UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Marcus Augusto Demétrio Pinheiro

Desenvolvimento de um modelo de classificação de imagens de peixes de Florianópolis usando *Deep Learning*

Florianópolis
2025

Marcus Augusto Demétrio Pinheiro

Desenvolvimento de um modelo de classificação de imagens de peixes de Florianópolis usando *Deep Learning*

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Alexandre Gonçalves Silva

Florianópolis
2025

RESUMO

Diante do progressivo desaparecimento do saber-fazer da pesca artesanal em Florianópolis, Santa Catarina, este trabalho teve como objetivo geral desenvolver um modelo de *Deep Learning* (DL) voltado à classificação de imagens de peixes e outros animais marinhos, visando a documentação e difusão deste patrimônio cultural. Para tanto, estruturou-se um *dataset* próprio com 963 imagens de oito espécies de relevância local (incluindo peixes e crustáceos), compilado a partir de fontes públicas e coleta primária. A metodologia empregou a técnica de *Transfer Learning* para implementar e comparar duas arquiteturas de Redes Neurais Convolucionais (CNNs): *MobileNetV2* e *ResNet50V2*. O desenvolvimento, conduzido no ambiente *Google Colab*, incluiu pré-processamento, aumento de dados (*Data Augmentation*) e quatro experimentos principais variando hiperparâmetros. Os resultados demonstraram a alta viabilidade da abordagem, com acurácias superiores a 95% em todos os testes. Conclui-se que, apesar do desempenho similar na classificação, o *MobileNetV2* é significativamente mais leve que o *ResNet50V2*, consolidando-se como a arquitetura mais eficiente e ideal para uma futura aplicação móvel.

Palavras-chave: Classificação de Imagens; Aprendizado Profundo; MobileNet; Pesca Artesanal; Redes Neurais Convolucionais; ResNet; Transfer Learning.

ABSTRACT

Given the progressive disappearance of the traditional knowledge (saber-fazer) of artisanal fishing in Florianópolis, Santa Catarina, this work's general objective was to develop a Deep Learning (DL) model focused on the classification of images of fish and other marine animals, aiming to document and disseminate this cultural heritage. To this end, a proprietary dataset was structured with 963 images of eight locally relevant species (including fish and crustacean), compiled from public sources and primary data collection. The methodology employed the Transfer Learning technique to implement and compare two Convolutional Neural Network (CNN) architectures: *MobileNetV2* and *ResNet50V2*. The development, conducted in the Google Colab environment, included preprocessing, Data Augmentation, and four main experiments varying hyperparameters. The results demonstrated the high viability of the approach, with accuracies exceeding 95% in all tests. It is concluded that, despite similar classification performance, *MobileNetV2* is significantly lighter than *ResNet50V2*, establishing itself as the most efficient and ideal architecture for a future mobile application.

Keywords: Image Classification; Deep Learning; MobileNet; Artisanal Fishing; Convolutional Neural Networks; ResNet; Transfer Learning.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A prática da pesca acompanha a humanidade desde os tempos pré-históricos, desempenhando papel crucial na subsistência de grupos ancestrais. Mais do que uma simples atividade extrativa, representa um saber construído pela interação direta com a natureza, na qual peixes, marés e correntes são elementos fundamentais para a compreensão e apropriação do ambiente (Cardoso, 2001). Esse conhecimento, transmitido entre gerações, envolve o uso de técnicas e a construção de sistemas simbólicos e cognitivos que moldam a relação do pescador com seu entorno natural e social. Dessa forma, a pesca ultrapassa a esfera econômica, configurando-se como um processo cultural e de conhecimento prático, enraizado na experiência cotidiana (Cardoso, 2001).

No litoral de Santa Catarina, a pesca ocupou historicamente uma função central na subsistência e no desenvolvimento regional. A presença de sambaquis e sítios arqueológicos ao longo da costa atesta a exploração de abundantes recursos marinhos desde o período pré-histórico, muito antes da colonização europeia (Destéfani, 2017; Medeiros, 2001). Por volta do ano 1400, a porção insular de Florianópolis, a Ilha de Santa Catarina, foi habitada pelos indígenas carijós, que adaptaram suas técnicas de navegação e pesca para subsistir no ambiente marinho complementando-as com a coleta de moluscos, a caça e a agricultura (Pinho, 2016).

Com a chegada dos colonizadores portugueses no século XVIII, a ilha adquiriu uma organização econômica mais estruturada, baseada na agricultura e na pesca, com especial enfoque na pesca da baleia (Pinho, 2016). Conforme o autor, essa atividade ganhou grande destaque, não apenas como fonte de alimento, mas também como atividade econômica essencial para a capitania de Santa Catarina. Assim, a pesca deixou de ser apenas de subsistência, tornando-se uma atividade lucrativa que despertou interesses econômicos e industriais. No entanto, ao final do século, a pesca da baleia entrou em declínio na costa catarinense, levando ao fechamento das armações baleeiras (Pinho, 2016).

Em contrapartida ao declínio, a pesca artesanal começou a ganhar maior destaque. Esta modalidade, desenvolvida desde a chegada dos imigrantes açorianos e praticada em comunidades e núcleos familiares, persistiu por muitos anos como atividade de subsistência complementar à agricultura familiar, consolidando sua importância na região (Medeiros, 2001; Pinho, 2016). Essa

articulação entre a pequena agricultura e a pesca configurou um modo de vida e cultura que se manteve até meados do século XX, quando passou a ser inviabilizado por pressões crescentes (Pinho, 2016).

Dentre essas pressões, destaca-se a expansão imobiliária e o avanço da pesca industrial, que geraram um cenário de concorrência predatória. Tais fatores, somados ao desenvolvimento urbano e à transformação socioeconômica da região, levaram a pesca a um processo de declínio que persiste até os dias atuais, impactando significativamente as condições e os modos de vida de comunidades tradicionais (Pinho, 2016).

Com o declínio da prática, o conhecimento antes transmitido entre gerações vem se perdendo devido à falta de incentivo comercial, à rápida urbanização e à supervalorização das áreas tradicionalmente utilizadas por pescadores. Perde-se, assim, um saber que englobava uma grande diversidade de técnicas e estratégias de pesca, construído a partir do conhecimento etnoecológico e resultante tanto da grande variedade de ambientes quanto da combinação entre as culturas indígenas e portuguesas no litoral de Santa Catarina (Medeiros, 2001).

Em entrevista ao Portal Catarinas (2021), Morgani Guzzo relata:

> Por meio da história oral, temos a possibilidade de registrar esses saberes e fazeres que estão se perdendo pelo avanço da urbanização e pelo desinteresse das novas gerações pela pesca artesanal. O conhecimento sobre o território, o mar, os ventos, os tipos de peixe da região foi passado de geração para geração. Mas, hoje, a maior parte dos pescadores tradicionais são idosos e é comum que seus filhos e filhas não mantenham a tradição dos pais e avós. Assim, acreditamos que esses saberes são fundamentais não só para a preservação da memória cultural de Florianópolis, mas também para a preservação ambiental da Ilha.

Diante do contexto de enfraquecimento da pesca artesanal e do progressivo desaparecimento do conhecimento associado, torna-se premente o desenvolvimento de ações que busquem valorizar e preservar essa cultura. Como contraponto a essa tendência, o presente trabalho propõe o desenvolvimento de um algoritmo de reconhecimento de imagens, focado em identificar as espécies de peixes mais comumente pescadas e conhecidas em Florianópolis. Este modelo computacional é apresentado como uma ferramenta de documentação e difusão do saber local, buscando preservar a história, valorizar as práticas e, sobretudo, reconectar a população (especialmente as novas gerações) a esse patrimônio cultural.

Para a execução desta tarefa, optou-se pela aplicação de técnicas de *Deep Learning* (DL), com o desenvolvimento e treinamento de *uma Convolutional Neural*

Network (CNN), metodologia que tem demonstrado eficácia no reconhecimento de padrões visuais. Modelos de CNN têm sido utilizados em diversos estudos para a identificação de animais, como o de Amorim (2022), por exemplo, voltado para a classificação de espécies de peixe. Contudo, trabalhos nessa área utilizam, frequentemente, conjuntos de dados genéricos, que agregam espécies de diversas regiões do mundo.

Neste contexto, o diferencial do presente trabalho reside na proposição de um modelo focado exclusivamente nos peixes da região de Florianópolis, espaço de grande importância histórica e cultural para a pesca. Ao restringir o escopo ao ecossistema marinho local, busca-se não apenas alcançar uma maior acurácia técnica na identificação de espécies catarinenses, mas também criar uma ferramenta de valorização cultural e um instrumento didático que auxilie na preservação da memória pesqueira da Ilha de Santa Catarina.

1.1 OBJETIVOS

Neste sentido, o objetivo geral do presente trabalho é desenvolver um modelo de *Deep Learning* voltado à classificação de imagens de peixes, com foco na identificação e diferenciação das espécies mais comuns na região de Florianópolis.

Para alcançar o objetivo geral, são propostos os seguintes objetivos específicos:

1. identificar técnicas para a criação de um modelo de classificação de imagens de peixes utilizando *Deep Learning*;
2. estruturar um conjunto de dados de imagens das espécies de peixes de interesse, unindo bases públicas e imagens próprias, e aplicar técnicas de pré-processamento para garantir a qualidade e consistência dos dados;
3. implementar e treinar modelos de *Deep Learning*, como Redes Neurais Convolucionais (CNNs), explorando a abordagem de *Transfer Learning* para otimizar o treinamento e a classificação das imagens;
4. realizar o ajuste dos hiperparâmetros (*hyperparameter tuning*) dos modelos implementados, visando maximizar o desempenho e a eficiência computacional;

5. avaliar o desempenho dos modelos por meio de métricas de classificação (como acurácia, precisão, *recall* e *F1-score*), comparando os resultados para determinar a abordagem mais eficiente;
6. validar o modelo final com um conjunto de testes, garantindo sua capacidade de generalização e acurácia na identificação das espécies;
7. disponibilizar o modelo final para contribuir na divulgação e preservação do conhecimento tradicional sobre a pesca local.

1.2 JUSTIFICATIVA

Este trabalho justifica-se por um conjunto de fatores que tangenciam a urgência na preservação de um patrimônio cultural imaterial e o potencial impacto social e educacional da solução proposta. A motivação central é a valorização e a divulgação da cultura pesqueira e da biodiversidade marinha da Ilha de Santa Catarina. Trata-se de uma região historicamente marcada pela riqueza de sua fauna, onde a pesca se consolidou como prática de subsistência, econômica e cultural ao longo de mais de dois séculos (Destéfani, 2017; Medeiros, 2001; Pinho, 2016).

A justificativa apoia-se, também, no notório e acelerado processo de desaparecimento do saber-fazer da pesca artesanal em Florianópolis, fator que torna urgente a documentação de um saber que é parte fundamental da identidade cultural e da história local. Como contraponto a essa perda, esta pesquisa propõe uma forma de registro e difusão digital desse conhecimento. Neste sentido, o modelo proposto, ao possibilitar a identificação precisa das espécies locais, poderá contribuir como uma ferramenta educativa e tecnológica, promovendo o conhecimento sobre a fauna local para a população em geral, turistas, pesquisadores e entusiastas da pesca.

Do ponto de vista técnico-científico, a justificativa reside na aplicação de uma metodologia avançada de visão computacional a um problema local específico. A escolha do *Deep Learning* deve-se ao seu comprovado potencial em resolver problemas complexos de reconhecimento de padrões com alta precisão. Conforme LeCun, Bengio e Hinton (2015), este ramo de aprendizado de máquina está fazendo grandes avanços na resolução de problemas, em diferentes domínios da ciência, superando outras técnicas de aprendizagem no reconhecimento de imagens. Contudo, enquanto muitos estudos aplicam Redes Neurais Convolucionais a

conjuntos de dados genéricos, este trabalho diferencia-se por direcionar essa tecnologia para a criação de um modelo focado na fauna de Florianópolis - SC.

1.3 METODOLOGIA DE PESQUISA

Quanto à abordagem, esta é uma pesquisa de natureza mista, que articula abordagens quantitativas e qualitativas. A pesquisa iniciou-se com uma revisão bibliográfica, fundamentada em leituras exploratórias sobre o tema, a fim de conhecer e analisar as principais contribuições teóricas existentes (Koche, 2015). Nesta etapa, foram identificados os trabalhos mais relevantes na área de classificação de imagens e as técnicas de *Deep Learning* em uso atualmente. A análise focou-se na compreensão do desenvolvimento de modelos para visão computacional, com ênfase nos métodos de Redes Neurais Convolucionais e na técnica de *Transfer Learning*, além de outras arquiteturas modernas aplicáveis ao problema.

Com base nesse levantamento, a análise foi aprofundada para identificar como essas técnicas vêm sendo utilizadas e adaptadas em diferentes contextos, com especial atenção às abordagens que poderiam ser aplicadas para o reconhecimento de espécies de peixes. Este aprofundamento investigou a escolha das arquiteturas, estratégias de pré-processamento dos dados e técnicas de ajuste de parâmetros (*hyperparameter tuning*) empregadas para melhorar o desempenho do modelo.

Na sequência, superada a etapa de fundamentação teórica, foi conduzida uma análise quantitativa para avaliar o desempenho dos modelos desenvolvidos. O objetivo dessa etapa foi extrair métricas de performance (como acurácia, precisão, *recall* e *F1-score*) desses modelos, permitindo a comparação de seus resultados, a identificação de pontos fortes e fracos, e a determinação do modelo mais adequado ao escopo desta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica que sustenta o desenvolvimento do modelo proposto. A exposição segue uma estrutura que parte de conceitos mais amplos da Inteligência Artificial (IA) até as arquiteturas específicas empregadas neste projeto. Inicialmente, define-se o Aprendizado de Máquina (*Machine Learning*) e seus paradigmas, com foco no aprendizado supervisionado, que norteia este trabalho. Na sequência, são detalhados os fundamentos das Redes Neurais Artificiais (*Neural Networks*), o mecanismo que simula o aprendizado. Aprofunda-se, então, o conceito de Aprendizado Profundo (*Deep Learning*) e, subsequentemente, das Redes Neurais Convolucionais (CNNs), que constituem a metodologia central para o processamento e classificação de imagens. Por fim, o capítulo detalha arquiteturas de CNNs (como *ResNet* e *MobileNet*) que informam o estado da arte e servem como base de referência para a implementação do modelo.

2.1 APRENDIZADO DE MÁQUINA (*MACHINE LEARNING*)

A Inteligência Artificial (IA) é um campo abrangente da ciência da computação dedicado à criação de sistemas que simulam a inteligência e o comportamento humano para realizar tarefas complexas (Ludermir, 2021). Dentro deste vasto domínio, o Aprendizado de Máquina (*Machine Learning*) emerge como uma subárea que utiliza algoritmos estatísticos e matemáticos para que um sistema aprenda autonomamente (Bergmann, 2025; Shinde; Shah, 2018). Para isto, o computador “aprende” por exemplos, ou seja, por meio da análise e identificação de padrões nos dados de treinamento (Ludermir, 2021). Subsequentemente, essa capacidade de reconhecimento de padrões possibilita que os modelos de ML façam inferências precisas ou tomem decisões sobre novos dados, sem necessitar de instruções explícitas e predefinidas (Bergmann, 2025).

Os algoritmos de ML são categorizados principalmente com base no tipo de *feedback* que recebem durante o processo de treinamento. As principais categorias de aprendizado são: supervisionado (*supervised*), não supervisionado (*unsupervised*) e por reforço (*reinforcement learning*). Neste contexto, Bergmann (2025) conceitua o aprendizado supervisionado como aquele em que se utiliza de dados rotulados para treinar modelos de classificação e previsão, otimizando-os

para corresponder aos exemplos fornecidos; enquanto o não supervisionado busca padrões ocultos em dados não rotulados. Já no aprendizado por reforço o modelo aprende por tentativa e erro, otimizando seus parâmetros e buscando a melhor estratégia ao longo do tempo (Bergmann, 2025).

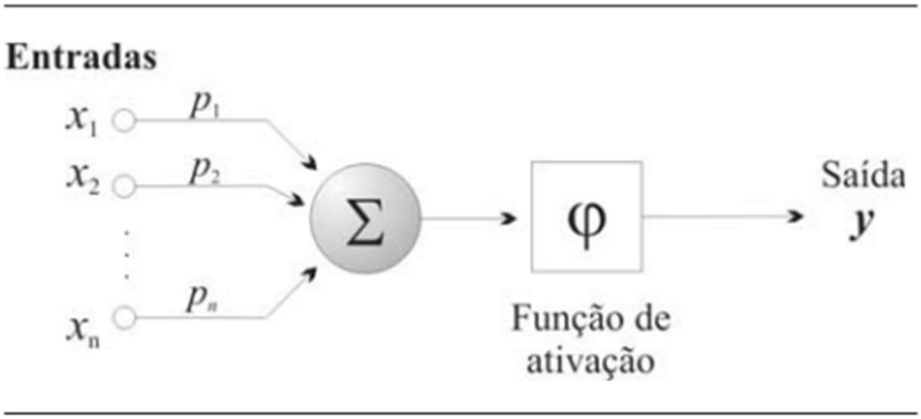
O objetivo do presente trabalho, de identificar e diferenciar as espécies, enquadra o projeto na categoria de aprendizado supervisionado. Esta abordagem metodológica exige que, para cada exemplo apresentado ao algoritmo, seja fornecida a resposta desejada, isto é, um rótulo informando a classe à qual o exemplo pertence (Ludermir, 2021). Portanto, esta categoria é a adequada, visto que a tarefa de classificação de imagens depende de um conjunto de dados onde as entradas (as imagens dos peixes de Florianópolis) já estão associadas a um rótulo de saída correto (o nome da espécie).

2.2 REDES NEURAIS (*NEURAL NETWORKS*)

O aprendizado supervisionado, especialmente em tarefas complexas, é frequentemente implementado através de Redes Neurais Artificiais (RNAs). As RNAs são uma técnica de ML bem-sucedida, que consiste em modelos matemáticos que buscam simular a estrutura biológica e o processamento de informação do cérebro humano (Ferneda, 2006; Ludermir, 2021). Conforme definido por Ferneda (2006, p. 26), elas “são compostas por unidades de processamentos simples, os neurônios, que se unem por meio de conexões sinápticas”.

Sendo assim, as redes neurais são compostas por camadas interconectadas de neurônios (Bergmann, 2025). Matematicamente, um neurônio (conforme a representação visual na Figura 1) é composto por três elementos básicos: conexões de entrada (x1, x2, …, xn) associadas a pesos (p1, p2, …, pn), um somador (Σ) que acumula os sinais ponderados de entrada, e uma função de ativação (φ) que processa a saída (y) (Ferneda, 2006). Os pesos das conexões, positivos ou negativos, determinam se a conexão é excitatória ou inibitória, afetando a intensidade do sinal resultante. A saída é calculada justamente por essa soma ponderada dos sinais, que por sua vez passa pela função de ativação (Ferneda 2006). É a partir do aprendizado – especificamente, do ajuste contínuo desses pesos – que esses modelos adquirem sua capacidade computacional para a resolução de problemas (Ludermir, 2021).

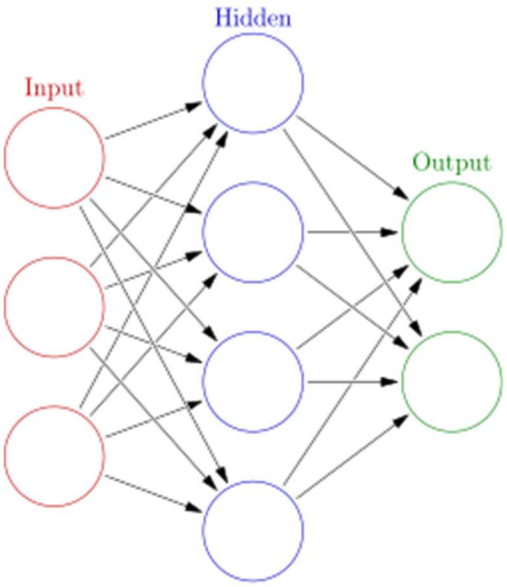
Figura 1 – Modelo matemático de um neurônio



Fonte: Ferneda (2006).

Como ilustrado na Figura 2, a arquitetura de uma rede neural é organizada em camadas de neurônios: uma camada de entrada (*input*), uma ou mais camadas ocultas (*hidden*) e uma camada de saída (*output*) (IBM, 2025). Cada neurônio (representado por um círculo) possui pesos e um valor de limiar associados. Desta forma, ele processa a informação recebida, calculando a soma ponderada de suas entradas (onde cada entrada é multiplicada por um peso) e, em seguida, a saída é processada por uma função de ativação, a qual determina o resultado final (IBM, 2025). Quando o valor de saída de um neurônio excede o limite especificado, esse será ativado, isto é, será transmitido como um dado para a próxima camada da rede.

Figura 2 – Representação de uma rede de neurônios artificial



Fonte: Wikipedia (2013).

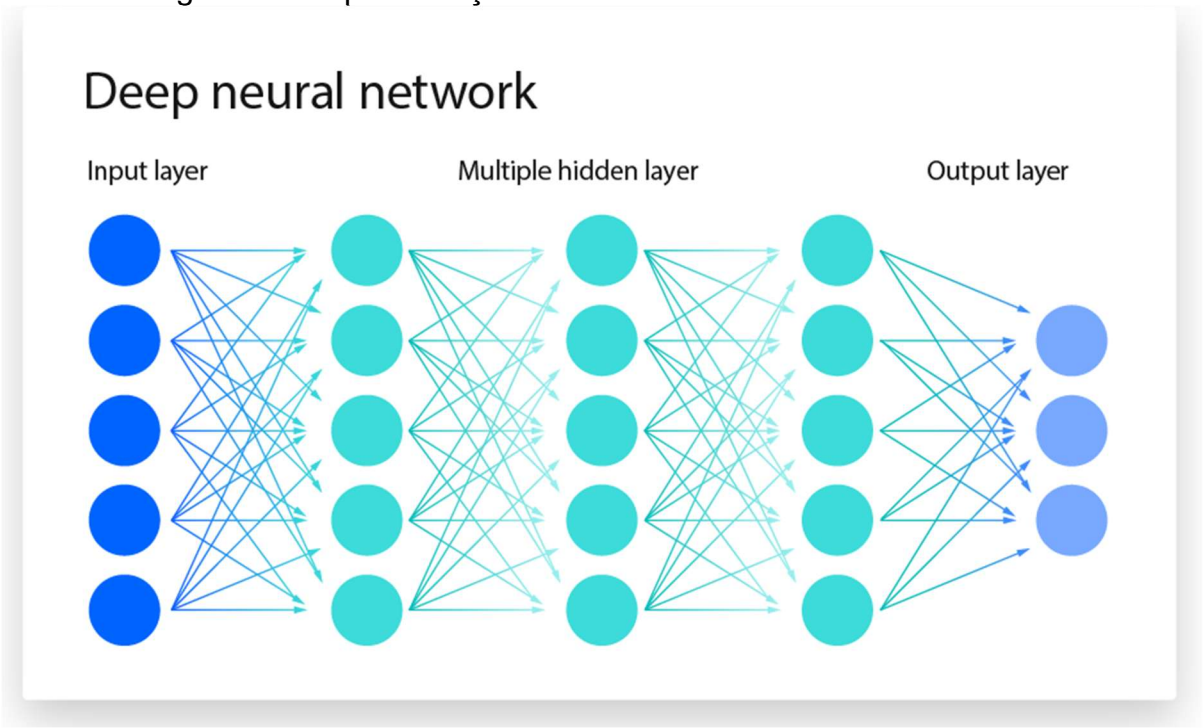
2.3 APRENDIZADO PROFUNDO (*DEEP LEARNING*)

Embora os termos *Deep Learning* (Aprendizado Profundo) e *Neural Network* (Rede Neural) sejam frequentemente usados como sinônimos, eles não são intercambiáveis. O *Deep Learning* (DL) é, na verdade, um do Aprendizado de Máquina (*Machine Learning*) que utiliza Redes Neurais Artificiais com uma arquitetura específica (Bergmann, 2025). A distinção fundamental reside na profundidade (o "*deep*" do termo), que se refere à quantidade de camadas da rede neural. Logo, "uma rede neural que consiste em mais de três camadas, que incluiriam as entradas e a saída, pode ser considerada um algoritmo de *deep learning*. Uma rede neural que tem apenas duas ou três camadas é apenas uma rede neural básica" (IBM, 2025).

O *Deep Learning* é composto por camadas sucessivas de neurônios artificiais, permitindo que os modelos computacionais aprendam representações de dados em diferentes níveis de abstração (LeCun; Bengio; Hinton, 2015). Sua capacidade de aprender diretamente dos dados brutos, sem a necessidade de engenharia de características manuais, o torna ideal para tarefas complexas, como classificação de imagens. No contexto da visão computacional, um modelo de arquitetura de DL predominantemente utilizado é a Rede Neural Convolucional (*Convolutional Neural Networks*).

De forma similar à Figura 2, a Figura 3 também representa visualmente uma rede neural, com círculos indicando os neurônios. A presença de três camadas ocultas, além das camadas de entrada e saída, é o que a caracteriza como uma rede neural "profunda" (*deep neural network*).

Figura 3 – Representação de uma rede com 3 camadas ocultas



Fonte: IBM (2025).

2.4 REDE NEURAL CONVOLUCIONAL (*CONVOLUTIONAL NEURAL NETWORKS*)

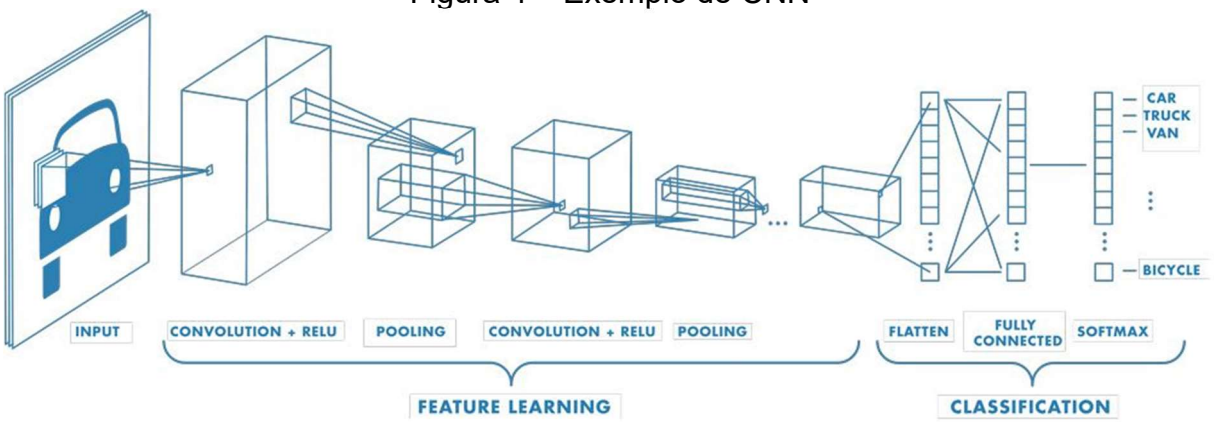
Como destacado anteriormente, a Rede Neural Convolucional (CNN) é uma arquitetura de *Deep Learning* formada por múltiplas camadas, isto é, uma de entrada, muitas camadas ocultas e uma camada de saída. Essa estrutura é predominantemente utilizada para o reconhecimento de imagens, pois é projetada especificamente para aprender a identificar características visuais ao longo do processo (IBM, 2025; MathWorks, 2025). Inicialmente, nas primeiras camadas, filtros são aplicados às imagens de entrada, permitindo que características simples, como contornos, linhas e brilhos, sejam detectadas (Google Cloud, 2025). À medida que os dados avançam pelas camadas ocultas, as operações realizadas se tornam mais complexas, permitindo que padrões, formas e, por fim, objetos inteiros e complexos sejam reconhecidos (Google Cloud, 2025; MathWorks, 2025).

Esse processo é construído por três componentes principais, que serão caracterizados conforme definições do Google Cloud (2025) e da MathWorks (2025). A camada inicial, denominada convolucional, aplica filtros para ativar características específicas e criar mapas de atributos. Em seguida, tem-se a camada de ativação

(ReLU), que introduz não-linearidade e acelera o treinamento ao remover valores negativos, garantindo que apenas os atributos mais relevantes passem para análise posterior. Na sequência, a camada de *pooling* (agrupamento) realiza a subamostragem, reduzindo a complexidade dos mapas de atributos e tornando o modelo computacionalmente mais eficiente. Ao repetir essas operações em várias camadas, a rede se torna capaz de identificar objetos em qualquer região da imagem, graças ao compartilhamento de pesos e vieses entre os neurônios.

Na etapa final, após o aprendizado das características, a rede organiza os dados em uma camada completamente conectada. Esta camada final atua como o classificador: ela analisa a combinação de alto nível dos atributos, calcula as probabilidades para cada classificação possível (Google Cloud, 2025) e, por fim, determina a saída, fornecendo a identificação do objeto presente na imagem. Esse processo torna as CNNs amplamente utilizadas em tarefas de visão computacional, como reconhecimento de imagens e classificação de objetos, independentemente de sua posição ou orientação (MathWorks, 2025). Todas essas etapas estão representadas na Figura 4.

Figura 4 – Exemplo de CNN



Fonte: MathWorks (2025).

Com o avanço do *Deep Learning*, surgiram arquiteturas de Redes Neurais Convolucionais mais sofisticadas e eficientes, que se tornaram amplamente utilizadas em tarefas de classificação de imagens devido ao seu desempenho superior. Neste contexto, optou-se por adaptar dois modelos previamente treinados em vastos conjuntos de dados, o *ResNet* e o *MobileNet*, que se destacam por suas abordagens distintas. A técnica utilizada para esta adaptação (*Transfer Learning*) é

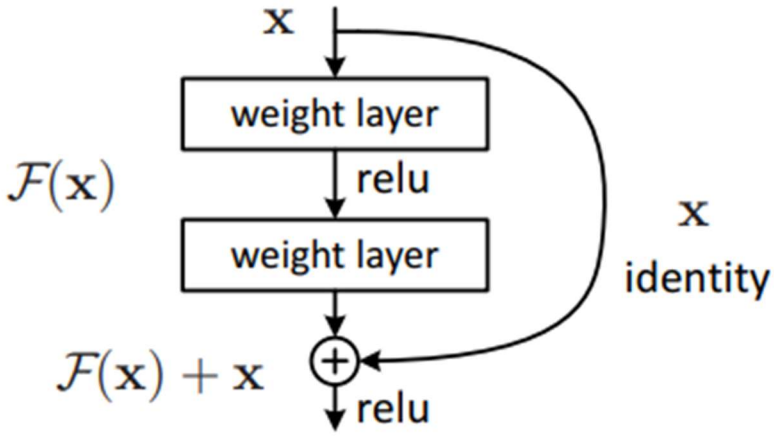
detalhada na seção 2.4.3, enquanto as características específicas de cada arquitetura são apresentadas a seguir (2.4.1 e 2.4.2).

2.4.1 ResNet

Em seu trabalho seminal “*Deep Residual Learning for Image Recognition*”, He *et al*. (2016a) introduziram a *ResNet* (Rede Residual), uma arquitetura desenvolvida para resolver problemas de degradação, os quais representam a maior dificuldade em treinar redes neurais que são “excessivamente profundas”. A inovação central são os blocos de aprendizado residual, que utilizam “conexões de atalho” (*skip connections*).

Como pode ser observado na Figura 5, essa conexão cria um "atalho" no fluxo de informações, pegando o sinal de entrada (x) e o adicionando diretamente à saída das camadas subsequentes. Em vez de forçar as camadas a aprender uma transformação completa, elas agora aprendem apenas a "função residual" (a diferença), o que é muito mais fácil de otimizar e permite o aprendizado eficiente em redes com centenas de camadas (He *et al*., 2016a).

Figura 5 – Um bloco de construção da ResNet



Fonte: MathWorks (2025).

2.4.2 MobileNet

A arquitetura *MobileNet* foi projetada especificamente para ser leve e eficiente, ideal para aplicações em dispositivos móveis e sistemas embarcados,

onde os recursos computacionais e de energia são limitados (Howard *et al.*, 2017). Conforme os autores, para alcançar essa eficiência, a *MobileNet* substitui a convolução padrão pela convolução separável em profundidade (*depthwise separable convolution*). Essa abordagem divide o processo em duas etapas mais simples: uma convolução *depthwise*, que aplica um único filtro por canal de entrada, e uma convolução *pointwise* (1x1), responsável por combinar os canais resultantes. Essa fatoração reduz drasticamente o custo computacional e o número de parâmetros, mantendo um bom desempenho. Além disso, a *MobileNet* permite ajustar a complexidade da rede por meio de hiperparâmetros, como o *width multiplier* e o *resolution multiplier*, que controlam a largura dos filtros e a resolução da imagem de entrada, respectivamente, possibilitando um equilíbrio entre velocidade e precisão (Howard *et al*., 2017).

2.4.3 Transfer Learning

As arquiteturas *ResNet* e *MobileNet*, apresentadas nas seções anteriores, são frequentemente disponibilizadas como modelos pré-treinados. Isso significa que seus parâmetros (pesos) já foram ajustados em um vasto conjunto de dados, conferindo-lhes a capacidade de extrair características visuais complexas, como texturas, formas e bordas.

Para utilizar essas redes no contexto específico da classificação de peixes em Florianópolis, emprega-se a técnica de *Transfer Learning* (Aprendizagem por Transferência). A motivação para o uso desta técnica, conforme definido por Weiss, Khoshgoftaar e Wang (2016), é a necessidade de criar um modelo de alto desempenho quando a coleta de dados de treinamento do domínio de destino é difícil, ou quando há uma oferta limitada desses dados.

A abordagem específica adotada neste trabalho é um exemplo de transferência de conhecimento por parâmetros. Nesta abordagem, os pesos da base convolucional (camadas internas da CNN) são mantidos e “congelados”, aproveitando o conhecimento prévio (Weiss; Khoshgoftaar; Wang, 2016). Como relatado pelos autores, o que é adaptado é a camada de classificação final (o 'topo' do modelo), que é substituída por uma nova camada, projetada para identificar as classes específicas do novo banco de dados (anchova, baiacu, etc.). Apenas essa nova camada é, então, treinada com os dados de destino (as imagens dos peixes).

2.5 CONFIGURAÇÕES INICIAIS

Antes de iniciar a etapa de desenvolvimento, foram definidos os hiperparâmetros basilares que nortearam os experimentos detalhados no Capítulo 4. Para o particionamento do conjunto de dados, adotou-se a proporção de 70% das imagens para treinamento, 15% para validação e 15% para teste. Segundo Muraina (2022), esta é a estratégia de separação recomendada para *datasets* com volume inferior a um milhão de amostras. Em relação ao tamanho do lote (*batch size*), Masters e Luschi (2018) indicam que, para bases de dados reduzidas, o valor ideal situa-se em 32, havendo viabilidade para lotes ainda menores. Quanto à duração do treinamento, Goodfellow, Bengio e Courville (2016) sugerem que, para conjuntos com menos de 10.000 imagens, o número de épocas deve ser limitado para evitar o *overfitting*. Com base nessa diretriz, definiu-se o valor de 15 épocas, o qual se mostrou adequado ao escopo do projeto, haja vista que a estabilização das métricas de desempenho foi observada, em geral, após a terceira época.

Já os parâmetros de aumento de dados foram definidos com base nas heurísticas estabelecidas por Chollet (2018) para o treinamento de redes neurais convolucionais em cenários com escassez de dados. A rotação de 40º e o cisalhamento de 0.2 visam simular a variabilidade da posição do peixe em relação à câmera, considerando que as fotos podem ser tiradas de diferentes ângulos manuais. Os deslocamentos horizontal e vertical de 20% (0.2) garantem que o modelo aprenda a identificar a espécie mesmo que o peixe não esteja perfeitamente centralizado na imagem. Por fim, o espelhamento horizontal é essencial na classificação de organismos vivos, pois a orientação lateral do peixe (cabeça para a esquerda ou direita) não altera sua classificação taxonômica.

2.6 ESTUDOS CORRELATOS

A classificação de espécies de peixes por meio de *Deep Learning*, especificamente Redes Neurais Convolucionais (CNNs), é um tema abordado por diversos trabalhos, cada um focado em diferentes estratégias, desafios e pontos de interesse. Analisam-se, a seguir, alguns desses estudos para ilustrar as diferentes abordagens.

O trabalho de Amorim (2022), por exemplo, propõe um modelo de Rede Neural Convolucional para a identificação automática de espécies de pescado, visando otimizar processos na indústria. Utilizando um banco de dados público com nove espécies (oito de peixes e uma de camarão da região do Mar Egeu), o estudo testou empiricamente cinco arquiteturas de rede distintas, variando o número de camadas convolucionais e filtros. O modelo M4, composto por três camadas convolucionais (filtros 64-128-256) e uma camada densa de 256 neurônios, alcançou o desempenho superior, com uma taxa de acurácia de 99% na fase de teste, demonstrando a alta viabilidade da abordagem para a automação industrial.

Abordando desafios diferentes, a dissertação de Srivastava (2023) investiga a classificação de espécies de peixes de água doce (truta arco-íris) para distinguir espécimes maduros de imaturos usando DL. O estudo aborda desafios críticos para a aquicultura, como a implementação em dispositivos de *edge computing* (Coral AI) com recursos limitados de processamento e memória e o manejo de dados altamente desbalanceados. Comparando as arquiteturas *ResNet-50* e *MobileNet* (V1, V2 e V3) em diferentes configurações de balanceamento de dados (desbalanceado, subamostrado e superamostrado com aumento de dados), a pesquisa concluiu que o *MobileNet* V1 apresentou o melhor desempenho geral para esta tarefa de classificação específica, equilibrando acurácia e eficiência computacional.

Focando no ambiente subaquático, o trabalho de Rathi *et al*. (2017) apresenta um modelo baseado em Redes Neurais Convolucionais para classificar espécies de peixes em imagens subaquáticas, abordando desafios como ruídos, distorções e condições ambientais adversas, incluindo iluminação inconsistente. O método combina técnicas de pré-processamento (borramento Gaussiano, binarização de Otsu e operações morfológicas) antes de alimentar as imagens na CNN. A arquitetura da rede não foi detalhada, mas os autores destacam o uso de funções de ativação como ReLU (que apresentou melhor desempenho), *tanh* e *softmax*. O modelo foi treinado e testado no conjunto de dados *Fish4Knowledge*, com 27.142 imagens de 23 espécies, alcançando uma precisão de 96,29%. Este trabalho enfatiza a aplicabilidade das CNNs em aplicações em tempo real, com um tempo médio de 0,00183 segundos por quadro.

O trabalho de Iqbal *et al*. (2021) foca na classificação automática de espécies de peixes, visando auxiliar biólogos marinhos na compreensão do ciclo de

vida e habitat das espécies. Os autores propõem um modelo customizado de CNN, descrito como uma "versão reduzida" da arquitetura *AlexNet*, composta por quatro camadas convolucionais e duas camadas totalmente conectadas. O modelo foi treinado utilizando 1.334 imagens de 6 espécies distintas. Os resultados da arquitetura proposta alcançaram uma acurácia de teste de 90,48%.

O Quadro 1 sintetiza as informações mais relevantes de cada trabalho, detalhando os modelos utilizados, as métricas de qualidade, o número de imagens e o número de espécies analisadas.

Quadro 1 – Comparação dos estudos correlatos

Estudo	Base de dados	Nº de imagens	Nº de espécies	Modelo	Métrica de qualidade
Rathi *et al*. (2017)	*Fish4 Knowledge*	27.142	23	CNN	Precisão = 96,29%
Amorim (2022)	*A Large Scale Fish Dataset (Kaggle)*	9.000	9 (8 peixes e 1 camarão)	CNN	Acurácia = 99,00%
Srivastava (2023)	Privado	29.966	1 (madura ou imatura)	CNN (ResNet-50, MobileNet V1, V2 e V3)	MobileNet V2 Acurácia = 91,10% e F1 Score = 0,90
Iqbal et al. (2021)	*QUT Dataset* (treino) e *LifeClef15* (teste)	1.334	6	CNN própria (base *AlexNet*)	Acurácia = 90,48%

Fonte: elaborado pelo autor (2025).

3 MATERIAIS E MÉTODOS

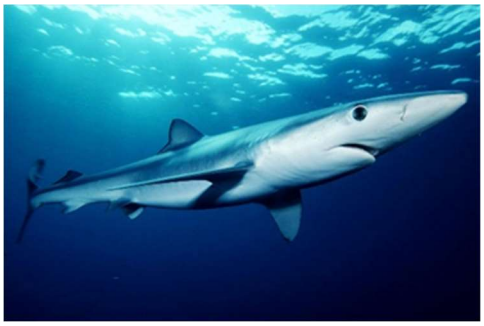
Este capítulo apresenta inicialmente a metodologia utilizada para a construção da base de dados, detalhando as fontes públicas e a coleta primária, bem como o processo de curadoria e tratamento de imagens. Na sequência, apresenta-se um diagrama que ilustra as etapas do trabalho. Subsequentemente, são especificadas as arquiteturas de CNNs selecionadas (*ResNet* e *MobileNet*), e suas características relevantes ao contexto de aplicação são detalhadas.

3.1 BASE DE DADOS

A base de dados (*dataset*) utilizada neste estudo foi estruturada a partir de técnicas mistas, integrando fontes de dados públicos e uma coleta primária realizada pelo autor. O objetivo desta abordagem foi compilar um conjunto de imagens diversificado e representativo das espécies selecionadas para o escopo deste trabalho. As espécies escolhidas (que incluem sete peixes e um crustáceo) e suas respectivas fontes de dados são detalhadas no Quadro 2.

Quadro 2 – Composição da base de dados

Imagem	Espécie	Base de dados (ano)
	Anchova *(Pomatomus saltatrix)*	Autoria própria (2024)
	Baiacu *(Dichotomyctere nigroviridis)*	Fish Dataset (2022)

	Cação *(Prionace glauca)*	Shark species (2020)
	Camarão (espécie não informada)	A Large Scale Fish Dataset (2020)
	Cavala *(Scomberomorus brasiliensis)*	Indrafish (2020)
	Garoupa (espécie não informada)	Indrafish (2020)
	Olho-de-Boi *(Priacanthus arenatus)*	Indrafish (2020)
	Tainha *(Mugil liza)*	Fish Dataset (2022)

Fonte: elaborado pelo autor (2025).

As imagens das espécies obtidas das bases de dados *Indrafish*, *Fish Dataset*, *Shark species* e *A Large Scale Fish Dataset* foram extraídas da plataforma Kaggle, uma comunidade online e repositório que hospeda uma vasta coleção de

datasets públicos para pesquisas. As imagens da espécie Anchova, por sua vez, foram capturadas pessoalmente pelo autor no Mercado Público de Florianópolis, com autorização prévia das peixarias locais para fins acadêmicos.

É importante ressaltar que os dados de ambas as fontes (públicas e primárias) não foram utilizados diretamente. Este material passou por um processo de triagem e curadoria manual para assegurar a adequação ao escopo da pesquisa. Durante essa etapa, foram excluídas imagens inadequadas, como desenhos ou ilustrações. Além disso, as fotografias que continham mais de um peixe foram sistematicamente recortadas para isolar espécimes individuais.

Concluída esta etapa de filtragem e tratamento, a base de dados personalizada foi finalizada. A Tabela 1 apresenta a distribuição quantitativa das imagens para cada uma das espécies selecionadas. Garantiu-se, assim, um *dataset* diversificado, representativo das espécies de peixes mais comuns na região de Florianópolis e otimizado para o treinamento e validação do modelo de classificação desenvolvido neste trabalho.

Tabela 1 – Composição final da base de dados

ESPÉCIE	NÚMERO DE IMAGENS
Anchova	44
Baiacu	22
Cação	84
Camarão	50
Cavala	153
Garoupa	274
Olho-de-Boi	294
Tainha	42
TOTAL	**963**

Fonte: elaborado pelo autor (2025).

A seleção das espécies para esta base de dados fundamentou-se em sua relevância cultural, econômica, gastronômica e histórica no contexto da pesca artesanal do estado de Santa Catarina, com foco no município de Florianópolis. Esta abordagem adota um recorte representativo, validado pelos critérios, dados de desembarque pesqueiro e levantamentos prévios junto às comunidades locais

utilizados por Ribas (2016) na composição do livro “Que peixe é este? O sabor da pesca artesanal na Ilha de Santa Catarina”. A organizadora priorizou aspectos que equilibram relevância comercial e cultural, tais como: o volume de captura, o baixo valor comercial, o uso tradicional gastronômico, a ausência nos cardápios de restaurantes locais. Seguindo esta curadoria, o presente trabalho selecionou espécies como a anchova, o cação, a cavala, o olho-de-boi e a tainha, identificadas no referido estudo como centrais para a pesca artesanal na região.

A garoupa, embora não figure na seleção de Ribas (2016), em virtude dos critérios específicos daquela obra, apresenta uma conexão intrínseca e histórica com a pesca artesanal. Esta modalidade de pesca foi, de fato, responsável pela maior parte das capturas da espécie no litoral brasileiro entre as décadas de 1970 e 1990 (Brasil, 2018). A inclusão da garoupa neste estudo justifica-se, portanto, por sua dupla relevância: além de ser um alvo tradicional da pesca local, a espécie encontra-se, atualmente, ameaçada de extinção (Brasil, 2018).

O baiacu, também incluído nesta base, é uma espécie comum em ecossistemas costeiros e estuários, presente na América do Sul, especialmente no Brasil. Sua relevância cultural reside em sua apreciação culinária. Na Ilha de Santa Catarina, o peixe está presente também na Lagoa da Conceição (Leão, 2010). O camarão, por sua vez, possui relevância central para a pesca artesanal na região de Florianópolis, sobretudo nas baías da Ilha de Santa Catarina. Sua inclusão justifica-se por sua tradição e seu papel indiscutível como fonte de renda para centenas de famílias e comunidades pesqueiras catarinenses (Santa Catarina, 2025).

3.2 ARQUITETURAS E VERSÕES SELECIONADAS

A seleção das arquiteturas para este estudo foi deliberada, focando especificamente nas versões "V2" de duas das mais influentes famílias de Redes Neurais Convolucionais: *MobileNetV2* e *ResNet50V2*. Esta escolha permite uma análise comparativa direta entre um modelo otimizado para eficiência computacional (*MobileNetV2*) e outro focado em maximizar a acurácia (*ResNet50V2*).

A *MobileNetV2* é uma arquitetura projetada para alto desempenho em dispositivos com restrições de hardware, como aplicações móveis. Ela aprimora sua predecessora (V1) ao introduzir duas inovações principais: os resíduos invertidos (*inverted residuals*) e os gargalos lineares (*linear bottlenecks*). Juntas, essas

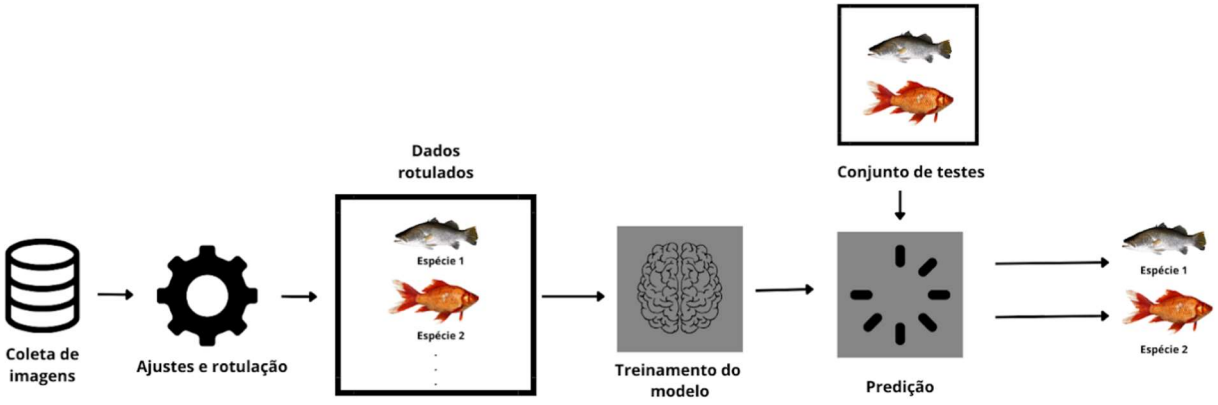
técnicas permitem a construção de redes mais profundas e eficientes com um número significativamente menor de parâmetros (Sandler *et al*., 2018).

Paralelamente, a *ResNet50V2* representa um refinamento da arquitetura *ResNet*, que popularizou o aprendizado residual. A versão V2 otimiza o fluxo de gradiente (crucial para o treinamento de redes profundas) ao modificar a ordem das operações dentro do bloco residual. Ela implementa o conceito de pré-ativação, onde a Normalização em Lote (*Batch Normalization*) e a função de ativação (ReLU) são aplicadas antes da operação de convolução, o que resulta em uma convergência mais estável e, frequentemente, em uma acurácia superior ao final do treinamento (He *et al*., 2016b).

3.3 ETAPAS DE TRABALHO

A Figura 6 ilustra o fluxo adotado para a construção do modelo. A primeira etapa consiste na coleta das imagens das espécies de peixes selecionadas, descritas anteriormente. A segunda etapa abrange o pré-processamento (ajustes) e a rotulação dos dados, onde as imagens são organizadas em diretórios (pastas) correspondentes a cada espécie. Subsequentemente, com os dados rotulados, inicia-se o treinamento do modelo, utilizando as arquiteturas e versões selecionadas. Por fim, a última etapa volta-se ao teste e avaliação, na qual o modelo é submetido a um conjunto de testes para aferir sua precisão.

Figura 6 – Diagrama de etapas para desenvolvimento do modelo

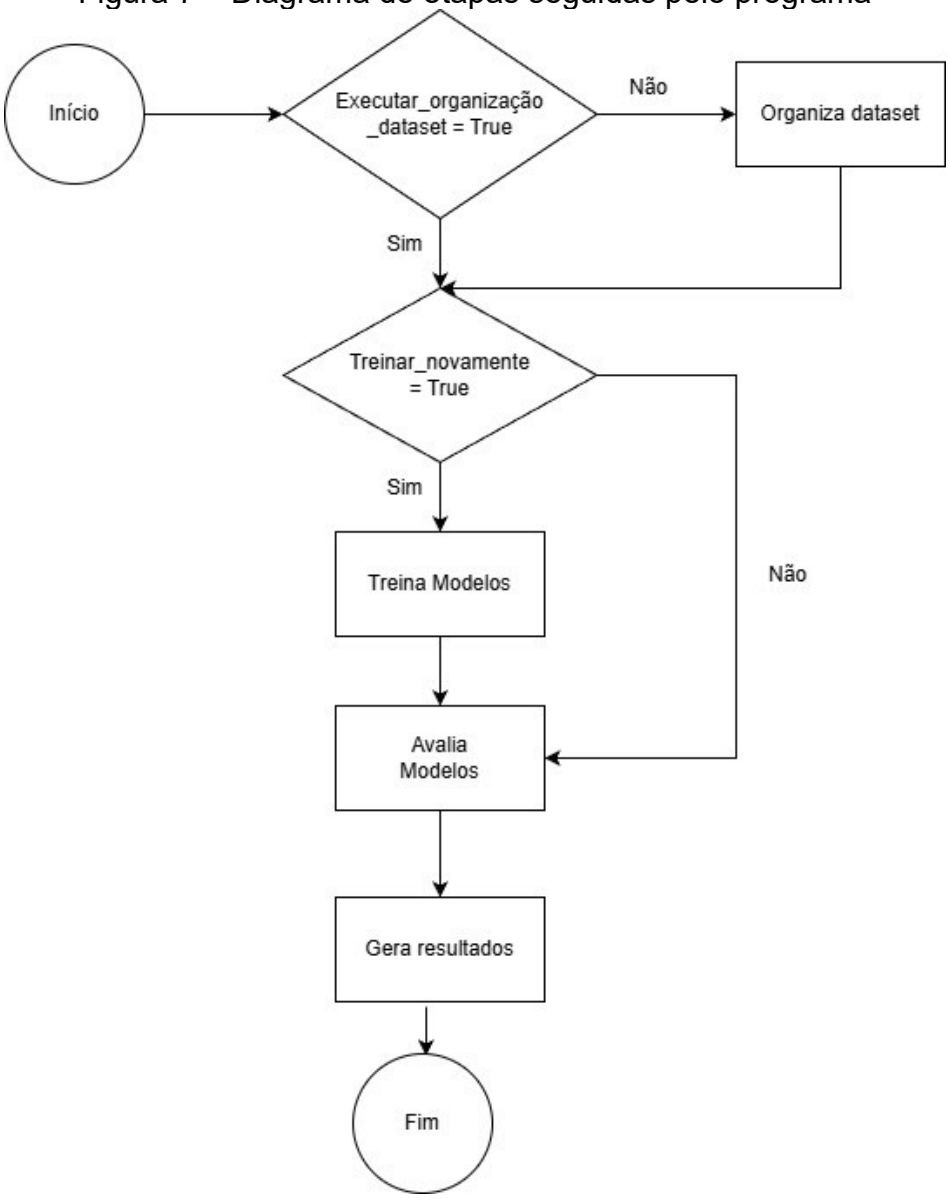


Fonte: elaborado pelo autor (2025).

4 DESENVOLVIMENTO

Esta seção detalha o processo de desenvolvimento do protótipo de *software* para o reconhecimento de espécies de peixes, conduzido integralmente no ambiente *Google Colaboratory* (Colab), utilizando a GPUs:T4. A metodologia seguiu cinco etapas principais: (1) organização e a separação do *dataset*; (2) pré-processamento e *data augmentation*; (3) seleção e implementação das arquiteturas de rede neural; (4) treinamento dos modelos; e (5) definição das métricas de avaliação. O diagrama a seguir ilustra as etapas a serem seguidas pelo programa (Figura 7).

Figura 7 – Diagrama de etapas seguidas pelo programa



Fonte: elaborado pelo autor (2025).

4.1 ORGANIZAÇÃO E SEPARAÇÃO DO *DATASET*

Para garantir um processo de treinamento e avaliação robusto e reprodutível, foi desenvolvido um *script* para automatizar a separação inicial dos dados em uma primeira execução. O controle desta etapa é gerenciado por uma variável booleana, EXECUTAR_ORGANIZACAO_DATASET, definida no início do programa. Após a primeira execução bem-sucedida, que realiza a separação, esta variável pode ser alterada para "*False"*.

O *script* utiliza duas variáveis principais para os caminhos: PATH_DADOS_ORIGINAIS aponta para o diretório que contém o conjunto de imagens original, enquanto PATH_DATASET_ORGANIZADO define o local de destino onde o conjunto já organizado, que será lido pelo modelo, será criado. Caso uma nova organização do *dataset* não seja mais necessária, este último caminho é utilizado para a leitura dos dados pelo modelo.

As imagens, originalmente agrupadas por pastas de classe, foram sistematicamente divididas em três conjuntos distintos:

a. conjunto de treinamento, que contém a maior porção dos dados, utilizada para treinar o modelo no reconhecimento dos padrões de cada classe;
b. conjunto de validação, que é utilizado durante o treinamento para monitorar o desempenho do modelo em dados "novos" (ou seja, dados não utilizados no treinamento) ao final de cada época, sendo um aspecto crucial para o ajuste de hiperparâmetros e a identificação de *overfitting* (sobreajuste);
c. conjunto de teste, que é um conjunto de dados completamente separado, isto é, mantido "cego" até o final do processo, de modo que o desempenho neste conjunto representa a avaliação final e imparcial da capacidade de generalização do modelo.

Para a definição do tamanho de cada conjunto é necessário alterar o valor de duas variáveis: TEST_SPLIT que define a porcentagem de dados alocada ao conjunto de teste, e VALIDATION_SPLIT que define a porcentagem destinada ao conjunto de validação. O conjunto de treinamento é automaticamente composto pela porção remanescente das imagens. Neste trabalho, os testes realizados utilizaram as proporções de 80%/10%/10% e 70%/15%/15% para Treinamento/Validação/Teste, respectivamente.

4.2 PRÉ-PROCESSAMENTO E *DATA AUGMENTATION*

Antes de alimentar as redes neurais, as imagens precisam ser pré-processadas. Este processo foi implementado utilizando a classe *ImageDataGenerator* da biblioteca Keras, abrangendo duas etapas principais. Primeiramente, todas as imagens (de treinamento, validação e teste) foram redimensionadas para um tamanho de entrada uniforme de 224x224 *pixels*, que é o padrão exigido pelas arquiteturas selecionadas. Além disso, os valores dos *pixels* foram normalizados, sendo reescalados do intervalo [0, 255] para [0, 1].

Em segundo lugar, ao conjunto de treinamento, foi aplicada a técnica de *Data Augmentation* (Aumento de Dados). Esta técnica visa aumentar artificialmente a variabilidade do conjunto de treinamento, criando novas imagens modificadas em tempo real a cada época. Isso torna o modelo mais robusto e ajuda a prevenir o *overfitting* (sobreajuste). As transformações aplicadas incluíram:

a. rotação (*rotation_range*=40): giros aleatórios de até 40º;
b. deslocamento de largura e altura (*width_shift_range*=0.2, *height_shift_range*=0.2): movimentação horizontal ou vertical da imagem em até 20%;
c. cisalhamento (*shear_range=0.2*): distorção angular da imagem;
d. espelhamento horizontal (*horizontal_flip=True*): inversão horizontal da imagem.

É fundamental notar que o *Data Augmentation* foi aplicado apenas ao conjunto de treinamento. Os conjuntos de validação e teste permaneceram inalterados (exceto pelo redimensionamento e normalização) para garantir que a avaliação do modelo reflita seu desempenho em dados reais e não modificados. A aplicação de *zoom* (ampliação), outra técnica comumente utilizada, foi descartada para garantir que a imagem seja analisada por completo, sem perda de características.

4.3 SELEÇÃO E IMPLEMENTAÇÃO DAS ARQUITETURAS DE REDE NEURAL

Para este projeto, foram selecionadas duas arquiteturas de Redes Neurais Convolucionais (CNNs): *MobileNetV2* e *ResNet50V2*. A abordagem utilizada foi o *Transfer Learning* (Aprendizagem por Transferência), já descrita anteriormente.

O processo de implementação seguiu três etapas principais (conforme a Função criar_modelo_baseado_em):

1. Carregar o modelo base: as arquiteturas *MobileNetV2* e *ResNet50V2* foram carregadas com os pesos aprendidos no *ImageNet* (*weights="imagenet"*), ou seja, um modelo já treinado, mas sem suas camadas finais de classificação (*include_top=False*). Esta etapa é crucial, pois remove o classificador original (treinado para 1.000 classes do *ImageNet*), permitindo que um novo topo de classificação seja adicionado.
2. Congelar a base: as camadas convolucionais (a "base") de ambos os modelos foram "congeladas" (*base_model.trainable = False*). Esta ação impede que seus pesos sejam alterados durante o treinamento inicial, preservando o conhecimento adquirido no *ImageNet*.
3. Adicionar um novo classificador (topo): um novo "topo" de classificação foi adicionado, conectado à saída da base congelada. Esse topo é composto, em sequência, por:
    a. uma camada *GlobalAveragePooling2D* para condensar os mapas de características;
    b. uma camada *Dropout*(0.2) para regularização, ajudando a prevenir o *overfitting* do topo ao desligar 20% dos neurônios aleatoriamente. Esta técnica também força a rede a não depender de um “super-neurônio”, isto é, depender excessivamente de neurônios específicos para classificar;
    c. uma camada Densa (*Dense*) com 512 neurônios (valor comumente utilizado por apresentar um bom equilíbrio), já que valores menores poderiam ter dificuldade em diferenciar espécies com características similares, enquanto valores maiores aumentam o risco de *overfitting*; além disso, a camada utiliza ativação ReLU (Unidade Linear Retificada), considerada o padrão mais eficiente em projetos modernos;
    d. uma camada de saída *Dense* final com 8 neurônios (um para cada classe) e ativação *Softmax*, que gera as probabilidades de classificação para cada espécie.

Com essa abordagem, o processo de treinamento concentra-se em treinar apenas o novo classificador (o "topo") a usar as características extraídas pela base congelada para identificar as espécies.

4.4 TREINAMENTO DOS MODELOS

A etapa de treinamento foi configurada para incluir *hyperparameter tuning*. Para tanto, os modelos foram treinados com diferentes configurações de tamanho do lote (*batch size*), definido na variável BATCH_SIZE, e de número máximo de épocas, definido na variável EPOCHS, em busca do melhor resultado. Os modelos foram compilados usando o otimizador Adam (com taxa de aprendizado de 0,001) e a função de perda Entropia Cruzada Categórica (*Categorical Cross-entropy*), adequada para problemas de classificação multiclasse.

Para garantir a qualidade do modelo final e otimizar o processo, dois *callbacks* da biblioteca Keras foram utilizados:

a. *ModelCheckpoint*: Este *callback* monitorou a *val_accuracy* (acurácia no conjunto de validação) ao final de cada época. Ele salvou automaticamente em disco (arquivo .h5) apenas o modelo (os pesos) que apresentou o melhor desempenho de validação até aquele momento.
b. *EarlyStopping*: Este *callback* monitorou a *val_loss* (perda no conjunto de validação). Se a perda de validação não apresentasse melhoria por 7 épocas consecutivas (*patience=7*), o treinamento era interrompido automaticamente. Esta interrupção economiza recursos computacionais e previne que o modelo continue treinando e entre em estado de *overfitting* (sobreajuste).

4.5 DEFINIÇÃO DAS MÉTRICAS DE AVALIAÇÃO

Para avaliar de forma abrangente o desempenho dos modelos finais (os melhores salvos pelo *ModelCheckpoint*), foi utilizado o conjunto de teste. A função *avaliar_modelo_completo* foi designada para calcular e salvar um conjunto robusto de métricas, permitindo uma análise detalhada da capacidade de generalização de cada modelo. As métricas selecionadas foram:

a. Relatório de Classificação: fornece as métricas de Acurácia, Precisão, Sensibilidade (*Recall*) e *F1-Score* por classe e em média (macro);
b. Acurácia: a porcentagem geral de acertos;
c. Precisão: mede a proporção de predições positivas que foram de fato corretas (por exemplo: dos peixes classificados como "tainha", quantos eram realmente "tainha");
d. Sensibilidade (*Recall*): mede a proporção de instâncias positivas reais que foram corretamente identificadas;
e. F1-Score: a média harmônica entre Precisão e Sensibilidade, útil para avaliar o equilíbrio do modelo;
f. Matriz de Confusão (absoluta e percentual): uma tabela que cruza as classes verdadeiras com as classes previstas, importante para identificar quais classes o modelo está confundindo entre si;
g. Tempo de Inferência: mede o tempo médio, em milissegundos, que o modelo leva para classificar uma única imagem. Esta é uma métrica importante para avaliar a viabilidade de uma aplicação em tempo real;
h. Tempo de Treinamento: mede o tempo total que o modelo levou para ser treinado;
i. Número de Parâmetros: o total de pesos e vieses que a rede aprende durante o treinamento.

Os resultados detalhados obtidos a partir dessas métricas serão apresentados na próxima seção (Capítulo 5).

4.6 ETAPA FINAL DE *BACKUP*

A etapa final do código implementa a funcionalidade de conexão e "montagem" do Google Drive ao ambiente do Google Colab. Uma vez autenticado, o *script* utiliza um caminho de destino, previamente especificado pelo usuário, para salvar todos os resultados.

5 RESULTADOS E DISCUSSÃO

Para garantir o melhor resultado, foram realizados diversos testes com os modelos, cada um com uma variação de parâmetros. Serão destacados aqui os melhores desempenhos, juntamente com os valores selecionados para: tamanho do lote (BATCH_SIZE), épocas (EPOCHS) e as proporções de imagens para teste e validação (TEST_SPLIT e VALIDATION_SPLIT). As métricas escolhidas para analisar cada um dos modelos foram: histórico de acurácia e perda (*loss*), acurácia (geral), *F1-Score*, tempo de treinamento, tempo de inferência e matriz de confusão.

5.1 TESTE 1

O Teste 1 foi executado com a seguinte configuração de parâmetros, cujos resultados são descritos na Tabela 2:

- tamanho do lote: 32;
- épocas: 15;
- divisão de teste: 15%;
- divisão de validação: 15%.

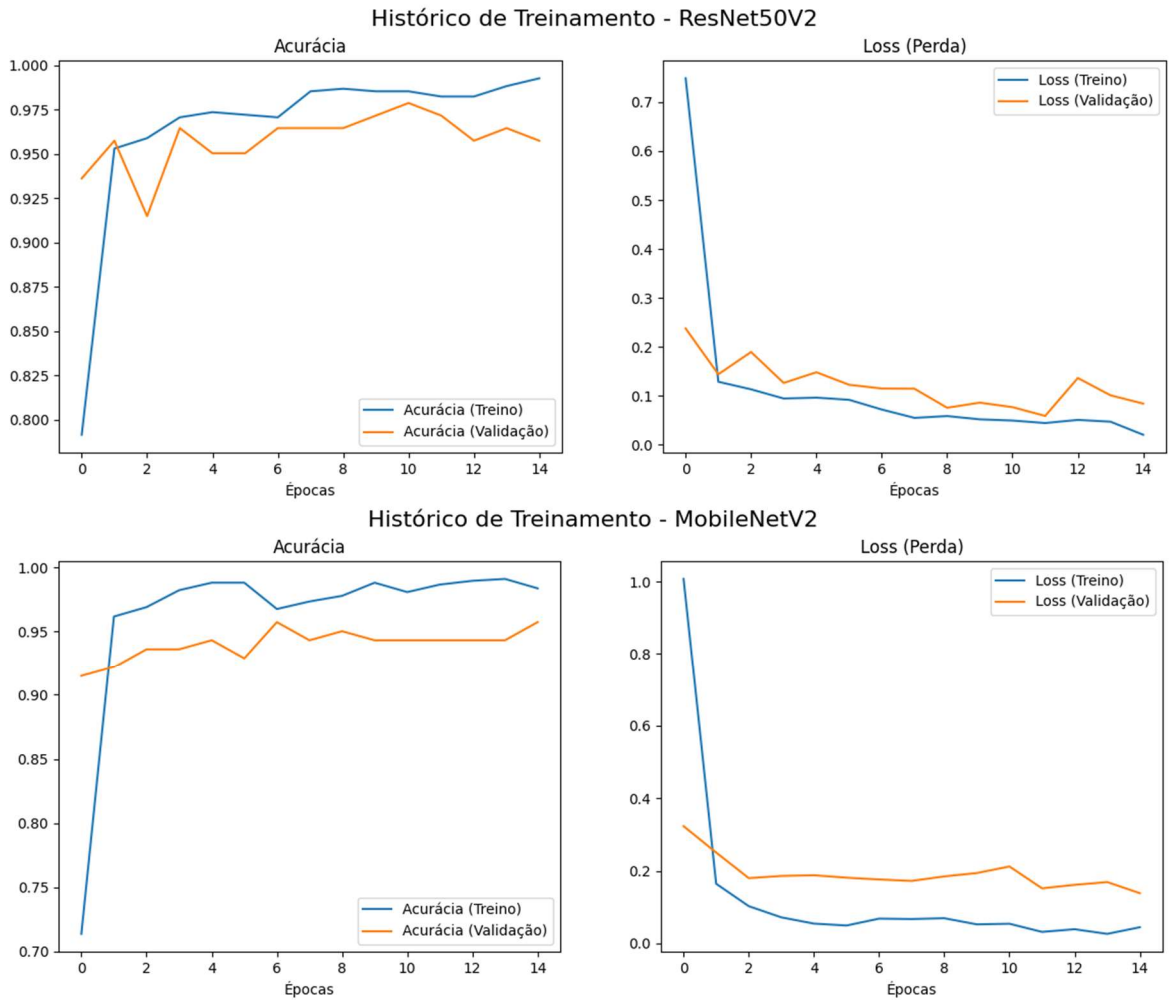
Tabela 2 – Resultado do teste 1

Modelo	Acurácia	F1-score	Treino (s)	Inferência (ms)	Parâmetros
MobileNetV2	**0,9929**	0,9971	266,1627	51,2782	2917960
ResNet50V2	**0,9929**	0,9859	161,3829	57,5195	24617992

Fonte: elaborado pelo autor (2025).

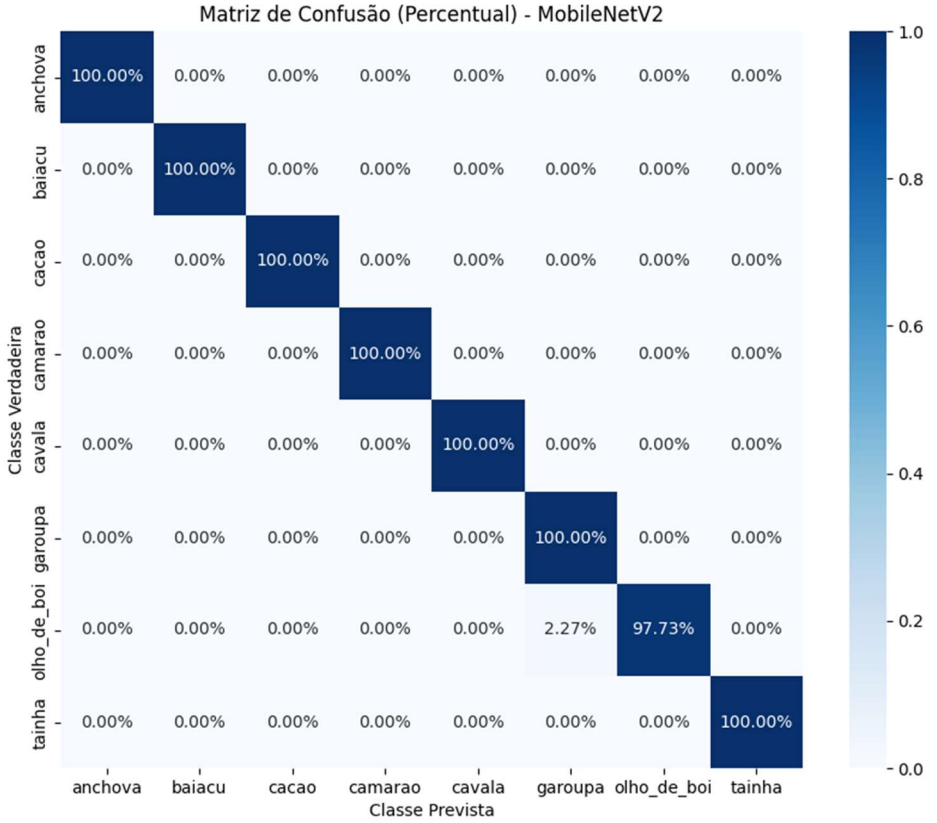
Para complementar os resultados sumarizados na Tabela 2, a Figura 8 apresenta o histórico de treinamento dos modelos. Na sequência, as Figura 9 (*MobileNetV2*) e Figura 10 (*ResNet50V2*) apresentam as matrizes de confusão.

Figura 8 – Histórico de Treinamento do teste 1



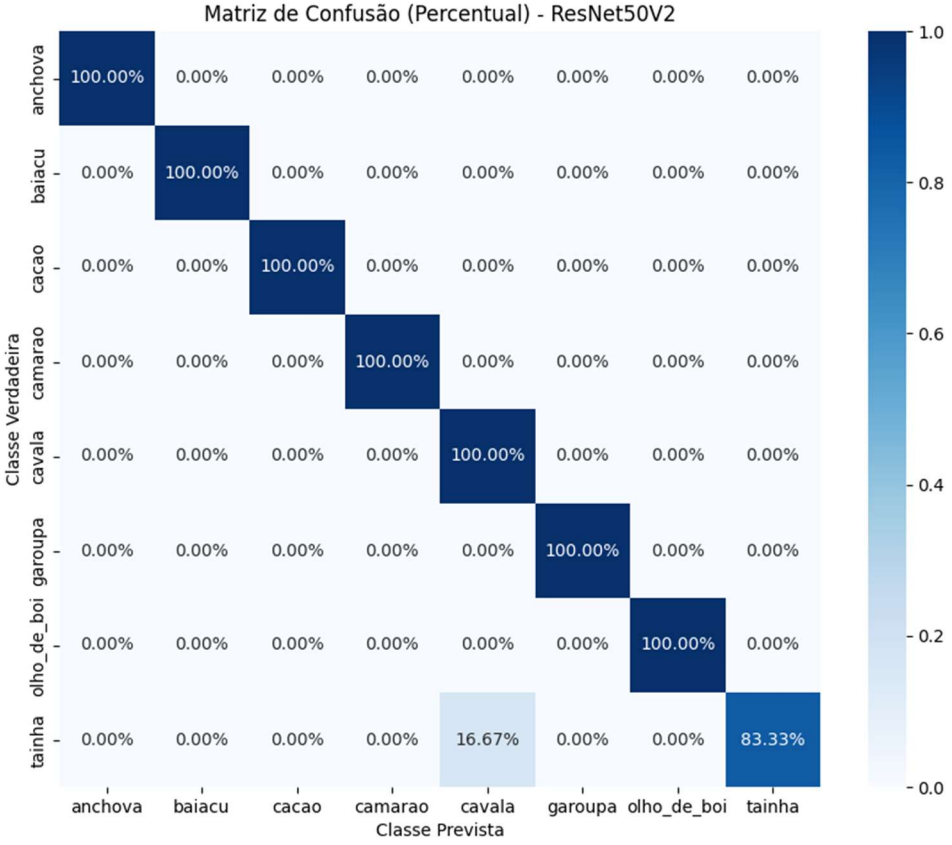
Fonte: elaborado pelo autor (2025).

Figura 9 – Matriz de confusão do teste 1 (*MobileNetV2*)



Fonte: elaborado pelo autor (2025).

Figura 10 – Matriz de confusão do teste 1 (*ResNet50V2*)



Fonte: elaborado pelo autor (2025).

Neste teste, a acurácia dos modelos foi a mesma e *o F1-score* foi similar. O desempenho por classe foi quase perfeito, com erros registrados apenas na classificação do olho-de-boi e da tainha. A maior diferença entre os modelos reside no número de parâmetros e no tempo de inferência: o *MobileNet* possui menos parâmetros, o que o torna um modelo muito mais leve, além de apresentar um tempo médio de inferência menor que o *ResNet.*

5.2 TESTE 2

O Teste 2 foi executado com a seguinte configuração de parâmetros, cujos resultados são descritos na Tabela 3:

- tamanho do lote: 32;
- épocas: 10;
- divisão de teste: 15%;
- divisão de validação: 15%.

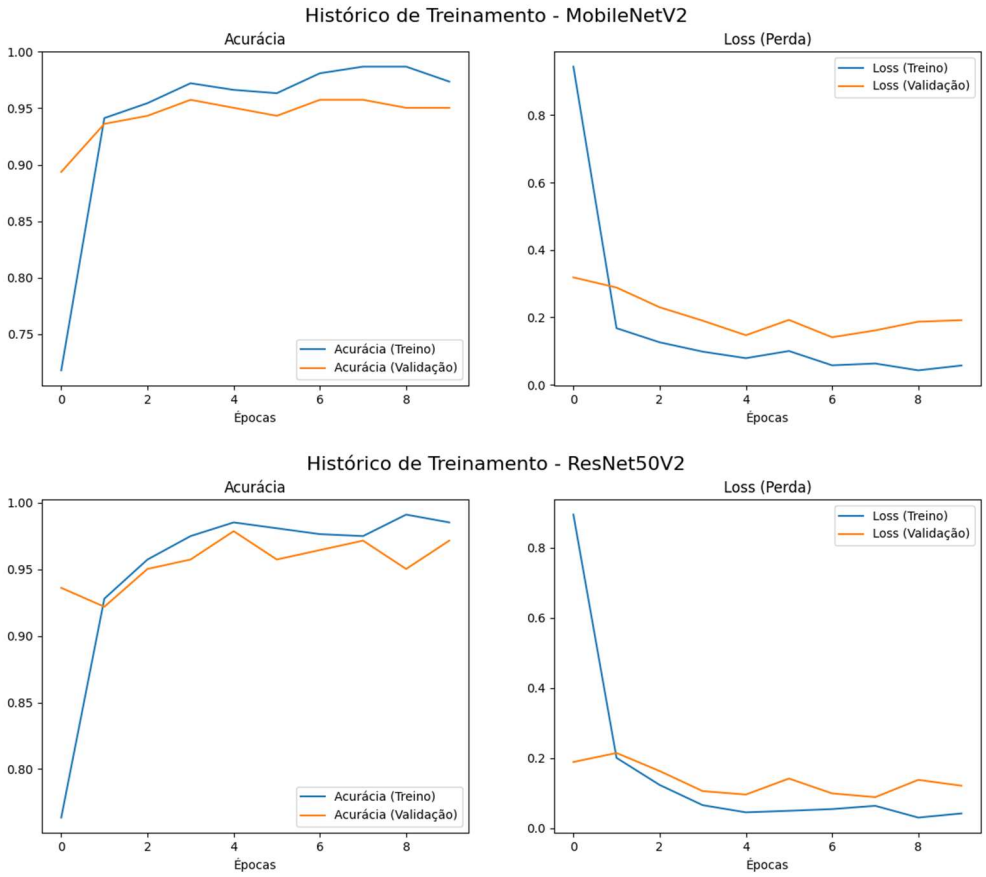
Tabela 3 – Resultado do teste 2

Modelo	Acurácia	F1-score	Treino (s)	Inferência (ms)	Parâmetros
MobileNetV2	**0,9574**	0,9374	449,1500	367,9667	2917960
ResNet50V2	**0,9787**	0,9815	321,6537	51,4752	24617992

Fonte: elaborado pelo autor (2025).

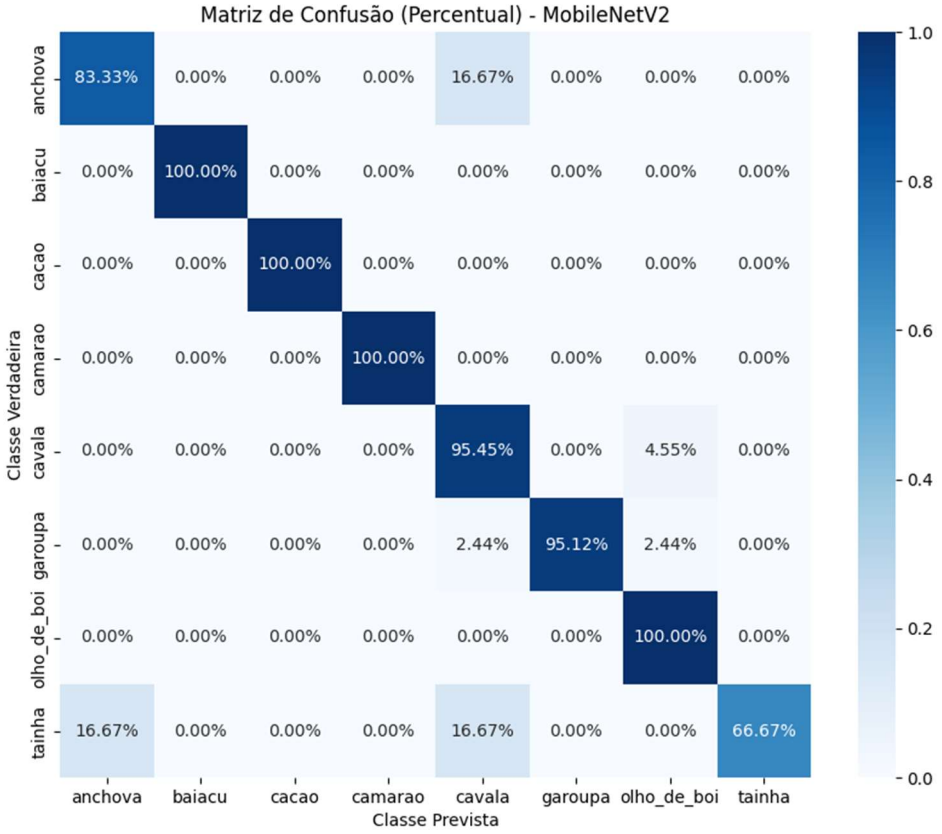
Para complementar os resultados sumarizados na Tabela 3, a Figura 11 apresenta o histórico de treinamento dos modelos. Na sequência, as Figura 12 (*MobileNetV2*) e Figura 13 (*ResNet50V2*) apresentam as matrizes de confusão.

Figura 11 – Histórico de Treinamento do teste 2



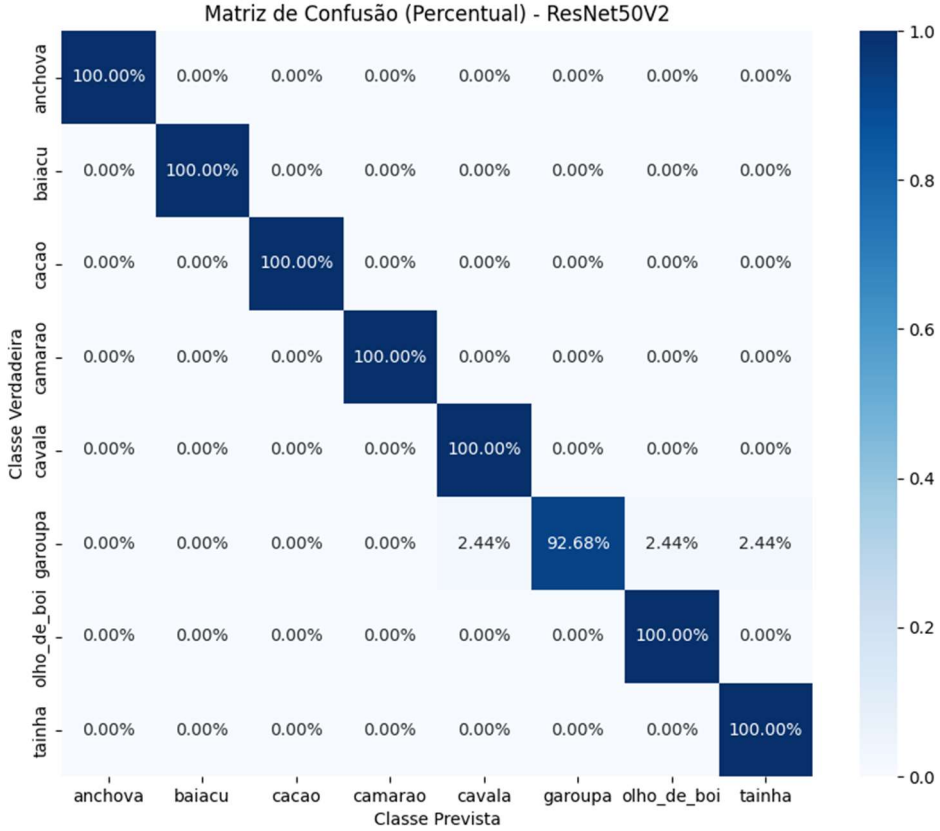
Fonte: elaborado pelo autor (2025).

Figura 12 – Matriz de confusão do teste 2 (*MobileNetV2*)



Fonte: elaborado pelo autor (2025).

Figura 13 – Matriz de confusão do teste 2 (*ResNet50V2*)



Fonte: elaborado pelo autor (2025).

No Teste 2, onde a única variável alterada em relação ao Teste 1 foi o número de épocas, é possível notar que o desempenho do *MobileNet* teve uma queda considerável, enquanto seu tempo de inferência aumentou. A partir desses dados, pode-se inferir que a configuração de 15 épocas (utilizada no Teste 1) se mostra mais adequada para os demais parâmetros testados.

5.3 TESTE 3

O Teste 3 foi executado com a seguinte configuração de parâmetros, cujos resultados são descritos na Tabela 4:

- tamanho do lote: 32;
- épocas: 15;
- divisão de teste: 10%;
- divisão de validação: 10%.

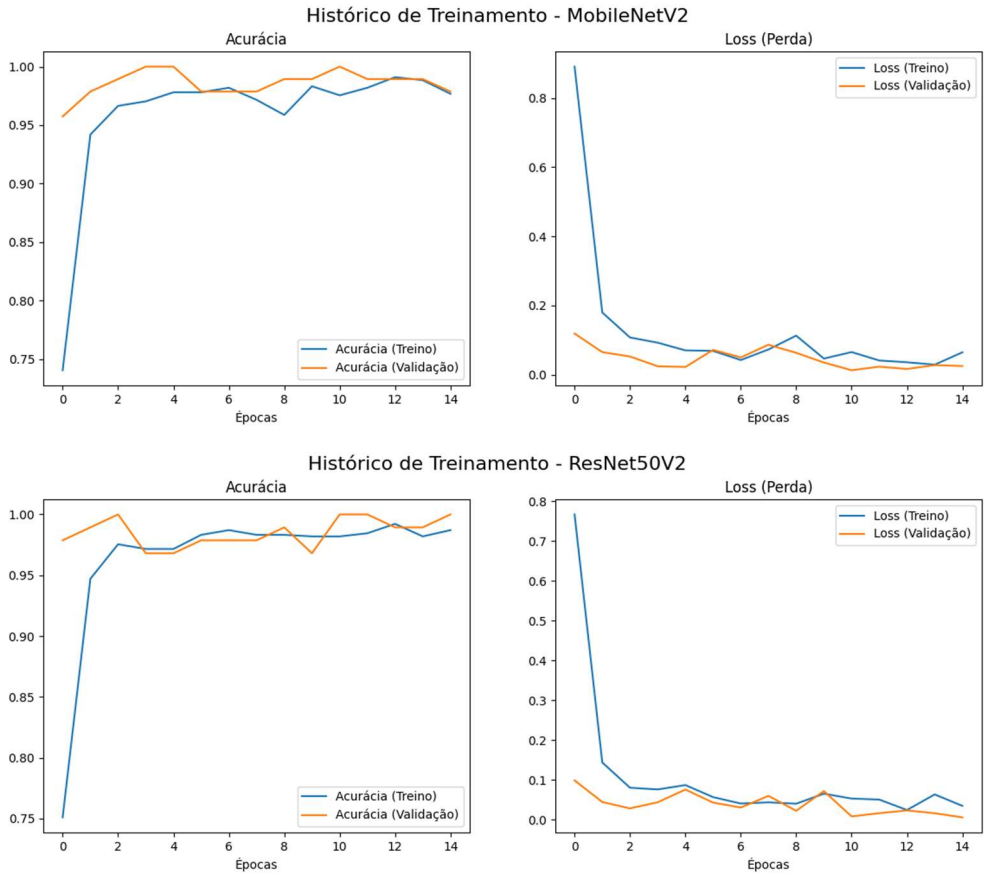
Tabela 4 – Resultado do teste 3

Modelo	Acurácia	F1-score	Treino (s)	Inferência (ms)	Parâmetros
MobileNetV2	**0,9787**	0,9638	425,6375	83,8664	2917960
ResNet50V2	**0,9681**	0,9844	424,1989	78,6293	24617992

Fonte: elaborado pelo autor (2025).

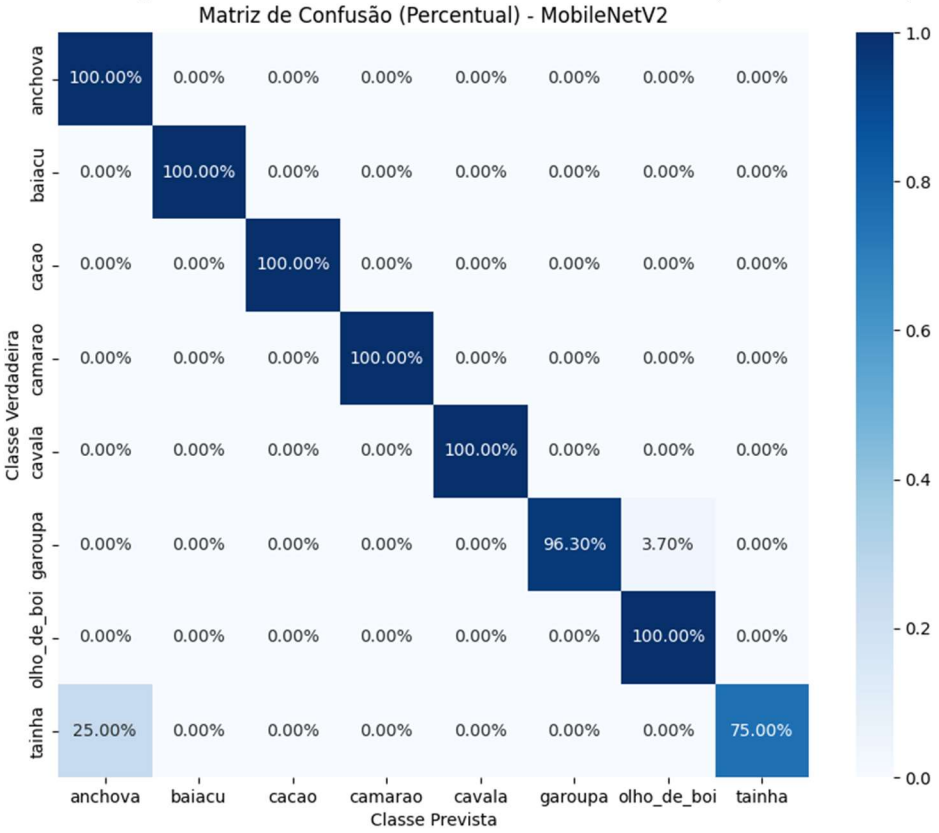
Para complementar os resultados sumarizados na Tabela 4, a Figura 14 apresenta o histórico de treinamento dos modelos. Na sequência, as Figura 15 (*MobileNetV2*) e Figura 16 (*ResNet50V2*) apresentam as matrizes de confusão.

Figura 14 – Histórico de Treinamento do teste 3



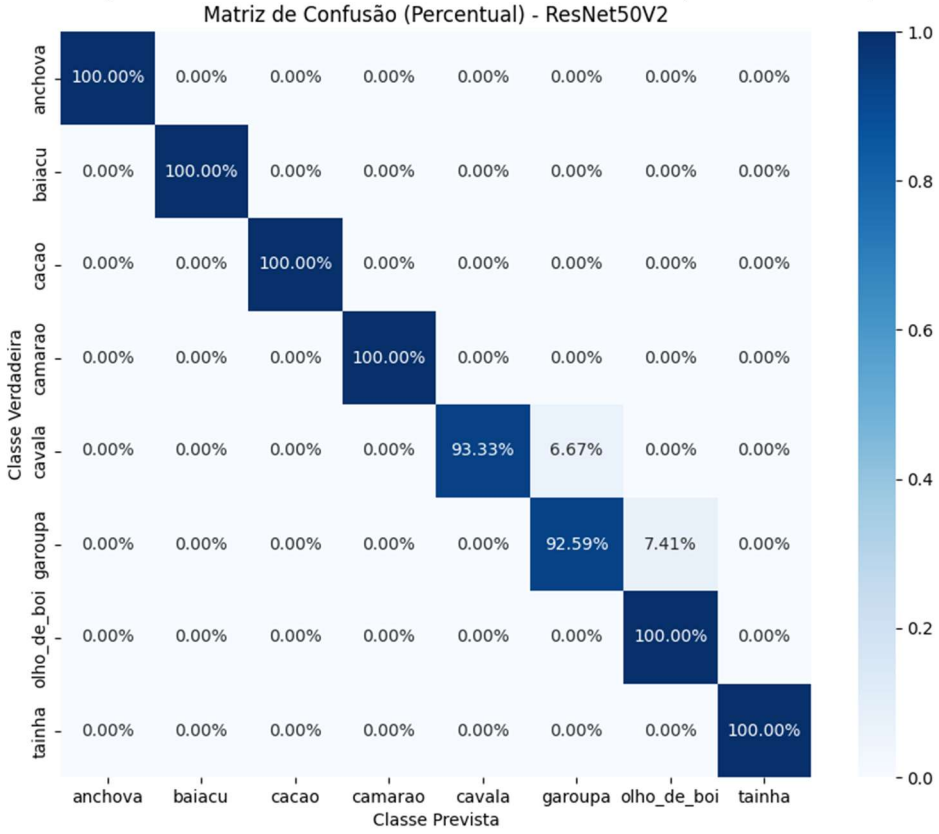
Fonte: elaborado pelo autor (2025).

Figura 15 – Matriz de confusão do teste 3 (*MobileNetV2*)



Fonte: elaborado pelo autor (2025).

Figura 16 – Matriz de confusão do teste 3 (*ResNet50V2*)



Fonte: elaborado pelo autor (2025).

O Teste 3 foi projetado para avaliar o impacto da alteração na divisão dos dados, aumentando o conjunto de treino para 80% (e reduzindo a validação e o teste para 10% cada). Ao comparar seus resultados (Tabela 4) com os do Teste 1 (Tabela 2), que utilizou 15% para validação e 15% para teste, nota-se que a configuração do Teste 1 obteve um desempenho superior. A redução dos conjuntos de validação e teste (de 15% para 10%), apesar de fornecer mais dados de treino, resultou em uma ligeira queda na acurácia final para ambos os modelos.

5.4 TESTE 4

O Teste 4 foi executado com a seguinte configuração de parâmetros, cujos resultados são descritos na Tabela 5:

- tamanho do lote: 16;
- épocas: 15;
- divisão de teste: 15%;
- divisão de validação: 15%.

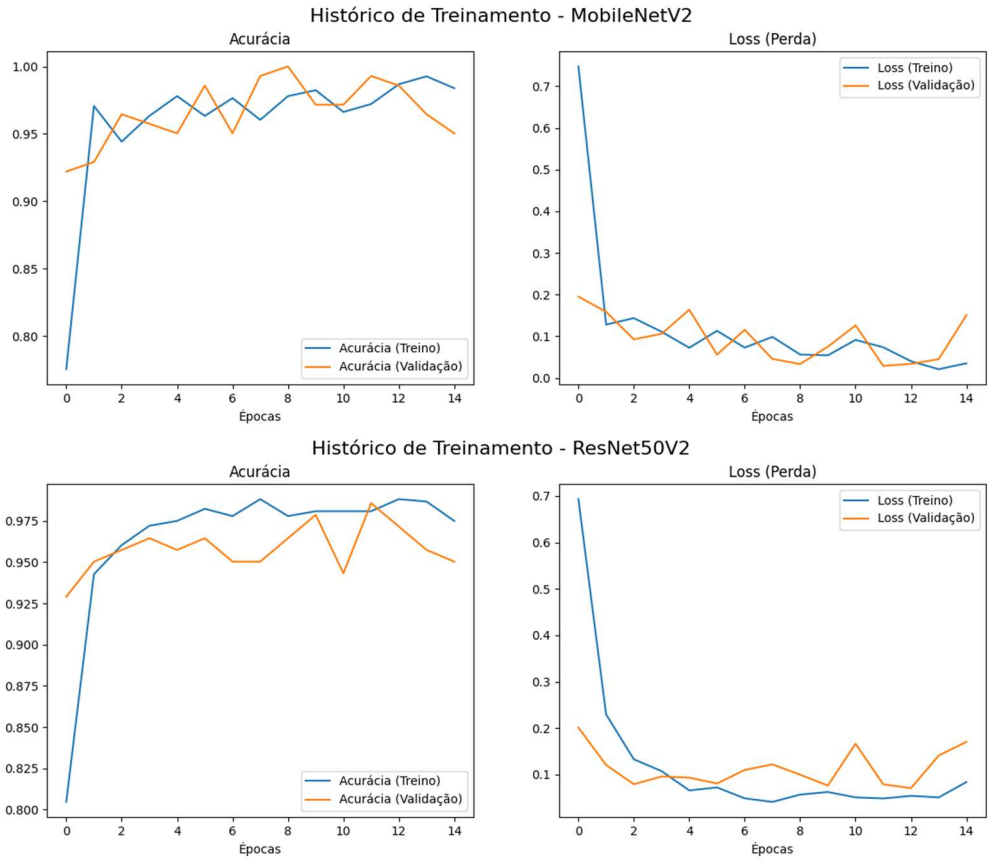
Tabela 5 – Resultado do teste 4

Modelo	Acurácia	F1-score	Treino (s)	Inferência (ms)	Parâmetros
MobileNetV2	**0,9787**	0,9733	409,6988	58,9792	2917960
ResNet50V2	**1,0000**	1,0000	411,5292	59,2750	24617992

Fonte: elaborado pelo autor (2025).

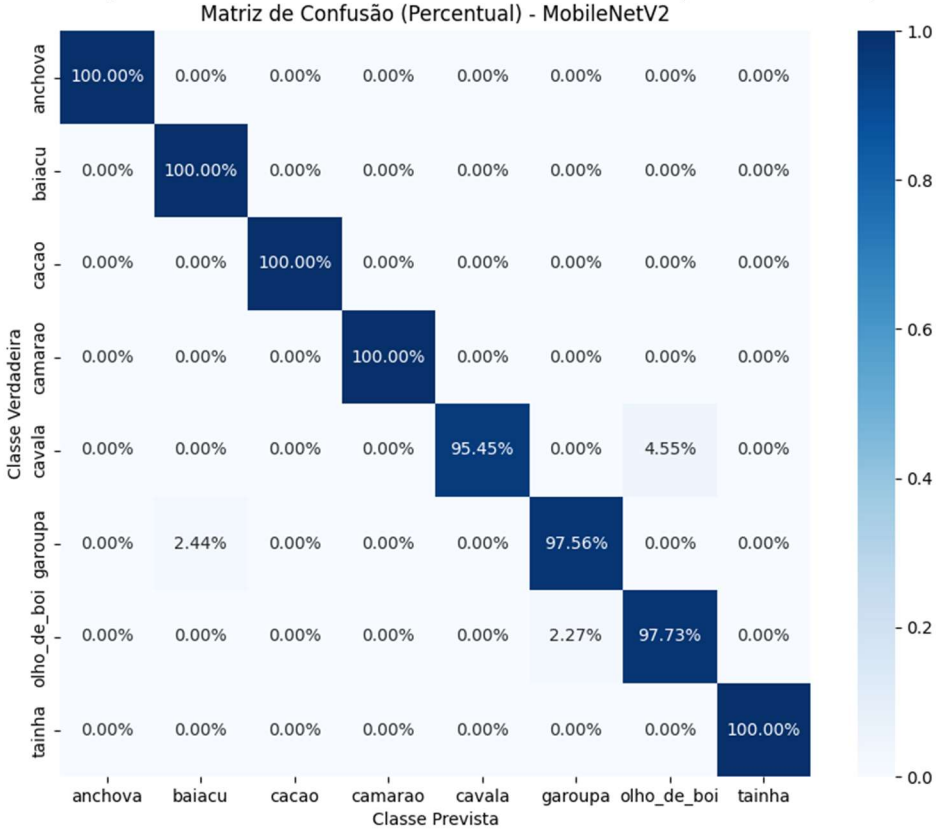
Para complementar os resultados sumarizados na Tabela 5, a Figura 17 apresenta o histórico de treinamento dos modelos. Na sequência, as Figura 18 (*MobileNetV2*) e Figura 19 (*ResNet50V2*) apresentam as matrizes de confusão.

Figura 17 – Histórico de Treinamento do teste 4



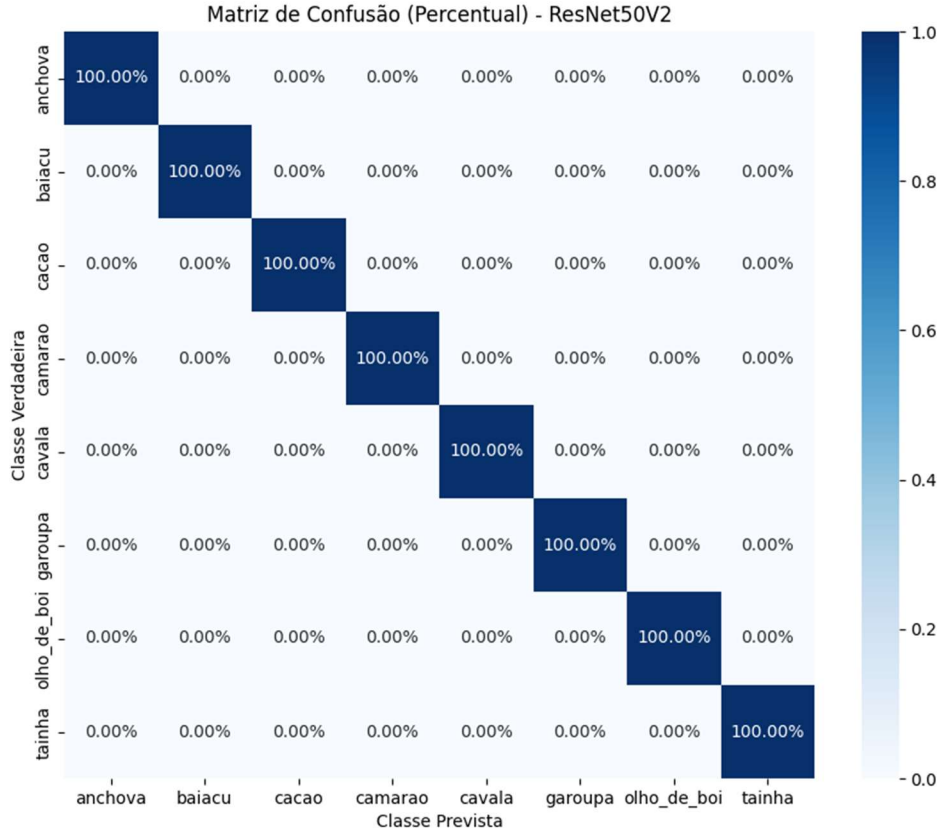
Fonte: elaborado pelo autor (2025).

Figura 18 – Matriz de confusão do teste 4 (*MobileNetV2*)



Fonte: elaborado pelo autor (2025).

Figura 19 – Matriz de confusão do teste 4 (*ResNet50V2*)



Fonte: elaborado pelo autor (2025).

A redução do tamanho do lote (*batch*) para 16 no Teste 4 levou o modelo *ResNet50V2* a um desempenho perfeito (100% de Acurácia e 1,0000 de *F1-Score*), superando os resultados dos testes anteriores. Isto é, este modelo conseguiu acertar todas as imagens do conjunto de teste. No entanto, este resultado não indica necessariamente o modelo mais eficiente. Um desempenho sem erros no conjunto de teste pode ser um indicador de sobreajuste (*overfitting*) ao *dataset* específico deste trabalho, o que sugere um risco de o modelo apresentar uma capacidade de generalização inferior ao analisar imagens externas, não vistas. Além disso, esta configuração elevou o custo computacional, apresentando tempos de treinamento e inferência (de aproximadamente 411s e 59ms, respectivamente) consideravelmente maiores que os registrados no Teste 1 (de aproximadamente 161s e 57ms).

5.5 SINTESE E COMPARAÇÃO COM ESTUDOS CORRELATOS

O Quadro 3 contextualiza os resultados deste trabalho frente aos estudos correlatos. A comparação com Srivastava (2023) é metodologicamente pertinente, pois, embora os conjuntos de dados e o número de classes sejam distintos, ambas as pesquisas implementaram as arquiteturas *ResNet* e *MobileNetV2.*

Quadro 3 – Comparação do modelo gerado com estudos correlatos

Estudo	Nº de imagens	Nº de espécies	Modelo	Acurácia	F1 Score
Srivastava (2023)	29.966	1 (madura ou imatura)	ResNet-50	0,805	0,820
			MobileNetV2	0,911	0,900
Este trabalho	963	8	*ResNet50V2*	1,000	1,000
			MobileNetV2	0,9929	0,9971

Fonte: elaborado pelo autor (2025).

A análise dos dados demonstra que os modelos desenvolvidos neste trabalho alcançaram métricas de acurácia elevadas (*ResNet50V2* atingindo 100% e *MobileNetV2* 99,29%). Em contraste, o estudo de Srivastava (2023), que empregou

estas arquiteturas para um *dataset* e um problema de classificação distintos (diferenciação de maturidade de trutas), obteve 80,5% e 90,6%, respectivamente. Sendo assim, os resultados deste trabalho mostram-se promissores. Este desempenho, alcançado mesmo com um *dataset* menor, sugere que a abordagem de *Transfer Learning* (pré-treinada no *ImageNet*) possui uma eficácia notável para a tarefa de classificação das espécies selecionadas.

O Quadro 4 estabelece uma segunda comparação com o trabalho de Iqbal *et al*. (2021), selecionado por apresentar um *dataset* com dimensões (1.334 imagens e 6 classes) mais próximas às do presente trabalho (963 imagens e 8 classes).

Quadro 4 – Comparação do modelo gerado com estudos correlatos

Estudo	**Nº de imagens**	**Nº de espécies**	**Modelo**	**Acurácia**
Iqbal *et al*. (2021)	1.334	6	Modelo próprio (CNN)	0,9048
Este trabalho	963	8	*ResNet50V2*	1,0000
			MobileNetV2	0,9929

Fonte: elaborado pelo autor (2025).

A análise demonstra que ambos os modelos deste trabalho (*ResNet50V2*: 100% e *MobileNetV2*: 99,29%) apresentaram métricas de acurácia elevadas. Em comparação, o modelo proposto por Iqbal *et al*. (2021) atingem 90,48% em seu *dataset*. Este resultado é particularmente relevante, pois sugere que a abordagem de *Transfer Learning*, adotada neste estudo, foi eficaz no treinamento para *datasets* desta magnitude.

6 CONCLUSÃO

O presente trabalho teve como objetivo geral desenvolver um modelo de *Deep Learning* (DL) voltado à classificação de imagens de peixes, com foco na identificação das espécies mais comuns na região de Florianópolis. A motivação central foi a valorização e preservação da cultura pesqueira local, propondo uma ferramenta tecnológica para a documentação e difusão do saber associado à biodiversidade marinha da Ilha de Santa Catarina.

Para alcançar este objetivo, foi estruturado um *dataset* próprio contendo 963 imagens de oito espécies de relevância cultural e econômica. Foram implementadas e comparadas duas arquiteturas de Redes Neurais Convolucionais, *MobileNetV2* e *ResNet50V2*, utilizando a técnica de *Transfer Learning*. O desenvolvimento foi conduzido integralmente no ambiente Google Colab, e os modelos foram avaliados em quatro testes principais, variando hiperparâmetros como épocas, divisão de dados e tamanho do lote.

Os objetivos específicos foram alcançados. Os resultados demonstraram a alta viabilidade da abordagem, com todos os testes apresentando acurácias superiores a 95% e atingindo desempenhos de 99,29% (Teste 1 por ambos os modelos) e 100% (Teste 4 pelo *ResNet*). A análise comparativa (Teste 1) indicou que, embora ambas as arquiteturas tenham atingido desempenho quase perfeito (99,29% de acurácia), o *MobileNetV2* (2,9M de parâmetros) é um modelo significativamente mais leve que o *ResNet50V2* (24,6M de parâmetros).

Embora o *ResNet50V2* tenha atingido 100% de acurácia no Teste 4 (com lote menor), ele o fez ao custo de um tempo de treinamento muito maior (411s contra 161s do Teste 1) e com potencial risco de sobreajuste (*overfitting*). O Teste 1 (70/15/15, 15 épocas) apresentou o melhor equilíbrio geral. O *MobileNetV2* se consolidou como a arquitetura mais eficiente, combinando alta acurácia (99,29%) com um tempo de inferência rápido (51ms) e um menor número de parâmetros, tornando-se a escolha ideal para uma eventual aplicação em dispositivos móveis.

7 LIMITAÇÕES E TRABALHOS FUTUROS

A principal limitação deste trabalho reside no conjunto de dados. Embora o modelo tenha alcançado alta acurácia, o *dataset* de 963 imagens é considerado pequeno para os padrões de *Deep Learning*. Além disso, as classes apresentam desbalanceamento (variando de 42 imagens de tainha a 294 de olho-de-boi). Outra limitação é o viés das imagens, muitas obtidas em peixarias (como a coleta própria da anchova) ou com fundo controlado, o que pode afetar a capacidade de generalização do modelo em outros contextos.

Convém salientar que a utilização de modelos pré-treinados em bases de dados genéricas (como o *ImageNet*) implica a adoção de extratores de características que não foram originalmente concebidos para a especificidade biológica das espécies aqui estudadas. Embora o treinamento de uma rede neural 'do zero' (*from scratch*) pudesse, teoricamente, resultar em filtros mais especializados para as particularidades morfológicas destes peixes, tal abordagem demandaria um volume massivo de dados rotulados. Diante das dificuldades inerentes à coleta de imagens em campo e da consequente restrição do *dataset*, a estratégia de *Transfer Learning* validou-se como a solução técnica mais eficaz, permitindo alcançar alta performance através do reaproveitamento de representações visuais previamente aprendidas, contornando a inviabilidade do treinamento inicial profundo.

Deve-se considerar, ainda, que o desempenho satisfatório obtido está atrelado a um escopo restrito de oito espécies. Considerando que o município de Florianópolis apresenta múltiplas espécies de animais marinhos, uma futura expansão do modelo (com mais imagens e classes) poderá exigir uma reavaliação das configurações atuais, que talvez não apresentem o mesmo desempenho.

Como trabalhos futuros, sugere-se a expansão e o balanceamento do *dataset*, aumentando significativamente o número de imagens, buscando um maior equilíbrio entre as classes e a inclusão de imagens em diferentes contextos. Ademais, sugere-se o desenvolvimento de uma aplicação móvel que integre o modelo para facilitar o acesso público, com o intuito de divulgar os saberes tradicionais locais. Esta etapa é importante para que o protótipo cumpra seu objetivo final de ferramenta de difusão cultural e educação ambiental, conforme proposto na justificativa deste trabalho.

REFERÊNCIAS

A LARGE SCALE FISH DATASET. **Kaggle**, 2020. 1 conjunto de dados. Disponível em: https://www.kaggle.com/datasets/crowww/a-large-scale-fish-dataset. Acesso em: 9 nov. 2025.

AMORIM, Leandro Barbosa. **Identificação de espécies de peixes e camarão utilizando técnicas de visão computacional e redes neurais convolucionais**. 2022. 33 f. Monografia (especialização) – Curso Pós-Graduação Lato Sensu em Engenharia Elétrica, Instituto Federal do Espírito Santo, Vitória, 2022.

BERGMANN, Dave. **What is machine learning?** IBM, 2025. Disponível em: https://www.ibm.com/think/topics/machine-learning. Acesso em: 9 dez. 2024.

BRASIL. Ministério do Meio Ambiente. **Plano de Recuperação da Garoupa-verdadeira (Epinephelus marginatus)**. Brasília, DF: MMA, 2018. Disponível em: https://www.gov.br/mma/pt-br/composicao/sbc/dpes/planos-de-recuperacao-para-especies-aquaticas-ameacadas-de-extincao/plano_de_recuperacao_da_garoupa-verdadeira.pdf. Acesso em: 3 set. 2025.

CARDOSO, Eduardo Schiavone. **Pescadores artesanais**: natureza, território, movimento social. 2001. 149 f. Tese (Doutorado) – Programa de Pós-Graduação em Geografia Física, Faculdade de Filosofia, Letras e Ciências Humanas, Universidade de São Paulo, São Paulo, 2001.

CHOLLET, François. **Deep Learning with Python**. Shelter Island: Manning Publications, 2018. 478 p.

DESTÉFANI, Homero Luiz. **Pesca e maricultura em Florianópolis, Santa Catarina, Brasil**: análise exploratória dos modos de vida e da percepção dos usuários sobre as atividades. 2017. 89 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Oceanografia, Centro de Filosofia e Ciências Humanas, Universidade Federal de Santa Catarina, Florianópolis, 2017.

FERNEDA, Edberto. Redes neurais e sua aplicação em sistemas de recuperação de informação. **Ciência da Informação**, v. 35, n. 1, p. 25–30, 2006. Disponível em: https://doi.org/10.1590/S0100-19652006000100003. Acesso em: 19 ago. 2025.

FISH DATASET. **Kaggle**, 2022. 1 conjunto de dados. Disponível em: https://www.kaggle.com/datasets/markdaniellampa/fish-dataset. Acesso em: 9 nov. 2025.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. Cambridge: MIT Press, 2016. Disponível em: http://www.deeplearningbook.org. Acesso em: 9 dez. 2025.

GOOGLE CLOUD. **O que é uma rede neural convolucional?** Google Cloud, 2025. Disponível em: https://cloud.google.com/discover/what-are-convolutional-neural-networks?hl=pt-BR. Acesso em: 7 ago. 2025.

HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep Residual Learning for Image Recognition. In: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 2016, Las Vegas. **Anais [...].** Las Vegas: Institute of Electrical and Electronics Engineers, 2016a. p. 770–778. Disponível em: https://ieeexplore.ieee.org/document/7780459. Acesso em: 8 out. 2025.

HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Identity Mappings in Deep Residual Networks. In: EUROPEAN CONFERENCE ON COMPUTER VISION (ECCV), 2016, Amsterdam. **Anais [...].** Cham: Springer, 2016b. p. 630-645. Disponível em: https://arxiv.org/abs/1603.05027. Acesso em: 8 out. 2025.

HOWARD, Andrew G.; ZHU, Menglong; CHEN, Bo; KALENICHENKO, Dmitry; WANG, Weijun; WEYAND, Tobias; ANDREETTO, Marco; HARTWIG, Adam. **MobileNets**: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint, 2017. Disponível em https://arxiv.org/pdf/1704.04861. Acesso em: 9 dez. 2024.

IBM. **O que é uma rede neural?** IBM, 2025. Disponível em: https://www.ibm.com/topics/neural-networks. Acesso em: 9 dez. 2024.

INDRAFISH. **Kaggle**, 2020. 1 conjunto de dados. Disponível em: https://www.kaggle.com/datasets/didiruh/longtail-tuna-thunnus-tonggol?resource=download&select=Grouper. Acesso em: 9 nov. 2025.

IQBAL, Muhammad Ather; WANG, Zhijie; ALI, Zain Anwar; RIAZ, Shazia. Automatic Fish Species Classification Using Deep Convolutional Neural Networks. **Wireless Personal Communications**, v. 116, p. 1043-1053, 2021. Disponível em: https://doi.org/10.1007/s11277-019-06634-1. Acesso em: 9 nov. 2025.

KÖCHE, José Carlos. **Fundamentos de metodologia científica**: teoria da ciência e iniciação à pesquisa. 34. ed. Petrópolis: Vozes, 2015. 182 p.

LEÃO, Lucas. **Peixes da Lagoa**. Florianópolis, 2010. Disponível em: https://pexesdalagoaumaova.blogspot.com/. Acesso em: 9 set. 2025.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **Nature**, v. 521, n. 7553, p. 436-444, 2015. Disponível em: https://www.nature.com/articles/nature14539. Acesso em: 8 jun. 2025.

LUDERMIR, Teresa Bernarda. Inteligência Artificial e Aprendizado de Máquina: estado atual e tendências. **Estudos Avançados**, v. 35, n. 101, p. 85–94, 2021. Disponível em: https://doi.org/10.1590/s0103-4014.2021.35101.007. Acesso em: 20 set. 2025.

MASTERS, Dominic; LUSCHI, Carlo. **Revisiting Small Batch Training for Deep Neural Networks**. arXiv, 2018. Disponível em: https://arxiv.org/abs/1804.07612. Acesso em: 9 dez. 2025.

MATHWORKS. **Convolutional Neural Network**. MathWorks, 2025. Disponível em:

https://www.mathworks.com/discovery/convolutional-neural-network.html. Acesso em: 7 jul. 2025.

MEDEIROS, Rodrigo Pereira. **Estratégias de pesca e usos dos recursos em uma comunidade de pescadores artesanais da Praia do Pântano do Sul (Florianópolis, Santa Catarina)**. 2001. 108 f. Dissertação (Mestrado) – Instituto de Biologia, Universidade Estadual de Campinas, Campinas, 2001.

MURAINA, Ismail Olaniyi. Ideal Dataset Splitting Ratios in Machine Learning Algorithms: General Concerns for Data Scientists and Data Analysts. In: INTERNATIONAL MARDIN ARTUKLU SCIENTIFIC RESEARCHES CONFERENCE, 7., 2022, Mardin, Turquia. **Anais [...].** Mardin: Artuklu, 2022. Disponível em: https://www.researchgate.net/publication/358284895. Acesso em: 9 dez. 2025.

PINHO, Ricardo. A pesca artesanal na Baía Sul da Ilha de Santa Catarina: um patrimônio da cultura local. **Revista Confluências Culturais**, Joinville, v. 5, n. 2, p. 9–28, 2016. Disponível em: https://periodicos.univille.br/RCC/article/view/370. Acesso em: 8 jun. 2025.

PORTAL CATARINAS. **Ilha invisível**: os desafios dos pescadores tradicionais da Costeira, em Florianópolis. Portal Catarinas, 2021. Disponível em: https://catarinas.info/ilha-invisivel-os-desafios-dos-pescadores-tradicionais-da-costeira-em-florianopolis/#:~:text=O%20n%C3%BAmero%20de%20ranchos%20de,que%20a%20mar%C3%A9%20est%C3%A1%20baixa. Acesso em: 3 dez. 2024.

RATHI, Dhruv; JAIN, Sushant; INDU, Sreedevi. Underwater Fish Species Classification using Convolutional Neural Network and Deep Learning. In: INTERNATIONAL CONFERENCE ON ADVANCES IN PATTERN RECOGNITION (ICAPR), 9., 2017, Bangalore. **Anais [...].** Bangalore: Institute of Electrical and Electronics Engineers, 2017. p. 1-6. DOI: 10.1109/icapr.2017.8593044. Acesso em: 11 dez. 2024.

RIBAS, Liz Cristina Camargo (Org.). **Que peixe é este?** O sabor da pesca artesanal na Ilha de Santa Catarina. Florianópolis: Publicação do IFSC, 2016. 350 p.

SANDLER, Mark; HOWARD, Andrew; ZHU, Menglong; ZHMOGINOV, Andrey; CHEN, Liang-Chieh. *MobileNetV2*: Inverted Residuals and Linear Bottlenecks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, Salt Lake City. **Anais [...].** Salt Lake City: Institute of Electrical and Electronics Engineers, 2018. p. 4510-4520. Disponível em: https://arxiv.org/abs/1801.04381. Acesso em: 10 out. 2025.

SANTA CATARINA (Estado). Secretaria de Estado da Aquicultura e Pesca. **Portaria garante regras para pesca artesanal de camarão nas baías da Ilha de Santa Catarina.** Florianópolis: SAQ, 2025. Disponível em: https://www.saq.sc.gov.br/portaria-garante-regras-para-pesca-artesanal-de-camarao-nas-baias-da-ilha-de-santa-catarina/. Acesso em: 20 out. 2025.

SHARK SPECIES. **Kaggle**, 2020. 1 conjunto de dados. Disponível em: https://www.kaggle.com/datasets/larusso94/shark-species. Acesso em: 9 nov. 2025.

SHINDE, Pramila P.; SHAH, Seema. A Review of Machine Learning and Deep Learning Applications. In: INTERNATIONAL CONFERENCE ON COMPUTING COMMUNICATION CONTROL AND AUTOMATION (ICCUBEA), 4., 2018, Pune. **Anais [...].** Pune: Institute of Electrical and Electronics Engineers, 2018, p. 1-6. Disponível em: https://ieeexplore.ieee.org/abstract/document/8697857. Acesso em: 2 out. 2025.

SRIVASTAVA, Varun. **Classification of Fish Species Using Deep Learning Models**. 2023. 158 f. Dissertação (Mestrado em Ciência da Computação Aplicada) – Department of Computer Science, Faculty of Information Technology and Electrical Engineering, Norwegian University of Science and Technology, Noruega, 2023.

WEISS, Karl; KHOSHGOFTAAR, Taghi M.; WANG, DingDing. A survey of transfer learning. **Journal of Big Data**, v. 3, n. 9, p. 1-40, 2016. Disponível em: https://doi.org/10.1186/s40537-016-0043-6. Acesso em: 10 out. 2025.

WIKIPEDIA. **Representação de uma rede de neurônios artificial**. Wikipédia, 2013. 1 imagem (PNG). Disponível em: https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/Colored_neural_network.svg/290px-Colored_neural_network.svg.png. Acesso em: 10 out. 2025.

APÊNDICE A – CÓDIGO FONTE DESENVOLVIDO

```
# -*- coding: utf-8 -*-
"""
TCC - Desenvolvimento de um Modelo de Classificação de Imagens de
Peixes de Florianópolis usando Deep Learning
"""

# 1. IMPORTAÇÃO DE BIBLIOTECAS

import os
import shutil
import glob
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Model
from  tensorflow.keras.layers  import  Dense,  GlobalAveragePooling2D,
Dropout
from tensorflow.keras.applications import MobileNetV2, ResNet50V2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    f1_score,
    roc_curve,
    auc,
)
from sklearn.preprocessing import LabelBinarizer

# Suprimir avisos de log do TensorFlow
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
tf.get_logger().setLevel("ERROR")

# 2. CONFIGURAÇÕES GLOBAIS


# Caminho para a pasta com imagens originais
PATH_DADOS_ORIGINAIS = r"/"

# Caminho para onde o dataset organizado será criado ou está montado
PATH_DATASET_ORGANIZADO = r"/"

# Mudar para True apenas na PRIMEIRA vez que rodar para organizar os
arquivos
#    Depois de organizar, mantenha como False.
EXECUTAR_ORGANIZACAO_DATASET = True
```

```
# Mudar para False se modelo ja estiver treinado e quiser avaliar novo
banco de imagens
TREINAR_NOVAMENTE = True


# Caminho para modelo de treinamento MobileNet
PATH_MODELO_MOBILENET = r"/"

# Caminho para modelo de treinamento ResNet
PATH_MODELO_RESNET = r"/"

# Definições das classes
CLASSES = [
    "anchova",
    "baiacu",
    "cacao",
    "camarao",
    "cavala",
    "garoupa",
    "olho_de_boi",
    "tainha",
]
NUM_CLASSES = len(CLASSES)

# Definições do Modelo
IMG_WIDTH = 224
IMG_HEIGHT = 224
IMG_SHAPE = (IMG_WIDTH, IMG_HEIGHT, 3) #Indica 3 canais de cor
BATCH_SIZE = 16
EPOCHS = 15


# Definições da divisão dos dados
TEST_SPLIT = 0.15
VALIDATION_SPLIT = 0.15

# 3. FUNÇÕES DE PRÉ-PROCESSAMENTO

def organizar_dataset(path_origem, path_destino, classes, test_split,
validation_split):

    print(f"Iniciando organização do dataset...")
    print(f"Origem: {path_origem} (lendo subpastas de classes)")
    print(f"Destino: {path_destino}")

    if not os.path.exists(path_origem):
        print(f"ERRO: O caminho de origem '{path_origem}' não existe.
Verifique a variável 'PATH_DADOS_ORIGINAIS'.")
        return False

    print(f"Limpando e criando diretórios de destino em
{path_destino}...")
    if os.path.exists(path_destino):
        shutil.rmtree(path_destino)

    # Criar estruturas de pasta de destino (train/val/test)
```

```
    path_train = os.path.join(path_destino, "train")
    path_val = os.path.join(path_destino, "validation")
    path_test = os.path.join(path_destino, "test")

    for path in [path_train, path_val, path_test]:
        for classe in classes:
            # Garante que as pastas de destino usem o nome padronizado
(com underscore)
            os.makedirs(os.path.join(path, classe), exist_ok=True)

    print("Procurando imagens nas subpastas de classes (jpg, jpeg,
png)...")
    extensions = ("*.jpg", "*.jpeg", "*.png")
    contadores = {classe: 0 for classe in classes}

    # Encontrar todas as subpastas em path_origem
    try:
        pastas_classes_origem = [d for d in os.listdir(path_origem) if
os.path.isdir(os.path.join(path_origem, d))]
    except FileNotFoundError:
        print(f"ERRO: Não foi possível listar diretórios em
'{path_origem}'. Verifique o caminho.")
        return False

    if not pastas_classes_origem:
        print(f"ERRO: Nenhuma subpasta de classe encontrada em
'{path_origem}'.")
        print("A estrutura esperada é: '.../imagens/anchova/',
'.../imagens/baiacu/', etc.")
        return False


    # Iterar sobre cada pasta de classe encontrada na origem
    for nome_pasta in pastas_classes_origem:
        # Normalizar o nome da pasta
        classe_identificada = nome_pasta.lower().replace("-", "_")

        # Verificar se esta pasta é uma das classes que queremos
processar
        if classe_identificada not in classes:
            print(f"Aviso: Ignorando pasta '{nome_pasta}' pois não está
na lista de classes-alvo.")
            continue

        print(f"Processando classe: '{classe_identificada}' (da pasta
'{nome_pasta}')")
        path_classe_origem = os.path.join(path_origem, nome_pasta)

        # Encontrar todos os arquivos de imagem para esta classe
        class_files = []
        for ext in extensions:
            class_files.extend(glob.glob(os.path.join(path_classe_orige
m, ext)))

        if not class_files:
            print(f"   Aviso: Nenhuma imagem (.jpg, .jpeg, .png)
encontrada em '{path_classe_origem}'.")
```

```
            continue

        # Misturar os arquivos *desta classe* de forma aleatória
        np.random.shuffle(class_files)

        # Calcular os tamanhos das divisões
        total_images_in_class = len(class_files)
        num_test = int(total_images_in_class * test_split)
        num_val = int(total_images_in_class * validation_split)
        num_train = total_images_in_class - num_test - num_val

        # Dividir os arquivos da classe
        test_files = class_files[:num_test]
        val_files = class_files[num_test : num_test + num_val]
        train_files = class_files[num_test + num_val :]

        # Copiar arquivos para os diretórios de destino
        for file_path in train_files:
            contadores[classe_identificada] += 1
            novo_nome                                                    =
f"{classe_identificada}_{contadores[classe_identificada]:04d}{os.path.s
plitext(file_path)[1]}"
            shutil.copy(file_path,              os.path.join(path_train,
classe_identificada, novo_nome))

        for file_path in val_files:
            contadores[classe_identificada] += 1
            novo_nome                                                    =
f"{classe_identificada}_{contadores[classe_identificada]:04d}{os.path.s
plitext(file_path)[1]}"
            shutil.copy(file_path,                os.path.join(path_val,
classe_identificada, novo_nome))

        for file_path in test_files:
            contadores[classe_identificada] += 1
            novo_nome                                                    =
f"{classe_identificada}_{contadores[classe_identificada]:04d}{os.path.s
plitext(file_path)[1]}"
            shutil.copy(file_path,              os.path.join(path_test,
classe_identificada, novo_nome))

    print("\n--- Organização Concluída! ---")
    print("Contagem final de imagens copiadas por classe:")
    print(pd.Series(contadores))

    classes_sem_imagens = [cls for cls, count in contadores.items() if
count == 0]
    if classes_sem_imagens:
        print(f"\nAviso:  As  seguintes  classes  não  tiveram  nenhuma
imagem encontrada: {classes_sem_imagens}")

    print("---------------------------------")
    return True

# 4. FUNÇÕES DE MODELAGEM E AVALIAÇÃO 1
```

```
def criar_geradores_de_dados(base_path, img_size, batch_size):
    """
    Cria os geradores de dados (DataLoaders) para treino, validação e
teste.
    """
    train_dir = os.path.join(base_path, "train")
    validation_dir = os.path.join(base_path, "validation")
    test_dir = os.path.join(base_path, "test")

    # Data Augmentation para o conjunto de treino
    train_datagen = ImageDataGenerator(
        rescale=1.0 / 255,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        horizontal_flip=True,
        fill_mode="nearest",
    )

    # Para validação e teste, apenas normalizamos (NÃO usamos
augmentation)
    test_val_datagen = ImageDataGenerator(rescale=1.0 / 255)

    # Gerador de dados de Treino
    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode="categorical",
        shuffle=True,
    )

    # Gerador de dados de Validação
    validation_generator = test_val_datagen.flow_from_directory(
        validation_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode="categorical",
        shuffle=False,  # Não misturar validação
    )

    # Gerador de dados de Teste
    test_generator = test_val_datagen.flow_from_directory(
        test_dir,
        target_size=img_size,
        batch_size=1,  # Batch size 1 para facilitar avaliação imagem a
imagem
        class_mode="categorical",
        shuffle=False,  # Fundamental não misturar o teste!
    )

    print("\n--- Documentação da Base de Dados ---")
    print(f"Resolução de entrada (Input Resolution):
{img_size[0]}x{img_size[1]} pixels")
```

```
    print(f"Classes:                                        {NUM_CLASSES}
{list(train_generator.class_indices.keys())}")
    print(f"Total de imagens de TREINO: {train_generator.samples}")
    print(f"Total        de        imagens        de        VALIDAÇÃO:
{validation_generator.samples}")
    print(f"Total de imagens de TESTE: {test_generator.samples}")
    print("-------------------------------------")

    return train_generator, validation_generator, test_generator


def criar_modelo_baseado_em(nome_base, input_shape, num_classes):
    """
    Cria um modelo de classificação usando Transfer Learning.
    nome_base: 'MobileNetV2' ou 'ResNet50V2'
    """
    keras.backend.clear_session()

    # 1. Carregar o Modelo Base (pré-treinado no ImageNet)
    if nome_base == "MobileNetV2":
        base_model = MobileNetV2(
            input_shape=input_shape,                include_top=False,
weights="imagenet"
        )
    elif nome_base == "ResNet50V2":
        base_model = ResNet50V2(
            input_shape=input_shape,                include_top=False,
weights="imagenet"
        )
    else:
        raise ValueError(f"Modelo base '{nome_base}' não suportado.")

    # Congelar as camadas do modelo base
    base_model.trainable = False

    # 2. Adicionar nosso "Topo" (Camadas de Classificação)
    inputs = keras.Input(shape=input_shape)
    x = base_model(inputs, training=False)
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)  # Regularização
    x = Dense(512, activation="relu")(x)
    outputs = Dense(num_classes, activation="softmax")(x)  # Softmax
para N classes
    modelo = Model(inputs, outputs)

    # 3. Compilar o Modelo
    modelo.compile(
        optimizer=keras.optimizers.Adam(learning_rate=0.001),
        loss="categorical_crossentropy",
        metrics=["accuracy"],
    )

    return modelo


def plotar_historia_treinamento(history, nome_modelo):
    """
```

```
    Plota e salva as curvas de Acurácia e Loss do treino.
    """
    print(f"Gerando gráficos de histórico para {nome_modelo}...")
    acc = history.history["accuracy"]
    val_acc = history.history["val_accuracy"]
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs_range = range(len(acc))

    plt.figure(figsize=(14, 5))
    plt.suptitle(f"Histórico de Treinamento - {nome_modelo}",
fontsize=16)

    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label="Acurácia (Treino)")
    plt.plot(epochs_range, val_acc, label="Acurácia (Validação)")
    plt.legend(loc="lower right")
    plt.title("Acurácia")
    plt.xlabel("Épocas")

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label="Loss (Treino)")
    plt.plot(epochs_range, val_loss, label="Loss (Validação)")
    plt.legend(loc="upper right")
    plt.title("Loss (Perda)")
    plt.xlabel("Épocas")

    filename = f"grafico_historia_{nome_modelo}.png"
    plt.savefig(filename)
    print(f"Gráfico salvo em: {filename}")
    # plt.show() # Descomente se quiser visualização interativa
    plt.close()


def avaliar_modelo_completo(modelo_path, nome_modelo, test_gen):
    """
    Carrega o melhor modelo salvo e roda TODAS as métricas de avaliação
    solicitadas no conjunto de TESTE.
    """
    print(f"\n--- AVALIAÇÃO COMPLETA: {nome_modelo} ---")

    # 1. Carregar o melhor modelo salvo pelo checkpoint
    modelo = keras.models.load_model(modelo_path)

    # 2. Obter previsões e rótulos verdadeiros do conjunto de teste
    test_gen.reset()
    class_names = list(test_gen.class_indices.keys())

    # Medir tempo de inferência (Anotação TCC: "tempo para teste")
    start_time_inference = time.time()
    y_pred_probs = modelo.predict(test_gen, steps=test_gen.samples,
verbose=1)
    end_time_inference = time.time()

    tempo_inferencia_total = end_time_inference - start_time_inference
    tempo_inferencia_por_imagem = tempo_inferencia_total /
test_gen.samples
```

```
    # Converter probabilidades (softmax) para a classe prevista
    y_pred_indices = np.argmax(y_pred_probs, axis=1)
    # Obter os rótulos verdadeiros
    y_true_indices = test_gen.classes

    # --- 3. Métrica: Acurácia, F1-Score, Sensibilidade (Relatório) ---
    print(f"\n[Relatório de Classificação - {nome_modelo}]")
    # 'recall' no scikit-learn é o mesmo que 'sensibilidade'
    report_dict = classification_report(
        y_true_indices,    y_pred_indices,    target_names=class_names,
output_dict=True
    )
    report_str = classification_report(
        y_true_indices,    y_pred_indices,    target_names=class_names,
digits=4
    )
    print(report_str)

    # Salvar o relatório de classificação completo em um CSV
    df_report = pd.DataFrame(report_dict).transpose()
    report_filename = f"classification_report_{nome_modelo}.csv"
    df_report.to_csv(report_filename)
    print(f"Relatório de classificação salvo em: {report_filename}")

    # Salvar métricas para a tabela comparativa
    metricas = {
        "Acurácia": accuracy_score(y_true_indices, y_pred_indices),
        "F1-Score  (Macro)":  f1_score(y_true_indices,  y_pred_indices,
average="macro"),
        "Tempo  Inferência  (ms/img)":  tempo_inferencia_por_imagem  *
1000,
    }

    # --- 4. Métrica: Matriz de Confusão (Absoluta e Percentual) ---
    print("Gerando Matrizes de Confusão...")
    cm = confusion_matrix(y_true_indices, y_pred_indices)

    # Absoluta
    plt.figure(figsize=(10, 8))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=class_names,
        yticklabels=class_names,
    )
    plt.title(f"Matriz de Confusão (Absoluta) - {nome_modelo}")
    plt.ylabel("Classe Verdadeira")
    plt.xlabel("Classe Prevista")
    filename_abs = f"matriz_confusao_abs_{nome_modelo}.png"
    plt.savefig(filename_abs)
    print(f"Matriz salva em: {filename_abs}")
    # plt.show()
    plt.close()
```

```
    # Percentual (Normalizada pela linha - % de acerto por classe
verdadeira)
    cm_percent = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
    plt.figure(figsize=(10, 8))
    sns.heatmap(
        cm_percent,
        annot=True,
        fmt=".2%",
        cmap="Blues",
        xticklabels=class_names,
        yticklabels=class_names,
    )
    plt.title(f"Matriz de Confusão (Percentual) - {nome_modelo}")
    plt.ylabel("Classe Verdadeira")
    plt.xlabel("Classe Prevista")
    filename_perc = f"matriz_confusao_perc_{nome_modelo}.png"
    plt.savefig(filename_perc)
    print(f"Matriz salva em: {filename_perc}")
    # plt.show()
    plt.close()

    # --- 5. Métrica: Curva ROC (One-vs-Rest) ---

    #Descomentar Linhas se for pegar curva ROC, não utilizada na versão
final
    # print("Gerando Curva ROC...")
    # Binarizar os rótulos para a abordagem OvR (One-vs-Rest)
    # lb = LabelBinarizer()
    # y_true_bin = lb.fit_transform(y_true_indices)

    # plt.figure(figsize=(10, 8))
    # fpr = dict()
    # tpr = dict()
    # roc_auc = dict()

    # for i in range(NUM_CLASSES):
    #           fpr[i],   tpr[i],   _   =   roc_curve(y_true_bin[:,   i],
y_pred_probs[:, i])
    #     roc_auc[i] = auc(fpr[i], tpr[i])

    # Plotar todas as curvas ROC
    # colors = plt.cm.get_cmap("tab10", NUM_CLASSES)
    #      for      i,      color      in      zip(range(NUM_CLASSES),
colors(range(NUM_CLASSES))):
    #     plt.plot(
    #         fpr[i],
    #         tpr[i],
    #         color=color,
    #         lw=2,
    #            label=f"Curva  ROC  da  classe  {class_names[i]}  (AUC  =
{roc_auc[i]:0.2f})",
    #     )

    # plt.plot([0, 1], [0, 1], "k--", lw=2)  # Linha de chance
    # plt.xlim([0.0, 1.0])
    # plt.ylim([0.0, 1.05])
    # plt.xlabel("Taxa de Falsos Positivos (FPR)")
    # plt.ylabel("Taxa de Verdadeiros Positivos (TPR) / Sensibilidade")
    # plt.title(f"Curva ROC (One-vs-Rest) - {nome_modelo}")
```

```
    # plt.legend(loc="lower right", fontsize="small")

    # filename_roc = f"curva_roc_{nome_modelo}.png"
    # plt.savefig(filename_roc)
    # print(f"Curva ROC salva em: {filename_roc}")
    # plt.show()
    # plt.close()

    print(f"--- Avaliação de {nome_modelo} concluída ---")
    return metricas

# 5. BLOCO DE EXECUÇÃO PRINCIPAL (MAIN)

def main():
    """
    Função principal que orquestra todo o pipeline do TCC.
    """
    print("=" * 60)
    print("INICIANDO SCRIPT DE TREINAMENTO E AVALIAÇÃO - TCC PEIXES")
    print("=" * 60)

    # --- PASSO 1: Organizar o Dataset (Apenas se
EXECUTAR_ORGANIZACAO_DATASET = True) ---
    if EXECUTAR_ORGANIZACAO_DATASET:
        print("\n[PASSO 1: ORGANIZANDO DATASET]")
        sucesso = organizar_dataset(
            PATH_DADOS_ORIGINAIS,
            PATH_DATASET_ORGANIZADO,
            CLASSES,
            TEST_SPLIT,
            VALIDATION_SPLIT,
        )
        if not sucesso:
            print("\nFalha na organização do dataset. Abortando.")
            return
        print("\nOrganização concluída. Mude
'EXECUTAR_ORGANIZACAO_DATASET' para False na próxima execução.")
    else:
        print("\n[PASSO 1: ORGANIZANDO DATASET]")
        print("Pulando organização (EXECUTAR_ORGANIZACAO_DATASET =
False).")
        if not os.path.exists(os.path.join(PATH_DATASET_ORGANIZADO,
"train")):
            print(f"ERRO: Pasta 'train' não encontrada em
'{PATH_DATASET_ORGANIZADO}'.")
            print("Mude 'EXECUTAR_ORGANIZACAO_DATASET' para True para
criar o dataset.")
            return

    # --- PASSO 2: Criar Geradores de Dados ---
    print("\n[PASSO 2: CARREGANDO DADOS]")
    try:
        train_gen, val_gen, test_gen = criar_geradores_de_dados(
            PATH_DATASET_ORGANIZADO, (IMG_WIDTH, IMG_HEIGHT),
BATCH_SIZE
        )
    except FileNotFoundError:
```

```
        print(f"ERRO: Não foi possível encontrar as pastas 'train', 'validation' ou 'test' em '{PATH_DATASET_ORGANIZADO}'.")
        print("Verifique o caminho 'PATH_DATASET_ORGANIZADO' ou rode a organização (Passo 1).")
        return

    # Dicionários para armazenar resultados
    historicos = {}
    tempos_treino = {}
    metricas_finais = {}
    num_parametros = {}

    # Callbacks em 7
    early_stop = EarlyStopping(
        monitor="val_loss", patience=7, restore_best_weights=True, verbose=1
    )

    #Verificar se modelo ja existe

    if TREINAR_NOVAMENTE:
      print('"\n[INICIANDO TREINAMENTO DE MODELOS]"')

      # --- PASSO 3: Treinamento - Modelo 1 (MobileNetV2) ---
      print("\n[PASSO 3: TREINAMENTO - MODELO 1: MobileNetV2]")
      modelo_mobilenet = criar_modelo_baseado_em("MobileNetV2", IMG_SHAPE, NUM_CLASSES)
      num_parametros["MobileNetV2"] = modelo_mobilenet.count_params()
      print(f"MobileNetV2 - Número de Parâmetros: {num_parametros['MobileNetV2']}")


      checkpoint_mobilenet = ModelCheckpoint(
          "modelo_MobileNetV2_best.h5",
          monitor="val_accuracy",
          save_best_only=True,
          mode="max",
          verbose=1,
      )

      start_time = time.time()
      historicos["MobileNetV2"] = modelo_mobilenet.fit(
          train_gen,
          epochs=EPOCHS,
          validation_data=val_gen,
          callbacks=[checkpoint_mobilenet, early_stop],
      )
      tempos_treino["MobileNetV2"] = time.time() - start_time
      print(f"Tempo total de treinamento (MobileNetV2): {tempos_treino['MobileNetV2']:.2f} segundos")

      # --- PASSO 4: Treinamento - Modelo 2 (ResNet50V2) ---
      print("\n[PASSO 4: TREINAMENTO - MODELO 2: ResNet50V2]")
      modelo_resnet = criar_modelo_baseado_em("ResNet50V2", IMG_SHAPE, NUM_CLASSES)
      num_parametros["ResNet50V2"] = modelo_resnet.count_params()
```

```
        print(f"ResNet50V2 - Número de Parâmetros: {num_parametros['ResNet50V2']}")

        checkpoint_resnet = ModelCheckpoint(
            "modelo_ResNet50V2_best.h5",
            monitor="val_accuracy",
            save_best_only=True,
            mode="max",
            verbose=1,
        )

        start_time = time.time()
        historicos["ResNet50V2"] = modelo_resnet.fit(
            train_gen,
            epochs=EPOCHS,
            validation_data=val_gen,
            callbacks=[checkpoint_resnet, early_stop],
        )
        tempos_treino["ResNet50V2"] = time.time() - start_time
        print(f"Tempo total de treinamento (ResNet50V2): {tempos_treino['ResNet50V2']:.2f} segundos")

    else:
        print("\n[MODO: APENAS AVALIAÇÃO (Treinamento pulado)]")
        tempos_treino["MobileNetV2"] = 0
        tempos_treino["ResNet50V2"] = 0

        # Verificação de segurança: os arquivos .h5 DEVEM existir
        modelos_existem = os.path.exists(PATH_MODELO_MOBILENET) and os.path.exists(PATH_MODELO_RESNET)

        if not modelos_existem:
            print("ERRO: `TREINAR_NOVAMENTE` está False, mas os arquivos 'modelo_..._best.h5' não foram encontrados.")
            print("Mude `TREINAR_NOVAMENTE` para True para treinar os modelos primeiro, ou suba os arquivos .h5.")
            return
        print("Modelos .h5 encontrados. Prosseguindo direto para a avaliação...")

        # Carrega modelos apenas para contar parâmetros se o treinamento foi pulado
        if os.path.exists(PATH_MODELO_MOBILENET):
            dummy_model_mb = keras.models.load_model(PATH_MODELO_MOBILENET)
            num_parametros["MobileNetV2"] = dummy_model_mb.count_params()
        else:
            num_parametros["MobileNetV2"] = "N/A"

        if os.path.exists(PATH_MODELO_RESNET):
            dummy_model_rn = keras.models.load_model(PATH_MODELO_RESNET)
            num_parametros["ResNet50V2"] = dummy_model_rn.count_params()
        else:
            num_parametros["ResNet50V2"] = "N/A"
```

```
    # --- PASSO 5: Avaliação dos Modelos ---
    print("\n[PASSO 5: AVALIAÇÃO DOS MODELOS NO CONJUNTO DE TESTE]")

    # Plotar histórico apenas se o treinamento foi executado
    if TREINAR_NOVAMENTE:
      plotar_historia_treinamento(historicos["MobileNetV2"],
"MobileNetV2")
      plotar_historia_treinamento(historicos["ResNet50V2"],
"ResNet50V2")

    # Avaliação MobileNetV2
    metricas_finais["MobileNetV2"] = avaliar_modelo_completo(
        PATH_MODELO_MOBILENET, "MobileNetV2", test_gen
    )

    # Avaliação ResNet50V2
    metricas_finais["ResNet50V2"] = avaliar_modelo_completo(
        PATH_MODELO_RESNET, "ResNet50V2", test_gen
    )

    # --- PASSO 6: Geração da Tabela Comparativa Final ---
    print("\n[PASSO 6: TABELA COMPARATIVA FINAL DE RESULTADOS]")

    dados_comparacao = {}
    for nome_modelo in ["MobileNetV2", "ResNet50V2"]:
        if nome_modelo in metricas_finais:
            dados_comparacao[nome_modelo] = {
                "Acurácia": metricas_finais[nome_modelo]["Acurácia"],
                "F1-Score   (Macro)":   metricas_finais[nome_modelo]["F1-
Score (Macro)"],
                "Tempo de Treino (s)": tempos_treino[nome_modelo],
                "Tempo          de          Inferência          (ms/img)":
metricas_finais[nome_modelo]["Tempo Inferência (ms/img)"],
                "Número de Parâmetros": num_parametros[nome_modelo]
            }

    df_comparacao       =       pd.DataFrame.from_dict(dados_comparacao,
orient="index")
    df_comparacao = df_comparacao.round(4)

    print("--- Tabela Comparativa dos Modelos Implementados (Resultados
no Teste) ---")
    print(df_comparacao.to_markdown(floatfmt=".4f"))

    # Salva tabela em CSV
    df_comparacao.to_csv("comparacao_modelos_tcc.csv")
    print("\nTabela        de        comparação        salva        em
'comparacao_modelos_tcc.csv'")

    print("\n" + "=" * 60)
    print("EXECUÇÃO DO SCRIPT CONCLUÍDA")
    print("Verifique os arquivos .png e .csv gerados na pasta do
script.")
    print("=" * 60)
```

```
if __name__ == "__main__":
    main()

import os
import glob
import shutil
from google.colab import drive

# 1. Monta o Google Drive no ambiente do Colab
try:
    drive.mount('/content/drive')
except Exception as e:
    print(f"Erro ao montar o drive: {e}")

# 2. Define a pasta de destino
destination_folder = '/'

# 3. Criar a pasta de destino, se ela não existir
os.makedirs(destination_folder, exist_ok=True)
print(f"Pasta de destino: {destination_folder}")

# 4. Encontrar todos os arquivos .h5, .png e .csv na pasta atual
files_to_save = []
files_to_save.extend(glob.glob('*.h5'))
files_to_save.extend(glob.glob('*.png'))
files_to_save.extend(glob.glob('*.csv'))

if not files_to_save:
    print("Nenhum arquivo (.h5, .png, .csv) encontrado para salvar.")
else:
    print(f"Arquivos encontrados para salvar: {files_to_save}")

    # 5. Copia cada arquivo para o Google Drive
    for file_path in files_to_save:
        try:
            shutil.copy(file_path, destination_folder)
            print(f"Copiado com sucesso: {file_path}")
        except Exception as e:
            print(f"ERRO ao copiar {file_path}: {e}")

print("\nBackup no Google Drive concluído!")
```

APÊNDICE B – ARTIGO NO FORMATO SBC

Desenvolvimento de um modelo de classificação de imagens de peixes de Florianópolis usando *Deep Learning*

Marcus Augusto Demétrio Pinheiro[1], Alexandre Gonçalves Silva[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis – SC – Brasil

marcus.a.d.p@grad.ufsc.br, alexandre.silva@inf.ufsc.br

Abstract. *Given the progressive disappearance of the traditional knowledge (saber-fazer) of artisanal fishing in Florianópolis/SC, this work aimed to develop a Deep Learning model for the classification of images of fish and other marine animals, with the goal of documenting this cultural heritage. Using Transfer Learning, the MobileNetV2 and ResNet50V2 architectures were compared in classifying a proprietary dataset (963 images, 8 species). The results achieved high accuracy (exceeding 95%). It is concluded that MobileNetV2 is the most efficient architecture, being significantly lighter, making it ideal for future mobile applications.*

Resumo. *Diante do progressivo desaparecimento do saber-fazer da pesca artesanal em Florianópolis/SC, este trabalho objetivou desenvolver um modelo de Deep Learning para a classificação de imagens de peixes e outros animais marinhos, visando a documentação deste patrimônio cultural. Utilizando Transfer Learning, comparou-se as arquiteturas MobileNetV2 e ResNet50V2 na classificação de um dataset próprio (963 imagens, 8 espécies). Os resultados alcançaram alta acurácia (superior a 95%). Conclui-se que o MobileNetV2 é a arquitetura mais eficiente, por ser significativamente mais leve, tornando-se ideal para futuras aplicações móveis.*

1. Introdução

A pesca é um saber cultural e prático que remonta a tempos pré-históricos no litoral de Santa Catarina. Historicamente, a região foi marcada pela exploração de recursos marinhos (Destéfani, 2017; Medeiros, 2001; Pinho, 2016). No século XVIII, a pesca da baleia impulsionou a economia, mas declinou no final do século. Em contrapartida, a pesca artesanal, praticada em comunidades e articulada com a agricultura, ganhou destaque, consolidando um modo de vida açoriano (Medeiros, 2001; Pinho, 2016). Esse modo de vida passou a ser inviabilizado por pressões como a expansão imobiliária e a pesca industrial, levando a um declínio que persiste e impacta as comunidades tradicionais. Consequentemente, o saber etnoecológico e as técnicas de pesca estão se perdendo devido à falta de incentivo e à rápida urbanização (Medeiros, 2001). A documentação desses saberes e fazeres é fundamental para a memória cultural e a preservação ambiental de Florianópolis. Há uma urgência em registrar este conhecimento, visto que os pescadores tradicionais são, em sua maioria, idosos, e as novas gerações não mantêm a tradição (Portal Catarinas, 2021).

Diante do contexto de enfraquecimento da pesca artesanal e do progressivo desaparecimento do conhecimento associado, torna-se premente o desenvolvimento de ações que busquem valorizar e preservar essa cultura. Como contraponto a essa tendência, o presente trabalho propõe o desenvolvimento de um algoritmo de reconhecimento de imagens, focado em identificar as espécies de peixes mais comumente pescadas e conhecidas em

Florianópolis. Este modelo computacional é apresentado como uma ferramenta de documentação e difusão do saber local, buscando preservar a história, valorizar as práticas e, sobretudo, reconectar a população (especialmente as novas gerações) a esse patrimônio cultural.

Para a execução desta tarefa, optou-se pela aplicação de técnicas de Deep Learning (DL), com o desenvolvimento e treinamento de uma Convolutional Neural Network (CNN), metodologia que tem demonstrado eficácia no reconhecimento de padrões visuais. Modelos de CNN têm sido utilizados em diversos estudos para a identificação de animais, como o de Amorim (2022), por exemplo, voltado para a classificação de espécies de peixe. Contudo, trabalhos nessa área utilizam, frequentemente, conjuntos de dados genéricos, que agregam espécies de diversas regiões do mundo.

Neste contexto, o diferencial do presente trabalho reside na proposição de um modelo focado exclusivamente nos peixes da região de Florianópolis, espaço de grande importância histórica e cultural para a pesca. Ao restringir o escopo ao ecossistema marinho local, busca-se não apenas alcançar uma maior acurácia técnica na identificação de espécies catarinenses, mas também criar uma ferramenta de valorização cultural e um instrumento didático que auxilie na preservação da memória pesqueira da Ilha de Santa Catarina.

2. Estudos correlatos

A classificação de espécies de peixes por meio de Deep Learning, especificamente Redes Neurais Convolucionais (CNNs), é um tema vastamente explorado em diversos trabalhos, cada um com estratégias e desafios distintos. Analisando algumas abordagens, nota-se que o trabalho de Amorim (2022) focou na otimização de processos industriais, propondo um modelo de CNN testado em um banco de dados de nove espécies do Mar Egeu e alcançando uma acurácia de 99% com um modelo customizado de três camadas convolucionais. Por outro lado, a pesquisa de Srivastava (2023) abordou desafios críticos da aquicultura, como o uso em dispositivos de edge computing e o manejo de dados desbalanceados para classificar a maturidade de trutas arco-íris. Comparando arquiteturas como ResNet-50 e MobileNet, Srivastava concluiu que o MobileNet V1 demonstrou o melhor desempenho geral, equilibrando acurácia e eficiência.

Em um contexto diferente, Rathi et al. (2017) concentraram-se na classificação de espécies em imagens subaquáticas, lidando com ruídos e iluminação inconsistente. O método empregou pré-processamento robusto (incluindo borramento Gaussiano e binarização de Otsu) antes de alimentar a CNN. O modelo, treinado no conjunto Fish4Knowledge (27.142 imagens, 23 espécies), atingiu uma precisão de 96,29% e destacou a aplicabilidade em tempo real. Finalmente, Iqbal et al. (2021) buscaram auxiliar biólogos marinhos na compreensão do ciclo de vida das espécies, propondo um modelo customizado de CNN (uma versão reduzida da AlexNet). Treinado com 1.334 imagens de seis espécies, essa arquitetura atingiu uma acurácia de teste de 90,48%.

3. Materiais e métodos

A pesquisa adotou uma abordagem mista, iniciando-se com uma etapa qualitativa de revisão bibliográfica para fundamentar as técnicas de Deep Learning (DL), como as Redes Neurais Convolucionais (CNNs) e o Transfer Learning, e avançando para uma etapa quantitativa de avaliação de desempenho dos modelos desenvolvidos.

3.1. Base de dados

O estudo estruturou um conjunto de dados (dataset) próprio contendo 963 imagens de oito espécies (anchova, baiacu, cação, camarão, cavala, garoupa, olho-de-boi e tainha) de relevância cultural e econômica para a pesca artesanal em Florianópolis. As imagens foram compiladas a partir de fontes públicas (Kaggle) e de coleta primária realizada pelo autor no Mercado Público de Florianópolis (espécie anchova). Todo o material passou por um rigoroso processo de curadoria e pré-processamento manual para isolar espécimes individuais e garantir a adequação ao escopo da pesquisa.

3.2. Implementação e arquiteturas

O desenvolvimento foi conduzido integralmente no ambiente Google Colab. Foram selecionadas duas arquiteturas de CNNs para comparação: MobileNetV2, otimizada para eficiência e aplicações móveis, e ResNet50V2, focada em alta acurácia. A implementação utilizou a técnica de Transfer Learning. Os modelos base foram carregados com os pesos pré-treinados do ImageNet e suas camadas convolucionais (a "base") foram congeladas. Um novo classificador, composto por camadas GlobalAveragePooling2D, Dropout e a camada final Dense (Softmax para 8 classes), foi adicionado e foi a única parte treinada com o dataset local.

3.3. Treinamento e avaliação

As imagens foram redimensionadas para o padrão 224x224 pixels e normalizadas antes do treinamento. A técnica de Data Augmentation (incluindo rotação, deslocamento e espelhamento horizontal) foi aplicada exclusivamente ao conjunto de treinamento para aumentar a variabilidade e prevenir o overfitting. O processo de hyperparameter tuning envolveu a realização de quatro testes principais, variando o batch size (16 e 32), o número de épocas (10 e 15) e a proporção de divisão dos dados (70%/15%/15% e 80%/10%/10% para Treino/Validação/Teste). Para a avaliação de desempenho, foram utilizadas métricas robustas como Acurácia, Precisão, Recall, F1-Score e Matriz de Confusão, além da medição do Tempo de Inferência e do Número de Parâmetros.

4. Resultados e discussões

Os modelos de Deep Learning foram avaliados em quatro testes principais com o objetivo de otimizar o desempenho por meio do ajuste de hiperparâmetros, incluindo batch size, épocas e a divisão dos dados. Os resultados demonstraram a alta viabilidade da abordagem de Transfer Learning para a classificação das espécies de Florianópolis, com todos os testes apresentando acurácias superiores a 95%.

O Teste 1 utilizou a configuração de linha de base (Lote 32, 15 Épocas, Divisão 70%/15%/15%) e estabeleceu o melhor equilíbrio geral. Ambos os modelos alcançaram um desempenho quase perfeito, com acurácias idênticas de 0,9929 (99,29%). O MobileNetV2 destacou-se imediatamente por ser significativamente mais leve (2,9 milhões de parâmetros) do que o ResNet50V2 (24,6 milhões de parâmetros) e por apresentar um tempo de inferência ligeiramente menor.

Em seguida, o Teste 2 buscou avaliar o impacto de uma redução nas épocas (10 épocas, mantendo Lote 32 e divisão 70%/15%/15%). A redução resultou em uma queda de desempenho para o MobileNetV2 (Acurácia de 0,9574 e F1-Score de 0,9374), indicando que a configuração de 15 épocas era mais adequada para otimizar o treinamento.

O Teste 3 manteve o Lote 32 e 15 Épocas, mas alterou a divisão dos dados para 80% Treino e 10% Validação/Teste. Paradoxalmente, o aumento da porção de treino e a redução

dos conjuntos de validação/teste resultaram em uma ligeira queda na acurácia para ambos os modelos em comparação com o Teste 1 (ResNet50V2: 0,9681; MobileNetV2: 0,9787), confirmando que a divisão 70%/15%/15% era superior.

Por fim, o Teste 4 investigou o impacto da redução do tamanho do lote para 16, mantendo as 15 épocas e a divisão 70%/15%/15%. Esta configuração levou o ResNet50V2 ao seu desempenho máximo, atingindo 1,0000 de acurácia e F1-Score. No entanto, este resultado sem erros, embora impressionante, foi alcançado com um custo computacional elevado, apresentando um tempo de treinamento consideravelmente maior do que o Teste 1 e indicando um potencial risco de sobreajuste (overfitting) ao conjunto de dados específico.

A análise demonstrou que o Teste 1 apresentou o melhor equilíbrio entre desempenho e eficiência. Embora o ResNet50V2 tenha atingido 100% no Teste 4, o MobileNetV2 (Teste 1) consolidou-se como a arquitetura mais eficiente devido à sua alta acurácia (99,29%), aliada ao seu número de parâmetros drasticamente inferior e tempo de inferência rápido. Esta eficiência o torna o modelo ideal para o objetivo de uma futura implementação móvel.

A comparação dos resultados com estudos correlatos que utilizaram datasets e problemas de classificação distintos (como Srivastava, 2023, com acurácias entre 80,5% e 91,1%, e Iqbal et al., 2021, com 90,48%) validou a eficácia da abordagem de Transfer Learning adotada para o dataset local de Florianópolis.

5. Conclusão

O trabalho atingiu o objetivo geral de desenvolver um modelo de Deep Learning (DL) para a classificação de espécies de peixes de Florianópolis, cumprindo a meta de criar uma ferramenta tecnológica para a documentação e preservação do patrimônio cultural da pesca artesanal. Os resultados demonstraram a alta viabilidade da abordagem, com todos os testes apresentando acurácias superiores a 95% e atingindo picos de 99,29% (Teste 1) e 100% (Teste 4). A análise comparativa indicou que, embora ambas as arquiteturas (MobileNetV2 e ResNet50V2) tivessem desempenho similar na classificação, o MobileNetV2 (Teste 1) consolidou-se como a arquitetura mais eficiente, sendo significativamente mais leve (2,9M de parâmetros) e ideal para futuras aplicações em dispositivos móveis.

Contudo, o estudo apresentou limitações, notadamente o tamanho reduzido do dataset (963 imagens) e o desbalanceamento entre as classes. Além disso, o viés das imagens, muitas obtidas em contextos controlados (peixarias), pode impactar a capacidade de generalização do modelo. Como trabalhos futuros, sugere-se a expansão e o balanceamento do dataset e, principalmente, o desenvolvimento de uma aplicação móvel que integre o modelo, visando a difusão cultural e a educação ambiental, conforme proposto na justificativa do trabalho.

Referências

AMORIM, Leandro Barbosa. Identificação de espécies de peixes e camarão utilizando técnicas de visão computacional e redes neurais convolucionais. 2022. 33 f. Monografia (especialização) – Curso Pós-Graduação Lato Sensu em Engenharia Elétrica, Instituto Federal do Espírito Santo, Vitória, 2022.

DESTÉFANI, Homero Luiz. Pesca e maricultura em Florianópolis, Santa Catarina, Brasil: análise exploratória dos modos de vida e da percepção dos usuários sobre as atividades. 2017. 89 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Oceanografia, Centro de Filosofia e Ciências Humanas, Universidade Federal de Santa Catarina, Florianópolis, 2017.

IQBAL, Muhammad Ather; WANG, Zhijie; ALI, Zain Anwar; RIAZ, Shazia. Automatic Fish Species Classification Using Deep Convolutional Neural Networks. Wireless Personal Communications, v. 116, p. 1043-1053, 2021. Disponível em: https://doi.org/10.1007/s11277-019-06634-1. Acesso em: 9 nov. 2025.

MEDEIROS, Rodrigo Pereira. Estratégias de pesca e usos dos recursos em uma comunidade de pescadores artesanais da Praia do Pântano do Sul (Florianópolis, Santa Catarina). 2001. 108 f. Dissertação (Mestrado) – Instituto de Biologia, Universidade Estadual de Campinas, Campinas, 2001.

PINHO, Ricardo. A pesca artesanal na Baía Sul da Ilha de Santa Catarina: um patrimônio da cultura local. Revista Confluências Culturais, Joinville, v. 5, n. 2, p. 9–28, 2016. Disponível em: https://periodicos.univille.br/RCC/article/view/370. Acesso em: 8 jun. 2025.

PORTAL CATARINAS. Ilha invisível: os desafios dos pescadores tradicionais da Costeira, em Florianópolis. Portal Catarinas, 2021. Disponível em: https://catarinas.info/ilha-invisivel-os-desafios-dos-pescadores-tradicionais-da-costeira-em-florianopolis/#:~:text=O%20n%C3%BAmero%20de%20ranchos%20de,que%20a%20mar%C3%A9%20est%C3%A1%20baixa. Acesso em: 3 dez. 2024.

RATHI, Dhruv; JAIN, Sushant; INDU, Sreedevi. Underwater Fish Species Classification using Convolutional Neural Network and Deep Learning. In: INTERNATIONAL CONFERENCE ON ADVANCES IN PATTERN RECOGNITION (ICAPR), 9., 2017, Bangalore. Anais [...]. Bangalore: Institute of Electrical and Electronics Engineers, 2017. p. 1-6. DOI: 10.1109/icapr.2017.8593044. Acesso em: 11 dez. 2024.

SRIVASTAVA, Varun. Classification of Fish Species Using Deep Learning Models. 2023. 158 f. Dissertação (Mestrado em Ciência da Computação Aplicada) – Department of Computer Science, Faculty of Information Technology and Electrical Engineering, Norwegian University of Science and Technology, Noruega, 2023.