Uma Abordagem Prática para o Reconhecimento de Aves através da Classificação de Imagens

Patrícia da Silva Rodrigues

MONOGRAFIA FINAL

MAC 499 — Trabalho de Formatura Supervisionado

Orientador: Prof. Dr. Roberto Marcondes Cesar Junior

São Paulo

29 de Maio de 2024

Agradecimentos

"Uma criança, um professor, um livro e uma caneta podem mudar o mundo."
- Malala Yousafzai

Gostaria de expressar minha profunda gratidão ao professor Roberto, cuja orientação e apoio foram muito importantes para a realização deste trabalho. Agradeço também a todos os professores do curso, que, com seu conhecimento e empenho, contribuíram para minha formação e me ajudaram a alcançar este importante marco.

Sou imensamente grata ao meu irmão Douglas e à minha tia Fátima pelo apoio emocional e financeiro. Sem a ajuda de vocês, eu não teria conseguido chegar até aqui. Agradeço também de coração às minhas amigas Lume, Samantha e Sabrina, e ao Carlos, meu companheiro, por todo o carinho, pelas risadas, pelos bandeijões, pela correria, pelos estudos, pelo apoio e pela companhia. Foi um prazer compartilhar essa jornada com vocês e contar com o carinho de cada um.

Agradeço, ainda, à minha psicóloga Lidiane, que me ajudou a enfrentar os desafios emocionais durante a graduação.

A todos vocês, meu muito obrigada. A presença e o incentivo de cada um foram fundamentais para que eu pudesse seguir em frente e conquistar este objetivo.

Patrícia da Silva Rodrigues. *Uma abordagem Prática para o Reconhecimento de Aves através da Classificação de Imagens.* Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2024.

Resumo

A classificação automática de imagens desempenha um papel relevante em diversas áreas, permitindo a identificação rápida e precisa de padrões visuais. No contexto da biodiversidade, a criação de modelos capazes de classificar espécies de aves a partir de imagens é muito importante para facilitar o monitoramento e a preservação das espécies. Este trabalho propõe o desenvolvimento de um sistema de classificação de aves com base em imagens, utilizando redes neurais convolucionais (CNN) implementadas em um modelo Keras sequencial. O objetivo principal é melhorar o desempenho do modelo aplicando técnicas de regularização, como dropout e data augmentation, para evitar overfitting e aprimorar a generalização do sistema. A validação do modelo foi realizada com conjuntos de dados independentes, a fim de garantir a precisão na identificação de diferentes espécies de aves. Por fim, o sistema desenvolvido busca fornecer uma solução prática para a classificação automatizada de aves.

Palavras-chave: Aprendizado profundo, Classificação de imagens, Rede neural convolucional

Patrícia da Silva Rodrigues. *A Practical Approach to Bird Recognition through Image Classification*. Monograph (Bachelor's Degree). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2024.

Abstract

Automatic image classification plays a relevant role in various fields, enabling the rapid and accurate identification of visual patterns. In the context of biodiversity, developing models capable of classifying bird species from images is very important for facilitating species monitoring and conservation. This work proposes the development of a bird classification system based on images, using convolutional neural networks (CNNs) implemented in a sequential Keras model. The primary goal is to improve the model's performance by applying regularization techniques, such as dropout and data augmentation, to prevent overfitting and enhance the system's generalization. The model was validated with independent datasets, ensuring accuracy and effectiveness in identifying different bird species. The developed system aims to provide a practical and efficient solution for automated bird classification.

Keywords: Deep learning, Image classification, Convolutional neural network

Conteúdo

1 Introdução

A classificação de imagens utilizando redes neurais convolucionais (CNNs) é uma ferramenta muito presente em diversas áreas, como no diagnóstico médico de doenças, incluindo a detecção de tumores e cânceres, que são capazes de auxiliar na identificação precoce de condições como doenças pulmonares, cardíacas e outras anomalias. Essas redes são importantes na análise de imagens médicas, como raios-X, tomografias e ressonâncias magnéticas, permitindo o reconhecimento de padrões e a detecção precoce de doenças. Da mesma forma, no campo da biodiversidade, as CNNs são aplicadas no monitoramento de ecossistemas, ajudando na identificação de espécies e no estudo de ambientes naturais, tanto aquáticos quanto terrestres.

Um exemplo do uso de técnicas de aprendizado profundo é o banco de dados Wisconsin Diagnostic Breast Cancer (WDBC), utilizado para o diagnóstico de câncer de mama. Este conjunto de dados contém 10 características extraídas de tumores de mama, coletadas de 569 pacientes, e utiliza 30 atributos para a análise da doença. Ademais, a crescente adoção de abordagens baseadas em aprendizado de máquina (ML) tem sido importante para melhorar a detecção e a classificação do câncer de mama com base nessas características. Como ilustrado por Essa, Ismaiel e Hinnawi (2024), as redes neurais convolucionais, combinadas à engenharia de características, têm mostrado grande eficácia na melhoria da precisão diagnóstica [5].

Entre as diversas técnicas empregadas, destaca-se o uso de uma Função de Base Radial (RBF) otimizada com o modelo Support Vector Machine (SVM) para diagnosticar o câncer de mama, que alcançou uma precisão de 96,91% e uma sensibilidade de 97,84% [26]. Além disso, outras abordagens, como a Regressão Logística (LR) nos formatos supervisionado (SL) e semissupervisionado (SSL), bem como o K-vizinho mais próximo (KNN), foram utilizadas para aprimorar a classificação de casos malignos e benignos de câncer de mama. Esses modelos demonstraram excelentes resultados, com a LR atingindo 97% de precisão (SL) e 98% (SSL), e o KNN alcançando precisão de até 98% (SL) e 97% (SSL), destacando sua relevância no diagnóstico precoce e na eficiência dos tratamentos [1].

Além dessas abordagens, outras técnicas, como o uso do SVM polinomial combinado com técnicas exploratórias de dados (DET), têm sido aplicadas. Este modelo obteve uma precisão de 99,03% e uma pontuação F1 de 99,3% [20]. A aplicação de redução de alta dimensionalidade de recursos com a Análise Discriminante Linear (LDA) no pré-processamento, juntamente com o SVM, resultou em uma precisão de 98,82% e sensibilidade de 98,41% [19]. Essas contribuições são exemplos do uso de aprendizado de máquina no aprimoramento da precisão diagnóstica.

No entanto, ao avançarmos para a classificação de aves a partir de imagens, surgem novos desafios. Um aspecto importante na construção de modelos é o desenvolvimento de sistemas capazes de lidar com dados complexos e diversificados, como as imagens de aves. Fatores como o dimorfismo sexual, em que machos e fêmeas de algumas espécies apresentam características físicas distintas, ilustram a variabilidade presente nas imagens. Portanto, depreende-se que o modelo tem que ser capaz de lidar com essa diversidade, que envolve desde a coleta de dados representativos até a escolha de uma arquitetura adequada, além da aplicação de técnicas para mitigar problemas como sobreajuste e subajuste. Neste contexto, o foco deste trabalho é o desenvolvimento de um modelo utilizando a biblioteca Keras, com uma arquitetura sequencial, incorporando técnicas de regularização como dropout e data augmentation, com o intuito de melhorar a generalização do modelo e reduzir erros durante o processo de inferência.

1.1 Motivação

A biodiversidade é um dos pilares para a manutenção da vida no planeta, e as aves têm uma importância de grande impacto nos ecossistemas, atuando como indicadores ambientais, controladores de pragas, polinizadores e dispersores de sementes. No entanto, os crescentes impactos negativos gerados por atividades humanas, como o desmatamento, a urbanização desordenada e as mudanças climáticas, têm causado consequências severas nas populações de aves em todo o mundo. Esse cenário é um exemplo que reforça a necessidade de métodos para monitorar espécies, identificar padrões populacionais e desenvolver estratégias de conservação.

Tradicionalmente, o estudo e a classificação de aves são conduzidos por ornitólogos e pesquisadores especializados, que realizam análises em campo e utilizam ferramentas convencionais para identificar as espécies. Embora esses métodos sejam confiáveis, eles frequentemente demandam um esforço considerável, tanto em termos de tempo quanto de recursos. Além disso, o número de profissionais capacitados é limitado, dificultando o acompanhamento em larga escala da diversidade de espécies em regiões com altos índices de biodiversidade, como o Brasil, que é um país de dimensão continental.

O avanço de tecnologias baseadas em inteligência artificial, especialmente redes neurais convolucionais (CNNs), foi muito importante para modificar a forma como tarefas complexas de classificação e reconhecimento são realizadas. Modelos de aprendizado profundo oferecem a capacidade de analisar imagens com grande precisão, algo que permite a identificação automática de padrões visuais. No contexto da ornitologia, esses modelos podem ser usados para a classificação de espécies a partir de fotografias, algo que fornece uma alternativa rápida, acessível e escalável para estudos de biodiversidade.

Apesar do potencial das CNNs, sua aplicação em cenários reais enfrenta desafios importantes. A construção de um bom modelo e confiável requer um conjunto de dados de alta qualidade, representativo das diferentes espécies e condições ambientais. Ademais, problemas como sobreajuste (overfitting), comuns em modelos treinados com dados limitados, podem comprometer a generalização do sistema para novos dados. Isso reflete a importância do uso de técnicas de regularização, como dropout, data augmentation e a incorporação de métodos de validação cruzada, para melhorar o desempenho do modelo em ambientes diversos.

Neste contexto, o presente trabalho busca contribuir para a área de classificação automatizada de aves, apresentando uma solução que alia simplicidade de implementação e técnicas avançadas de aprendizado de máquina. Por fim, espera-se que este estudo inspire novas iniciativas na interseção entre inteligência artificial e conservação ambiental e que promova a aplicação de tecnologia em desafios de grande relevância.

1.2 Objetivos

Este trabalho tem como principal objetivo desenvolver um modelo de classificação de aves utilizando a biblioteca Keras, com a implementação de técnicas de regularização para aumentar a eficiência e a capacidade de generalização do modelo. A classificação de aves, especialmente em um cenário de grande biodiversidade como o brasileiro, é um desafio que requer diferentes abordagens para superar problemas como sobreajuste, dados limitados ou desbalanceados e variabilidade nas condições das imagens. Para enfrentar esses desafios, a primeira etapa consiste na construção e organização de uma base de dados representativa, com imagens que contemplem a diversidade de espécies de aves, bem como diferentes condições ambientais e de iluminação. Essa base de dados será necessária para garantir que o modelo seja treinado de forma abrangente e que ele seja capaz de capturar a variedade presente nos dados, além de poder ser aplicado de modo a obter bons resultados em cenários reais.

Além disso, será projetada uma arquitetura sequencial de redes neurais convolucionais (CNNs), aproveitando a flexibilidade da biblioteca Keras para experimentar diferentes combinações de camadas e parâmetros. A escolha de uma arquitetura sequencial se justifica pela possibilidade de construção do modelo, algo que permite que cada etapa do processamento seja ajustada às características do problema. Durante o desenvolvimento, serão incorporadas técnicas de regularização, como dropout e data augmentation. O dropout será usado para reduzir a coadaptação excessiva das unidades neuronais, aumentando a capacidade do modelo de generalizar para novos dados. Já o data augmentation será aplicado para expandir artificialmente o conjunto de treinamento, criando variações das imagens originais e ajudando o modelo a aprender características mesmo em dados com ligeiras perturbações.

Por fim, o desempenho do modelo será avaliado em um conjunto de dados de teste, utilizando métricas como acurácia, precisão e recall. Essas métricas permitirão medir a performance do modelo em identificar corretamente as espécies de aves, além de fornecer insights sobre possíveis ajustes necessários. A validação do modelo, além de garantir sua aplicabilidade em cenários reais, também contribui para o desenvolvimento de sistemas mais confiáveis e acessíveis para a classificação automática de espécies.

Após essa etapa, o próximo passo será treinar a base de dados em uma arquitetura moderna e já consolidada, com o objetivo de comparar os desempenhos entre essa arquitetura e a minha proposta.

2 Metodologia

2.1 Base de dados

O dataset selecionado para o treinamento do modelo foi o BIRD SPECIES CLASSIFICATION with DEEP LEARNING, extraído da plataforma kaggle, uma plataforma que fornece datasets gratuitamente para estudo. O dataset pode ser acessado através do link `https://www.kaggle.com/code/nostal1563/525-birds-beginner-s-image-classification/`. Foram coletadas mais de 100 mil imagens de 525 espécies de aves.

Figura 1: Amostra de aves da base de dados. Fonte: Kaggle.

O conjunto de treinamento, teste e validação possuem, respectivamente: 84635 imagens pertencentes a 525 classes. 2625 imagens pertencentes a 525 classes. 2625 imagens pertencentes a 525 classes.

Os datasets foram divididos de modo a destinar 94% para o conjunto de treinamento, 3% para o conjunto de teste e 3% para o conjunto de validação. As imagens utilizadas para teste são completamente independentes do processo de treinamento e validação do modelo, a fim de não gerar overfitting. Todas as imagens foram coletadas de aves em seus habitats naturais.

Para construir o conjunto de dados, foram considerados diversos fatores para evitar viés de amostragem e garantir a representatividade. O conjunto de dados inclui uma ampla variedade de espécies de

aves, visando garantir que o classificador seja capaz de reconhecer diferentes tipos de aves.

O conjunto de dados também conta com uma grande variabilidade de dados, visto que uma grande quantidade de imagens foi considerada, mitigando o **viés de amostragem** e garantindo que o modelo seja treinado com dados representativos. Isso permite que o modelo aprenda padrões gerais que possam ser aplicados a novas imagens de aves.

As imagens utilizadas para o treinamento, teste e validação do modelo possuem as seguintes características: Alguns exemplos de imagens presentes no dataset:

- **Dimensões:** 224 x 224 pixels
- **Formato:** JPEG
- **Modo de cor:** RGB
- **Tamanho do arquivo:** 24,49 KB

Figura 2: Abbott's BabblerFonte: Kaggle.

Figura 3: Black-throated Huet

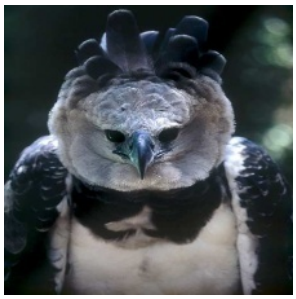

Figura 4: Harpy Eagle

Figura 5: Paradise Tanager

A seguir, a tabela com o nome das espécies de aves que foram usadas para treinar o classificador.

Tabela 1: Tabela de Espécies - Página 1

Abbotts Booby	Abbotts Babbler	Abyssinian Ground Hornbill
African Crowned Crane	African Firefinch	African Emerald Cuckoo
African Pygmy Goose	African Oyster Catcher	African Pied Hornbill
Alexandrine Parakeet	Albatross	Alberts Towhee
Altamira Yellowthroat	American Avocet	Alpine Chough
American Dipper	American Coot	American Bittern
American Pipit	American Flamingo	American Kestrel
American Goldfinch	American Robin	American Wigeon
American Redstart	Andean Lapwing	Anhinga
Andean Siskin	Amethyst Woodstar	Andean Goose
Anianiau	Annas Hummingbird	Antbird
Apapane	Antillean Euphonia	Asian Crested Ibis
Asian Green Bee Eater	Apostlebird	Ashy Storm Petrel
Araripe Manakin	Asian Dollard Bird	Ashy Thrushbird
Auckland Shaq	Austral Canastero	Asian Openbill Stork
Azure Tit	Azure Tanager	Azaras Spinetail
Avadavat	Azure Jay	Azure Breasted Pitta
Australasian Figbird	Bald Eagle	Baikal Teal
Bald Ibis	Bananaquit	Bali Starling
Banded Stilt	Baltimore Oriole	Banded Pita
Band Tailed Guan	Banded Broadbill	Bar-tailed Godwit
Barn Owl	Barn Swallow	Bay-breasted Warbler
Bearded Bellbird	Bearded Barbet	Bearded Reedling
Barred Puffbird	Barrows Goldeneye	Belted Kingfisher
Black And Yellow Broadbill	Black Baza	Bird Of Paradise
Black Necked Stilt	Black Headed	Caique
Black Cockato	Black Faced Spoonbill	Black Francolin
Black Breasted Puffbird	Black Skimmer	Black Throated Bushtit
Black Swan	Black Tail Crake	Black-capped Chickadee
Black Throated Warbler	Black Throated Huet	Black-necked Grebe
Black-throated Sparrow	Black Vented Shearwater	Black Vulture
Blood Pheasant	Blackburniam Warbler	Blonde Crested Woodpecker
Blue Gray Gnatcatcher	Blue Malkoha	Blue Coau
Blue Dacnis	Blue Heron	Blue Grouse
Blue Grosbeak	Blue Throated Toucanet	Bobolink
Blue Throated Piping Guan	Bornean Pheasant	Bornean Bristlehead
Brown Crepper	Brewers Blackbird	Brandt Cormarant
Bornean Leafbird	Brown Headed Cowbird	Brown Noody
Bufflehead	Brown Thrasher	Bush Turkey
California Condor	Cabots Tragopan	Cactus Wren
Bulwers Pheasant	Caatinga Cacholote	Burchells Courser
California Quail	California Gull	Campo Flicker
Capped Heron	Cape Rock Thrush	Cape Longclaw
Canvasback	Cape Glossy Starling	Cape May Warbler
Canary	Carmine Bee-eater	Capuchinbird
Caspian Tern	Chestnut Winged Cuckoo	Cassowary
Chattering Lory	Chestnet Bellied Euphonia	Cedar Waxwing
Cerulean Warbler	Chara De Collar	Chipping Sparrow
Chinese Bamboo Partridge	Chinese Pond Heron	Chukar Partridge
Cinnamon Flycatcher	Clarks Nutcracker	Cinnamon Attila
Chucao Tapaculo	Cinnamon Teal	Clarks Grebe

Tabela 2: Tabela de Espécies - Página 1

Cock Of The Rock	Cockatoo	Collared Aracari
Common Poorwill	Common House Martin	Common Firecrest
Common Loon	Common Grackle	Common Iora
Collared Crescentchest	Common Starling	Coppersmith Barbet
Coppery Tailed Coucal	Crested Auklet	Crested Caracara
Crested Fireback	Crane Hawk	Crab Plover
Crested Coua	Cream Colored Woodpecker	Crested Nuthatch
Crested Kingfisher	Crested Oropendola	Crimson Chat
Crimson Sunbird	Crow	Crested Shriketit
Crested Wood Partridge	Crested Serpent Eagle	Cuban Tody
Cuban Trogon	D-arnauds Barbet	Curl Crested Aracuri
Double Barred Finch	Double Brested Cormarant	Daurian Redstart
Dark Eyed Junco	Dalmatian Pelican	Darjeeling Woodpecker
Demoiselle Crane	Dunlin	Downy Woodpecker
Double Eyed Fig Parrot	Eastern Golden Weaver	Eastern Bluebird
Eastern Bluebonnet	Eared Pita	Dusky Lory
Dusky Robin	Eastern Golden Weaver	Eastern Bluebird
Eastern Bluebonnet	Eared Pita	Dusky Lory
Dusky Robin	Eastern Meadowlark	Eastern Towee
Eastern Rosella	Eastern Wip Poor Will	Elliots Pheasant
Ecuadorian Hillstar	Emerald Tanager	Elegant Trogon
Egyptian Goose	Eastern Yellow Robin	Emperor Penguin
Eurasian Bullfinch	Enggano Myna	Emu
Eurasian Magpie	European Goldfinch	European Turtle Dove
Evening Grosbeak	Fairy Bluebird	Fairy Penguin
Eurasian Golden Oriole	Fairy Tern	Fasciated Wren
Fan Tailed Widow	Frigate	Flame Bowerbird
Fiery Minivet	Fire Tailled Myzornis	Forest Wagtail
Fiordland Penguin	Flame Tanager	Frill Back Pigeon
Gambels Quail	Gang Gang Cockatoo	Glossy Ibis
Gold Wing Warbler	Go Away Bird	Gilded Flicker
Golden Cheeked Warbler	Gila Woodpecker	Golden Bower Bird
Golden Parakeet	Golden Chlorophonia	Golden Eagle
Gouldian Finch	Golden Pheasant	Grandala
Gray Kingbird	Golden Pipit	Gray Partridge
Gray Catbird	Great Argus	Great Gray Owl
Great Jacamar	Great Kiskadee	Great Tinamou
Greater Prairie Chicken	Great Potoo	Great Xenops
Greator Sage Grouse	Greater Pewee	Green Magpie
Green Broadbill	Green Jay	Gyrfalcon
Gurneys Pitta	Guineafowl	Guinea Turaco
Green Winged Dove	Grey Headed Fish Eagle	Grey Plover
Grey Cuckooshrike	Groved Billed Ani	Grey Headed Chachalaca
Hamerkop	Harlequin Duck	Harlequin Quail
Hawaiian Goose	Harpy Eagle	Himalayan Bluetail
Himalayan Monal	Hawfinch	Helmet Vanga
Hepatic Tanager	Hoopoes	Hooded Merganser
Hoatzin	House Finch	Horned Guan
Iberian Magpie	House Sparrow	Horned Sungem
Hyacinth Macaw	Horned Lark	Ibisbill
Imperial Shaq	Inca Tern	Indian Vulture
Indigo Bunting	Indian Pitta	Indian Roller
Indian Bustard	Inland Dotterel	Indigo Flycatcher

Tabela 3: Tabela de Espécies - Página 1

Ivory Billed Aracari	Iwi	Ivory Gull
Jacobin Pigeon	Japanese Robin	Java Sparrow
Jabiru	Jack Snipe	Jandaya Parakeet
Jocotoco Antpitta	Kakapo	Killdear
Kagu	Knob Billed Duck	King Vulture
Kookaburra	King Eider	Lark Bunting
Laughing Gull	Kiwi	Lazuli Bunting
Lesser Adjutant	Lilac Roller	Loggerhead Shrike
Looney Birds	Lucifer Hummingbird	Long-eared Owl
Limpkin	Magpie Goose	Little Auk
Malabar Hornbill	Malachite Kingfisher	Malagasy White Eye
Masked Bobwhite	Maleo	Marabou Stork
Mallard Duck	Mangrove Cuckoo	Masked Booby
Mandrin Duck	Merlin	Mckays Bunting
Masked Lapwing	Myna	Nicobar Pigeon
Northern Beardless Tyrannulet	Mourning Dove	Noisy Friarbird
Mikado Pheasant	Military Macaw	Northern Cardinal
Northern Fulmar	Northern Flicker	Northern Jacana
Northern Gannet	Northern Goshawk	Northern Shoveler
Ocellated Turkey	Northern Parula	Northern Red Bishop
Okinawa Rail	Northern Mockingbird	Oilbird
Oriental Bay Owl	Orange Breasted Trogon	Orange Brested Bunting
Palila	Osprey	Oyster Catcher
Painted Bunting	Ovenbird	Ornate Hawk Eagle
Ostrich	Paradise Tanager	Parakett Auklet
Palm Nut Vulture	Patagonian Sierra Finch	Parus Major
Pink Robin	Phainopepla	Peregrine Falcon
Peacock	Philippine Eagle	Plush Crested Jay
Puffin	Pomarine Jaeger	Puna Teal
Purple Finch	Purple Swamphen	Pyrrhuloxia
Purple Gallinule	Purple Martin	Pygmy Kingfisher
Razorbill	Rainbow Lorikeet	Quetzal
Red Faced Cormorant	Red Browed Finch	Red Billed Tropicbird
Red Bellied Pitta	Red Crossbill	Red Faced Warbler
Red Bearded Bee Eater	Red Headed Duck	Red Fody
Red Headed Woodpecker	Red Tailed Thrush	Red Winged Blackbird
Red Shouldered Hawk	Red Naped Trogon	Red Knot
Red Tailed Hawk	Red Legged Honeycreeper	Red Wiskered Bulbul
Ring-necked Pheasant	Regent Bowerbird	Rose Breasted Grosbeak
Rosy Faced Lovebird	Roseate Spoonbill	Rough Leg Buzzard
Rose Breasted Cockatoo	Rock Dove	Roadrunner
Royal Flycatcher	Ruby Crowned Kinglet	Ruby Throated Hummingbird
Sand Martin	Samatran Thrush	Rufous Trepe
Rufuos Motmot	Rudy Kingfisher	Ruddy Shelduck
Rufous Kingfisher	Sandhill Crane	Satyr Tragopan
Says Phoebe	Scarlet Ibis	Shoebill
Scarlet Faced Liocichla	Scarlet Tanager	Scarlet Crowned Fruit Dove
Scarlet Macaw	Short Billed Dowitcher	Snow Goose
Smiths Longspur	Snow Partridge	Sora
Snowy Owl	Spangled Cotinga	Splendid Wren
Snowy Plover	Snowy Egret	Snowy Sheathbill
Spoon Biled Sandpiper	Spotted Whistling Duck	Spotted Catbird
Striated Caracara	Stork Billed Kingfisher	Sri Lanka Blue Magpie
Squacco Heron	Striped Owl	Steamer Duck
Stripped Manakin	Stripped Swallow	Sunbittern

Tabela 4: Tabela de Espécies - Página 1

Superb Starling	Takahe	Tailorbird
Surf Scoter	Tawny Frogmouth	Swinhoes Pheasant
Tasmanian Hen	Taiwan Magpie	Tit Mouse
Touchan	Teal Duck	Tropical Kingbird
Tricolored Blackbird	Townsends Warbler	Turkey Vulture
Tree Swallow	Turquoise Motmot	Trumpter Swan
Varied Thrush	Umbrella Bird	Veery
Violet Cuckoo	Violet Green Swallow	Victoria Crowned Pigeon
Verdin	Violet Backed Starling	Vermilion Flycather
Venezuelian Troupial	Violet Turaco	Visayan Hornbill
Vulturine Guineafowl	Wattled Curassow	White Browed Crake
White Breasted Waterhen	White Cheeked Turaco	Whimbrel
Wall Creaper	Wattled Lapwing	White Crested Hornbill
White Eared Hummingbird	White Necked Raven	Willow Ptarmigan
White Throated Bee Eater	White Tailed Tropic	Wild Turkey
Wood Thrush	Wilsons Bird Of Paradise	Wood Duck
Yellow Bellied Flowerpecker	Wrentit	Woodland Kingfisher
Zebra Dove	Yellow Breasted Chat	Yellow Headed Blackbird
Yellow Cacique		

2.2 Arquitetura do Modelo

O modelo Keras definido é uma rede neural convolucional (CNN) projetada para tarefas de classificação de imagens. A arquitetura do modelo é descrita e explicada abaixo.

```
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(
        1./255,
        input_shape=(img_height, img_width, 3)
    ),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Listing 1: Definição e Compilação de um Modelo Keras

2.2.1 Camadas

Camada de Redimensionamento (Rescaling)

A camada de redimensionamento realiza a normalização das imagens de entrada. Ela ajusta os valores dos pixels das imagens de um intervalo de 0-255 para 0-1. Isso é feito dividindo cada valor do pixel por 255. Além disso, esta camada define a forma de entrada das imagens, que é $(\text{img\_height}, \text{img\_width}, 3)$, onde img_height e img_width são as dimensões da imagem e 3 representa os três canais de cor (RGB).

Camadas Convolucionais e MaxPooling

O bloco de construção mais importante de uma CNN é a camada convolucional.

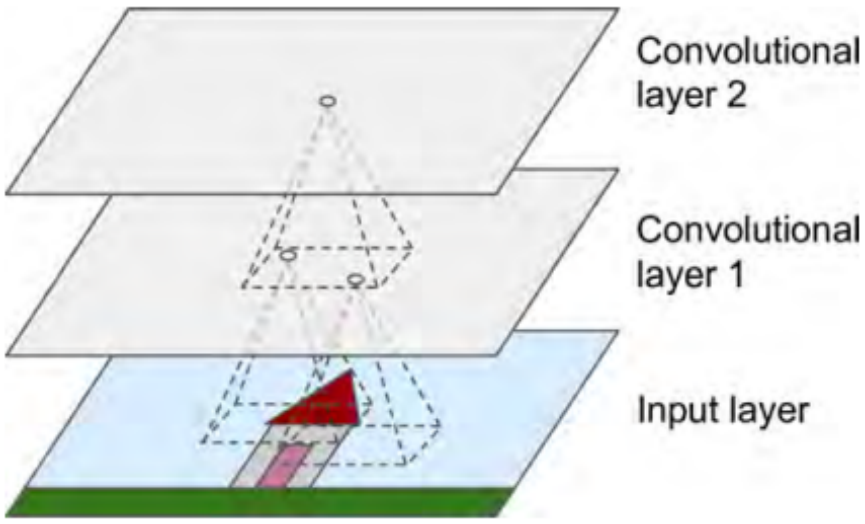

Figura 6: Download from finelybook www.finelybook.com

Camada de Achatar (Flatten)

A camada de achatar converte a saída tridimensional das camadas convolucionais e de pooling em um vetor unidimensional. Esse vetor é necessário para conectar as camadas convolucionais com as camadas densas subsequentes.

Camadas Densas (Fully Connected)

- **Primeira Camada Densa:** Possui 128 neurônios e utiliza a função de ativação `relu`. Esta camada aprende combinações não lineares das características extraídas pelas camadas convolucionais.
- **Camada Final Densa:** Possui um número de neurônios igual ao número de classes (`num_classes`). Esta camada não utiliza uma função de ativação, pois a perda de entropia cruzada esparsa, utilizada para a classificação, lida diretamente com logits.

A compilação do modelo é realizada com o otimizador `Adam`, que é conhecido por combinar as melhores propriedades dos otimizadores Adagrad e RMSProp. Ele adapta as taxas de aprendizado para cada parâmetro, melhorando a performance do modelo.

A função de perda utilizada é a entropia cruzada esparsa, adequada para tarefas de classificação múltipla com rótulos inteiros. O parâmetro `from_logits=True` é especificado para indicar que a saída da camada final são logits, e não probabilidades.

A métrica usada para avaliar o desempenho do modelo durante o treinamento e a validação é a acurácia, que mede a proporção de previsões corretas.

Definição da compilação do modelo:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(
              from_logits=True
              ),
              metrics=['accuracy'])
```

Listing 2: Definição e Compilação de um Modelo Keras

2.2.2 Sumário da arquitetura

O modelo combina camadas convolucionais para extração de características, camadas de pooling para redução dimensional e camadas densas para a classificação final, compondo uma arquitetura para a classificação de imagens.

Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_1 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_2 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 64)	0
flatten (Flatten)	(None, 30976)	0
dense (Dense)	(None, 128)	3,965,056
dense_1 (Dense)	(None, 525)	67,725

Total params: 4,056,365 (15.47 MB)
Trainable params: 4,056,365 (15.47 MB)
Non-trainable params: 0 (0.00 B)

Figura 7: sumário da arquitetura

A arquitetura da rede neural convolucional consiste nos seguintes blocos:

1. Rescaling (224, 224, 3): Normaliza os valores dos pixels para a faixa [0, 1].
2. Conv2D (16 filtros, 3x3): Extrai características com 448 parâmetros, saída $(224, 224, 16)$.

3. MaxPooling2D: Reduz a dimensionalidade para $(112, 112, 16)$.
4. Conv2D (32 filtros, 3x3): Aumenta a captura de características, saída $(112, 112, 32)$ com 4.640 parâmetros.
5. MaxPooling2D: Reduz para $(56, 56, 32)$.
6. Conv2D (64 filtros, 3x3): Captura características mais detalhadas, saída $(56, 56, 64)$ com 18.496 parâmetros.
7. MaxPooling2D: Reduz para $(28, 28, 64)$.
8. Flatten: Converte para um vetor unidimensional de tamanho 50.176.
9. Dense (128 neurônios): Aprende combinações complexas com 6.422.656 parâmetros.
10. Dense (525 neurônios): Camada de saída com 67.725 parâmetros.

Resumo de Parâmetros
Total de parâmetros: 6.513.965, todos treináveis.
Referência: TensorFlow - Classificação de imagem [25]

2.3 Métricas de Avaliação

Para avaliar o desempenho do modelo de classificação de imagens, várias métricas são utilizadas, cada uma fornecendo uma visão específica sobre a performance do modelo. As principais métricas utilizadas incluem:

- **Acurácia (Accuracy)**: A acurácia mede a proporção de previsões corretas em relação ao total de previsões feitas, ou seja, a porcentagem de classificações corretas no conjunto de dados de teste. Essa métrica é útil para obter uma visão geral da performance do modelo.
- **Precisão (Precision)**: A precisão indica a proporção de previsões positivas corretas (verdadeiros positivos) em relação ao total de previsões positivas feitas pelo modelo, ou seja, quantas das previsões de uma classe realmente pertencem a essa classe.
- **Revocação (Recall)**: A revocação, ou sensibilidade, mede a proporção de positivos verdadeiros corretamente identificados pelo modelo em relação ao total de instâncias reais da classe, ou seja, quantas instâncias de uma classe foram corretamente identificadas entre todas as que realmente pertencem a essa classe.
- **F1-Score**: O F1-Score é a média harmônica entre precisão e revocação. Ele fornece uma única métrica que balanceia tanto a precisão quanto a revocação, sendo útil quando há um desequilíbrio entre as classes ou quando é importante equilibrar essas duas métricas.
- **Top-5 Accuracy**: Esta métrica avalia se a verdadeira classe está entre as 5 classes mais prováveis previstas pelo modelo. Isso é particularmente útil em tarefas de classificação com um grande número de categorias, onde a classe correta pode não estar na primeira posição, mas pode estar entre as cinco primeiras.
- **Curva ROC e AUC (Área Sob a Curva)**: A curva ROC é uma representação gráfica que ilustra o desempenho do modelo ao classificar cada classe, mostrando a relação entre as taxas de verdadeiros positivos e falsos positivos. A AUC (Área sob a Curva) quantifica essa relação, indicando a capacidade do modelo em distinguir entre classes. Quanto maior a AUC, melhor o modelo.

A tabela a seguir descreve como é feito o cálculo das métricas.

	Previsão Positiva	**Previsão Negativa**
Classe Positiva Real	Verdadeiro Positivo (VP)	Falso Negativo (FN)
Classe Negativa Real	Falso Positivo (FP)	Verdadeiro Negativo (VN)

1. **Acurácia:** A acurácia mede a proporção de previsões corretas.

$$\text{Acurácia} = \frac{\text{Número de previsões corretas}}{\text{Número total de previsões}}, \quad 0 \leq \text{Acurácia} \leq 1$$

2. **Precisão:** A precisão indica a proporção de positivos previstos que são realmente positivos.

$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}}, \quad 0 \leq \text{Precisão} \leq 1$$

3. **Recall (Revocação):** A revocação mede a proporção de positivos reais que foram corretamente identificados.

$$\text{Recall} = \frac{\text{VP}}{\text{VP} + \text{FN}}, \quad 0 \leq \text{Recall} \leq 1$$

4. **F1-Score:** O F1-Score é a média harmônica entre precisão e revocação.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precisão} \cdot \text{Recall}}{\text{Precisão} + \text{Recall}}, \quad 0 \leq \text{F1-Score} \leq 1$$

Por fim, será feita uma comparação entre a arquitetura construída e a Xception será feita por meio das métricas de desempenho.

3 Conceitos sobre Classificação de Imagens com Redes Neurais Convolucionais

A classificação de imagens é muito relevante em diversas aplicações da visão computacional, como diagnóstico médico, reconhecimento facial e monitoramento de objetos, como mencionado anteriormente. As Redes Neurais Convolucionais (CNNs) se destacam como uma das abordagens mais usadas atualmente para essa tarefa, devido à sua capacidade de capturar características hierárquicas e complexas diretamente das imagens. A estrutura das CNNs, que combina convoluções, pooling e camadas densas, permite que elas aprendam boas representações dos dados. Este capítulo explora os conceitos para a construção e treinamento de modelos baseados em CNNs, começando pela importância de bases de dados de qualidade.

3.1 Conceitos para a Construção da Base de Dados

A criação de um bom modelo está intrinsecamente ligada à construção de uma base de dados representativa. É muito importante considerar os princípios fundamentais que orientam essa etapa, pois a qualidade da base de dados influencia diretamente o desempenho e a generalização do modelo.

3.1.1 Qualidade Insuficiente dos Dados de Treinamento

Uma quantidade insuficiente de dados é um fator que pode impactar negativamente no treinamento de um modelo. Em um famoso artigo publicado em 2001, os pesquisadores da Microsoft Michele Banko e Eric Brill mostraram que diferentes algoritmos de aprendizado de máquina apresentam desempenhos semelhantes em um problema de desambiguação de linguagem natural quando lhes é fornecida uma quantidade suficiente de dados [2].

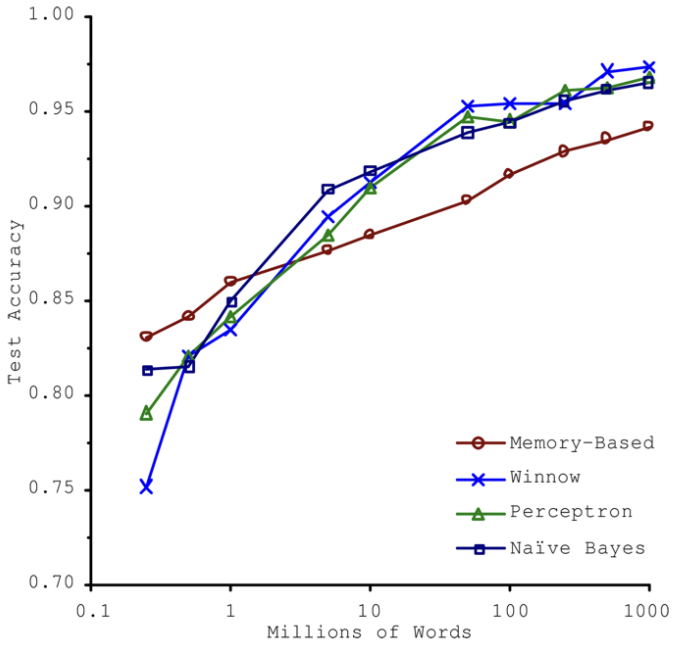

Figura 8: Figure from Banko and Brill (2001), "Learning Curves for Confusion Set Disambiguation." [2].

O autor pontuou: "these results suggest that we may want to reconsider the trade-off between spending time and money on algorithm development versus spending it on corpus development." A ideia de que dados eram mais importantes que algoritmos para problemas complexos foi popularizada por Peter Norvig et al. em um artigo intitulado "The Unreasonable Effectiveness of Data" publicado em 2009 [15].

3.1.2 Dados de Treinamento não Representativos

Em um modelo de aprendizagem de máquina, é essencial que a base de dados de treinamento seja representativa para os novos casos que se deseja generalizar. Se a amostra for muito pequena, podemos

gerar o *sampling noise* (ou seja, dados não representativos resultantes do acaso) [7]. Mas mesmo amostras muito grandes podem ser não representativas se o método de amostragem for falho. Isso é conhecido como *sampling bias* (viés de amostragem) [7].

Um exemplo famoso de sampling bias ocorreu durante as eleições presidenciais de 1936, nas quais disputavam Landon contra Roosevelt. Foi conduzida uma pesquisa que enviou correspondências para cerca de 10 milhões de pessoas. O pesquisador obteve 2,4 milhões de respostas e previu com alta confiança que Landon receberia 57% dos votos.

No entanto, Roosevelt venceu com 62% dos votos. Mais tarde, descobriu-se que a falha estava no método de amostragem utilizado. Para obter as respostas, o Literary Digest utilizou listas telefônicas, listas de assinantes de revistas e listas de membros de clubes. O público que costuma consumir esse tipo de conteúdo geralmente são pessoas mais ricas, e esse público tradicionalmente vota no partido republicano, ao qual Landon pertencia.

Ademais, menos de 25% das pessoas que receberam a pesquisa responderam, e essas pessoas não foram levadas em conta no método de amostragem, pois o estudo não deu relevância a pessoas que não se importam tanto com a política ou pessoas que não gostam do jornal Literary Digest, conhecido como *nonresponse bias* (viés de não resposta) [7].

3.1.3 Engenharia de Recursos

Uma das partes críticas para o sucesso do modelo de aprendizado de máquina é encontrar um bom conjunto de características para treinar. A extração de características é importante nesse caso. Pode-se extrair características relevantes (ou seja, selecionar as características mais úteis para treinar dentre as características existentes), além de combinar características existentes para produzir uma mais útil, e isso é feito por meio de algoritmos de redução de dimensionalidade [2].

3.1.4 Overfitting: Sobreajuste dos Dados de Treinamento

O sobreajuste ocorre quando um modelo de aprendizado de máquina apresenta um desempenho excepcional nos dados de treinamento, mas não consegue generalizar para novos dados [8].

Esse fenômeno pode ser comparado a um turista que, após ser enganado por um taxista em um país estrangeiro, generaliza essa experiência negativa, assumindo que todos os taxistas daquele país são desonestos. Essa tendência à generalização excessiva pode levar a um modelo que aprende padrões espúrios, capturando ruídos em vez de relações verdadeiras nos dados.

Modelos complexos, como redes neurais profundas, têm a capacidade de detectar padrões sutis nos dados. No entanto, se o conjunto de treinamento for pequeno ou contiver muito ruído, o modelo pode acabar aprendendo padrões que não se aplicam a novos dados. Por exemplo, um modelo de satisfação da vida que incorpora atributos irrelevantes pode identificar padrões que ocorrem apenas por acaso, como a correlação entre a letra "w"no nome de um país e uma alta satisfação. Essa abordagem não se sustenta quando aplicada a novos casos, pois o modelo estaria apenas capturando coincidências que não têm valor preditivo real.

Para mitigar o sobreajuste, algumas estratégias podem ser adotadas:

- Simplificar o modelo: Escolher um modelo com menos parâmetros ou reduzir a quantidade de atributos nos dados de treinamento.
- Aumentar o conjunto de dados: Coletar mais dados de treinamento pode ajudar a proporcionar uma base mais representativa para o aprendizado.
- Reduzir o ruído: Corrigir erros nos dados e remover outliers pode ajudar a limpar o conjunto de dados.

A regularização é uma técnica que visa simplificar o modelo, controlando a complexidade do mesmo. Um hiperparâmetro de regularização pode ser ajustado para equilibrar o ajuste aos dados de treinamento e a capacidade de generalização do modelo.

3.1.5 Underfitting dos Dados de Treinamento

O subajuste é o oposto do sobreajuste e ocorre quando o modelo é muito simples para capturar a complexidade dos dados subjacentes [9]. Por exemplo, um modelo linear pode não ser capaz de representar

adequadamente a realidade complexa da satisfação da vida, resultando em previsões imprecisas, mesmo para os dados de treinamento.

As opções principais para corrigir o subajuste incluem:

- Selecionar um modelo mais poderoso: Optar por modelos com mais parâmetros que possam captar melhor a estrutura dos dados.
- Melhorar as características: Realizar engenharia de características para fornecer atributos mais informativos ao algoritmo de aprendizado.
- Reduzir as restrições do modelo: Diminuir o hiperparâmetro de regularização pode permitir que o modelo se adapte melhor aos dados.

3.1.6 Teste e Validação

Após o treinamento de um modelo, é importante que se possa confiar em sua capacidade de generalização e também avaliá-lo. Portanto, a avaliação do desempenho do modelo em dados que não foram vistos durante o treinamento é necessária para garantir sua aplicabilidade em cenários do mundo real [10].

Os métodos de teste e validação incluem:

- **Divisão dos dados:** Separar os dados em conjuntos de treinamento, validação e teste. Isso permite treinar o modelo em um subconjunto, ajustar hiperparâmetros com base em outro subconjunto e avaliar a performance final no conjunto de teste.
- **Validação cruzada:** Técnica que envolve a divisão do conjunto de dados em múltiplos subconjuntos para garantir que o modelo seja testado em diferentes partes dos dados, aumentando a confiabilidade da avaliação.
- **Métricas de desempenho:** Utilizar métricas adequadas, como precisão, recall e F1-score, para quantificar o desempenho do modelo em diferentes cenários.

A validação e o teste são etapas críticas no ciclo de vida do aprendizado de máquina, pois garantem que o modelo seja capaz de aprender a partir dos dados de treinamento e se comportar bem com novos dados.

3.1.7 Descompasso de Dados

No desenvolvimento de modelos de aprendizado de máquina, um desafio comum é a discrepância entre os dados utilizados para treinamento e aqueles que o modelo encontrará em produção [11].

Embora seja fácil coletar grandes quantidades de dados, esses dados podem não ser representativos do cenário real.

Um exemplo prático é o desenvolvimento de um aplicativo móvel para identificar espécies de flores a partir de fotos. É possível baixar milhões de imagens de flores da internet, mas essas imagens podem não refletir com precisão as fotos tiradas pelos usuários do aplicativo em diferentes condições e contextos. Por exemplo, pode-se ter apenas 10.000 fotos representativas, ou seja, realmente tiradas com o aplicativo [11].

Para garantir que o modelo funcione adequadamente em produção, é importante que os conjuntos de validação e teste sejam compostos exclusivamente por imagens representativas. É recomendável embaralhar essas imagens e dividi-las em dois conjuntos: metade para validação e metade para teste, evitando duplicatas [11].

- **Criar um Conjunto Train-Dev**: Reserve uma parte das imagens de treinamento como um conjunto adicional para avaliação do modelo.
- **Avaliar Desempenho**: Deve-se testar o modelo no conjunto train-dev após o treinamento para verificar se há superajuste.
- **Analisar Resultados**:
  - Bom desempenho no train-dev indica que o modelo não está superajustado.

– Desempenho ruim no conjunto de validação sugere descompasso de dados.

- **Pré-processamento das Imagens**: Deve-se ajustar as imagens da web para que se pareçam mais com as fotos tiradas pelo aplicativo.
- **Aprimorar o Modelo**: Se o desempenho no train-dev for insatisfatório, deve-se simplificar o modelo, obter mais dados de treinamento ou limpar os dados existentes.

3.2 Convolução em Redes Neurais

3.2.1 O que é Convolução e Por que é Usada?

A convolução é uma operação matemática fundamental utilizada em redes neurais convolucionais (CNNs) para o processamento de imagens. No contexto das CNNs, a convolução tem a função de extrair características importantes de uma imagem, aplicando filtros ou kernels. Esses filtros são responsáveis por detectar padrões específicos, como bordas, texturas e outros detalhes visuais presentes na imagem.

A utilização da convolução nas CNNs justifica-se por vários motivos:

- **Detecção de Características:** A convolução é essencial na identificação de características importantes da imagem, como bordas e texturas. Cada kernel pode ser treinado para detectar um tipo específico de característica, permitindo a extração de informações cruciais para o entendimento da imagem.
- **Preservação da Estrutura Espacial:** A convolução preserva a relação entre pixels adjacentes, o que é fundamental para a interpretação de padrões visuais. Isso permite que a estrutura espacial da imagem seja mantida durante o processo de extração de características.
- **Performance Computacional:** A aplicação da convolução reduz o número de parâmetros e cálculos necessários em comparação com abordagens vetoriais diretas. Isso resulta em uma maior eficiência computacional, tornando o treinamento e a inferência mais rápidos e com menor demanda de recursos.

3.2.2 Passo a Passo para a Convolução em Imagens

1. **Escolha do Kernel:** Define-se um kernel (ou filtro), uma matriz de pesos tipicamente pequena, como 3×3 ou 5×5.
2. **Aplicação do Kernel:** O kernel é movido sobre a imagem. Para cada posição, o valor do pixel é multiplicado pelo valor correspondente no kernel.
3. **Cálculo da Soma dos Produtos:** Para cada posição do kernel, calcula-se a soma dos produtos das correspondências entre o kernel e a região da imagem. Essa soma resulta em um valor na nova matriz de saída.
4. **Gerar o Mapa de Características:** Repetindo o processo, obtém-se um mapa de características ou feature map, que contém informações sobre a presença da característica detectada pelo kernel.
5. **Aplicar Função de Ativação:** Após a convolução, uma função de ativação (como ReLU) é aplicada para introduzir não-linearidades, permitindo que a rede aprenda padrões complexos.
6. **Subamostragem (Pooling):** Técnicas de pooling reduzem a resolução da imagem e a quantidade de dados, preservando as características importantes.

3.2.3 Definição Matemática para a Convolução

De acordo com o livro *Digital Image Processing* de Rafael C. Gonzalez e Richard E. Woods, a convolução [6] pode ser representada como:

$$g(x) = \sum_{s=-a}^{a} w(s)\, f(x+s)$$

Onde:

- $g(x)$ é o valor resultante da convolução na posição x da imagem.
- $f(x+s)$ representa o valor do pixel da imagem na posição deslocada pela matriz do kernel.
- $w(s)$ é o valor do kernel na posição correspondente.
- A soma é feita sobre todas as posições do kernel, ou seja, para cada posição s do kernel, o valor $w(s)$ é multiplicado pelo valor correspondente da imagem $f(x+s)$ e os resultados são somados.
- x é a posição atual na imagem onde a convolução está sendo calculada.
- s é o índice que percorre as posições do kernel, variando de $-a$ até a (onde a é metade do tamanho do kernel).
- $f(x+s)$ é o valor do pixel da imagem na posição deslocada.
- $w(s)$ é o valor do kernel na posição s.

3.2.4 Aplicação da Convolução: Exemplo

Dado a seguinte imagem f:

Imagem f:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

e kernel w:

Kernel:

$$\begin{bmatrix} 2 & 4 & 2 \\ 4 & 8 & 4 \\ 4 & 8 & 4 \end{bmatrix}$$

Para aplicar a convolução $(w \star f)$, devemos seguir o passo a passo descrito em *Digital Intensity Transformation and Spatial Filtering* de Gonzalez, Rafael C. e Woods, capítulo 3 [6].

Passo 1: Aplicar a técnica de 0 padding :

Ao aplicar a técnica de zero padding em uma imagem $n \times m$ antes da convolução, preserva-se o tamanho original na imagem de saída.

Matriz f:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Passo 2: Inverter o kernel:

w = kernel_invertido:

$$\begin{bmatrix} 4 & 8 & 4 \\ 4 & 8 & 4 \\ 2 & 4 & 2 \end{bmatrix}$$

Passo 3: Aplicar a fórmula da convolução com o kernel invertido:
Matriz f:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Kernel Invertido:

$$\begin{bmatrix} 4 & 8 & 4 \\ 4 & 8 & 4 \\ 2 & 4 & 2 \end{bmatrix}$$

$$\begin{aligned} &1 \times 4 + 0 \times 8 + 0 \times 4 + \\ &0 \times 4 + 1 \times 8 + 1 \times 4 + \\ &0 \times 2 + 1 \times 4 + 0 \times 2 = 16 \end{aligned}$$

Resultado:

$$\begin{bmatrix} 16 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Segunda Matriz f:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Kernel Invertido:

$$\begin{bmatrix} 4 & 8 & 4 \\ 4 & 8 & 4 \\ 2 & 4 & 2 \end{bmatrix}$$

$$\begin{aligned} &0 \times 4 + 0 \times 8 + 0 \times 4 + \\ &1 \times 4 + 1 \times 8 + 1 \times 4 + \\ &1 \times 2 + 0 \times 4 + 1 \times 2 = 20 \end{aligned}$$

Resultado:

$$\begin{bmatrix} 16 & 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

. . .

Desta forma, a aplicação do kernel é realizada de forma sequencial na imagem, prosseguindo até que o resultado final seja obtido.

$$\begin{bmatrix} 16 & 20 & 20 & 20 & 20 \\ 24 & 28 & 28 & 28 & 28 \\ 20 & 20 & 20 & 20 & 20 \\ 22 & 24 & 24 & 24 & 22 \\ 20 & 24 & 24 & 24 & 20 \end{bmatrix}$$

A dimensão da imagem resultante é 5×5, correspondendo exatamente à dimensão da imagem inicial. Este resultado acontece devido à aplicação de *zero padding* antes da operação de convolução, o que assegura que as dimensões da imagem resultante permaneçam inalteradas.

3.2.5 Efeitos da Convolução em Imagens

A convolução resulta em uma rotação de 180° do kernel antes de aplicá-lo à imagem. Isso significa que a convolução reflete o kernel antes de aplicá-lo à imagem. Além disso, a convolução está presente na filtragem de imagem, na qual o kernel atua como um filtro para suavizar, realçar ou extrair características específicas da imagem. Em muitos casos, a convolução é usada para aplicar filtros lineares, como filtros de média, gaussiano, Sobel, entre outros, que ajudam a reduzir ruídos, detectar bordas ou realizar outras operações de processamento de imagem. Ademais, ao contrário da correlação, a convolução é comutativa, e isso significa que a ordem dos operandos não altera o resultado. Portanto, enquanto a correlação é mais adequada para encontrar padrões na imagem, a convolução é frequentemente usada para aplicar filtros à imagem para diferentes fins de processamento, como suavização, realce de bordas, detecção de características, entre outros.

Nota-se, portanto, a importância de observar os resultados obtidos em imagens reais, uma vez que isso possibilita a visualização da performance da operação de convolução no realce de características específicas. Para exemplificar esse ponto, é possível fazer a aplicação de um filtro gaussiano a uma imagem da ave. A ave usada no exemplo é chamada de saíra-pintor (Tangara fastuosa), da ordem Passeriformes e família Thraupidae.

```
1
2  def convolucao(caminho_imagem):
3      imagem_original = cv2.imread(caminho_imagem, cv2.IMREAD_COLOR)
4
5      imagem_original_rgb =cv2.cvtColor(
6          imagem_original,
7          cv2.COLOR_BGR2RGB
8      )
9      imagem_cinza = cv2.cvtColor(
10         imagem_original,
11         cv2.COLOR_BGR2GRAY
12     )
13
14     #filtro gaussiano
15     kernel = np.array([[1, 2, 1],
16                        [2, 4, 2],
17                        [1, 2, 1]]) / 16
18
19     imagem_convoluida = cv2.filter2D(imagem_cinza, -1, kernel)
20     plt.figure(figsize=(10, 5))
21
22     plt.subplot(1, 3, 1)
23     plt.title("Imagem Original")
24     plt.imshow(imagem_original_rgb)
25     plt.axis('off')
26
27     plt.subplot(1, 3, 2)
28     plt.title("Imagem em Niveis de Cinza")
29     plt.imshow(imagem_cinza, cmap='gray')
30     plt.axis('off')
31
32     plt.subplot(1, 3, 3)
33     plt.title("Imagem Apos Convolucao")
34     plt.imshow(imagem_convoluida, cmap='gray')
35     plt.axis('off')
36
37     plt.tight_layout()
38     plt.show()
```

Listing 3: Convulução em imagem

O resultado obtido foi:

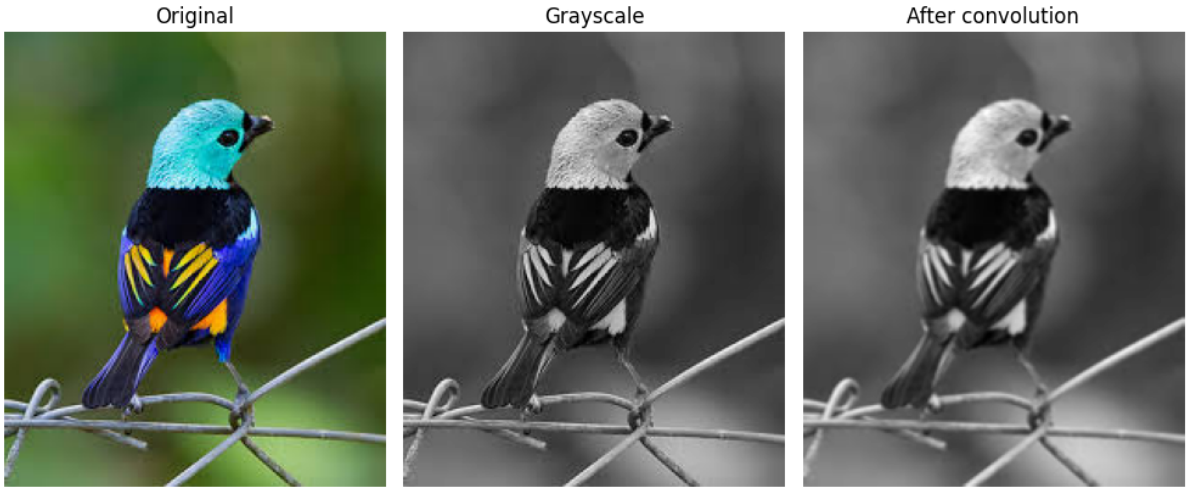

Figura 9: Download from https://www.wikiaves.com.br/wiki/saira-pintor

É possível observar os seguintes efeitos na imagem:

- **Suavização**: A imagem resultante apresenta uma suavização, reduzindo detalhes finos e texturas, o que contribui para uma aparência mais homogênea.
- **Desfoque**: O efeito de desfoque é evidente, manifestando-se na diminuição da nitidez de bordas e contornos, o que leva a transições mais graduais entre as diferentes intensidades de cor.
- **Atenuação de Detalhes**: Elementos pequenos ou finos na imagem são atenuados, tornando-se menos perceptíveis, enquanto as estruturas maiores e mais proeminentes geralmente são preservadas.
- **Alteração nas Intensidades**: Observa-se uma alteração nas intensidades dos pixels, o que pode resultar em modificações no contraste global da imagem.

Outro exemplo interessante é o filtro Sobel vertical e horizontal, que, aplicado a uma imagem, pode gerar 2 mapas de características diferentes [12].

3.2.6 Convolução para filtros Sobel

O filtro Sobel é um operador de processamento de imagem utilizado para detectar bordas em imagens. Ele é baseado na convolução da imagem original com duas máscaras (ou kernels) que calculam as derivadas da intensidade da imagem em direções horizontais e verticais.

No caso do filtro Sobel horizontal, as linhas brancas horizontais são realçadas, enquanto o restante é borrado. De forma análoga, acontece com o filtro Sobel vertical, onde as linhas brancas verticais são realçadas, enquanto o restante é borrado. Para exemplificar, o código abaixo faz a convolução da imagem em níveis de cinza usando um filtro vertical Sobel [24].

```
1
2  # convolution function
3  def convolve(img, filter_, k, x, y):
4      sum = 0
5      # Determine the starting point of the kernel
6      k_start = int(-np.floor(k/2))
7      # Convolution: multiply the image and filter elements and sum them
8      for i in range(k):
9          for j in range(k):
10             sum += img[y+i+k_start][x+j+k_start] * filter_[i][j]
```

```
    # Return the result of convolution for a given point
    return sum

# Convolution function for the entire image
def convolve_image(img, filter_):
    # Determine the size of the kernel
    k = filter_.shape[0].

    # Define the boundaries within which convolution will be performed
    row_start = int(np.floor(k/2))
    row_end = height-k+2
    col_start = int(np.floor(k/2))
    col_end = width-k+2

    # Create a new image filled with zeros, smaller than the original
    by the size of the kernel
    new_image = np.zeros((height-k+1, width-k+1))
    for y in range(row_start, row_end):
        # for x in range(col_start, col_end):
            # Convolve for each point in the area defined by the borders
            new_image[y-row_start][x-col_start] = convolve(img, filter_, k, x, y)
    return new_image

# Load the image
img = cv2.imread('Lenna_(test_image).png', 0)
height, width = img.shape

# Create a figure with 2 graphs
fig, axs = plt.subplots(ncols=2, nrows=1, figsize=(14, 6))

# Define the size and type of filter

"""
For k = 3, the filter will look like this:
[[0, 1, 0],
 [0, 1, 0],
 [0, 1, 0]],
"""
k = 15
filter_ = np.zeros((k, k), dtype=int)
filter_[:, k // 2] = 1
```

Listing 4: Definição e Compilação de um Modelo Keras

No exemplo acima, foi usado um filtro de 15 para refletir melhor a diferença entre a imagem original e a que foi convolvida. Obtemos o resultado abaixo:

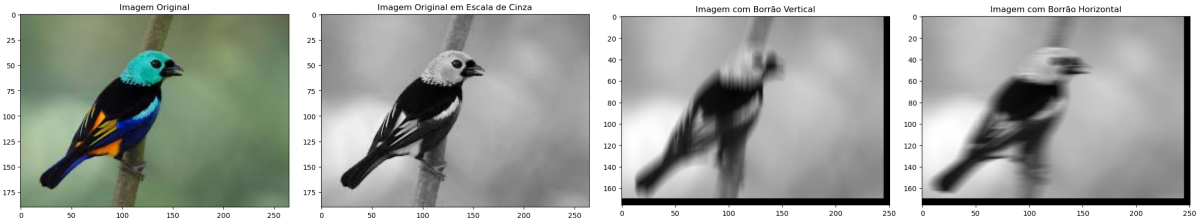

Figura 10: Fonte: WikiAves

O código acima realiza a convolução da imagem utilizando um filtro 2D vertical (filtro Sobel), empregado para a detecção de bordas em uma imagem. Como a imagem de saída é menor que a imagem de entrada, é possível observar a presença de barras pretas na parte inferior e à direita da imagem resultante, que correspondem a valores zero. O fato de o filtro Sobel ser particularmente usado na detecção de bordas verticais se deve ao fato de ele possuir valores diferentes de zero na coluna central, o que torna

as bordas verticais mais evidentes após a aplicação do filtro. O resultado desse processo será duas imagens: a imagem original e a imagem resultante da convolução, na qual as bordas detectadas na imagem original são claramente evidenciadas. Além disso, são demonstradas diferentes versões deste filtro para uma melhor compreensão do funcionamento do algoritmo [24].

3.3 Técnicas de Regularização

3.3.1 Dropout (ou Abandono)

Em busca de bons modelos preditivos, a capacidade de generalização torna-se uma característica importante do modelo. O ideal é que o modelo apresente um bom desempenho em dados não vistos, evitando o overfitting, que ocorre quando o modelo aprende detalhes e ruídos dos dados de treinamento. A teoria clássica da generalização sugere que a simplicidade do modelo é importante para reduzir essa lacuna entre o desempenho no conjunto de treinamento e no conjunto de teste.

3.3.2 A Teoria da Simplicidade

A simplicidade de um modelo pode ser entendida de várias formas, como, por exemplo, um número reduzido de dimensões, a regularização dos parâmetros ou ainda a suavidade das funções. A suavidade, em caso particular, refere-se à capacidade de um modelo de não ser excessivamente sensível a pequenas variações nos dados de entrada. Por exemplo, um teste dessa resiliência seria: ao classificar imagens, adicionar ruído aleatório deve ter um impacto mínimo na classificação final.

Bishop (1995) formalizou essa noção ao demonstrar que o treinamento com ruído de entrada pode ser visto como uma forma de regularização. Essa conexão matemática ilustra a importância da suavidade e resistência a perturbações, conceitos que são importantes para a construção de um bom modelo. [4]

3.3.3 O Surgimento do Dropout

Para abordar o **dropout** como técnica de regularização em redes neurais, é conveniente entender sua popularidade e como foi relevante no contexto de classificação de imagens, como apresentada por G. E. Hinton em 2012 [13]. A ideia do abandono foi, posteriormente, detalhada por Srivastava et al. (2014) como uma extensão das ideias discutidas por Bishop. O método de abandono consiste em injetar ruído durante o treinamento de redes neurais, especificamente nas camadas internas. A técnica de **dropout** é chamada de "dropout"porque envolve "abandonar"aleatoriamente uma fração de neurônios durante o treinamento. Isso significa que, em cada iteração, uma parte dos neurônios é desativada, algo que força a rede a aprender representações melhores [4].

A razão do **dropout** ser usado é devido a sua capacidade de aumentar a precisão de redes neurais, chegando a melhorar de 1 a 2% em modelos avançados. Embora possa parecer um pequeno incremento, essa diferença é significativa quando a precisão já é elevada, isso porque, por exemplo, ao passar de 95% para 97%, a taxa de erro é reduzida em cerca de 40% (de 5% para 3%) [13].

Uma abordagem possível e razoável para aprimorar esse processo é o emprego da técnica de **Dropout**, que será discutida a seguir.

3.3.4 Implementação do Dropout

Conforme descrito no capítulo 11.6 do livro *Hands-On Machine Learning*, o funcionamento do **dropout** é bastante direto: em cada etapa de treinamento, existe uma probabilidade p de que cada neurônio (exceto os neurônios de saída) seja temporariamente "desativado". Isso significa que ele será ignorado naquela etapa específica, mas poderá estar ativo na próxima. A taxa de dropout, ou p, é frequentemente definida em torno de 50%. Após o treinamento, os neurônios permanecem ativos permanentemente. A simplicidade desse algoritmo é vista como surpreendente, especialmente ao considerar o impacto que ele causa na performance de redes neurais [13].

Uma abordagem prática para essa implementação é, primeiramente, gerar amostras de uma distribuição uniforme no intervalo entre 0 e 1 para cada nó da camada. Em seguida, mantêm-se os nós para os quais a amostra correspondente é maior que $1 - p$, desativando os demais [4].

A analogia interessante descrita no capítulo sobre Dropout compara sua aplicação ao funcionamento de uma empresa que precisa se adaptar à incerteza. Se os colaboradores decidissem aleatoriamente ir ou não trabalhar, a empresa teria que distribuir tarefas essenciais entre várias pessoas e fomentar a colaboração geral, tornando o modelo mais resiliente e menos dependente de poucos indivíduos específicos. De maneira análoga, os neurônios que passam pelo Dropout são forçados a desenvolver capacidades

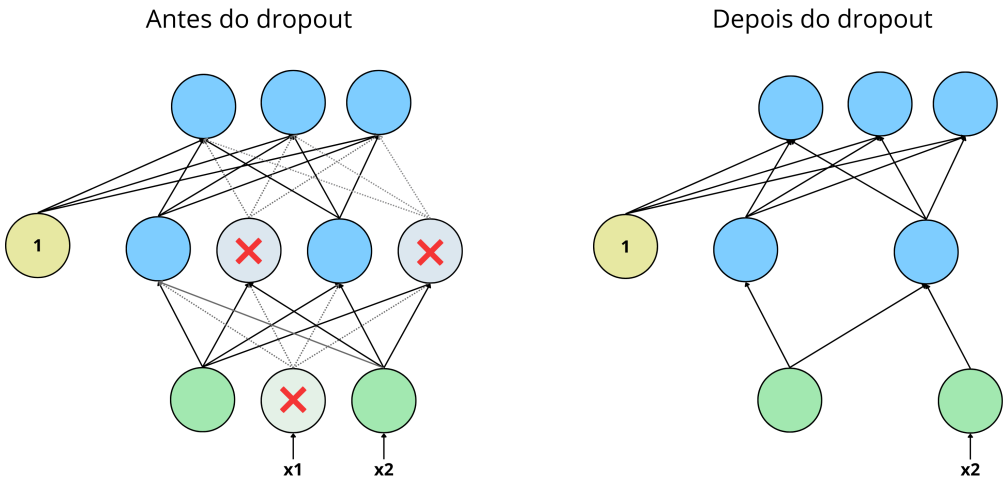

Figura 11: Dropout

independentes, uma vez que não podem se especializar excessivamente ou depender de neurônios vizinhos. Isso resulta em uma rede neural mais consistente, capaz de generalizar melhor [13].

O **Dropout** pode ser interpretado como a formação de uma nova rede neural a cada etapa de treinamento. Como cada neurônio tem uma chance de estar ativado ou desativado, isso gera, teoricamente, 2^N redes diferentes, onde N é o número total de neurônios. Embora essas redes compartilhem os mesmos pesos, cada instância de treinamento funciona como uma rede única. Dessa forma, ao término do treinamento, a rede neural final se comporta como uma média de várias redes menores, o que contribui para melhorar a capacidade de generalização e a robustez do modelo.

1. **Ideia do Dropout como "Redes Diferentes"**: Ao usar o Dropout, temporariamente "desliga-se"ou "ativa-se"aleatoriamente alguns neurônios a cada etapa de treinamento. Isso significa que, em cada rodada, a rede neural terá uma configuração ligeiramente diferente, visto que alguns neurônios estarão inativos e outros ativos. Portanto, a cada nova rodada, haverá uma rede ligeiramente diferente.

2. **Quantas Redes Diferentes?** Se considerar todas as possíveis combinações em que cada neurônio pode estar ativo ou inativo, haverá 2^N redes potenciais (onde N é o número total de neurônios). Embora, na prática, não se treine todas essas redes separadamente, o *dropout* simula essa variedade ao longo do treinamento.

3. **Compartilhamento de Pesos**: Mesmo com as mudanças na rede a cada etapa, todas essas "configurações"de redes compartilham os mesmos pesos, que são ajustados com base nos dados de cada rodada. Portanto, elas se adaptam em conjunto.

4. **Resultado Final – Uma Média de Redes**: Ao final do treinamento, o modelo pode ser visto como uma média de todas essas "pequenas redes"que foram criadas com o *dropout*. Cada configuração ensina algo ligeiramente diferente aos pesos e isso os ajuda a criar uma rede melhor e menos dependente de neurônios específicos. Isso reduz o risco de *overfitting*, pois a rede aprende a ser mais geral.

3.3.5 Quebrando a Co-adaptação

Um dos principais benefícios do abandono é que ele ajuda a evitar a co-adaptação dos neurônios. Em um estado de co-adaptação, as ativações de um neurônio são altamente dependentes de padrões específicos de ativações em neurônios anteriores, e isso leva a um modelo que se torna excessivamente ajustado aos dados de treinamento. A técnica de abandono interrompe essa dependência, promovendo uma forma de aprendizado que é mais semelhante à reprodução sexual, onde a diversidade genética é favorecida. Essa analogia sugere que a quebra da co-adaptação pode levar a representações mais gerais e adaptáveis.

O abandono é uma técnica implementada em diversas bibliotecas de aprendizado profundo, como TensorFlow e PyTorch. A simplicidade de uso e os resultados satisfatórios em melhorar a generalização

fazem dele uma escolha padrão no treinamento de redes neurais. Ao definir a fração de neurônios a serem abandonados, é possível controlar o grau de regularização aplicado.

3.3.6 Aumento de Dados

Uma última técnica de regularização, chamada de data augmentation (aumento de dados), consiste em gerar novas instâncias de treinamento a partir das já existentes, resultando em um aumento artificial do tamanho do conjunto de treinamento. Como consequência, isso reduz o overfitting, tornando-se uma técnica de regularização. [14]

Por exemplo, se o modelo foi projetado para classificar imagens de cogumelos, é possível aplicar leves transformações como deslocamento, rotação e redimensionamento em cada imagem do conjunto de treinamento, gerando novas versões dessas imagens e adicionando-as ao conjunto. Essas alterações ajudam o modelo a se tornar mais resiliente à posição, orientação e tamanho dos cogumelos nas fotos. Para melhorar a tolerância do modelo a condições de iluminação, uma alternativa é criar versões das imagens com diferentes níveis de contraste. Se os cogumelos apresentarem simetria, também é possível espelhar as imagens horizontalmente. Combinando todas essas técnicas, é possível expandir consideravelmente o conjunto de treinamento. [14]

O overfitting geralmente ocorre quando há um pequeno número de exemplos de treinamento. O aumento de dados adota a abordagem de gerar dados de treinamento adicionais a partir de seus exemplos existentes, aumentando-os usando transformações aleatórias que produzem imagens de aparência confiável. Isso ajuda a expor o modelo a mais aspectos dos dados com ligeiras perturbações, e isso faz com que o modelo consiga generalizar melhor.

A seguir, a aplicação da técnica de regularização na base de dados de aves.

```
1
2  data_augmentation = keras.Sequential(
3    [
4      layers.RandomFlip("horizontal",
5                        input_shape=(img_height,
6                                     img_width,
7                                     3)),
8      layers.RandomRotation(0.1),
9      layers.RandomZoom(0.1),
10   ]
11 )
```

Listing 5: Definição e Compilação de um Modelo Keras

As imagens a seguir foram selecionadas da base de dados de aves, junto com suas versões modificadas por meio de transformações aplicadas pelas funções *RandomFlip*, *RandomRotation* e *RandomZoom*.

Figura 12: Resultado após Data Augmentation Fonte: Kaggle

Constata-se que, de cada imagem, foram geradas 6 novas, o que resulta em um aumento de 6 vezes na base de dados. Esse aumento traz diversas vantagens, como, por exemplo, a melhoria na capacidade de generalização do modelo, ao expô-lo a variações de rotação e escala, reduzindo a propensão ao overfitting. Ademais, o incremento na quantidade de dados possibilita a criação de modelos com melhor capacidade de classificação, o que melhora o desempenho em dados reais.

4 Arquiteturas de Redes Neurais

Entre os anos de 2014 e 2015, algumas das principais arquiteturas foram desenvolvidas e proporcionaram uma grande mudança na forma como a classificação de imagens é feita até os dias de hoje.

As arquiteturas evoluíram com o passar do tempo, começando com o LeNet [18] e sendo aprimoradas na AlexNet [17] em 2012. Aprofundaram-se com contribuições de Zeiler e Fergus em 2013 [27] e a arquitetura VGG em 2014 [21]. Em 2014, surgiu o Inception, apresentado como GoogLeNet (Inception V1) por Szegedy et al., e refinado em versões subsequentes, como Inception V3 [23] e Inception-ResNet [22]. O desenvolvimento de redes neurais convolucionais (CNNs) proporcionou avanços importantes no campo de visão computacional, especialmente em tarefas como classificação, detecção e segmentação de objetos. Essas mudanças puderam melhorar a precisão dos modelos e também facilitaram a adaptação das redes para diferentes domínios e recursos computacionais.

4.1 AlexNet

Em 2012, foi publicado o artigo seminal *"ImageNet Classification with Deep Convolutional Neural Networks"*, desenvolvido por Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton, todos da Universidade de Toronto. Esse trabalho foi um marco na área de aprendizado profundo e inspirou, posteriormente, o desenvolvimento de novas arquiteturas. Foi apresentada a **AlexNet**, uma rede neural convolucional que obteve resultados muito bons na tarefa de classificação de imagens no ImageNet.

No paper [17], é descrito que o reconhecimento de objetos é fortemente dependente de métodos de aprendizado de máquina, além de que, para melhorar seu desempenho, é necessário utilizar conjuntos de dados maiores, modelos mais poderosos e boas técnicas de prevenção de overfitting. Anteriormente, conjuntos pequenos limitavam o desempenho, mas com a base ImageNet, com mais de 15 milhões de imagens em 22.000 categorias, possibilitou avanços que não se tinham visto até o momento. As Redes Neurais Convolucionais (CNNs), especialmente com o avanço das GPUs e otimizações de convoluções 2D, foram pioneiras no treinamento em grande escala. O trabalho apresenta uma das maiores CNNs treinadas até aquele momento e alcançou resultados de ponta nos desafios ILSVRC-2010 e ILSVRC-2012. A arquitetura proposta consistia em cinco camadas convolucionais e três camadas totalmente conectadas, com o uso de novas técnicas para melhorar o desempenho e reduzir o tempo de treinamento, com a inclusão de boas estratégias para evitar overfitting. Os autores atestaram que, com GPUs mais poderosas, seria possível melhorar ainda mais os resultados.

Em relação ao conjunto de dados, vale destacar que o **ImageNet** é um banco de dados com mais de 15 milhões de imagens rotuladas em alta resolução, distribuídas por cerca de 22.000 categorias, coletadas da web e rotuladas por meio do Amazon Mechanical Turk. O ILSVRC, competição anual desde 2010, utiliza um subconjunto com 1.000 categorias e aproximadamente 1.000 imagens por categoria, totalizando 1,2 milhão de imagens de treinamento, 50.000 de validação e 150.000 de teste.

As imagens, com resoluções variadas, foram redimensionadas para 256x256 pixels (ajustando a menor dimensão para 256 e cortando o centro). A normalização foi realizada subtraindo a média dos pixels sobre o conjunto de treinamento. O desempenho foi avaliado por meio da taxa de erro top-1 (onde a resposta correta é a mais provável) e da taxa de erro top-5 (onde a resposta correta está entre as cinco mais prováveis).

4.1.1 Arquitetura

A arquitetura da rede é composta por oito camadas treináveis: cinco convolucionais e três totalmente conectadas.

Não-Linearidade ReLU

A função tradicional $f(x) = \tanh(x)$ ou $f(x) = \frac{1}{1+e^{-x}}$ treina mais lentamente em comparação com

$$f(x) = \max(0, x),$$

a **ReLU**. Redes com ReLUs treinam significativamente mais rápido, como mostrado no CIFAR-10, onde atingem 25% de erro **seis vezes mais rápido** que redes com **tanh**. As **ReLUs** aceleram o aprendizado, especialmente em redes profundas com grandes conjuntos de dados.

4.1.2 Treinamento e abordagem

Devido à limitação de memória da **GPU GTX 580** (3 GB), a rede foi distribuída entre duas GPUs. Cada GPU processa metade dos núcleos, resultando em uma redução nas taxas de erro top-1 e top-5 em

1,7% e 1,2%, respectivamente, com um tempo de treinamento ligeiramente mais rápido, em comparação com o uso de uma única GPU.

Normalização de Resposta Local: A normalização de resposta local melhorou a generalização ao implementar inibição lateral entre as saídas dos neurônios. Ela reduziu as taxas de erro top-1 e top-5 em 1,4% e 1,2%, respectivamente. No CIFAR-10, uma rede com normalização obteve 11% de erro, contra 13% sem ela [16].

Pooling Sobreposto: Foi utilizado **pooling sobreposto** com s=2s=2 e z=3z=3, o que resultou em uma redução das taxas de erro top-1 e top-5 em 0,4% e 0,3%, respectivamente, em comparação com o pooling não sobreposto. Esse esquema também aumentou a resistência ao overfitting durante o treinamento.

Portanto, o paper descreve a arquitetura da rede como composta por oito camadas treináveis: cinco convolucionais e três totalmente conectadas. A saída da última camada totalmente conectada é alimentada em uma softmax de 1.000 vias. A rede maximiza a regressão logística multinomial, maximizando a média da log-probabilidade da classe correta.

As camadas convolucionais 2, 4 e 5 conectam-se apenas aos mapas da mesma GPU, enquanto a camada 3 se conecta a todos os mapas da camada 2. Camadas de normalização de resposta e max-pooling seguem as camadas convolucionais, e a última camada de max-pooling é aplicada antes da camada totalmente conectada. A função ReLU é aplicada em todas as camadas convolucionais e totalmente conectadas. A primeira camada convolucional usa 96 núcleos 11×11×3, com stride de 4 pixels, para filtrar a imagem 224×224×3 [17].

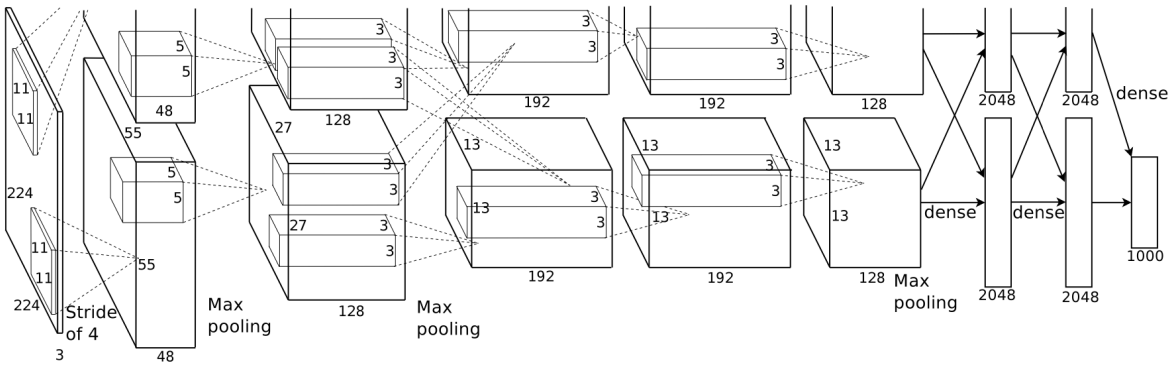

Figura 13: Uma ilustração da arquitetura da nossa CNN.
Imagem extraída do paper ImageNet Classification with Deep Convolutional Neural Networks [17].

4.1.3 Técnicas de regularização Aplicadas

No trabalho de Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton, a arquitetura de rede neural possui 60 milhões de parâmetros, algo que, levando em consideração as 1.000 classes do ILSVRC, não é suficiente para aprender sem sofrer overfitting. Para combater isso, os autores propõem duas abordagens principais, descritas a seguir.

Aumento de Dados

A primeira estratégia utilizada pelos autores é o aumento de dados, que consiste em ampliar artificialmente o conjunto de dados de treinamento por meio de transformações que preservam os rótulos. Duas formas de aumento são aplicadas:

- **Translações e reflexões horizontais:** São extraídos patches de 224×224 das imagens originais (e suas reflexões horizontais), ampliando o conjunto de dados por um fator de 2048.
- **Alteração das intensidades RGB:** Os autores aplicam PCA sobre as intensidades dos pixels RGB em todo o conjunto de treinamento, adicionando variações baseadas nos componentes principais das imagens. Esse processo melhora a generalização e resulta em uma redução de mais de 1% na taxa de erro top-1.

Essas transformações são feitas com boa performance, pois as imagens aumentadas são geradas durante o treinamento, sem a necessidade de armazenamento em disco.

Dropout

Além do aumento de dados, os autores utilizam a técnica de *dropout* para reduzir o overfitting. Essa técnica, como descrito anteriormente, desliga aleatoriamente metade dos neurônios durante o treinamento, forçando a rede a aprender características mais importantes. Durante o teste, todos os neurônios são utilizados, mas suas saídas são multiplicadas por 0,5. O *dropout* ajuda a reduzir o overfitting, embora dobre o número de iterações necessárias para a convergência do modelo.

4.1.4 Treinamento

Os autores relatam que os modelos foram treinados utilizando descida de gradiente estocástico, com tamanho de lote de 128 exemplos, momento de 0,9 e decaimento de peso de 0,0005. Eles destacam que esse pequeno valor de decaimento foi importante para reduzir o erro de treinamento, funcionando como um regularizador e também como uma técnica que ajuda na aprendizagem efetiva do modelo.

Além disso, os autores inicializaram os pesos das camadas com uma distribuição Gaussiana de média zero e desvio padrão de 0,01, enquanto os vieses nas camadas convolucionais e ocultas foram definidos como 1.

4.1.5 Resultados

No artigo seminal de **Krizhevsky, Sutskever e Hinton** sobre a arquitetura **AlexNet**, os autores apresentam os seguintes resultados no **ILSVRC-2010**:

- **Taxas de erro do modelo AlexNet:**
  - Top-1: 37,5%
  - Top-5: 17,0%
- Os melhores resultados obtidos na competição ILSVRC-2010 foram de 47,1% e 28,2%, com uma abordagem que média as previsões de seis modelos de codificação esparsa. Após a competição, os melhores resultados publicados foram de 45,7% e 25,7%, utilizando uma abordagem com Fisher Vectors (FVs).

Modelo	Top-1	Top-5
Codificação esparsa	$47,1\%$	$28,2\%$
SIFT + FVs	$45,7\%$	$25,7\%$
CNN (AlexNet)	$37,5\%$	$17,0\%$

No **ILSVRC-2012**, os autores reportam que o modelo AlexNet alcançou uma taxa de erro de **18,2% no Top-5**, e ao combinar a média de cinco redes CNN semelhantes, essa taxa foi reduzida para **16,4%**. Além disso, ao treinar uma CNN com uma camada convolucional adicional, a taxa de erro foi de **16,6%**.

A segunda melhor entrada na competição obteve **26,2%** no Top-5, usando uma abordagem baseada em Fisher Vectors (FVs).

No conjunto de dados **ImageNet Fall 2009**, com 10.184 categorias e 8,9 milhões de imagens, o modelo AlexNet obteve as seguintes taxas de erro:

- Top-1: 67,4%
- Top-5: 40,9%

Os melhores resultados publicados neste conjunto de dados foram de **78,1%** e **60,9%**, respectivamente [17].

4.2 Xception

Outra arquitetura que surgiu posteriormente, também inspirada pelo artigo seminal, foi a Xception. No artigo [3] publicado em 2016, François Chollet faz uma descrição das arquiteturas CNN em ordem cronológica e, após isso, destaca o Inception.

O autor diz que o Inception se destaca pelo módulo Inception, que, ao contrário das redes VGG, usa uma pilha de módulos em vez de camadas convolucionais simples. Esses módulos aprendem representações mais ricas com menos parâmetros, superando as convoluções tradicionais.

A seguir, o autor descreve a hipótese do Inception, onde afirma que a hipótese central do Inception é a de que as correlações entre canais e as espaciais podem ser suficientemente desacopladas para que seja melhor tratá-las separadamente. O módulo Inception realiza isso aplicando convoluções 1x1 para reduzir as dimensões e, em seguida, utiliza convoluções 3x3 ou 5x5 para mapear as correlações espaciais. Dessa forma, o processo de aprendizado das correlações entre canais e espaciais é dividido em etapas independentes, melhorando a performance do modelo.

4.2.1 Arquitetura

O autor então propõe a arquitetura Xception, que seria uma rede neural convolucional baseada em camadas de convolução separáveis em profundidade, que desacopla completamente o mapeamento das correlações entre canais e as correlações espaciais. Além disso, o autor relata que a arquitetura possui 36 camadas convolucionais divididas em 14 módulos, com conexões residuais lineares, exceto no primeiro e último módulo. O modelo é fácil de definir e modificar, levando apenas 30 a 40 linhas de código em bibliotecas como Keras ou TensorFlow-Slim. Uma implementação de código aberto está disponível no módulo Keras Applications, sob a licença MIT.

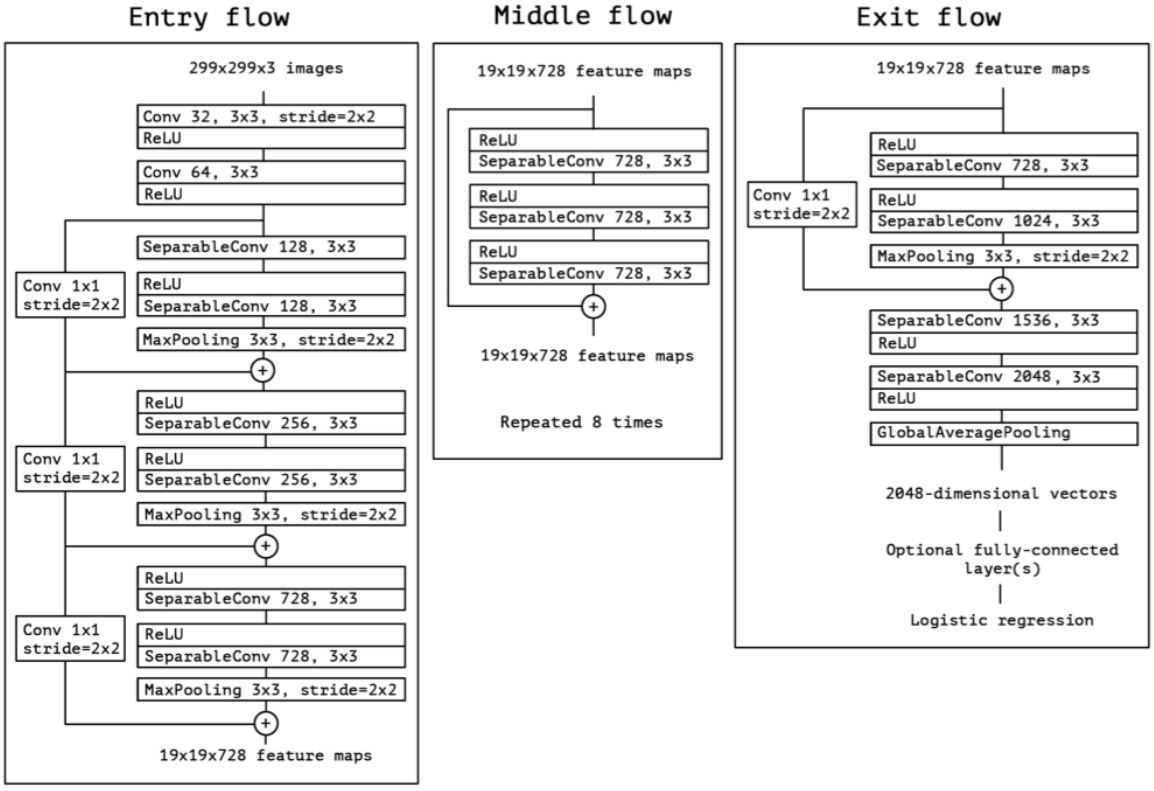

Figura 14: Uma ilustração da arquitetura da nossa CNN.
Imagem extraída do paper ImageNet Classification with Deep Convolutional Neural Networks [3].

4.2.2 Técnicas de regularização e Abordagem

O autor descreve que o modelo Xception envolveu o uso de decay de peso de 1×10^{-5}, enquanto o modelo Inception V3 utilizou 4×10^{-5}, sendo que a escolha do valor para o Xception não foi otimizada de forma

extensiva. Esse valor foi mantido em ambos os experimentos de ImageNet e JFT. No caso do dropout, o modelo Xception utilizou uma taxa de 0,5 antes da regressão logística no experimento do ImageNet, mas foi omitido no JFT devido ao grande tamanho do conjunto de dados, que tornava o overfitting improvável. Além disso, a torre de perda auxiliar, que é opcional no Inception V3 e serve como um mecanismo adicional de regularização, não foi incluída em nenhum dos modelos para simplificação.

4.2.3 Treinamento

O autor descreve que as redes foram implementadas no TensorFlow e treinadas em 60 GPUs NVIDIA K80. Para os experimentos no ImageNet, utilizou-se gradiente descendente síncrono, enquanto no JFT foi usado o gradiente assíncrono para acelerar o processo. Os experimentos com ImageNet levaram cerca de 3 dias, enquanto os de JFT duraram mais de um mês, sem que fosse alcançada a convergência completa.

4.2.4 Desempenho e comparação com outros modelos

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	**0.790**	**0.945**

Figura 15: Uma ilustração da arquitetura da nossa CNN.
Imagem extraída do paper ImageNet Classification with Deep Convolutional Neural Networks [3].

O Xception superou o Inception V3, especialmente em grandes datasets como o JFT. No ImageNet, teve um desempenho ligeiramente superior. Sua arquitetura de convoluções separáveis por profundidade se mostrou com melhor desempenho em tarefas de grande escala e superou outros modelos, como a ResNet, no ImageNet. Isso sugere que o Xception é melhor para classificações complexas.

5 Resultados experimentais

5.1 Treinamento com Modelo Simplificado

O primeiro treinamento foi feito com 10 Épocas.

```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

Listing 6: Treinamento do modelo com 10 Épocas

5.1.1 Técnicas de Regularização Aplicadas

Para fortalecer a generalização do modelo de classificação de aves, foi aplicada a técnica de dropout nas camadas totalmente conectadas finais, usando a base de dados aumentada. Uma taxa de dropout de 50% foi configurada, o que faz com que, em cada iteração de treinamento, metade dos neurônios nessas camadas seja desativada aleatoriamente.

```
    model = Sequential([
      data_augmentation,
      layers.Rescaling(1./255),
      layers.Conv2D(16, 3, padding='same', activation='relu'),
      layers.MaxPooling2D(),
      layers.Conv2D(32, 3, padding='same', activation='relu'),
      layers.MaxPooling2D(),
      layers.Conv2D(64, 3, padding='same', activation='relu'),
      layers.MaxPooling2D(),
      layers.Dropout(0.5),
      layers.Flatten(),
      layers.Dense(128, activation='relu'),
      layers.Dense(num_classes, name="outputs")
    ])
```

Listing 7: Definição e Compilação de um Modelo Keras

5.1.2 Treinamento

Treinamento do modelo com 10 Épocas

Primeiramente, é realizado o treinamento de um modelo de rede neural ao longo de 10 épocas, utilizando um conjunto de dados de treinamento (train_ds) e um conjunto de validação (val_ds). A base de dados de validação é utilizada para monitorar o desempenho do modelo em dados inéditos, sem interferir diretamente no ajuste dos pesos. A adoção desse procedimento permite identificar momentos em que o modelo se ajusta adequadamente ao conjunto de treinamento, mas começa a apresentar sinais de overfitting. Esse fenômeno ocorre quando o modelo passa a memorizar o conjunto de treinamento em vez de aprender padrões generalizáveis.

O método fit ajusta os parâmetros do modelo com base nos dados de treinamento, enquanto a avaliação em cada época com o conjunto de validação permite monitorar o desempenho do modelo em dados não vistos durante o ajuste. Esse processo tem como intuito otimizar a capacidade do modelo de generalizar para novos dados, reduzindo o erro de previsão e aprimorando a precisão nas classificações.

```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

Listing 8: Treinamento do modelo com 10 Épocas

O gráfico abaixo ilustra a curva de aprendizagem e a curva de perda do modelo.

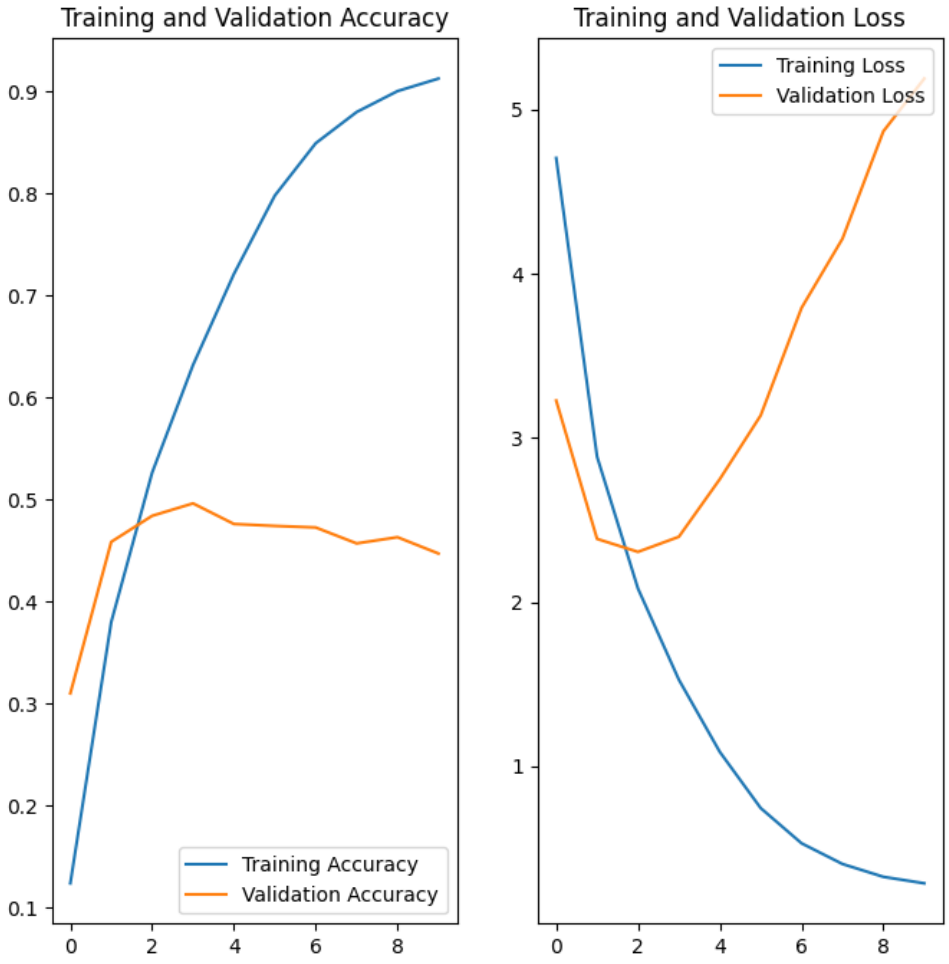

Figura 16: Grafico de Acurácia de Treinamento e Validação e Perda de Treinamento e Validação

No primeiro gráfico *Training and Validation Accuracy*, é possível observar que, embora a curva azul "Training accuracy"(acurácia de treinamento) evolua em função do tempo de treinamento, a curva *Validation Accuracy*, que inicialmente diminui, começa a subir após um certo ponto, indicando que o modelo está se ajustando excessivamente aos dados de treinamento. Isso acontece porque, embora o modelo tenha um bom desempenho nos dados que viu durante o treinamento, sua capacidade de generalização para novos dados está comprometida. Em outras palavras, é possível dizer que o modelo se torna muito bom em aprender com seus próprios dados, mas não consegue aplicar esse conhecimento a novos conjuntos de dados. [8]

Algumas imagens mostram a performance do modelo em novos dados (dados independentes do processo de treinamento e validação), onde é possível observar uma quantidade significativa de erros de classificação e scores baixos. Isso pode indicar uma grande incerteza nas previsões do modelo, algo que indica dificuldades em generalizar para dados não vistos durante o treinamento.

Resultado para o modelo com 10 Épocas, sem dropout e sem data augmentation

Conclusão:

Figura 17: Predição para novos dados. Fonte: Kaggle

A análise das métricas foi capaz de revelar que, no modelo com 10 épocas, sem técnicas de regularização e aumento de dados, o erro nas previsões é considerável, e o modelo apresenta dificuldades para equilibrar precisão e revocação, o que é refletido pelo F1 score baixo. Por outro lado, a introdução de dropout e data augmentation no modelo com 15 épocas parece ter melhorado seu desempenho, permitindo uma melhor generalização e um equilíbrio mais notável entre as métricas de precisão e revocação, algo que indica que o uso dessas técnicas pode ser muito importante para melhorar a performance do modelo, principalmente em casos com dados limitados.

Treinamento do modelo com 15 Épocas

Após a aplicação do aumento de dados e da técnica de regularização por abandono, o modelo é treinado em 15 épocas.

```
epochs=15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

Listing 9: Treinamento do modelo com 15 Épocas

O gráfico abaixo ilustra a curva de aprendizagem e a curva de perda do modelo após o novo treinamento.

No gráfico à direita (Training and Validation Loss), que representa a perda de treinamento e validação, observa-se que tanto a curva de treinamento quanto a de validação diminuem ao longo do treinamento, conforme esperado, uma vez que o modelo ajusta seus parâmetros para minimizar a perda. A perda de validação apresenta uma queda mais acentuada nas primeiras épocas, seguida por uma redução gradual, indicando que o modelo ajusta bem seus parâmetros, sem sobreajustar-se aos dados de treinamento.

No gráfico à esquerda (Training and Validation Accuracy), observa-se que ambas as curvas evoluem de forma consistente ao longo das épocas. Essa evolução sugere que o modelo conseguiu aprimorar sua capacidade de generalização para novos dados. As técnicas implementadas ajudam o modelo a se ajustar adequadamente aos dados de treinamento e a apresentar um desempenho consistente em dados não vistos, algo que reflete um aprendizado consistente.

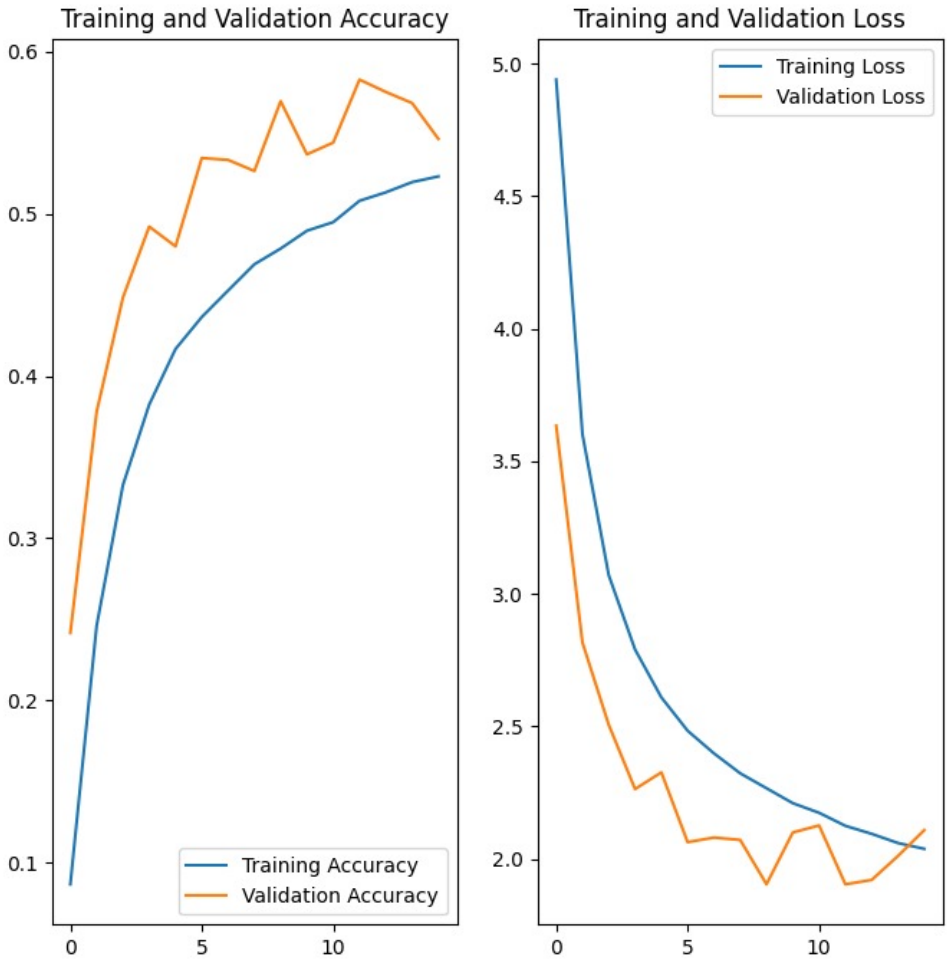

Figura 18: Resultado Acurácia e Perda 15

Treinamento em 30 Épocas

A fim de melhorar ainda mais o desempenho do modelo, a quantidade de épocas foi ajustada para 30.

```
epochs=30
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

Listing 10: Treinamento do modelo com 30 Épocas

O gráfico abaixo ilustra a curva de aprendizagem e a curva de perda do modelo após o novo treinamento com 30 Épocas.

No gráfico à direita (Training and Validation Loss), que representa a perda de treinamento e validação, observa-se que tanto a curva de treinamento quanto a de validação diminuem ao longo do treinamento, conforme esperado, sem indicação de overfitting.

No gráfico à esquerda (Training and Validation Accuracy), observa-se que as curvas evoluem ao longo das épocas, o que sugere uma melhoria na generalização ao longo do tempo.

5.1.3 Métricas de desempenho

A fim de verificar o desempenho do modelo com mais épocas, algumas métricas foram calculadas.

O modelo apresenta resultados que indicam uma performance moderada, com resultados consistentes entre acurácia e recall. Embora a precisão seja ligeiramente superior, o F1-Score mostra que o modelo

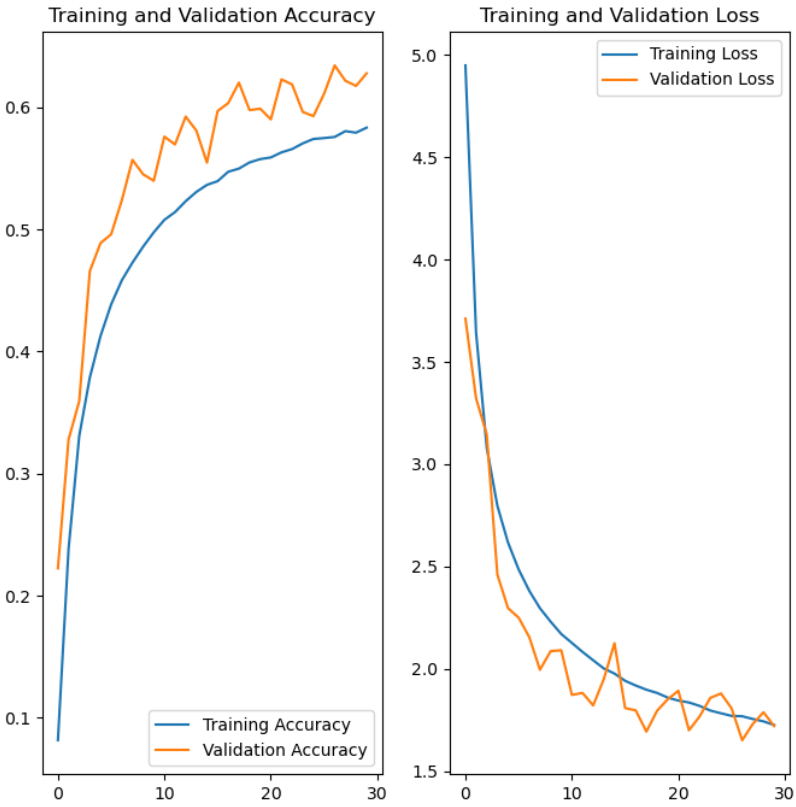

Figura 19: Resultado após o aumento de Épocas para 30

```
Acurácia: 0.6488
Precisão (Weighted): 0.6848
Revocação (Weighted): 0.648
F1 Score (Weighted): 0.6316
```

Figura 20: Métricas de desempenho após 30 Épocas

pode ser melhorado.

5.1.4 Análise qualitativa

Ao analisar os resultados de maneira qualitativa, é possível verificar que o modelo se confunde entre aves que são visualmente parecidas. Até para um ser humano, isso poderia causar confusão, como no exemplo a seguir.

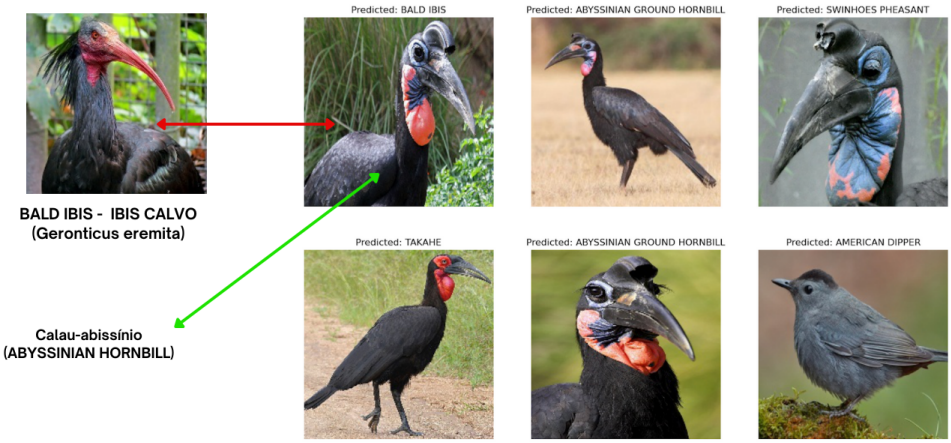

Figura 21: Fonte: Kaggle

O modelo classifica a ave como BALD IBIS (Íbis Calvo), mas, na verdade, trata-se de uma ABYSSINIAN HORNBILL (Calau-abissínio). Esse é um exemplo de confusão do modelo.

5.2 Treinamento com Transfer Learning utilizando Xception

A arquitetura que inclui o Xception (modelo pré-treinado) foi utilizada para comparar os resultados com o modelo da arquitetura implementada anteriormente. Essa técnica de associação com um modelo pré-treinado é conhecida como **Transfer Learning**. Essa abordagem é interessante, pois aproveita o conhecimento pré-treinado do ImageNet, o que acelera o treinamento e melhora o desempenho em tarefas com conjuntos de dados menores.

```
1
2  base_model = Xception(
3      include_top=False,
4      weights='imagenet',
5      input_shape=(img_height, img_width, 3)
6  )
7
8  base_model.trainable = False
9
10 model = models.Sequential([
11     layers.InputLayer(input_shape=(img_height, img_width, 3)),
12     base_model,
13     layers.GlobalAveragePooling2D(),
14     layers.Dense(1024, activation='relu'),
15     layers.Dense(num_classes, activation='softmax')
16 ])
```

Listing 11: Treinamento com Xception 10 Épocas

5.2.1 Resultados

O gráfico abaixo ilustra a curva de aprendizagem e a curva de perda do Xception após o treinamento com 30 épocas.

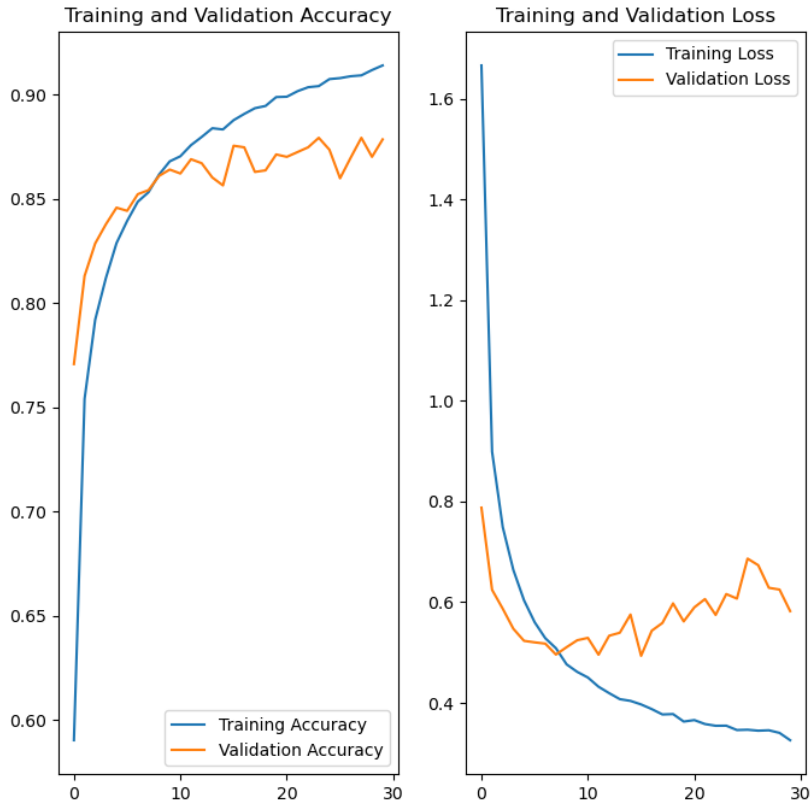

Figura 22: Resultado para xception com 30 Épocas

No gráfico à direita (Training and Validation Loss), podemos ver que a Perda de Treinamento representa uma diminuição, enquanto a de Validação apresenta um aumento lento, algo que pode indicar overfitting.

No gráfico à esquerda (Training and Validation Accuracy), observa-se que as curvas evoluem de forma consistente ao longo das épocas, algo que sugere a melhora na generalização ao longo do tempo.

5.2.2 Métricas de desempenho

Para verificar o desempenho do modelo com a nova arquitetura associada, algumas métricas foram calculadas:

```
Top-5 Accuracy: 0.9878
Accuracy: 0.9010
Recall: 0.9010
Precision: 0.9185
F1-Score: 0.8987
```

Figura 23: Métricas de desempenho após 30 Épocas

O modelo apresentou um bom desempenho, com Top-5 Accuracy de 98,78%, o que indica que a resposta correta está entre as 5 mais prováveis em quase 99% das vezes. A acurácia de 90,10% e o

recall de 90,10% indicam boa capacidade de classificação e identificação das classes corretas. A precisão de 91,85% também reflete uma baixa taxa de falsos positivos, e o F1-Score de 89,87% demonstra um equilíbrio sólido entre precisão e recall. Portanto, o modelo pode ser considerado um bom modelo que generaliza bem para novos dados.

5.2.3 Exemplos de classificação

A seguir temos alguns exemplos da classificação do modelo com a técnica Transfer Learning

Figura 24: Exemplos classificação Xception. Fonte: Kaggle

6 Conclusão

O presente trabalho apresentou uma abordagem prática para a classificação de aves utilizando redes neurais convolucionais (CNNs), sendo capaz de demonstrar o uso dessas técnicas de classificação de imagens no campo da conservação ambiental e da ornitologia. O estudo explorou a performance das CNNs para tarefas de classificação de imagens e destacou a importância de técnicas de regularização, como dropout, e estratégias de aumento de dados para lidar com os desafios de sobreajuste e limitações de amostragem.

Os resultados iniciais do modelo treinado sem a inclusão dessas técnicas mostraram limitações, como alto erro de previsão e um desempenho inconsistente em dados de validação e teste, algo que evidenciou a necessidade de implementar práticas que promovam a generalização, especialmente em cenários onde a diversidade dos dados é importante para garantir um bom modelo.

Ao aplicar as técnicas de dropout e data augmentation, o modelo mostrou uma evolução na capacidade de generalização, com curvas de aprendizado estáveis e uma melhor adaptação aos dados não vistos. No entanto, o modelo ainda pode ser melhorado, como as métricas evidenciaram. Ademais, com a adição do modelo Xception, através da técnica de Transfer Learning, foi possível observar uma melhoria nos resultados, e as métricas refletem essa evolução.

O estudo foi capaz de destacar o uso de ferramentas de aprendizado profundo em iniciativas voltadas para a conservação ambiental e os desafios para a construção de um modelo de classificação de aves no campo da biodiversidade.

Referências

[1] N. Al-Azzam and I. Shatnawi. Comparando modelos de aprendizado de máquina supervisionados e semissupervisionados no diagnóstico de câncer de mama. *Annals of Medicine and Surgery*, 62:53–64, 2021.

[2] Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 26–33, 2001.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[4] Dive into Deep Learning. *Dropout*, chapter 5.6, pages 1–2. Dive into Deep Learning, 2024. Acessado em 31 de agosto de 2024.

[5] HA Essa, E. Ismaiel, and MFA Hinnawi. Detecção baseada em características de câncer de mama usando rede neural convolucional e engenharia de características. *Scientific Reports*, 14:22215, 2024.

[6] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*, chapter 3, pages 119–160. Pearson, 4 edition, 2020. Intensity Transformations and Spatial Filtering.

[7] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, chapter 1, pages 25–26. O'Reilly Media, 2nd edition, 2019.

[8] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, chapter 1, pages 28–29. O'Reilly Media, 2019. Acessado em 31 de agosto de 2024.

[9] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, chapter 1, pages 29–30. O'Reilly Media, 2019. Acessado em 31 de agosto de 2024.

[10] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, chapter 1, page 30. O'Reilly Media, 2019. Acessado em 31 de agosto de 2024.

[11] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, chapter 1, page 32. O'Reilly Media, 2019. Acessado em 31 de agosto de 2024.

[12] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, chapter 13, page 357. O'Reilly Media, 2019. Acessado em 31 de agosto de 2024.

[13] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, chapter 11.9, page 357. O'Reilly Media, 2019. Acessado em 01 de agosto de 2024.

[14] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, chapter 11.9, pages 31–32. O'Reilly Media, 2nd edition, 2019.

[15] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[16] Alex Krizhevsky. Cuda-convnet. `https://code.google.com/p/cuda-convnet/`, 2012. Acesso em: 15 dez. 2024.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25, pages 1097–1105. NeurIPS, 2012. Accessed: 2024-12-15.

[18] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

[19] DA Omondiagbe, S. Veeramani, and AS Sidhu. Técnicas de classificação de aprendizado de máquina para diagnóstico de câncer de mama. *IOP Conference Series: Materials Science and Engineering*, 495:012033, 2019.

[20] A. Rasool and et al. Modelos preditivos baseados em aprendizado de máquina aprimorados para diagnóstico de câncer de mama. *International Journal of Environmental Research and Public Health*, 19:3211, 2022.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[22] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2015.

[24] Svitla Team. Math at the heart of cnn, 2023. Acessado em: 04 Nov. 2024.

[25] TensorFlow. Classificação de imagens. `https://www.tensorflow.org/tutorials/images/classification?hl=pt-br`, 2023.

[26] WH Wolberg, WN Street, and OL Mangasarian. Conjunto de dados de câncer de mama wisconsin (diagnóstico). `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)`, 1992. Figshare.

[27] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *arXiv preprint arXiv:1311.2901*, 2013.