobreposta com o heatmap.
        """
        # Converte o heatmap para uma imagem RGB usando o colormap.
        heatmap_colored = plt.cm.get_cmap(colormap)(heatmap)[:, :, :3]  # Remove o␣
    ↪canal alpha do colormap.

        # Certifica-se de que a imagem está normalizada entre 0 e 1.
        if isinstance(image, tf.Tensor):
            image = image.numpy()
        if isinstance(heatmap_colored, tf.Tensor):
```

```
        heatmap_colored = heatmap_colored.numpy()

    image = np.clip(image, 0, 1)  # Normaliza os valores da imagem.
    heatmap_colored = np.clip(heatmap_colored, 0, 1)  # Normaliza o heatmap.

    # Redimensiona o heatmap para corresponder ao tamanho da imagem.
    heatmap_resized = tf.image.resize(heatmap_colored, (image.shape[0], image.
↪shape[1])).numpy()

    # Combina a imagem e o heatmap com o fator alpha.
    overlayed_image = (1 - alpha) * image + alpha * heatmap_resized
    overlayed_image = np.clip(overlayed_image, 0, 1)  # Garante que os valores␣
↪estão no intervalo válido.

    return overlayed_image
```

[ ]:
```
print(best_model_effv2.layers[0].output)
```

```
<KerasTensor shape=(None, 12, 12, 1280), dtype=float32, sparse=False,
name=keras_tensor_512>
```

[ ]:
```
def make_gradcam_heatmap(img_array, model, last_conv_layer_name,␣
↪pred_index=None):

    grad_model = keras.models.Model(
      inputs=model.input,
      outputs=[
          model.get_layer(last_conv_layer_name).output,  # Saída da camada␣
↪convolucional.
          model.output  # Saída final.
      ]
    )

    with tf.GradientTape() as tape:
      last_conv_layer_output, loss = grad_model(img_array)
      loss = loss[0]
      print('loss == ', loss)

    grads = tape.gradient(loss, last_conv_layer_output)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = tf.reduce_sum(pooled_grads * last_conv_layer_output, axis=-1)

    # Normalização
    heatmap = tf.maximum(heatmap, 0) / tf.reduce_max(heatmap)
    return heatmap.numpy()
```

```
current_cnn = vgg16_folder
model_builder = func_model_vgg16
img_size = (360, 360)

#last_conv_layer_name = "block14_sepconv2_act" # xception
#last_conv_layer_name = "top_conv" #'stem_conv' #"top_conv" $effv2
last_conv_layer_name = "block5_conv3" #vgg

data_gen = DataGenerator(get_test_folder(vegetation_name),␣
 ↪get_experiment_folder(vegetation_name), batch_size)

real_values, images = data_gen.get_real_values_and_images()

remove_and_recreate_directory(get_heatmap_folder(current_cnn))

for idx in range(0, len(images)):
  print(f'{idx+1}/{len(images)}')
  real_values, images = data_gen.get_real_values_and_images()
  img = images[-idx]
  img_array = img
  img_array = tf.expand_dims(img, axis=0)  # Expande a dimensão para (1,␣
 ↪altura, largura, canais)
  heatmap = make_gradcam_heatmap(img_array, model_builder, last_conv_layer_name)

  combined_img = overlay_heatmap_on_image(img, heatmap, alpha=0.5)

  #plt.imshow(combined_img)
  #plt.show()

  # Salva a imagem do heatmap no diretório.
  file_name = f"heatmap_{idx}.png"
  file_path = os.path.join(get_heatmap_folder(current_cnn), file_name)

  plt.figure()
  plt.imshow(combined_img)
  plt.axis('off')  # Remove os eixos para uma visualização limpa
  plt.savefig(file_path, bbox_inches='tight', pad_inches=0)
  plt.close()  # Fecha a figura para evitar sobrecarregar a memória
```

APÊNDICE E – CLASSE PYTHON - DATAGENERATOR

Este apêndice apresenta o código-fonte completo do *data generator* desenvolvido para o treinamento das redes neurais. Esse componente foi utilizado para alimentar as redes durante os experimentos.

```python
import tensorflow as tf
import math
import os
import json
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import random
import cv2

class DataGenerator(tf.keras.utils.Sequence):

    def __init__(self, data_path, experiment_data_path, batch_size, image_size = 720, resize = True, resize_size = 360):
        self._resize = resize
        self._resize_size = resize_size
        self._image_size = image_size

        self.data_min = 0
        self.data_max = 0

        self.data_path = data_path
        self.experiment_data_path = experiment_data_path
        self.batch_size = batch_size

        files_list = os.listdir(data_path)
        files_list.sort(key=lambda x: x.rjust(20, '0'))

        self.images_files = files_list
        self.dynamic_images_files = files_list.copy()

        self.__load_resumes()

    def __len__(self):
        return math.ceil(len(self.images_files) / self.batch_size)

    def on_epoch_end(self):
        random.shuffle(self.dynamic_images_files)

    def __getitem__(self, idx):
        batch_images_files = self.dynamic_images_files[idx * self.batch_size:(idx + 1) * self.batch_size]

        batch_x = np.empty(shape=(len(batch_images_files), self._image_size, self._image_size, 3))
        batch_y = []

        if(self._resize):
            batch_x = np.empty(shape=(len(batch_images_files), self._resize_size, self._resize_size, 3))

        counter = 0

        for image_path in batch_images_files:
```

```
50              try:
51                  x, y = self.__get_x_y(image_path)
52                  batch_x[counter] = x
53                  batch_y.append(y)
54                  counter = counter + 1
55              except:
56                  print('Image path error: ' + image_path)
57                  raise
58
59          return batch_x, np.asarray(batch_y).astype('float32')
60
61      def __get_x_y(self, image_file):
62          '''
63          Retorna x e y, sendo:
64              x = a imagem como uma numpy array.
65              y = altura média do pasto no pasto na imagem.
66          '''
67          id = self.__get_file_id(image_file)
68
69          # Usar OpenCV para carregar e converter a imagem
70          image = cv2.imread(os.path.join(self.data_path, image_file))
71          image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Converter para
               RGB se necessário
72          image = self.resize(image)
73
74          x = image.astype('float32') / 255
75
76          resume = self.__find_resume(id)
77          y = float(resume['Height'])
78
79          return x, y
80
81      def get_images_files_names(self):
82          '''
83              Retorna uma tupla (files_names, ids), onde:
84              files_names = Nomes de todos os arquivos em data_path.
85              ids = Ids de cada arquivo em data_path.
86          '''
87          files = []
88          ids = []
89
90          for file in self.images_files:
91              files.append(file)
92              ids.append(self.__get_file_id(file))
93
94          return files, ids
95
96      def get_images(self, files):
97          '''
98              files = Nomes do arquivos de imagens para retornar.
99              Retorna uma ou mais nparray das imagens.
100         '''
101         batch_images = np.empty(shape=(len(files), self._image_size,
```

```
            self._image_size, 3))
        counter = 0

        if(self._resize):
            batch_images = np.empty(shape=(len(files),
              self._resize_size, self._resize_size, 3))

        for file in files:
            image = cv2.imread(os.path.join(self.data_path, file))
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Converter
              para RGB se necessário.
            image = self.resize(image)
            image = tf.keras.utils.img_to_array(image).astype
              ('float32') / 255
            batch_images[counter] = image
            counter = counter + 1

        return batch_images

    def resize(self, image):
        if(not self._resize):
            return image

        # Usar cv2.resize para redimensionar a imagem do OpenCV.
        return cv2.resize(image, (self._resize_size,
          self._resize_size))

    def get_y(self, ids=None):
        y = []

        for file in self.images_files:
            id = self.__get_file_id(file)

            if(ids is None or id in ids):
                y.append(self.__get_just_y(id))

        return np.asarray(y).astype('float32')

    def __get_just_y(self, id):
        return self.__find_resume(id)['Height']

    def __load_resumes(self):
        '''
        Carrega os resumos do experimento.
        Cada resumo contém informações sobre determinada imagem, tais
          como:
        distância da foto tirada e média da altura na foto.
        '''
        experiment = ''
        with open(os.path.join(self.experiment_data_path,
          'experiment.json'), 'r') as f:
            experiment = f.read()

```

```
148         resumes = json.loads(experiment)
149         heights = []
150         ids = []
151 
152         for resume in resumes:
153             heights.append(float(resume['Mean']))
154             ids.append(resume['Id'])
155 
156         scaler = MinMaxScaler()
157         heights = np.array(heights).reshape(-1, 1)
158         heights = scaler.fit_transform(heights)
159 
160         self.data_min = scaler.data_min_[0]
161         self.data_max = scaler.data_max_[0]
162 
163         self.experiment_resumes = []
164 
165         for idx in range(0, len(ids)):
166             self.experiment_resumes.append({'Id': ids[idx], 'Height':
                  heights[idx][0]})
167 
168     def __find_resume(self, id):
169         return list(filter(lambda resume: int(resume['Id']) == int(id),
                 self.experiment_resumes))[0]
170 
171     def __get_file_id(self, file_name):
172         return file_name.split('_')[0]
173 
174     def get_real_values(self):
175         '''
176         Retorna os valores reais (alturas) para as imagens na ordem em
              que são carregadas.
177         Lê o ID de cada imagem e busca o valor correspondente no
              experimento.
178         '''
179         real_values = []
180 
181         # Para cada foto no dataset (self.files_list), buscar o valor
              correto no experimento.
182         for image_file in self.images_files:
183             # Extrair o ID da imagem (parte antes do "_").
184             image_id_str = image_file.split('_')[0]
185 
186             # Encontrar o resume que corresponde ao ID da imagem.
187             resume = next((resume for resume in self.experiment_resumes
                    if int(resume['Id']) == int(image_id_str)), None)
188 
189             if resume:
190                 real_values.append(float(resume['Height']))
191             else:
192                 raise ValueError(f"Resume com ID {image_id_str} não
                      encontrado.")
193 
```

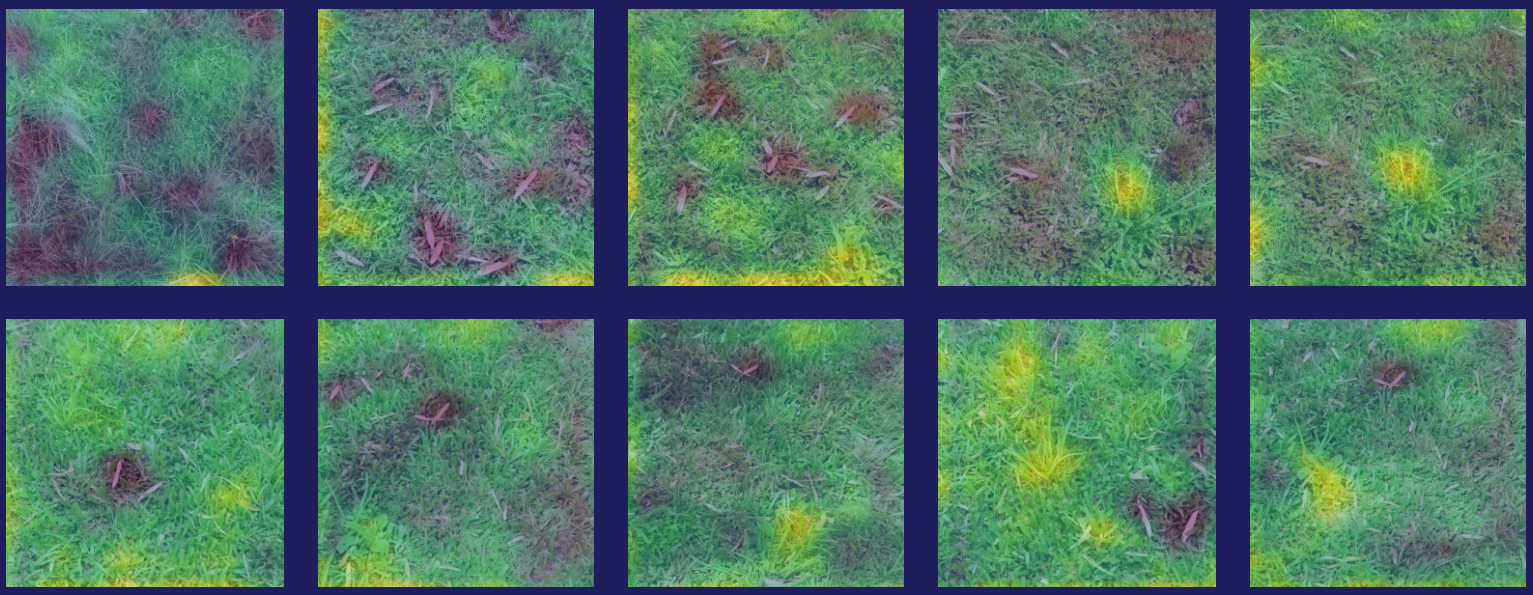
```
194         return real_values
195
196     def get_real_values_and_images(self):
197         '''
198         Retorna os valores reais (alturas) e as imagens na ordem em que são carregadas.
199         '''
200         real_values = []
201         images = []
202
203         for file in self.images_files:
204             image_id = int(file.split('_')[0])  # Extrai o ID da imagem do nome do arquivo.
205             # Encontrar o valor real correspondente ao ID da imagem.
206             height = next((resume['Height'] for resume in self.experiment_resumes if resume['Id'] == image_id), None)
207
208             if height is not None:
209                 real_values.append(height)
210                 images.append(self.get_images([file])[0])
211
212         return real_values, images
```

APÊNDICE F – *HEATMAPS* DAS ARQUITETURAS DESENVOLVIDAS POR TIPO DE VEGETAÇÃO

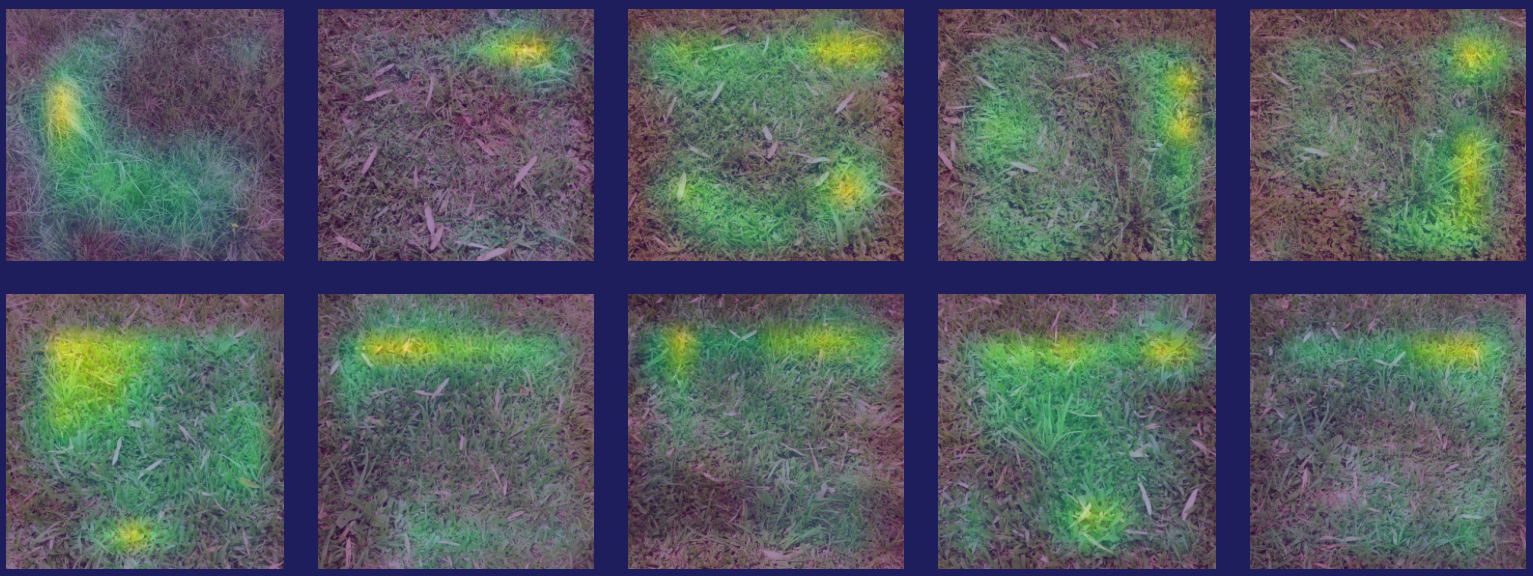
Este apêndice apresenta os *heatmaps* gerados pelas redes baseadas em VGG16, Xception e EfficientNetV2S para as vegetações nativa, azevém e capim-sudão. Os *heatmaps* foram obtidos utilizando o método Grad-CAM (Gradient-weighted Class Activation Mapping), que identifica as regiões mais relevantes das imagens de entrada para as estimativas realizadas pelas redes.

NATIVO

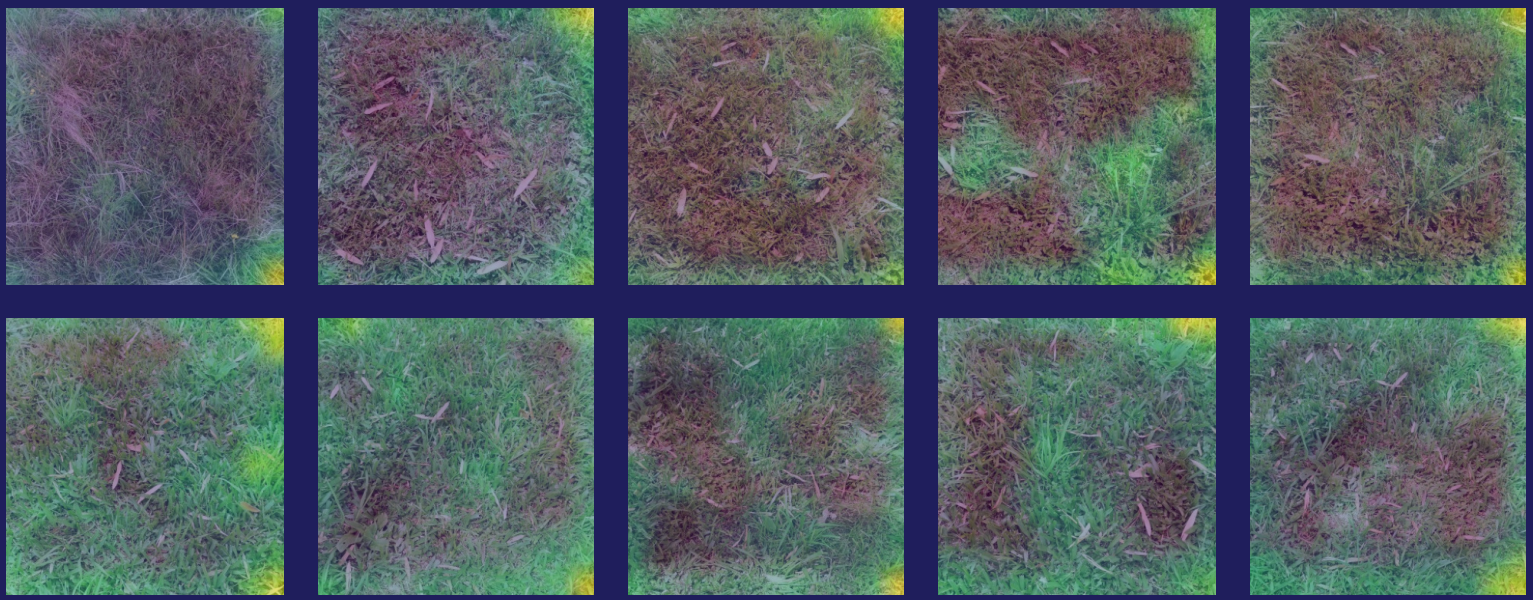
VGG16



Xception

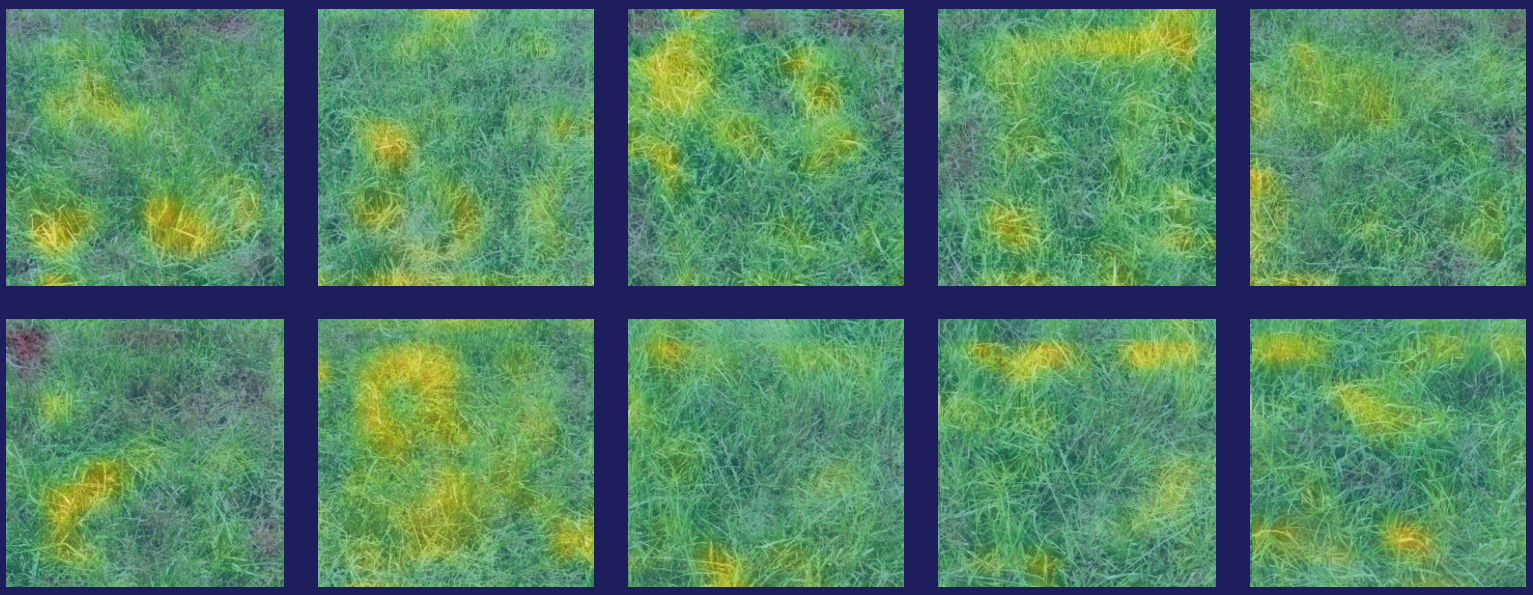


EfficientNetV2

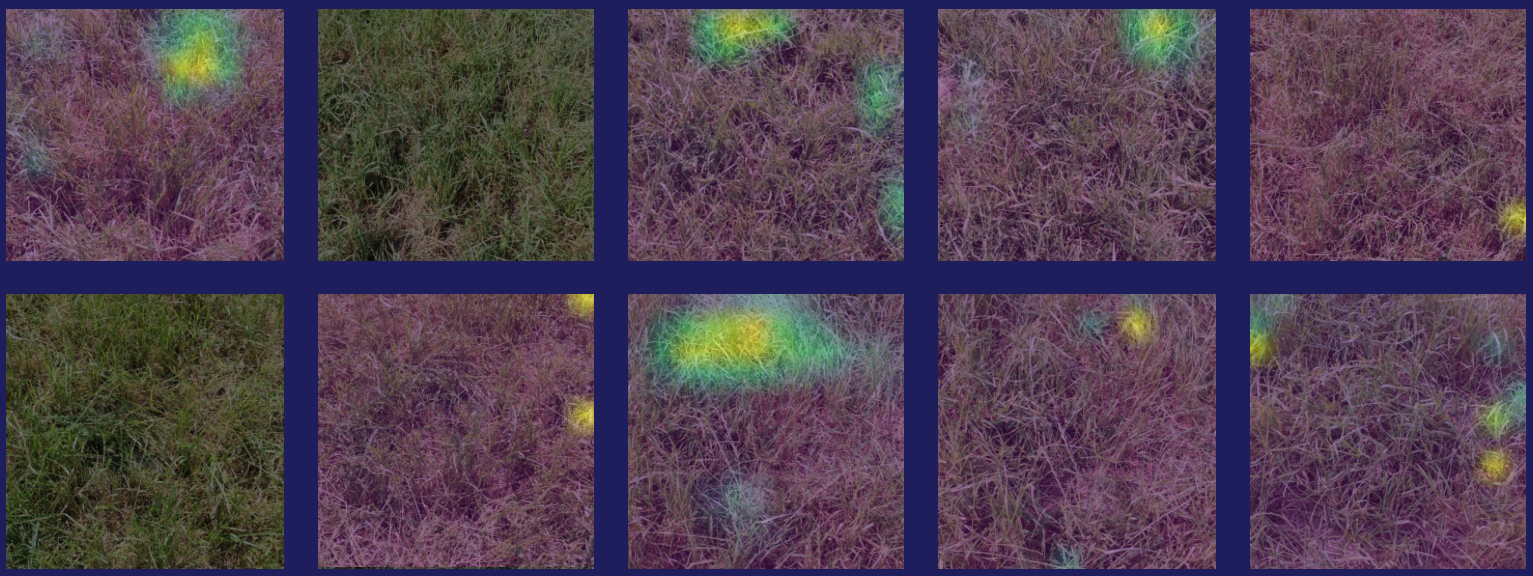


AZEVÉM

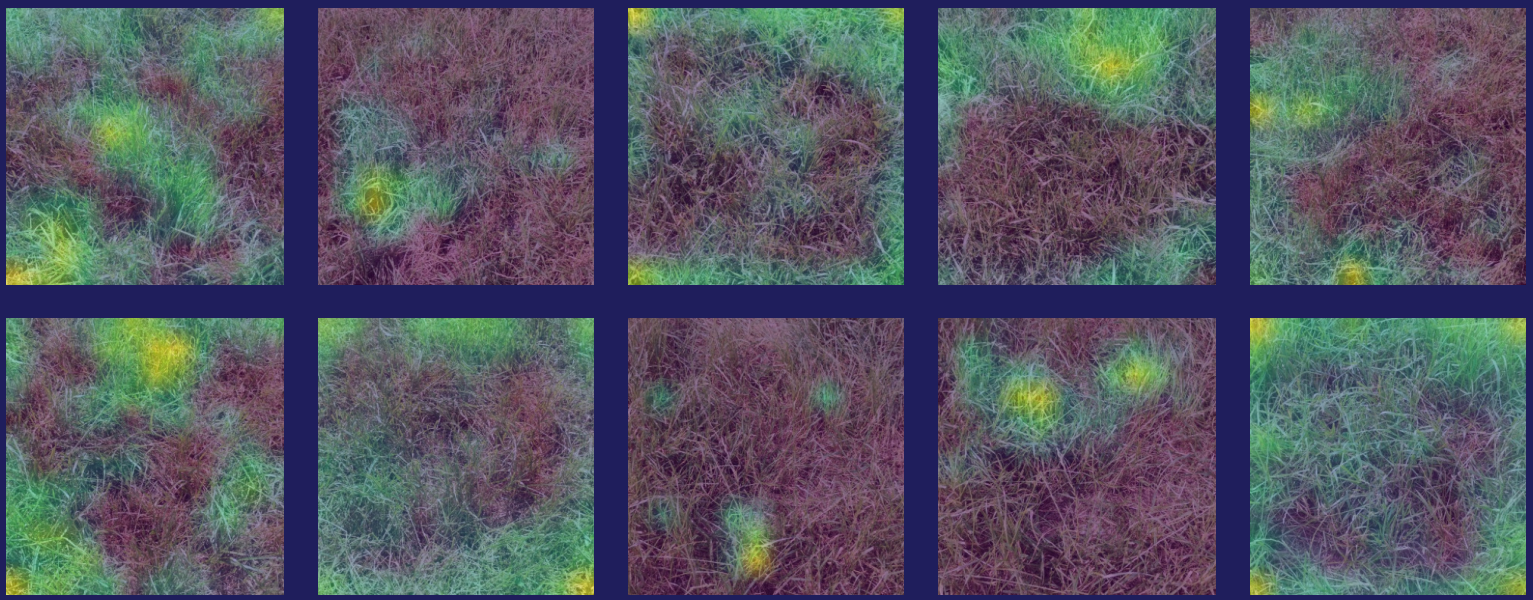
VGG16



Xception



EfficientNetV2



CAPIM-SUDÃO

VGG16

Xception

EfficientNetV2