Aluno(a):	Alan Gabriel de Souza Alvarado	**Matrícula:**	2110870
Aluno(a):	Marcelo Feliciano Figueiredo	**Matrícula:**	2110871
Professora:	Lyndainês Santos	**Documento:**	Avaliação AV3 - 1ª chamada
Disciplina:	P.D.I	**Turma / Semestre:**	09-19 / 2025.1

Avaliação Prática - AV3

Descrição do Projeto

O presente projeto tem como objetivo a aplicação de técnicas de Processamento Digital de Imagens (PDI) e Visão Computacional para realizar a detecção, classificação e análise de moedas brasileiras a partir de imagens digitais. O sistema desenvolvido é capaz não apenas de reconhecer o valor das moedas presentes em uma imagem, mas também de avaliar seu estado de conservação, identificando possíveis danos como manchas.

Para atingir esses objetivos, o projeto combina técnicas clássicas de processamento de imagens, como a detecção de círculos utilizando o método de Transformada de Hough, com métodos modernos baseados em Inteligência Artificial, através do uso de um modelo de detecção de objetos da família YOLO (You Only Look Once), previamente treinado para reconhecer diferentes classes de moedas brasileiras. Além disso, é aplicada uma análise de textura utilizando o canal de brilho (canal V no espaço HSV) para avaliar a qualidade física das moedas.

A demonstração da implementação do projeto se dará por partes, a primeira será sobre o treinamento do modelo personalizado do YOLO, a segunda será a implementação da Transformada de Hough para detecção de moedas, a terceira será a implementação do modelo do YOLO no código para classificação das moedas, e a quarta parte será responsável pela criação de manchas simuladas nas moedas. A quinta etapa será para identificação de danos presentes nas moedas.

Metodologia – Parte 1 – Treinamento do Modelo YOLO

Para o desenvolvimento do presente projeto, o primeiro passo fundamental foi o treinamento de um modelo de detecção de objetos utilizando a arquitetura YOLO, amplamente reconhecida pela sua eficiência em tarefas de detecção em tempo real. Este treinamento foi essencial para capacitar o sistema a identificar e classificar corretamente as diferentes moedas brasileiras presentes nas imagens.

Etapas do Processo

Inicialmente, foram coletadas 54 fotografias reais das moedas brasileiras, abrangendo todas as classes: 5 centavos, 10 centavos, 25 centavos, 50 centavos e 1 real. As imagens foram capturadas em diferentes ângulos, iluminações e posições, garantindo assim uma maior robustez e capacidade de generalização do modelo.

O processo de anotação das imagens, fundamental para o treinamento supervisionado, foi realizado por meio da plataforma Make Sense, ferramenta que oferece suporte tanto para anotação quanto para organização de datasets destinados a modelos de visão computacional. Cada moeda presente nas imagens foi devidamente rotulada com sua respectiva classe.

Organização das Pastas

Após o download dos arquivos .txt gerados pelo Make Sense no formato YOLO, foi organizada a seguinte estrutura de pastas localmente no computador:

A organização das pastas do projeto ficou da seguinte forma:

- Dentro da pasta dataset, foram criadas duas subpastas principais: images e labels.
- A pasta images contém duas pastas internas:
  - train: que armazena as imagens utilizadas para o treinamento do modelo.
  - val: que armazena as imagens utilizadas para a validação do modelo.
- A pasta labels segue a mesma estrutura da pasta de imagens, contendo:

  - train: onde estão os arquivos .txt com os rótulos correspondentes às imagens de treinamento.
  - val: que contém os arquivos de rótulos das imagens de validação.
- Além disso, na raiz da pasta dataset, encontra-se o arquivo data.yaml, que é o arquivo de configuração utilizado pelo YOLO. Ele define os caminhos para as pastas de imagens, os nomes das classes e outras informações necessárias para o treinamento do modelo.

Procedimento no CMD - Treinamento

O treinamento foi executado utilizando o terminal do Windows (CMD) com o pacote Ultralytics YOLOv8 instalado via Python. Os principais comandos utilizados foram:

1. Acesso ao diretório onde os arquivos estão localizados:

   cd C:\Users\AlanG\Documentos\dataset

2. Execução do comando de treinamento:

   yolo detect train data=data.yaml model=yolov8n.pt epochs=100 imgsz=640

- data=data.yaml — Arquivo de configuração do dataset.
- model=yolov8n.pt — Modelo base utilizado (YOLOv8 Nano).
- epochs=100 — Número de épocas definidas para o treinamento.
- imgsz=640 — Tamanho das imagens utilizado no treinamento.

Durante o treinamento, o YOLO criou automaticamente uma pasta chamada runs/detect/train4, onde train4 representa o número da execução do treinamento

Dentro desta pasta foram gerados:

- Pesos do modelo (best.pt e last.pt).
- Imagens com as predições do modelo sobre os dados de validação.
- Arquivos de logs contendo métricas de desempenho.

Desempenho e Acurácia

Ao término das 100 épocas de treinamento, o modelo alcançou os seguintes resultados na validação:

- Precisão (Precision): 0.95
- Revocação (Recall): 0.94
- mAP50 (Mean Average Precision @ IoU 50%): 0.995
- mAP50-95: 0.913

Esses resultados indicam um alto desempenho na detecção e classificação correta das moedas, validando a efetividade do modelo YOLOv8 no contexto proposto.

Exemplos de Imagens Geradas Durante o Treinamento

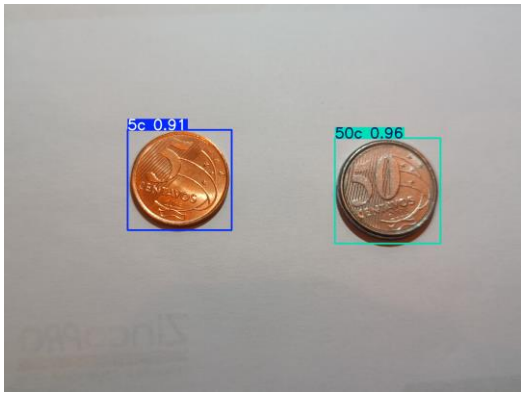

Figura 1 – Foto de duas moedas, a da esquerda de 5 centavos e a da direita de 50 centavos, detectadas corretamente.

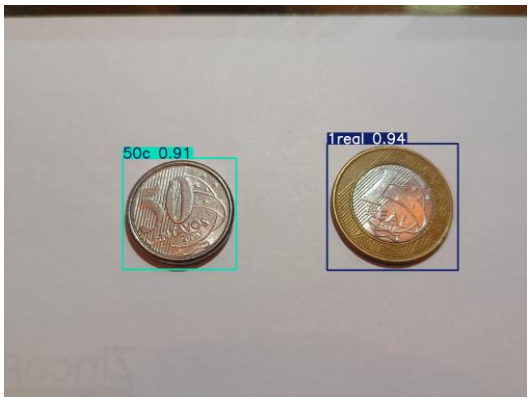

Figura 2 – Foto de duas moedas, a da esquerda de 50 centavos e a da direita de 1 real, detectadas corretamente.

Metodologia – Parte 2 – Transformada de Hough

O código apresentado tem como objetivo realizar a detecção de moedas em uma imagem utilizando técnicas de Processamento Digital de Imagens (PDI) e Visão Computacional. As bibliotecas utilizadas para o desenvolvimento foram:

- OpenCV (cv2) – para operações de processamento de imagem, detecção de círculos e manipulações visuais;
- NumPy (np) – para operações matriciais e matemáticas;
- Matplotlib (plt) – para exibir as imagens processadas;
- Ultralytics YOLO – para a detecção e classificação das moedas;
- Random – utilizada na etapa de simulação de danos.

A Transformada de Hough é uma técnica clássica utilizada na detecção de formas geométricas em imagens, como linhas, círculos e elipses. No caso da detecção de círculos, o método busca por pontos na imagem que satisfaçam a equação paramétrica de um círculo. A função cv2.HoughCircles() implementa o algoritmo baseado no gradiente, permitindo detectar círculos mesmo em imagens com ruídos ou bordas imperfeitas.

Descrição dos Passos do Código

1) Carregamento da imagem:

A imagem contendo as moedas é carregada utilizando a função cv2.imread(). É realizada uma verificação para garantir que a imagem foi carregada corretamente, evitando erros no processamento posterior.

Figura 3 – Imagem carregada para demostração do código

2) Conversão para escala de cinza:

A imagem colorida é convertida para escala de cinza com cv2.cvtColor(), pois a Transformada de Hough opera sobre imagens monocromáticas, focando apenas na variação de intensidade de pixels.

3) Suavização da imagem:

Aplica-se um filtro Gaussiano através da função cv2.GaussianBlur() com um kernel de tamanho (5,5). Este passo tem como objetivo reduzir ruídos da imagem, preservando as bordas, o que facilita a detecção de círculos.

4) Detecção dos círculos usando Hough Gradient:

A função cv2.HoughCircles() é utilizada para detectar os círculos (moedas). Os principais parâmetros definidos foram:

- dp=1.0: resolução do acumulador igual à imagem original.
- minDist=900: define a distância mínima entre os centros dos círculos detectados.
- param1=100: limite superior do detector de bordas interno (Canny).
- param2=50: limite do acumulador de detecção. Valores maiores tornam a detecção mais rigorosa.
- minRadius=400 e maxRadius=500: definem os limites dos raios dos círculos esperados (moedas).
-

5) Desenho dos círculos detectados:

Caso a função Hough encontre círculos, eles são convertidos para valores inteiros e desenhados sobre a imagem original. A função cv2.circle() é utilizada para desenhar o contorno de cada círculo na cor verde, com uma espessura de 5 pixels.

6) Exibição dos resultados:

A imagem resultante, agora com os círculos detectados, é convertida de BGR para RGB (para compatibilidade com Matplotlib) usando cv2.cvtColor(). Em seguida, é exibida na tela por meio da função plt.imshow() com um título indicando a quantidade de moedas detectadas.

Figura 4 – Imagem após a Transfomada de Hough, detectando duas moedas.

7) Impressão no terminal:

O código finaliza essa etapa informando no terminal a quantidade de moedas detectadas com a mensagem:

```
=== DETECÇÃO DE MOEDAS ===
Total de moedas detectadas: 2
```

Figura 5 – Terminal mostrando a quantidade de moedas na imagem.

Metodologia – Parte 3 – Implementação do modelo treinado YOLO

Nesta etapa, o código realiza a detecção e a classificação das moedas utilizando o modelo YOLO, um dos algoritmos de detecção de objetos mais eficientes atualmente. O YOLO permite, além de localizar as moedas na imagem, identificar o valor de cada uma delas (5 centavos, 10 centavos, 25 centavos, 50 centavos e 1 real).

Descrição dos Passos do Código

7) Inicializa estruturas para contagem e valores:

Define-se um dicionário chamado contagem para armazenar quantas unidades de cada moeda foram detectadas. Também é criado outro dicionário valores_moedas que associa cada rótulo de moeda ao seu respectivo valor em reais. Além disso, uma variável valor_total é inicializada para acumular o valor total presente na imagem.

8) Realiza a detecção das moedas com YOLO:

O modelo previamente treinado é executado sobre a imagem (saida) usando a função modelo.predict().

- O parâmetro source=saida indica qual imagem será analisada.
- save=False desativa o salvamento automático dos resultados.
- verbose=False oculta saídas detalhadas no terminal.

O resultado da detecção é armazenado em resultados, e como é uma lista, pegamos a primeira posição com resultado = resultados[0].

9) Processa as detecções e anota na imagem:

O loop for box in resultado.boxes: percorre cada detecção realizada pelo YOLO. Dentro desse loop:

- A classe (cls_id) é extraída e usada para obter o nome da moeda (ex.: '25c', '1real') através de resultado.names[cls_id].
- A confiança da detecção (conf) é capturada para informar a precisão daquele reconhecimento.
- As coordenadas do retângulo da caixa delimitadora são extraídas de box.xyxy.

Então:

- Desenha-se um retângulo verde na imagem sobre a moeda com cv2.rectangle().
- Adiciona-se uma legenda acima da moeda com o nome e a confiança usando cv2.putText().

- Atualiza-se o dicionário contagem somando uma unidade à moeda detectada.
- Soma-se o valor dessa moeda ao valor_total com base no dicionário valores_moedas.

Também é impresso no terminal cada moeda detectada com sua respectiva confiança.

10) Exibe a imagem final com as detecções YOLO:

A imagem é novamente convertida para RGB com cv2.cvtColor() e exibida através do matplotlib.

O título da imagem é "Resultado Final com Detecção YOLO", e os quadrados e labels mostram cada moeda identificada.

Figura 6 - Detecção de 2 moedas, a da esquerda de 25 centavos e da direita de 1 real, com 0,94 e 0,95 de confiança respectivamente.

11) Imprime o valor final:
Por fim, o terminal apresenta um relatório final da contagem de moedas:

- Lista quantas unidades de cada moeda foram encontradas.
- Exibe o valor total somado de todas as moedas presentes na imagem, já convertido em reais.

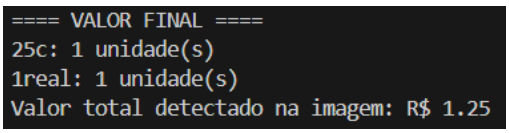

Figura 7 – Terminal mostrando o tipo e quantidade de moedas presentes na imagem, como também a soma dos valores, que para esta imagem é R$ 1,25

Metodologia – Parte 4 – Implementação das manchas simuladas

Nesta etapa, o objetivo é simular defeitos ou danos nas moedas presentes na imagem, com o intuito de avaliar posteriormente o estado de conservação de cada moeda (se está em bom estado ou danificada). Essa simulação é feita aplicando manchas pretas de tamanhos variados em posições aleatórias dentro das moedas.

Descrição dos Passos do Código

12) Cria uma máscara circular para isolar a moeda:
Para cada moeda detectada pelo algoritmo de Hough, é criada uma máscara circular. Isso é feito com np.zeros() para gerar uma matriz preta do tamanho da imagem e, em seguida, cv2.circle() desenha um círculo branco na posição da moeda, isolando sua área.

13) Associa a moeda detectada pelo Hough com a detectada pelo YOLO:
Para sabermos qual é o nome da moeda (ex.: 5c, 1real), é feita uma busca da caixa detectada pelo YOLO que está mais próxima do centro da moeda detectada pelo Hough. O

cálculo da distância é feito usando a fórmula da distância euclidiana. A menor distância encontrada define qual é o nome da moeda.

14) Define a quantidade aleatória de manchas:

É definido um número aleatório de manchas que serão aplicadas na moeda, variando de 0 a 2 manchas, utilizando random.randint(0, 2). Isso permite simular moedas mais danificadas, pouco danificadas ou sem nenhum dano.

15) Define o tamanho da mancha:

Cada mancha tem seu raio definido como 25% do raio da moeda. Isso garante que a mancha seja proporcional ao tamanho da moeda.

16) Calcula a posição aleatória da mancha dentro da moeda:

Para evitar que a mancha fique fora da moeda, é utilizado um laço while que sorteia posições (offsets em x e y) e testa se essas posições estão dentro do círculo da moeda. A condição distancia_ao_centro <= (r - raio_manchas) ** 2 assegura que o ponto está dentro da moeda, considerando o raio da própria mancha.

17) Aplica a mancha na imagem:

Com a posição da mancha definida, desenha-se um círculo preto ((0, 0, 0)) usando cv2.circle() sobre a imagem saída. Isso simula visualmente um defeito, desgaste, corrosão ou sujeira na moeda.

18) Registra no terminal a simulação feita:

Para cada moeda, o código imprime no terminal a quantidade de manchas aplicadas e o nome da moeda, permitindo acompanhar o processo da simulação.

```
=== SIMULAÇÃO DE DANO ===
Adicionadas 0 manchas simuladas na moeda de 25c
Adicionadas 2 manchas simuladas na moeda de 1real
```

Figura 8 – Terminal mostrando a quantidade de manchas, neste caso 0 manchas para a moeda de 25 centavos e 2 manchas para a moeda de 1 real.

19) Exibe a imagem com as manchas aplicadas:

Após aplicar todas as manchas, a imagem é convertida de BGR para RGB e exibida com matplotlib. No título, aparece "Imagem com Mancha Simulada nas Moedas", permitindo visualizar claramente quais moedas receberam manchas simulando defeitos.

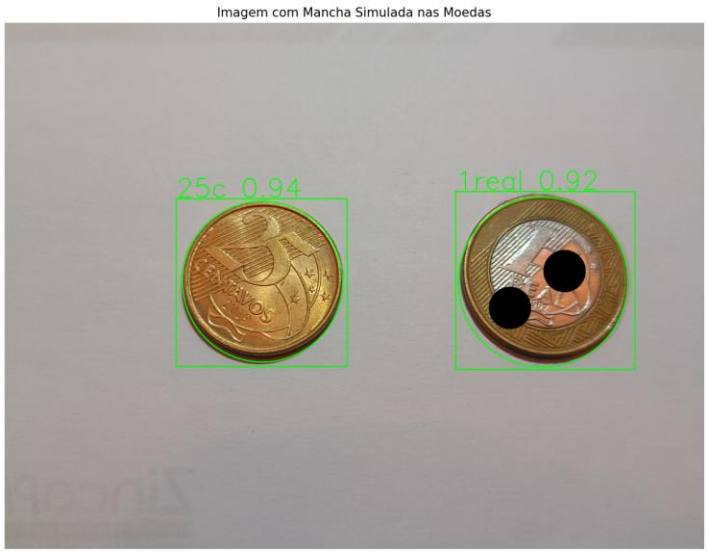

Figura 9 – Imagem das moedas com manchas.

Metodologia – Parte 5 – Verificação do estado das moedas

Nesta etapa, o objetivo é analisar o estado físico de cada moeda presente na imagem, verificando se ela está "BOA" ou "DANIFICADA". Essa análise é feita com base na variação da textura da moeda, utilizando o desvio padrão do canal de brilho (canal V na escala HSV).

Descrição dos Passos do Código

19) Conversão para espaço HSV:

A imagem é convertida de BGR para HSV utilizando cv2.cvtColor(). O foco é trabalhar no canal V (Value), que representa o brilho da imagem. Isso é eficiente para análise de textura e identificação de variações na superfície da moeda.

20) Associa a moeda detectada pelo Hough com a detectada pelo YOLO:

De forma semelhante à etapa de simulação de dano, é feito um mapeamento entre as moedas detectadas por Hough e as detectadas por YOLO. Busca-se o nome da moeda através da caixa detectada pelo YOLO mais próxima do centro da moeda (x, y), usando distância euclidiana.

21) Cria uma máscara circular para isolar a moeda:

Gera uma máscara preta onde somente a região da moeda fica branca, isolando a área da moeda na imagem.

22) Aplica a máscara no canal de brilho (V):

A operação cv2.bitwise_and() mantém apenas os pixels da moeda, eliminando o fundo e outras partes da imagem.

23) Suaviza a imagem da moeda:

Aplica-se um filtro Gaussiano (cv2.GaussianBlur()) sobre a moeda isolada para reduzir ruídos e pequenos reflexos. Isso garante que o cálculo da textura seja mais robusto e preciso.

24) Extrai os pixels da moeda:

Seleciona os pixels que estão dentro da máscara (onde a máscara é igual a 255) para análise. Esses pixels representam exclusivamente a região da moeda.

25) Calcula o desvio padrão da moeda:

O desvio padrão dos valores de brilho é calculado com np.std(). Um valor baixo indica uma superfície uniforme e bem preservada (boa). Um valor alto sugere variações, manchas ou desgaste (danificada).

26) Classifica a moeda:

Se o desvio padrão for maior que 50, a moeda é classificada como "DANIFICADA" e recebe uma marcação em vermelho. Se for menor ou igual a 50, é classificada como "BOA" e recebe uma marcação em verde.

```
=== VERIFICAÇÃO DO ESTADO DAS MOEDAS ===
25c - Desvio Padrão: 37.70
Estado da moeda de 25c: BOA
1real - Desvio Padrão: 62.48
Estado da moeda de 1real: DANIFICADA
```

Figura 10 – Terminal mostrando a classificação das moedas.

27) Anota na imagem o estado da moeda:

Utiliza-se cv2.putText() para desenhar sobre a imagem o texto informando o estado da moeda (BOA ou DANIFICADA) próximo da sua posição.

28) Exibe a imagem com a classificação:

Após aplicar a classificação, a imagem é convertida de BGR para RGB e exibida com matplotlib. No título, aparece " Estado Final das Moedas (Boa ou Danificada) ", permitindo visualizar claramente qual classificação cada moeda recebeu

Figura 11 – Imagem final, com a classificação de cada moeda na imagem.

Conclusão

O projeto alcançou com sucesso seus objetivos, realizando a detecção, classificação e contagem de moedas brasileiras por meio da integração de técnicas de Processamento Digital de Imagens (PDI) e Inteligência Artificial com YOLOv8. Além da identificação, foi possível analisar o estado físico das moedas, simulando danos e avaliando sua conservação com base na variação de textura.

A proposta demonstrou ser eficiente, com resultados precisos tanto na detecção dos objetos quanto na análise de possíveis desgastes. Conclui-se que a combinação de PDI com redes neurais é uma abordagem eficaz e aplicável a diversos cenários práticos.

Devido ao tamanho dos arquivos e às limitações de armazenamento do GitHub, este repositório contém apenas o código do projeto e um link auxiliar do Google Drive. Todos os arquivos completos, incluindo imagens, datasets e modelos treinados, estão disponíveis no Google Drive.

Repositório Github:
https://github.com/AlanGabriel312/T896_N35AB_ALAN_ALVARADO/tree/main/Av3_PDI

Google Drive:
https://drive.google.com/drive/folders/1QJFTzFYrQt_ZehjAeX1TekBZ6Iub84hr?usp=sharing