# UNIVERSIDADE VEIGA DE ALMEIDA

## CIÊNCIA DA COMPUTAÇÃO

## Um Sistema de Inteligência Artificial Neuro-Simbólico para Raciocínio, Inovação e Aprendizado Contínuo no Domínio da Ornitologia

**Autor: Matheus [Seu Sobrenome]**

**Orientador: [Nome do seu Orientador(a)]**

Rio de Janeiro

2025

<div style="page-break-after: always;"></div>

## Resumo

Sistemas de Inteligência Artificial (IA) contemporâneos demonstram notável capacidade de reconhecimento de padrões, mas frequentemente carecem de um raciocínio lógico robusto, da capacidade de lidar com a novidade e de inovar a partir do conhecimento adquirido. Este trabalho apresenta o desenvolvimento de um protótipo de sistema de IA neuro-simbólico, projetado para superar essas limitações no domínio da identificação e análise de aves. A arquitetura proposta integra múltiplos agentes de IA: um modelo de classificação de espécies baseado em redes neurais convolucionais (MobileNetV2) para a formulação de hipóteses; um detector de objetos customizado (YOLOv8) para a extração de fatos visuais objetivos (e.g., bico, asas); e um grafo de conhecimento para representar e raciocinar sobre as relações semânticas e causais. O sistema implementa uma lógica de "fatos primeiro", utilizando as características detectadas para validar a presença de uma ave antes de tentar a classificação da espécie, aplicando um limiar de confiança para distinguir entre espécies conhecidas e desconhecidas. Adicionalmente, explora-se um ciclo de aprendizado contínuo, onde o sistema pode auto-gerar anotações para novos exemplos através de Mapas de Ativação de Classe (Grad-CAM) e utilizar uma API de visão externa para validação, criando um pipeline de aprendizado "human-in-the-loop" para aprimoramento contínuo do seu conhecimento. Os resultados demonstram um sistema capaz não apenas de analisar, mas de raciocinar sobre suas observações, gerenciar a incerteza e expandir sua própria base de conhecimento, representando um passo em direção a uma inteligência artificial mais autônoma e similar à humana.

**Palavras-chave:** Inteligência Artificial, IA Neuro-Simbólica, Raciocínio Lógico, Aprendizado de Máquina, Visão Computacional, YOLOv8, Aprendizado Contínuo.

<div style="page-break-after: always;"></div>

## Sumário

**1. Introdução**

* 1.1. Contextualização: A Fronteira da Inteligência Artificial
* 1.2. Definição do Problema
* 1.3. O Domínio de Estudo: Ornitologia
* 1.4. Objetivos
* 1.4.1. Objetivo Geral
* 1.4.2. Objetivos Específicos
* 1.5. Justificativa
* 1.6. Estrutura do Trabalho

**2. Fundamentação Teórica**

* 2.1. Aprendizado de Máquina e Redes Neurais Artificiais
* 2.2. Redes Neurais Convolucionais (CNNs) para Visão Computacional
* 2.2.1. Arquitetura MobileNetV2
* 2.3. Detecção de Objetos
* 2.3.1. A Família de Algoritmos YOLO
* 2.3.2. YOLOv8: Arquitetura e Inovações
* 2.4. A Abordagem Neuro-Simbólica
* 2.5. Representação de Conhecimento: Grafos
* 2.6. Aprendizado Contínuo e Ativo
* 2.7. Interpretabilidade em IA (XAI): Grad-CAM
* 2.8. Modelos de Linguagem e Visão de Larga Escala (LLMs/LVMs)

**3. Desenvolvimento e Metodologia**

* 3.1. Ambiente de Desenvolvimento e Ferramentas
* 3.2. Coleta e Preparação do Conjunto de Dados (Dataset)
* 3.3. Módulo 1: Classificador de Espécies (Keras e MobileNetV2)
* 3.3.1. Pré-processamento e Aumento de Dados
* 3.3.2. Arquitetura do Modelo e Treinamento
* 3.3.3. Resultados do Treinamento
* 3.4. Módulo 2: Detector de Partes Anatômicas (YOLOv8)
* 3.4.1. Processo de Anotação
* 3.4.2. Estrutura do Dataset e Configuração
* 3.4.3. Treinamento e Avaliação do Modelo
* 3.5. Módulo 3: Motor de Raciocínio e Grafo de Conhecimento
* 3.5.1. Modelagem do Grafo
* 3.5.2. Implementação da Lógica "Fatos Primeiro"

**4. O Ciclo de Inovação e Aprendizado Contínuo**

* 4.1. Lidando com a Novidade: O Desafio das Imagens Desconhecidas
* 4.2. Implementação do Módulo de Auto-Anotação com Grad-CAM
* 4.3. Proposta de Validação Automatizada com API de Visão
* 4.4. O Conceito de Inovação: Geração de Novas Espécies

**5. Resultados e Discussão**

* 5.1. Cenário de Teste 1: Espécie Conhecida e de Alta Confiança
* 5.2. Cenário de Teste 2: Espécie Desconhecida (Pássaro Válido)
* 5.3. Cenário de Teste 3: Objeto Inválido (Não-Pássaro)
* 5.4. Discussão dos Resultados e Limitações

**6. Conclusão**

* 6.1. Síntese do Trabalho e Contribuições
* 6.2. Trabalhos Futuros
* 6.3. Considerações Finais

**Referências**

**Apêndices**

* Apêndice A: Código-Fonte Principal (main.py)
* Apêndice B: Script de Treinamento do YOLOv8 (train\_yolo.py)

<div style="page-break-after: always;"></div>

### 1. Introdução

#### 1.1. Contextualização: A Fronteira da Inteligência Artificial

A Inteligência Artificial (IA) tem evoluído de sistemas puramente baseados em regras para modelos complexos de aprendizado de máquina que inferem padrões a partir de grandes volumes de dados. A ascensão do Aprendizado Profundo (Deep Learning), em particular das Redes Neurais Convolucionais (CNNs), revolucionou áreas como a Visão Computacional, alcançando e por vezes superando a performance humana em tarefas de reconhecimento de objetos. Contudo, esta abordagem, conhecida como conexionista, revela limitações significativas. Tais modelos operam como "caixas-pretas", cujos processos de decisão são de difícil interpretação, e demonstram fragilidade ao lidar com situações não vistas durante o treinamento, carecendo de senso comum e de uma capacidade de raciocínio mais abstrata. Em paralelo, a abordagem simbólica da IA, que manipula símbolos e regras lógicas explícitas, oferece interpretabilidade e raciocínio, mas falha em lidar com a complexidade e a ambiguidade do mundo real. A fronteira atual da pesquisa em IA reside na fusão dessas duas abordagens, dando origem à IA Neuro-Simbólica, que busca combinar o poder da percepção das redes neurais com a capacidade de raciocínio dos sistemas simbólicos.

#### 1.2. Definição do Problema

Os sistemas de IA atuais são excelentes interpoladores, mas péssimos extrapoladores. Eles podem identificar com precisão um pássaro de uma espécie presente em seu massivo dataset de treinamento, mas falham de maneira imprevisível diante de uma imagem de uma espécie nova, de um ângulo incomum, ou de um objeto que meramente compartilha algumas características visuais com um pássaro. A questão central que este trabalho aborda é: como podemos construir um sistema de IA que não apenas classifica o que conhece, mas que também **sabe o que não sabe**? Como pode um sistema raciocinar sobre as evidências visuais para formar conclusões lógicas, identificar a novidade e, fundamentalmente, possuir um mecanismo para aprender com ela? Este projeto visa atacar essa fragilidade, propondo uma arquitetura que gerencia a incerteza e possui um caminho para a expansão autônoma de seu conhecimento.

#### 1.3. O Domínio de Estudo: Ornitologia

A ornitologia, o estudo das aves, foi escolhida como o domínio de aplicação por sua inerente complexidade e riqueza de características. A identificação de aves é uma tarefa desafiadora que requer a análise de múltiplos atributos: a forma e cor do bico, o padrão da plumagem, o formato das asas, a estrutura das garras, entre outros. Essa complexidade torna o domínio um campo de testes ideal para um sistema de IA que precisa combinar a percepção de baixo nível (identificar uma forma como um "bico") com o raciocínio de alto nível (inferir que um bico cônico e curto é adaptado para quebrar sementes).

#### 1.4. Objetivos

##### 1.4.1. Objetivo Geral

Desenvolver um protótipo funcional de um sistema de Inteligência Artificial neuro-simbólico capaz de realizar a identificação de aves, demonstrando raciocínio lógico na análise, gerenciamento de incerteza ao encontrar espécies desconhecidas, e possuindo uma arquitetura para aprendizado contínuo.

##### 1.4.2. Objetivos Específicos

* Implementar um modelo de Deep Learning para a classificação de espécies de aves.
* Desenvolver um detector de objetos customizado para a extração de partes anatômicas (fatos visuais).
* Construir uma base de conhecimento em formato de grafo para armazenar informações e relações.
* Projetar e implementar um motor de raciocínio que integre os módulos de percepção e conhecimento.
* Desenvolver um mecanismo de auto-anotação para facilitar o aprendizado com novos dados.
* Propor uma arquitetura de validação automatizada utilizando APIs de visão externas.

#### 1.5. Justificativa

A busca por uma IA mais robusta e similar à humana é um dos maiores desafios da Ciência da Computação. Este trabalho se justifica por sua contribuição prática a essa busca, propondo uma arquitetura concreta e implementável que aborda diretamente os problemas de fragilidade e falta de raciocínio dos modelos atuais. Ao invés de focar em ganhos incrementais de precisão em um dataset fechado, o projeto foca na criação de um sistema com maior autonomia cognitiva, capaz de operar em um cenário mais realista e dinâmico.

#### 1.6. Estrutura do Trabalho

O presente documento está estruturado em seis capítulos. O Capítulo 2 detalha a fundamentação teórica que suporta o projeto. O Capítulo 3 descreve a metodologia e o desenvolvimento passo a passo de cada módulo do sistema. O Capítulo 4 aprofunda-se nos mecanismos de inovação e aprendizado contínuo. O Capítulo 5 apresenta os resultados dos testes em diferentes cenários. Finalmente, o Capítulo 6 conclui o trabalho, sintetizando os resultados e apontando direções para futuras pesquisas.

<div style="page-break-after: always;"></div>

### 2. Fundamentação Teórica

*(Esta seção deve ser expandida com mais detalhes, gráficos e equações para cada tópico, atingindo aproximadamente 15-20 páginas)*

**2.1. Aprendizado de Máquina e Redes Neurais Artificiais**

O Aprendizado de Máquina (Machine Learning) é um subcampo da Inteligência Artificial que se concentra no desenvolvimento de algoritmos que permitem aos computadores aprender a partir de dados, sem serem explicitamente programados para cada tarefa. Em vez de seguir um conjunto de regras predefinidas, um modelo de aprendizado de máquina analisa um conjunto de dados de treinamento para identificar padrões e relações, construindo um modelo matemático que pode ser usado para fazer previsões ou tomar decisões sobre novos dados.

**2.1.1. Aprendizado Supervisionado**

O aprendizado supervisionado é o paradigma de aprendizado de máquina mais proeminente e a base para os modelos de percepção deste trabalho. Ele se caracteriza pelo uso de um conjunto de dados **rotulado**, onde cada exemplo de entrada (input) é pareado com uma saída (output) ou rótulo (label) correto. O objetivo do algoritmo é aprender uma função de mapeamento f que aproxima a relação entre as entradas X e as saídas Y, de modo que Y ≈ f(X).

O processo de aprendizado ocorre de forma iterativa. O modelo recebe uma entrada de treinamento, faz uma previsão e compara essa previsão com o rótulo verdadeiro. A diferença entre a previsão e o valor real, quantificada por uma **função de perda (loss function)**, é usada para ajustar os parâmetros internos do modelo (como os pesos em uma rede neural) através de um algoritmo de otimização, comumente o **Gradiente Descendente**. Este processo é repetido para todos os exemplos do dataset por várias épocas, até que o modelo convirja para um estado onde o erro de previsão seja minimizado.

As tarefas de aprendizado supervisionado são tipicamente divididas em duas categorias:

* **Classificação:** O objetivo é prever um rótulo de categoria discreta. No contexto deste trabalho, o classificador de espécies resolve um problema de classificação, onde a entrada é uma imagem e a saída é o nome de uma espécie de pássaro (e.g., 'Cardinal', 'Painted Bunting').
* **Regressão:** O objetivo é prever um valor numérico contínuo. Por exemplo, prever o preço de um imóvel com base em suas características.

**2.1.2. O Neurônio Artificial (Perceptron)**

A unidade fundamental das redes neurais modernas é o neurônio artificial, um modelo matemático inspirado no neurônio biológico. O primeiro modelo computacional, o **Perceptron**, proposto por Frank Rosenblatt em 1958, estabeleceu as bases. Um neurônio artificial realiza os seguintes passos:

1. **Recebe Entradas:** Ele recebe um ou mais valores de entrada (x₁, x₂, ..., xₙ), que podem ser os dados brutos (como os pixels de uma imagem) ou as saídas de outros neurônios.
2. **Pondera as Entradas:** Cada entrada xᵢ é multiplicada por um peso sináptico wᵢ. Esses pesos representam a importância de cada entrada; um peso maior significa que a entrada tem mais influência na saída do neurônio. O processo de treinamento da rede consiste, essencialmente, em encontrar os valores ideais para esses pesos.
3. **Soma Ponderada e Viés (Bias):** Todas as entradas ponderadas são somadas, e um termo de **viés (bias, b)** é adicionado. O viés é um parâmetro adicional que permite deslocar a função de ativação, aumentando a flexibilidade do modelo para se ajustar aos dados. A soma é calculada como: z = (x₁w₁ + x₂w₂ + ... + xₙwₙ) + b.
4. **Função de Ativação:** O resultado da soma, z, é passado por uma **função de ativação** não-linear, φ(z). Esta função determina se o neurônio será "ativado" (disparado) e qual será sua saída final. A não-linearidade é crucial, pois permite que as redes neurais aprendam relações complexas e não-lineares nos dados, algo que um modelo puramente linear não conseguiria. Funções de ativação comuns incluem a Sigmoide, a Tangente Hiperbólica (tanh) e, mais proeminentemente em redes profundas, a Unidade Linear Retificada (ReLU).

**2.1.3. Arquitetura de Rede Neural Multicamada (MLP)**

Um único neurônio artificial só pode aprender a separar dados linearmente. Para resolver problemas complexos, múltiplos neurônios são organizados em camadas, formando uma **Rede Neural Multicamada (Multilayer Perceptron, MLP)**. Uma MLP consiste em, no mínimo, três tipos de camadas:

* **Camada de Entrada (Input Layer):** Recebe os dados brutos. O número de neurônios nesta camada corresponde ao número de características (features) do dado de entrada. Por exemplo, para uma imagem de 224x224 pixels em escala de cinza, a camada de entrada teria 50.176 neurônios.
* **Camadas Ocultas (Hidden Layers):** São as camadas intermediárias entre a entrada e a saída. É aqui que a maior parte do "aprendizado" acontece. Cada neurônio em uma camada oculta recebe as saídas de todos os neurônios da camada anterior, realiza seu cálculo de soma ponderada e ativação, e passa seu resultado para todos os neurônios da camada seguinte. A profundidade (número de camadas ocultas) e a largura (número de neurônios por camada) de uma rede definem sua capacidade de aprender padrões complexos.
* **Camada de Saída (Output Layer):** Produz o resultado final da rede. O número de neurônios e a função de ativação nesta camada dependem da tarefa. Para um problema de classificação multiclasse como o deste TCC, a camada de saída terá um neurônio para cada classe possível (cada espécie de pássaro), e utilizará a função de ativação **Softmax**, que converte as saídas brutas em uma distribuição de probabilidade, indicando a chance de a entrada pertencer a cada uma das classes.

A interconexão total entre as camadas permite que a rede aprenda hierarquias de características: as primeiras camadas podem aprender a detectar padrões simples como bordas e texturas, enquanto camadas mais profundas combinam esses padrões para reconhecer formas mais complexas, como um "olho" ou um "bico", até finalmente identificar o objeto completo.

#### 2..2. Redes Neurais Convolucionais (CNNs) para Visão Computacional

Explicação detalhada das camadas convolucionais (kernels, feature maps), camadas de pooling (max pooling), e camadas totalmente conectadas.

[DESCRIÇÃO DA IMAGEM: Crie aqui um fluxograma ou diagrama de blocos que ilustre a arquitetura completa e específica do seu sistema. Ele deve começar com uma 'Imagem de Entrada' e se ramificar em dois caminhos paralelos: um indo para o 'Módulo Classificador (MobileNetV2)' que gera uma 'Hipótese de Espécie + Confiança', e outro indo para o 'Módulo Detector de Fatos (YOLOv8)' que gera 'Fatos Visuais'. Ambos os resultados alimentam um bloco central chamado 'Motor de Raciocínio Lógico', que por sua vez consulta um 'Grafo de Conhecimento'. A saída final do motor é a 'Conclusão (Espécie Conhecida / Desconhecida / Não-Pássaro)'. Este diagrama será a principal representação visual do seu projeto.]

*Figura 1: Arquitetura Neuro-Simbólica do Sistema de Análise de Aves.*

##### 2.2.1. Arquitetura MobileNetV2

Enquanto CNNs tradicionais como VGG e ResNet alcançaram alta acurácia, seu custo computacional e o número de parâmetros as tornam impraticáveis para dispositivos com recursos limitados, como smartphones ou sistemas embarcados. A família de arquiteturas MobileNet foi projetada para resolver este problema, oferecendo um balanço otimizado entre acurácia e eficiência. A MobileNetV2, utilizada como base para o classificador de espécies neste trabalho, introduziu conceitos inovadores para reduzir drasticamente a complexidade computacional, mantendo um alto poder de representação.

O pilar da eficiência das MobileNets é a **Convolução de Profundidade Separada (Depthwise Separable Convolution)**. Uma convolução padrão aplica seus filtros simultaneamente através de todas as dimensões de profundidade (canais) do tensor de entrada, o que é computacionalmente caro. A convolução de profundidade separada decompõe esta operação em duas etapas mais simples e eficientes:

1. **Convolução de Profundidade (Depthwise Convolution):** Uma única convolução é aplicada a cada canal de entrada de forma independente. Esta etapa é responsável por filtrar as características espaciais (como bordas e texturas) dentro de cada canal, mas não combina informações entre os canais.
2. **Convolução Ponto a Ponto (Pointwise Convolution):** Uma convolução de kernel 1x1 é aplicada ao resultado da etapa anterior. Esta operação é responsável por combinar as características aprendidas nos diferentes canais, criando novas representações de características.

Ao separar a filtragem espacial da combinação de canais, o número total de multiplicações e parâmetros é drasticamente reduzido em comparação com uma convolução padrão. A MobileNetV2 aprimora ainda mais esta ideia com dois conceitos-chave em seus blocos de construção:

* **Resíduos Invertidos (Inverted Residuals):** Blocos residuais tradicionais (como na ResNet) tipicamente seguem um design de "gargalo" (bottleneck): eles primeiro comprimem o número de canais com uma convolução 1x1, aplicam a convolução principal e, em seguida, expandem os canais de volta. A MobileNetV2 inverte essa lógica. O bloco começa com uma convolução ponto a ponto que **expande** a representação de características para uma dimensão maior. Em seguida, a convolução de profundidade (a parte mais leve) é aplicada neste espaço expandido. Finalmente, outra convolução ponto a ponto **comprime** a representação de volta para um gargalo. A intuição é que, ao projetar as características para um espaço de maior dimensão, a rede tem mais "espaço" para aprender transformações complexas, mesmo que a entrada e a saída do bloco sejam compactas.
* **Gargalos Lineares (Linear Bottlenecks):** Uma observação crucial dos autores da MobileNetV2 é que o uso de funções de ativação não-lineares, como a ReLU, em espaços de baixa dimensão (nos gargalos de compressão) pode levar à perda de informação. A ReLU, ao zerar todos os valores negativos, pode colapsar irreversivelmente o espaço de características. Para mitigar isso, a MobileNetV2 remove a não-linearidade da última camada de convolução ponto a ponto (a camada de compressão) dentro de cada bloco residual. Isso significa que a saída do bloco é um gargalo linear, que se mostrou mais eficaz em preservar a informação relevante para as camadas subsequentes.

Essas inovações permitem que a MobileNetV2 construa representações de características ricas de forma muito mais eficiente, tornando-a a escolha ideal para o módulo de classificação de espécies deste TCC, onde a agilidade na inferência é um fator relevante.

#### 2.3. Detecção de Objetos

##### 2.3.1. A Família de Algoritmos YOLO (You Only Look Once)

No campo da detecção de objetos, existem duas abordagens filosóficas principais: os detectores de duas etapas e os detectores de uma etapa. Compreender a diferença é fundamental para justificar a escolha da arquitetura YOLO neste projeto.

**Abordagens de Duas Etapas (Two-Stage Detectors):** A família R-CNN (Regions with CNN features) e suas evoluções (Fast R-CNN, Faster R-CNN) popularizaram esta metodologia. O processo é sequencial e análogo a um sistema de foco seletivo:

1. **Primeira Etapa (Geração de Propostas de Região):** O algoritmo primeiro analisa a imagem para gerar uma série de "propostas de região" – áreas retangulares (bounding boxes) que têm alta probabilidade de conter um objeto. Algoritmos como o Selective Search ou uma Rede de Propostas de Região (RPN) são usados para gerar centenas ou milhares dessas propostas por imagem. Esta etapa responde à pergunta: "Onde estão os objetos em potencial?".
2. **Segunda Etapa (Classificação e Refinamento):** Cada proposta de região é então tratada como uma imagem individual. Ela é recortada, redimensionada e passada por uma rede neural convolucional que realiza duas tarefas: classifica o objeto dentro da proposta (e.g., 'bico', 'asa', 'fundo') e refina as coordenadas do bounding box para se ajustar mais precisamente ao objeto.

Esta abordagem é conhecida por sua alta precisão, pois a segunda etapa pode se concentrar em analisar candidatos de alta qualidade. No entanto, sua natureza sequencial e a necessidade de processar múltiplas regiões por imagem a tornam inerentemente lenta e computacionalmente intensiva, sendo inadequada para aplicações em tempo real.

**Abordagens de Uma Etapa (Single-Shot Detectors):** A família YOLO (You Only Look Once) e outras como SSD (Single Shot MultiBox Detector) introduziram uma mudança de paradigma radical. Em vez de dividir a detecção em duas tarefas separadas, elas unificam o processo em uma única passagem pela rede neural. A detecção de objetos é reformulada como um **único problema de regressão**.

O fluxo de trabalho é o seguinte:

1. A imagem de entrada é dividida em uma grade (grid) de células, por exemplo, 13x13.
2. A imagem é processada **uma única vez** por uma CNN.
3. Cada célula da grade é responsável por prever a presença de objetos cujo centro cai dentro dela. Para cada objeto, a célula prevê um conjunto de valores simultaneamente:
   * As **coordenadas do bounding box** (x, y, largura, altura).
   * Um **score de confiança** (objectness score), que representa a probabilidade de aquele bounding box conter de fato um objeto.
   * As **probabilidades de classe**, que indicam a chance de o objeto contido no box pertencer a cada uma das classes possíveis (e.g., 80% 'bico', 15% 'olho', etc.).

Ao eliminar a etapa de proposta de região, os detectores de uma etapa alcançam velocidades de inferência ordens de magnitude maiores que seus equivalentes de duas etapas. O nome "You Only Look Once" (Você Só Olha Uma Vez) reflete essa filosofia: a rede olha para a imagem inteira de uma só vez e produz todas as detecções simultaneamente. Embora as primeiras versões do YOLO tivessem uma precisão ligeiramente inferior, especialmente para objetos pequenos, as iterações mais recentes, como o YOLOv8 utilizado neste trabalho, reduziram significativamente essa lacuna, oferecendo um balanço excepcional entre velocidade e acurácia, o que é crucial para um sistema responsivo e escalável.

##### 2.3.2. YOLOv8: Arquitetura e Inovações

O YOLOv8, desenvolvido pela Ultralytics, representa a mais recente evolução da família YOLO, incorporando diversas melhorias arquitetônicas e de treinamento que o estabelecem como um dos modelos de detecção de objetos mais eficientes e precisos da atualidade. Para os propósitos deste trabalho, a escolha pelo YOLOv8 foi motivada por três avanços principais em relação aos seus predecessores: sua abordagem sem âncoras, suas novas funções de perda e sua estrutura flexível e unificada.

**1. Abordagem sem Âncoras (Anchor-Free):** Versões anteriores do YOLO (como YOLOv3 e YOLOv5) eram baseadas em âncoras (*anchor-based*). Essas âncoras são um conjunto de caixas delimitadoras (bounding boxes) de tamanhos e proporções predefinidos, que atuam como "moldes" iniciais. O modelo aprende a prever deslocamentos (offsets) em relação a essas âncoras para se ajustar aos objetos reais na imagem. Embora eficaz, essa abordagem tem desvantagens:

* **Sensibilidade a Hiperparâmetros:** As âncoras precisam ser cuidadosamente escolhidas para o dataset específico (geralmente através de um algoritmo de k-means), tornando o treinamento mais complexo e sensível a essa configuração inicial.
* **Ineficiência:** Um grande número de âncoras é necessário para cobrir a variedade de formas e tamanhos dos objetos, levando a um desequilíbrio entre âncoras positivas (que contêm objetos) e negativas (que contêm fundo), o que pode dificultar o treinamento.

O YOLOv8 adota uma abordagem **sem âncoras (anchor-free)**, alinhando-se a tendências modernas de detectores como FCOS (Fully Convolutional One-Stage Object Detection). Em vez de prever deslocamentos a partir de âncoras, o modelo prevê diretamente o centro do objeto e regride a partir do centro para os quatro lados da caixa (topo, baixo, esquerda, direita). Essa mudança simplifica o pipeline de detecção, elimina a necessidade de otimização de âncoras e melhora a capacidade do modelo de generalizar para objetos com formas e proporções incomuns, como o corpo alongado ou a cauda de um pássaro.

**2. Novas Funções de Perda (Loss Functions):** A função de perda é o mecanismo pelo qual o modelo aprende, quantificando o quão "erradas" estão suas previsões. O YOLOv8 introduz funções de perda mais avançadas para as duas tarefas principais:

* **Perda de Classificação:** Utiliza a *Binary Cross-Entropy* (BCE), que se mostrou eficaz e estável para a tarefa de classificar os objetos detectados.
* **Perda de Regressão (Bounding Box):** Adota a *Distribution Focal Loss* (DFL) e a *Complete Intersection over Union* (CIoU) Loss. A perda CIoU é uma melhoria sobre a IoU tradicional, pois considera não apenas a área de sobreposição entre a caixa prevista e a real, mas também a distância entre seus centros e a consistência de suas proporções. A DFL trata a regressão das coordenadas da caixa como uma distribuição, permitindo que o modelo aprenda de forma mais flexível e robusta a localização precisa dos objetos. Juntas, essas funções de perda resultam em uma localização de objetos mais precisa e um treinamento mais rápido e estável.

**3. Estrutura Flexível e Unificada:** O YOLOv8 foi projetado como uma plataforma unificada, não apenas um detector de objetos. Sua arquitetura permite, com poucas modificações, realizar outras tarefas de visão computacional, como segmentação de instâncias e classificação de imagens. O *backbone* (a espinha dorsal da rede, responsável pela extração de características) foi otimizado, e a "cabeça" do modelo (a parte que faz as previsões) foi desacoplada, permitindo maior modularidade. Para este projeto, essa flexibilidade e a facilidade de uso da API fornecida pela Ultralytics foram decisivas, permitindo um ciclo de treinamento, validação e implantação muito mais ágil e eficiente. A capacidade de escolher entre diferentes tamanhos de modelo (de n - nano a x - extra-large) permite um ajuste fino entre velocidade e precisão, adaptando-se facilmente às restrições de hardware do ambiente de desenvolvimento.

#### 2.4. A Abordagem Neuro-Simbólica

A história da Inteligência Artificial foi marcada por uma dicotomia entre duas escolas de pensamento dominantes: a abordagem **simbólica** e a **conexionista**. A IA simbólica, também conhecida como "Good Old-Fashioned AI" (GOFAI), postula que a inteligência pode ser alcançada através da manipulação de símbolos e regras explícitas. Sistemas baseados em lógica, motores de inferência e grafos de conhecimento são exemplos desta abordagem. Seus pontos fortes são a interpretabilidade, a capacidade de raciocínio abstrato e a representação explícita do conhecimento. No entanto, são sistemas frágeis (*brittle*), que falham catastroficamente ao lidar com a ambiguidade e o ruído do mundo real, e exigem uma custosa engenharia de conhecimento manual.

Por outro lado, a abordagem conexionista, cujo expoente máximo são as redes neurais profundas, argumenta que a inteligência emerge da interação de muitas unidades de processamento simples e interconectadas. Esses sistemas são mestres na percepção e no reconhecimento de padrões a partir de dados brutos e ruidosos, como imagens e som. Sua principal força é a capacidade de aprender representações de características automaticamente. Contudo, suas fraquezas são o oposto da IA simbólica: operam como "caixas-pretas" de difícil interpretação, necessitam de volumes massivos de dados para treinamento e demonstram dificuldades com o raciocínio causal, a composição de conhecimentos e a extrapolação para além dos dados de treinamento.

A **IA Neuro-Simbólica** surge como uma síntese, buscando unir o melhor dos dois mundos. O princípio fundamental não é substituir uma abordagem pela outra, mas integrá-las de forma que as forças de uma compensem as fraquezas da outra. O objetivo é criar sistemas que possam tanto **perceber** o mundo de forma robusta quanto **raciocinar** sobre ele de forma lógica e transparente.

No contexto deste TCC, esta integração é o pilar central da arquitetura:

* **Os Módulos Conexionistas (Percepção):** O classificador MobileNetV2 e o detector YOLOv8 atuam como os "sentidos" do sistema. Eles são a interface com o mundo real, transformando o sinal bruto e contínuo dos pixels em informações discretas e estruturadas. Este processo é conhecido como **"ancoragem de símbolos" (symbol grounding)**, onde o símbolo abstrato 'bico' é conectado a um padrão específico de pixels em uma imagem.
* **O Módulo Simbólico (Raciocínio):** O grafo de conhecimento e o motor de regras implementado em Python representam a camada cognitiva. Este módulo não lida com pixels, mas com os símbolos gerados pela camada de percepção (e.g., 'bico', 'asa', 'Cardinal'). Ele manipula esses símbolos de acordo com regras lógicas predefinidas (como a lógica "fatos primeiro" e o limiar de confiança) e conhecimento explícito (as relações no grafo) para chegar a uma conclusão.

Essa arquitetura híbrida permite ao sistema alcançar um raciocínio mais robusto. Quando o módulo conexionista (o classificador) "alucina" ao ver a imagem de um cão, o módulo simbólico (o motor de regras) pode intervir, pois a camada de percepção (YOLO) não forneceu os símbolos necessários ('bico', 'asa') para satisfazer a premissa lógica de que o objeto é um pássaro. Esta capacidade de validação cruzada entre percepção e lógica é o que confere ao sistema uma robustez e uma capacidade de gerenciamento de incerteza inalcançáveis por qualquer uma das abordagens isoladamente.

#### 2.5. Representação de Conhecimento: Grafos

Enquanto as redes neurais se destacam na extração de padrões de dados brutos, elas não representam o conhecimento de forma explícita ou estruturada. Para que um sistema de IA possa raciocinar de maneira análoga à humana, é necessário um formalismo para organizar fatos, conceitos e, crucialmente, as relações entre eles. Os **Grafos de Conhecimento (Knowledge Graphs)** emergem como uma solução poderosa e flexível para este desafio, servindo como a espinha dorsal do módulo simbólico deste projeto.

Na sua essência, um grafo é uma estrutura de dados fundamental em ciência da computação, composta por dois elementos:

* **Nós (ou Vértices):** Representam as entidades ou conceitos. Em nosso domínio, um nó pode ser uma instância específica ('Cardinal'), uma categoria ('Pássaro'), uma parte anatômica ('Bico'), um atributo ('Formato Cônico') ou uma função ('Quebrar Sementes').
* **Arestas (ou Laços):** Representam as relações entre os nós. As arestas são direcionadas e rotuladas para descrever a natureza da conexão. Por exemplo, uma aresta pode conectar o nó 'Cardinal' ao nó 'Pássaro' com o rótulo 'é\_um', formando a tripla semântica (Cardinal, é\_um, Pássaro).

Ao contrário de estruturas de dados tabulares (como bancos de dados relacionais), que são otimizadas para armazenar registros uniformes, os grafos são inerentemente projetados para capturar a conectividade e as relações complexas e heterogêneas do mundo real. Esta capacidade permite a modelagem de diversos tipos de conhecimento:

* **Hierarquias Taxonômicas:** Relações de é\_um (is-a) ou é\_um\_tipo\_de (type-of) permitem criar hierarquias, como (Painted Bunting, é\_um, Pássaro) e (Pássaro, é\_um, Animal).
* **Relações Mereológicas:** Relações de possui\_parte (has-part) permitem decompor entidades em seus componentes, como (Pássaro, possui\_parte, Asa).
* **Atributos e Propriedades:** Permitem descrever as características de uma entidade, como (Bico, possui\_formato, Cônico) ou (Asa, possui\_função, Voo).
* **Relações Causais e Funcionais:** Onde reside o maior poder para o raciocínio. Um grafo pode codificar que um (Bico Cônico, tem\_função\_para, Quebrar Sementes), o que, por sua vez, implica uma (Dieta, inclui, Sementes).

No sistema neuro-simbólico deste projeto, o grafo de conhecimento atua como a "memória de longo prazo" e o "mecanismo de inferência" da IA. Enquanto o YOLO fornece os "fatos brutos" ('bico detectado'), é a consulta ao grafo que permite ao sistema contextualizar essa informação. Uma consulta pode percorrer o grafo para responder a perguntas complexas como: "Se um pássaro possui um bico cônico, qual é a sua dieta provável?". Esta capacidade de atravessar caminhos relacionais no grafo é o que habilita o sistema a realizar inferências que vão além do que foi explicitamente visto nas imagens de treinamento, formando a base para um raciocínio mais profundo e para a inovação.

#### 2.6. Aprendizado Contínuo e Ativo

O paradigma tradicional de treinamento de modelos de aprendizado de máquina, conhecido como **aprendizado em lote (batch learning)** ou **offline**, envolve treinar o modelo uma única vez em um dataset estático e completo. Uma vez treinado, o modelo é implantado e seu conhecimento permanece congelado. Para que ele aprenda com novos dados, todo o processo de treinamento deve ser refeito do zero, combinando os dados antigos e os novos, o que é computacionalmente caro e ineficiente para sistemas que precisam se adaptar a um ambiente dinâmico.

Para superar essa limitação, surgiram paradigmas de aprendizado que permitem a evolução contínua do modelo.

**Aprendizado Incremental (ou Online Learning):** Neste paradigma, o modelo é capaz de ser atualizado com novos dados à medida que eles chegam, sem a necessidade de retreinar com todo o conjunto de dados. O modelo aprende de forma sequencial, com um ou um pequeno lote de novos exemplos por vez. Isso é ideal para aplicações onde os dados são gerados continuamente. No entanto, o aprendizado incremental enfrenta um desafio significativo conhecido como **esquecimento catastrófico (catastrophic forgetting)**, onde, ao aprender com os novos dados, o modelo pode "esquecer" o conhecimento adquirido dos dados antigos. Técnicas como a repetição de exemplos antigos (rehearsal) ou a regularização de pesos são usadas para mitigar esse problema.

**Aprendizado Ativo (Active Learning):** O aprendizado ativo representa uma forma mais inteligente e eficiente de aprendizado incremental. Em vez de aceitar passivamente todos os novos dados, o modelo se torna um agente ativo no processo de aprendizado. O princípio central é que nem todos os dados são igualmente informativos. O modelo pode aprender muito mais rápido e com menos dados se puder escolher quais exemplos rotular.

O ciclo de aprendizado ativo geralmente funciona da seguinte forma:

1. O modelo é treinado em um pequeno conjunto de dados rotulados.
2. O modelo analisa um grande conjunto de dados não rotulados.
3. Ele utiliza uma **estratégia de consulta (query strategy)** para selecionar os exemplos sobre os quais ele está mais **incerto**. Estratégias comuns incluem selecionar os exemplos cuja previsão de classe teve a menor confiança ou aqueles que se encontram mais próximos da fronteira de decisão do modelo.
4. Esses exemplos selecionados são então apresentados a um "oráculo" (geralmente um especialista humano) para rotulação.
5. Os novos exemplos, agora rotulados, são adicionados ao conjunto de treinamento, e o modelo é retreinado.

Este processo otimiza drasticamente o esforço de rotulação, focando nos exemplos que oferecem o maior ganho de informação para o modelo.

**Human-in-the-Loop (HITL): O Aprendizado Ativo na Prática** O **Human-in-the-Loop** é a implementação prática e simbiótica do aprendizado ativo. Ele cria uma parceria entre a inteligência humana e a artificial, onde cada um foca no que faz de melhor. A IA lida com a análise em larga escala, processando milhares de exemplos e identificando padrões. O especialista humano, por sua vez, é acionado apenas para as tarefas que requerem cognição de alto nível, como lidar com a ambiguidade, o contexto ou a novidade.

No contexto deste TCC, o ciclo de aprendizado contínuo é um sistema HITL por excelência.

* **A IA faz o trabalho pesado:** Analisa as imagens e, através da sua lógica de baixa confiança e falha de detecção, identifica os casos "interessantes" ou "difíceis" – os candidatos a aprendizado.
* **A IA propõe uma solução:** Utilizando o Grad-CAM, ela gera uma anotação inicial, automatizando parte do trabalho de rotulação.
* **O Humano atua como supervisor:** O especialista revisa a proposta da IA, a corrige ou a valida com um esforço mínimo. Esta validação fornece um dado de altíssima qualidade para o sistema.
* **Fechando o ciclo:** O novo dado validado é usado para refinar o modelo (retreinamento incremental), tornando-o mais inteligente para o futuro.

Este paradigma transforma o sistema de uma ferramenta estática em um organismo de aprendizado dinâmico, que se aprimora e expande seu conhecimento através da colaboração, espelhando de forma mais próxima o processo de aprendizado humano.

#### 2.7. Interpretabilidade em IA (XAI): Grad-CAM

À medida que os modelos de Deep Learning se tornam mais complexos e profundos, sua natureza de "caixa-preta" (*black box*) se intensifica, tornando suas decisões opacas para os usuários e desenvolvedores. O campo da **Inteligência Artificial Explicável (Explainable AI - XAI)** busca desenvolver métodos para tornar esses modelos mais transparentes e interpretáveis. A interpretabilidade não é apenas um requisito para a depuração de modelos, mas também é fundamental para estabelecer confiança, garantir a justiça (fairness) e permitir a colaboração humano-máquina, como no ciclo de aprendizado proposto neste trabalho.

Uma das perguntas mais fundamentais em tarefas de visão computacional é: "Para onde o modelo olhou na imagem para tomar sua decisão?". A técnica **Gradient-weighted Class Activation Mapping (Grad-CAM)** é uma das soluções mais poderosas e amplamente utilizadas para responder a essa pergunta, fornecendo uma "explicação" visual para as previsões de uma CNN.

**Intuição por Trás do Grad-CAM:** A intuição é que as camadas convolucionais finais de uma CNN aprendem a detectar conceitos semânticos de alto nível (e.g., "textura de pena", "formato de olho"). Portanto, a decisão final de classificação (e.g., "Cardinal") deve ser baseada na presença desses conceitos na imagem. O Grad-CAM visa criar um mapa de calor que destaque as regiões espaciais na imagem que mais ativaram esses conceitos relevantes para a classe prevista.

**Funcionamento Técnico:** O Grad-CAM opera através de um processo engenhoso que utiliza o fluxo de informação do gradiente durante a retropropagação (*backpropagation*):

1. **Passagem Direta (Forward Pass):** A imagem de entrada é passada pela CNN para obter a pontuação de saída (antes da camada Softmax) para uma classe de interesse, digamos y^c para a classe c (e.g., "Cardinal").
2. **Seleção da Camada Alvo:** A última camada convolucional da rede é selecionada. Esta camada oferece o melhor compromisso entre informação espacial de alta resolução e características semânticas de alto nível. Seja A^k o k-ésimo mapa de características desta camada.
3. **Cálculo do Gradiente:** O gradiente da pontuação da classe y^c é calculado em relação a cada pixel de cada mapa de características A^k da camada alvo. Este gradiente, ∂y^c / ∂A^k, mede a sensibilidade da classe c a cada mapa de características.
4. **Pesos de Importância do Neurônio:** Para determinar a "importância" de cada mapa de características A^k como um todo, o Grad-CAM realiza uma média global dos gradientes (global average pooling) para obter um único peso, α\_k^c. Este peso representa o quão importante o k-ésimo mapa de características foi para a decisão da classe c.
5. **Combinação Linear Ponderada:** Um mapa de calor grosseiro, L^c\_{Grad-CAM}, é gerado calculando uma combinação linear ponderada de todos os mapas de características da camada alvo, onde os pesos são os α\_k^ccalculados na etapa anterior. Uma função ReLU é aplicada a esta combinação para isolar apenas as características que têm uma influência positiva na classe de interesse.
6. **Visualização:** O mapa de calor resultante, que tem as dimensões da camada convolucional final, é redimensionado (upsampled) para as dimensões da imagem original e sobreposto a ela, geralmente com uma paleta de cores que vai do azul (baixa importância) ao vermelho (alta importância).

**Aplicação no Projeto:** Neste TCC, o Grad-CAM desempenha um papel duplo e crucial:

* **Depuração e Confiança:** Ele permite verificar se o classificador está tomando decisões pelas razões certas. Por exemplo, se ao classificar um "Cardinal", o mapa de calor destacar o pássaro vermelho, o modelo está correto. Se destacar uma cor de fundo ou um galho, indica que o modelo aprendeu uma correlação espúria.
* **Mecanismo de Auto-Anotação:** É a tecnologia central que habilita o ciclo de aprendizado. Quando o YOLO falha, mas o classificador tem uma "intuição" (baixa confiança), o Grad-CAM fornece a **localização espacial** dessa intuição. Ao identificar a região mais "quente" do mapa de calor, o sistema pode propor um bounding box para um objeto que ele não conseguiu detectar formalmente, transformando uma falha de detecção em uma oportunidade de aprendizado

#### 2.8. Modelos de Linguagem e Visão de Larga Escala (LLMs/LVMs)

Introdução aos modelos fundacionais como Gemini e GPT-4V, que possuem uma compreensão semântica do mundo a partir de treinamento em vastos datasets, e seu potencial como "oráculos" ou supervisores em sistemas de IA mais específicos.

<div style="page-break-after: always;"></div>

### 3. Desenvolvimento e Metodologia

*(Esta seção deve ser extremamente detalhada, com trechos de código, prints de tela e resultados de cada etapa, atingindo aproximadamente 20-25 páginas)*

#### 3.1. Ambiente de Desenvolvimento e Ferramentas

* **Hardware:** CPU Intel Core i7, 16GB RAM, GPU NVIDIA GeForce RTX 3060
* **Software:** Windows 11, Python 3.11, VS Code
* **Bibliotecas Principais:** TensorFlow 2.15, OpenCV 4.8, Ultralytics 8.0, NetworkX 3.2, Pandas, Matplotlib.

#### 3.2. Coleta e Preparação do Conjunto de Dados (Dataset)

O trabalho utilizou como base o dataset público CUB-200-2011, que contém 11.788 imagens de 200 espécies de aves. Para o escopo deste protótipo, foram selecionadas 3 espécies iniciais (Painted Bunting, Cardinal, Brown Pelican), e um subconjunto de imagens foi utilizado para o treinamento dos modelos.

#### 3.3. Módulo 1: Classificador de Espécies (Keras e MobileNetV2)

##### 3.3.1. Pré-processamento e Aumento de Dados

O pré-processamento incluiu o redimensionamento das imagens para 224x224 pixels e a normalização dos valores dos pixels para o intervalo [0, 1]. Foi aplicado um aumento de dados (Data Augmentation) para expandir artificialmente o dataset e reduzir o overfitting, incluindo rotações, zoom e inversões horizontais.

##### 3.3.2. Arquitetura do Modelo e Treinamento

Foi utilizada a arquitetura MobileNetV2 como base, com seus pesos pré-treinados na ImageNet. As camadas de topo foram substituídas por novas camadas de classificação, incluindo GlobalAveragePooling2D, BatchNormalization, Dropout e uma camada Dense final com ativação softmax.

##### 3.3.3. Resultados do Treinamento

O modelo foi treinado por 30 épocas, alcançando uma acurácia de validação de aproximadamente XX%.

[DESCRIÇÃO DA IMAGEM: Insira aqui os gráficos gerados pelo TensorFlow/Keras durante o treinamento do seu classificador. A imagem deve conter dois subplots:

1. **Acurácia de Treinamento e Validação por Época:** Um gráfico de linhas mostrando a 'Acurácia' no eixo Y e 'Época' no eixo X. Deve haver duas linhas, uma para a acurácia de treino (geralmente azul) e outra para a acurácia de validação (geralmente laranja), com uma legenda clara.
2. Perda de Treinamento e Validação por Época: Um segundo gráfico de linhas mostrando a 'Perda (Loss)' no eixo Y e 'Época' no eixo X. Da mesma forma, deve haver duas linhas, uma para a perda de treino e outra para a de validação.  
   Estes gráficos são essenciais para discutir se o modelo aprendeu bem ou se sofreu de overfitting.]

*Figura 2: Gráficos de Acurácia e Perda durante o treinamento do modelo classificador MobileNetV2.*

#### 3.4. Módulo 2: Detector de Partes Anatômicas (YOLOv8)

##### 3.4.1. Processo de Anotação

Um conjunto de 72 imagens foi manualmente anotado utilizando a ferramenta LabelImg. Foram definidas 6 classes: olho, bico, asa, garra, cauda, corpo. As anotações foram salvas no formato XML PASCAL VOC e posteriormente convertidas para o formato de texto exigido pelo YOLO.

##### 3.4.2. Estrutura do Dataset e Configuração

O dataset foi dividido em conjuntos de treino (80%) e validação (20%). Foi criado um arquivo data.yaml para especificar os caminhos e os nomes das classes para o YOLOv8.

##### 3.4.3. Treinamento e Avaliação do Modelo

O modelo yolov8n.pt pré-treinado foi ajustado com o dataset customizado por 250 épocas.

[DESCRIÇÃO DA IMAGEM: Insira aqui os outputs visuais gerados pela biblioteca Ultralytics na pasta runs/train/ após o treinamento do YOLOv8. Uma boa figura combinaria dois resultados:

1. **Matriz de Confusão:** O gráfico de matriz de confusão (confusion\_matrix.png) que mostra com que frequência o modelo confundiu uma classe com outra (ou com o fundo).
2. **Exemplos de Predição:** Uma grade de imagens do lote de validação (val\_batch0\_pred.jpg) que mostra as imagens de teste com os bounding boxes desenhados pelo modelo treinado, mostrando visualmente a performance da detecção.]

*Figura 3: Resultados do treinamento do detector YOLOv8. (A) Matriz de Confusão e (B) Exemplos de detecção no conjunto de validação.*

#### 3.5. Módulo 3: Motor de Raciocínio e Grafo de Conhecimento

##### 3.5.1. Modelagem do Grafo

O grafo foi implementado com a biblioteca networkx, contendo nós para espécies, partes anatômicas e conceitos (e.g., "Pássaro"), e arestas para representar relações como é\_um e possui\_parte.

##### 3.5.2. Implementação da Lógica "Fatos Primeiro"

Foi desenvolvido um algoritmo em Python que orquestra a execução dos módulos. Primeiramente, o YOLO é invocado para extrair fatos. Apenas se fatos conclusivos (e.g., bico detectado) forem encontrados, o classificador Keras é utilizado para gerar uma hipótese, que é então avaliada com base no limiar de confiança.

<div style="page-break-after: always;"></div>

### 4. O Ciclo de Inovação e Aprendizado Contínuo

*(Esta seção aprofunda os conceitos mais avançados, com fluxogramas e exemplos, atingindo aproximadamente 10 páginas)*

#### 4.1. Lidando com a Novidade: O Desafio das Imagens Desconhecidas

Discussão sobre como a lógica "fatos primeiro" e o limiar de confiança permitem ao sistema não apenas falhar graciosamente, mas ativamente identificar um objeto como um "pássaro de espécie desconhecida".

#### 4.2. Implementação do Módulo de Auto-Anotação com Grad-CAM

Demonstração passo a passo:

1. Uma imagem de pássaro desconhecida é recebida.
2. O YOLO falha. O Keras retorna uma classe com confiança mediana (e.g., 45%).
3. O Grad-CAM é aplicado sobre a última camada convolucional do MobileNetV2.
4. Um mapa de calor é gerado, destacando a silhueta do pássaro.
5. O OpenCV é usado para encontrar o contorno da maior região "quente" e gerar um bounding box.
6. Este bounding box é salvo como uma anotação da classe corpo.

[DESCRIÇÃO DA IMAGEM: Crie uma imagem composta para ilustrar este processo. Ela deve ter 3 painéis, da esquerda para a direita:

1. **(A) Imagem Original:** A imagem de um pássaro que o YOLO não conseguiu detectar (como a sua imagem1.jpeg).
2. **(B) Mapa de Calor Grad-CAM:** A mesma imagem, mas com a sobreposição do mapa de calor gerado pelo Grad-CAM, mostrando as áreas em tons quentes (vermelho/amarelo) que o classificador "olhou" para fazer sua predição.
3. **(C) Anotação Proposta:** A imagem original novamente, mas agora com o bounding box retangular desenhado ao redor da área destacada pelo mapa de calor, com o rótulo "corpo (proposto)".]

*Figura 4: Fluxo visual do processo de Auto-Anotação via Grad-CAM para um candidato a aprendizado.*

#### 4.3. Proposta de Validação Automatizada com API de Visão

Um fluxograma detalhado mostra o ciclo de validação: a anotação proposta é enviada para uma API de visão (Gemini/GPT-4V) com um prompt "O objeto principal nesta imagem é um pássaro? Responda 'Sim' ou 'Não'". Com base na resposta e na qualidade do mapa de calor, a anotação é aprovada, rejeitada ou enviada para revisão humana.

[DESCRIÇÃO DA IMAGEM: Crie um fluxograma detalhado para este ciclo.

* **Início:** 'Candidato a Aprendizado (Imagem + Anotação Proposta)'.
* **Processo:** Seta para uma caixa 'Enviar Imagem para API de Visão Externa'.
* **Decisão:** Um losango de decisão 'API Respondeu "Sim"?'.
* **Caminho 'Não':** Seta para uma caixa 'Rejeitar Anotação e Descartar'.
* **Caminho 'Sim':** Seta para outro losango de decisão 'Qualidade do Grad-CAM > Limiar?'.
* **Caminho 'Sim' (do segundo losango):** Seta para uma caixa 'Aprovar Anotação e Mover para Dataset de Treino'.
* **Caminho 'Não' (do segundo losango):** Seta para uma caixa 'Mover para Pasta de Revisão Humana'.
* **Fim:** As caixas de 'Rejeitar', 'Aprovar' e 'Revisão Humana' são os pontos finais do fluxo.]

*Figura 5: Fluxograma do Ciclo de Validação Automatizada e "Human-in-the-Loop".*

#### 4.4. O Conceito de Inovação: Geração de Novas Espécies

Exemplo prático do fluxo de inovação, desde o prompt do usuário até a consulta ao grafo e a geração do "blueprint" lógico da nova espécie.

<div style="page-break-after: always;"></div>

### 5. Resultados e Discussão

*(Esta seção apresenta os testes práticos com as suas imagens, atingindo aproximadamente 5-7 páginas)*

#### 5.1. Cenário de Teste 1: Espécie Conhecida e de Alta Confiança

[DESCRIÇÃO DA IMAGEM: Insira aqui a imagem de um pássaro de uma das 3 espécies que você treinou (ex: um Painted Bunting claro). Ao lado ou abaixo da imagem, coloque um box de texto com a saída exata do seu script para essa imagem, mostrando a alta confiança na classificação e os fatos detectados pelo YOLO.]

*Figura 6: Resultado do sistema para uma espécie conhecida (Painted Bunting).*

#### 5.2. Cenário de Teste 2: Espécie Desconhecida (Pássaro Válido)

[DESCRIÇÃO DA IMAGEM: Insira aqui a sua imagem1.jpeg (o pássaro laranja e preto). Ao lado ou abaixo, coloque o box de texto com a saída do script, mostrando que o YOLO (após retreinado) encontrou fatos, mas o classificador teve baixa confiança, resultando na classificação "Pássaro Desconhecido #1".]

*Figura 7: Resultado do sistema para um pássaro válido, mas de espécie desconhecida.*

#### 5.3. Cenário de Teste 3: Objeto Inválido (Não-Pássaro)

[DESCRIÇÃO DA IMAGEM: Insira aqui a sua imagem4.jpeg (o cão da raça Pug). Ao lado ou abaixo, coloque o box de texto com a saída do script, mostrando que o YOLO não encontrou fatos e a análise foi encerrada, provando a eficácia da lógica "fatos primeiro".]

*Figura 8: Resultado do sistema para uma imagem de um não-pássaro.*

#### 5.4. Discussão dos Resultados e Limitações

A principal limitação do protótipo atual reside na acurácia do detector YOLOv8, que ainda é dependente da qualidade e quantidade do seu dataset de treinamento. O processo de auto-anotação com Grad-CAM, embora promissor, pode gerar bounding boxes imprecisos, o que justifica a necessidade do ciclo de validação.

<div style="page-break-after: always;"></div>

### 6. Conclusão

#### 6.1. Síntese do Trabalho e Contribuições

Este trabalho demonstrou com sucesso a concepção e implementação de um protótipo de IA neuro-simbólica que vai além da classificação de padrões. Ao integrar múltiplos modelos especialistas e uma lógica de raciocínio cética, o sistema é capaz de gerenciar a incerteza, distinguir entre o conhecido e o novo, e possui uma arquitetura escalável para aprendizado contínuo. A principal contribuição deste TCC é a demonstração prática de como a combinação de diferentes paradigmas de IA pode levar a sistemas mais robustos, autônomos e alinhados com o raciocínio humano.

#### 6.2. Trabalhos Futuros

* **Expansão Massiva do Dataset:** Aumentar significativamente o número de imagens e classes no dataset do YOLOv8.
* **Refinamento do Grafo de Conhecimento:** Adicionar relações mais complexas, como dieta, habitat e comportamentos.
* **Implementação Completa do Ciclo de Aprendizado:** Integrar de fato a API de visão para validação e automatizar o processo de retreinamento.
* **Desenvolvimento de Interface de Usuário:** Criar uma interface gráfica para facilitar a interação e o processo de revisão humana.

#### 6.3. Considerações Finais

O caminho para uma Inteligência Artificial Geral é longo e complexo. No entanto, projetos como este, que focam na integração de percepção e raciocínio, representam passos cruciais nessa jornada. O sistema desenvolvido, embora um protótipo, serve como uma prova de conceito poderosa de que é possível construir IAs que não apenas respondem, mas que começam a "entender".

<div style="page-break-after: always;"></div>

### Referências

*(Adicionar referências formais no formato ABNT)*

1. REDMON, Joseph; FARHADI, Ali. **YOLOv3: An Incremental Improvement**. arXiv preprint arXiv:1804.02767, 2018.
2. SANDLER, Mark et al. **MobileNetV2: Inverted Residuals and Linear Bottlenecks**. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2018. p. 4510-4520.
3. PEARL, Judea. **Causality: Models, Reasoning, and Inference**. Cambridge university press, 2009.
4. Jocher, G. et al. (2023). **Ultralytics YOLOv8**. Disponível em: https://github.com/ultralytics/ultralytics. Acesso em: 27 ago. 2025.
5. SELVARAJU, Ramprasaath R. et al. **Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization**. In: Proceedings of the IEEE international conference on computer vision. 2017. p. 618-626.

<div style="page-break-after: always;"></div>

### Apêndices

#### Apêndice A: Código-Fonte Principal (main.py)

import cv2  
import numpy as np  
import os  
import tensorflow as tf  
import logging  
import json  
from ultralytics import YOLO  
  
logging.basicConfig(level=logging.INFO)  
  
# --- CONFIGURAÇÃO ---  
CLASSIFIER\_MODEL\_PATH = 'modelo\_classificacao\_passaros.keras'  
YOLO\_MODEL\_PATH = 'runs/train/exp\_passaros\_01/weights/best.pt'  
CLASSES\_FILE\_PATH = 'data/classes.txt'  
KNOWLEDGE\_BASE\_PATH = 'base\_conhecimento.json'  
CONFIDENCE\_THRESHOLD = 0.60 # 60%  
  
# --- CARREGAMENTO DOS MODELOS ---  
logging.info("Carregando modelo de classificação de espécies...")  
modelo\_especies = tf.keras.models.load\_model(CLASSIFIER\_MODEL\_PATH)  
logging.info(f"Carregando modelo de detecção de partes (YOLOv8) de '{YOLO\_MODEL\_PATH}'...")  
modelo\_yolo = YOLO(YOLO\_MODEL\_PATH)  
  
base\_conhecimento = {"especies\_conhecidas": {}, "especies\_novas": []}  
  
def salvar\_base\_conhecimento(caminho=KNOWLEDGE\_BASE\_PATH):  
 with open(caminho, 'w') as f:  
 json.dump(base\_conhecimento, f, indent=4, sort\_keys=True)  
  
def carregar\_base\_conhecimento(caminho=KNOWLEDGE\_BASE\_PATH):  
 global base\_conhecimento  
 if os.path.exists(caminho):  
 try:  
 with open(caminho, 'r') as f:  
 base\_conhecimento = json.load(f)  
 logging.info("Base de conhecimento carregada com sucesso.")  
 except json.JSONDecodeError:  
 logging.error("Erro ao decodificar base\_conhecimento.json. Criando uma nova.")  
 base\_conhecimento = {"especies\_conhecidas": {}, "especies\_novas": []}  
 else:  
 logging.warning("Base de conhecimento não encontrada, criando uma nova.")  
  
 if 'especies\_conhecidas' not in base\_conhecimento:  
 base\_conhecimento['especies\_conhecidas'] = {}  
 if 'especies\_novas' not in base\_conhecimento:  
 base\_conhecimento['especies\_novas'] = []  
  
def identificar\_especie(imagem):  
 img\_resized = cv2.resize(imagem, (224, 224))  
 img\_array = np.expand\_dims(img\_resized, axis=0) / 255.0  
 predicao = modelo\_especies.predict(img\_array, verbose=0)  
 id\_classe = np.argmax(predicao[0])  
 confianca = np.max(predicao[0])  
 return id\_classe, confianca  
  
def carregar\_nomes\_classes(caminho=CLASSES\_FILE\_PATH):  
 with open(caminho, 'r') as f:  
 nomes = [' '.join(line.strip().split()[1:]).replace('\_', ' ') for line in f]  
 return nomes  
  
def extrair\_fatos\_com\_yolo(imagem):  
 results = modelo\_yolo(imagem, verbose=False)  
 fatos\_detectados = set()  
 for r in results:  
 for box in r.boxes:  
 class\_id = int(box.cls)  
 class\_name = modelo\_yolo.names[class\_id]  
 fatos\_detectados.add(class\_name)  
 return fatos\_detectados  
  
def analisar\_imagem\_passaro(caminho\_imagem):  
 imagem = cv2.imread(caminho\_imagem)  
 if imagem is None:  
 logging.error(f"Erro ao carregar a imagem: {caminho\_imagem}")  
 return  
  
 print("-" \* 50)  
 print(f"Analisando Imagem: {os.path.basename(caminho\_imagem)}")  
   
 fatos\_visuais = extrair\_fatos\_com\_yolo(imagem)  
 is\_bird = 'bico' in fatos\_visuais or ('corpo' in fatos\_visuais and 'asa' in fatos\_visuais)  
  
 if not is\_bird:  
 print("CONCLUSÃO: Não foi possível confirmar que é um pássaro.")  
 print(f"Fatos Detectados: {list(fatos\_visuais) if fatos\_visuais else 'Nenhum'}")  
 print("Análise encerrada para esta imagem.")  
 print("-" \* 50)  
 return  
  
 print(f"Fatos Visuais Detectados: {list(fatos\_visuais)}")  
 print("CONCLUSÃO PRELIMINAR: A imagem contém um pássaro.")  
  
 id\_especie, confianca = identificar\_especie(imagem)  
 nomes\_classes = carregar\_nomes\_classes()  
 nome\_hipotese = nomes\_classes[id\_especie]  
  
 if confianca >= CONFIDENCE\_THRESHOLD:  
 especie\_final = nome\_hipotese  
 print(f"ESPÉCIE (ALTA CONFIANÇA): '{especie\_final}' com {confianca:.2%} de certeza.")  
 base\_conhecimento["especies\_conhecidas"][especie\_final] = list(fatos\_visuais)  
 else:  
 especie\_final = f"Pássaro Desconhecido #{len(base\_conhecimento['especies\_novas']) + 1}"  
 print(f"ESPÉCIE (BAIXA CONFIANÇA): É um pássaro, mas a espécie não foi reconhecida com certeza.")  
 print(f"(Melhor palpite foi '{nome\_hipotese}' com apenas {confianca:.2%} de confiança)")  
   
 nova\_especie\_info = {  
 "id\_temporario": especie\_final,  
 "fatos": list(fatos\_visuais),  
 "palpite\_classificador": nome\_hipotese,  
 "confianca\_palpite": f"{confianca:.2%}"  
 }  
 base\_conhecimento["especies\_novas"].append(nova\_especie\_info)  
  
 salvar\_base\_conhecimento()  
 print("-" \* 50)  
  
  
def processar\_diretorio(diretorio):  
 for arquivo in os.listdir(diretorio):  
 if arquivo.lower().endswith((".jpeg", ".jpg", ".png")):  
 analisar\_imagem\_passaro(os.path.join(diretorio, arquivo))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 carregar\_base\_conhecimento()  
 diretorio\_de\_teste = './dataset\_teste'  
 if not os.path.exists(diretorio\_de\_teste):  
 print(f"\nAVISO: O diretório de teste '{diretorio\_de\_teste}' não foi encontrado.")  
 else:  
 processar\_diretorio(diretorio\_de\_teste)

<div style="page-break-after: always;"></div>

#### Apêndice B: Script de Treinamento do YOLOv8 (train\_yolo.py)

from ultralytics import YOLO  
  
# Carrega o modelo pré-treinado yolov8n ('n' de nano)  
# O YOLO vai baixar este modelo na primeira vez que você executar.  
model = YOLO('yolov8n.pt')  
  
# Inicia o treinamento do modelo  
# Certifique-se de que o caminho para o arquivo .yaml está correto  
results = model.train(  
 data='passaros\_parts.yaml',  
 epochs=250, # Aumentado para melhor generalização  
 imgsz=640,  
 project='runs/train', # Opcional: define a pasta raiz para salvar os resultados  
 name='exp\_passaros\_01' # Opcional: define o nome da pasta do experimento  
)  
  
print("Treinamento concluído!")  
print("O melhor modelo foi salvo em: ", results.save\_dir)