Projeto Final introdução a redes neurais - Detecção de Objetos em Câmeras Veiculares

Guilherme Pereira Campos RA:163787
Lucas Oliveira Nery de Araújo RA:158882
Universidade Federal de São Paulo

I. Introdução

Este projeto tem como objetivo desenvolver e avaliar um sistema de detecção de objetos utilizando redes neurais convolucionais (**CNNs**), com foco no modelo **YOLO (*You Only Look Once*)**, aplicado a câmeras veiculares. A proposta principal é identificar e localizar automaticamente elementos presentes em cenários de tráfego, como pedestres, veículos e sinais de trânsito, a partir de vídeos capturados por câmeras embarcadas em veículos.

Neste trabalho, o modelo **YOLOv8** será avaliado quanto à sua precisão na identificação de objetos em diferentes condições de tráfego e escalas. Para isso, será utilizado o dataset **BDD100K**, que contém vídeos capturados em ambientes urbanos. O modelo será testado inicialmente em **3 frames aleatórios** extraídos do dataset para validação pontual, seguido pela aplicação em **3 vídeos aleatórios** completos para avaliar sua performance em sequência. O sistema desenvolvido busca demonstrar a aplicabilidade do YOLO em situações práticas no campo da visão computacional.

II. Dataset

Neste projeto, os dados utilizados consistem em frames e vídeos extraídos do dataset **BDD100K**, que contém vídeos capturados por câmeras veiculares em ambientes urbanos com variadas condições de tráfego. Três exemplos de imagens extraídas deste dataset foram utilizados para verificar a qualidade dos dados.

O código implementado realiza as seguintes operações:

- Busca os vídeos com extensão `.mp4` a partir de um diretório específico.
- Seleciona aleatoriamente três vídeos.
- Extrai o primeiro frame de cada vídeo.
- Converte os frames de *BGR* para *RGB* para a visualização correta.
- Plota os frames lado a lado utilizando a biblioteca `matplotlib`.

A seguir, descrevem-se as funções e métodos utilizados no processo:

- **glob.glob**: Busca todos os arquivos que correspondem a um padrão especificado.
- **os.path.join**: Concatena partes de um caminho de forma segura, garantindo os separadores corretos para o sistema operacional.
- **random.sample**: Seleciona aleatoriamente um número específico de elementos de uma sequência sem repetição.
- **os.path.basename**: Extrai o nome do arquivo a partir de um caminho completo.
- **cv2.VideoCapture**: Abre e captura frames de um vídeo utilizando a biblioteca OpenCV.
- **cap.read()**: Retorna um booleano indicando se a leitura do frame foi bem-sucedida, juntamente com o frame capturado.
- **cv2.cvtColor**: Converte a representação de cores de uma imagem, utilizado para alterar de *BGR* para *RGB*.

III. Redes Convolucionais

A. Definição

As redes convolucionais (*CNN*) são uma classe de redes neurais especialmente eficazes para o processamento de dados estruturados, como imagens, séries temporais e outros. Elas utilizam operações de convolução para extrair características locais dos dados, reduzindo o número de parâmetros e melhorando a generalização.

B. Campos Receptivos Locais

- O cérebro não analisa a imagem de forma global; ao invés disso, células específicas se concentram em regiões específicas, formando os chamados *campos receptivos locais*.
- Essa abordagem é a base das **Redes Convolucionais (CNN)**.

C. Comparação com Redes MLP

- Ao contrário das redes *MLP*, as *CNNs* utilizam campos receptivos locais, e não globais.
- Há replicação dos neurônios em um mesmo filtro.
- Isso resulta em uma significativa redução do número de parâmetros livres.

D. Convolução

A operação de convolução pode ser definida como:

$$f_{og}(x) = \sum_{k=-M}^{M} f(x-k)\, g(k)$$

- Em redes convolucionais, a convolução pode ser aplicada a múltiplos canais.

E. Camada de Pooling

- A camada de *Pooling* reduz o mapa de características gerado pela camada convolucional.
- Diminui o custo computacional do modelo.
- Introduz invariância à translação, melhorando a robustez do modelo.

F. Stride e Padding

- **Stride**: deslocamento da janela durante a aplicação da convolução (por exemplo, *stride* = 1).
- **Padding**: preenchimento das bordas da imagem para controlar o tamanho do mapa de características resultante.
  - Pode ser definido como 0 (sem preenchimento), por replicação, com um valor constante ou de forma circular.

IV. YOLO (*You Only Look Once*)

YOLO (*You Only Look Once*) é um algoritmo de detecção de objetos que identifica e localiza múltiplos objetos em uma única passagem pela rede, tornando-o extremamente rápido e ideal para aplicações em tempo real, como em câmeras veiculares.

A. Arquitetura do YOLOv8 (Ultralytics)

A versão do YOLOv8 utilizada neste projeto segue uma arquitetura modular composta por três blocos principais:

a) Backbone:: O backbone é responsável por extrair características relevantes da imagem por meio de uma longa sequência de operações convolucionais. No YOLOv8, utiliza-se o **CSPDarknet53**, que introduz conexões parciais entre estágios (*Cross-Stage Partial Connections*) para melhorar o fluxo de informações e otimizar o treinamento. Além disso, emprega funções de ativação como a SiLU para aprimorar a extração de padrões, definida por

$$\text{SiLU}(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}.$$

b) Neck:: A etapa do neck integra as múltiplas escalas de features extraídas pelo backbone. O YOLOv8 utiliza o módulo **C2f**, uma evolução dos blocos CSP, para combinar informações semânticas de alto nível com detalhes espaciais de baixo nível. Adicionalmente, incorpora a camada **SPPF** (*Spatial Pyramid Pooling Fast*), que captura características em múltiplas escalas para melhorar a detecção de objetos em tamanhos variados. Por exemplo, a operação de *max pooling* pode ser descrita como

$$y[i,j] = \max_{m,n \in \text{janela}} x[i \cdot s + m,\ j \cdot s + n],$$

onde s representa o stride da janela.

c) Head:: A head é composta por camadas finais convolucionais que convertem as features integradas em predições. O YOLOv8 utiliza um head sem âncoras (*anchor-free*), eliminando a necessidade de caixas delimitadoras pré-definidas. Em cada célula da grade, a head realiza:

- Regressão das coordenadas das bounding boxes,
- Estimativa dos scores de confiança, e
- Classificação dos objetos.

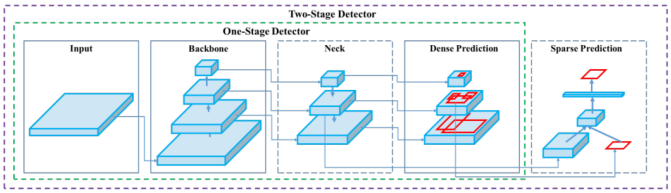

Fig. 1. Arquitetura do YOLO

B. Inovações no YOLOv8

O YOLOv8 introduz diversas inovações que aprimoram sua precisão e eficiência:

- **Atenção Espacial:** Mecanismos que destacam as partes mais relevantes da imagem para melhorar a localização dos objetos.
- **Fusão de Features:** O módulo **C2f** combina informações semânticas e espaciais em escalas variadas.
- **SPPF:** A camada **SPPF** capta características em múltiplas escalas com baixo custo computacional.
- **Treinamento com Dados Aumentados:** Técnicas como *mosaic augmentation* são empregadas para melhorar a capacidade de generalização do modelo.

C. Pré-Treinamento

O modelo vem pré-treinado no dataset **COCO (Common Objects in Context)**, que contém mais de 200.000 imagens anotadas em 80 classes. Durante o treinamento, o YOLOv8 aprendeu a identificar e localizar uma ampla variedade de objetos em cenários diversos, garantindo robustez e generalização para novas aplicações, como a detecção em vídeos de câmeras veiculares.

A saída fornecida pelo comando `model.info()` indica os seguintes pontos importantes sobre a arquitetura do modelo:

- **225 camadas:** O YOLOv8n possui um total de 225 camadas, incluindo operações convolucionais, de pooling, de ativação, entre outras, o que reflete a profundidade e complexidade da sua rede.
- **3.157.200 parâmetros:** Esse número representa o total de parâmetros aprendíveis, ou seja, os pesos e vieses ajustados durante o treinamento. Com cerca de 3,16 milhões de parâmetros, esta é uma das variantes mais leves, adequada para aplicações em tempo real.
- **0 gradientes:** Indica que, no momento da exibição, o modelo não estava computando gradientes (modo inferência), otimizando o uso de memória durante os testes.

- **8.9 GFLOPs:** Refere-se à complexidade computacional, mostrando que o modelo realiza aproximadamente 8,9 bilhões de operações de ponto flutuante para processar cada imagem, o que contribui para sua rapidez e eficiência.

Em resumo, o YOLOv8n é um modelo compacto e eficiente, com 225 camadas e cerca de 3,16 milhões de parâmetros, que requer 8,9 GFLOPs para processar uma imagem. Essa configuração o torna ideal para aplicações em tempo real, onde a velocidade e a precisão são essenciais.

V. Implementação

A. Imagens

Neste código, realizamos a inferência do modelo YOLOv8 em três imagens aleatórias de teste extraídas de vídeos de câmeras veiculares, com o objetivo de demonstrar a capacidade de detecção de objetos em diferentes cenários. A abordagem empregada consiste nas seguintes etapas:

1) **Seleção Aleatória de Vídeos:** São listados todos os vídeos presentes na pasta `BDD/test/camera_videos` e, em seguida, três vídeos são selecionados aleatoriamente. Essa seleção assegura que a demonstração seja representativa e abrangente.
2) **Extração dos Frames:** Para cada vídeo selecionado, utiliza-se o `OpenCV` para abrir o arquivo e extrair o primeiro frame, que servirá como imagem de teste para a aplicação do modelo.
3) **Aplicação do Modelo YOLOv8:** Cada frame é redimensionado para a resolução padrão de 1280x720, garantindo consistência no processamento. Posteriormente, o modelo é aplicado à imagem, realizando a detecção de objetos. O método `previsao_yolo[0].plot()` gera o frame anotado, contendo as caixas delimitadoras e os rótulos dos objetos detectados.
4) **Visualização dos Resultados:** Por fim, cada frame anotado é convertido de *BGR* para *RGB* para a visualização correta com `matplotlib` e exibido lado a lado, acompanhado do nome do vídeo de origem. Essa etapa facilita a inspeção visual dos resultados, demonstrando a eficácia do YOLOv8 na tarefa de detecção de objetos.

B. Vídeos

Nota: Os vídeos processados com o modelo fazendo a detecção encontram-se na pasta `videos`, que está dentro do arquivo *zip* que acompanha este relatório.

Nesta etapa do projeto, o objetivo foi aplicar o modelo YOLOv8 pré-treinado em três vídeos selecionados aleatoriamente a partir do dataset de câmeras veiculares. Essa abordagem demonstra como o modelo realiza a inferência em cada frame, anotando as detecções (bounding boxes, rótulos e scores) e produzindo vídeos processados que evidenciam a eficácia do sistema para aplicações em tempo real.

a) Etapas do Processo:

1) **Seleção Aleatória dos Vídeos:** São listados os arquivos de vídeo contidos na pasta `BDD/test/camera_videos` e, em seguida, três vídeos são escolhidos de forma aleatória, garantindo que diferentes cenários sejam avaliados.
2) **Processamento dos Vídeos:** Para cada vídeo selecionado:
   - O vídeo é aberto utilizando `cv2.VideoCapture`.
   - Cada frame é redimensionado para uma resolução fixa de **1280x720**, assegurando a consistência dos dados.
   - O modelo YOLOv8 é aplicado a cada frame para realizar a detecção dos objetos, e o método `plot()` gera o frame anotado com as caixas delimitadoras e os rótulos dos objetos detectados.
   - Os frames processados são gravados em um novo vídeo utilizando o `cv2.VideoWriter` configurado para **30 FPS**.
3) **Conversão dos Vídeos:** Após o processamento, cada vídeo é convertido para um formato compatível com o player do IPython. Para tanto, utiliza-se a ferramenta **FFmpeg** para converter o vídeo para o codec **H264 (libx264)**, definindo o formato dos pixels como **yuv420p** e aplicando a flag **+faststart** para otimizar a reprodução.
4) **Exibição dos Vídeos Processados:** Por fim, os vídeos convertidos são exibidos diretamente no notebook, permitindo visualizar os resultados da inferência e a performance do modelo em cenários reais extraídos das câmeras veiculares.

VI. Conclusão

Neste projeto, desenvolveu-se um sistema de detecção de objetos utilizando redes neurais convolucionais (CNNs), com foco no modelo **YOLOv8 (*You Only Look Once*)**, aplicado a vídeos capturados por câmeras veiculares. O objetivo principal era identificar e localizar automaticamente elementos presentes em cenários de tráfego, como pedestres, veículos e sinais de trânsito, demonstrando a eficiência do modelo em situações reais.

O modelo YOLOv8, pré-treinado no dataset COCO, mostrou-se altamente eficaz para a tarefa de detecção em tempo real, combinando precisão e velocidade. Sua arquitetura modular – composta por *backbone*, *neck* e *head* – permite uma extração eficiente de características e predições rápidas, mesmo em vídeos de alta resolução. Além disso, a integração com ferramentas como OpenCV e FFmpeg possibilitou o processamento de múltiplos vídeos e a visualização dos resultados diretamente

no notebook.

Os resultados obtidos evidenciam que o YOLOv8 é uma solução robusta para sistemas de visão computacional aplicados a câmeras veiculares. Ele é capaz de identificar objetos relevantes em cenários dinâmicos e variados. Este projeto demonstra como redes neurais convolucionais modernas podem ser aplicadas para resolver problemas práticos no campo da visão computacional.