ATIVIDADE 05 – PRÉ-PROCESSAMENTO, SEGMENTAÇÃO E DETECÇÃO/CLASSIFICAÇÃO DE IMAGENS

Aluno: Adriano Paula de Araújo
Data: 25/10/2025

Introdução

O processamento digital de imagens é um campo interdisciplinar que combina computação, matemática e inteligência artificial com o objetivo de extrair informações úteis de imagens digitais. Essa área é a base da Visão Computacional, um ramo da inteligência artificial que permite às máquinas perceber, interpretar e tomar decisões a partir de dados visuais, simulando a capacidade humana de enxergar e compreender o mundo ao redor.

Com o avanço das câmeras, sensores e redes neurais profundas (Deep Learning), tornou-se possível desenvolver sistemas capazes de reconhecer padrões complexos e realizar tarefas visuais com precisão comparável — e em alguns casos superior — à humana. Essa tecnologia é amplamente aplicada em áreas como reconhecimento facial, diagnóstico médico por imagem, segurança pública, agricultura de precisão, automação industrial e veículos autônomos.

Neste contexto, três pilares fundamentais sustentam os sistemas modernos de visão computacional: pré-processamento, segmentação e detecção/classificação de imagens. Cada uma dessas etapas desempenha um papel essencial na transformação de imagens brutas em informações estruturadas e acionáveis.

O pré-processamento é a etapa inicial e crítica, responsável por preparar e limpar as imagens antes que sejam utilizadas em modelos de Machine Learning (ML). Seu objetivo é garantir que os algoritmos recebam dados padronizados e livres de ruídos ou distorções que possam comprometer o desempenho. Entre as técnicas mais comuns estão o redimensionamento, normalização, remoção de ruído e equalização de histograma.

A seguir, a segmentação de imagens permite dividir a imagem em regiões ou objetos de interesse, facilitando a análise de partes específicas. Por fim, a detecção e classificação são responsáveis por identificar e rotular objetos presentes na imagem, permitindo que o sistema compreenda o conteúdo visual de forma estruturada.

1. Pré-processamento de Imagens

O pré-processamento de imagens é a primeira e mais importante etapa da Visão Computacional, responsável por preparar e otimizar os dados visuais antes que sejam submetidos a algoritmos de Machine Learning ou Deep Learning.

Seu principal objetivo é garantir que as imagens estejam padronizadas, limpas e consistentes, reduzindo ruídos e distorções que possam comprometer a precisão dos modelos de análise.

Em sistemas de inteligência artificial, essa etapa é essencial para melhorar o desempenho, a generalização e a estabilidade do aprendizado.

As imagens capturadas no mundo real podem apresentar variações de iluminação, contraste, ruído, ângulo ou tamanho, e o pré-processamento corrige esses fatores, tornando o conjunto de dados mais uniforme e adequado para treinamento.

Entre as principais tarefas realizadas nessa fase estão:

- Redimensionamento (Resizing): ajusta todas as imagens para dimensões padronizadas (como 224×224 pixels), garantindo compatibilidade com redes neurais convolucionais (CNNs) modernas.
- Normalização e Padronização: transforma os valores de pixel (0–255) em uma escala entre 0 e 1 ou aplica normalização z-score, reduzindo discrepâncias e acelerando a convergência do aprendizado.
- Remoção de ruído: utiliza filtros como Gaussiano, Mediano ou Bilateral para eliminar interferências visuais sem perder detalhes importantes.
- Equalização de histograma: redistribui os níveis de intensidade, melhorando o contraste em imagens com iluminação desigual.
- CLAHE (Contrast Limited Adaptive Histogram Equalization): técnica avançada que corrige contrastes locais, ideal para ambientes com iluminação variável.
- Conversão de cores: muda o espaço de cor da imagem (RGB → HSV, LAB ou Grayscale) para destacar informações específicas conforme o tipo de análise.
- Aumento de Dados (Data Augmentation): gera versões artificiais da imagem original por rotação, espelhamento, corte ou zoom, ampliando o volume e a diversidade do conjunto de dados e reduzindo o overfitting.

Essas operações garantem que o algoritmo receba dados consistentes e otimizados, permitindo que ele aprenda padrões reais e não ruídos aleatórios.

Técnicas e Componentes fundamentais

1. Redimensionamento (Resizing):
   Padroniza o tamanho das imagens. Essencial para modelos como VGG16, ResNet e MobileNet, que exigem dimensões fixas.

2. Normalização / Padronização:
   Escala os valores de pixel entre 0 e 1 ou ajusta pela média e desvio padrão, prevenindo o estouro de gradiente e acelerando o backpropagation.

3. Aumento de Dados (Data Augmentation):
   Amplia o conjunto de treinamento com variações artificiais, aumentando a robustez do modelo.

4. Remoção de Ruído:
   Filtros de suavização (como o Gaussian Blur) reduzem interferências provocadas por sensores ou compressão de imagem.

5. Equalização de Histograma e CLAHE:
   Corrigem o brilho e o contraste global ou local, melhorando a nitidez visual e o desempenho da segmentação posterior.

Frameworks e Ferramentas essenciais

- OpenCV (cv2): principal biblioteca para manipulação de imagens e operações de baixo nível, como leitura, conversão de cor, redimensionamento e aplicação de filtros.

- Scikit-Image: fornece ferramentas para processamento científico de imagens integradas ao NumPy.

- Albumentations: framework moderno e eficiente para data augmentation em tempo real, amplamente usado em treinamentos com PyTorch e TensorFlow.

- TensorFlow / Keras e PyTorch: oferecem módulos nativos (como preprocess_input e torchvision.transforms) para normalização e aumentos de dados automáticos durante o treinamento.

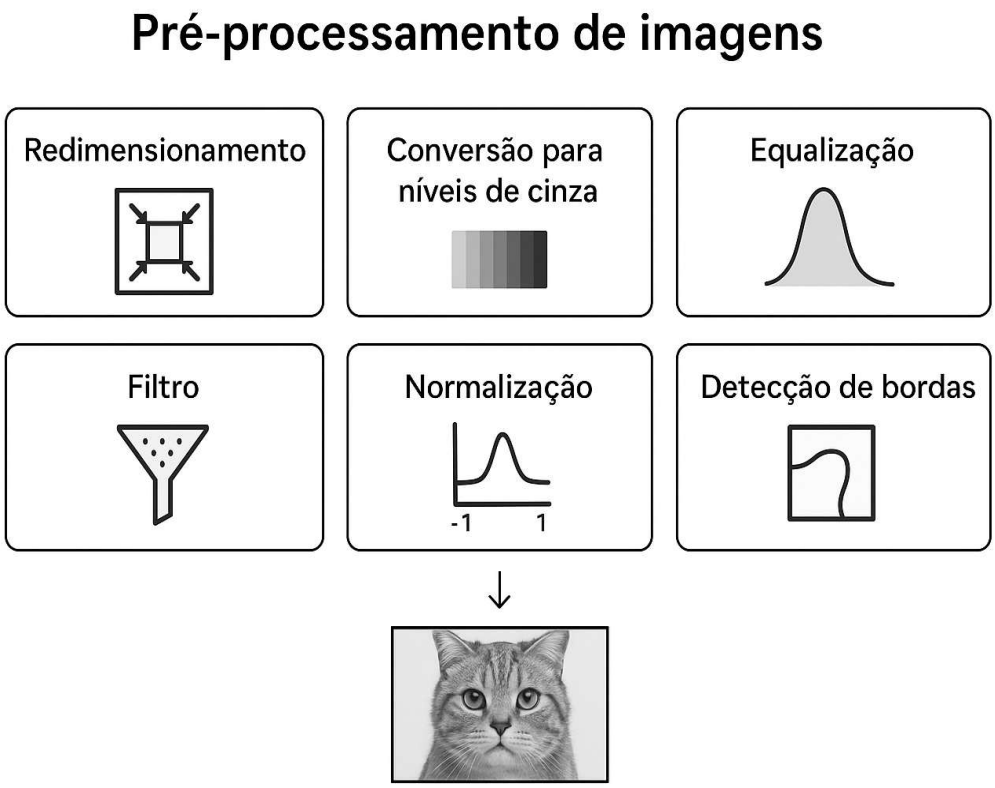
Exemplo Prático 1 — Pré-processamento Básico (Keras + NumPy)

```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# 1. Carregar e redimensionar a imagem (ex: 224x224)
caminho_img = "imagem_exemplo.jpg"
img = load_img(caminho_img, target_size=(224, 224))

# 2. Converter a imagem para um array NumPy
img_array = img_to_array(img)

# 3. Normalizar valores de pixel (0-255 → 0-1)
img_normalizada = img_array / 255.0

print(f"Dimensões: {img_normalizada.shape}")
print(f"Máximo valor após normalização: {np.max(img_normalizada):.2f}")
```

Exemplo Prático 2 — Remoção de Ruído e Detecção de Bordas (OpenCV)

```python
import cv2

# Carrega a imagem original
img = cv2.imread('imagem.jpg')

# 1. Conversão para tons de cinza (reduz complexidade)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 2. Equalização de histograma (melhora o contraste)
img_eq = cv2.equalizeHist(img_gray)

# 3. Aplicação de filtro Gaussiano (remoção de ruído)
img_blur = cv2.GaussianBlur(img_eq, (5, 5), 0)

# 4. Detecção de bordas com Canny
bordas = cv2.Canny(img_blur, 100, 200)
```

Figura 1 – Etapas do pré-processamento de imagens.



O sucesso de qualquer modelo de visão computacional depende diretamente da qualidade das imagens que alimentam o algoritmo.
Um pré-processamento eficaz garante:

- Melhor convergência durante o treinamento;
- Maior estabilidade e precisão nas previsões;
- Menor tempo de processamento;
- E maior generalização do modelo em diferentes contextos.

Assim, o pré-processamento atua como a fundação da inteligência visual, assegurando que o aprendizado da máquina seja confiável, eficiente e representativo do mundo real.

2. Segmentação de Imagens

A segmentação de imagens é uma das etapas mais importantes da visão computacional, sendo responsável por dividir uma imagem em regiões significativas ou objetos de interesse.
O principal objetivo dessa técnica é isolar partes específicas da imagem para posterior análise, reconhecimento ou medição.

Diferentemente da classificação global, a segmentação atribui um rótulo (label) a cada pixel, resultando em um mapa de segmentação que identifica com precisão quais partes pertencem a cada objeto ou classe.

Essa técnica é amplamente utilizada em áreas como diagnóstico médico (delimitação de tumores e lesões), veículos autônomos (detecção de pedestres e faixas), agricultura de precisão (análise de lavouras e pragas) e inspeção industrial (detecção de falhas e defeitos em produtos).

2.1 Tipos de Segmentação

A segmentação pode ser dividida em três categorias principais:

- Segmentação Semântica: cada pixel é classificado como pertencente a uma classe específica (por exemplo, carro, estrada, pessoa). Todos os objetos da mesma classe são tratados como uma única região.

- Segmentação por Instância: além de classificar, diferencia cada objeto individual da mesma classe (por exemplo, carro 1, carro 2). Modelos como Mask R-CNN são amplamente usados nesse tipo de tarefa.

- Segmentação Panóptica: combina as duas abordagens anteriores, permitindo identificar tanto as classes quanto suas instâncias de forma simultânea.

2.2 Técnicas Clássicas de Segmentação

Antes da popularização das redes neurais, diversos métodos tradicionais já eram usados para separar regiões em imagens:

- Limiarização (Thresholding): separa os objetos com base na intensidade dos pixels. O método de Otsu define automaticamente o limiar ideal para dividir fundo e objeto.
- Watershed: técnica baseada na topografia da imagem, onde regiões de interesse são "inundadas" até encontrar fronteiras naturais.
- Crescimento de Regiões (Region Growing): parte de sementes iniciais e expande as áreas vizinhas com base em critérios de similaridade.
- K-Means: algoritmo de agrupamento (clustering) que divide os pixels da imagem em k grupos de cores ou intensidades semelhantes.

Esses métodos ainda são amplamente utilizados em sistemas embarcados e aplicações que demandam baixo custo computacional.

2.3 Modelos Modernos de Segmentação com Deep Learning

O uso de redes neurais convolucionais (CNNs) revolucionou a segmentação de imagens. Modelos avançados conseguem aprender representações visuais complexas e realizar segmentações extremamente precisas, mesmo em cenários com ruídos e iluminação irregular.

Principais modelos modernos:

- **U-Net:** desenvolvida para segmentação biomédica, sua arquitetura em forma de “U” combina camadas de codificação (extração de características) e decodificação (reconstrução da máscara), conectadas por skip connections que preservam detalhes finos.
- **Mask R-CNN:** extensão do Faster R-CNN que combina detecção de objetos e segmentação por instância na mesma rede.
- **DeepLabv3+:** usa convoluções dilatadas e encoder-decoder para capturar contextos globais com precisão.
- **Segment Anything Model (SAM – Meta AI, 2023):** um modelo universal capaz de segmentar qualquer objeto em uma imagem sem necessidade de re-treinamento específico.

Esses modelos têm aplicações em mapeamento urbano, medicina, agricultura e inspeção automatizada, mostrando a versatilidade das abordagens modernas.

2.3 Frameworks e Bibliotecas Utilizadas

- **OpenCV (cv2):** fornece funções para limiarização, watershed, filtragem e operações morfológicas. Ideal para implementações clássicas.
- **TensorFlow / Keras:** utilizados para desenvolver e treinar modelos baseados em U-Net e DeepLab.
- **PyTorch:** framework flexível, ideal para pesquisas e aplicações customizadas.
- **Detectron2**: biblioteca desenvolvida pela Meta (Facebook AI) que implementa modelos avançados como Mask R-CNN, Panoptic FPN e RetinaNet.
- **Scikit-Image:** oferece funções matemáticas e de segmentação clássica, integradas ao ecossistema NumPy.

Essas ferramentas permitem desde a segmentação simples até a criação de modelos de aprendizado profundo otimizados para tarefas específicas.

2.4 Exemplo de Segmentação Clássica com OpenCV

O exemplo abaixo ilustra o processo básico de limiarização binária com o OpenCV, separando o objeto do fundo em tons de cinza.

```python
# Segmentação Clássica: Limiarização (Thresholding) com OpenCV
import cv2

# 1. Carregar a imagem em tons de cinza
caminho_img = "imagem_para_segmentar.png"
img_cinza = cv2.imread(caminho_img, cv2.IMREAD_GRAYSCALE)

# 2. Aplicar limiar (Threshold)
limiar_valor = 127
(T, img_binaria) = cv2.threshold(img_cinza, limiar_valor, 255, cv2.THRESH_BINARY)

# 3. Salvar ou exibir o resultado
cv2.imwrite("mascara_segmentada.png", img_binaria)
```

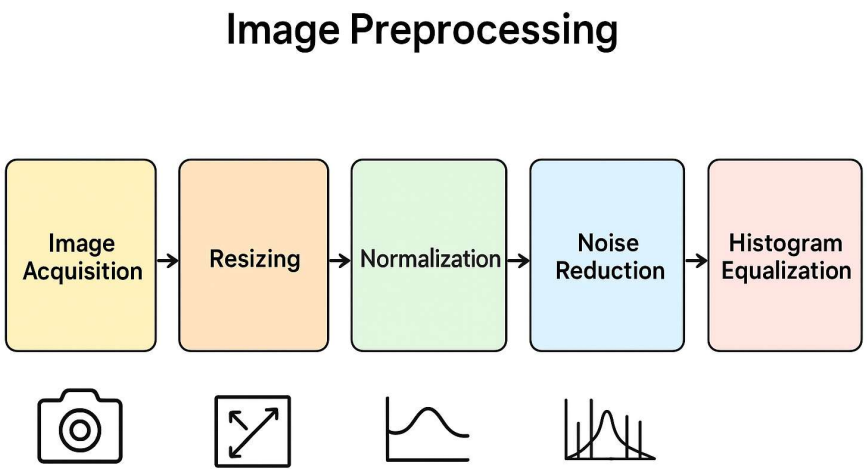
2.6 Exemplo com Limiarização Automática (Método de Otsu)

```python
from skimage.filters import threshold_otsu
import cv2
import matplotlib.pyplot as plt

# Carregar a imagem e converter para tons de cinza
img = cv2.imread("imagem_teste.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Determinar o limiar automaticamente
thresh_val = threshold_otsu(gray)

# Aplicar o limiar
binary = gray > thresh_val

# Exibir resultado
plt.imshow(binary, cmap='gray')
plt.title("Segmentação Automática - Método de Otsu")
plt.show()
```

2.7 Fluxo Visual

A Figura 2 representa a segmentação de uma imagem em regiões distintas, demonstrando visualmente a separação entre diferentes classes e objetos.

Exemplo de segmentação de imagem em regiões distintas.



2.8 Importância da Segmentação

A segmentação de imagens é um dos pilares da visão computacional moderna, pois permite transformar imagens complexas em estruturas compreensíveis por algoritmos.
Sem essa etapa, modelos de detecção e classificação teriam dificuldade em reconhecer limites e contextos dos objetos.

Um processo de segmentação bem implementado garante:

- Maior precisão em diagnósticos e análises automáticas;
- Redução de ruído e redundância nos dados;
- Melhor desempenho em tarefas de reconhecimento e classificação;
- Maior confiabilidade na tomada de decisão automatizada.

Assim, a segmentação é a ponte entre o pré-processamento (limpeza e padronização) e a detecção/classificação (interpretação do conteúdo), consolidando-se como uma das etapas mais críticas da pipeline de visão computacional.

3. Detecção e Classificação de Imagens

A detecção e classificação de imagens são etapas fundamentais na Visão Computacional, responsáveis por identificar, nomear e localizar objetos dentro de uma imagem digital. Enquanto a classificação atribui um único rótulo global à imagem (por exemplo, “gato” ou “carro”), a detecção de objetos vai além — ela reconhece múltiplos elementos simultaneamente e define suas posições exatas por meio de caixas delimitadoras (bounding boxes).

Essas tarefas são essenciais em uma ampla gama de aplicações modernas:

- Veículos autônomos: identificação de pedestres, sinais e faixas de rodagem.
- Segurança e monitoramento: reconhecimento facial e detecção de comportamentos anômalos.
- Medicina: classificação de imagens radiológicas e localização de lesões.
- Agricultura de precisão: contagem e reconhecimento de plantas ou frutos.
- Varejo e indústria: detecção automática de produtos, controle de estoque e inspeção visual.

Com o avanço do Deep Learning, a precisão e a velocidade dos modelos de detecção e classificação atingiram níveis próximos — e em alguns casos superiores — à percepção humana.

3.1 Diferença entre Classificação e Detecção

Tarefa	Descrição	Exemplo
Classificação de Imagens	Atribui um único rótulo à imagem inteira.	“Esta imagem é um cachorro.”
Detecção de Objetos	Localiza e identifica múltiplos objetos em uma imagem. Cada objeto recebe um rótulo e uma caixa delimitadora.	“Cachorro”, “Pessoa”, “Carro” com coordenadas específicas.

A detecção é mais complexa porque combina duas tarefas simultâneas:

1. Classificação (qual é o objeto);
2. Regressão espacial (onde ele está na imagem).

3.2 Modelos Modernos de Classificação e Detecção

Os modelos de detecção e classificação atuais utilizam Redes Neurais Convolucionais (CNNs) e Transformers Visuais (ViTs), treinados em grandes bases de dados como ImageNet, COCO e Open Images.

Principais Modelos de Classificação

- VGG16 / VGG19: arquiteturas profundas com camadas convolucionais sequenciais.
- ResNet (Residual Networks): introduziu conexões residuais que evitam a perda de gradiente.
- MobileNet e EfficientNet: otimizados para dispositivos móveis e aplicações embarcadas, com alta eficiência energética.
- Inception e DenseNet: arquiteturas híbridas que combinam filtros de diferentes tamanhos para extrair mais detalhes da imagem.

Principais Modelos de Detecção

- YOLO (You Only Look Once): modelo de detecção em tempo real, de estágio único (single-shot), ideal para aplicações que exigem velocidade.
- SSD (Single Shot MultiBox Detector): balanceia rapidez e precisão, amplamente utilizado em dispositivos embarcados.
- Faster R-CNN: modelo de dois estágios com alta acurácia, muito empregado em tarefas que exigem precisão detalhada.
- DETR (Detection Transformer): utiliza transformers para simplificar o pipeline de detecção, eliminando a necessidade de ancoragens manuais.

3.2 Técnicas Complementares e Transfer Learning

- **Transfer Learning:** aproveita pesos de modelos pré-treinados (como ResNet, VGG ou MobileNet no ImageNet) e os adapta para novas tarefas com menor quantidade de dados.
- **Fine-Tuning:** ajusta parcialmente as camadas de um modelo já treinado para melhorar o desempenho em um domínio específico.
- **Classificação Hierárquica:** organiza os rótulos de forma hierárquica (exemplo: animal → mamífero → cão), permitindo maior precisão em classificações complexas.
- **Data Augmentation:** aumenta o conjunto de dados com variações artificiais (rotação, espelhamento, ruído) para reduzir overfitting.

3.4 Frameworks e Ferramentas Utilizadas

- **TensorFlow / Keras:** oferece bibliotecas de alto nível para treinamento e inferência de modelos CNN e MobileNet.
- **PyTorch / TorchVision:** fornece arquiteturas pré-treinadas como ResNet, Faster R-CNN e DETR.
- **Ultralytics YOLO:** implementação moderna do YOLOv5 e YOLOv8, amplamente usada em aplicações de tempo real.
- **OpenCV:** suporte à integração de modelos de detecção, além de ferramentas de pré e pós-processamento.
- **ONNX e TensorRT:** utilizados para otimizar modelos de IA em ambientes de produção e inferência acelerada por GPU.

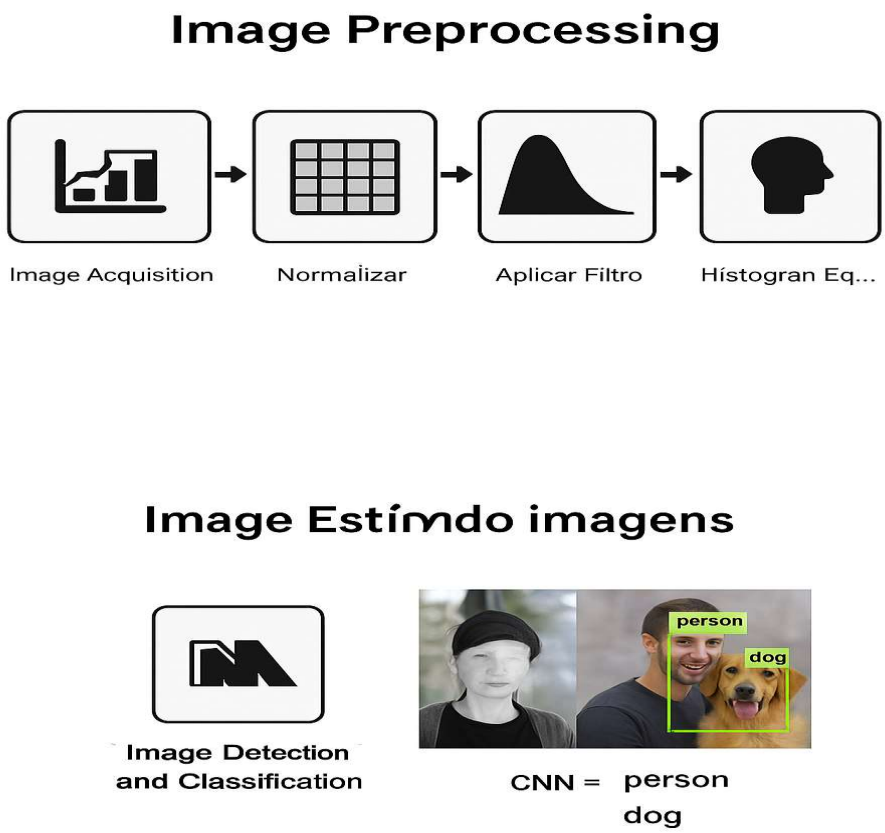
3.5 Classificação de Imagem (MobileNetV2 – TensorFlow/Keras)

```python
from tensorflow.keras.applications.mobilenet_v2 import (
    MobileNetV2, preprocess_input, decode_predictions
)
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Carregar o modelo pré-treinado no ImageNet
model = MobileNetV2(weights='imagenet')

# Carregar e preparar a imagem (224x224 pixels)
img = load_img("imagem_para_classificar.jpg", target_size=(224, 224))
img_array = img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_pre = preprocess_input(img_array)

# Realizar a predição
pred = model.predict(img_pre)
for classe_id, nome, prob in decode_predictions(pred, top=3)[0]:
    print(f"{nome}: {prob * 100:.2f}%")
```

3.6 Exemplo de Detecção com YOLOv8 (PyTorch / Ultralytics)

```python
# Exemplo de detecção com YOLOv8
from ultralytics import YOLO

# Carregar o modelo YOLOv8 pré-treinado
model = YOLO("yolov8n.pt")

# Executar a detecção de objetos em uma imagem
results = model("imagem_para_detectar.jpg", show=True)

# Exibir rótulos e coordenadas
for result in results:
    boxes = result.boxes
    for box in boxes:
        print(f"Classe: {box.cls}, Confiança: {box.conf}, Coordenadas: {box.xyxy}")
```

Figura 3 – Exemplo de detecção e classificação de objetos com CNN.



3.7 Evolução Tecnológica: CNNs e Vision Transformers (ViTs)

A detecção e classificação evoluíram consideravelmente com o uso das Redes Neurais Convolucionais (CNNs) e, mais recentemente, dos Transformers Visuais (ViT).
Os ViTs tratam as imagens como sequências de “pedaços” (patches), permitindo capturar relações de longo alcance entre regiões diferentes. Essa abordagem trouxe ganhos significativos de precisão e generalização, principalmente em modelos como Swin Transformer e DETR.

Apesar dos avanços, ainda existem desafios importantes:

- A necessidade de grandes volumes de dados anotados para treinar os modelos;
- O alto consumo de energia computacional;

- E a importância da explicabilidade e confiabilidade dos resultados, especialmente em áreas sensíveis como a medicina e segurança pública.

3.8 Fluxo Visual e Ilustração

A Figura 3, já presente em seu documento, exemplifica o resultado típico de uma detecção e classificação, com caixas delimitadoras (bounding boxes) e rótulos descritivos associados a cada objeto identificado.

3.9 Tendências e Futuro da Visão Computacional

O futuro da visão computacional está cada vez mais integrado à Internet das Coisas (IoT) e à Computação de Borda (Edge AI), permitindo o processamento de imagens em tempo real diretamente em dispositivos locais, sem depender de servidores na nuvem.

Além disso, a convergência entre Visão Computacional e Inteligência Artificial Generativa está abrindo novas possibilidades, como:

- Reconstrução e aprimoramento de imagens degradadas;
- Geração de dados sintéticos para treinamento de modelos;
- Criação de ambientes tridimensionais e simulações realistas;
- Explicabilidade visual, onde os sistemas justificam suas previsões com regiões destacadas da imagem.

Essa combinação de visão e IA generativa promete revolucionar setores como medicina diagnóstica, segurança inteligente, entretenimento digital e manufatura 4.0.

3.10 Conclusão da Seção

A detecção e classificação de imagens são o cérebro da visão computacional aplicada, permitindo que máquinas compreendam e interajam com o ambiente visual de forma autônoma. A integração entre CNNs, Transformers e IA generativa está levando a área a um novo patamar, no qual imagens não apenas são reconhecidas, mas também interpretadas e contextualizadas.

Essas técnicas, quando combinadas com um bom pré-processamento e uma segmentação eficiente, formam o alicerce dos sistemas modernos de percepção artificial — transformando pixels em informação, informação em conhecimento e conhecimento em decisão.

4. Conclusão

A visão computacional representa um dos campos mais promissores e transformadores da Inteligência Artificial, permitindo que sistemas e máquinas interpretem o mundo visual de forma cada vez mais precisa e autônoma. Por meio das etapas de pré-processamento, segmentação e detecção/classificação, é possível transformar imagens brutas em informações estruturadas, facilitando a tomada de decisões em diferentes contextos.

O pré-processamento atua como a base para qualquer aplicação visual, garantindo a padronização, remoção de ruídos e correção de contrastes. Essa etapa é essencial para que os modelos de aprendizado de máquina operem de maneira eficiente e obtenham resultados confiáveis. Já a segmentação possibilita a separação das regiões relevantes de uma imagem, permitindo que sistemas reconheçam objetos ou estruturas com precisão. Técnicas como limiarização, watershed e modelos baseados em redes neurais, como U-Net e Mask R-CNN, demonstram alta eficácia em áreas críticas como medicina e inspeção industrial.

Por fim, a detecção e classificação de imagens elevam o nível de análise ao identificar, nomear e localizar múltiplos objetos simultaneamente, utilizando arquiteturas avançadas como YOLO, SSD e Faster R-CNN. Esses modelos, aliados ao uso de transfer learning e deep learning**,** possibilitam aplicações em tempo real, desde veículos autônomos até sistemas de monitoramento e controle de qualidade.

O futuro da visão computacional está intimamente ligado à computação de borda (Edge AI**)** e à Internet das Coisas (IoT), permitindo o processamento de imagens diretamente em dispositivos locais, com menor latência e maior privacidade. Além disso, a integração com IA generativa abre novas perspectivas para reconstrução de imagens, geração de dados sintéticos e criação de modelos tridimensionais.

Portanto, compreender e aplicar corretamente essas etapas — do pré-processamento à classificação — é essencial para o desenvolvimento de soluções inteligentes, escaláveis e precisas. A visão computacional não apenas automatiza processos, mas também amplia as fronteiras do conhecimento, conectando a percepção humana à capacidade computacional de forma sinérgica e inovadora.

5. Referências

- GONZALEZ, Rafael C.; WOODS, Richard E. Processamento Digital de Imagens. 4. ed. São Paulo: Pearson, 2018.
- GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep Learning**.** Cambridge: MIT Press, 2016.
- SZELISKI, Richard. Computer Vision: Algorithms and Applications. 2nd ed. Springer, 2022.
- RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas**.** U-Net: Convolutional Networks for Biomedical Image Segmentation**.** *arXiv preprint arXiv:1505.04597*, 2015.
- REN, Shaoqing et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 39, n. 6, p. 1137–1149, 2017.
- REDMON, Joseph et al. You Only Look Once: Unified, Real-Time Object Detection (YOLO)**.** *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- HE, Kaiming et al. Mask R-CNN. *IEEE International Conference on Computer Vision (ICCV)*, 2017.
- O'SHEA, Keiron; NASH, Ryan. An Introduction to Convolutional Neural Networks**.** *arXiv preprint arXiv:1511.08458*, 2015.
- ZHOU, Zongwei et al. DeepLabV3+: Encoder-Decoder with Atrous Convolution for Semantic Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.
- META AI. Segment Anything Model (SAM)**.** Meta Research, 2023. Disponível em: https://segment-anything.com/. Acesso em: 24 out. 2025.