PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Instituto Politécnico PUC Minas

Clarissa Lima Loures

Prova 2

Belo Horizonte

2024

Introdução

Neste trabalho, realizamos a construção de um sistema de reconhecimento de cartas de baralho utilizando a rede neural convolucional YOLO (You Only Look Once). A abordagem do YOLO é conhecida por sua capacidade de detectar objetos em tempo real com alta precisão, tornando-a ideal para a tarefa de identificar cartas específicas, como ases e reis, em imagens capturadas de uma webcam.

O projeto foi desenvolvido em várias etapas, iniciando com a criação de um conjunto de dados rotulado contendo imagens de cartas de baralho, onde foram destacados e anotados os ases e reis. A seguir, utilizamos esses dados para treinar a rede YOLO, ajustando seus parâmetros para otimizar a detecção das cartas de interesse.

Após o treinamento, desenvolvemos um programa que captura imagens em tempo real usando uma webcam, aplica o modelo YOLO treinado para identificar ases e reis nas imagens, e compara estas cartas com as de outra imagem de referência para determinar o vencedor de uma partida de cartas.

O objetivo deste relatório é detalhar cada uma dessas etapas, apresentando os métodos utilizados, os desafios enfrentados, os resultados obtidos e as conclusões derivadas do desenvolvimento e implementação do sistema de reconhecimento de cartas. A demonstração do funcionamento da rede treinada será apresentada ao professor em sala de aula, conforme solicitado.

Uso do LabelImg para Geração de Rotulação das Imagens para o YOLO

Para treinar a rede YOLO com precisão, é essencial ter um conjunto de dados bem rotulado, onde cada objeto de interesse nas imagens é devidamente marcado. Para esta tarefa, utilizamos a ferramenta LabelImg, que é uma aplicação gráfica popular para rotulação de imagens em projetos de visão computacional.

Instalação e Configuração do LabelImg

O primeiro passo foi instalar o LabelImg. Esta ferramenta pode ser instalada via pip com o comando:

```
pip install labelImg
```

Após a instalação, a ferramenta pode ser iniciada com o comando:

```
labelImg
```

Processo de Rotulação

O processo de rotulação com o LabelImg envolveu várias etapas:

1. **Carregar as Imagens**:
    - Abrimos o LabelImg e carregamos as imagens de cartas de baralho de nosso conjunto de dados.
2. **Configuração do Formato YOLO**:
    - Antes de iniciar a rotulação, configuramos o LabelImg para salvar os rótulos no formato YOLO. Isso é feito selecionando a opção "YOLO" no menu "Save" format.
3. **Rotulação das Cartas**:
    - Para cada imagem, desenhamos caixas delimitadoras (bounding boxes) ao redor das cartas de interesse (ases e reis).
    - Ao desenhar cada caixa delimitadora, selecionamos o nome da classe correspondente (por exemplo, "a_paus" para Ás de Paus).
4. **Salvar os Rótulos**:
    - Após rotular todas as cartas em uma imagem, salvamos os rótulos, que são armazenados em arquivos de texto com o mesmo nome das imagens, mas com a extensão `.txt`. Cada linha nos arquivos de rótulo contém:
        - O índice da classe
        - As coordenadas normalizadas da caixa delimitadora (centro x, centro y, largura, altura)

Exemplo de um arquivo de rótulo para uma imagem contendo um Ás de Paus:

```
0 0.5 0.5 0.1 0.2
```

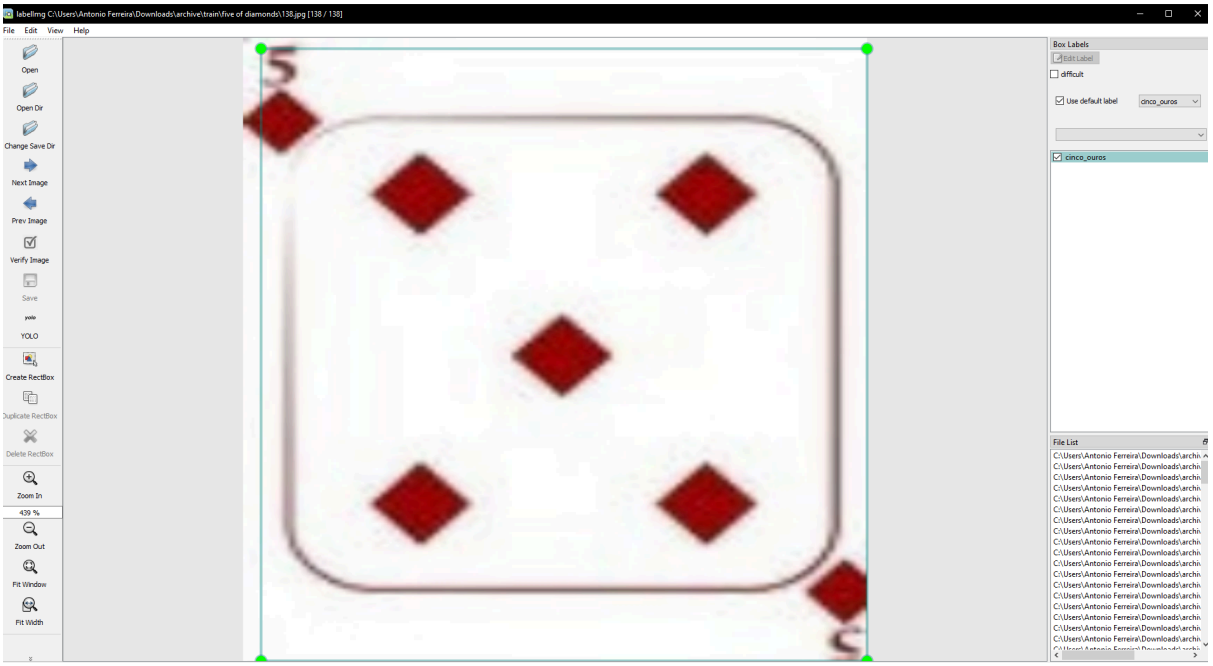
Neste exemplo, `0` representa o índice da classe "a_paus", e os números seguintes representam as coordenadas normalizadas da caixa delimitadora.

Benefícios do LabelImg

O uso do LabelImg proporcionou vários benefícios significativos para o nosso projeto:

- **Interface Intuitiva**: A interface gráfica do LabelImg é fácil de usar, o que facilitou o processo de rotulação, mesmo para usuários com pouca experiência.
- **Compatibilidade com YOLO**: A ferramenta oferece suporte direto ao formato YOLO, eliminando a necessidade de conversão manual de rótulos.
- **Eficiência**: A capacidade de rapidamente desenhar caixas delimitadoras e atribuir classes permitiu a criação de um conjunto de dados rotulado de alta qualidade em um tempo reduzido.

O uso do LabelImg foi fundamental para garantir a precisão e eficiência na criação de nossos dados de treinamento. A ferramenta simplificou a tarefa de rotulação, permitindo que focássemos mais no desenvolvimento e treinamento da rede YOLO, resultando em um modelo mais robusto e eficaz para a identificação de cartas de baralho.



Treinamento

O processo de treinamento da rede YOLO foi realizado utilizando um conjunto de dados composto por imagens de diferentes tipos de cartas de baralho. A seguir, apresentamos o script utilizado para preparar os dados e realizar o treinamento do modelo.

Preparação dos Dados

O conjunto de dados foi dividido em diretórios contendo imagens e rótulos correspondentes para cada tipo de carta. As cartas incluídas no treinamento foram:

- Ás de paus
- Ás de ouros
- Ás de copas
- Oito de copas
- Cinco de ouros

Para preparar o conjunto de dados, foi desenvolvida uma função para criar diretórios de validação (`val`) e mover uma parte das imagens de treino para esses diretórios. Essa função foi aplicada a cada diretório de cartas, conforme o código abaixo:

```
from pathlib import Path
from shutil import copyfile
from ultralytics import YOLO
import os
```

```
import random

# Diretórios das imagens e dos rótulos
train_dir_apaus = "C:\\Users\\Clarissa\\Desktop\\prova2\\train\\ace of
clubs"
train_dir_aouros = "C:\\Users\\Clarissa\\Desktop\\prova2\\train\\ace of
diamonds"
train_dir_acopas = "C:\\Users\\Clarissa\\Desktop\\prova2\\train\\ace of
hearts"
train_dir_oito_copas =
"C:\\Users\\Clarissa\\Desktop\\prova2\\train\\eight of hearts"
train_dir_cinco_ouros =
"C:\\Users\\Clarissa\\Desktop\\prova2\\train\\five of diamonds"
val_dir = "C:\\Users\\Clarissa\\Desktop\\prova2\\var_dir\\"

# Função para criar diretórios 'val' e mover imagens e rótulos
def create_val_and_move_images(src_dir, dest_dir, val_ratio=0.2):
    os.makedirs(dest_dir, exist_ok=True)
    files = os.listdir(src_dir)
    random.shuffle(files)
    num_files_val = int(len(files) * val_ratio)
    val_files = files[:num_files_val]
    for file_name in val_files:
        # Verificar se existe um arquivo de rótulo correspondente
        label_file = file_name.replace(".jpg", ".txt")
        if label_file not in files:
            continue  # Ignorar a imagem se não houver arquivo de rótulo
        # Mover imagens
        img_src = os.path.join(src_dir, file_name)
        img_dest = os.path.join(dest_dir, file_name)
        copyfile(img_src, img_dest)
        # Mover rótulos
        label_src = os.path.join(src_dir, label_file)
        label_dest = os.path.join(dest_dir, label_file)
        copyfile(label_src, label_dest)

# Criar diretórios 'val' e mover imagens
create_val_and_move_images(train_dir_apaus, os.path.join(val_dir,
"a_paus"))
create_val_and_move_images(train_dir_aouros, os.path.join(val_dir,
"a_ouros"))
create_val_and_move_images(train_dir_acopas, os.path.join(val_dir,
"a_copas"))
create_val_and_move_images(train_dir_oito_copas, os.path.join(val_dir,
"oito_copas"))
create_val_and_move_images(train_dir_cinco_ouros, os.path.join(val_dir,
```

```
"cinco_ouros"))
```

Configuração do Arquivo YAML

Foi criado um arquivo de configuração `data_custom.yaml` para especificar os diretórios de treino e validação, o número de classes (`nc`) e os nomes das classes (`names`):

```
data_custom_yaml = """
train:
  - {}
  - {}
  - {}
  - {}
  - {}
val: {}
nc: 5
names: ["a_paus", "a_ouros", "a_copas", "oito_copas", "cinco_ouros"]
""".format(train_dir_apaus, train_dir_aouros, train_dir_acopas,
train_dir_oito_copas, train_dir_cinco_ouros, val_dir)

with open("data_custom.yaml", "w") as f:
    f.write(data_custom_yaml)
```

Treinamento do Modelo

O treinamento da rede YOLO foi realizado utilizando a versão YOLOv8n pré-treinada como ponto de partida. O treinamento foi configurado para 30 épocas, com um tamanho de imagem de 640 e um batch size de 8:

```
model = YOLO("yolov8n.pt")

# Train the model
results = model.train(data="data_custom.yaml", epochs=30, imgsz=640,
batch=8)

# Save the trained model
model.save("yolov8n_trained.pt") # não usar esse e sim o que está
escrito na saida do treinamento do yolo
```

Quantidade de Imagens Utilizadas

Aqui está a lista das cartas e a quantidade de imagens utilizadas para cada uma delas. Insira a quantidade de imagens conforme apropriado:

- Ás de paus: 120
- Ás de ouros: 129
- Ás de copas: 171
- Oito de copas: 152
- Cinco de ouros: 138

Resultados do Treinamento

Após a preparação dos dados e a rotulação das imagens, iniciamos o treinamento da rede YOLO utilizando o script fornecido. O treinamento foi conduzido ao longo de 30 épocas e os principais resultados são apresentados a seguir:

Principais Métricas de Treinamento e Validação

Durante o treinamento, acompanhamos diversas métricas que são cruciais para avaliar o desempenho do modelo. As métricas incluem perdas de caixa delimitadora (`box_loss`), perda de classificação (`cls_loss`), perda de regressão de bordas (`dfl_loss`), precisão, recall, mAP50, e mAP50-95. Abaixo, destacamos os valores mais relevantes obtidos ao longo das 30 épocas de treinamento:

- **Perda de Caixa Delimitadora (train/box_loss)**:
  - Início (Época 1): 0.98258
  - Final (Época 30): 0.45311
- **Perda de Classificação (train/cls_loss)**:
  - Início (Época 1): 2.7742
  - Final (Época 30): 0.54432
- **Perda de Regressão de Bordas (train/dfl_loss)**:
  - Início (Época 1): 1.4976
  - Final (Época 30): 1.1465
- **Precisão (metrics/precision(B))**:
  - Início (Época 1): 0.34223
  - Final (Época 30): 0.96522
- **Recall (metrics/recall(B))**:
  - Início (Época 1): 0.5273
  - Final (Época 30): 0.95254
- **mAP50 (metrics/mAP50(B))**:
  - Início (Época 1): 0.30473
  - Final (Época 30): 0.97675
- **mAP50-95 (metrics/mAP50-95(B))**:
  - Início (Época 1): 0.14355
  - Final (Época 30): 0.84891

Análise dos Resultados

1. **Redução das Perdas**: Observamos uma redução significativa nas perdas de caixa delimitadora, classificação e regressão de bordas ao longo das épocas. Isto indica que o modelo conseguiu melhorar sua capacidade de prever com precisão as caixas delimitadoras e as classes das cartas de baralho ao longo do treinamento.
2. **Melhoria na Precisão e Recall**: Houve um aumento substancial tanto na precisão quanto no recall. A precisão aumentou de 0.34223 para 0.96522, e o recall aumentou de 0.5273 para 0.95254. Isso significa que o modelo se tornou muito mais eficaz em detectar corretamente as cartas de interesse (ases e reis).
3. **Aumento do mAP50 e mAP50-95**: A métrica mAP50 (mean Average Precision at IoU=0.50) melhorou significativamente, indicando que o modelo tem uma alta precisão na localização das cartas. A mAP50-95, que é uma métrica mais desafiadora, também mostrou um aumento significativo, refletindo a capacidade do modelo de manter uma boa performance em diferentes limiares de IoU (Intersection over Union).
4. **Convergência do Modelo**: As métricas de perda e as métricas de desempenho (precisão, recall e mAP) mostram uma tendência de convergência, indicando que o modelo está treinado de forma eficaz e que mais épocas de treinamento provavelmente não resultariam em melhorias significativas adicionais.

Os resultados do treinamento são promissores, indicando que o modelo YOLO treinado é eficaz na detecção de ases e reis nas imagens de cartas de baralho. A significativa redução nas perdas, combinada com o aumento na precisão, recall e mAP, demonstra que o modelo tem uma boa capacidade de generalização e desempenho robusto.

Aplicação para Comparação de Cartas de Baralho

A aplicação desenvolvida utiliza o modelo YOLO treinado para detectar e comparar cartas de baralho, determinando o vencedor com base no valor das cartas. A seguir, detalho o funcionamento da aplicação.

Importações e Carregamento do Modelo

Primeiro, importamos as bibliotecas necessárias e carregamos o modelo YOLO que foi treinado anteriormente.

```
from ultralytics import YOLO
import matplotlib.pyplot as plt
import cv2

# Carregar o modelo treinado
model =
YOLO("C:\\Users\\Clarissa\\Desktop\\prova2\\yolo\\train9\\weights\\best.
pt")
```

Mapeamento das Classes de Cartas para Valores Numéricos

Definimos um mapeamento que converte as classes das cartas detectadas pelo modelo para valores numéricos, facilitando a comparação.

```
# Definir um mapeamento para as classes de cartas e seus valores
numéricos
card_values = {
    'a_ouros': 1,
    'a_copas': 1,
    'a_paus': 1,
    'cinco_ouros': 5,
    'oito_copas': 8
}
```

Função de Carregamento e Predição

A função `load_and_predict` é responsável por carregar uma imagem, realizar a predição utilizando o modelo YOLO, e retornar a imagem original, a imagem com as predições e a classe predita.

```
def load_and_predict(image_path):
    # Realizar a predição com o modelo
    results = model(image_path)  # prever na imagem

    # Extrair a classe predita
    predicted_class = results[0].names[int(results[0].boxes[0].cls)]

    # Converter BGR para RGB para exibição com matplotlib
    predicted_image = results[0].plot()
    predicted_image_rgb = cv2.cvtcolor(predicted_image,
cv2.COLOR_BGR2RGB)

    # Ler a imagem original e converter para RGB
    original_image = cv2.imread(image_path)
    original_image_rgb = cv2.cvtcolor(original_image, cv2.COLOR_BGR2RGB)

    return original_image_rgb, predicted_image_rgb, predicted_class
```

Função Principal

A função `main` coleta os caminhos das imagens dos jogadores, realiza as predições, compara os valores das cartas e exibe os resultados.

```
def main():
    # Obter os caminhos das imagens dos jogadores
    image_path_player1 = input("Digite o caminho da imagem da carta do
Jogador 1: ")
    image_path_player2 = input("Digite o caminho da imagem da carta do
Jogador 2: ")

    # Carregar e prever para ambos os jogadores
    original_image1, predicted_image1, predicted_class1 =
load_and_predict(image_path_player1)
    original_image2, predicted_image2, predicted_class2 =
load_and_predict(image_path_player2)

    # Obter os valores numéricos das classes preditas
    value1 = card_values[predicted_class1]
    value2 = card_values[predicted_class2]

    # Comparar as classes preditas para determinar o vencedor
    if value1 > value2:
        winner = "Jogador 1 venceu!"
    elif value1 < value2:
        winner = "Jogador 2 venceu!"
    else:
        winner = "Empate!"

    # Exibir as imagens originais e preditas com o resultado
    plt.figure(figsize=(15, 7))

    plt.subplot(2, 2, 1)
    plt.title("Imagem Original - Jogador 1")
    plt.imshow(original_image1)
    plt.axis('off')

    plt.subplot(2, 2, 2)
    plt.title("Imagem Predita - Jogador 1")
    plt.imshow(predicted_image1)
    plt.axis('off')

    plt.subplot(2, 2, 3)
    plt.title("Imagem Original - Jogador 2")
    plt.imshow(original_image2)
    plt.axis('off')

    plt.subplot(2, 2, 4)
    plt.title("Imagem Predita - Jogador 2")
    plt.imshow(predicted_image2)
```

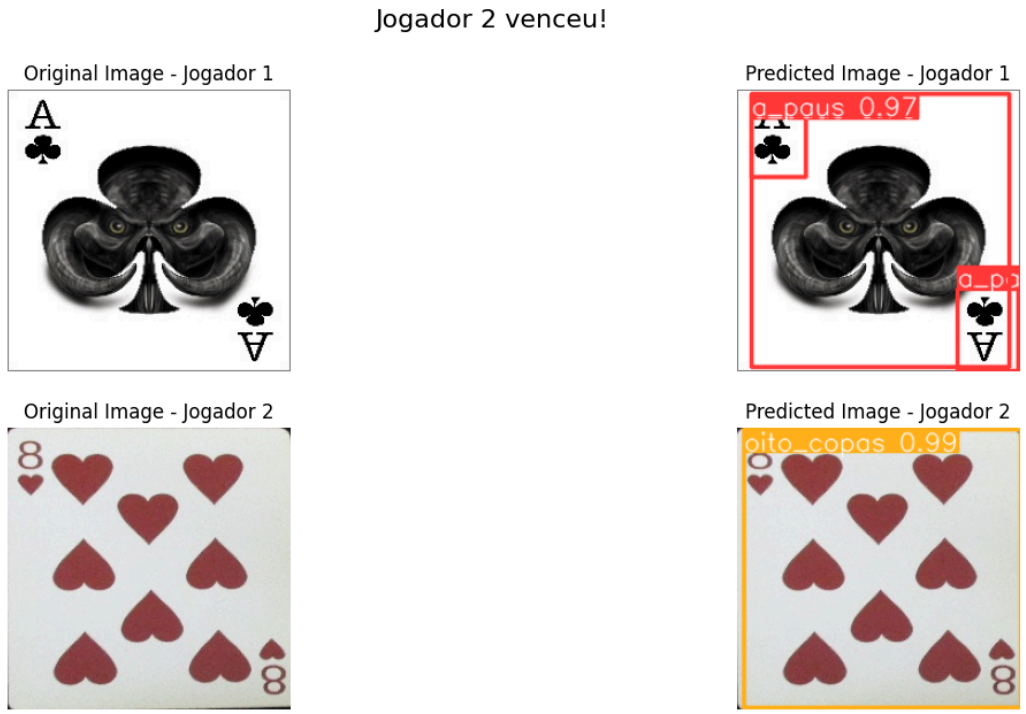
```
    plt.axis('off')

    plt.suptitle(winner, fontsize=16)
    plt.show()

if __name__ == "__main__":
    main()
```

Explicação do Funcionamento

1. **Carregamento do Modelo**: O modelo YOLO treinado é carregado a partir do caminho especificado.
2. **Predição**: Para cada imagem de entrada (uma para cada jogador), a função `load_and_predict` realiza a predição da classe da carta, converte a imagem predita para RGB e lê a imagem original.
3. **Comparação**: As classes preditas são convertidas para valores numéricos usando o dicionário `card_values`. Estes valores são então comparados para determinar o vencedor.
4. **Exibição**: As imagens originais e preditas são exibidas lado a lado usando o Matplotlib, juntamente com o resultado da partida.



Conclusão

Esta aplicação demonstra como usar um modelo YOLO treinado para detectar e comparar cartas de baralho em imagens, determinando o vencedor com base nos valores das cartas. O uso do YOLO permite uma detecção rápida e precisa, facilitando a automação do processo de comparação de cartas.