Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Classificação de espécies de aranhas utilizando Transformers Visuais e Redes Convolucionais

Arthur Teixeira Magalhães

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisora: Nina S. T. Hirata

Durante o desenvolvimento deste trabalho, o autor recebeu auxílio financeiro da FAPESP – processo nº 2015/22308-2 e 2022/14286-2

São Paulo

2023

O conteúdo deste trabalho é publicado sob a licença CC BY 4.0
(Creative Commons Attribution 4.0 International License)

Agradecimentos

Believe you can and you’re halfway there.
— Theodore Roosevelt

À minha família, que sempre me apoiou, aos amigos que fiz e que me acompanharam, aos professores pelos ensinamentos valiosos e à professora Nina Hirata pela orientação neste trabalho.

Resumo

Arthur Teixeira Magalhães. **Classificação de espécies de aranhas utilizando Transformers Visuais e Redes Convolucionais**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

A identificação de espécies de aranhas é fundamental, não apenas para evitar acidentes, mas também para aplicações em diversos campos como agricultura, biologia, educação e pesquisa acadêmica. Propomos uma abordagem por meio de aprendizagem profunda para classificar espécies de aranhas a partir de suas imagens. Para tanto, utilizamos um conjunto de 25 espécies brasileiras e cerca de 25.000 imagens disponíveis publicamente no site INaturalist. Empregamos uma série de arquiteturas eficazes em classificação, incluindo ResNet, ResNeXt, ConvNeXt, ViT, Swin e MaxViT para avaliar suas eficácias e obter um modelo final preciso. Os resultados indicaram que, entre os Transformers, o MaxViT se destaca com uma acurácia de 88.84%. Quanto às redes convolucionais, ConvNeXt superou suas concorrentes, registrando uma acurácia de 89.38%, além de superar os outros modelos em uma comparação direta de suas métricas de desempenho. Em conjunto, esses resultados podem contribuir para o desenvolvimento de aplicações práticas destinadas a identificar aranhas e fornecer informações de interesse sobre as espécies.

Palavras-chave: Aprendizado de Máquina . Deep Learning . Visão Computacional . Classificação de Imagens . Redes Neurais Convolucionais . Vision Transformers .

Abstract

Arthur Teixeira Magalhães. **Spider species classification using Vision Transformers and Convolutional Neural Networks**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

The identification of spider species is essential not only to avoid accidents but also for applications in various fields such as agriculture, biology, education, and academic research. We propose a deep-learning approach to classify spider species based on their images. To this end, we used a set of 25 Brazilian species and around 25000 images publicly available on the INaturalist website. We employed several effective classification architectures, including ResNet, ResNeXt, ConvNeXt, ViT, Swin, and MaxViT to evaluate their effectiveness and obtain an accurate final model. The results indicated that, among the Transformers, MaxViT stands out with an accuracy of 88.84%. As for the convolutional networks, ConvNeXt outperformed its counterparts, registering an accuracy of 89.38%, as well as outperforming the other models in a direct comparison of their performance metrics. Together, these results could contribute to the development of practical applications aimed at identifying spiders and providing information of interest about the species.

Keywords: Machine Learning . Deep Learning . Computer Vision . Image Classification . Convolutional Neural Networks . Vision Transformers .

Lista de abreviaturas

ML	Aprendizado de Máquina (*Machine Learning*)
IA	Inteligência Artificial
NN	Rede Neural (*Neural Network*)
FF(N)	*Feed Forward (Network)*
FC	Totalmente Conectada (*Fully Connected*)
LR	Taxa de Aprendizado (*Learning Rate*)
CNN	Rede Neural Convolucional (*Convolutional Neural Network*)
ViT	Transformers Visuais (*Vision Transformer*)
NLP	Processamento de Linguagem Natural (*Natural Language Processing*)
MSA	*Multi-head self-attention*
W-MSA	*Window Based multi-head self-attention*
SW-MSA	*Shifted Window multi-head self-attention*
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Lista de figuras

2.1 Rede MLP 5
2.2 Operação de convolução 8
2.3 Convolução de bordas 8
2.4 Rede VGG16 9
2.5 Parâmetros VGG16 10
2.6 Arquitetura ResNet 11

2.7 Bloco ResNeXt 12
2.8 Arquitetura ConvNeXt 13
2.9 Encoder de um Transformer 14
2.10 Estrutura de multi-head attention 15
2.11 Vision Transformer 16
2.12 Swin Transformer 17
2.13 SW-MSA Cíclico 18
2.14 Arquitetura MaxViT 18

3.1 Exemplo de imagens por espécie. 21
3.2 Distribuição de imagens de aranhas do subconjunto selecionado. 22
3.3 Variação entre fotos da espécie *Alpaida grayi*. 24
3.4 Exemplo de transformação de imagem com flip, rotação e zoom. 25

4.1 Matriz de confusão ConvNeXt. 31
4.2 Comparação entre espécies Ctenus ornatus e Lycosa erythrognata. 32
4.3 Comparação entre espécies Corythalia conferta e Hasarius adansoni. . . 32
4.4 Mapas de gradients integrados das redes CNN. 33
4.5 Mapas de gradients integrados das redes ViT. 34

B.1 Matrizes de confusão dos modelos apresentados. 41

Lista de tabelas

3.1 Quantidade de parâmetros treináveis, não treináveis e totais por modelo. 25
3.2 Espaço de busca de hiperparâmetros para otimização. 26

4.1 Média e desvio padrão para métricas de validação por modelo. 28
4.2 Valores mínimos e máximos das métricas de validação por modelo. . . . 29
4.3 Tamanho do modelo em Megabytes (MB) e taxa de inferência em número de imagens por segundo (imgs/s). 29
4.4 Métricas no conjunto de teste. 30

A.1 Tabelas de hiperparâmetros utilizados para cada rede. 37

Sumário

1 Introdução **1**

2 Conceitos Fundamentais **3**

2.1 Aprendizado de Máquina 3

2.2 Arquitetura de Redes Neurais 3

2.2.1 Redes *Multilayer Perceptron* 4

2.2.2 Componentes Fundamentais 5

2.3 Redes Neurais Convolucionais 7

2.3.1 VGGNet 9

2.3.2 ResNet 10

2.3.3 ResNeXt 11

2.3.4 ConvNeXt 12

2.4 Transformers Visuais 13

2.4.1 Transformers 13

2.4.2 Vision Transformers 15

2.4.3 Swin Transformers 16

2.4.4 MaxViT 17

3 Desenvolvimento **20**

3.1 Coleta de Dados 20

3.2 Treinamento das Redes Neurais 21

3.2.1 Separação do Conjunto de Dados 21

3.2.2 PyTorch 22

3.2.3 PyTorch Lightning 23

3.2.4 Transformações de Imagens 23

3.2.5 Aumento de Dados 23

3.2.6 Redes Pré-Treinadas 24

3.2.7 Otimização de Hiperparâmetros 25

3.2.8 Treinamento das Redes 26
3.2.9 Análise dos Gradientes 27
3.3 Repositório 27

4 Resultados **28**
4.1 Resultados de Treinamento 28
4.1.1 Resultados de Teste 30
4.1.2 Matrizes de Confusão 30
4.2 Análise de Gradientes 31

5 Conclusão **35**

Apêndices

A Tabelas de Hiperparâmetros **37**

B Matrizes de Confusão **38**

Referências **43**

Capítulo 1

Introdução

Existem mais de 48.000 espécies de aranhas conhecidas no mundo, das quais mais de 4.500 são encontradas no Brasil (Secretaria Municipal da Saúde, 2020). São predadoras responsáveis por um papel fundamental no controle das populações de insetos, o que inclui o controle biológico de pragas em residências, jardins e plantações. No entanto, como podem ser encontrados em locais como armários, cantos de paredes, porões, garagens e quintais, podem entrar em contato com humanos e causar acidentes. No período de 2017 a 2021, o boletim epidemiológico do Ministério da Saúde (Secretaria de Vigilância em Saúde, 2022) informou que foram registrados mais de 150 mil casos de acidentes causados por aranhas, ocupando o terceiro lugar no ranking de ocorrências registradas com animais peçonhentos. O Instituto Butantan[1], por exemplo, recebe mais de cem aranhas por ano, capturadas por pessoas em suas residências e arredores, que buscam informações sobre o tipo, os riscos e o que fazer em caso de picadas.

Como o método tradicional de classificação taxonômica depende de profissionais especializados, ferramentas que possam fornecer informações sobre a espécie de aranha, o habitat, o potencial de dano e outras características são uma alternativa atraente. Diante disso, modelos de aprendizagem profunda de visão computacional têm se mostrado eficientes no processamento de dados complexos e não estruturados (Goodfellow *et al.*, 2016) como imagens, videos e textos. Embora essa tarefa de classificação com modelos profundos ainda represente um desafio devido à necessidade de uma grande quantidade de dados e alto custo computacional, as suas vantagens tem sido amplamente exploradas em diversos campos da medicina, agricultura, biodiversidade, entre outros.

Nesse sentido, essas ferramentas podem ser utilizadas tanto para evitar espécies peçonhentas, possivelmente reduzindo o número de acidentes e o extermínio de aranhas inofensivas como também serem utilizadas para propósitos relacionados à biodiversidade, à agricultura e a outras áreas de pesquisa acadêmica. Assim, propomos classificar 25 espécies de aranhas brasileiras a partir de suas imagens do repositório de fotos disponíveis publicamente no site INaturalist. Para tanto, aproveitamos dos avanços em aprendizagem profunda nos últimos anos e utilizamos duas arquiteturas que superaram o estado da arte e demonstraram proficiência em classificação de imagens: Redes Convolucionais

[1] https://butantan.gov.br/

e Transformers Visuais. Essas arquiteturas foram, então, comparadas em dois aspectos principais: capacidade de desempenho eficaz e possíveis disparidades entre elas.

Capítulo 2

Conceitos Fundamentais

Neste capítulo revisaremos conceitos fundamentais de aprendizado de máquina e visão computacional utilizados neste trabalho.

2.1 Aprendizado de Máquina

O aprendizado de máquina, conhecido em inglês como *Machine Learning* (ML), é uma das áreas da inteligência artificial (IA) que foca em desenvolver algoritmos e modelos estatísticos para "aprender" a resolver uma tarefa e generalizar esse conhecimento adquirido, sem serem explicitamente programados. Diante dessa visão geral, o conceito de aprendizado pode ser refinado ao dividir as técnicas de aprendizado em subclasses, popularmente encaixadas nas seguintes categorias: supervisionado, semi supervisionado, não supervisionado e por reforço.

Em particular, daremos atenção para o aprendizado supervisionado, que envolve um conjunto de dados rotulados, isto é, um conjunto de objetos de entrada e seus respectivos valores de saídas desejados para treinar um modelo preditivo. Idealmente, espera-se que o algoritmo produza um modelo generalizador, capaz de determinar corretamente valores de saída para instâncias não vistas anteriormente no conjunto de treinamento. Em outras palavras, o modelo aprende a distribuição de probabilidade fora de amostra (*out-of-sample*) (Abu-Mostafa *et al.*, 2012).

Ainda, podemos bipartir o aprendizado supervisionado em regressão e classificação: a regressão prevê valores contínuos, enquanto a classificação categoriza os dados em rótulos discretos. No caso de nossos dados, utilizamos modelos classificadores de espécies baseado em imagens, onde os dados estão estruturados em tuplas do tipo ⟨imagem, espécie⟩.

2.2 Arquitetura de Redes Neurais

Uma rede neural (*neural network*, NN) é um paradigma de programação inspirado na biologia que permite que um computador aprenda com dados observacionais (Nielsen,

2018). Para entender seu funcionamento de modo geral, começamos com a arquitetura mais simples, denominada *multilayer perceptron*.

2.2.1 Redes *Multilayer Perceptron*

A arquitetura de uma rede neural *Multilayer Perceptron* (MLP) consiste em uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída, onde cada camada tem um determinado número de nós (ou neurônios), e cada nó possui múltiplas entradas correspondente às saídas dos nós da camada anterior. Às entradas de um nó, associam-se pesos, os quais definem o grau de influência de uma entrada na saída do nó, ou seja, um peso amplifica ou atenua um sinal de entrada. A saída de uma camada é determinada pela soma ponderada das entradas, utilizando os pesos atribuídos a cada conexão e, a essa soma, adiciona-se um termo adicional conhecido como *bias*. Esse termo funciona de maneira similar à constante de uma equação linear, utilizado para deslocar o resultado. Enfim, essa soma passa por uma função de ativação, com propósito de quebrar a linearidade e permitir que a rede aprenda representações mais complexas. A saída gerada é alimentada para os nós na próxima camada da rede ou, no caso da última camada, torna-se parte da predição da rede.

Nessa rede, as conexões são comumente chamadas de *Feed Forward Networks* (FF/FFN), pois a informação se propaga apenas para frente durante a inferência e não possuem ciclos nem loops. Ainda, um outro termo que se aplica é *Fully Connected* (FC), onde cada nó de uma camada está conectado a todos os nós da camada seguinte. É comum utilizar esses termos de forma intercambiável.

Esse tipo de rede pode ser descrita matematicamente em forma matricial, ilustrada na Figura 2.1 onde há a camada de entrada, uma camada oculta e a camada de saída. Seja $X = (x_1, x_2, \cdots, x_n)$ uma instância de entrada, $W^{(i)}$ a matriz de pesos e $b^{(i)}$ o vetor de *bias* respectivamente, σ a função de ativação e $H^{(i)}$ a saída do nó i. Então, para calcular a saída da rede:

$$\begin{aligned} H^{(1)} &= \sigma(XW^{(1)} + b^{(1)}) \\ H^{(i)} &= \sigma(H^{(i-1)}W^{(i)} + b^{(i)}) \\ O &= \sigma(H^{(out)}W^{(out)} + b^{(out)}) \end{aligned}$$

Treinar uma rede neural envolve ajustar os pesos para minimizar a diferença entre as saídas previstas e as reais com base no conjunto de dados de treinamento. Assim, dada uma previsão da rede durante essa etapa, calcula-se o valor de uma função de perda L, que representa a discrepância entre as previsões da rede $\hat{y}$ e os valores reais y: $\text{Erro} = L(y, \hat{y})$. O método de retropropagação (*backpropagation*) é utilizado para propagar o erro de volta através da rede, calculando os gradientes que são empregados para atualizar os pesos por $W \rightarrow W - \eta(\partial \text{Erro}/\partial W)$, onde η é a taxa de aprendizado (*learning rate*, LR). Essa taxa determina o tamanho do passo em cada iteração enquanto se move em direção ao mínimo de uma função de perda. Esse processo é repetido até que o desempenho da rede seja satisfatório, treinando efetivamente o MLP para produzir previsões mais precisas.

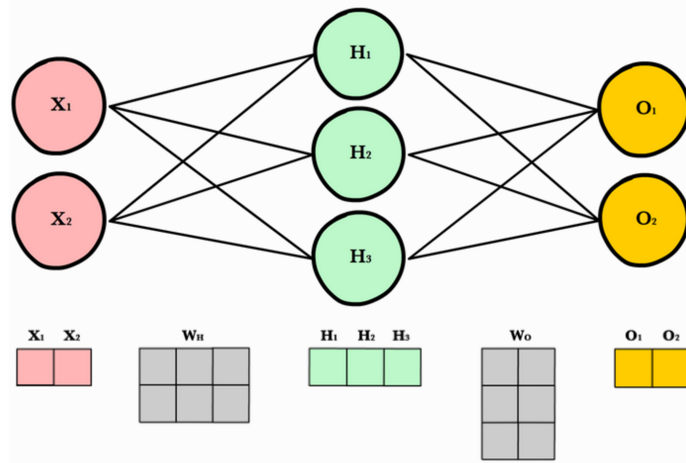


Figura 2.1: *Exemplo de uma rede MLP composta por uma camada de entrada X, uma camada oculta H e uma camada de saída O, descrita de modo equivalente em notação matricial (fonte).*

De acordo com o Teorema da Aproximação Universal (Hornik *et al.*, 1989), uma rede neural FF com apenas uma camada oculta e função de ativação é capaz de aproximar qualquer função contínua $f(x)$. Entretanto, não especifica quão complexa a rede deve ser, de modo que algumas funções podem ser altamente complexas, e aproximá-las adequadamente requer uma rede maior do que é prático ou viável, sem contar a quantidade de dados de treinamento necessários. Assim, outras técnicas de extração de informação foram desenvolvidas e combinadas com redes MLP para refinar a performance no campo de aprendizado de máquina, como redes convolucionais e Transformers (Seções 2.3 e 2.4).

2.2.2 Componentes Fundamentais

As redes neurais modernas compartilham alguns componentes fundamentais que são necessários para um entendimento completo. Iremos discutir brevemente quatro deles: funções de ativação, função de perda, otimizadores e métricas.

2.2.2.1 Funções de Ativação

O objetivo da função de ativação é introduzir a não linearidade na rede, permitindo que ela aprenda representações mais complexas. Algumas das funções de ativações mais utilizadas são Sigmoide, *Rectified Linear Unit* (ReLU) e *Gaussian Error Linear Unit* (GELU).

Dado um vetor de entrada x, matriz de pesos W e vetor de *bias* b, a função Sigmoide converte os valore $z = xW + b$ de saída de um nó para o intervalo $(0, 1)$ da seguinte maneira:

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

Uma extensão da função Sigmoide é a *Softmax*. A ideia é transformar os valores de saída em uma medida de probabilidade relativa, pois os mapeia ao intervalo $(0, 1)$ e a soma

das saídas é 1. É frequentemente usada na camada final do classificador para representar as probabilidades de várias classes. Para uma saída z_i, calculamos seu valor por:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

A função ReLU é utilizada em camadas intermediárias, é eficiente de computar e reduz significativamente os tempos de treinamento. Além disso, durante a etapa de *backpropagation*, o gradiente é constante para todas as entradas positivas e, portanto, não tem problema de desaparecimento de gradientes, o que ajuda na eficiência de treinamento. A função é dada por

$$\text{ReLU}(x) = \max(0, x)$$

Por fim, uma outra função muito utilizada é a GELU, que demonstrou ter melhor desempenho do que a ReLU em modelos mais profundos, pois tem uma forma mais suave e contínua, o que pode torná-la mais eficaz no aprendizado de padrões complexos nos dados. Ela é dada pela fórmula abaixo, onde $\Phi(x)$ é a função de distribuição acumulada da distribuição normal padrão (gaussiana).

$$\text{GELU}(x) = x * \Phi(x)$$

2.2.2.2 Função de Perda

Uma função de perda (ou custo) mede o desempenho das previsões do modelo em relação aos valores esperados (*ground-truth*). O objetivo durante o treinamento é minimizar essa perda, o que, em contrapartida, tende a melhorar a performance do modelo.

Uma função de perda comumente usada em tarefas de classificação é a *Cross-Entropy Loss*, que pode ser descrita como a soma negativa das probabilidades logarítmicas para a classe correta. Essa função é equivalente à utilizar LogSoftmax seguida de uma perda *Negative Log Likelihood Loss* (NLLLoss). LogSoftmax consiste simplesmente em uma operação de log sobre uma função Softmax pois é mais estável numericamente. NLLLoss seleciona a probabilidade de log correspondente à classe verdadeira e a nega. Essa abordagem penaliza diretamente o modelo com base na probabilidade de log da classe verdadeira.

Para cada dado no conjunto de treinamento $\langle x, y \rangle$, calcula-se a perda por $l_n = -x_{n,y_n}$ onde x_{n,y_n} é a probabilidade logarítmica Softmax para o n-ésimo dado com classe alvo y_n. A perda total $l(x, y) = L = \{l_1, \cdots, l_N\}^\top$ é o agregado dessas perdas individuais. Por fim, a perda é dada pela soma das perdas individuais. Se o modelo estiver confiante na classe correta, esse valor será próximo de zero, mas se o modelo estiver confiante na classe errada, esse valor torna-se um número muito negativo.

2.2.2.3 Métricas

Em classificação, as principais métricas de desempenho são acurácia, precisão, revocação e F1. Para calcular essas métricas, podemos criar uma tabela chamada Matriz

de Confusão, onde cada linha é a classe real e a coluna é a classe prevista. No caso de um classificador binário, temos uma matriz 2×2 onde VP é a quantidade de verdadeiros positivos, VN a de verdadeiros negativos, FP a de falso positivos e FN a de falso negativos, com métricas descritas pelas fórmulas abaixo.

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN}$$

$$\text{Precisão} = \frac{VP}{VP + FP}$$

$$\text{Revocação} = \frac{VP}{VP + FN}$$

$$\text{F1} = 2 \times \frac{\text{Precisão} \times \text{Revocação}}{\text{Precisão} + \text{Revocação}}$$

Além disso, o cálculo das métricas pode ser dividido entre Micro e Macro. Em Micro, agrega-se todos os resultados para então calcular a métrica. Em Macro, calcula-se a métrica independentemente por classe e então agrega-se em uma média, ou seja, todas as classes são tratadas igualmente. No caso de multi-classe (N), temos uma matriz $N \times N$ e as predições são analisadas por classe, onde as demais são consideradas negativas. Por causa dessa estrutura, os valores micro são iguais para todas as 4 métricas descritas.

2.2.2.4 Benchmarks

Benchmarks fornecem um conjunto de dados e critérios de avaliação, permitindo uma comparação justa de diferentes algoritmos e modelos. Assim, permitem o acompanhamento do progresso por meio da comparação de resultados de novas técnicas em relação às anteriores. Alguns benchmarks conhecidos em visão computacional são MNIST, COCO, CIFAR e ImageNet (Deng *et al.*, 2009), sendo este último um dos mais utilizados para classificação de imagens. Possui duas vertentes: ImageNet1K com mais de um milhão de imagens e um total de 1000 classes e ImageNet22K com 14 milhões de imagens e 21841 categorias. Benchmarks como esses são essenciais para o campo da classificação de imagens, pois fornecem uma base de comparação utilizada para determinar os modelos estado da arte.

2.3 Redes Neurais Convolucionais

As redes neurais convolucionais (*Convolutional Neural Networks*, CNNs) são caracterizadas pelo uso de camadas convolucionais e lideraram a revolução no campo de processamento e análise de imagens.

Em uma CNN, a camada convolucional é responsável pela detecção de características (*features*) como bordas, texturas e outros padrões na imagem. Isso é feito por meio da aplicação de vários kernels convolucionais à entrada de uma camada, onde cada um

realiza convolução sobre a entrada, gerando um mapa de características (*feature map*). Uma convolução é uma operação matemática que combina duas funções e produz uma terceira função que representa como a forma de uma é modificada pela outra. Para uma matriz de entrada I, kernel K e um termo *bias* b, podemos representar a convolução por $S(i,j) = \sum_m \sum_n I(m,n) \cdot K(i-m, j-n) + b$. A Figura 2.2 ilustra essa operação para a saída $S(0,0)$ e a Figura 2.3 exemplifica uma convolução de detecção de bordas com uma imagem.

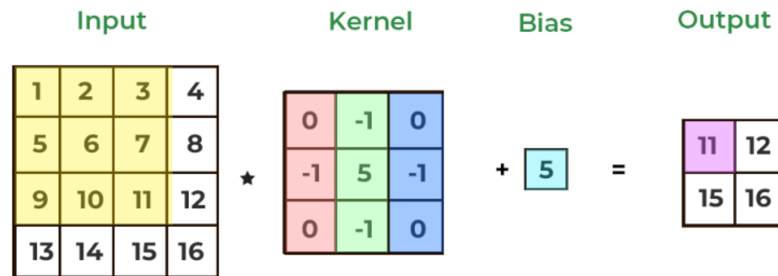


Figura 2.2: *Ilustração de uma convolução com kernel* 3×3 (*fonte*).

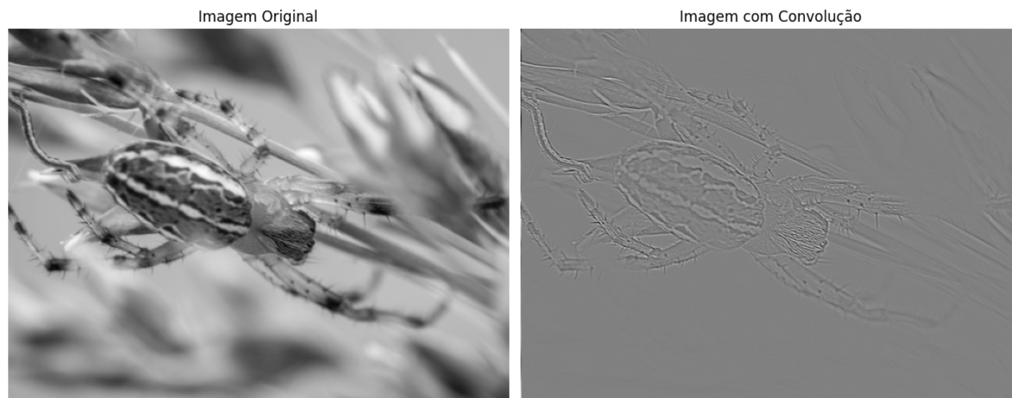


Figura 2.3: *Exemplo de uma convolução com filtro de detecção de bordas.*

Um filtro é composto de vários kernels e o seu tamanho é determinado não apenas pelo tamanho da convolução (3x3, 5x5, etc), mas também pelo número de canais na entrada que ele processa. Por exemplo, se a entrada tiver 3 canais (RGB), um filtro teria dimensão 3x3x3. Isso garante que o filtro capture padrões e recursos de todos os canais da entrada, integrando efetivamente as informações desses canais durante a operação de convolução. Ainda, mais duas variáveis afetam o *feature map*: *stride* e *padding*. O *stride* determina quantos pixels o filtro de uma convolução se move pela imagem de entrada, de modo que um *stride* maior resulta em uma dimensão de saída menor (e vice-versa). O *padding* adiciona pixels extras nas margens da imagem de entrada para que os filtros não percam informações nas regiões de borda. Então, para uma imagem de tamanho W, convolução quadrada K, *padding* P e *stride* S, a saída da convolução irá gerar um mapa de tamanho $\big((W + 2P - K)/S\big) + 1$.

Como uma convolução produz um conjunto de ativações lineares, a saída passa por uma função de ativação não-linear, de modo similar às redes FC. Em seguida, é comum

adicionar uma etapa de processamento denominada de *pooling*, de modo que a saída da rede em um determinado local é transformada em uma estatística resumida das saídas próximas de modo que a representação fique aproximadamente invariante a pequenas translações da entrada (Goodfellow *et al.*, 2016). Em geral, os tipos mais comuns de *pooling* utilizados são *average pooling* e *max pooling*, representando média e valor máximo de uma vizinhança retangular respectivamente. Após as camadas iniciais de convolução e pooling, a rede pode possuir mais camadas convolucionais ocultas que continuam o processo de extração de características. Por fim, a matriz multidimensional de saída das últimas camadas convolucionais é convertida em um vetor unidimensional que é alimentado para camadas FC para sintetizar as *features* em resultados finais de decisão da rede. Essencialmente, a ideia é que a rede aprenda os pesos dos kernels e os pesos das redes FC. Esses conceitos são ilustrados na próxima seção (2.3.1).

2.3.1 VGGNet

Uma das arquiteturas que contribuiu significativamente para o campo de *deep learning* foi a VGGNet desenvolvida pelo Visual Graphics Group (VGG) da Universidade de Oxford (Simonyan e Zisserman, 2015) e é reconhecida por sua arquitetura profunda com 16 a 19 camadas no total e milhões de parâmetros treináveis. Essa rede pode ser utilizada para demonstrar efetivamente o funcionamento de uma rede CNN. Na Figura 2.4, temos uma ilustração de uma VGG16 onde a imagem de entrada $224 \times 224 \times 3$ passa por 64 filtros de dimensão $3 \times 3 \times 3$ com *stride* e *padding* igual a 1, seguidos de uma função ReLU, gerando um mapa de ativação inicial de dimensão $224 \times 224 \times 64$. Então, esse mapa passa por uma camada de *max pooling* 2×2, reduzindo a dimensão e a saída é direcionada para a camada convolucional seguinte da mesma maneira. No final da rede, temos as redes FC com um classificador Softmax de 1000 classes.

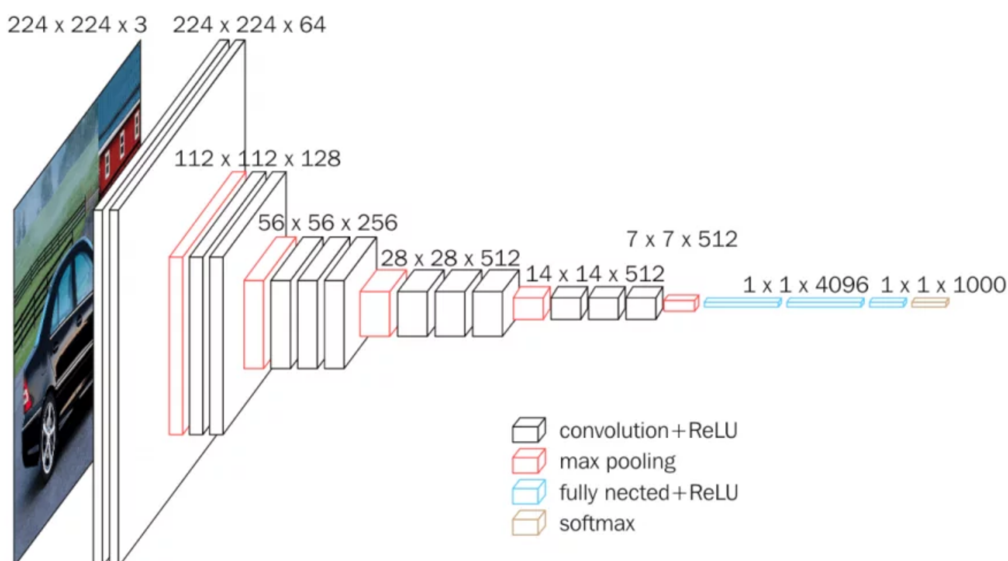


Figura 2.4: *Arquitetura de uma rede VGG16 (fonte).*

Para calcular o número de parâmetros a serem aprendidos pela rede, basta analisar as dimensões dos filtros e das redes FC. Por exemplo, para a terceira camada da rede (camada

convolucional), teremos 64 canais de entrada e 128 canais de saída para kernels de dimensão 3×3, além de um termo de *bias* por filtro. Então, $Params = (3 * 3 * 64) * 128 + 128 = 73856$, assim como ilustrado na Figura 2.5.

VGG16 - Structural Details													
#	**Input Image**			**output**			**Layer**	**Stride**	**Kernel**		**in**	**out**	**Param**
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
							Total						**138,423,208**

Figura 2.5: *Cálculo da quantidade de parâmetros de uma rede VGG16 (fonte).*

2.3.2 ResNet

Uma das limitações da redes VGG está ligada à sua arquitetura. Quando as redes profundas conseguem começar a convergir há um problema de degradação: com o aumento da profundidade da rede, a precisão se estabiliza e depois diminui rapidamente e leva a um erro de treinamento mais alto (He *et al.*, 2015). Nesse artigo, os autores introduzem a rede ResNet (*Residual Network*) e o conceito de blocos residuais (*residual blocks*) que se baseia em, ao invés de esperar que as camadas empilhadas se ajustem diretamente a um mapeamento ideal desejado $\mathcal{H}(\mathbf{x})$, deixa-se que essas camadas se ajustem a um mapeamento residual $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$ (Figura 2.6). Essencialmente, depois de aprender o residual $\mathcal{F}(\mathbf{x})$, a rede o adiciona $\mathbf{x}$ à entrada original por meio de *skip connections*, e isso seria uma outra maneira da rede de representar o mapeamento original $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$.

A hipótese inicial era que seria mais fácil otimizar o mapeamento residual do que otimizar o mapeamento original. Os resultados a confirmam: modelos ResNet superam o VGG na maioria das tarefas de visão computacional, são mais eficientes em quantidade de parâmetros e têm menos problemas com desaparecimento de gradientes (*vanishing gradients*).

De maneira mais detalhada, a arquitetura de uma ResNet também utiliza *bottleneck blocks* e *global average pooling* (GAP). Um bloco *bottleneck* é utilizado para melhorar performance em redes profundas e é constituído de três camadas sequenciais de convolução do tipo 1×1, 3×3 e 1×1, onde a primeira camada 1×1 reduz as dimensões de entrada, a camada 3×3 é utilizada para processamento principal e a última camada 1x1 aumenta as dimensões de saída. Quanto à técnica GAP, é similar à de *average pooling*, de modo que

reduz cada *feature map* no final da rede em um único valor escalar para ser alimentada na rede FC classificadora.

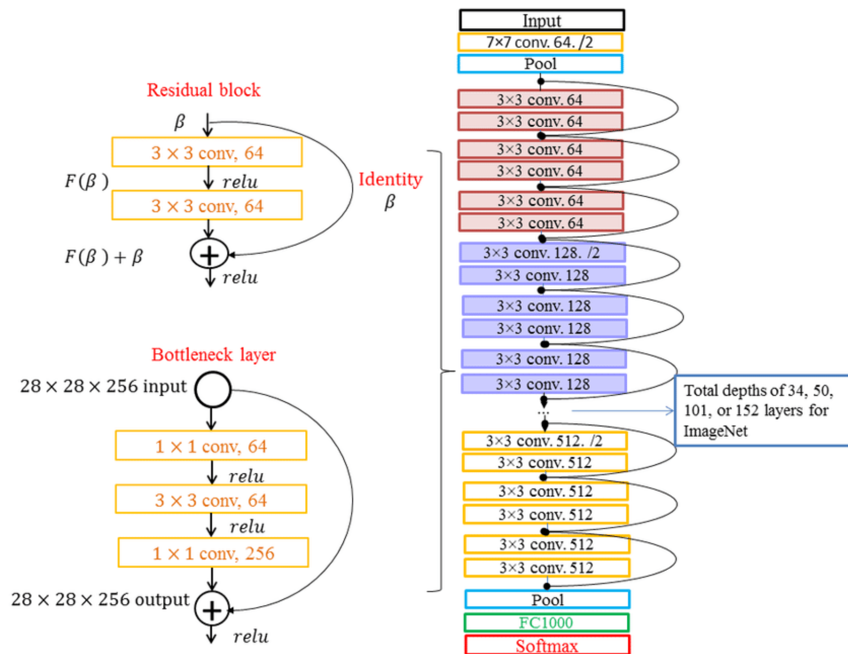


Figura 2.6: *Arquitetura geral de uma rede ResNet (Partovi et al., 2019).*

As redes ResNet também são classificadas dependendo do tamanho de sua arquitetura e quantidade de blocos, como ResNet34, ResNet50, etc. Em uma rede ResNet50 por exemplo, há um total de 50 camadas e as camadas convolucionais são dividas em blocos residuais de tamanhos (3, 4, 6, 3). Isso significa que há primeiramente 3 blocos residuais com 64 filtros, 4 blocos com 128 filtros e assim por diante, assim como ilustrado na imagem à direita da Figura 2.6.

2.3.3 ResNeXt

Uma das arquiteturas que apresentou evoluções significativas para CNNs foi a ResNeXt, introduzida por Xie *et al.* (2017). No artigo, os autores implementam a ideia de *split-transform-merge* de redes Inception (Szegedy *et al.*, 2014) e introduzem uma nova dimensão C referida por cardinalidade. Nessa ideia, a entrada é separada (*split*) em C caminhos, passa por transformações (*transform*) e, no fim, agrega-se os resultados (*merge*). Formalmente, é representada pela equação 2.1 abaixo:

$$\mathcal{F}(\mathbf{x}) = \sum_{i=1}^{C} \mathcal{T}_i(\mathbf{x}) \tag{2.1}$$

onde $\mathcal{T}_i(\mathbf{x})$ poderia ser qualquer função arbitrária, porém os autores definem as transformações $\mathcal{T}_i$ em arquitetura de *bottleneck* e estruturalmente idênticas. Essa arquitetura possui uma variável d que controla o número de canais nas transformações, utilizada para equilibrar a troca de eficiência computacional por poder de representação. Uma

comparação entre blocos ResNet e ResNeXt ($C = 32$, $d = 4$) está apresentada na Figura 2.7. Após a agregação, a saída $\mathbf{y}$ é dada por:

$$\mathbf{y} = \mathbf{x} + \sum_{i=1}^{C} \mathcal{T}_i(\mathbf{x}) \tag{2.2}$$

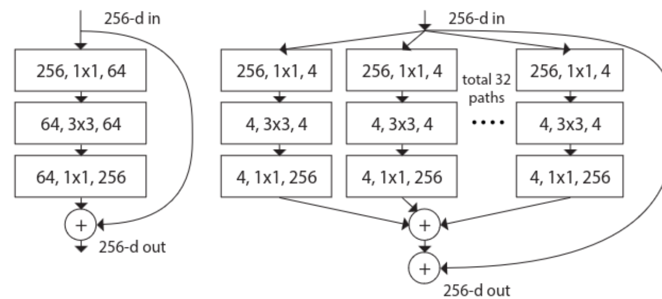


Figura 2.7: *Comparação de um bloco ResNet (esquerda) com um bloco ResNeXt (direita). (Xie et al., 2017).*

Os experimentos demonstram que o aumento da cardinalidade é uma forma mais eficaz de aumentar a acurácia do que a profundidade ou a largura e que as redes ResNext superam o desempenho das redes ResNet, mesmo ao manter a complexidade computacional e o tamanho do modelo semelhantes.

2.3.4 ConvNeXt

O ConvNeXt representa uma evolução no projeto de redes convolucionais, inspirado nas ideias introduzidas em *Vision Transformers* (Seção 2.4), mas mantém uma arquitetura puramente convolucional. Zhuang Liu *et al.* (2022) utilizam a rede ResNet-50 como ponto de partida e dividem as mudanças estruturais em 5 partes: Macro Design, ResNeXt, Bottlenecks, Tamanho de Kernel e Micro Design.

Como parte da modificação em macro escala, a rede é dividida em quatro estágios, assim como na rede ResNet-50, porém a distribuição de blocos é ajustada de $(3, 4, 6, 3)$ para $(3, 3, 9, 3)$ (Figura 2.8 (A)). Além disso, mudam a abordagem inicial (*stem*) de lidar com a imagem de entrada, substituindo a convolução inicial 7×7 por uma camada de uma convolução sem sobreposição 4×4 com stride 4.

Utilizando a ideia de usar mais grupos e expandir a largura introduzidas nas redes ResNeXt, utilizam convolução em profundidade (*depthwise convolution*), onde cada canal de entrada tem um kernel diferente, seguida de convolução 1×1. Os autores mencionam que isso leva a uma separação da mistura espacial e de canal, ou seja, cada operação mistura informações na dimensão espacial ou na dimensão do canal, mas não em ambas.

A terceira e quarta mudanças estruturais são os *bottleneck* invertidos, para eficiência e extração aprimorada de recursos e o aumento no tamanho dos kernels, saturando os ganhos em tamanho 7×7. Por fim, as mudanças micro foram substituir função ReLU por GELU, utilizar menos funções de ativação, menos camadas de normalização e substituir

normalização de batch por normalização de camada. A Figura 2.8 (B) representa essas modificações.

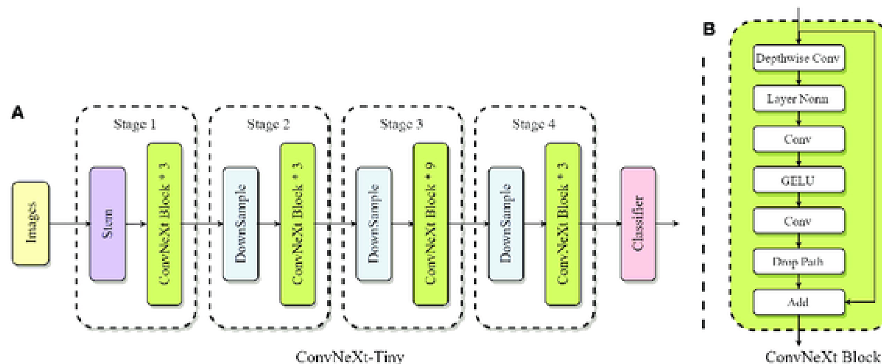


Figura 2.8: *(A) Arquitetura geral de uma rede ConvNeXt-Tiny; (B) Bloco ConvNeXt (Jiang et al., 2023).*

Os resultados indicam que ConvNeXt consegue ter um desempenho geral no mesmo nível do ViT em classificação. Para os modelos maiores, ConvNeXts ainda têm um desempenho igual ou melhor do que os *Swin Transformers* (Seção 2.4.2) de tamanho semelhante.

2.4 Transformers Visuais

2.4.1 Transformers

Transformers foram primeiramente apresentados no artigo "Attention Is All You Need" (Vaswani *et al.*, 2023) para tarefas de processamento de linguagem natural (*natural language processing*, NLP). A inovação introduzida foi utilizar exclusivamente um mecanismo de auto-atenção (*self-attention*) sem empregar camadas recorrentes ou convolucionais. A ideia principal desse mecanismo é poder ponderar a importância de diferentes partes dos dados em relação umas às outras, de modo que se concentre em partes específicas da entrada e, ao mesmo tempo, considere todo o contexto.

As arquiteturas de Transformers podem ser divididas em três tipos: *encoder-decoder*, *encoder-only* e *decoder-only*, adequadas para diferentes tipos de tarefas. A arquitetura *encoder-decoder* é usada em tarefas em que tanto a entrada quanto a saída são sequências, como tradução. *Decoder-only* é utilizada para gerar uma sequência de saída com base na entrada e no contexto gerado, por exemplo GPT-3 (Brown *et al.*, 2020). Por fim, *encoder-only* emprega apenas o codificador do Transformer, ideal para tarefas de extração de informação, compreensão e classificação, como é o caso deste trabalho.

As unidades de entrada são denominadas de tokens, os quais podem ser caracteres, palavras, pixels, entre outros. Esses tokens são transformados em vetores reais de tamanho d_{model} que compõem o *input embedding*. Em seguida, adiciona-se a codificação posicional, com objetivo de fornecer informações sobre a posição relativa ou absoluta dos tokens na sequência, utilizados como entrada pelo codificador. De forma geral, o *encoder* é composto por uma sequência de N camadas idênticas onde cada uma possui um mecanismo de

atenção *multi-head self-attention* (MSA), seguido por uma rede FF, com conexões residuais e *layer normalization* nas saídas intermediária e final (fig. 2.9).

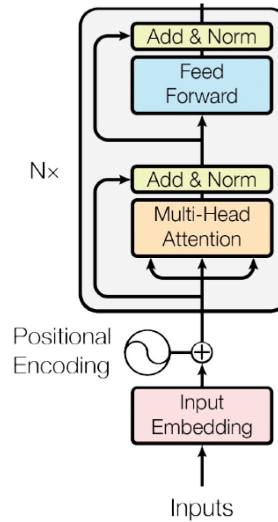


Figura 2.9: *Encoder de um Transformer (Vaswani et al., 2023).*

O valor de *self-attention* é calculado utilizando três matrizes principais: *Queries* (Q), *Keys* (K) e *Values* (V). No contexto de treinamento do *encoder*, essas matrizes são derivadas da matriz de entrada X, onde cada linha representa um vetor do *input embedding*, e das matrizes de pesos W de modo que $Q = XW^Q$, $K = XW^K$, $V = XW^V$. A ideia é que cada vetor de palavras (*query*) é comparado aos outros vetores de palavras (*keys*). As pontuações obtidas são normalizadas pela função Softmax, transformando-as em probabilidades (score). Então, calcula-se a atenção ao multiplicar a matriz V pelos scores (eq. 2.3), a fim de manter intactos os inputs com alta relevância (score alto) e ignorar os irrelevantes (score baixo).

$$Attention(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

Diante disso, é possível calcular efetivamente o *multi-head self-attention*. Seja h o número de *heads*, d_k a dimensão das *queries* e *keys* e d_v a dimensão de *values*. Esse mecanismo de atenção introduz três matrizes paramétricas para cada uma das cabeças, onde $d_k = d_v = d_{model}/h$, definido pelos autores.

$$\begin{aligned} Q_i &= XW_i^Q \qquad (W_i^Q \in \mathbb{R}^{d_{model} \times d_k}) \\ K_i &= XW_i^K \qquad (W_i^K \in \mathbb{R}^{d_{model} \times d_k}) \\ V_i &= XW_i^V \qquad (W_i^V \in \mathbb{R}^{d_{model} \times d_v}) \end{aligned}$$

Assim, as h cabeças são concatenadas em uma matriz (restaurando a dimensão original) que é multiplicada por uma outra matriz W^O para calcular a saída final, processo descrito

pelas equações abaixo. A estrutura dessa arquitetura está ilustrada na Figura 2.10.

$$\begin{aligned} \text{head}_i &= \text{Attention}(Q_i, K_i, V_i) \\ \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \cdots, \text{head}_h) \cdot W^O \qquad (W^O \in \mathbb{R}^{hd_v \times d_{model}}) \end{aligned}$$

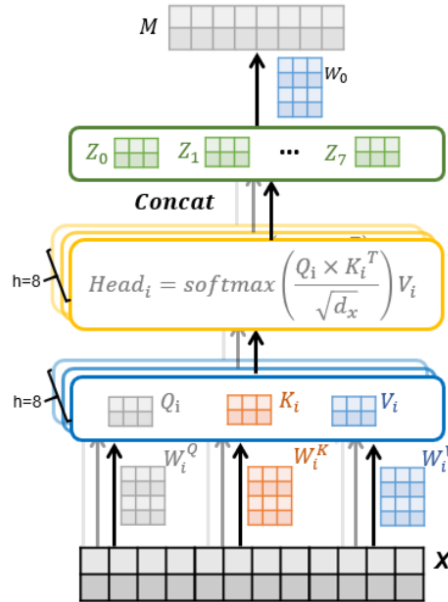


Figura 2.10: *Estrutura completa de multi-head attention (Zheng et al., 2019).*

A *multi-head self-attention* permite que o modelo considere conjuntamente a informações de diferentes subespaços de representação em posições diferentes

2.4.2 Vision Transformers

Inspirados pelos sucessos de Transformers em NLP, Dosovitskiy *et al.* (2021) aplicaram um Transformer padrão com mínimas modificações para imagens, designado *Vision Transformer* (ViT), ilustrado na Figura 2.11.

Para conseguir comportar imagens 2D, a imagem de entrada $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ é transformada em uma sequência de blocos (*patches*) achatados em vetores 1D do tipo $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 * C)}$, onde cada bloco é de dimensão $P \times P$ com total de $N = HW/P^2$ blocos. Como o Transformer usa um vetor latente de tamanho D constante em todas as suas camadas, os blocos de entrada são mapeados para dimensão D por uma projeção linear treinável de modo análogo ao *embedding* de um Transformer padrão.

Feito isso, adiciona-se um token especial [CLS] aprendível ao início da sequência. Para tarefas de classificação, a saída precisa ser reduzida a um único vetor e esse token é uma maneira simples e eficiente de obter uma representação que agrega informações de toda a sequência de entrada pois é utilizado especificamente para otimizar sua representação. Então, a codificação posicional é adicionada, o vetor é direcionado como entrada do *encoder*, e a saída do encoder na posição 0 (token [CLS]) é alimentado na rede MLP para gerar a classificação.

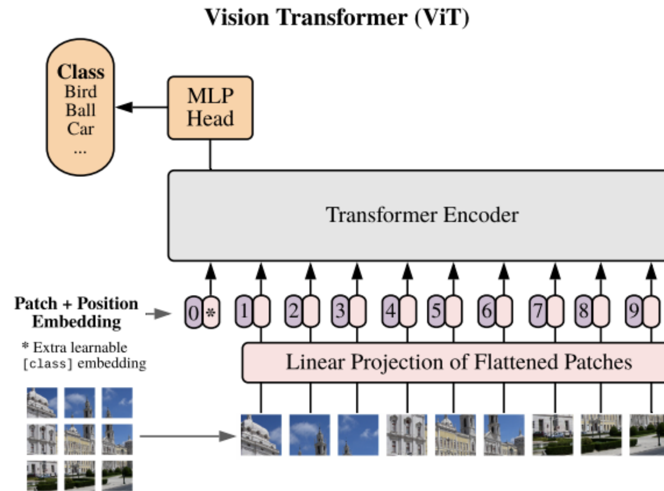


Figura 2.11: *Modelo de um Vision Transformer (Dosovitskiy et al., 2021).*

Os ViTs superaram CNNs tradicionais em conjuntos de dados extensos, mas apresentam mais *overfitting* do que ResNets em bancos de dados menores.

2.4.3 Swin Transformers

Com propósito de expandir a aplicabilidade do Transformer, Ze Liu *et al.*, 2021 desenvolveram uma nova arquitetura com uma visão que ela pudesse servir como um backbone de uso geral para a visão computacional de modo semelhante às redes CNNs. No artigo, argumentam que, diferentemente dos tokens de palavras, os elementos visuais podem variar substancialmente entre características detalhadas e características gerais simultaneamente. Outra diferença é a resolução muito maior dos pixels nas imagens em comparação com as palavras em trechos de texto, isto é, densidade de informação entre textos e imagens é diferente. Para solucionar esses problemas, propuseram o Swin Transformer (*Shifted Windows*) que constrói *feature maps* hierárquicos com complexidade linear no tamanho da imagem.

As imagens de entrada $\langle H \times W \rangle$ são separadas em *patches*, convertidos em $\frac{H}{4} \times \frac{W}{4}$ tokens, projetados linearmente em uma dimensão arbitrária C e alimentados ao primeiro bloco Swin. Para construir uma representação hierárquica, utilizam *patches* pequenos nas primeiras camadas e os mesclam (*patch merging*) em outros maiores nas camadas mais profundas da rede. A primeira camada de mesclagem de blocos concatena as *features* de grupos 2×2 de *patches* vizinhos de tamanho 4×4, resultando em uma área concatenada de 8×8 e um vetor de dimensão $4C$ para cada grupo, seguido de uma redução de dimensionalidade para $2C$ ($\frac{H}{8} \times \frac{W}{8} \times 2C$). Esse processo é repetido para os outros blocos do Swin, ilustrado na Figura 2.12 (a) e (d), e conjuntamente produzem uma representação hierárquica, com um *feature map* com as mesmas resoluções das redes CNN típicas como ResNet.

Um bloco Swin possui dois mecanismos de atenção: W-MSA e SW-MSA e denotam *window based multi-head self-attention* usando particionamento regular e *shifted window*, respectivamente, seguido de uma rede MLP. Ambos utilizam LayerNorm (LN) e conexões

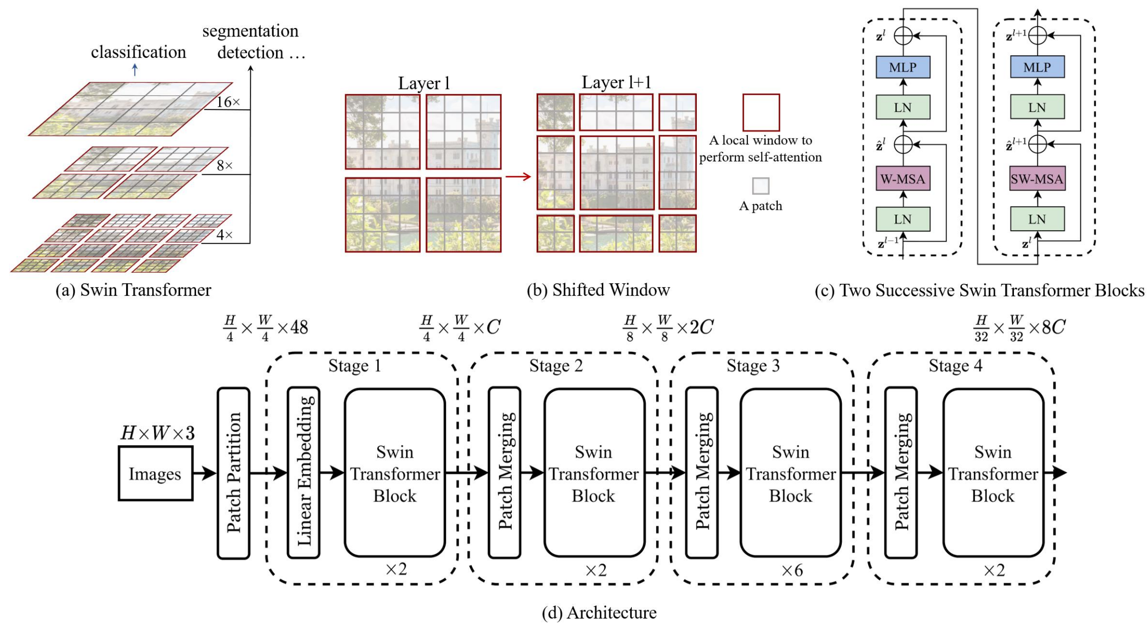


Figura 2.12: *(a) Mesclagem de patches (em cinza); (b) auto-atenção dentro de cada janela local (em vermelho); (c) Bloco Swin com mecanismos de atenção; (d) Arquitetura de um Swin-Tiny. (Ze Liu et al., 2021)*

residuais (Figura 2.12 (c)). O mecanismo W-MSA consiste em delimitar janelas que não se sobrepõem a um conjunto de *patches* e calcular a atenção local independentemente de modo a determinar o grau de foco que cada patch deve dar a todos os outros patches na mesma janela. Essa abordagem tem como objetivo capturar as relações espaciais em regiões locais da imagem e é computacionalmente eficiente. O mecanismo SW-MSA emprega uma modificação no janelamento, onde janelas contendo $M \times M$ *patches* são deslocadas por $(\lfloor \frac{M}{2} \rfloor, \lfloor \frac{M}{2} \rfloor)$ das posições regulares. O problema dessa abordagem é que ela resulta em mais janelas, de $\lceil \frac{h}{M} \rceil \times \lceil \frac{w}{M} \rceil$ para $\left(\lceil \frac{h}{M} \rceil + 1 \right) \times \left(\lceil \frac{w}{M} \rceil + 1 \right)$ e algumas das janelas serão menores do que $M \times M$, ilustrado na Figura 2.12 (b).

Para contornar esse problema, apresentaram uma abordagem de computação em batch eficiente onde a imagem é deslocada ciclicamente (*cyclic shift*), conforme apresentado na Figura 2.13. Quando um *patch* de uma janela deslocada se move para além do limite da imagem, a imagem é rotacionada pelo lado oposto da mesma janela. Então, computa-se MSA com uma máscara, dado que alguns *patches* são adjacentes nas janelas rotacionadas, mas que não são nos *patches* originais. No fim, os patches são revertidos para o formato inicial.

Os resultados demonstram que Swin supera ViT em classificação de imagens, detecção de objetos e segmentação semântica.

2.4.4 MaxViT

Apesar de ter mais flexibilidade e generalização do que a atenção usada no ViT, observou-se que a atenção baseada em janelas do Swin tem capacidade limitada devido à perda de não-localidade. Em Tu *et al.* (2022), os autores apresentam um novo tipo

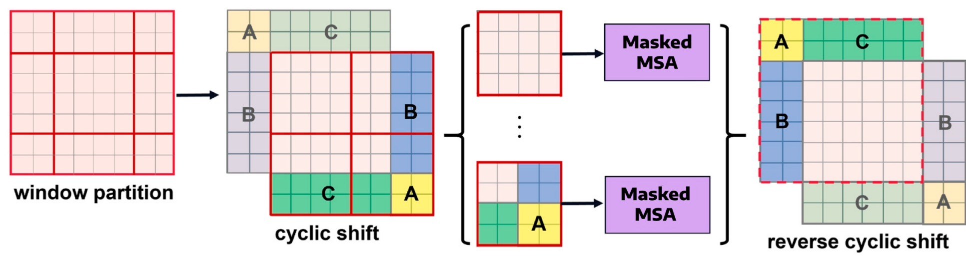


Figura 2.13: *Processo de computação de SW-MSA com deslocamento cíclico (Ze Liu et al., 2021).*

de módulo Transformer, denominado *multi-axis self-attention* (Max-SA), que é capaz de realizar interações espaciais locais e globais em um único bloco com complexidade linear, o que permite maior flexibilidade e eficiência. Baseado nesse mecanismo, desenvolvem um backbone chamado *Multi-axis Vision Transformer* (MaxViT), empilhando hierarquicamente blocos repetidos compostos de Max-SA e convoluções.

A arquitetura em si é muito parecida com ConvNets: entrada, *stem*, estágios S1 a S4 e uma cabeça FC (Figura 2.14). A diferença está nos blocos, utilizando um modelo híbrido de convolução, atenção local e atenção global. Um bloco MaxViT começa com um bloco de MBConv, onde utiliza *inverted bottleneck* com *depthwise convolution* (semelhante a ConvNeXt). Argumentam que o uso dessas camadas antes da atenção oferecem vantagem de que as convoluções em profundidade podem ser consideradas como codificação de posição condicional, tornando o modelo livre de camadas de codificação posicional explícitas (como no ViT por exemplo).

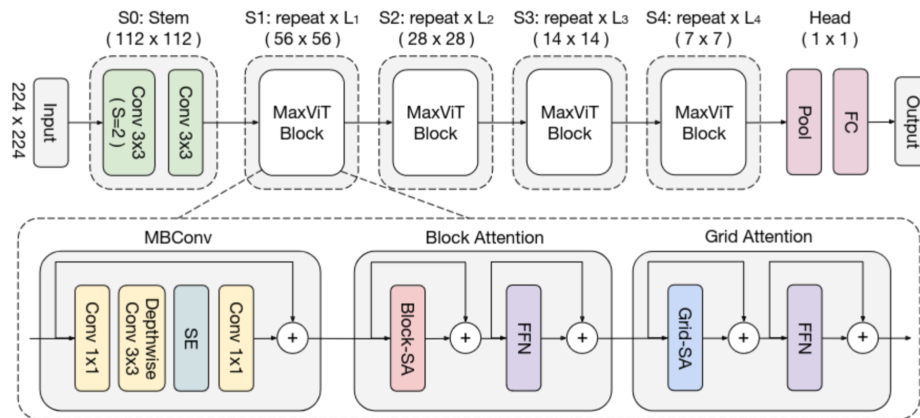


Figura 2.14: *Arquitetura de MaxViT (Tu et al., 2022).*

Seja $X \in \mathbb{R}^{H \times W \times C}$ o mapa de características de entrada. O *feature map* é separado em um tensor de forma $\langle \frac{H}{P} \times \frac{W}{P}, P \times P, C \rangle$, representando o particionamento em janelas não sobrepostas, cada uma de tamanho $P \times P$. Então, o mecanismo de atenção aplicado nessas dimensões locais, isto é, *block attention*.

Apresentam também uma maneira simples, mas eficaz, de obter atenção global esparsa, que é chamada de *grid attention*. O tensor agora é particionado na forma $\langle G \times G, \frac{H}{G} \times \frac{W}{G}, C \rangle$ usando uma grade uniforme $G \times G$ fixa, resultando em janelas com tamanho adaptável

$\frac{H}{G} \times \frac{W}{G}$, de modo que a atenção é aplicada diretamente nessa grade.

De acordo com os resultados indicados, o MaxViT melhora significativamente o desempenho do estado da arte para uma ampla gama de tarefas visuais, incluindo classificação, detecção e segmentação de objetos e geração de imagens e indicam que o módulo Max-SA proposto pode ser um substituto do módulo de atenção Swin com exatamente o mesmo número de parâmetros e FLOPs.

Capítulo 3

Desenvolvimento

Neste capítulo, explicaremos as etapas de desenvolvimento, metodologias e experimentos realizados para alcançar os objetivos propostos.

3.1 Coleta de Dados

Para começar a análise de classificação de espécies, criamos uma coleção de imagens do site INaturalist[1]. Esse site é uma plataforma online sem fins lucrativos com propósito de registrar biodiversidade por meio da prática de observar organismos selvagens e compartilhar informações sobre eles. Funciona como uma rede social em que os usuários podem compartilhar suas observações contribuindo para o banco de dados, agora com mais de 7 milhões de usuários e 180 milhões de observações (https://www.inaturalist.org/stats).

Para este trabalho, coletamos imagens de aranhas brasileiras (ordem *Araneae*) que tinham mais de 100 imagens e que estavam classificadas como adequadas para pesquisa (*research grade*), critério específico de qualidade e verificação para as observações, onde é necessário um nível significativo de concordância entre os membros da comunidade com relação à identificação da espécie. A primeira condição permite que tenhamos uma quantidade mínima de dados para treinar as redes e a segunda garante a consistência de nossos rótulos, dado que rotulagem inconsistente ou incorreta pode levar um modelo a fazer previsões imprecisas. Finalizada essa etapa, tínhamos 65 espécies com 30609 imagens no total.

Naturalmente, há um desequilíbrio significativo de classes, influenciado por uma variedade de fatores, incluindo condições geográficas e ambientais, atividade humana, visibilidade, entre outros. Por esse motivo, decidimos utilizar um subconjunto do dados, que consiste em 25 espécies e 24570 imagens, o que representa cerca de 80% do total de dados coletados, ilustrado nas Figuras 3.1 e 3.2.

[1] https://www.inaturalist.org/

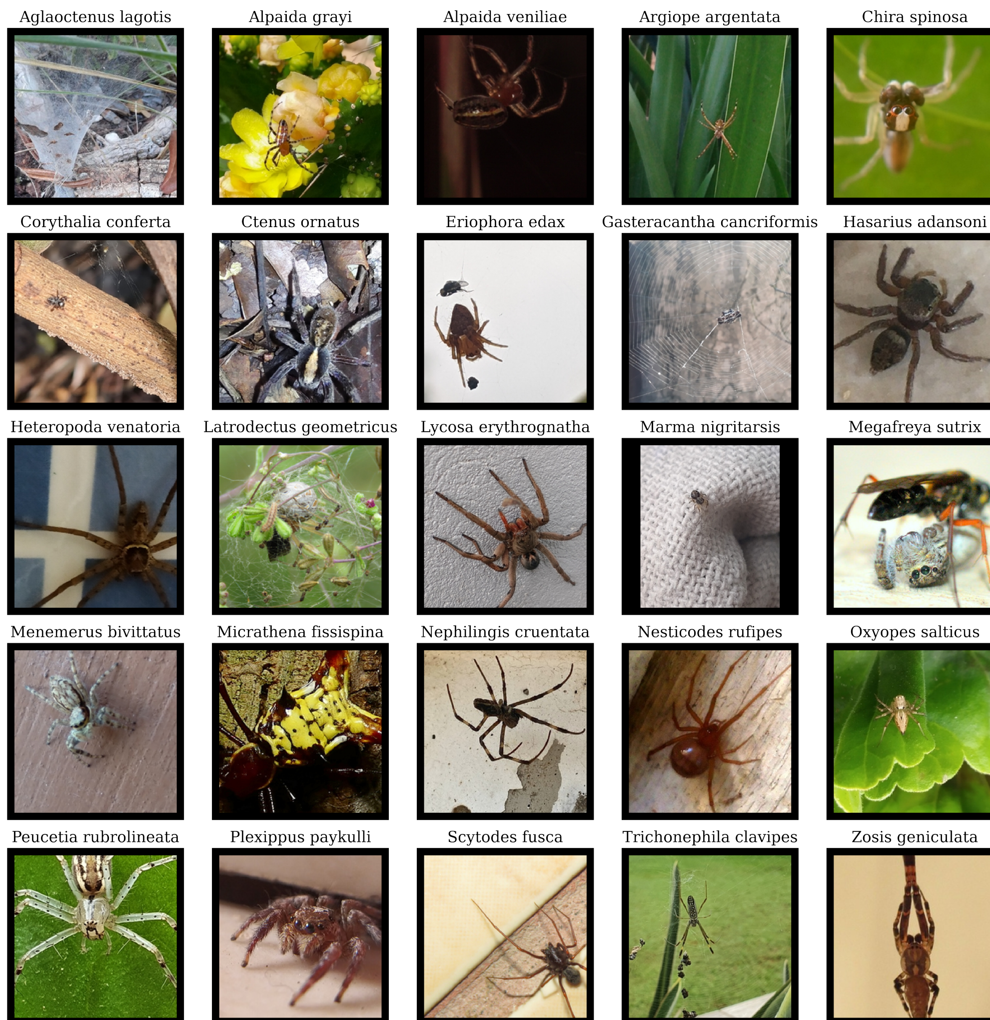


Figura 3.1: *Exemplo de imagens por espécie.*

3.2 Treinamento das Redes Neurais

3.2.1 Separação do Conjunto de Dados

No processo de separação do dataset, primeiro nos concentramos em reservar um conjunto de teste. Isto é uma etapa crucial para garantir que tenhamos um subconjunto independente e imparcial que possa ser utilizado para avaliar a capacidade de generalização do modelo após o treinamento. Para tanto, o conjunto de dados é embaralhado aleatoriamente para evitar qualquer possível viés de ordenação dos dados e 10% é separado para o subconjunto de teste, mantendo a mesma proporção de cada classe que o conjunto original. Esse processo é controlado por uma semente que fixamos de antemão.

Para treinar os modelos, utilizamos a técnica de validação cruzada estratificada *k-fold*, em que o conjunto de dados é divido em *k* partes (ou *folds*), mantendo-se a proporção original de cada classe em cada parte. O processo de treinamento é repetido *k* vezes. A cada rodada, uma das partes é usada para validação enquanto as demais são usadas no treinamento propriamente dito. A métrica de validação cruzada é, então, a média das

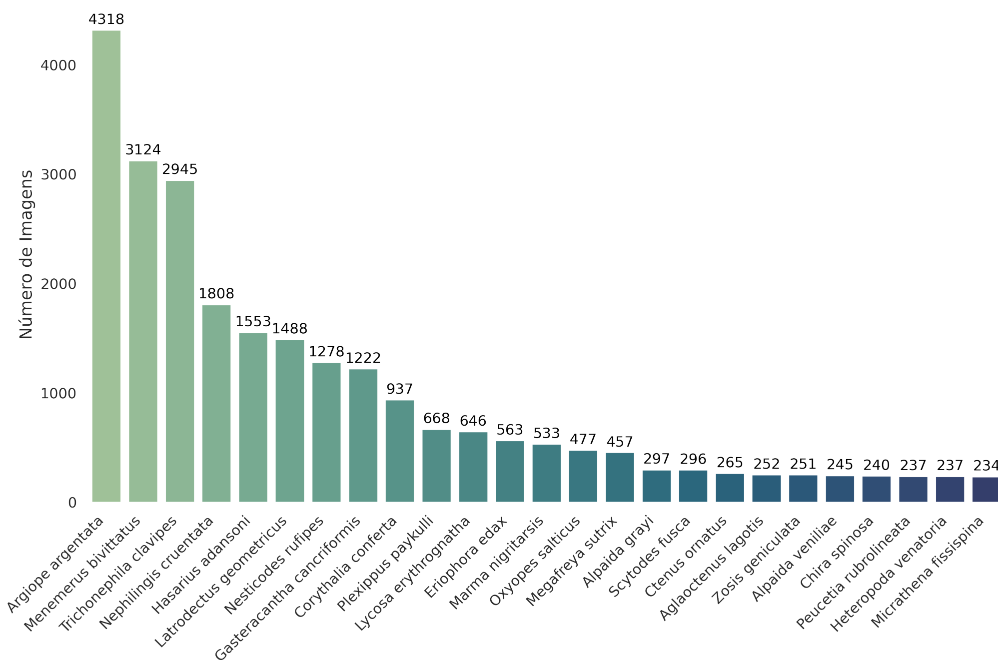


Figura 3.2: *Distribuição de imagens de aranhas do subconjunto selecionado.*

métricas calculadas sobre o conjunto de validação em cada um dos $k folds$. Esse método fornece uma estimativa mais confiável estatisticamente do desempenho do que uma única divisão de conjunto em treinamento/validação. Em particular, definimos $k = 5$ e, assim como no conjunto de teste, utilizamos uma semente para controlar a divisão. Além disso, utilizamos um amostrador somente no conjunto de treinamento, onde pondera inversamente cada amostra com base em sua frequência de classe, a fim de melhorar a performance da rede em espécies com menos imagens.

3.2.2 PyTorch

Uma das ferramentas mais conhecidas para tarefas de aprendizagem profunda é o PyTorch[2]: um *framework open-source* de aprendizado de máquina em Python baseado na biblioteca Torch usado para aplicações como processamento de linguagem natural e visão computacional. No começo deste trabalho, o utilizamos para treinar as redes, o que implica mexer diretamente com todas as etapas do processo de treinamento. Isso significa seguir o passo a passo de

- Lidar com dispositivos manualmente (cpus, gpus)
- Controlar as ativações de gradientes durante as etapas de treinamento e validação
- Controlar as épocas e processamento de *batches*
- Processar o *forward pass* para obter as predições e calcular a função de perda
- Utilizar explicitamente funções de *backpropagation* e otimizador

[2] https://pytorch.org/

- Fazer os cálculos das métricas baseado nas predições

Embora esse processo permita customização de cada etapa, gerenciar detalhes de “baixo nível” (controle fino) aumenta a complexidade do código, requer mais tempo de desenvolvimento, necessidade de mais manutenção, menos legibilidade e, claramente, não é o foco da pesquisa. Entretanto, é importante mencionar que proporcionou entendimento sólido do funcionamento interno de treinamento dos modelos.

3.2.3 PyTorch Lightning

Dadas as restrições mencionadas anteriormente, passamos a utilizar a biblioteca *PyTorch Lightning*[3] que fornece uma interface de mais alto nível para o PyTorch. Ela automatiza o loop de treinamento ao encapsular mecanismos como *fit, validate, test, predict* e conversão automática de dispositivos, reduzindo consideravelmente boilerplate, trivializa a utilização de ambientes distribuídos para aumento de performance e possui uma interface intuitiva. A parte interessante é que ainda é possível ter controle e flexibilidade em qualquer ponto do processo caso necessário.

Ainda, implementa ferramentas poderosas de customização. Isso inclui uma *Command Line Interface* (CLI) com *Callbacks* e arquivos de configuração customizáveis, API *TorchMetrics* para facilitar o cálculo de métricas e *Loggers* personalizáveis como TensorBoard, Wandb e, no nosso caso, Comet[4].

3.2.4 Transformações de Imagens

No conjunto de dados, existe variabilidade significativa nas imagens, que é originada de aspectos como tamanho da imagem, zoom, cor, brilho e blur, além de fatores naturais como plano de fundo e outros objetos na cena. A Figura 3.3 exemplifica essa variabilidade entre imagens de uma única espécie.

Como de praxe na área de *deep learning*, as imagens primeiro passam por uma etapa de pre-processamento sequencial de padronização. Primeiramente, são redimensionadas para um tamanho uniforme de 256×256 com interpolação bilinear, seguido de um corte central 224×224 para focar no centro da imagem (em geral onde as aranhas se encontram) e simultaneamente remover ruídos de fundo. Então, os valores de pixels são normalizados para uma escala comum ($z = (x - \mu)/\sigma$), o que reduz o risco de tornar o modelo sensível à características que dependem da magnitude absoluta dos valores. Utilizamos valores do ImageNet ($\mu = [0.485, 0.456, 0.406]$, $\sigma = [0.229, 0.224, 0.225]$) uma vez que implementamos redes pré-treinadas nesse dataset (Seção 3.2.6).

3.2.5 Aumento de Dados

Um processo importante para melhorar a performance da rede é utilizar um processo de aumento de dados artificialmente a partir dos dados já existentes, denominado *image augmentation*. Para lidar com a variabilidade no nosso conjunto de imagens, aplicamos

[3] https://lightning.ai/

[4] https://www.comet.com

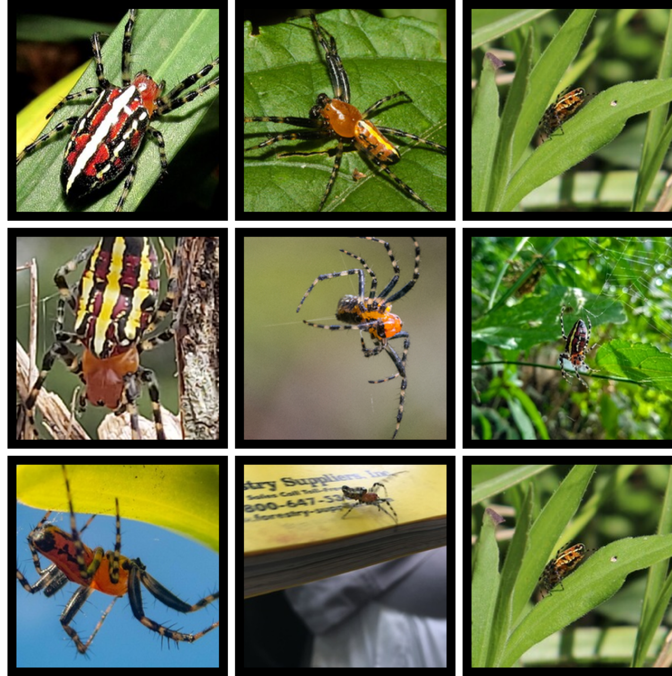


Figura 3.3: *Variação entre fotos da espécie* Alpaida grayi.

diferentes escalas de zoom, luminosidade, rotação e inversão horizontal e vertical (fig 3.4). Particularmente, como as aranhas têm uma ampla gama de orientações devido à sua anatomia especializada, essa estratégia pode se mostrar ainda mais eficaz à medida que a rede aprende a reconhecê-las de vários ângulos. Expor o modelo a mais variações contribui para que o modelo generalize melhor, seja mais robusto a alterações nos dados e reduza possível ocorrência de *overfitting* (Perez e Wang, 2017).

3.2.6 Redes Pré-Treinadas

Para fazer a classificação, utilizamos as arquiteturas mencionadas nas Seções 2.3 e 2.4. Como ponto de partida para treiná-las, empregamos uma abordagem conhecida como *transfer learning*, em que um modelo que já foi pré-treinado em outro conjunto de dados é usado como base para treinar um modelo em uma tarefa relacionada. No escopo deste projeto, usamos modelos pré-treinados disponibilizados pelo PyTorch que foram treinados no conjunto de dados do ImageNet.

No entanto, foram necessárias duas modificações para fazer uso desses modelos. A primeira foi modificar o classificador da rede e alterar a saída da rede para que fosse compatível com o número de espécies de aranhas que constituem o conteúdo de nossos dados. A segunda foi definir quais dos parâmetros permaneceriam congelados (não mudam durante o treinamento) e quais deles seriam treináveis para que o modelo fosse capaz de se adaptar às características específicas do novo conjunto de dados. Yosinski *et al.* (2014) mencionam que a opção de fazer ou não o ajuste fino das primeiras *n* camadas da rede de interesse depende do tamanho do conjunto de dados e do número de parâmetros nas primeiras *n* camadas. Se o conjunto de dados for pequeno e o número de parâmetros for grande, o ajuste fino poderá resultar em *overfitting*, então as camadas permanecem

Figura 3.4: *Exemplo de transformação de imagem com flip, rotação e zoom.*

geralmente congeladas. Por outro lado, se o conjunto de dados for grande ou o número de parâmetros for pequeno, as camadas podem ser treinadas para a nova tarefa para melhorar o desempenho. Ainda, afirmam que redes neurais profundas modernas, quando treinadas em imagens, tendem a aprender características na primeira camada que se assemelham a filtros de Gabor ou manchas de cor. Por outro lado, *features* da última camada dependem consideravelmente da tarefa e do conjunto de dados escolhidos.

Diante desse contexto, observamos em nossos experimentos que descongelar apenas as últimas camadas das redes para aproximadamente 15.5 milhões de parâmetros era um bom equilíbrio entre performance e *overfitting*, resumidos na tabela 3.1.

Modelo	Nome	Treináveis (M)	Não Treináveis (M)	Totais (M)
ResNet	resnet50	15.5	8.5	24.0
ResNeXt	resnext50_32x4d	15.1	8.4	23.5
ConvNeXt	convnext_tiny	15.7	12.3	28.0
ViT	vit_b_16	14.4	71.6	86.0
Swin	swin_v2_t	15.6	12.2	27.8
MaxViT	maxvit_t	17.7	12.9	30.5

Tabela 3.1: *Quantidade de parâmetros treináveis, não treináveis e totais por modelo.*

3.2.7 Otimização de Hiperparâmetros

Antes de iniciar a fase de treinamento de fato, foi necessário definir alguns hiperparâmetros que, ao contrário dos parâmetros do modelo que são aprendidos durante o processo de treinamento, eles determinam as características e o comportamento de aprendizado da rede durante o treinamento, afetando sua capacidade de generalização e, portanto, afetando seu desempenho geral. Diante disso, utilizamos o framework Optuna[5] para explo-

[5] https://optuna.org/

rar um espaço de hiperparâmetros predefinido (tabela 3.2), com o objetivo de identificar os hiperparâmetros ideais para cada modelo em um período de tempo viável. Ao invés de buscar de modo ingênuo, emprega-se o amostrador *Tree-Structured Parzen Estimator* (TPE) (WATANABE, 2023), que utiliza um método eficiente e sofisticado de otimização Bayesiana.

Hiperparâmetro	Espaço de Busca
LR	[1e-4, 1e-1]
L2	[1e-5, 1e-2]
Tamanho Batch	{64, 128, 256}
Função de Perda	{Cross Entropy Loss}
Otimizador	{SGD, Adam, AdamW}
Agendador	{ExponentialLR, StepLR, CosineAnnealingLR}

Tabela 3.2: *Espaço de busca de hiperparâmetros para otimização.*

Os hiperparâmetros finais utilizados para cada rede estão nas Tabelas A.1.

3.2.8 Treinamento das Redes

Para garantir a reprodutibilidade de nossos experimentos, utilizamos arquivos de configurações (yaml) que guardam todas as informações de parâmetros e variáveis de um treinamento, além de ter uma semente global para controlar todos os processos randômicos. Após configurar o pipeline descrito, treinamos as redes nas máquinas disponíveis na rede Vision (IME-USP) com 220GB de memória RAM, processador Intel Xeon Silver 4110 @2.10GHz 16 núcleos e duas GPU NVIDIA GeForce GTX TITAN X 12GB. Estabelecemos precisão de multiplicação de matrizes (*matmul*) com configuração *high*, que fornece tanto precisão e quanto eficiência pois utiliza técnicas de computação com tipos *bfloat16* (HENRY *et al.*, 2019).

O treinamento começa ao carregar as imagens e um arquivo de configuração com todas as características da rede, hiperparâmetros, variáveis e sementes, como explicado anteriormente e utiliza o treinador do *Lightning* para fazer o processo internamente. A rede é treinada em vários *batches* ao longo de, no máximo, 25 épocas. Dependendo do agendador do LR e da técnica de parada antecipada (*early stopping*) para monitorar ocorrências de *overfitting*, a rede pode parar antecipadamente antes de completar todas as épocas. Para avaliar o desempenho, utilizamos três métricas principais: Acurácia Micro, Acurácia Macro e F1 Macro. Essas métricas são capazes de fornecer uma compreensão abrangente da eficácia de um modelo e, ao mesmo tempo, funcionam como meios confiáveis de comparação. Em vista disso, selecionamos como *checkpoint* a época do modelo em que atinge maior Acurácia Macro no conjunto de validação como a métrica representativa do desempenho.

Por fim, treinamos as redes no conjunto combinado de treinamento e validação com mesmos hiperparâmetros e por um número de épocas igual o valor médio das melhores épocas da validação cruzada. Essa rede treinada é a que avaliamos no conjunto de teste.

3.2.9 Análise dos Gradientes

Para fazer uma análise qualitativa das imagens da rede, utilizamos Gradientes Integrados (Sundararajan *et al.*, 2017). A ideia é tentar explicar quais partes da entrada de uma imagem são mais importantes para a decisão do modelo em relação a uma entrada base (*baseline*) por meio de atribuição. Os autores argumentam que, quando atribuímos culpa a uma determinada causa, consideramos implicitamente a ausência da causa como uma linha de base para comparar os resultados. A linha de base é normalmente um ponto neutro em que a previsão deve ter um resultado mínimo ou neutro e, no caso de redes profundas para classificação de imagens, utilizamos uma imagem preta. Seja $F : \mathbb{R}^n \to [0, 1]$ a função que representa a rede profunda e uma entrada $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$. Uma atribuição da previsão na entrada x relativa à base x' é um vetor $A_F(x, x') = (a_1, \ldots, a_n) \in \mathbb{R}^n$ onde a_i é a contribuição de x_i para a previsão $F(x)$.

O método de Gradientes Integrados satisfaz duas propriedades que o tornam confiável para interpretabilidade. A primeira é sensibilidade: se uma entrada e uma linha de base diferirem em apenas uma *feature* e essa diferença levar a previsões distintas pelo modelo, então essa *feature* deve receber uma atribuição diferente de zero. A segunda é a invariância à implementação, ou seja, as atribuições são sempre idênticas para duas redes funcionalmente equivalentes.

Para obter efetivamente os gradientes, os autores consideram uma linha reta em $\mathbb{R}^n$ da base x' até a entrada x, isto é, $\gamma(\alpha) = x' + \alpha(x - x')$ $(\alpha \in [0, 1])$ e acumulam os gradientes ao longo desse caminho. O gradiente integrado ao longo de uma dimensão i é dado por

$$\mathrm{IG}_i(x) ::= \int_{\alpha=0}^{1} \frac{\partial F(\gamma(\alpha))}{\partial \gamma_i(\alpha)} \frac{\partial \gamma_i(\alpha)}{\partial \alpha} d\alpha$$

Esse método de Gradientes Integrados é um desenvolvimento importante na compreensão dos modelos de redes profundas e são essenciais para examinar os mecanismos de decisão. Para calcular os gradientes, utilizamos Captum[6], uma biblioteca para interpretabilidade de modelos criada com base no PyTorch.

3.3 Repositório

Os códigos utilizados estão disponíveis neste repositório do Github e o dataset disponível nesta página.

[6] https://captum.ai/

Capítulo 4

Resultados

Nesse capítulo, iremos fazer uma análise quantitativa e qualitativa dos resultados obtidos e, então, utilizá-las como base de comparação entre as possíveis diferenças de performance das redes neurais.

4.1 Resultados de Treinamento

Após treinar as rede com validação cruzada, as médias, desvios padrões e intervalos de valores das métricas de validação durante o treinamento estão sumarizadas nas tabelas 4.1 e 4.2 abaixo, separadas entre Redes Convolucionais (primeiro bloco) e Transformers Visuais (segundo bloco). Iremos nos referir a acurácia micro por micro, acurácia macro por macro e F1 macro por F1.

Nome	Acurácia Micro (%)	Acurácia Macro (%)	F1 Macro (%)
ResNet	88.36 ± 0.56	86.10 ± 1.08	85.77 ± 0.97
ResNeXt	89.66 ± 0.61	87.26 ± 1.20	87.26 ± 1.17
ConvNeXt	91.17 ± 0.46	89.35 ± 0.70	89.13 ± 0.58
ViT	86.41 ± 0.45	83.65 ± 1.20	83.40 ± 0.92
Swin	89.37 ± 0.50	88.66 ± 1.02	87.26 ± 1.01
MaxViT	90.98 ± 0.52	89.58 ± 0.66	88.89 ± 0.78

Tabela 4.1: *Média e desvio padrão para métricas de validação por modelo.*

Primeiramente, podemos observar que a rede ResNet teve o menor desempenho médio entre as CNNs testadas. O mesmo ocorre para ViT entre os Transformers, porém com as menores métricas entre todos os outros modelos. Esse resultado está alinhado com as percepções apresentadas por Dosovitskiy *et al.* (2021), onde supõem que as redes neurais convolucionais (como a ResNet) tendem a apresentar desempenho superior em conjuntos de dados menores devido ao viés indutivo de localidade, isto é, tendência relacionadas à captura de padrões espaciais locais pela própria estrutura convolucional. Em contrapartida, os Transformers não têm esse tipo de viés indutivo e, como aprendem os padrões relevantes

Nome	Acurácia Micro (%)	Acurácia Macro (%)	F1 Macro (%)
ResNet	87.45 - 88.87	84.79 - 87.29	84.44 - 86.71
ResNeXt	88.92 - 90.46	85.75 - 88.61	85.75 - 88.65
ConvNeXt	90.37 - 91.55	88.35 - 90.41	88.25 - 89.99
ViT	85.83 - 87.11	81.85 - 84.64	82.05 - 84.35
Swin	88.63 - 89.98	87.18 - 90.08	86.06 - 88.45
MaxViT	90.42 - 91.79	88.77 - 90.23	87.84 - 90.23

Tabela 4.2: *Valores mínimos e máximos das métricas de validação por modelo.*

diretamente dos dados, exigem datasets substancialmente maiores. Naturalmente, podemos comparar as evoluções respectivas dessas duas redes para ver como superam as deficiências de seus antecessores. A ResNeXt apresentou não só uma melhora macro considerável de 1.16%, mas também 1.3% e 1.49% para micro e F1. Para Swin, houve uma melhora macro significativa de 5.01%, 2.96% micro e 3.86% F1. Ambas as redes superam ResNet e ViT e Swin supera ResNeXt em acurácia macro por 1.4%.

Por fim, examinamos as redes mais modernas: ConvNeXt atinge acurácia macro média de 89.35%, o que representa um aumento expressivo de 2.09% em relação a ResNeXt, e MaxViT 0.92% em relação ao Swin. Em termos de performance, MaxViT supera ConvNeXt no quesito de acurácia macro por apenas 0.23%, com aproximadamente a mesma variância. Todavia, ConvNeXt atinge maiores médias micro e F1 entre todos os concorrentes além de apresentar o menor desvio padrão. Portanto, argumentamos que ela possui melhor capacidade de classificação para o escopo do nosso conjunto de dados.

À vista dessa análise, é possível fazer ainda duas observações principais. A primeira é que tanto CNNs quanto ViTs são capazes de executar essa tarefa com eficiência e que as respectivas performances observadas se alinham com as performances esperadas tais como relatadas nas bibliografias da área. A segunda é em relação às métricas. Os valores das acurácias micro são aproximadamente $1 \sim 2\%$ maiores que acurácias macro, o que é esperado: em geral, um modelo aprende a prever melhor classes com mais dados. Entretanto, a discrepância média entre acurácia macro e F1 macro é de apenas 0.2% para CNNs e de 0.8% para ViTs, o que sugere que as técnicas que aplicamos para atenuar desequilíbrio de classe são eficazes e os modelos são capazes de apresentar desempenho consistente entre as diversas classes.

Nome	Tamanho do Modelo (MB)	Taxa de Inferência (imgs/s)
ResNet	96.15	351.5 ± 4.7
ResNeXt	94.04	259.3 ± 0.8
ConvNeXt	112.09	147.4 ± 0.4
ViT	344.00	61.4 ± 0.1
Swin	111.14	148.0 ± 2.3
MaxViT	118.80	104.6 ± 0.2

Tabela 4.3: *Tamanho do modelo em Megabytes (MB) e taxa de inferência em número de imagens por segundo (imgs/s).*

Nesse ponto, decidimos escolher um modelo representativo para ser utilizado efetivamente em dados do mundo real. Para tomar essa decisão, utilizamos não somente as métricas de validação obtidas como também medidas de tamanho de arquivo do modelo e taxa de inferência. Para esta, medimos o tempo médio de inferência em 12300 imagens em 5 iterações. Os resultados estão representados na tabela 4.3. Com exceção do ViT, os modelos tem tamanho médio de 105 MB, o que torna esse parâmetro redundante. Entretanto, a taxa de inferência é mais rápida para família ResNet, similar para ConvNeXt, Swin e MaxViT e a pior para ViT.

Diante desses resultados, escolhemos a rede **ConvNeXt** como mais apropriada para a tarefa de classificação de espécies de aranha e plausível de ser utilizada para aplicações externas.

4.1.1 Resultados de Teste

Após treinar cada modelo novamente no conjunto combinado de treinamento e validação, finalmente utilizamos o conjunto de teste independente para avaliar a capacidade de generalização dos nossos modelos. Novamente, agregamos os resultados na tabela 4.4.

Nome	Acurácia Micro (%)	Acurácia Macro (%)	F1 Macro (%)
ResNet	88.85	85.44	86.07
ResNeXt	90.15	87.77	88.48
ConvNeXt	91.04	89.38	89.85
ViT	87.51	84.20	84.41
Swin	89.46	88.37	87.86
MaxViT	90.88	88.84	88.66

Tabela 4.4: *Métricas no conjunto de teste.*

Uma comparação direta entre esta tabela com a tabela 4.1 demonstra que a semelhança entre as métricas de teste e validação é acentuada, por menos de 1% de diferença absoluta, sugerindo um desempenho confiável e boa capacidade de generalização dos modelos com os métodos empregados.

4.1.2 Matrizes de Confusão

Um ponto de interesse é entender quais as situações que levam o modelo a fazer classificações incorretas. Portanto, analisamos as matrizes de confusões dos modelos estudados, onde a nomenclatura das espécies estão abreviadas pelas primeiras três letras dos dois nomes. Na Figura 4.1 temos a matriz de confusão ConvNeXt e as demais na Figura B.1.

Um dos fatores que mais contribui para erro de classificação em todos os modelos é a espécie *Ctenus ornatus* (Cte orn) sendo classificada como *Lycosa erythrognata* (Lyc ery). Na Figura 4.2 abaixo, mostramos essas espécies a fim de ilustrar a alta semelhança entre elas. Embora as redes tenham sido capazes de identificá-las neste exemplo, quando as imagens

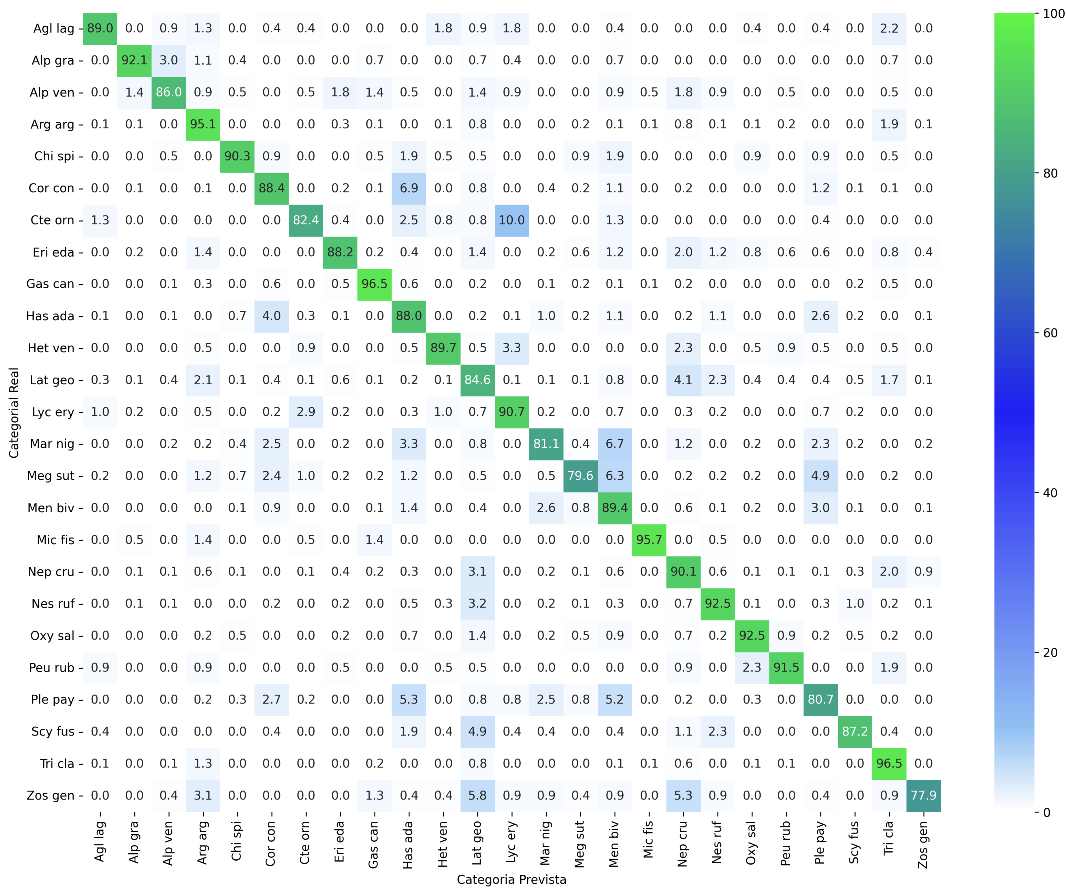


Figura 4.1: *Matriz de confusão ConvNeXt.*

apresentavam condições pouco adequadas (zoom, foco, iluminação, etc) houve maior erro de classificação.

Outro exemplo é a espécie *Corythalia conferta* (Cor con) sendo classificada como *Hasarius adansoni* (Has ada), na Figura 4.3. Como essas aranhas são muito pequenas, algumas fotos de aranhas tem resolução mais baixa nos detalhes da imagem, possivelmente afetando a classificação.

Além desses exemplos principais, há outros pares de espécies que influenciam negativamente na performance final das redes. Para uma análise futura, poderíamos introduzir outras metodologias para tentar lidar com os esses pontos fracos a fim de aperfeiçoar nossas redes.

4.2 Análise de Gradientes

Como mencionado anteriormente, utilizamos gradientes integrados para tentar explicar quais partes da entrada são importantes para a previsão de um modelo. Para cada uma das redes estudadas, mostramos um exemplo qualitativo de uma imagem da espécie *Argiope argentata*. Na Figura 4.4, podemos ver que os mapas individuais de CNNs possuem um certo formato similar à da aranha (4.4a). Esse mapa possui um grau de refinamento contínuo,

(a) *Ctenus ornatus.*

(b) *Lycosa erythrognata.*

Figura 4.2: *Comparação entre espécies Ctenus ornatus e Lycosa erythrognata.*

(a) *Corythalia conferta.*

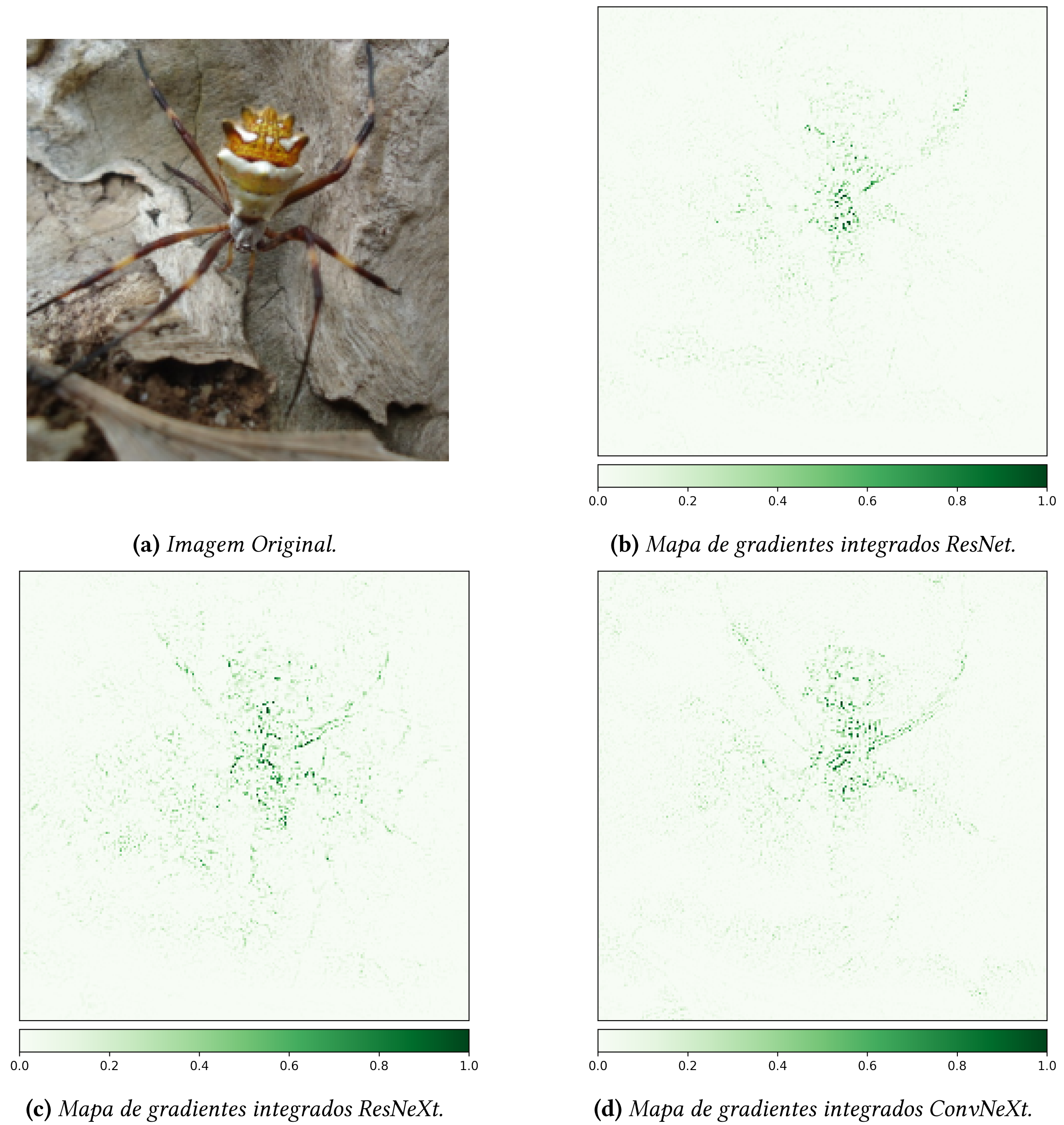
(b) *Hasarius adansoni.*

Figura 4.3: *Comparação entre espécies Corythalia conferta e Hasarius adansoni.*

onde os pontos estão mais dispersos na rede ResNet (4.4b) e ficam mais contidos na rede ConvNeXt (4.4d). Isso pode explicar, em parte, as diferenças de performance observadas entre essas redes.

Para as redes Transformers, a análise é repetida na Figura 4.5. Podemos notar um fato interessante na Figura 4.5b da rede ViT: como a entrada da rede é de dimensão 224×224, temos 14×14 *patches* de tamanho 16×16 visíveis na imagem. Quanto à rede Swin (4.5c), temos um contorno claro, porém com ruídos ao redor, um dos fatores que pode ter afetado a sua performance. Por fim, o mapa MaxViT (4.5d) também aparenta ser mais refinado, com foco especial no abdomen da aranha nesse caso.

De forma geral, também observamos que, tanto para CNNs e ViTs, outras imagens apresentaram um pouco de ruído de fundo (folhas, galhos, etc), com foco predominante no cefalotórax e abdome e, em menor intensidade, nas pernas das aranhas.



(a) *Imagem Original.*

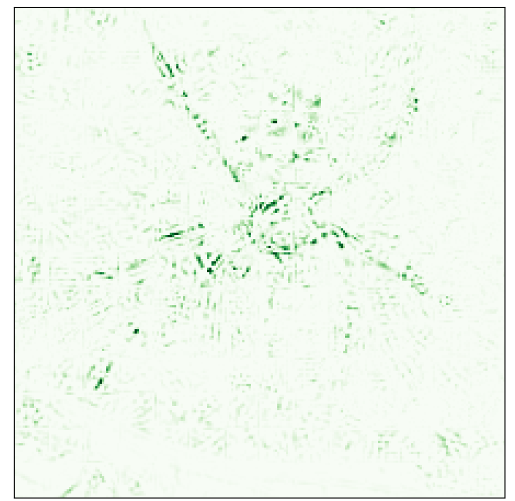
(b) *Mapa de gradientes integrados ResNet.*

(c) *Mapa de gradientes integrados ResNeXt.*

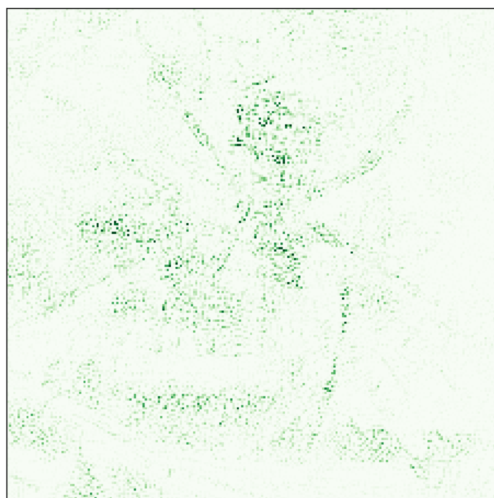
(d) *Mapa de gradientes integrados ConvNeXt.*

Figura 4.4: *Mapas de gradients integrados das redes CNN.*

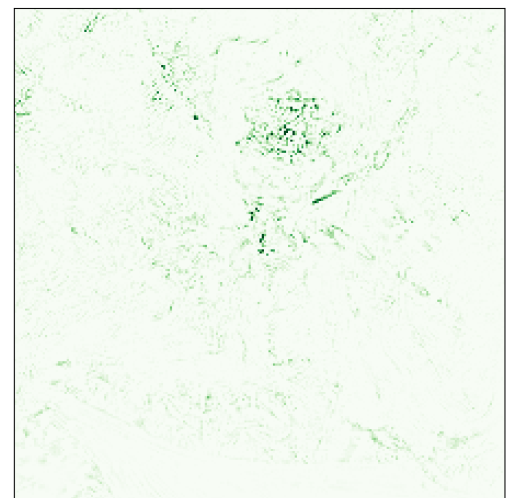
(a) *Imagem Original.*



(b) *Mapa de gradientes integrados ViT.*



(c) *Mapa de gradientes integrados Swin.*



(d) *Mapa de gradientes integrados MaxViT.*

Figura 4.5: *Mapas de gradients integrados das redes ViT.*

Capítulo 5

Conclusão

A motivação deste trabalho foi classificar espécies de aranhas com redes modernas de aprendizado profundo. Utilizando o conjunto de dados de imagens coletadas do site INaturalist, utilizamos metodologias de aprendizado de máquina para treinar e testar as redes e conseguir atingir os propósitos estabelecidos. As arquiteturas utilizadas foram ResNet, ResNeXt e ConvNeXt para redes convolucionais e ViT, Swin e MaxViT para redes Transformers Visuais, onde analisamos as suas respectivas performances quantitativas e qualitativas. Os resultados demonstram que a rede ConvNeXt apresentou o melhor resultado médio e acurácia de 89.38% no conjunto de teste, com boa capacidade de generalização. Assim, essa rede poderia ser potencialmente utilizada para dados reais com aplicações práticas tal como um aplicativo que, uma vez reconhecida a espécie, utiliza diversas informações de um outro banco de dados para fornecer informações de interesse.

Nesse contexto, ainda há espaço para uma futura continuidade do trabalho. A mais intuitiva seria ampliar a quantidade de espécies que as redes conseguem classificar, com uma maior quantidade de dados. Além disso, poderíamos aumentar a resolução da entrada da rede para que possam capturar mais detalhes e características que são perdidos com o redimensionamento, ao custo de recursos computacionais. Também seria possível introduzir metadados no processo de classificação como informações taxonômicas e de geolocalização para possivelmente aumentar a performance das redes. Apesar disso, espera-se que os resultados obtidos nesta análise possam ser utilizados para cumprir o objetivo proposto.

Este trabalho foi apresentado no congresso WUW/SIBGRAPI (6 a 9 de novembro de 2023, Rio Grande-RS) e SIICUSP (14 de novembro e 8 de dezembro, IME-USP).

Apêndice A

Tabelas de Hiperparâmetros

Hiperparâmetro	Valor
LR	0.00121
L2	0.00017
Batch	128
Gamma	0.68445
Otimizador	AdamW
Agendador	ExponentialLR
Função de Perda	Cross Entropy Loss

(a) *Hiperparâmetros ResNet.*

Hiperparâmetro	Valor
LR	0.00549
L2	0.00190
Batch	128
Gamma	0.68887
Otimizador	AdamW
Agendador	ExponentialLR
Função de Perda	Cross Entropy Loss

(b) *Hiperparâmetros ResNeXt.*

Hiperparâmetro	Valor
LR	0.00205
L2	0.00011
Batch	128
Eta_min	0.00013
Otimizador	Adam
Agendador	CosineAnnealingLR
Função de Perda	Cross Entropy Loss

(c) *Hiperparâmetros ConvNeXt.*

Hiperparâmetro	Valor
LR	0.00179
L2	0.00005
Batch	128
Gamma	0.61859
Otimizador	AdamW
Agendador	ExponentialLR
Função de Perda	Cross Entropy Loss

(d) *Hiperparâmetros ViT.*

Hiperparâmetro	Valor
LR	0.00143
L2	0.00017
Batch	128
Gamma	0.63190
Otimizador	AdamW
Agendador	ExponentialLR
Função de Perda	Cross Entropy Loss

(e) *Hiperparâmetros Swin.*

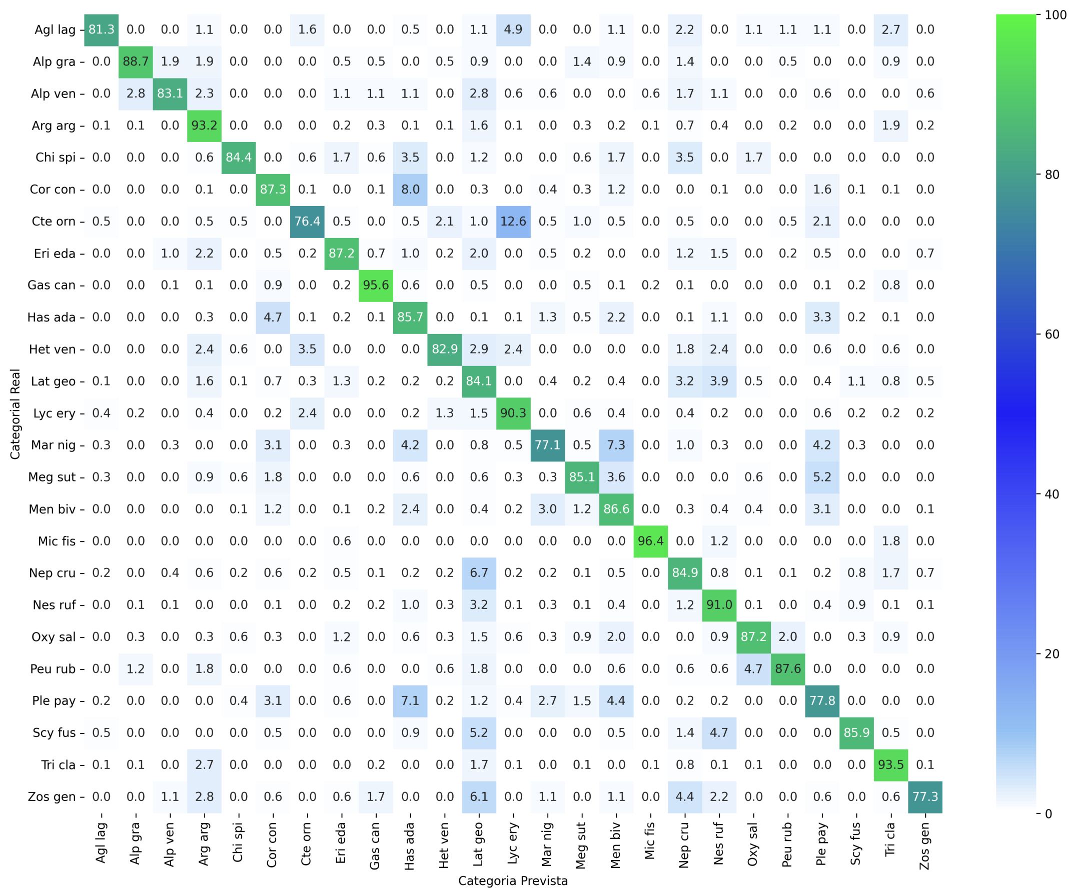
Hiperparâmetro	Valor
LR	0.00168
L2	0.00002
Batch	128
Gamma	0.8276
Otimizador	AdamW
Agendador	ExponentialLR
Função de Perda	Cross Entropy Loss

(f) *Hiperparâmetros MaxViT.*

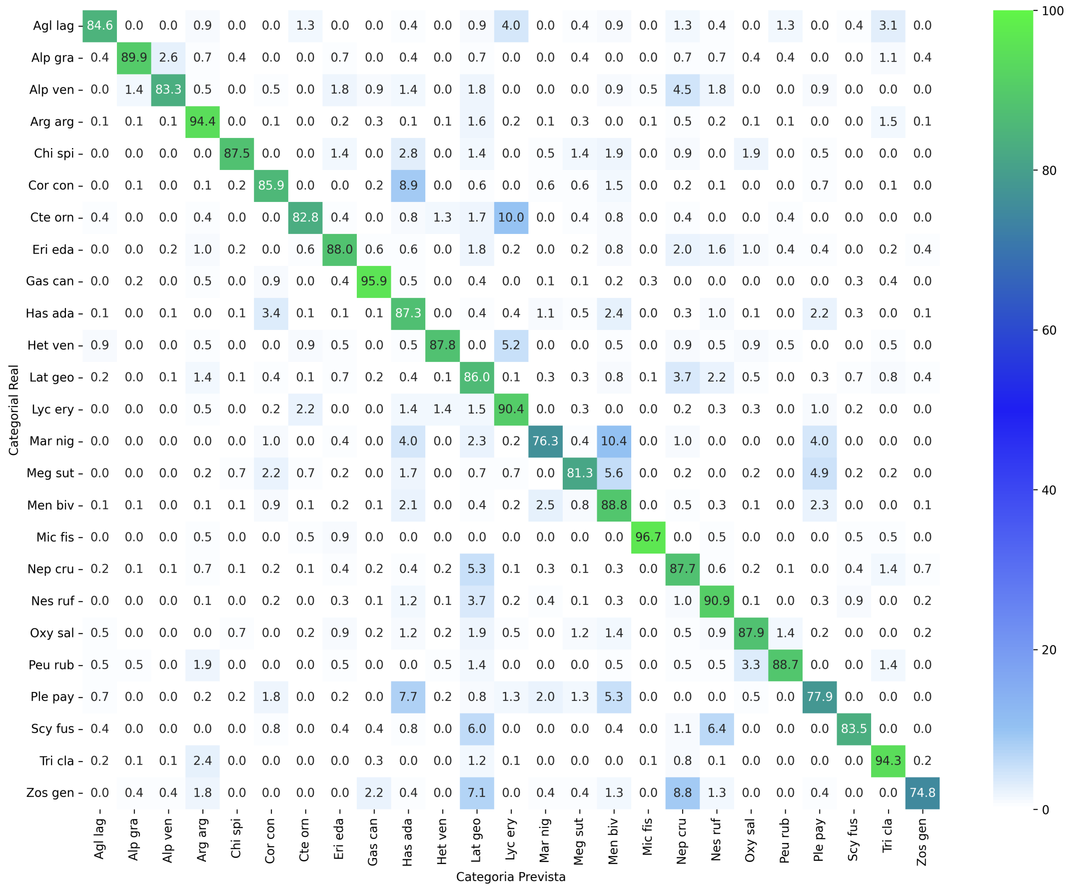
Tabela A.1: *Tabelas de hiperparâmetros utilizados para cada rede.*

Apêndice B

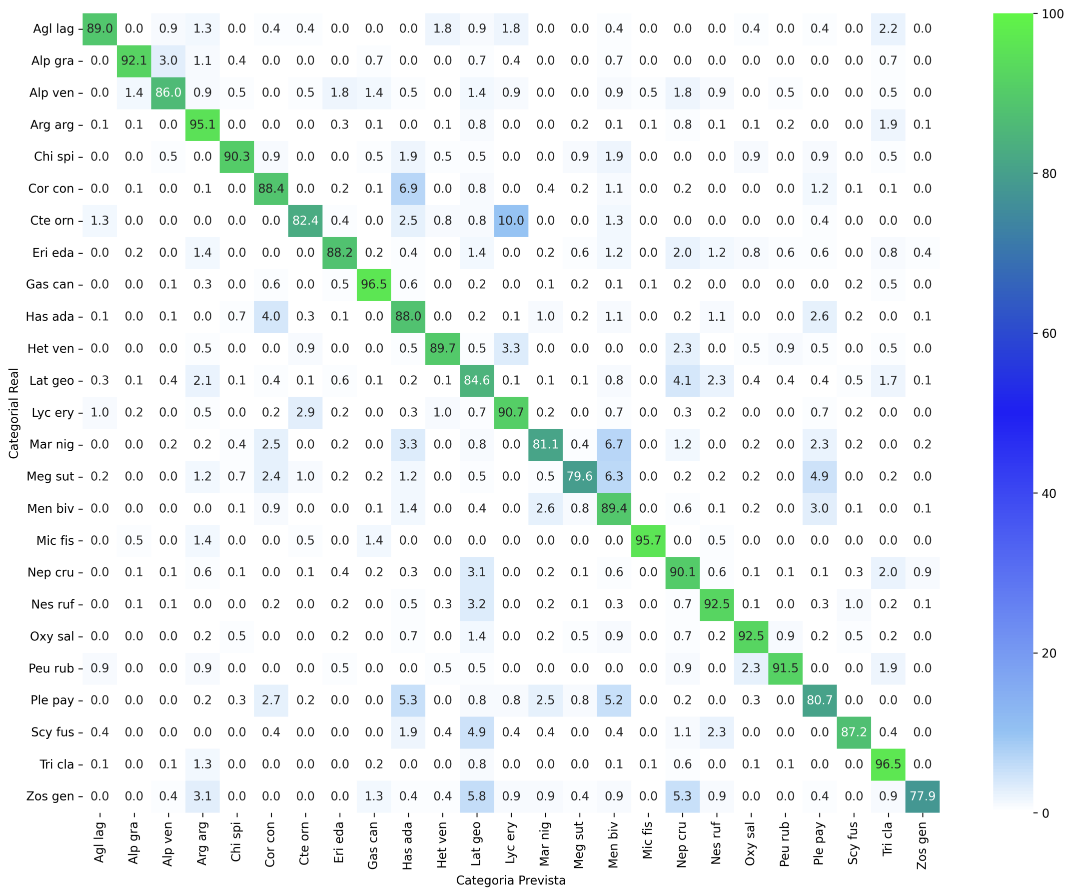
Matrizes de Confusão



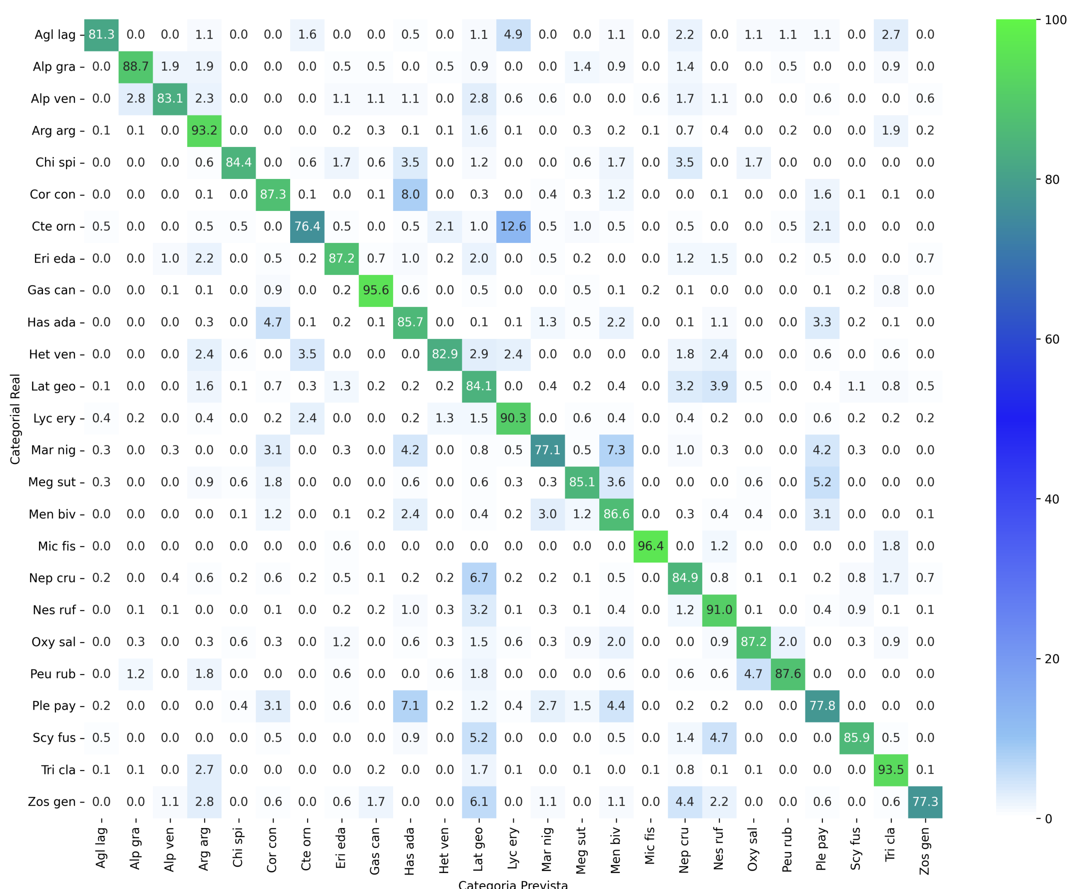
(a) *Matriz de confusão ResNet.*



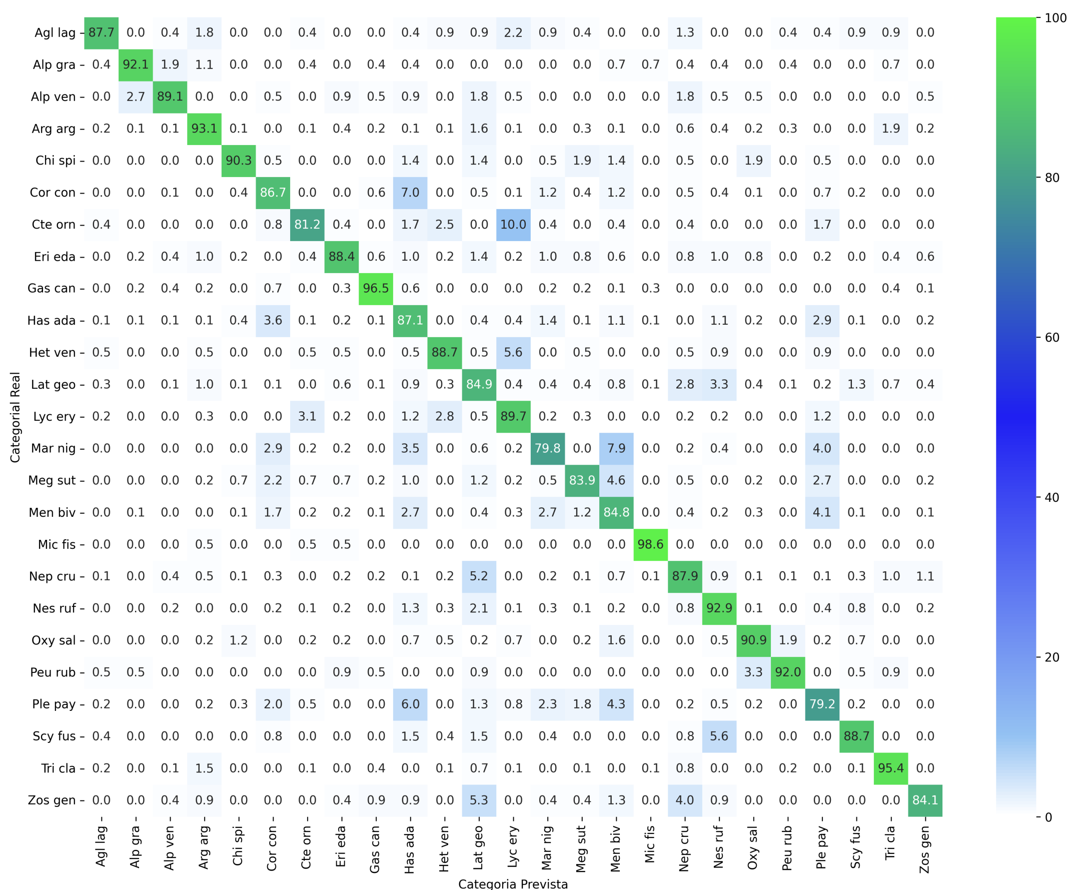
(b) *Matriz de confusão ResNeXt.*



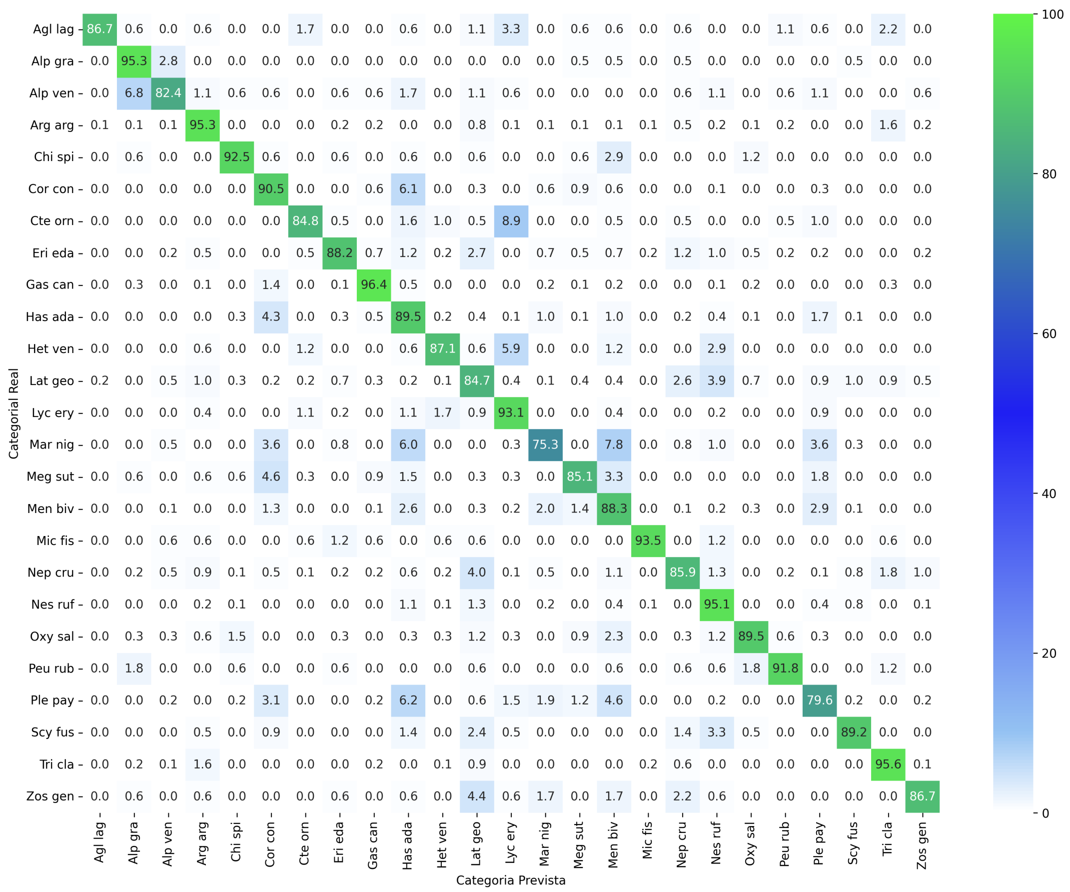
(c) *Matriz de confusão ConvNeXt.*



(d) *Matriz de confusão ViT.*



(e) *Matriz de confusão Swin.*



(f) *Matriz de confusão MaxViT.*

Figura B.1: *Matrizes de confusão dos modelos apresentados.*

Referências

[Abu-Mostafa *et al.* 2012] Yaser S. Abu-Mostafa, Malik Magdon-Ismail e Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012 (citado na pg. 3).

[Brown *et al.* 2020] Tom B. Brown *et al. Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL] (citado na pg. 13).

[Deng *et al.* 2009] Jia Deng *et al.* "Imagenet: a large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848 (citado na pg. 7).

[Dosovitskiy *et al.* 2021] Alexey Dosovitskiy *et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV] (citado nas pgs. 15, 16, 28).

[Goodfellow *et al.* 2016] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. MIT Press, 2016. url: http://www.deeplearningbook.org (citado nas pgs. 1, 9).

[He *et al.* 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren e Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (citado na pg. 10).

[Henry *et al.* 2019] Greg Henry, Ping Tak Peter Tang e Alexander Heinecke. *Leveraging the bfloat16 Artificial Intelligence Datatype For Higher-Precision Computations*. 2019. arXiv: 1904.06376 [cs.MS] (citado na pg. 26).

[Hornik *et al.* 1989] K. Hornik, M. Stinchcombe e H. White. "Multilayer feedforward networks are universal approximators". *Neural Networks* 2.5 (1989), pp. 359–366 (citado na pg. 5).

[Jiang *et al.* 2023] Lingjie Jiang, Baoxi Yuan, Wenyun Ma e Yuqian Wang. "Jujubenet: a high-precision lightweight jujube surface defect classification network with an attention mechanism". *Frontiers in Plant Science* 13 (jan. de 2023). doi: 10.3389/fpls.2022.1108437 (citado na pg. 13).

[Ze Liu *et al.* 2021] Ze Liu *et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV] (citado nas pgs. 16–18).

[Zhuang Liu *et al.* 2022] Zhuang Liu *et al.* *A ConvNet for the 2020s*. 2022. arXiv: 2201.03545 [cs.CV] (citado na pg. 12).

[Nielsen 2018] Michael A Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2018. url: http://neuralnetworksanddeeplearning.com/ (citado na pg. 3).

[Partovi *et al.* 2019] Tahmineh Partovi, Friedrich Fraundorfer, Reza Bahmanyar, Hai Huang e Peter Reinartz. "Automatic 3-d building model reconstruction from very high resolution stereo satellite imagery". *Remote Sensing* 11 (jul. de 2019), p. 1660. doi: 10.3390/rs11141660 (citado na pg. 11).

[Perez e Wang 2017] Luis Perez e Jason Wang. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017. arXiv: 1712.04621 [cs.CV] (citado na pg. 24).

[Secretaria de Vigilância em Saúde 2022] Secretaria de Vigilância em Saúde. *Panorama dos acidentes causados por aranhas no Brasil, de 2017 a 2021*. https://www.gov.br/saude/pt-br/centrais-de-conteudo/publicacoes/boletins/epidemiologicos/edicoes/2022/boletim-epidemiologico-vol-53-no31. 2022 (citado na pg. 1).

[Secretaria Municipal da Saúde 2020] Secretaria Municipal da Saúde. *Aranhas*. https://www.prefeitura.sp.gov.br/cidade/secretarias/saude/vigilancia_em_saude/controle_de_zoonoses/animais_sinantropicos/. 2020 (citado na pg. 1).

[Simonyan e Zisserman 2015] Karen Simonyan e Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV] (citado na pg. 9).

[Sundararajan *et al.* 2017] Mukund Sundararajan, Ankur Taly e Qiqi Yan. *Axiomatic Attribution for Deep Networks*. 2017. arXiv: 1703.01365 [cs.LG] (citado na pg. 27).

[Szegedy *et al.* 2014] Christian Szegedy *et al.* *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV] (citado na pg. 11).

[Tu *et al.* 2022] Zhengzhong Tu *et al.* *MaxViT: Multi-Axis Vision Transformer*. 2022. arXiv: 2204.01697 [cs.CV] (citado nas pgs. 17, 18).

[Vaswani *et al.* 2023] Ashish Vaswani *et al.* *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL] (citado nas pgs. 13, 14).

[Watanabe 2023] Shuhei Watanabe. *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. 2023. arXiv: 2304.11127 [cs.LG] (citado na pg. 26).

[Xie *et al.* 2017] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu e Kaiming He. *Aggregated Residual Transformations for Deep Neural Networks*. 2017. arXiv: 1611.05431 [cs.CV] (citado nas pgs. 11, 12).

[Yosinski *et al.* 2014] Jason Yosinski, Jeff Clune, Yoshua Bengio e Hod Lipson. *How transferable are features in deep neural networks?* 2014. arXiv: 1411.1792 [cs.LG] (citado na pg. 24).

[Zheng *et al.* 2019] Jie Zheng, Andi Xia, Lin Shao, Tao Wan e Zengchang Qin. *Stock Volatility Prediction Based on Self-attention Networks with Social Information*. In Proceedings of the 2019 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr). 2019. doi: 10.1109/CIFEr.2019.8759115 (citado na pg. 15).