Avaliação de Dor em Camundongos com Redes Neurais e Visão Computacional

Marcio Salmazo Ramos

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2025

Marcio Salmazo Ramos

Avaliação de Dor em Camundongos com Redes Neurais e Visão Computacional

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Maurício Cunha Escarpinati
Coorientador: Daniel Duarte Abdala

Uberlândia
2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Título do trabalho**" por **Nome do aluno** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação.**

Uberlândia, ___ de _____ de _____

Orientador: _____
Prof. Dr. Nome do orientador
Universidade Federal de Uberlândia

Coorientador: _____
Prof. Dr. Nome do coorientador
Universidade Federal de Uberlândia
(quando houver)

Banca Examinadora:

_____
Prof. Dr. Membro da banca 1
Instituição de Ensino Superior

_____
Prof. Dr. Membro da banca 2
Instituição de Ensino Superior

Agradecimentos

Faça os agradecimentos àqueles que direta ou indiretamente contribuíram para que você tivesse obtido êxito. Inclua na sua lista agradecimentos aos órgãos de fomento, quando for o caso.

DEVE SER ESCRITO POR ÚLTIMO

Resumo

Segundo a NBR, o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (...) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto

Palavras-chave: Latex. Abntex. Normas USP.

Abstract

Segundo a NBR, o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (. . .) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto

Keywords: Latex. Abntex..

Lista de ilustrações

Figura 1 – Exemplo de pré-processamento para adequação ao treinamento. Fonte: elaboração própria. 31
Figura 2 – Comparativo entre as abordagens de aprendizado supervisionado e não-supervisionado (MORITZ, 2024). 33
Figura 3 – Arquiteura de uma RNA (ALQAHTANI, 2020). 35
Figura 4 – Comparativo entre MLP e DNN (WEHBE, 2020). 36
Figura 5 – Camadas convolucionais em uma CNN (JIANG, 2024). 39
Figura 6 – Estrutura de construção da *ResNet50* (STACK, 2025). 42
Figura 7 – Topologia da arquiteura Vision Transformer (ViT) (HUANG, 2023). . . 50
Figura 8 – Exemplo intuitivo do *transfer learning*, combinando o *Linear Probing* com o *Fine-tuning* (PRABHA, 2021). 53
Figura 9 – Interface inicial do software desenvolvido para a geração da base de dados. Fonte: elaboração própria. 66
Figura 10 – Sub-menu dedicado à extração de frames. Fonte: elaboração própria. . 67
Figura 11 – Sub-menu dedicado à verificação das imagens armazenadas. Fonte: elaboração própria. 68
Figura 12 – Janela principal do software desenvolvido para a construção e o treinamento utilizando a arquitetura ViT. Fonte: elaboração própria. 70
Figura 13 – Parâmetros de configuração dos dados definidos pelo usuário no momento da seleção da base. Fonte: elaboração própria. 71
Figura 14 – Parâmetros estruturais do modelo definidos pelo usuário no momento da construção. Fonte: elaboração própria. 72
Figura 15 – Parâmetros definidos pelo usuário antes da execução do treinamento. Fonte: elaboração própria. 72
Figura 16 – Janela principal do software desenvolvido para a construção e o treinamento utilizando a arquitetura ResNet50. Fonte: elaboração própria. . 73

Lista de tabelas

Tabela 1 – Matriz de confusão ilustrativa para classificação binária 58

Lista de siglas

Sumário

1 INTRODUÇÃO 19
1.1 Motivação 20
1.2 Objetivos 21
1.3 Hipótese 22
1.4 Revisão do Estado da Arte 22
1.5 Contribuições 26
1.6 Organização da Dissertação ou Tese 26

2 FUNDAMENTAÇÃO TEÓRICA 29
2.1 Processamento Digital de Imagens 29
2.2 Aprendizado de máquina 31
2.3 Aprendizado baseado em Redes Neurais 34
2.4 Visão computacional 37
2.4.1 Redes Neurais Convolucionais 38
2.4.2 Transformers e Vision Transformers 43
2.5 Transfer Learning 51
2.6 TensorFlow 54
2.7 Avaliação dos resultados em *Machine Learning* 57
2.7.1 TensorBoard para a avaliação de resultados 60

3 METODOLOGIA DE DESENVOLVIMENTO E PESQUISA 63
3.1 Aquisição e preparação da base de dados 63
3.2 Estudo exploratório dos classificadores 64
3.3 Tratamento dos dados coletados 65
3.4 Construção e treinamento dos modelos 68
3.5 Técnicas para avaliação de resultados 73

4 EXPERIMENTOS E ANÁLISE DOS RESULTADOS 75

4.1 Método para a Avaliação 75
4.2 Experimentos 75
4.3 Avaliação dos Resultados 75

5 CONCLUSÃO 77
5.1 Principais Contribuições 77
5.2 Trabalhos Futuros 77
5.3 Contribuições em Produção Bibliográfica 78

REFERÊNCIAS 79

Capítulo 1

Introdução

Um dos principais desafios para as pesquisas em ambientes clínicos veterinários é a avaliação precisa de estímulos dolorosos em animais, uma vez que a ausência da linguagem verbal nos animais exige métodos indiretos de avaliação, como a observação de sinais fisiológicos e comportamentais. A dor não tratada compromete seriamente o bem-estar destes animais, podendo levar a consequências fisiológicas e comportamentais duradouras, como sensibilização do sistema nervoso central, hiperalgesia (sensibilidade exagerada à dor ou estímulos dolorosos), bem como alterações no comportamento social.(FONTE)

Por serem incapazes de verbalizar suas sensações, a dor em camundongos (e outros animais de modo geral) costuma ser avaliada com base em comportamentos observáveis e alterações fisiológicas. Nesse contexto, foi criada a *Grimace Scale*, uma escala de avaliação desenvolvida para mensurar a dor com base em alterações sutis nas expressões faciais dos animais. A *Mouse Grimmace Scale* (MGS), por exemplo, observa mudanças em cinco regiões faciais distintas do camundongo (orelhas, olhos, bochechas, nariz e bigodes), as quais sofrem alterações expressivas em resposta a estímulos dolorosos.

A introdução da *Grimmace Scale* apresentou um avanço importante para o processo de qualificação da dor, permitindo uma avaliação mais sistemática e baseada em critérios visuais objetivos. Contudo, sua aplicação ainda depende da interpretação subjetiva de observadores humanos, o que introduz variabilidade nos resultados e pode comprometer a precisão e a reprodutibilidade, especialmente em contextos clínicos e experimentais. Adicionalmente, existe uma necessidade de que os analistas que se utilizem desta escala tenham um certo grau de especialização (normalmente médicos veterinários ou fisiologistas), o grande problema está na baixa disponibilidade de tal mão de obra para os diferentes grupos de pesquisa.

Por mais que a *grimmace scale* não exija diretamente observadores especializados, é inegável que sua presença se faça necessária em análises que demandam precisão. Neste contexto, técnicas automatizadas baseadas em visão computacional e aprendizado de máquina surgem como alternativas promissoras para reduzir o viés humano e a necessidade de mão de obra especializada, além de padronizar a análise da dor em animais de laboratório,

gerando resultados mais rápidos e assertivos.

Este trabalho propõe o desenvolvimento de um software cujo objetivo é identificar em imagens de camundongos, os marcadores previstos pela *grimmace scale*, classificando-as automaticamente quanto à presença e ao nível de dor. Optou-se por validar por meio da utilização camundongos uma vez que, em ambiente laboratorial, esses animais são os mais utilizados. Essa prevalência se deve a uma combinação de fatores biológicos e práticos:

- Biologicamente, os camundongos compartilham cerca de 95% de semelhança genética com os seres humanos, o que os torna modelos valiosos para o estudo de resposta a estímulos fisiológicos (incluindo dor), e avaliação de tratamentos. Além disso, sua curta expectativa de vida e ciclo reprodutivo acelerado permitem o acompanhamento de efeitos em múltiplas gerações em um curto intervalo de tempo.

- Do ponto de vista prático, camundongos são de baixo custo de manutenção, facilmente manipuláveis em laboratório, e possuem uma vasta disponibilidade de linhagens geneticamente modificadas. Esses fatores, combinados com o fato de que são pequenos, dóceis e se adaptam bem ao ambiente de laboratório, fazem deles um modelo ideal para experimentação padronizada e reprodutível.

Em conjunto com a *grimmace scale*, a proposta também integra tecnologias de visão computacional associadas ao aprendizado de máquina, como redes neurais convolucionais (CNNs) e *Vision Transformers* (ViTs), a fim de reconhecer padrões faciais sutis de forma rápida, automatizada e escalável.

1.1 Motivação

A motivação para este trabalho parte da premissa de que a dependência da interpretação visual por parte do avaliador, mesmo com o uso de uma escala padronizada, constitui um obstáculo à precisão dos diagnósticos e à eficiência no cuidado com os animais. A metodologia descrita pela MGS foi importante por padronizar os sinais visuais que identificam a presença de dor, permitindo comparações mais rigorosas e sólidas entre estudos. No entanto, os resultados ainda permanecem condicionados à percepção humana, a qual pode comprometer a confiabilidade e precisão das análises em decorrência de vieses.

Adicionalmente, a identificação de dor em animais representa um desafio ético e científico. A indução de dor, se não for devidamente controlada, viola princípios fundamentais de bem-estar animal. O problema está na dificuldade em reconhecer os padrões fisiológicos que denotam o sofrimento, o que pode levar à negligência e, consequentemente, estender o período de desconforto do animal, impactando diretamente em um caráter ético e moral. No âmbito científico, a exposição prolongada à dor pode representar um fator de confusão nos experimentos, afetando outras variáveis fisiológicas e comportamentais (como a

hiperalgesia e o estresse). Caso este cenário não seja bem monitorado e controlado, pode haver um comprometimento da validade dos resultados de um determinado experimento.

O desenvolvimento de uma ferramenta automatizada, baseada em técnicas de aprendizado de máquina e visão computacional, que seja capaz de detectar em tempo real a presença de dor com base nos critérios estabelecidos pela *Grimace Scale* tem forte potencial para sanar os problemas descritos previamente. Tratamentos preventivos ou paliativos podem ser aplicados de maneira segura e eficiente, levando à redução do sofrimento. Análises clínicas, onde diferentes estímulos precisam ser aplicados ao animal, também podem ser concluídas com maior agilidade, sem demandar a análise de um especialista. Tal imediatismo ajuda a mitigar o período de estresse no animal e torna o processo eficaz e confiável.

A consolidação de tal ferramenta também apresenta um grande potencial comercial. A criação de um aplicativo ágil e confiável para a detecção da dor permite que o período do sofrimento animal possa ser consideravelmente reduzido, além de promover a redução de custos em ambientes clínicos, uma vez que dispensa a necessidade de profissionais especializados para este tipo de análise. Tornando-se desejável a inúmeros laboratórios e instituições científicas ao redor do mundo.

1.2 Objetivos

Este trabalho tem como principal objetivo o desenvolvimento de uma ferramenta automatizada, voltada para a detecção e classificação de padrões faciais que expressam a presença de dor em animais. A proposta se baseia na *Grimmace Scale* associada a técnicas de visão computacional e aprendizado de máquina, visando superar as limitações dos métodos convencionais, que ainda dependem predominantemente da observação humana manual.

O estudo também se propõe a avaliar, de forma abrangente, a aplicabilidade da ferramenta em contextos de produção. Para isso, diferentes métodos de classificação baseados em redes neurais, integrados à *Grimace Scale*, foram explorados com o objetivo de identificar a abordagem mais eficaz. Espera-se, portanto, agregar maior confiabilidade aos resultados das análises, tornando-a atrativa para instituições científicas e clínicas na área veterinária.

A partir dessa premissa, o estudo se desdobra nos seguintes objetivos específicos:

1. Levantamento do estado da arte. Identificar e estudar os métodos de classificação de imagens mais adequados que estão disponíveis na atualidade;

2. Especificar detalhadamente os equipamentos e protocolos empregados para a captura das imagens;

3. Desenvolvimento de um software capaz de extrair e organizar as imagens. Tal ferramenta será responsável por gerar a base de dados;

4. Realizar a organização e o pré-processamento das imagens, a fim de adequá-las à entrada dos classificadores;

5. Implementar diferentes modelos de classificadores, essenciais para processar os dados colhidos previamente;

6. Treinar os modelos implementados e armazenar seus resultados;

7. Realizar uma análise comparativa entre diferentes modelos aplicados, buscando determinar quais abordagens são mais eficazes para o projeto;

8. Refinar os modelos, ajustando seus hiper-parâmetros a fim de extrair o melhor resultado possível;

9. Investigar as implicações práticas da implementação dessa tecnologia no setor veterinário, o que envolve uma análise detalhada de custo-benefício, impactos positivos, potenciais desafios técnicos e operacionais.

1.3 Hipótese

A hipótese deste estudo parte da premissa de que classificadores inteligentes possam identificar diferentes regiões de interesse em imagens de camundongos que, ao serem analisadas seguindo os padrões estabelecidos pela MGS, permitem apontar a presença de dor e quantificá-la. Acredita-se que técnicas de visão computacional associadas ao treinamento de máquina, sejam capazes de detectar nuances sutis nas imagens que não são facilmente perceptíveis ao observador não especialista ou que não são capturadas por métodos convencionais.

1.4 Revisão do Estado da Arte

O trabalho em questão lida essencialmente com um problema classificatório baseado em imagens, mais especificamente, a classificação de dor em camundongos com base em seus retratos faciais, utilizando como parâmetro a MGS. Neste contexto, diferentes autores propuseram trabalhos acadêmicos com metodologias interessantes para abordar tal problema.

O estudo desenvolvido por Tuttle et al. 2018 propõe uma abordagem automatizada da MGS para identificar dor em camundongos com base em imagens de suas expressões faciais após laparotomia (procedimento cirúrgico que envolve a abertura da cavidade abdominal por meio de uma incisão na parede abdominal), dessa forma, é possível verificar a

eficácia dos medicamentos analgésicos administrados e reduzir a necessidade de observação humana intensiva e altamente especializada para pontuar imagens. O projeto apresentado nesta monografia, apresenta uma abordagem mais generalizada, com o intuito de classificar dor em diferentes contextos, não se limitando apenas ao cenário pós-cirurgico.

Os autores treinaram uma rede neural convolucional (CNN) baseada na arquitetura InceptionV3, utilizando um conjunto de mais de 5.700 imagens rotuladas manualmente por especialistas. O resultado do treinamento alcançou uma precisão de 94%, mantendo alta correlação com as pontuações humanas (Correlação de Pearson: r = 0,75).

O modelo demonstrou não apenas alta precisão interna, mas também boa capacidade de generalização, identificando corretamente a dor nas imagens avaliadas. Contudo, os autores chamaram a atenção para a dificuldade do modelo, e também dos avaliadores humanos, em diferenciar imagens de camundongos adormecidos daquelas com expressões de dor sutil. Adicionalmente, reforçaram a necessidade de novos conjuntos de treinamento para calibrar o sistema a fim de abranger diferentes raças, como as de pelagem preta ou aguti.

Outro estudo de destaque nesta área foi desenvolvido por Wotton et al. (2020), que propôs um sistema automatizado de fenotipagem comportamental para camundongos submetidos ao teste da formalina. O trabalho busca atender a uma demanda de métodos objetivos, reprodutíveis e escaláveis para avaliação de comportamentos nocifensivos (indicam resposta à percepção de dor), como lamber, morder ou levantar a pata. O foco do estudo foi automatizar a identificação desses comportamentos sem a necessidade da intervenção constante de observadores humanos. Neste estudo os padrões estipulados pela MGS não foram utilizados para a análise, uma vez que tais comportamentos não se restringem à estímulos faciais. Como resultado, o sistema foi capaz de alcançar 98% de concordância com a avaliação humana.

A metodologia adotada foi estruturada em três módulos distintos. Inicialmente, foi dado foco à detecção de pontos-chave *(key point detection)* por meio do DeepLabCut, uma ferramenta baseada em redes neurais convolucionais com arquitetura ResNet-50, especializada em rastrear partes específicas do corpo dos animais sem necessidade de marcações físicas. No segundo módulo, os autores realizaram a extração de características quadro a quadro, calculando distâncias euclidianas e ângulos entre os pontos do corpo em diferentes janelas temporais (as quais serviram como base para representar os comportamentos ao longo do tempo). A etapa final compreendeu a classificação dos comportamentos de dor, particularmente o ato de lamber e morder a pata traseira, utilizando o algoritmo *GentleBoost*, um modelo de aprendizado supervisionado baseado em ensembles de árvores de decisão.

Trabalhos similares de classificação de dor também foram desenvolvidos levando em consideração outras espécies de animais. Ainda no âmbito veterinário, o estudo conduzido por Lencioni et al. (2021) buscou monitorar a dor em cavalos por meio da classi-

ficação automática de suas expressões faciais , seguindo os padrões da *Grimmace Scale*. A construção do sistema envolveu o treinamento de redes neurais convolucionais (CNNs) específicas para cada ponto-chave da face (orelhas, olhos e boca/narinas), as quais foram posteriormente combinadas em um classificador mais abrangente para integrar as decisões das CNNs regionais e classificar a dor em imagens completas dos cavalos.

Os resultados demonstraram que o modelo treinado para analisar a posição das orelhas apresentou a maior acurácia individual (90,3%), enquanto os modelos de olhos e boca/narinas obtiveram acurácias de 65,5% e 74,5%, respectivamente. Ao combinar essas informações para avaliar imagens completas, o sistema atingiu uma acurácia geral de 75,8% na classificação em três níveis de dor. Quando a tarefa foi simplificada para distinguir apenas entre 'presença' ou 'ausência' de dor, a acurácia aumentou significativamente para 88,3%.

Diferentes estudos de classificação de dor também foram desenvolvidos para um contexto humano, como é o caso do trabalho descrito por Karamitsos et al. (2021), que propõe uma abordagem baseada em CNNs para detecção automática de dor em pacientes clínicos, utilizando expressões faciais como principal fonte de informação. O estudo busca oferecer uma solução que possa ser integrada a sistemas hospitalares para monitoramento contínuo e automático da dor em pacientes com limitações verbais.

A metodologia emprega a base de dados *UNBC-McMaster Shoulder Pain Expression Archive*, um dos conjuntos mais amplamente utilizados na literatura, contendo vídeos de 100 pacientes com dor genuína induzida por movimentos articulares. Após o pré-processamento, as imagens foram redimensionadas e utilizadas para treinar uma arquitetura modificada da VGG16, adaptada especificamente para a tarefa de classificação binária.

A avaliação do modelo revelou acurácia de 92,5% e *recall* de 86,96% (mede a capacidade do modelo de identificar corretamente os casos positivos) em um conjunto de teste de 400 imagens. Esses resultados superam os de outras abordagens mencionadas na literatura, incluindo modelos híbridos e redes recorrentes. O estudo também discute o impacto do desbalanceamento das classes e destaca a importância de aumentar a diversidade da base em futuras pesquisas.

Apesar dos avanços alcançados, os autores reconhecem limitações importantes: a necessidade de ampliar e balancear ainda mais a base de dados, melhorar a qualidade das imagens utilizadas. Também destacam que o modelo atual ainda não contempla comportamentos faciais que possam ser confundidos com dor, sugerindo que etapas futuras de aprimoramento envolvam a detecção e correção de falsos positivos e a inclusão de mais variabilidade nos dados de treinamento.

Os estudos apresentados adotaram abordagens similares para o problema classificatório, com destaque para o uso de Redes Neurais Convolucionais (CNNs). Esses modelos foram projetados especificamente para processar dados com estrutura espacial, como ima-

gens, e se popularizaram na área devido à capacidade de extrair características relevantes de forma automatizada, à eficiência computacional e ao histórico consolidado na literatura (especialmente após o sucesso da arquitetura AlexNet, em 2012).

Apesar dos avanços proporcionados, o foco em padrões locais dificulta a integração de informações mais amplas dentro da imagem, o que pode comprometer o desempenho em tarefas que exigem maior contextualização espacial. Além disso, operam sobre *grids* fixos e requerem pré-processamentos específicos, o que pode limitar sua flexibilidade frente a diferentes tipos de entrada. Nesse cenário, surgem novas alternativas, como os Vision Transformers (ViT), que propõem uma abordagem distinta para o aprendizado de representações visuais.

O estudo desenvolvido por Barman et al. (2024) propõe uma solução para a detecção de doenças em folhas de tomate, trazendo comparativos entre a ViT e a *Inception v3*, um modelo baseado em CNNs. Adicionalmente, é proposto o desenvolvimento de um aplicativo para dispositivos *Android* capaz de realizar a detecção em tempo real a partir de imagens capturadas por smartphones, com potencial aplicação na agricultura de precisão.

Os autores utilizaram um subconjunto do *PlantVillage Dataset* como banco de imagens, contendo 10.010 imagens divididas entre folhas saudáveis e nove classes de doenças comuns do tomateiro. Os dados foram pré-processados, normalizados e utilizados para treinar ambos os modelos com 30 épocas em cada arquitetura. Como resultado, a arquitetura baseada na ViT demonstrou maior precisão, recall e F1-score em quase todas as classes de doença, obtendo acurácia de 97,34% no treino e 95,76% na validação. A Inception V3 obteve acurácia de 94,8% no treino e 94,02% na validação. O que demonstra um forte potencial das ViTs para o processo classificatório.

No contexto de detecção de dor, o trabalho conduzido por Fiorentini, Ertugrul e Salah (2022) propõe a primeira pipeline totalmente baseada em *Vision Transformer* e *Video Vision Transformers* (ViViT) na tarefa de classificação binária de dor facial. Para o treinamento, foi utilizado o dataset UNBC-McMaster Shoulder Pain, que contém vídeos de pacientes com dor induzida clinicamente e rotulados com base na escala PSPI (Prkachin and Solomon Pain Intensity), a qual codifica ações faciais relacionadas à dor.

Ambos os modelos ViT e ViViT foram treinados para prever a presença de dor com base no PSPI. As imagens foram registradas em 3D com a ferramenta PRNet e convertidas para uma visualização frontal padronizada usando *Face3D*, aumentando a consistência entre quadros e indivíduos. Importante salientar que o ViViT é uma extensão do Vision Transformer (ViT), adaptada especificamente para dados de vídeo. Enquanto o ViT processa imagens estáticas, o ViViT processa sequências de quadros, capturando informações espaciais e temporais simultaneamente.

Como resultado, os modelos ViT-1 e ViViT-1 alcançaram um F1-score médio de 0.55, superando abordagens anteriores baseadas em CNNs, como o modelo *Pain Facial Deep Learning* (PFDL), que obteve F1 = 0,47. O modelo ViViT-2, embora com média ligei-

ramente inferior (F1 = 0,49), apresentou menor desvio padrão, indicando maior consistência entre as dobras de validação. Um dos principais diferenciais do estudo é a análise qualitativa dos mapas de atenção, que demonstram o foco dos modelos em regiões anatomicamente relevantes para a detecção de dor, convergindo com as especificações descritas pela escala PSPI. Esse aspecto confere aos modelos um caráter interpretável, qualidade especialmente desejável em aplicações clínicas e experimentais sensíveis, como a avaliação automatizada da dor.

Compreender as características e diferenças entre as arquiteturas ViTs e CNNs é essencial para contextualizar os avanços recentes em classificação de imagens no campo da visão computacional. O estudo descrito por Maurício et al. (2023) apresenta uma revisão sistemática da literatura, com foco na comparação entre ambas as arquiteturas em tarefas de classificação. Considerando a consolidação das CNNs como abordagem predominante e o surgimento recente dos ViTs, os autores analisam as condições em que cada arquitetura apresenta melhor desempenho, além de discutir suas vantagens, limitações e aplicações práticas.

Os resultados mostraram que os ViTs tendem a apresentar melhor desempenho quando aplicados a *datasets* menores, por conseguirem capturar relações globais via mecanismos de autoatenção. Em contrapartida, as CNNs foram mais robustas em conjuntos de dados maiores e mais diversos, além de manterem melhor generalização em tarefas com menos ruído. Arquiteturas híbridas — que combinam convoluções com atenção — também se destacaram por unir o melhor dos dois mundos. A revisão conclui que não há um "vencedor absoluto" entre ViTs e CNNs, mas sim cenários mais adequados para cada arquitetura.

1.5 Contribuições

Liste as contribuições do seu trabalho. Lembre-se que publicações não são contribuições científicas do seu trabalho. Haverá uma seção específica com esse fim. **Fazer após a conclusão do projeto, dessa forma é possível definir quais foram as contribuições que o projeto ofereceu para a comunidade científica**

1.6 Organização da Dissertação ou Tese

O presente documento encontra-se dividido em X capítulos organizados da seguinte forma:

- No capítulo 2 são apresentados os fundamentos teóricos essenciais para a compreensão das ferramentas e técnicas apresentados nesta pesquisa, em especial, os fundamentos relacionados ao Processamento Digital de Imagem (PDI), Aprendizado de Máquina e Redes Neurais.

❑

Capítulo 2

Fundamentação Teórica

Os modelos de aprendizado profundo voltados para a análise espacial, como as Redes Neurais Convolucionais (CNNs) e os *Vision Transformers* (ViTs), obtiveram maior predominância para os problemas classificatórios, principalmente quando aplicados à detecção de dor. Para sua implementação, torna-se necessária a compreensão de diversas etapas que estruturam o processo de construção de classificadores eficientes. Entre essas etapas, destacam-se o pré-processamento dos dados, necessário para adaptar o conjunto de imagens aos requisitos de entrada da rede; a definição da arquitetura do modelo, com base nas características do problema e na complexidade dos padrões visuais envolvidos; e a validação dos resultados, por meio de métricas específicas que avaliem o desempenho e a capacidade de generalização do classificador. Cada uma dessas etapas é fundamental para garantir a robustez e a confiabilidade do sistema, sendo exploradas ao longo desta seção.

2.1 Processamento Digital de Imagens

O processamento digital de imagens (PDI) é uma área da ciência da computação voltada à manipulação e análise de imagens representadas em formato digital. Seu foco está nas etapas que ocorrem após a aquisição da imagem, abrangendo desde o tratamento das informações visuais até a extração de informações relevantes que possam ser utilizadas em processos de análise ou tomada de decisão.

Uma imagem digital pode ser compreendida como uma representação discreta de dados que incorpora tanto informações espaciais (relacionadas à sua estrutura geométrica) quanto informações de intensidade, como cores, contraste e níveis de luminosidade. Trata-se de um meio eficiente de comunicação visual, capaz de transmitir informações visualmente atraentes de forma rápida e acessível, facilitando a compreensão e a análise de dados complexos. Por consequência, as imagens digitais tornaram-se uma ferramenta fundamental em diversas áreas do conhecimento, incluindo a ciência, medicina, segurança, comunicação e tecnologia (FONTE).

Com a aquisição da imagem em formato digital, torna-se possível aplicar diferentes operações computacionais para adaptá-la ao contexto ideal. As operações típicas incluem a normalização de intensidades, realce de contraste, remoção de ruído, segmentação de regiões de interesse e extração de características visuais, como formas, texturas e contornos. Essas transformações visam preparar as imagens para etapas posteriores, como classificação automatizada, reconhecimento de padrões ou visualização interpretável. Dessa forma, o PDI atua como uma etapa fundamental para garantir a qualidade, consistência e eficácia na extração de informações visuais.

De acordo com diferentes literaturas, o processamento digital da imagem é sub-divido em diferentes etapas bem definidas, indo desde a aquisição até o pós-processamento (Importante destacar que, dependendo do contexto, nem todas as etapas são obrigatórias). São elas:

- **Aquisição da imagem:** Trata-se da etapa inicial, em que a imagem é capturada por sensores, câmeras digitais ou dispositivos específicos. O resultado é uma representação numérica em formato matricial que descreve a imagem. Essa imagem pode conter ruídos, distorções ou variações de iluminação que precisarão ser tratadas;

- **Pré-processamento:** Etapa essencial para garantir a consistência e reduzir variações indesejadas no conjunto de dados. Tem como objetivo melhorar a qualidade da imagem e prepará-la para etapas mais avançadas de análise. As operações comuns incluem: redimensionamento (ajuste do tamanho da imagem), normalização (padronização intensidade dos *pixels*), filtros para remoção de ruído (ex: filtros de média, Gaussiano) e ajuste de contraste ou brilho;

- **Segmentação:** Etapa responsável por dividir a imagem em regiões significativas, informações visuais relevantes são destacadas para facilitar análises futuras. Métodos comuns de segmentação envolvem a limiarização, detecção de bordas e agrupamento de *pixels* semelhantes;

- **Extração de características:** Etapa responsável por extrair as informações visuais mais expressivas de uma imagem, como formas, texturas, bordas, gradientes e padrões. Tais características são utilizadas para alimentar os algoritmos de classificação ou reconhecimento, como CNNs ou ViTs.

- **Classificação ou Reconhecimento:** Aqui são utilizados diferentes algoritmos voltados para a interpretação dos padrões extraído, com o objetivo de atribuir rótulos ou categorias aos dados visuais.

- **Pós-processamento:** Etapa opcional, voltada para o refinamento dos dados e dos resultados. De acordo com a necessidade, seriam aplicadas correções de classificação ou uma visualização gráfica dos resultados.

No contexto deste estudo, um bom exemplo prático de aplicação das técnicas de PDI pode ser observado na preparação de dados visuais para entrada em modelos de aprendizado de máquina. De forma geral, esses modelos exigem que as imagens de entrada sigam um padrão específico de formatação, tanto em termos de tamanho quanto de estrutura e escala de valores. Assim, é possível garantir compatibilidade com a arquitetura do modelo e qualidade no processo de aprendizado.

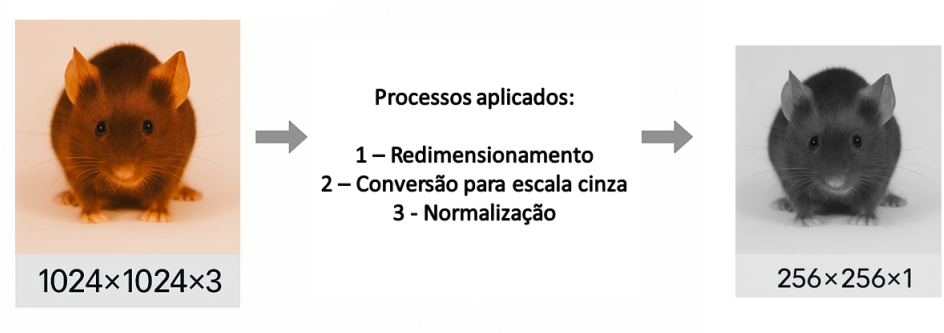

Figura 1 – Exemplo de pré-processamento para adequação ao treinamento. Fonte: elaboração própria.

O exemplo descrito pela Figura 1, supõe que um determinado modelo de rede espera imagens com dimensões fixas de 256×256 pixels e apenas um canal (escala de cinza). Ao trabalhar com uma base contendo imagens originais de 1024×1024 pixels em RGB (3 canais), é necessário aplicar uma sequência de transformações: redimensionamento da imagem para o tamanho esperado, conversão para tons de cinza (reduzindo os canais de cor) e normalização dos valores de *pixel*, reescalando-os da faixa original para o intervalo entre 0 e 1.

Essas etapas não apenas asseguram que os dados estejam tecnicamente compatíveis com o modelo, como também evitam distorções nas ativações internas da rede, reduzem o risco de instabilidade no treinamento e contribuem para uma melhor generalização do modelo em dados novos.

Para obter um conhecimento mais detalhado sobre as temáticas que envolvem o processamento digital de imagens, ver o livro Processamento Digital de Imagens, GONZALES, 2009. Por fim, enquanto o processamento digital foca na melhoria e extração de atributos da imagem, a visão computacional vai além, buscando atribuir significado a esses atributos com base em modelos computacionais de inferência e aprendizado.

2.2 Aprendizado de máquina

O aprendizado de máquina, ou machine learning (ML) advém de uma área da inteligência artificial que se dedica a construir sistemas capazes de interpretar e atribuir significados ao conjunto de dados. De modo geral, a interpretabilidade dos dados está

associada à identificação de padrões complexos e á extração de características relevantes dos dados fornecidos.

Esse processo permite previsões, classificações e tomadas de decisão de maneira automática e generalizada, ou seja, ao invés de um sistema ser programado levando cada situação em consideração, o sistema passa a aprender qual ação tomar de forma autônoma a partir de exemplos.

A aprendizagem de máquina podem ser categorizada de três maneiras distintas, de acordo com o método de interpretar e rotular os dados, são elas: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço.

- **Aprendizado supervisionado:** uma das abordagens mais comuns de ML, onde um modelo é treinado com base em um conjunto de dados previamente conhecidos e rotulados. Isso significa que o algoritmo recebe pares de entrada e saída conhecidos e aprende a mapear essas entradas para as saídas corretas. A ideia principal é ajustar o modelo para que ele possa generalizar e fazer previsões precisas para novos conjuntos de dados;

- **Aprendizado não supervisionado:** uma abordagem comum para lidar com dados que não possuem rótulos definidos. Neste caso, a ideia do algoritmo é explorar a estrutura subjacente dos dados e identificar padrões ocultos, agrupando os dados com base em características similares.

- **Aprendizado por reforço:** uma abordagem diferente, que atua por meio de um 'agente' responsável por aprender com base na interação com um determinado ambiente. A cada ação tomada, o agente recebe uma recompensa (ou penalidade) com base no impacto de suas decisões. O objetivo é maximizar as recompensas acumuladas ao longo do tempo, ajustando o comportamento do agente para melhorar a performance.

Entre as diferentes abordagens no campo do aprendizado de máquina, destaca-se o agrupamento de dados (*clustering*), uma técnica de aprendizado não supervisionado considerada uma das mais simples e intuitivas. Seu objetivo é particionar um conjunto de dados em grupos formados com base na semelhança entre as amostras, sem a necessidade de rótulos previamente definidos. Para isso, os algoritmos de agrupamento buscam organizar os dados de forma coesa, de modo que os elementos pertencentes a um mesmo grupo compartilhem características semelhantes, enquanto os de grupos distintos apresentem diferenças significativas.

O conceito de similaridade é central nesse processo, pois se refere a uma medida que quantifica o grau de semelhança entre duas ou mais amostras. Trata-se de um critério matemático que avalia a proximidade entre os elementos, com base em suas representações numéricas. Entre as métricas mais utilizadas destacam-se: a distância Euclidiana, a

distância de Manhattan e a similaridade do cosseno. De maneira geral, quanto menor a distância entre os vetores, maior é a similaridade, e, consequentemente, maior a probabilidade de que os dados pertençam ao mesmo grupo. Para maiores detalhes sobre técnicas de agrupamento e similaridade, consultar a literatura: Data Clustering: A Review

Embora técnicas de agrupamento, como K-means, DBSCAN e Expectation Maximization, não façam uso de rótulos previamente definidos, seus resultados podem ser interpretados como uma forma de classificação automática, uma vez que associam cada amostra a um grupo específico. A principal diferença, no entanto, está na abordagem não supervisionada dessas técnicas, que realizam essa associação com base em métricas matemáticas de similaridade, sem o conhecimento prévio das classes verdadeiras. Em contrapartida, o aprendizado supervisionado parte do pressuposto que cada amostra do conjunto de treinamento possui um rótulo conhecido (ground-truth), o que permite ao modelo ajustar seus parâmetros de forma iterativa, a fim de minimizar os erros cometidos durante o processo de aprendizagem. A ideia central visa construir modelos capazes de generalizar o conhecimento adquirido, ou seja, classificar corretamente novas amostras com base nos padrões aprendidos.

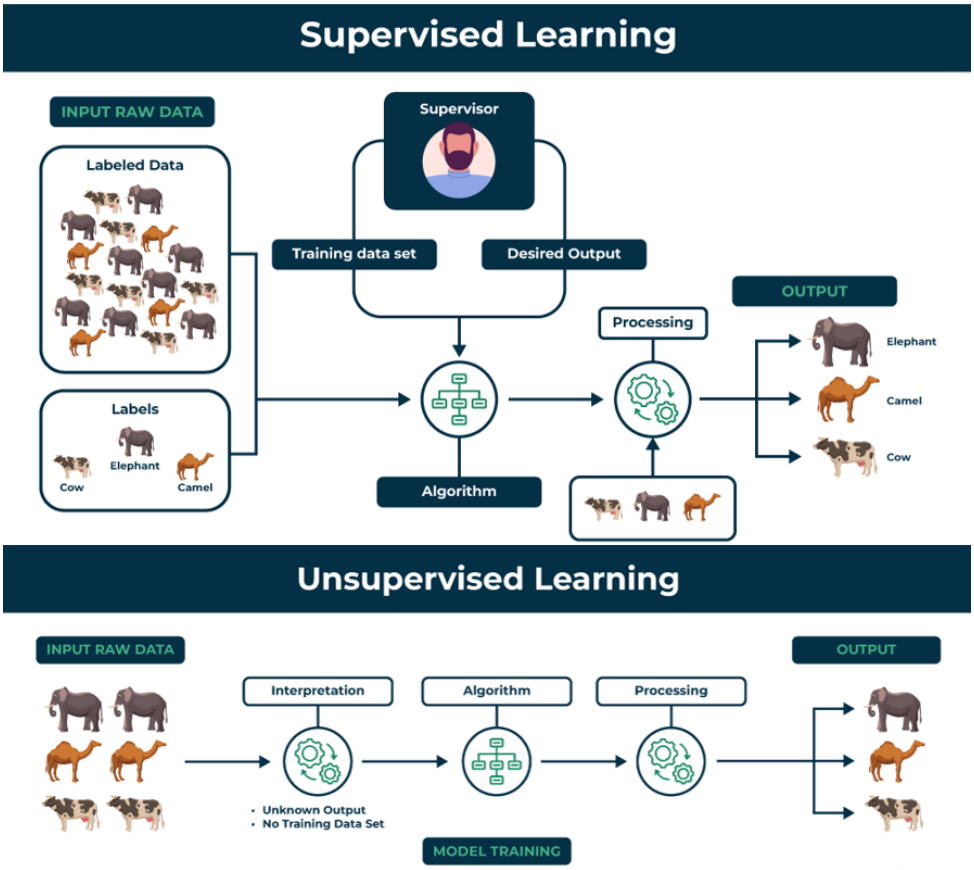

Figura 2 – Comparativo entre as abordagens de aprendizado supervisionado e não-supervisionado (MORITZ, 2024).

A Figura 2 apresenta um comparativo entre as abordagens de aprendizado supervisionado e não supervisionado. No aprendizado não supervisionado, observa-se que os dados

de entrada não possuem rótulos associados, sendo interpretados e organizados com base em padrões similares observáveis. No exemplo apresentado, os animais são agrupados automaticamente conforme suas características visuais, sem conhecimento prévio de suas espécies. Já na abordagem supervisionada, destaca-se a presença de um agente supervisor, responsável por fornecer ao sistema um conjunto de dados previamente rotulado. A partir dessas associações explícitas entre entrada e saída, o algoritmo é treinado para reconhecer padrões e aprender a relacioná-los com as classes corretas.

A literatura apresenta uma ampla variedade de técnicas supervisionadas para problemas de classificação e regressão. Entre os métodos clássicos, destacam-se: Regressão Logística, Árvores de Decisão e Support Vector Machines (SVM). Para maiores detalhes sobre tais técnicas, consultar o livro Machine Learning for Brain Disorders, Neuromethods - CAP 2. Já entre os modelos mais avançados, destacam-se as técnicas baseadas em redes neurais, que tem como principal diferencial a automatização no processo de extração de características e reajuste de pesos (via *backpropagation*), tornando-se especialmente úteis para a análise e classificação de dados complexos. É importante destacar que, as técnicas supervisionadas de aprendizagem baseadas em redes neurais exigem um treinamento prévio com uma base robusta, a fim de promover o ajuste de pesos e, consequentemente a generalização

2.3 Aprendizado baseado em Redes Neurais

Redes Neurais Artificiais (RNAs) constituem uma das principais arquiteturas do aprendizado de máquina, especialmente quando se busca modelar relações complexas em dados de alta dimensionalidade. Inspiradas no funcionamento do cérebro humano, essas redes são compostas por unidades de processamento chamadas neurônios artificiais, organizadas em camadas interconectadas, conforme ilustrado pela figura 3. Um dos principais diferenciais das RNAs é sua capacidade de extrair representações ocultas e abstratas de forma autônoma, o que as torna altamente eficazes em tarefas como classificação, regressão e tomada de decisão.

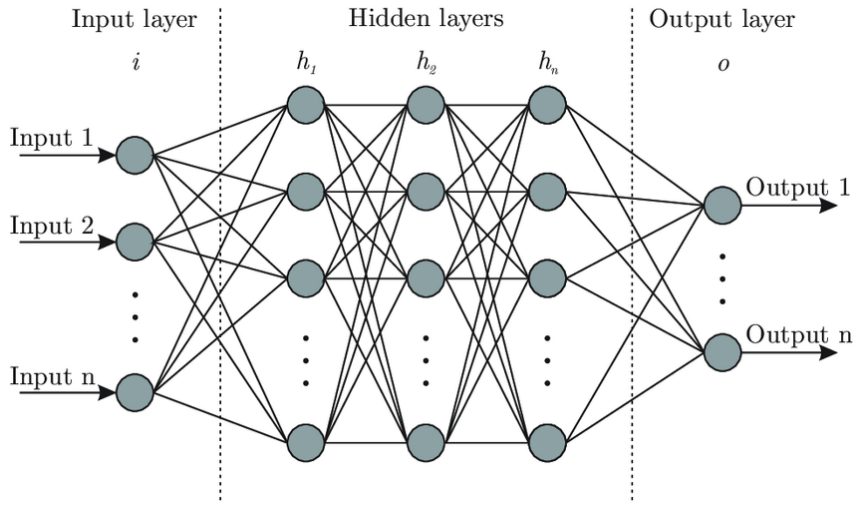

Figura 3 – Arquiteura de uma RNA (ALQAHTANI, 2020).

'Input layer' é responsável exclusivamente pelo transporte de dados para as camadas ocultas; *'Hidden Layer'* são responsáveis por realizar transformações mais críticas que permitem a extração de características dos dados pela rede; *'Output Layer'* é responsável por fornecer a previsão ou o resultado do modelo.

Os neurônios artificiais são as unidades básicas de processamento das redes neurais. Sua função é receber estímulos, processá-las por meio de operações matemáticas simples e transmitir um sinal de saída para os neurônios da camada seguinte. Embora sua estrutura seja conceitualmente simples, a organização desses neurônios em múltiplas camadas interconectadas permite à rede modelar relações complexas entre os dados. À medida que a informação percorre as camadas mais profundas, os neurônios atuam de forma cooperativa, identificando padrões em diferentes níveis de abstração. Por exemplo, em uma rede neural treinada com imagens, os neurônios das camadas iniciais tendem a aprender padrões simples bordas e texturas; já nas camadas intermediárias e profundas, a rede passa a reconhecer formas mais complexas, como objetos ou até mesmo expressões faciais completas.

Um componente essencial no funcionamento dos neurônios artificiais é o peso sináptico (valor numérico atribuído a cada conexão da rede), responsável por definir o grau de importância de cada entrada recebida pelo neurônio. Esses pesos representam os principais parâmetros ajustáveis do modelo, sendo modificados iterativamente durante o treinamento por meio do algoritmo de retropropagação do erro (backpropagation), que consiste em propagar o erro de saída final da rede para cada uma das camadas anteriores. Durante esse processo, são calculados os gradientes da função de erro em relação a cada peso, permitindo a atualização dos parâmetros de forma progressiva e minimizando os erros de predição. Para maiores detalhes técnicos acerca do funcionamento interno de uma RNA, consultar o survey State-of-the-art in artificial neural network applications: A survey

As redes neurais artificiais podem ser implementadas de diferentes formas, conforme os requisitos específicos de cada aplicação. Essas variações buscam, por exemplo, melhorar

o desempenho, reduzir o custo computacional ou adaptar a rede a diferentes tipos de dados. Nesse contexto, o termo arquitetura de rede refere-se à forma como os neurônios artificiais são organizados e conectados entre si, incluindo a definição das camadas, funções de ativação, direção do fluxo de dados e outras estratégias como regularização e métodos de treinamento.

Dentre as arquiteturas mais conhecidas, destaca-se a Multilayer Perceptron (MLP), considerada uma evolução direta do perceptron simples, que é modelo base de neurônio artificial. A MLP é composta por camadas densamente conectadas (ou fully connected), o que significa que cada neurônio de uma camada está ligado a todos os neurônios da camada seguinte. Além disso, trata-se de uma rede do tipo feedforward, pois a informação flui em uma única direção, da entrada até a saída, sem conexões cíclicas ou recorrentes.

Essa arquitetura é capaz de aprender representações complexas e não lineares dos dados, sendo especialmente útil em tarefas de classificação e regressão com dados vetoriais ou tabulares. No entanto, seu desempenho pode ser limitado quando aplicada a dados com estruturas complexas ou alta dimensionalidade. Nesse contexto, as MLPs assumem um papel fundamental por servirem como base para o desenvolvimento de redes neurais mais profundas, como as Redes Neurais Densas (DNNs). Conforme ilustrado na figura 4, as DNNs mantêm a estrutura das MLPs, mas ampliam sua profundidade com múltiplas camadas ocultas, o que aumenta a capacidade de abstração e expressividade da rede, capacitando-a para tarefas mais exigentes.

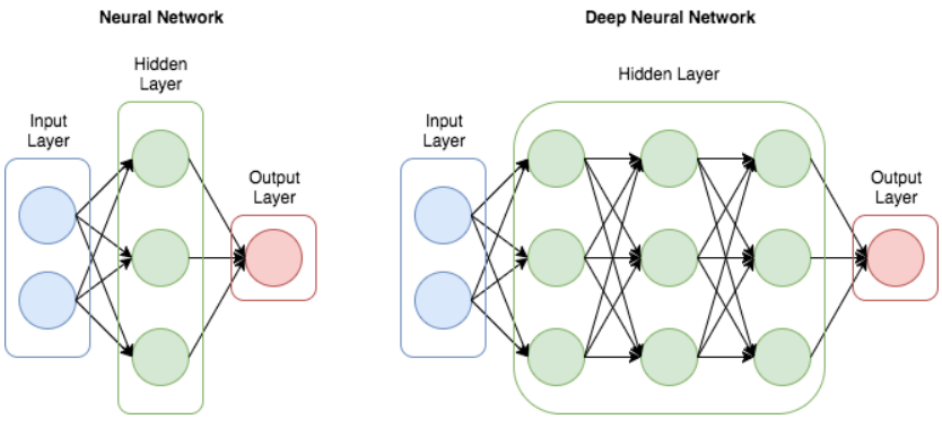

Figura 4 – Comparativo entre MLP e DNN (WEHBE, 2020).

Ambas contém a mesma estrutura, contudo as redes neurais densas ampliam a quantidade de camadas ocultas, conforme a necessidade de abstração

Assim como as MLPs serviram de base para o surgimento das redes neurais profundas, as DNNs também desempenharam um papel importante no avanço de arquiteturas voltadas para o campo da visão computacional. Embora inicialmente projetadas para lidar com dados vetoriais ou tabulares, sua capacidade de modelar dados de alta dimensionalidade permitiu que fossem aplicadas também a dados com estruturas especiais, como

imagens, desde que estas fossem vetorizadas, ou seja, convertidas em uma representação linear.

Esse histórico evidencia a flexibilidade das redes neurais artificiais: sua estrutura pode ser ajustada, expandida ou adaptada conforme a complexidade da tarefa, o tipo de dado ou o objetivo do modelo. Essa capacidade de adaptação é justamente o que permite a constante evolução das arquiteturas, seja para atender a demandas específicas ou para otimizar implementações anteriores. Neste contexto as DNNs foram fundamentais para pavimentar o caminho da utilização de redes profundas na visão computacional, contribuindo para o desenvolvimento de arquiteturas mais especializadas, como as CNNs e as ViTs.

2.4 Visão computacional

A visão computacional é um campo da inteligência artificial voltado para o processamento e interpretação de informações visuais provenientes do mundo real. Seu objetivo central é permitir que sistemas computacionais sejam capazes de adquirir, analisar e interpretar imagens ou vídeos, executando tarefas que tradicionalmente exigiriam a percepção visual humana.

Uma aplicação em visão computacional normalmente envolve um conjunto de etapas fundamentais, que possibilitam a conversão da informação visual em representações computacionais interpretáveis. Essas etapas incluem:

- **Aquisição da imagem:** corresponde à captura da imagem ou vídeo por meio de câmeras digitais, microscópios ou sensores, responsáveis por digitalizar a cena e convertê-la em uma matriz de *pixels* que possa ser processada por um sistema computacional;

- **Pré-processamento:** etapa em que são aplicadas técnicas de processamento digital de imagens (PDI), como filtros, redimensionamento, normalização e realce de contraste. O objetivo é melhorar a qualidade da imagem ou adaptá-la às exigências da análise;

- **Extração de características:** corresponde à identificação de padrões relevantes para a análise da imagem, como bordas, contornos, texturas ou formas geométricas. Atualmente, essa etapa é automatizada por meio de redes neurais especializadas, como as CNNs e os ViTs;

- **Análise, classificação e interpretação:** etapa em que são aplicadas técnicas de aprendizado de máquina, utilizando as características extraídas para classificar, segmentar ou reconhecer padrões visuais. Nesta fase, modelos treináveis como CNNs,

ViTs, Random Forests e SVMs são empregados para gerar predições e interpretações.

A evolução da visão computacional está fortemente relacionada ao desenvolvimento de redes neurais profundas, principalmente no que se refere à automatização no processo de extração de características. Tradicionalmente eram utilizadas técnicas manuais para a identificação de padrões visuais relevantes, baseada em conhecimento humano especializado. O avanço do aprendizado profundo, possibilitou a construção de modelos capazes de aprender automaticamente quais características são relevantes para determinada tarefa.

Uma das arquiteturas mais influentes no campo da visão computacional foi a CNN, que utiliza filtros convolucionais para extrair automaticamente representações cada vez mais abstratas e informativas da imagem, dispensando a intervenção humana. Recentemente, também foram exploradas arquiteturas mais flexíveis, como as ViTs. Embora sejam inspirados em modelos originalmente desenvolvidos para o processamento de linguagem natural, as ViTs aplicam mecanismos de atenção para analisar imagens como uma sequência de fragmentos, permitindo capturar relações mais globais entre as regiões da imagem.

O uso de arquiteturas como as CNNs e ViTs na visão computacional representa um marco na transição de modelos baseados em engenharia manual para modelos de aprendizado autônomos e escaláveis, melhorando significativamente o desempenho dos sistemas de análise visual. Para maiores conhecimentos acerca da integração de redes neurais profundas na área de visão computacional, consultar o survey A Comprehensive Survey of Deep Learning Approaches in Image Processing

2.4.1 Redes Neurais Convolucionais

As Redes Neurais Convolucionais (CNNs) constituem uma arquitetura de rede neural profunda projetada especificamente para lidar com dados com estrutura espacial, como imagens e vídeos. Seu principal diferencial está na capacidade de extrair automaticamente características visuais relevantes, eliminando a necessidade de pré-processamento manual. Para isso, são utilizados filtros convolucionais, que percorrem a imagem para destacar e extrair padrões como bordas, texturas e curvas.

Diferentemente das DNNs (que também são capazes de processar imagens contanto que elas sejam previamente vetorizadas), as CNNs preservam a estrutura bidimensional da imagem ao longo do processo de aprendizado. Isso permite capturar padrões locais com maior precisão, o que é especialmente importante no contexto visual, já que os *pixels* vizinhos geralmente apresentam alta correlação.

Além disso, as CNNs utilizam duas estratégias fundamentais: o compartilhamento de pesos e a organização hierárquica de camadas, conforme ilustrado na 5. O compartilhamento de pesos refere-se ao uso de filtros convolucionais, responsáveis por aplicar um

conjunto pequeno de pesos (em formato bidimensional) sobre toda a imagem, repetidamente. Isso reduz drasticamente o número de parâmetros uma vez que o mesmo conjunto de filtros é aplicado em diferentes regiões da imagem.

A hierarquia de camadas, por outro lado, refere-se à estrutura de múltiplas camadas convolucionais empilhadas (responsáveis pela extração de características). Isso permite que a rede aprenda representações progressivamente mais abstratas, indo de traços simples (como bordas) até estruturas complexas (como formas ou objetos), promovendo generalização e eficiência no reconhecimento de padrões visuais.

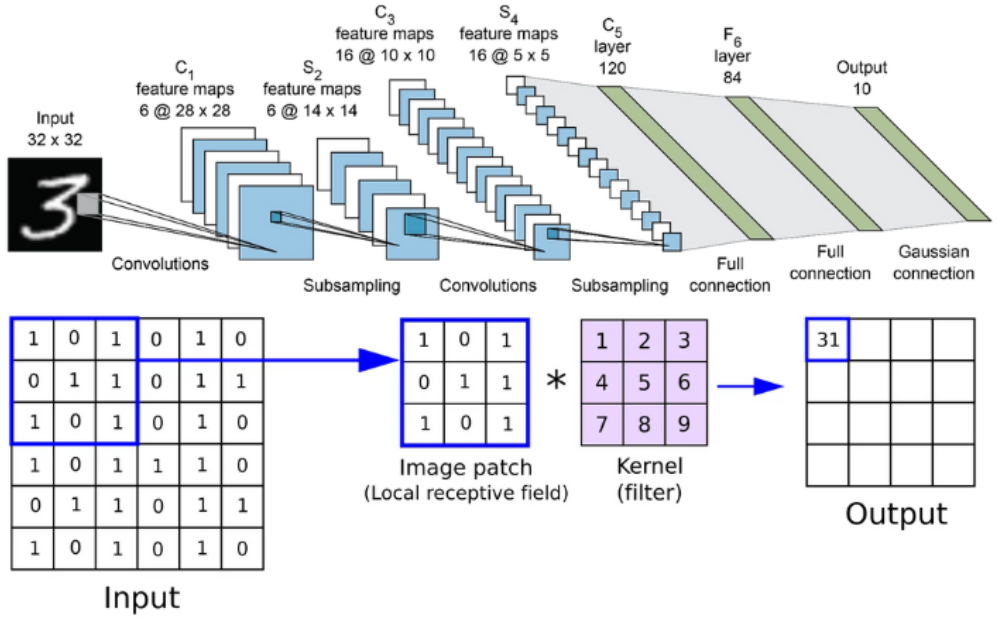

Figura 5 – Camadas convolucionais em uma CNN (JIANG, 2024).

O processo de convolução entre os filtros (kernels) e diferentes regiões da imagem permite capturar padrões locais. À medida que as camadas aumentam em profundidade, a rede aprende representações cada vez mais abstratas, fundamentais para tarefas como classificação ou reconhecimento de padrões visuais.

A arquitetura de uma CNN é composta por diferentes tipos de camadas, cada uma contendo uma função específica para o processo de extração e interpretação de características visuais:

- **Camada convolucional (*convolutional layer*):** é responsável por realizar a extração automática de características locais da imagem. Isso é feito por meio da aplicação de filtros (*kernels*), que são pequenas matrizes de pesos (geralmente de tamanho 3×3 ou 5×5).

  Durante o processo de convolução, cada filtro é 'deslizado' sobre a imagem de entrada, multiplicando seus valores com pequenas regiões da imagem em cada posição. O resultado dessas operações forma uma nova matriz chamada de mapa de carac-

terísticas (*feature map*), que registra a presença do padrão aprendido pelo filtro naquela imagem.

Como cada filtro é capaz de detectar um padrão visual específico (como bordas, texturas ou curvas) a aplicação de múltiplos filtros permite à rede representar diferentes aspectos da imagem em profundidade, construindo uma representação rica e progressiva da informação visual. Conforme também é ilustrado na imagem 5;

- **Camada de ativação:** é responsável por introduzir a não-linearidade aos resultados por meio de uma função de ativação. Sem este recurso, a rede seria uma combinação linear, o que limita a sua expressividade e impediria a modelagem de relações complexas entre os dados.

  Padrões lineares descrevem uma relação entre dados que podem ser separadas ou descritas por uma reta linear, um plano ou um hiperplano (dependendo da dimensionalidade). Em um contexto prático, nem sempre os dados são organizados de forma que tal separação seja possível, aqui se destaca a necessidade da não linearidade por meio de funções de ativação como *Sigmoid*, *Tahn* ou *ReLu*;

- **Camada de *pooling* ou subamostragem:** tem como principal objetivo reduzir as dimensões espaciais dos mapas de características gerados pelas camadas anteriores. Ao fazer isso, ela preserva apenas as informações mais relevantes de cada região, descartando os detalhes redundantes ou pouco significativos.

  Essa operação contribui diretamente para a redução do custo computacional da rede, além de reduzir a sensibilidade do modelo a pequenas variações de posição, como deslocamentos leves de um objeto na imagem. Com isso, as camadas seguintes conseguem trabalhar com representações mais abstratas, o que favorece a generalização e a identificação de padrões em diferentes contextos;

- **Camadas totalmente conectadas (fully connected layers):** nesta etapa, os mapas de características provenientes da camada de *pooling* passam por um processo denominado *flattening*, que consiste em sua conversão para um vetor unidimensional. Isso permite que o vetor seja utilizado como entrada para uma camada totalmente conectada, também conhecida como rede neural densa (DNN).

  A partir desse ponto na arquitetura da CNN, o funcionamento do sistema torna-se análogo ao de uma rede neural tradicional, a qual é responsável por aprender os padrões extraídos pelos filtros convolucionais e realizar a predição final, geralmente por meio de uma função de ativação do tipo *softmax*, quando se trata de tarefas de classificação.

As CNNs não seguem um padrão fixo: sua arquitetura pode ser ajustada em diversos aspectos, como o número de camadas, a profundidade, o tipo de filtros e a forma de

conexão entre os neurônios. Essa flexibilidade arquitetural é uma das maiores forças das CNNs, permitindo que sejam adaptadas a diferentes tipos de tarefas, desde classificações simples até análises visuais mais complexas. Arquiteturas consagradas como *AlexNet*, *VGGNet* e *Inception* mostram como diferentes estratégias estruturais podem impactar o desempenho e a escalabilidade dos modelos em visão computacional. Para maiores detalhes técnicos sobre as CNNs e as diferentes estratégias de aplicação, consultar o survey: A review of convolutional neural networks in computer vision.

Neste estudo, a arquitetura ResNet-50 foi adotada para a tarefa de classificação. Essa arquitetura foi desenvolvida com o propósito de superar limitações observadas em redes neurais muito profundas, como o desaparecimento do gradiente (*vanishing gradient*) e a degradação do desempenho à medida que novas camadas são adicionadas. Em redes convencionais, o aumento da profundidade tende a dificultar o processo de aprendizado, levando à piora na acurácia e à instabilidade durante o treinamento, mesmo quando não há overfitting.

A ResNet solucionou esses desafios ao introduzir o conceito de aprendizado residual, uma estratégia que permite o treinamento de redes mais profundas de forma estável e eficiente. Essa abordagem é implementada por meio dos chamados blocos residuais, que incorporam conexões de atalho (*skip connections*) entre as camadas da rede. Com essas conexões, o modelo deixa de aprender diretamente a transformação completa entre a entrada e a saída e passa a aprender apenas a diferença (ou resíduo) entre elas. Em seguida, a saída do bloco é obtida somando-se essa transformação à entrada original.

Essa estrutura aparentemente simples traz ganhos expressivos: o fluxo de informação e o gradiente do erro podem se propagar diretamente através das conexões de atalho, reduzindo significativamente o risco de desaparecimento do gradiente e tornando o processo de otimização mais eficiente. Além disso, o aprendizado residual permite que a rede preserve informações importantes de camadas anteriores e até 'ignore' determinadas transformações quando elas não forem necessárias, propagando a informação original sem modificações. Esse mecanismo contribui para a estabilidade do treinamento e para o ganho de desempenho em tarefas complexas, mesmo em redes com dezenas ou centenas de camadas.

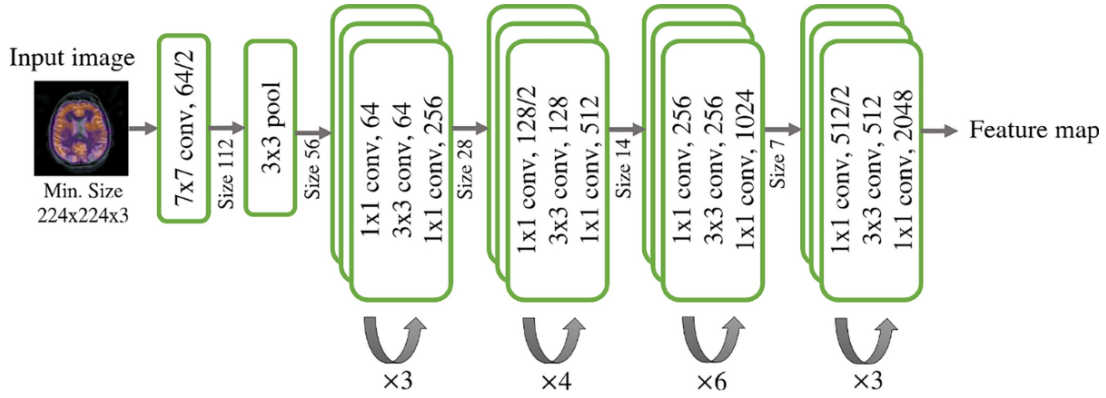

Figura 6 – Estrutura de construção da *ResNet50* (STACK, 2025).

Conforme ilustrado na Figura 6 sua construção é composta por 50 camadas treináveis organizadas em estágios denominados stem, bottleneck (conv2_x,conv3_x, conv4_x, conv5_x) e head. Tais camadas são descritas da na sequência:

- **Camada Inicial (*stem*):** responsável pelo pré-processamento inicial da imagem de entrada, com o objetivo de reduzir sua resolução espacial e extrair características visuais básicas, como bordas e contrastes. Essa camada é composta por uma operação convolucional simples que aplica 64 filtros com dimensões 7×7, varrendo a imagem a um passo de dois *pixels* (*stride* = 2), o que resulta em uma redução significativa da resolução da imagem — por exemplo, de 224×224×3 para 112×112×64;

  Após a convolução, são aplicadas rotineiramente três operações fundamentais: a normalização em lote (*Batch Normalization*), que estabiliza o treinamento ao padronizar as ativações; a função de ativação *ReLU*, que introduz não-linearidade ao modelo; e uma operação de *pooling*, que reduz ainda mais a dimensionalidade e aumenta a robustez da rede a pequenas variações locais;

- **Bloco de gargalo (*bottleneck*):** é considerado a unidade fundamental da arquitetura *ResNet*. Sua principal função é permitir a construção de redes profundas com baixo custo computacional, mantendo a eficiência ao longo da rede. Cada bloco é formado por três camadas convolucionais sequenciais, organizadas de maneira a transformar a entrada de forma eficiente e a permitir o uso de conexões residuais;

  A primeira camada do bloco aplica uma convolução 1Ö1 com o objetivo de reduzir a dimensionalidade da entrada e diminuir o custo computacional. Em seguida, a segunda camada realiza uma convolução 3Ö3, responsável pela extração de características locais da imagem, como bordas, texturas e formas específicas. Por fim, a terceira camada aplica novamente uma convolução 1Ö1, desta vez com o intuito de restaurar a profundidade original dos canais, permitindo que a estrutura de dados mantenha sua representatividade ao longo da rede. Após o processamento pelas três camadas, o resultado obtido é somado diretamente à entrada original do bloco, caracterizando o processo de conexão residual;

A arquitetura da *ResNet50* contém quatro grandes estágios estruturais, organizados a partir da repetição de blocos *bottleneck*. Cada um desses estágios é convencionalmente denominado de Conv2_x, Conv3_x, Conv4_x e Conv5_x, seguindo uma nomenclatura que representa a progressão das camadas ao longo da rede. Os blocos *bottleneck* são empilhados de forma diferente em cada um desses estágios, conforme a profundidade da rede aumenta;

- **Camada final (*Head*):** responsável por classificar os mapas de características extraídos na última camada convolucional. Inicialmente, aplica-se uma operação de *Average Pooling* global sobre os mapas tridimensionais, reduzindo cada canal a um único valor representativo. Em seguida, o resultado é transformado em um vetor unidimensional por meio do processo de *flattening*, possibilitando sua utilização como entrada para uma camada totalmente conectada. Nesta etapa, o funcionamento ocorre da mesma forma que uma DNN, cumprindo o papel classificatório.

De acordo com a topologia padrão descrita pela literatura, a configuração dos demais hiperparâmetros relevantes da arquitetura *ResNet-50* são:

- **Inicialização:** *He initialization* define a distribuição inicial dos pesos de forma que a variância das ativações de cada camada se mantém aproximadamente constante, mesmo em redes profundas. É especialmente projetado para operar com a *ReLU*, visto que essa função 'corta' metade das ativações (todas as negativas viram zero), então a variância precisa ser ajustada para compensar essa redução.)
- **Otimizador:** SGD com momentum 0.9
- **Taxa de aprendizado:** 0.1 com *batch_size* aproximado de 256)
- **Regularização:** Weight_decay aproximado de 10^{-4}

O impacto da ResNet na área de aprendizado profundo foi expressivo. Sua proposta de aprendizado residual tornou possível o treinamento de redes com mais de 150 camadas sem perda de desempenho, algo considerado inviável antes de sua introdução. Além disso, sua modularidade e estabilidade inspiraram diversas variações e aprimoramentos, como as arquiteturas ResNeXt, que introduz o conceito de cardinalidade (vários caminhos convolucionais paralelos dentro do bloco), Wide-ResNet, que aumenta a largura em vez da profundidade, e ResNetV2, que altera a ordem de normalização e ativação para maior estabilidade de treinamento.

2.4.2 Transformers e Vision Transformers

As *Vision Transformers* (ViTs) representam um avanço significativo no campo da visão computacional, sendo uma adaptação dos *Transformers*, uma arquitetura originalmente desenvolvida no campo do Processamento de Linguagem Natural (NLP) que

se baseia no mecanismo de autoatenção (*self-attention*), responsável por permitir que o modelo aprenda as relações globais entre os elementos de entrada. Diferentemente de abordagens convencionais, que processam as dependências de forma sequencial e local, os *Transformers* analisam todos os elementos simultaneamente, atribuindo diferentes níveis de importância a cada um. Essa característica possibilita a modelagem de dependências de longo alcance logo no inicio do processo, além de possibilitar um alto grau de paralelização durante o treinamento — aspectos fundamentais para o sucesso da arquitetura em tarefas como tradução automática, sumarização de textos e modelos de linguagem em larga escala.

As ViTs seguem princípios muito semelhantes aos *Transformers* empregados em NLP. Para compreender o funcionamento do mecanismo de autoatenção, é interessante observar um exemplo do contexto linguístico. Considerando a frase: 'O gato subiu na mesa para comer o peixe', é correto afirmar que nem todas as palavras possuem o mesmo peso para a interpretação do significado geral. Ao analisar o verbo 'comer', por exemplo, o modelo precisa concentrar maior atenção nas palavras 'gato' e 'peixe', pois são elas que estabelecem a relação semântica mais relevante. O mecanismo de atenção, portanto, tem como objetivo atribuir pesos diferentes a cada elemento da sequência, de acordo com sua importância contextual, produzindo os denominados scores de atenção — valores que indicam a influência relativa de cada palavra em relação às demais. Dessa forma, o modelo consegue representar com precisão as relações globais existentes na sequência analisada.

A primeira etapa do processo consiste em converter o texto não estruturado em um formato compreensível para o modelo computacional. Para isso, são aplicadas duas técnicas fundamentais: *tokenização* e *embeddings*. A *tokenização* corresponde à decomposição do texto em unidades menores, denominadas *tokens*, que podem ser palavras, subpalavras, símbolos ou até caracteres individuais. Essa etapa transforma o fluxo contínuo de texto em uma sequência discreta de elementos analisáveis. Em seguida, cada token é convertido em uma representação vetorial contínua, denominada embedding, os quais mapeiam elementos semelhantes para pontos próximos em um espaço vetorial multidimensional, permitindo que o modelo capture relações semânticas e sintáticas entre tais elementos. No exemplo anterior, os *tokens* 'gato' e 'peixe' teriam vetores próximos nesse espaço, pois compartilham semelhanças semânticas (ambos representam animais). Essas representações são aprendidas automaticamente durante o treinamento do modelo.

Adicionalmente, além do significado das palavras, o *Transformer* também precisa compreender a posição de cada *token* na sequência, já que, diferentemente das redes recorrentes, ele não processa as entradas em ordem temporal. Para isso, o modelo utiliza os *embeddings* posicionais (*positional embeddings*), vetores que codificam a posição de cada token na sequência. Assim, a entrada final para o modelo é obtida pela soma do *embedding* do *token* com seu respectivo *embedding* posicional — uma combinação que permite ao *Transformer* considerar simultaneamente o conteúdo e a posição dos elementos durante

o processamento nas camadas de autoatenção.

Após a etapa de formatação e codificação posicional, os vetores de entrada são submetidos ao mecanismo de autoatenção, responsável por modelar as relações internas entre os diferentes tokens da sequência. O processo tem início com a projeção de cada token em três espaços vetoriais distintos, obtidos por meio de transformações lineares aprendíveis. Tais projeções dão origem a três matrizes fundamentais: Query (Q), Key (K) e Value (V); Cada uma exerce um papel específico na construção das relações de atenção entre os elementos:

- **Vetor Q (Query)** → Representa a 'pergunta' que um token faz ao resto da sequência - 'o que eu preciso saber do contexto agora?';
- **Vetor K (Key)** → Funciona como uma 'chave' ou 'assinatura' associada a cada *token*, descrevendo o tipo de informação que ele pode oferecer aos demais - "o que eu tenho para oferecer a quem me perguntar?";
- **Vetor V (Value)** → Representa o conteúdo efetivo que será transmitido ou agregado quando um *token* recebe atenção, correspondendo à informação contextual que compõe a nova representação final.

De forma mais intuitiva, o mecanismo de autoatenção consiste em medir quanto cada *token* deve prestar atenção aos demais. Para isso, cada *query* é comparada com todas as *keys* da sequência, produzindo um conjunto de pesos que indicam o grau de relevância de cada *token* em relação ao outro. Esses pesos são, então, utilizados para calcular uma combinação ponderada dos *values*, gerando uma nova representação contextualizada para cada elemento de entrada. Formalmente, são usadas três matrizes de projeção W_Q, W_K, $W_V \in R^{dXd_k}$. Tais matrizes são parâmetros treináveis, ou seja, o modelo ajusta essas projeções durante o treinamento para que a atenção aprenda relações úteis para a tarefa. Com isso, as matrizes Q, K e V são definidas por: $Q = XW_Q$; $K = XW_K$; $V = XW_V$ (Sendo X o vetor de embeddings).

É importante destacar que as matrizes de projeção W_Q, W_K e W_V são necessárias por permitirem que o modelo aprenda subespaços de representação especializados para diferentes propósitos: buscar informações (Q), indexar elementos relevantes (K) e retornar o conteúdo contextualizado (V). Caso o próprio vetor de embedding fosse utilizado diretamente como query, key e value, o modelo ficaria restrito a um único espaço de representação, limitando sua capacidade de capturar relações complexas entre os tokens. As projeções lineares, ampliam a expressividade do modelo ao permitir que a atenção opere sobre perspectivas distintas do mesmo dado, como aspectos sintáticos, semânticos ou posicionais. De maneira mais didática, temos:

1. **Separação de responsabilidades** → cada matriz de projeção tem uma função específica dentro do cálculo de atenção:

- ❏ W_Q transforma cada x_i num vetor que expressa o que esse *token* está buscando compreender no contexto (sua "pergunta");
- ❏ W_K transforma cada x_j num vetor que expressa o que o *token* j oferece como informação disponível (sua "assinatura");
- ❏ W_V projeta o vetor de entrada para o espaço do conteúdo efetivo que será transmitido e agregado (sua "resposta").

2. **Mudança de base / subespaços:** as projeções criam subespaços vetoriais onde as relações de similaridade entre *queries* e *keys* tornam-se significativas para a tarefa. Por exemplo, um subespaço pode capturar padrões sintáticos (como dependências entre verbos e sujeitos), enquanto outro pode capturar relações semânticas (como associações entre conceitos relacionados). Essa separação facilita o aprendizado de múltiplas dimensões de significado dentro de uma mesma sequência.

Com a definição das matrizes Q e K, torna-se possível calcular as pontuações de atenção (*attention scores*), que quantificam o grau de compatibilidade (atenção) entre cada par de *tokens*. Matematicamente, o modelo compara cada vetor *query* Q_i com todos os vetores *key* K_j da sequência, por meio de um produto escalar, resultando em uma medida de similaridade entre Q_i e K_j:

$$Score(i, j) = Q_i \cdot K_j^T \tag{1}$$

O resultado dessa operação para todos os elementos gera uma matriz de pontuações de atenção, em que cada valor $Score(i, j)$ representa o quanto o *token* na posição i deve considerar o *token* na posição j ao atualizar seu próprio estado. Entretanto, esses valores ainda não são probabilidades; tratam-se apenas de medidas numéricas de similaridade, que podem variar amplamente em escala.

Para que possam ser interpretadas como pesos de atenção normalizados, aplica-se a função *Softmax* sobre cada linha dessa matriz, transformando as pontuações em valores positivos entre 0 e 1 cuja soma é igual a 1 (probabilidades). Isso garante que o modelo distribua sua atenção de maneira ponderada entre todos os *tokens*. Adicionalmente, antes da aplicação da *Softmax*, as pontuações são divididas por $\sqrt{d_k}$, onde d_k corresponde à dimensionalidade dos vetores *key*. Essa normalização tem a função de evitar que os produtos escalares cresçam excessivamente com o aumento da dimensionalidade, o que poderia tornar a *Softmax* sensível a pequenas variações numéricas e, consequentemente, gerar gradientes instáveis durante o treinamento. Matematicamente, temos:

$$Attention\,Weights(i, j) = Softmax(\frac{Q_i.K_j^T}{\sqrt{d_k}}) \tag{2}$$

De forma simplificada, podemos representá-la também como:

$$Attention\,Weights(i,j) = Softmax(\frac{Score(i,j)}{\sqrt{d_k}}) \tag{3}$$

Após essa etapa, o modelo sabe, para cada palavra da sequência, quais outros elementos são mais relevantes para sua interpretação no contexto global. Como exemplo, podemos considerar os seguintes pesos atribuídos aos *tokens* da frase 'O gato comeu':

- Atenção em "O": 0.05
- Atenção em "gato": 0.45
- Atenção em "comeu" (ela mesma): 0.70

Esses valores indicam que a palavra "comeu" concentra a maior parte de sua atenção em si mesma (caracterizando o princípio da autoatenção), mas também atribui uma atenção significativa à palavra 'gato' (0.45), reconhecendo que ela é o sujeito da ação expressa pelo verbo. Por outro lado, a palavra 'O' recebe um peso pequeno (0.05), o que indica que sua contribuição semântica para a compreensão de "comeu" é menos relevante.

Com os pesos calculados, o modelo realiza o passo final da autoatenção: a combinação ponderada dos vetores de valor (*Value*). Cada vetor V_j contém a informação contextual de um *token* da sequência, e o peso de atenção $Attention : Weights(i,j)$, responsável por definir quanto dessa informação deve ser incorporada ao token i. O resultado é obtido pela soma ponderada de todos os *Values*, dada pela seguinte expressão:

$$Output(i) = \sum_j Attention\,Weights(i,j)\ \times\ V_j \tag{4}$$

Assim, o vetor de saída $Output(i)$ representa uma versão contextualizada do *token* i. Ele não reflete apenas a informação original da palavra, mas também agrega, de maneira ponderada, partes das informações de todas as outras palavras que receberam atenção. No exemplo anterior, a nova representação de 'comeu' passa a carregar não apenas a ideia de uma ação, mas também a noção de quem a executa ("gato"), enriquecendo a semântica do *token*. Em contraste com arquiteturas sequenciais tradicionais, o *Transformer* constrói simultaneamente todas as relações de atenção, resultando em representações altamente expressivas e interconectadas.

É importante destacar que, em modelos reais, o desempenho é mais expressivo quando o sistema é capaz de examinar diferentes tipos de relações entre os elementos de entrada simultaneamente. No entanto, isso não ocorre de forma plena quando o mecanismo de autoatenção é aplicado uma única vez. Nesse caso, o modelo tende a capturar uma única perspectiva das relações entre *tokens*, ou seja, uma única métrica de similaridade em um único subespaço vetorial.

Neste contexto introduz-se o conceito de *multi-head attention*, cuja ideia central é calcular múltiplos mecanismos de atenção em paralelo, cada um operando em um subespaço

distinto das representações de entrada, e posteriormente combinar os resultados obtidos. De forma intuitiva, seria como observar uma mesma cena por diferentes lentes: uma lente realça as cores, outra enfatiza contornos e outra destaca texturas. Ao combinar as observações de todas essas lentes, obtém-se uma representação mais rica do que qualquer lente isolada poderia oferecer.

Matematicamente, *multi-head attention* divide a projeção total em h cabeças. Em vez de projetar cada embedding X de dimensão d diretamente nos vetores Q, K e V de dimensões d_k, o mecanismo cria h conjunto de matrizes $W_Q^{(i)}, W_K^{(i)}, W_V^{(i)}$ que projetam para subespaços de dimensão menor (normalmente $d_k = d/h$). Para cada cabeça i calcula-se a atenção, sendo:

$$head_i = Attention(Q^{(i)}, K^{(i)}, V^{(i)}) \tag{5}$$

Onde:

$$Q^{(i)} = XW_Q^{(i)}; K = XW_K^{(i)}; V = XW_V^{(i)} \tag{6}$$

Cada uma das saídas geradas por essas cabeças resulta em uma matriz de dimensão $n \times d_k$, que representa o número de *tokens* (n) e a dimensão de cada subespaço (d_k). Em seguida, todas as saídas são concatenadas para formar uma única matriz de dimensão $n \times (h \cdot d_k)$ e, por fim, projetadas novamente em um espaço de dimensão d por meio de uma matriz de projeção final $W_O \in R^{(hd_k)\times d}$, obtendo-se assim a saída consolidada da camada de atenção:

$$MultiHead(X) = Concat(head_1, \ldots, head_h)W_o. \tag{7}$$

A arquitetura dos *Transformers* é composta por blocos modulares empilhados, que combinam mecanismos de atenção com camadas *feedforward*, normalização e conexões residuais. Essa estrutura caracteriza o *encoder* do modelo e permite o aprendizado eficiente de dependências complexas entre os elementos da entrada. Cada camada do *encoder* realiza duas operações principais, executadas em sequência:

1. Uma subcamada de atenção (tipicamente a *multi-head self-attention*) que permite que cada posição da sequência acesse informações provenientes de todas as demais, capturando relações globais entre os elementos;

2. Uma subcamada *feedforward* aplicada após a atenção, que executa transformações ponto a ponto de forma independente para cada posição. Entre essas subcamadas, são inseridas conexões residuais (*skip connections*) e mecanismos de normalização (geralmente *Layer Normalization* ou *LayerNorm*).

A subcamada *feedforward* corresponde, na prática, a uma pequena rede neural densamente conectada, aplicada individualmente a cada *token*. Em geral, ela é composta por duas projeções lineares separadas por uma função de ativação não linear (como ReLU ou GELU). Essa etapa amplia a capacidade de modelagem local das representações: enquanto o mecanismo de atenção redistribui informações entre diferentes posições, a camada densa permite que cada posição refine e recombine suas próprias características de maneira não linear, enriquecendo a representação antes que ela seja propagada para a próxima camada.

As conexões residuais e a normalização são cruciais para viabilizar o empilhamento profundo de blocos. A conexão residual adiciona a entrada de uma subcamada à sua saída (De maneira similar à arquitetura *ResNet*), o que facilita o fluxo de gradiente durante o treinamento e permite que camadas extras não degradem o desempenho. Em seguida, aplica-se a normalização, a fim de estabilizar a distribuição das ativações, melhorar a convergência e favorecer a precisão do treinamento.

Na aplicação original, a arquitetura dos *Transformers* é composta por dois componentes principais: o *encoder* e o *decoder*. O *encoder* é responsável por processar a sequência de entrada, extraindo e representando suas características em um espaço vetorial de maior expressividade. Já o *decoder* utiliza essas representações para gerar uma sequência de saída, etapa essencial em tarefas como tradução automática ou geração de texto.

Nas ViTs, o processo segue a mesma filosofia dos *Transformers* originais, mas é adaptado ao domínio visual. Nesse caso, a entrada não é uma sequência de palavras, mas sim uma sequência de *patches*, que representam pequenas regiões extraídas da imagem, como exemplo: em uma imagem de dimensões 224×224 *pixels*, é comum dividir o conteúdo em blocos de 16×16 *pixels*, o que resulta em 196 *patches*. Cada um desses blocos é achatado (vetorizado) e projetado linearmente em um vetor de *embedding* de dimensão fixa.

São adicionados dois elementos fundamentais à sequência de embeddings dos patches: um token de classe (*class token*), que agrega a informação global da imagem e serve como representação final para tarefas de classificação, e os *embeddings* posicionais, responsáveis por fornecer à rede uma noção explícita de localização espacial (Análogo ao *positional encoding* das NLPs). A sequência resultante é então processada pelo *encoder* do *Transformer*, cujos blocos de atenção permitem que cada patch troque informações com todos os outros, capturando dependências espaciais de longo alcance. As redes *feedforward* subsequentes, por sua vez, refinam as representações locais de cada *patch*, complementando a informação contextual aprendida pela atenção.

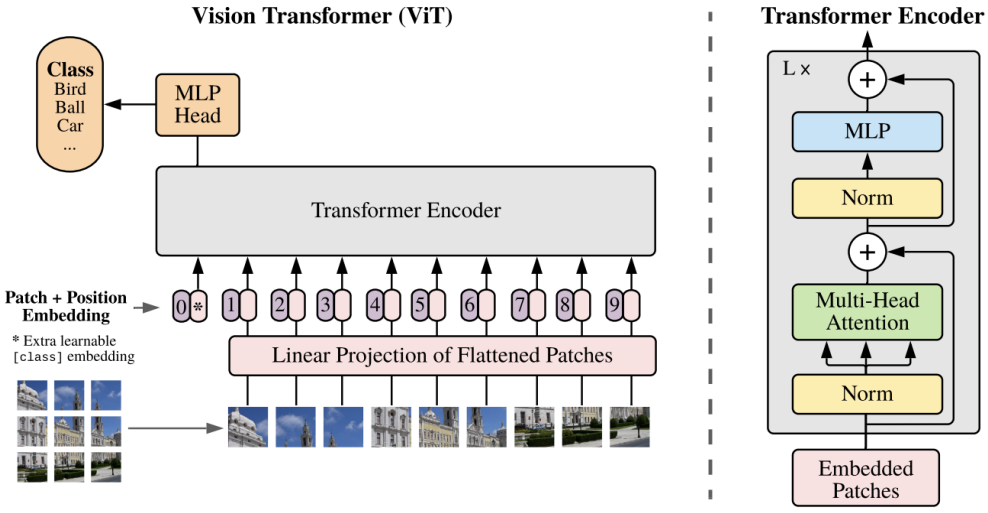

Figura 7 – Topologia da arquiteura Vision Transformer (ViT) (HUANG, 2023).

A Figura 7 ilustra de forma esquemática a topologia padrão da arquitetura *Vision Transformer* (ViT), conforme descrita na literatura. A seguir, apresenta-se uma descrição intuitiva de sua estrutura, bem como a configuração de seus principais componentes e hiperparâmetros:

1. ***Input*:**

   - ❏ A imagem de entrada possui dimensões $224 \times 224 \times 3$, correspondendo a imagens RGB de resolução padrão;
   - ❏ Cada imagem é dividida em *patches* não sobrepostos de tamanho 16×16, resultando em 196 *patches* distintos;
   - ❏ Cada *patch* é achatado (*flattened*) e projetado linearmente e formar o espaço de embeddings

2. ***Patch Embedding* e *Positional Encoding*:**

   - ❏ Cada *patch* é transformado por uma camada linear (projeção aprendível) que mapeia o vetor achatado de dimensão $16 \times 16 \times 3 = 768$ para um espaço de embedding de mesma dimensão ($d = 768$);
   - ❏ Um vetor adicional denominado *class token* [CLS] é concatenado ao início da sequência de *patches*, a fim de agregar a informação global da imagem;
   - ❏ São adicionados embeddings posicionais aprendíveis (*learnable positional embeddings*) aos vetores de entrada, preservando a informação espacial.

3. ***Transformer Encoder*:**

- ❏ O modelo padrão ViT é composto por 12 blocos de *Transformer Encoder*, cada um contendo duas subcamadas principais: *(i) Multi-Head Self-Attention* (MHSA) e *(ii) Feed-Forward Network* (FFN), ambas com conexões residuais e normalização de camada (*LayerNorm*);
- ❏ Cada bloco utiliza 12 cabeças de atenção, cada uma com dimensão de 64, totalizando $12 \times 64 = 768$ dimensões, correspondendo à dimensão do embedding;
- ❏ A subcamada *Feed-Forward* interna (MLP) é composta por duas camadas densas com ativação não linear GELU, sendo a primeira de dimensão $4d = 3072$ e a segunda de dimensão $d = 768$;
- ❏ É aplicado *dropout* de 0.1 nas conexões e ativações para reduzir o risco de *overfitting*;
- ❏ O modelo é otimizado com o algoritmo *AdamW*, utilizando *learning rate* inicial de 0.001 e *warmup* nas primeiras iterações, conforme recomendado para estabilizar o treinamento de arquiteturas baseadas em atenção.

4. ***Class Token* e *MLP Head*:**
   - ❏ Após o processamento pelos blocos do *encoder*, o vetor correspondente ao *class token* [CLS] é extraído como representação global da imagem;
   - ❏ Esse vetor é passado por uma camada *LayerNorm* e posteriormente por uma cabeça de classificação (*MLP Head*), composta por uma ou mais camadas densas, cuja saída é dimensionada conforme o número de classes da tarefa;
   - ❏ A ativação final geralmente é do tipo *Softmax*, produzindo uma distribuição de probabilidades sobre as classes.

É importante também reconhecer limitações e considerações deste tipo de arquitetura. O custo computacional do mecanismo de autoatenção cresce quadraticamente em relação ao número de *patches*, o que pode tornar o treinamento inviável para imagens de alta resolução. Além disso, modelos baseados em *Transformers* costumam demandar grandes volumes de dados para alcançar bom desempenho quando treinados do zero, exigindo estratégias adicionais de regularização, ajustes de taxa de aprendizado e inicialização criteriosa dos parâmetros para garantir estabilidade e convergência adequadas.

2.5 Transfer Learning

As redes neurais profundas são projetadas para aprender hierarquias de representações, em que as camadas iniciais capturam padrões visuais de baixa complexidade, como bordas e texturas, as camadas intermediárias identificam composições locais e as camadas finais abstraem conceitos de alto nível. Como muitos desses padrões de baixo e médio nível são

compartilhados entre diferentes domínios visuais, o seu reaproveitamento reduz a carga de aprendizado sobre o conjunto de dados alvo. Nesse contexto, insere-se o conceito de aprendizado por transferência (*Transfer Learning*).

O *Transfer Learning* refere-se ao conjunto de técnicas que visam aproveitar o conhecimento adquirido por um modelo durante o treinamento em uma tarefa-fonte para melhorar o desempenho em uma tarefa-alvo, geralmente mais específica ou com menor volume de dados. A ideia central consiste em reutilizar representações previamente aprendidas (como pesos, filtros e *embeddings*) provenientes de uma base de dados rica e diversa, a fim de acelerar o treinamento, reduzir a necessidade de rotulagem extensiva e aprimorar a capacidade de generalização do modelo em contextos de dados limitados.

No campo da visão computacional, o aprendizado por transferência tem se mostrado vantajoso em cenários nos quais o conjunto de dados é considerado pequeno (até 10.000 imagens) ou moderado (entre 10.000 e 100.000 imagens). Nesses casos, o esforço computacional é concentrado nas especifidades da atividade proposta, o que agiliza a convergência e evita redundâncias. No contexto deste estudo, por exemplo, a inicialização do modelo com pesos pré-treinados na base *ImageNet* — um conjunto massivo composto por aproximadamente 1,2 milhão de imagens distribuídas em 1.000 categorias — permite que o processo de treinamento concentre-se na extração de características específicas e relevantes para a detecção de dor em camundongos, evitando o custo de aprendizado redundante das estruturas visuais fundamentais.

Existem duas estratégias amplamente utilizadas para a implementação do *Transfer Learning*, baseadas no seguinte questionamento: "quanto do modelo original deve ser reaproveitado e quanto deve ser re-treinado?". São elas:

1. ***Feature Extraction* (ou *Linear Probing*)**: consiste em uma abordagem mais simples e conservadora do aprendizado por transferência, na qual o modelo pré-treinado é utilizado apenas como extrator de características, sem que seus pesos internos sejam modificados. Na prática, o modelo é 'congelado', ou seja, todas as suas camadas permanecem fixas, e apenas uma nova camada final (classificadora) é adicionada e treinada. Essa camada costuma ser uma rede densa (*fully connected*) com número de neurônios igual à quantidade de classes da nova tarefa.

   Como exemplo, pode-se considerar um modelo baseado na *ResNet-50* pré-treinada na base *ImageNet*. Nesse caso, a última camada que originalmente é responsável por classificar 1.000 categorias, é removida e substituída por uma nova camada com três saídas, adequando o modelo a uma tarefa de classificação para três classes distintas. Durante o treinamento, apenas os pesos da nova camada são atualizados, de modo que o modelo aprende apenas a interpretar as *features* (características) já extraídas pelas camadas anteriores, sem modificar o processo de extração em si.

   Essa estratégia apresenta como principais vantagens o baixo custo computacional

e o treinamento mais rápido, devido à menor quantidade de parâmetros ajustáveis. Entretanto, as *features* aprendidas na tarefa original podem não ser totalmente adequadas à nova aplicação, o que pode limitar a capacidade de adaptação do modelo ao novo domínio.

2. ***Fine-tuning***: trata-se de uma abordagem mais flexível, na qual parte (ou toda) a rede pré-treinada é re-treinada, permitindo o ajuste de seus pesos com base no novo conjunto de dados. A ideia é inicializar o treinamento a partir de um modelo já pré-treinado, mas, diferentemente da estratégia anterior, um subconjunto das camadas é descongelado (geralmente as últimas) para que seus pesos sejam refinados durante o processo de otimização. Dessa forma, o modelo preserva os conhecimentos gerais aprendidos (como bordas, texturas e formas), mas também passa a incorporar nuances específicas do novo domínio.

   O *fine-tuning* oferece uma adaptação mais profunda ao novo domínio, permitindo ao modelo refinar suas representações internas para capturar padrões particulares da tarefa. Entretanto, o treinamento se torna computacionalmente mais caro e há maior risco de *overfitting*, especialmente quando o conjunto de dados disponível é pequeno.

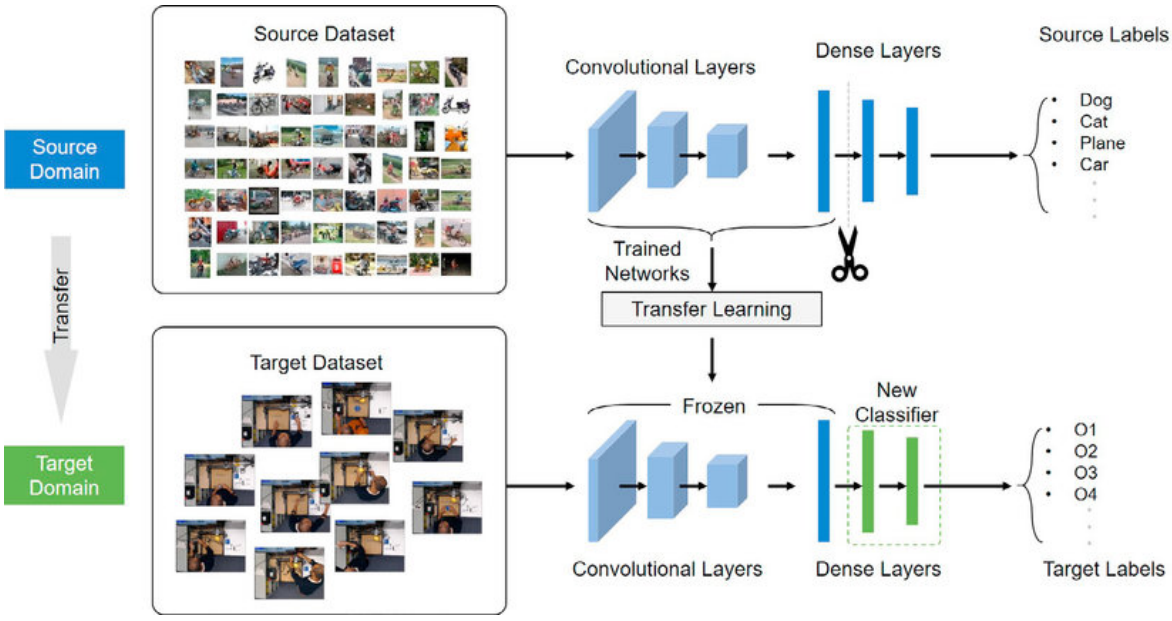

Figura 8 – Exemplo intuitivo do *transfer learning*, combinando o *Linear Probing* com o *Fine-tuning* (PRABHA, 2021).

Na prática, é comum combinar as duas estratégias descritas de forma sequencial. A Figura 8 demonstra a aplicação inicial do *Linear Probing* para treinar apenas a nova camada de classificação, garantindo estabilidade ao modelo. Em seguida, é feito o *Fine-tuning* progressivo, no qual algumas camadas superiores são gradualmente descongeladas, e o treinamento continua com uma taxa de aprendizado reduzida. Essa abordagem é

conhecida como *progressive fine-tuning* ou *two-stage transfer learning* e tende a produzir os melhores resultados, pois evita a instabilidade inicial de re-treinar toda a rede de uma só vez e promove uma adaptação mais controlada e eficiente ao novo domínio. Para maiores detalhes técnicos sobre a técnica de aprendizagem por transferência e as diferentes estratégias de aplicação, consultar o survey: FONTE

2.6 TensorFlow

Levando em consideração os objetivos deste projeto, uma ferramenta essencial para o desenvolvimento das tarefas relacionadas ao campo de inteligência artificial foi o *TensorFlow*, um *framework* de código aberto voltado à computação numérica e ao aprendizado de máquina, desenvolvido pelo *Google*. Essa ferramenta foi projetada para expressar e executar fluxos de dados numéricos de forma eficiente e escalável. O *TensorFlow* abrange tanto componentes de baixo nível, como operações tensorais, cálculo automático de gradientes e execução simbólica; quanto interfaces de alto nível, como a API *tf.keras*, que simplifica a construção e o treinamento de redes neurais.

A ideia central dessa ferramenta está na representação dos dados por meio de tensores (*arrays* multidimensionais) e na descrição das operações matemáticas como nós interconectados em um grafo computacional. Esse paradigma permite que os modelos sejam definidos de maneira declarativa (especificando 'o que calcular' em vez de 'como calcular') e executados de forma paralela e otimizada em dispositivos distintos como CPU e GPU, sem a necessidade de alterar sua estrutura lógica. De modo geral, os tensores estão presentes em praticamente todas as etapas do processo de aprendizado, desempenhando diferentes funções dentro do fluxo de dados do modelo:

- **Entradas:** os dados de entrada (imagens, textos, séries temporais) são convertidos em tensores para que possam ser processados pelas camadas subsequentes da rede;
- **Pesos (*weights*):** cada camada da rede possui tensores treináveis, que representam os parâmetros internos (pesos e vieses) ajustados durante o processo de aprendizado;
- **Ativações:** os resultados intermediários (saídas produzidas por cada camada) também são tensores, os quais passam por funções de ativação (como *ReLU* ou *Sigmoid*) responsáveis por introduzir não-linearidades no modelo;
- **Gradientes:** durante o treinamento, o *TensorFlow* calcula automaticamente os tensores de gradiente, ou seja, as derivadas parciais da função de perda em relação a cada parâmetro da rede, permitindo a retropropagação do erro.

Tensores podem ser definidos como estruturas matemáticas que generalizam os conceitos de escalar, vetor e matriz para dimensões superiores. Intuitivamente, cada tensor é

um *array* multidimensional que armazena valores numéricos (sejam eles, inteiros, reais ou complexos) e que pode ser processado por operações matemáticas otimizadas em hardware. É importante destacar que os tensores não alteram as propriedades fundamentais das representações originais, apenas as estendem para um formato mais geral e flexível. Cada tensor é descrito essencialmente por três características principais:

1. ***Rank* (ordem):** indica o número de dimensões ou eixos que compõem o tensor. Essa propriedade permite compreender os tensores como extensões naturais das estruturas matemáticas básicas:

   - ❏ Escalares (*rank-0 tensors*) → representam um único valor numérico;
   - ❏ Vetores (*rank-1 tensors*) → representam uma sequência unidimensional de valores;
   - ❏ Matrizes (*rank-2 tensors*) → são usadas, para representar lotes de dados tabulares bidimensionais ou transformações lineares entre camadas;
   - ❏ Matrizes (*rank-2 tensors*) → representam dados bidimensionais, frequentemente utilizados para armazenar dados tabulares ou realizar transformações lineares entre camadas;
   - ❏ Tensores de ordem superior ($rank \geq 3$) → são utilizados em domínios como visão computacional, representando imagens (3D — altura, largura e canais de cor), vídeos (4D — adicionando a dimensão temporal) e dados volumétricos (5D ou mais).

2. ***Shape* (forma):** define o tamanho e a organização dos tensores em relação às suas dimensões. Por exemplo, um tensor com $shape = (4, 3)$ representa uma matriz composta por quatro linhas e três colunas.

3. **Tipo de dado (*dtype*):** especifica o tipo dos elementos armazenados no tensor, como *float32*, *int64* ou outros tipos compatíveis com o hardware de execução.

Essas propriedades permitem que o *TensorFlow* (ou qualquer outra biblioteca voltada ao *deep learning*) interprete automaticamente como realizar operações entre tensores de tamanhos e tipos distintos, otimizando a alocação de memória e o paralelismo. Em sua estrutura interna, as operações sobre tensores são representadas como nós de um grafo computacional, enquanto os próprios tensores funcionam como arestas que transportam dados entre esses nós.

Como exemplo: supondo a operação de multiplicação matricial $Z = XW + b$, os tensores X, W e b representam, respectivamente, as entradas, os pesos e o viés da transformação, resultando em um novo tensor Z. Durante o treinamento, cada operação armazena informações sobre o caminho computacional percorrido, permitindo ao *TensorFlow* calcular automaticamente os gradientes necessários para a retropropagação do erro. Para uma

descrição mais aprofundada das propriedades e operações sobre tensores, recomenda-se a consulta ao trabalho de referência [INSERIR FONTE].

O *TensorFlow* disponibiliza um vasto conjunto de operações destinadas à manipulação de tensores, à execução de cálculos numéricos e ao gerenciamento de fluxos computacionais, abstraindo parte significativa da complexidade envolvida na implementação de modelos de aprendizado de máquina.Tais ferramentas tornam o *framework* adequado tanto para pesquisas experimentais quanto para aplicações em larga escala, desde a etapa de prototipagem até a implantação em produção. Dentre as principais funcionalidades, destacam-se:

- **Transformações estruturais:** englobam funções que permitem alterar a forma ou a organização dos tensores sem modificar seu conteúdo, possibilitando adaptações eficientes entre diferentes etapas de processamento. Entre as mais utilizadas estão: `reshape()`, `transpose()`, `concat()` e `stack()`;

- **Operações matemáticas:** reúnem funções destinadas à realização de cálculos numéricos elementares ou compostos entre tensores, como multiplicação matricial (`matmul()`), soma (`add()`), subtração (`subtract()`), cálculo de médias (`reduce_mean()`), entre outras;

- **Indexação e *slicing*:** permitem acessar ou manipular regiões específicas de um tensor, de maneira análoga ao que ocorre com *arrays* na biblioteca *NumPy*, favorecendo o controle detalhado sobre subconjuntos de dados durante o processamento;

- **Broadcasting:** mecanismo que viabiliza a execução de operações entre tensores de dimensões distintas, ajustando automaticamente as dimensões de menor ordem para compatibilidade. Esse recurso é amplamente empregado em operações vetoriais e matriciais;

- **API para modelagem:** integra a API *Keras*, que fornece uma interface de alto nível voltada à definição, compilação e treinamento de modelos de aprendizado profundo. O *Keras* padroniza a criação de camadas, funções de perda, métricas e ciclos de treinamento, além de oferecer métodos prontos para uso, como `Model.fit()` (treinamento), `Model.evaluate()` (avaliação de desempenho) e `Model.predict()` (inferência sobre novos dados), com suporte a *callbacks* e registros de execução;

- **Otimizadores, funções de perda e métricas:** integração de algoritmos de otimização, incluindo *SGD*, *Adam*, *AdamW* e *RMSProp*, além de funções de perda clássicas, como *cross-entropy*, *mean squared error (MSE)* e *Huber loss*. Adicionalmente, o framework oferece métricas de desempenho como *accuracy*, *precision*, *recall* e *AUC*, fundamentais para o acompanhamento da evolução do treinamento.

- **Distribuição e escalabilidade:** disposição de ferramentas para o treinamento distribuído, que realizam sincronização de gradientes, divisão de lotes de dados e gerenciamento de recursos computacionais, permitindo que modelos sejam treinados de forma eficiente em múltiplas GPUs, TPUs ou máquinas distintas, sem necessidade de alterações significativas na API;

- **Implantação e produção:** oferece suporte nativo à exportação e execução de modelos em diferentes ambientes, por meio de ferramentas como *SavedModel* (formato padrão de exportação, preservando pesos e assinaturas de entrada/saída), *TensorFlow Lite* (otimização para dispositivos móveis e embarcados) e *TensorFlow.js* (execução direta em navegadores web);

- **Reutilização de conhecimento e monitoramento:** o *Keras* também fornece acesso simplificado a modelos e pesos pré-treinados, viabilizando o uso de técnicas de *transfer learning*. Adicionalmente, o *TensorBoard* pode ser integrado ao processo de treinamento por meio de *callbacks* e registros automáticos de *logs*, permitindo a visualização de métricas, histogramas, imagens e embeddings em tempo real.

Em suma, o *TensorFlow* configura-se como uma plataforma completa voltada à pesquisa e à aplicação prática em aprendizado de máquina. Seu ecossistema abrange desde operações fundamentais sobre tensores até recursos avançados de treinamento distribuído, monitoramento e implantação de modelos em produção. Para maiores detalhes técnicos acerca de sua estrutura e funcionalidades, recomenda-se a consulta ao survey: FONTE.

2.7 Avaliação dos resultados em *Machine Learning*

Os resultados obtidos no processo de experimentação científica exigem um processo de avaliação sólido e sistemático, de modo a verificar o comportamento e a eficácia dos modelos desenvolvidos. A aplicação de métricas e ferramentas de avaliação permitem mensurar o desempenho e a capacidade de generalização do modelo, além de reduzir o risco de interpretações imprecisas e orientar a tomada de decisões fundamentadas.

Dentre as ferramentas mais utilizadas para análise de desempenho em tarefas de classificação, destaca-se a Matriz de Confusão, uma estrutura tabular que compara as previsões realizadas pelo modelo com os valores reais das classes. Essa matriz fornece uma visão detalhada dos acertos e erros cometidos, sendo amplamente empregada para avaliar a confiabilidade e a robustez de classificadores supervisionados. A matriz é composta por quatro elementos fundamentais, que representam as possíveis combinações entre os rótulos reais e as previsões realizadas pelo modelo:

- **Verdadeiros positivos (*True positive* - TP)**: casos rotulados como verdadeiros, que foram corretamente previstos;

- ❑ **Verdadeiros negativos (*True Negatives* - TN)**: casos rotulados como negativos que corretamente previstos;
- ❑ **Falsos positivos (*False Positive* - FP)**: casos negativos classificados como positivos (indica um alarme falso);
- ❑ **Falsos negativos (*False Netgative* - FN)**: casos positivos classificados como negativos (indica uma falha de detecção do modelo).

A construção da matriz de confusão segue o princípio de que as **linhas** representam as classes reais e as **colunas** representam as classes previstas. Dessa forma, cada célula (i, j) contém a quantidade de amostras cuja classe real é i e que foram classificadas como pertencentes à classe j. Essa disposição permite identificar padrões de erro e compreender quais classes apresentam maior confusão entre si. Como exemplo, temos:

Tabela 1 – Matriz de confusão ilustrativa para classificação binária

		Valor Predito	
		Sim	**Não**
Real	**Sim**	**Verdadeiro Positivo (TP)**	**Falso Negativo (FN)**
	Não	**Falso Positivo (FP)**	**Verdadeiro Negativo (TN)**

A partir destes valores, é possível definir as métricas básicas de avaliação (levando em consideração o problema de classificação binária). São elas:

1. **Acurácia (*Accuracy*)**: refere-se à proporção de predições corretas feitas pelo modelo. É uma métrica intuitiva, porém enganosa em classes desbalanceadas.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}; \tag{8}$$

2. **Precisão (*Precision*)**: é a medida do quão confiáveis são as predições positivas do modelo, portanto, a alta precisão sugere a existência de poucos falsos positivos (alarme falso). Essa métrica responde ao seguinte questionamento: "dentre as amostras previstas como positivas, qual fração é realmente positiva?".

$$Precision = \frac{TP}{TP + FP}; \tag{9}$$

3. **Revocação / Sensibilidade / *Recall***: refere-se à proporção de positivos reais que foram corretamente detectados, portanto, um alto *recall* sugere a existência de poucos falsos negativos (falha de detecção). Essa métrica responde ao seguinte questionamento: "entre todos os positivos reais, quantos foram detectados?".

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

4. ***F1-score***: métrica de avaliação que combina precisão e *recall* em um único valor, representando um equilíbrio entre as duas métricas. É calculada como a média harmônica de precisão e *recall*, sendo mais útil do que a acurácia em conjuntos de dados desequilibrados.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}; \tag{11}$$

5. **Especificidade / *True Negative Rate* (TRN)** : refere-se à capacidade de um modelo identificar corretamente instâncias que pertencem à classe negativa. Essa métrica é crucial quando o custo de falsos positivos é alto, como na detecção de fraude ou na triagem de doenças.

$$TRN = \frac{TN}{TN + FP}; \tag{12}$$

6. **AUC-ROC e Curva ROC**: Curva ROC é um gráfico que exibe o desempenho de um modelo de classificação binária, plotando a Taxa de Verdadeiros Positivos (Recall) contra a Taxa de Falsos Positivos (definido por $1 - TRN(Especificidade)$) em diferentes limites de decisão. A AUC (Área sob a Curva ROC) é um valor único entre 0 e 1 que resume a performance geral do modelo, representando a probabilidade de que o modelo classifique corretamente uma instância positiva aleatória à frente de uma negativa aleatória.

É importante destacar que em tarefas multiclasses, existem várias matrizes de confusão, uma para cada classe (considera-se cada classe como 'positiva' e todas as demais como 'negativas'). Dessa forma, as métricas descritas previamente são calculadas em um esquema '*onde-vs-rest*' em que uma classe é confrontada com as demais. Os resultados para cada são agregados posteriormente em uma única métrica global. Neste contexto surgem as três estratégias de agregação

1. ***Macro-average:*** é feita a média aritmética das métricas por classe, dessa forma, todas as classes são tratadas igualmente independentemente de quantos exemplos ela possua. É a abordagem ideal quando se busca avaliar o desempenho balanceado do modelo entre todas as classes, ou seja, garantir o mesmo grau de contribuição entre classes pequenas e grandes.

$$Precision_{macro} = \frac{1}{N} \cdot \sum_{i} Precision_i \tag{13}$$

O exemplo acima também é válido para as demais métricas como Recall e F1-Score;

2. ***Micro-average:*** trata todas as predições igualmente, somando os valores globais de verdadeiros positivos (TP), falsos positivos (FP) e falsos negativos (FN) de todas as classes antes de calcular as métricas.Dessa forma, as classes com mais exemplos tem maior influência no resultado final.

$$Precision_{micro} = \frac{\sum_i TP_i}{\sum_i (TP_i + FN_i)} \tag{14}$$

O exemplo acima também é válido para as demais métricas como Recall e F1-Score;

3. ***Weighted average:*** calcula as métricas individuais por classe, mas faz a média final ponderando cada classe pela quantidade de amostras reais que ela contém (n_i). Trata-se de um meio-termo entre as abordagens anteriores, considerando o peso relativo de cada classe (como na *micro*), mas mantendo o cálculo por classe antes da agregação (como na *macro*). Dessa forma, classes maiores tem maior influência sobre o resultado, mas sem eliminar totalmente a análise das menores.

$$Precision_{weighted} = \frac{\sum_i (n_i \times Precision_i)}{\sum_i n_i} \tag{15}$$

O exemplo acima também é válido para as demais métricas como Recall e F1-Score.

2.7.1 TensorBoard para a avaliação de resultados

Neste estudo, o *TensorBoard* foi adotado com o propósito de facilitar a análise e o acompanhamento dos resultados ao longo do processo de aprendizado, integrando-se à etapa de avaliação dos modelos. Conceitualmente, o *TensorBoard* consiste em uma ferramenta de visualização e monitoramento desenvolvida de forma integrada ao *TensorFlow*, permitindo inspecionar de maneira interativa o comportamento de modelos de aprendizado de máquina durante o treinamento.

Durante a execução do treinamento, a ferramenta utiliza um objeto do tipo *callback* (geralmente `tf.keras.callbacks.TensorBoard`) para registrar informações relevantes sobre a execução do modelo. Tais registros são realizados periodicamente ao final de cada época ou ao término do treinamento, e incluem métricas de desempenho, tempo de execução, estrutura do grafo computacional, além de histogramas dos pesos e gradientes. Tais informações são gravadas em arquivos locais, geralmente organizados em um diretório denominado `logs/`.

Vale ressaltar que, em *Python*, uma função de *callback* é uma função passada como argumento para outra função, sendo executada em um ponto específico da execução, geralmente após a conclusão de uma tarefa. Esse mecanismo possibilita a personalização do comportamento de processos, permitindo que ações complementares sejam realizadas automaticamente conforme o andamento do programa.

As informações coletadas são posteriormente interpretadas pelo *TensorBoard*, que pode ser inicializado via terminal por meio do comando `tensorboard -logdir=./logs`. Uma vez iniciado, o painel torna-se acessível por meio de um navegador web (por padrão, em *http://localhost:6006*), exibindo um conjunto de abas interativas que permitem a análise visual e detalhada dos resultados obtidos durante o treinamento.

O *TensorBoard* é modular e organiza suas funcionalidades em diferentes abas, cada uma voltada a um tipo específico de análise. Entre as principais, destacam-se as seções de escalares, grafos e distribuições, as quais se fazem presentes em maioria das aplicações de aprendizado de máquina:

1. ***Scalars* (Escalares):** apresenta gráficos de métricas quantitativas registradas durante o treinamento. Dentre elas destacam-se a função de perda (nos conjuntos de treino e validação), acurácia, taxa de aprendizado e demais métricas personalizadas como *F1-Score*, precisão e revocação. A análise destas métricas são essenciais para identificar *overfitting*, monitorar a convergência e comparar diferentes execuções do mesmo modelo;

2. ***Graphs* (Grafos Computacionais):** exibe a estrutura do modelo em forma de grafo, mostrando as conexões entre as camadas e os fluxos de dados (tensores) entre elas. Tal análise permite Verificar se o modelo foi construído corretamente e se existem camadas desconectadas ou redundantes;

3. **Distribuições e Histogramas:** exibe a distribuição dos pesos e gradientes das camadas ao longo do tempo de treinamento. Tal análise permite verificar se há explosão ou desaparecimento de gradientes e como se comporta o grau de saturação dos pesos (se estão ficando muito altos ou muito baixos);

É importante destacar que o *TensorBoard* também dispõe de funcionalidades voltadas a análises mais específicas, como a visualização de amostras de imagens pertencentes aos conjuntos de treino, validação ou teste ao longo do processo de treinamento. É uma funcionalidade é útil em modelos de visão computacional, uma vez que permite verificar se as imagens estão sendo carregadas e pré-processadas corretamente, além de possibilitar a inspeção visual de mapas de ativação, *heatmaps* e mecanismos de atenção (quando exportados como imagens). Para informações mais detalhadas sobre o uso do *TensorBoard* e suas ferramentas complementares, recomenda-se a consulta ao survey: FONTE.

Essa ferramenta não apenas simplifica o monitoramento técnico, como também possui relevância metodológica e científica. Suas funcionalidades favorecem a análise comparativa entre diferentes arquiteturas e configurações de hiperparâmetros, melhoram a interpretação dos resultados e auxiliam na detecção de possíveis falhas ou inconsistências durante o treinamento. Tais benefícios agregam maior confiabilidade às conclusões obtidas, eviden-

ciando que os resultados foram derivados de um processo de monitoramento sistemático, controlado e verificável.

Capítulo 3

Metodologia de desenvolvimento e Pesquisa

Neste capítulo, são descritas e formalizadas as estratégias metodológicas adotadas para o desenvolvimento desta pesquisa. Evidenciam-se os procedimentos, técnicas e recursos utilizados em cada etapa do projeto, que incluem: aquisição e preparação da base de dados, estudo exploratório de classificadores disponíveis, pré-processamento das informações coletadas, construção e treinamento dos modelos selecionados, bem como a posterior avaliação dos resultados obtidos.

O objetivo é proporcionar uma compreensão clara e detalhada das abordagens empregadas, evidenciando a coerência entre os métodos utilizados e os objetivos do trabalho, além de oferecer transparência e reprodutibilidade ao processo.

3.1 Aquisição e preparação da base de dados

Este estudo adota a *Mouse Grimace Scale (MGS)* como referência para a identificação de características faciais associadas à presença e intensidade da dor em camundongos. Com base nisso, a base de dados deve ser composta por imagens que evidenciem a face dos animais, com ênfase em regiões expressivas como orelhas, olhos, focinho e boca.

Para a aquisição das imagens, os camundongos foram mantidos em mini-isoladores destampados, organizados sequencialmente sobre uma bancada iluminada. A captura dos vídeos foi realizada com câmeras de celulares (modelo *iPhone* 13) configuradas para gravação em resolução 4K a 60 fps. Cada sessão consistiu em 15 gravações de dois minutos, com intervalos de um minuto entre elas. Posteriormente, *frames* representativos desses vídeos foram extraídos manualmente por um especialista, que também realizou a rotulagem das imagens, gerando o *ground truth* necessário para o treinamento supervisionado dos modelos.

Parte dos animais foram submetidos à indução de dor por meio da injeção subcutânea de 20μL de formalina a 5% na região plantar da pata traseira direita, conforme

o protocolo descrito por Langford et al. (2010). Essa indução provoca uma resposta bifásica: uma fase aguda (0 a 15 minutos) e uma fase inflamatória ou tônica (15 a 30 minutos). Imediatamente após a aplicação da formalina, os animais foram filmados segundo o mesmo protocolo descrito anteriormente, com o objetivo de registrar expressões faciais compatíveis com o processo.

Trinta minutos após a indução, os animais deste grupo receberam analgesia com sulfato de morfina na dose de 5 a 10 mg/kg, por via subcutânea. Quatro horas após o procedimento, foi administrado cloridrato de tramadol (50 mg/kg, via intraperitoneal), repetido também nas 24 e 48 horas subsequentes. Paralelamente, para controle da resposta inflamatória, os animais receberam meloxicam (3 mg/kg, subcutâneo) nas mesmas janelas temporais: 6, 24 e 48 horas após a aplicação da formalina.

Durante o período de 72 horas de observação, os animais foram gravados sempre uma hora antes e uma hora após a administração de cada dose analgésica. Essa abordagem permite avaliar se o protocolo foi eficaz em atenuar as expressões faciais de dor, bem como distinguir entre animais saudáveis, animais com dor ativa e animais sob efeito de tratamento analgésico. Ao término do período experimental, todos os animais foram submetidos à eutanásia humanitária por meio de anestesia (associação de cetamina e xilazina, via intraperitoneal), seguida de deslocamento cervical e descarte adequado, conforme as normas institucionais de bem-estar animal.

3.2 Estudo exploratório dos classificadores

A definição do modelo classificador é crucial para a identificação e rotulação automática das expressões faciais associadas à presença e intensidade de dor em camundongos, tornando-se o principal componente deste estudo. Com isso, a escolha da arquitetura de rede considerou aspectos de desempenho, capacidade de generalização e adequação ao contexto da visão computacional, buscando a opção mais adequada para alcançar os objetivos propostos.

Foi realizada uma revisão bibliográfica de caráter exploratório sobre arquiteturas de redes neurais aplicadas a tarefas de classificação de imagens, com o objetivo de identificar os modelos mais relevantes e consolidados na literatura recente. Entre as arquiteturas convolucionais analisadas, destacaram-se a *AlexNet*, reconhecida por popularizar o uso do aprendizado profundo em visão computacional; a *InceptionNet*, caracterizada pelo uso de múltiplos filtros convolucionais em paralelo para capturar padrões em diferentes escalas; e a *MobileNet*, projetada para aplicações em dispositivos com recursos limitados, utilizando convoluções separáveis (*depthwise separable convolutions*) para reduzir o custo computacional.

Dentre as opções analisadas, a *ResNet* se apresentou como uma forte candidata para este estudo em virtude de sua capacidade de aprendizado residual, que permite o trei-

namento de redes profundas de forma estável e eficiente, bem como à sua consolidação no campo de visão computacional (sendo bem documentada pela literatura). Além disso, a *ResNet* apresenta um excelente equilíbrio entre profundidade e desempenho computacional, sendo amplamente utilizada em tarefas de detecção de objetos, reconhecimento facial e análise de expressões. Tais características a tornam particularmente adequada ao problema investigado neste trabalho, que demanda precisão na identificação de padrões visuais sutis presentes nas faces dos camundongos.

A revisão exploratória também identificou arquiteturas baseadas em *Transformers*, aplicadas ao campo da visão computacional. Originalmente desenvolvidos para o processamento de linguagem natural (NLP), os *Transformers* introduziram uma abordagem inovadora que substitui o uso de convoluções por um mecanismo denominado atenção (*attention*). O principal diferencial está no fato de que esses modelos avaliam simultaneamente as inter-relações entre todos os elementos de entrada por meio do mecanismo de autoatenção (*self-attention*), permitindo a captura de dependências globais desde as primeiras camadas.

A transposição desse paradigma para o domínio visual resultou nos *Vision Transformers* (ViT), que tratam imagens como sequências de pequenos blocos (*patches*). Ao contrário das CNNs, os ViTs não incorporam convoluções locais para a extração de características, mas exploram a atenção para modelar relações tanto locais quanto globais entre regiões da imagem. Essa característica é se mostra relevante no contexto deste projeto, em que a interpretação de expressões faciais sutis depende de correlações distribuídas entre diferentes regiões faciais, como olhos, orelhas e focinho. Adicionalmente, a arquitetura dos ViTs apresentam alta compatibilidade com estratégias de pré-treinamento em larga escala e *transfer learning*.

Após a análise, optou-se pela utilização das arquiteturas *ResNet*, representando a abordagem convolucional tradicional, e o *Vision Transformer* (ViT), correspondente à abordagem baseada em *Transformers*. Essa escolha visa aplicar, em paralelo, uma metodologia consolidada no campo da visão computacional e uma estratégia recente e inovadora, de modo a comparar seus desempenhos na tarefa de classificação das expressões faciais dos camundongos. Com isso, a expectativa é alcançar o melhor resultado possível, bem como realizar uma análise comparativa entre estes paradigmas, fornecendo meios para compreender as tendências futuras na área de visão computacional.

3.3 Tratamento dos dados coletados

O conjunto de vídeos coletados dos camundongos foi processado por meio de um software desenvolvido especificamente para este experimento. O objetivo da aplicação é permitir a reprodução dos vídeos gravados, oferecendo ao usuário ferramentas para seleção, extração e organização automatizada dos *frames*, facilitando a construção do

ground truth de forma mais rápida e padronizada.

O software foi inteiramente desenvolvido na linguagem *Python* (versão 3.11), utilizando o ambiente de desenvolvimento *PyCharm Community Edition*. A interface gráfica foi construída com o uso da biblioteca PyQt5, visando proporcionar uma experiência simples e amigável ao usuário. Para viabilizar a reprodução e o controle de vídeos em diferentes formatos, a biblioteca VLC foi integrada à aplicação. Como ilustrado na Figura 9, foram incorporadas ferramentas específicas para manipulação precisa da reprodução, permitindo ao usuário selecionar e extrair os quadros desejados com maior exatidão.

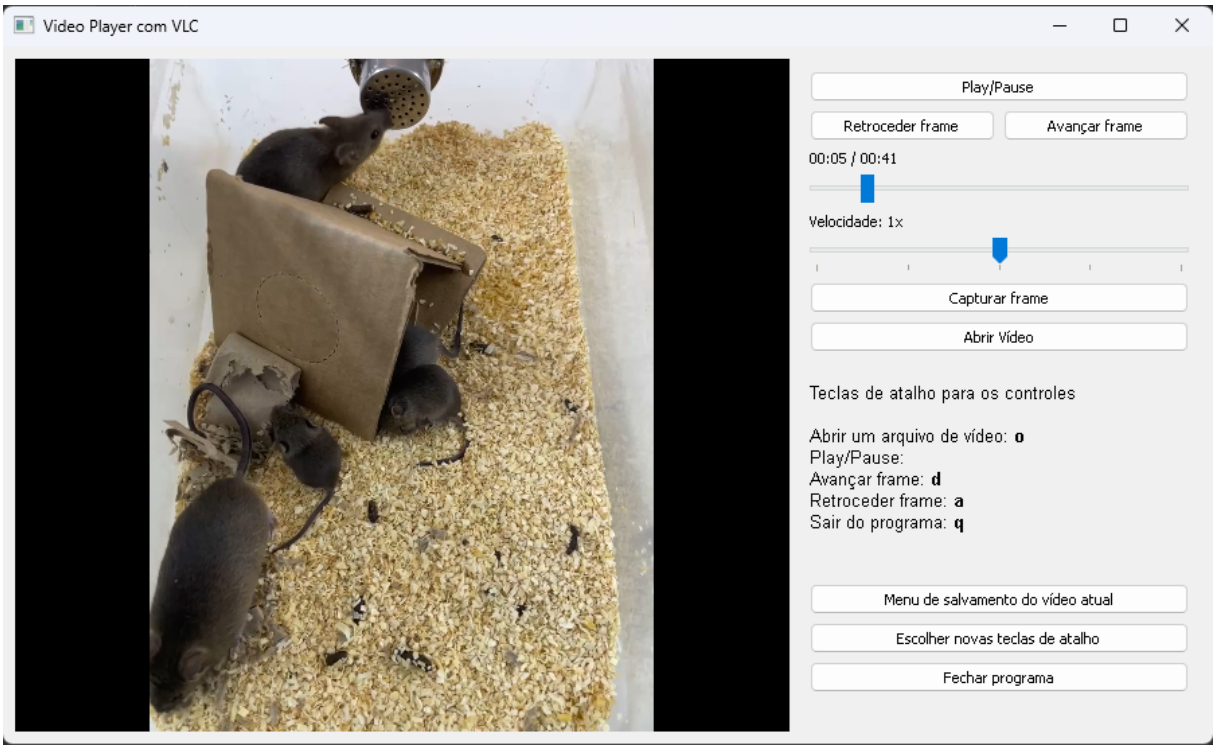

Figura 9 – Interface inicial do software desenvolvido para a geração da base de dados. Fonte: elaboração própria.

Dentre as funcionalidades implementadas, destaca-se o sub-menu dedicado à extração de imagens. Essa ferramenta permite que múltiplas seções de um mesmo *frame* sejam salvas separadamente, o que agiliza a construção da base de dados e permite especificar a área facial do animal, especialmente em casos onde o vídeo apresenta mais de um camundongo simultaneamente. O sub-menu pode ser acessado por meio do botão rotulado como “Capturar *Frame*”, o qual pausa a reprodução do vídeo antes de abrir uma nova janela. Cabe ao usuário identificar e selecionar manualmente as regiões de interesse mais relevantes para o estudo.

Conforme ilustrado na Figura 10, o sub-menu exibe o *frame* correspondente ao instante exato em que a funcionalidade foi acionada e apresenta uma série de botões para definir o rótulo da região selecionada. Nessa janela, o usuário pode livremente selecionar uma área da imagem (ajustada automaticamente a uma forma quadrada) e atribuí-la a uma das quatro categorias disponíveis: ’Indolor’, ’Pouca dor’, ’Muita dor’ ou ’Incerto’.

Ao acionar os botões de salvamento, o sistema cria uma estrutura de diretórios correspondentes a cada categoria na pasta raiz do programa, organizando automaticamente as imagens conforme o rótulo definido. Vale destacar que o algoritmo implementa um mecanismo de verificação para evitar que uma mesma seleção seja salva em mais de uma categoria. Sempre que uma nova seleção é armazenada, os demais diretórios são verificados para a detecção e remoção de duplicatas, garantindo a consistência da base de dados.

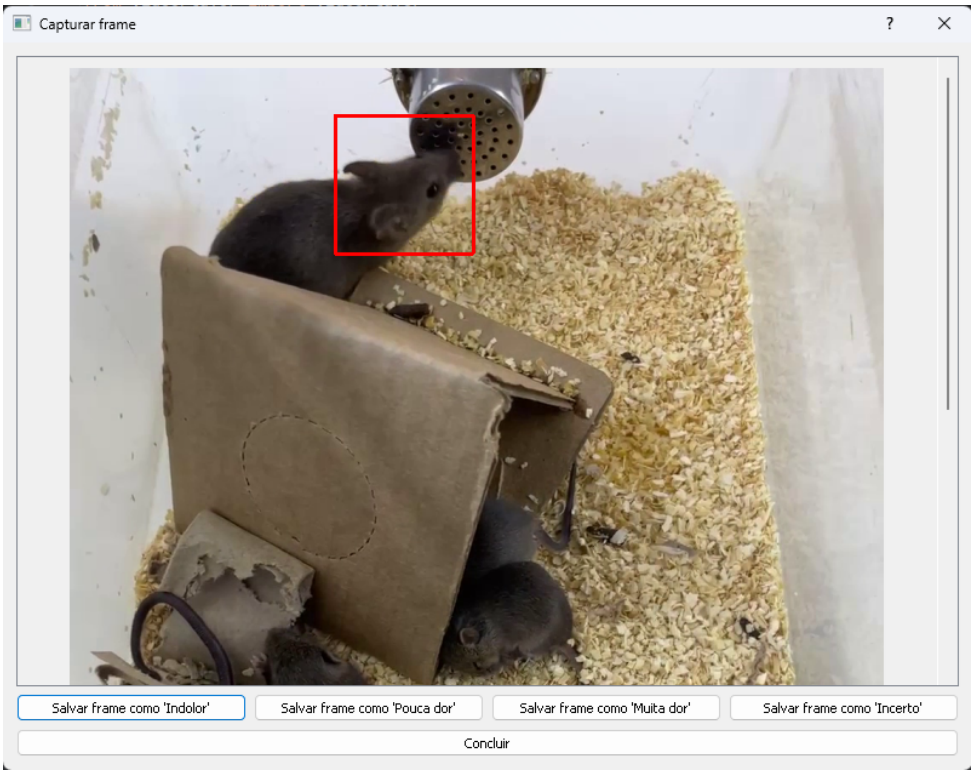

Figura 10 – Sub-menu dedicado à extração de frames. Fonte: elaboração própria.

Outra funcionalidade de destaque é o sub-menu voltado para o gerenciamento das seleções salvas, denominado no programa como 'Menu de Salvamento'. Seu principal objetivo é permitir que o usuário visualize quais imagens estão armazenadas na base de dados e as exclua, caso necessário. Esse recurso facilita a organização e oferece maior segurança à base, dispensando a necessidade de alterações manuais na estrutura de diretórios criada pelo sistema.

A Figura 11 ilustra a interface do sub-menu no momento de sua ativação. As imagens são organizadas conforme o rótulo atribuído e dispostas em categorias distintas. Cada imagem possui um botão de exclusão individual, que pode ser acionado pelo usuário para removê-la da base de dados. Vale destacar que esse sub-menu exibe apenas as imagens associadas ao vídeo que está sendo reproduzido no momento do seu acionamento, o que contribui para um gerenciamento mais preciso e contextualizado dos dados.

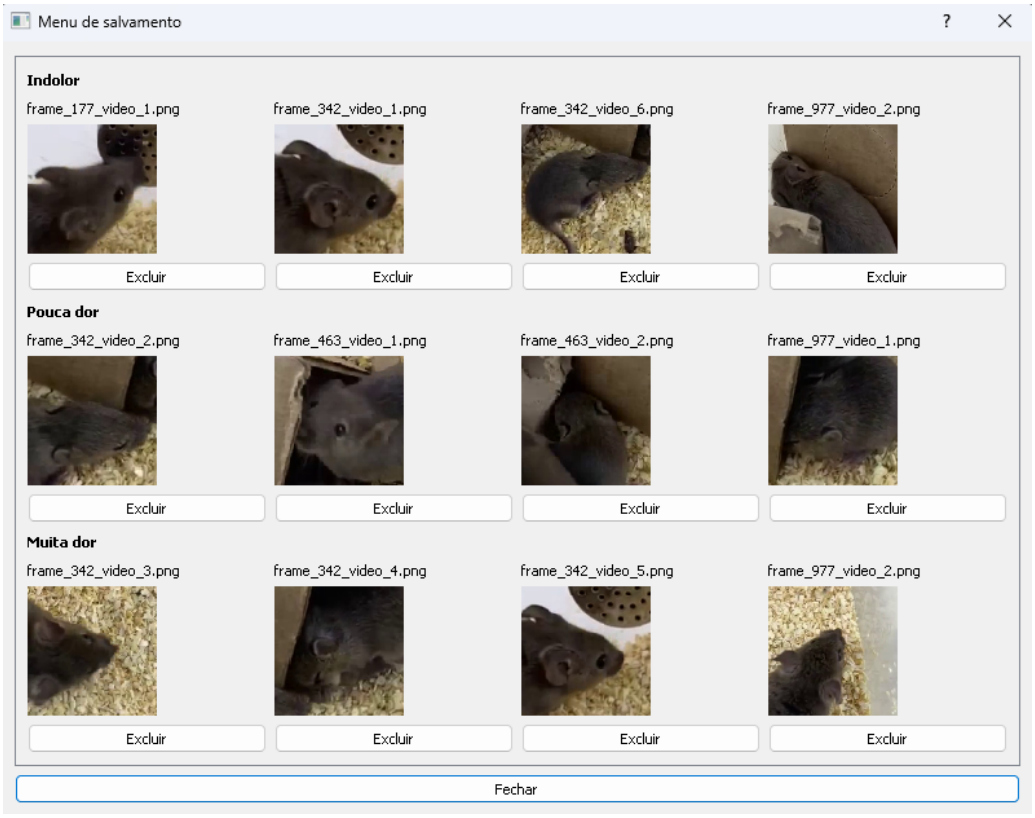

Figura 11 – Sub-menu dedicado à verificação das imagens armazenadas. Fonte: elaboração própria.

Essa ferramenta foi disponibilizada à equipe veterinária responsável pela manipulação dos camundongos durante a coleta de dados. Os *feedbacks* recebidos contribuíram significativamente para o refinamento do sistema, tornando-o mais adequado à tarefa proposta. Além de facilitar o trabalho da equipe, a ferramenta também garantiu a construção de uma base de dados sólida e consistente, essencial para a realização dos experimentos futuros.

3.4 Construção e treinamento dos modelos

A construção dos modelos compreende a definição e configuração de toda a estrutura matemática e computacional necessária para que a rede seja capaz de resolver um problema proposto. Em outras palavras, essa etapa estabelece como o modelo será organizado, como os dados são processados e de que forma seus parâmetros serão ajustados ao longo do treinamento. O primeiro passo consiste na escolha da arquitetura de rede, responsável por definir a estrutura lógica de processamento. Neste estudo, foram desenvolvidos dois modelos distintos: o primeiro, baseado em uma arquitetura convolucional derivada da *ResNet-50*, e o segundo, fundamentado na arquitetura de *Transformers*, aplicada ao domínio da visão computacional.

Para o modelo convolucional, adotou-se a topologia padrão da arquitetura *ResNet-*

50, conforme descrita em sua publicação original. Essa escolha assegura a construção de um modelo estável e reprodutível, uma vez que as especificações dessa configuração encontram-se amplamente consolidadas e validadas na literatura científica. Além disso, o modelo foi inicializado com pesos pré-treinados na base *ImageNet*, utilizando a técnica de *transfer learning*. Esse procedimento permite que o conhecimento adquirido pelo treinamento anterior sirva como ponto de partida para o aprendizado de características mais específicas da base de dados empregada neste estudo.

O *transfer learning* também é um fator de exigência para o uso da topologia original da *ResNet-50*, uma vez que os pesos pré-treinados foram otimizados considerando essa estrutura específica. Modificações nas dimensões dos blocos convolucionais, nas conexões residuais ou na profundidade da rede comprometeriam a compatibilidade desses pesos, exigindo um novo treinamento completo a partir do zero. Assim, a adoção da configuração original garante não apenas a consistência experimental, mas também o aproveitamento do conhecimento adquirido durante o pré-treinamento com a *ImageNet*.

É importante destacar que alguns hiperparâmetros que não alteram diretamente a topologia da rede podem ser ajustados de acordo com o contexto experimental. A modificação controlada desses parâmetros possibilita maior flexibilidade no treinamento e permite avaliar o comportamento do modelo sob diferentes configurações de aprendizado, sendo eles:

- ❑ **input_size:** define a dimensão da imagem de entrada;
- ❑ **batch_size:** define a quantidade de amostras processadas por iteração;
- ❑ **epochs:** define a quantidade de ciclos completos de treinamento;
- ❑ **validation_split:** define a proporção de dados destinada à validação.

A construção do modelo baseado na arquitetura *Vision Transformer (ViT)* também seguiu a topologia padrão definida em sua publicação original, preservando as dimensões e configurações de referência, a fim de garantir estabilidade experimental. Entretanto, além da possibilidade de modificação de hiperparâmetros que não alteram diretamente a topologia da rede, a construção deste modelo permite a variação de determinados parâmetros estruturais, possibilitando a busca por configurações mais adequadas ao conjunto de dados e aos recursos computacionais disponíveis. Entre os parâmetros passíveis de ajuste encontram-se:

- ❑ **patch_size:** permite alterar o tamanho dos blocos da imagem que são processados pelo modelo, modificando a quantidade de *tokens* extraídos e o fluxo de atenção;
- ❑ **projection_dim:** refere-se à dimensão do embedding no ViT, responsável por definir o tamanho do espaço onde os vetores Q, K e V são alocados;

- **transformer_layers:** define a quantidade de blocos de atenção empilhados;
- **num_heads:** define a quantidade de cabeças de atenção, consequentemente alterando a forma como a projeção é dividida nas diferentes cabeças;
- **mlp_units:** refere-se à quantidade de unidades na rede *feed-forward*.

A construção efetiva dos modelos foi desenvolvida inteiramente na linguagem Python (versão 3.11), por meio do ambiente de desenvolvimento PyCharm Community Edition. Dois softwares distintos foram desenvolvidos (um para cada modelo), mas ambos seguem o mesmo padrão de construção e contemplam uma interface gráfica gerada a partir da biblioteca PyQt5, a fim de proporcionar uma experiência mais amigável ao usuário, bem como automatizar e abstrair o processo de construção e treinamento do modelo.

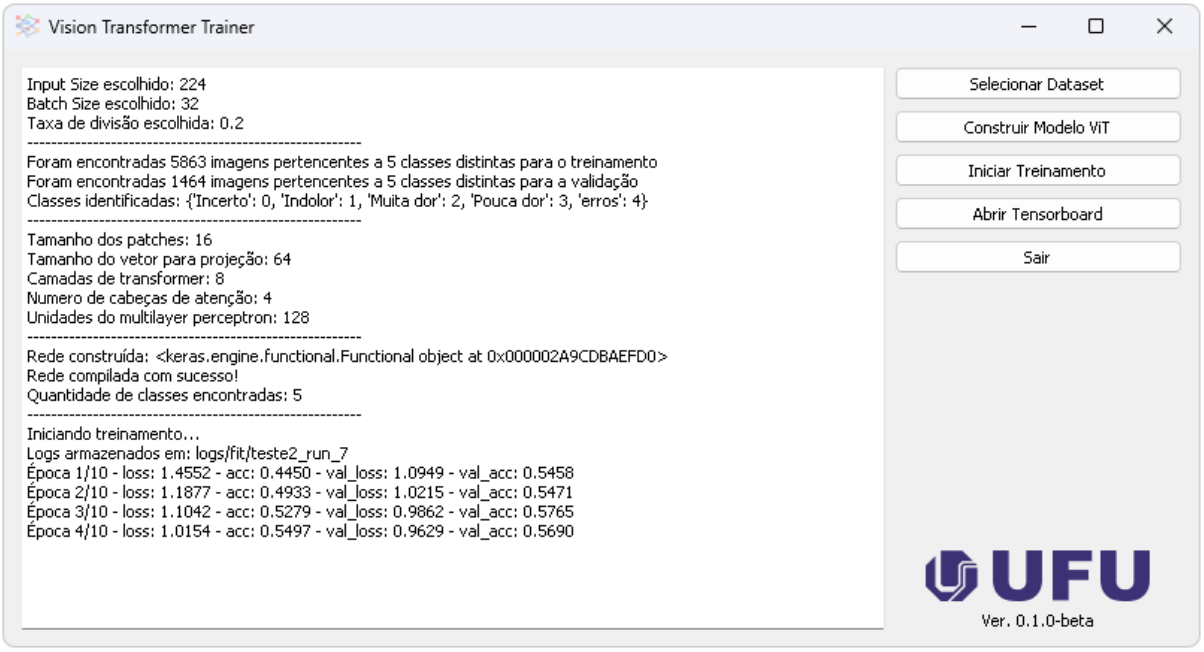

Figura 12 – Janela principal do software desenvolvido para a construção e o treinamento utilizando a arquitetura ViT. Fonte: elaboração própria.

A Figura 12 apresenta a interface desenvolvida para a implementação da arquitetura *Vision Transformer* e execução do treinamento. O ambiente foi projetado de forma a integrar todas as etapas do fluxo de construção e ajuste do modelo, oferecendo uma interface que centraliza as principais funcionalidades.

À esquerda, observa-se o painel de monitoramento, responsável por exibir o status das operações realizadas. Nesse espaço, o usuário pode acompanhar: os parâmetros de entrada definidos antes do treinamento; a divisão da base de dados, quantificando as imagens destinadas ao treino e à validação; as classes identificadas automaticamente na base; os parâmetros estruturais da rede; o diretório de armazenamento dos arquivos de log; e as métricas de desempenho ao longo das épocas. À direta da janela de execução, estão

alocadas as cinco funcionalidades principais do programa, as quais permitem ao usuário, interagir de fato com a ferramenta. Suas descrições e propósitos são apresentados a seguir:

- **Selecionar *Dataset*:** possibilita a escolha da base de dados a ser utilizada no treinamento. Ao acionar essa opção, uma janela do explorador de arquivos é aberta para que o usuário selecione o diretório que contém a base organizada em subpastas, sendo cada subpasta correspondente a uma classe distinta.

  Após a seleção do diretório, o sistema solicita a definição dos parâmetros referentes ao pré-processamento e ao tratamento inicial dos dados, conforme ilustrado na Figura 13. O usuário deve informar o `Input_size` (dimensão das imagens de entrada), o `Batch_size` (tamanho do lote processado por iteração) e a proporção destinada à validação (`Validation_split`).

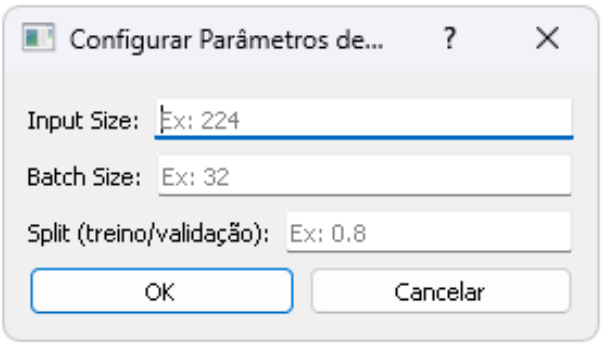

Figura 13 – Parâmetros de configuração dos dados definidos pelo usuário no momento da seleção da base. Fonte: elaboração própria.

- **Construir Modelo ViT:** executa a construção do modelo de acordo com os parâmetros fornecidos. Essa etapa depende da seleção prévia da base de dados, uma vez que a configuração do modelo utiliza informações sobre as dimensões de entrada e a estrutura das classes. Ao solicitar a construção da rede, o programa apresenta uma nova janela (Figura 14), na qual o usuário define os hiperparâmetros estruturais da ViT: `Patch_size`, `Projection_dim`, `Transformer_layers`, `Attention_heads` e `MLP_units`. Essa flexibilidade permite variações em relação à topologia original, possibilitando testes comparativos entre diferentes configurações.

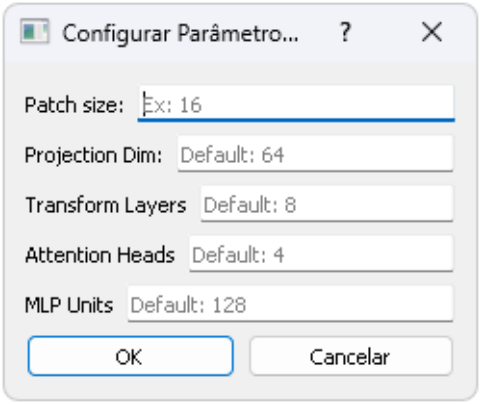

Figura 14 – Parâmetros estruturais do modelo definidos pelo usuário no momento da construção. Fonte: elaboração própria.

- **Iniciar Treinamento:** inicia o processo de ajuste dos pesos do modelo por meio da função `model.fit()` da API Keras (integrada ao *TensorFlow*). Antes do início do treinamento, o sistema solicita ao usuário a quantidade de épocas (`epochs`) e o nome do diretório em que os arquivos de log serão armazenados, conforme apresentado na Figura 15. Essa configuração permite registrar o histórico de treinamento e viabiliza a análise posterior via *TensorBoard*.

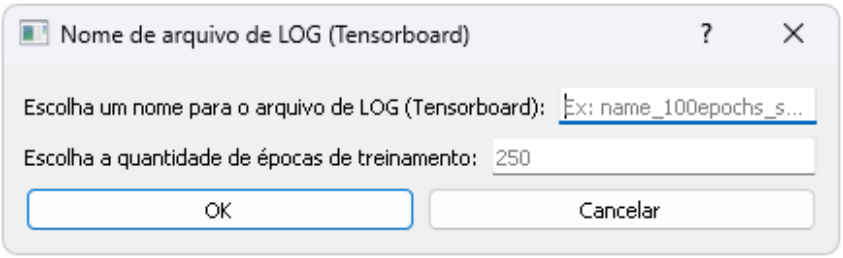

Figura 15 – Parâmetros definidos pelo usuário antes da execução do treinamento. Fonte: elaboração própria.

- **Abrir *TensorBoard*:** inicializa automaticamente o servidor local do *TensorBoard*, direcionando-o para o diretório de logs gerado durante o treinamento. Após a execução, o navegador é aberto no endereço padrão (*http://localhost:6006*), permitindo a visualização interativa das métricas e curvas de desempenho da rede;

- **Sair:** finaliza a execução do programa e, caso um treinamento esteja em andamento, interrompe o processo via truncamento, mas garante a integridade dos logs e dos pesos já armazenados.

A construção da aplicação desenvolvida para a implementação da arquitetura *ResNet-50* e a respectiva execução do treinamento segue o mesmo fluxo e estrutura apresentado pela implementação anterior (Incluindo as janelas de configurações ilustradas nas Figuras 13 e 15). A única diferença é que no momento da construção da rede, acessada pelo

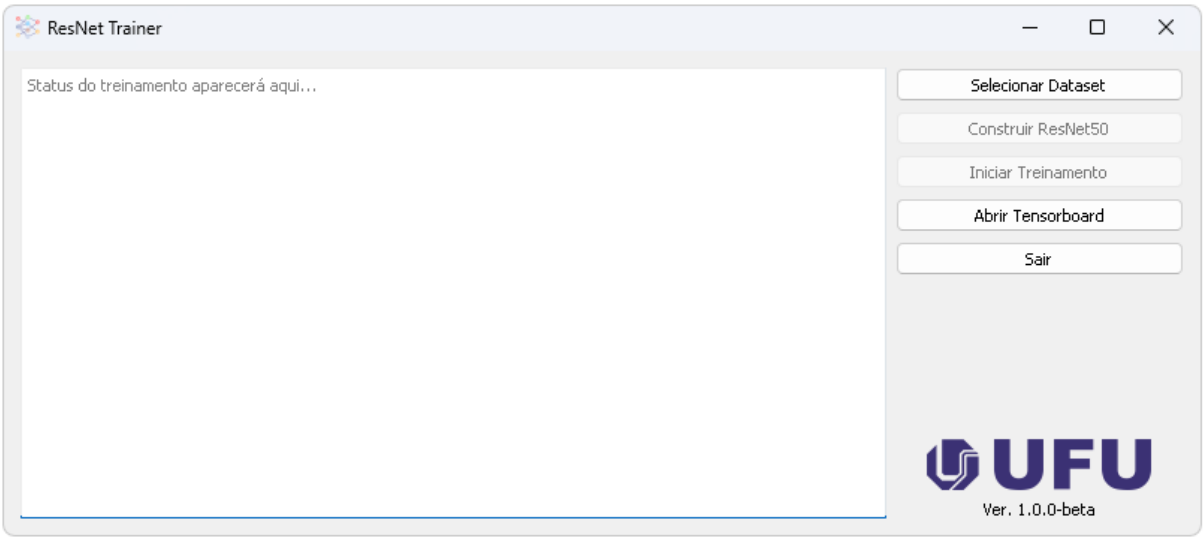

Figura 16 – Janela principal do software desenvolvido para a construção e o treinamento utilizando a arquitetura ResNet50. Fonte: elaboração própria.

botão 'Construir ResNet50' não é solicitado ao usuário a definição de hiperparâmetros estruturais, uma vez que a implementação usa a técnica de *transfer learning* (a qual exige a topologia padrão estabelecida pela literatura). A interface inicial deste software é ilustrada pela Figura 16

A construção da aplicação desenvolvida para a implementação da arquitetura *ResNet-50*, bem como a execução de seu treinamento, segue o mesmo fluxo e estrutura descritos anteriormente para a arquitetura *Vision Transformer*, incluindo as janelas de configuração ilustradas nas Figuras 13 e 15. A interface inicial do software desenvolvido para essa implementação é apresentada na Figura 16.

A principal diferença entre as duas implementações está na etapa de construção do modelo, acessada por meio do botão *"Construir ResNet50"*. Nesta aplicação, não é solicitado ao usuário o ajuste de hiperparâmetros estruturais, uma vez que a rede utiliza a técnica de *transfer learning*, a qual requer a manutenção da topologia original da arquitetura, conforme estabelecida na literatura. Essa abordagem se faz necessária para garantir a compatibilidade com os pesos pré-treinados da base *ImageNet* e preservar as características fundamentais da estrutura da *ResNet-50*.

3.5 Técnicas para avaliação de resultados

A avaliação dos resultados constitui uma etapa crucial na metodologia deste estudo, uma vez que permite mensurar a eficácia dos modelos e assegurar a validade das conclusões obtidas. Por isso, foi definido um protocolo de avaliação que visa atender aos seguintes requisitos:

- ❏ Mensurar a capacidade dos modelos em detectar e quantificar estímulos de dor a

partir das imagens faciais dos camundongos;

- ❏ Comparar o desempenho entre as arquiteturas testadas (ResNet e ViT);
- ❏ Avaliar a robustez e a capacidade de generalização dos modelos frente a variações ou ruídos na base de dados;
- ❏ Investigar se as regiões relevantes para a classificação correspondem aos padrões descritos pela *Mouse Grimace Scale* (MGS), garantindo a interpretabilidade dos resultados.

Para a coleta e análise das métricas de desempenho, foram utilizados os registros gerados pelo *TensorFlow*, em conjunto *TensorBoard* para a tarefa de monitoramento. Durante o treinamento, os principais indicadores, como a função de perda (*loss*) e a acurácia, foram registrados automaticamente em arquivos de *logs* utilizando *callbacks* configurados no próprio código de treinamento. Esses registros foram organizados em diretórios específicos, permitindo a análise de cada execução.

O *TensorBoard* foi utilizado como ferramenta de apoio para a avaliação visual e quantitativa do aprendizado, possibilitando o acompanhamento das curvas de desempenho ao longo das épocas de treinamento. Essa visualização facilita a identificação de padrões de convergência, eventuais sinais de *overfitting* e diferenças de comportamento entre execuções, sendo fundamental para uma análise mais assertiva e minuciosa.

Adicionalmente, planeja-se complementar a análise por meio de técnicas estatísticas adicionais, como a construção de matrizes de confusão, cálculo de precisão (*precision*), revocação (*recall*) e F1-Score, permitindo uma avaliação mais abrangente do desempenho dos modelos, levando em consideração o equilíbrio entre as classes e o comportamento das arquiteturas frente aos dados apresentados.

Capítulo 4

Experimentos e Análise dos Resultados

4.1 Método para a Avaliação

Descreva os métodos utilizados para validar a sua hipótese incluindo as medidas de avaliação, conjunto de parâmetros, bases de dados e os trabalhos com os quais a sua proposta será comparada.

4.2 Experimentos

De acordo com o que foi descrito na Seção **??**, apresente os resultados dos seus experimentos. A apresentação dos resultados pode ser feita via gráficos ou tabelas. O importante é que haja clareza.

4.3 Avaliação dos Resultados

A avaliação dos resultados pode ser feita à medida em que os resultados dos experimentos são apresentados, ou em uma seção separada. É importante que você aponte os acertos e as limitações da sua proposta e justifique os resultados obtidos. É fundamental apresentar evidências de que sua hipótese é verdadeira.

Capítulo 5

Conclusão

TRECHO PROVENIENTE DA INTRODUÇÃO:

Embora este trabalho utilize especificamente a *Mouse Grimace Scale*, o conceito por trás dessa metodologia apresenta forte potencial de aplicação em outras espécies, o que traz ao estudo um fator de escalabilidade muito forte para outras aplicações e áreas do conhecimento. A capacidade de escalabilidade da *grimmace scale* se dá por alguns fatores, dentre eles temos: a universalidade de indicadores faciais de dor e a base fisiológica comum entre mamíferos quanto às reações de dor.

Conforme descrito previamente, a *grimmace scale* busca traços faciais e microexpressões específicas para definir a presença e o nível de um dado estímulo doloroso, traços estes que se apresentam de forma bem semelhante em diferentes espécies (como olhos, boca, focinho, etc). Além disso, as microexpressões avaliadas pela *grimmace scale* costumam ter um certo grau de similaridade entre as diferentes espécies, como o fechamento dos olhos ou a contração do focinho, por exemplo. Com isso, um sistema automatizado que se utilize dessa ferramenta não precisaria de grandes mudanças para se adequar ao reconhecimento de tais estímulos em outras espécies, o que reflete diretamente no potencial de escalabilidade deste projeto.

5.1 Principais Contribuições

Nessa seção destaque ainda mais as suas contribuições, mostrando que sua hipótese foi validada pelos experimentos executados.

5.2 Trabalhos Futuros

Destaque nessa seção o que pode ser melhorado no método proposto para resolver as possíveis falhas que você identificou e descreveu na seção **??**. Indique quais outros projetos podem ser gerados a partir do seu trabalho.

5.3 Contribuições em Produção Bibliográfica

Liste a produção bibliográfica resultante do seu trabalho.

Referências

ALQAHTANI, H. **Context pre-modeling: an empirical analysis for classification based user-centric context-aware predictive modeling**. 2020. Acesso em: 21 out. 2025. Disponível em: <https://www.researchgate.net/figure/An-example-of-neural-network-with-multiple-hidden-layers_fig2_343177704>.

HUANG, Z. **Self-Supervised Dementia Prediction From MRI Scans With Metadata Integration**. 2023. Acesso em: 21 out. 2025. Disponível em: <https://www.researchgate.net/figure/sion-Transformer-architecture-figure-from-the-original-paper-DBK-20_fig1_374982152>.

JIANG, W. **A traffic and road signal recognition method through deep learning**. 2024. Acesso em: 21 out. 2025. Disponível em: <https://www.researchgate.net/figure/How-a-convolution-layer-is-utilized_fig1_382297179>.

MORITZ, L. **AI Snippet of the Week: Supervised vs. Unsupervised Learning**. 2024. Acesso em: 21 out. 2025. Disponível em: <https://www.linkedin.com/posts/laura-moritz_machinelearning-supervisedlearning-neuralnetworks-activity-7240316292115685376-ig44/>.

PRABHA, A. **Intelligent Mask Detection Using Deep Learning Techniques**. 2021. Acesso em: 21 out. 2025. Disponível em: <https://www.researchgate.net/figure/Transfer-Learning-Architecture-3-TensorFlow-API-The-object-detector-using-Tensorflow_fig3_351926911>.

STACK, T. **A Comprehensive Guide to ResNet50: Architecture and Implementation**. 2025. Acesso em: 21 out. 2025. Disponível em: <https://www.thinkingstack.ai/blog/business-use-cases-11/a-comprehensive-guide-to-resnet50-architecture-and-implementation-56>.

WEHBE, L. **HMM (Markov Chain) — Machine Learning Algorithms — 2**. 2020. Acesso em: 21 out. 2025. Disponível em: <https://medium.com/@larawehbee/hmm-markov-chain-machine-learning-algorithms-2-39c62b00a198>.