

Visão Computacional Aplicada ao Controle do Tráfego Urbano: Detecção, Classificação e Contagem de Veículos

Guilherme Ferreira

Especialização em Inteligência Artificial e Computacional
Universidade Federal de Viçosa

guilherme.ferreira2@ufv.br

Resumo:

Análise e gerenciamento do fluxo de veículos em centros urbanos pode ser feita por meio de ferramentas de inteligência artificial, mormente aquelas que adotam técnicas avançadas, algumas recém descobertas, na área da visão computacional, destinadas à detecção, classificação, rastreamento e contagem de objetos em determinada cena exibida por imagem ou vídeo. No presente trabalho, apresentamos um modelo inicial para explorar o potencial da visão computacional para a gestão inteligente do tráfego de veículos em uma cidade de médio porte.

I. INTRODUÇÃO

O congestionamento é um dos grandes problemas dos centros urbanos, devido ao grande volume de veículos que trafegam diariamente por ruas e avenidas. Como consequência, provoca atrasos, frustração, estresse e, em alguns casos, violência. Sem contar os acidentes, que geram perdas em vidas e patrimônio, e o impacto nas condições climáticas. Por esse motivo, a implantação de sistemas inteligentes de controle do fluxo de veículos pode reduzir os congestionamentos e melhorar a qualidade de vida dos habitantes[1].

O monitoramento inteligente do trânsito apresenta alguns desafios, tais como o reconhecimento, a localização, o acompanhamento e a contagem de veículos, em situações de dificuldade devido às condições ambientais (qualidade da luz, sombreamento, ocultamento etc), densidade dos objetos, dentre outros fatores[2].

Juiz de Fora é um município com mais de 500 mil habitantes, situado na Zona da Mata Mineira, com uma topografia montanhosa que restringe a expansão da malha viária e gera reflexos negativos para a mobilidade urbana. No último levantamento de frota realizado pelo IBGE, em 2022, possuía mais de 290 mil veículos em circulação[3].

A Secretaria de Mobilidade Urbana conta com um sistema de câmeras para monitoramento do trânsito, instaladas nas principais vias e cruzamentos do centro da cidade, que podem ser acessadas livremente, através do endereço web:

https://pjf.mg.gov.br/secretarias/smu/servicos/transito_agora/index.php. Os arquivos de 60 segundos são disponibilizados a cada 2 minutos, portanto, não podemos dizer que as imagens sejam em tempo real.



O presente trabalho visa apresentar um projeto que utiliza ferramentas, métodos e técnicas de visão computacional para o controle de tráfego urbano, para detecção, classificação, rastreamento e contagem de veículos, de modo a fornecer dados para a tomada de decisão informada de gestores públicos. Será utilizado o *framework* **YOLOv8**, a ser detalhado a seguir.

Essencialmente, o projeto consiste na captura de imagens de vídeo, em diversos pontos da cidade e em horários diferenciados, tais como horários de pico, dias comerciais, feriados e finais de semana, em diferentes condições climáticas, para abranger as mais variadas situações do trânsito, permitindo assim gerar dados abrangentes e consistentes, úteis para a gestão do trânsito.

II. DEFINIÇÕES

A. Visão Computacional

Visão Computacional é uma área de aplicação da **Deep Learning** - subconjunto de aprendizado de máquina ou **machine learning** – que busca modelar matematicamente os princípios da visão humana tridimensional.

As pesquisas avançadas permitem a utilização da visão computacional em diversas aplicações do mundo real: reconhecimento de caracteres manuscritos, processamento de imagens diagnósticas, veículos autônomos, reconhecimento facial e vigilância, dentre outros usos.

B – Detecção de objetos

Detecção de objetos é uma técnica de visão computacional para localizar determinado objeto na imagem ou vídeo. Fornece as coordenadas espaciais do objeto, normalmente limitadas por uma figura geométrica, em geral retangular, bem como a indicação da classe a que pertence.

Qual a diferença entre **classificação**, **detecção** e **segmentação** de objetos?

Classificação é um tipo de reconhecimento de imagem que identifica a classe do objeto presente na imagem exibida.

Detecção é um tipo de reconhecimento de imagem utilizado para definir as coordenadas espaciais do objeto e a classe a que pertence.

Segmentação é uma técnica de reconhecimento de imagem para identificar objetos em uma imagem tomando o *pixel* como unidade de análise, embora os objetos também sejam delimitados por retângulos.

III – Breve histórico

Esta área de pesquisa teve início na década de 1970, quando os pesquisadores começaram a desenvolver métodos automatizados para detecção de objetos[4].

Os primeiros algoritmos se dedicaram ao reconhecimento de bordas e cantos. A primeira detecção real de objetos foi proposta por **Viola Jones**, em 2001, que utilizou janelas retangulares deslizantes para extrair características de imagens. Mais tarde este método foi utilizado para reconhecimento facial em *smartphones* e câmeras fotográficas.

Em 2005, **Dalal e Triggs** desenvolveram o algoritmo denominado HOG(*Histogram oriented gradient*), que utiliza um método que dá ênfase à forma dos objetos, ao extrair o gradiente e orientação das bordas para detectar características de pessoas em uma imagem.

Então sobreveio a revolução das redes neurais convolucionais em 2012, quando a rede **AlexNet16** venceu o desafio **Imagenet** de visão computacional, que na realidade constituiu em um problema de classificação de imagens, não de detecção.

Depois vieram as redes **RCNN**, que apresentaram bons resultados ao usarem regiões seletivas para classificar objetos, apesar da lentidão do processamento.

Em seguida, foram lançados os modelos **Fast-RCNN** e **Faster-RCN**, mais rápidos, mas não ainda em tempo real.

Em 2015, descrito originalmente por Joseph Redmon et al., foi lançado **YOLO**, modelo de rede neural convolucional que superou os demais existentes, em termos de performance e precisão, ao ser executado em tempo real, atualmente considerado o *estado-da-arte* em visão computacional[5].

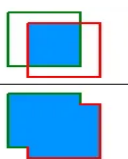
Como abordagem diferente, empregava uma única passagem da imagem de entrada para fazer previsões, daí o acrônimo para *You Only Look Once*(YOLO). A **RCNN**, por exemplo, utiliza a proposta de seleção de regiões para realizar múltiplas interações com a mesma imagem, enquanto a **YOLO** realiza a mesma tarefa com uma única interação. Isto a torna o algoritmo mais eficiente, mesmo que sua arquitetura também seja baseada em redes neurais convolucionais[6].

IV – Métricas de avaliação de desempenho

Há duas métricas principais utilizadas para avaliar o desempenho do modelo de detecção de objetos.

Localização

IoU: Intersection Over Union – determina se a predição foi realizada corretamente; ajuste representado pela razão entre a Área de Interseção e a Área de União de duas caixas delimitadoras.

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{área de interseção}}{\text{área de união}}$$


Source: <https://github.com/rafaelpadilla/Object-Detection-Metrics>

Revela a precisão da caixa delimitadora ou o quanto se aproxima dos limites verdadeiros do objeto. Seu valor varia de 0 a 1: se as caixas se sobrepõem, então a detecção é perfeita, o valor de IoU=1. Se se sobrepõem em alguma superfície, o valor de IoU varia de 0 a 1. Se não houver nenhuma sobreposição, então o valor de IoU é igual a 0.

Classificação

AP – *Average Precision* (para classe unitária)

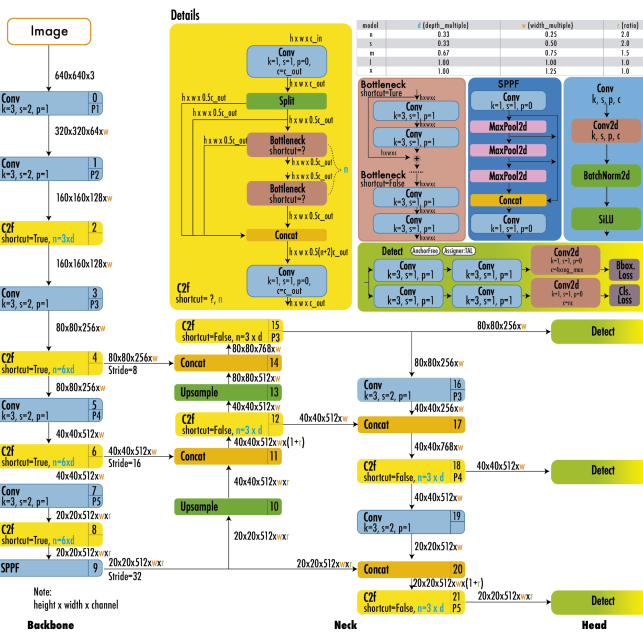
mAP – Mean Average Precision (para múltiplas classes)

A **AP** fornece uma medida única que resume o desempenho de um modelo ao levar em conta a precisão em diferentes níveis de *recall*. É uma métrica útil para avaliar a capacidade de um modelo de detecção de objetos em retornar detecções precisas em relação ao número total de instâncias positivas no conjunto de dados[7].

IV – MATERIAL E MÉTODOS

Vamos utilizar oitava versão do algoritmo de detecção de objetos denominado **YOLO**. O código será executado no Google Colab.

Arquitetura YOLOv8



A arquitetura usa um backbone CSPDarknet53 modificado. Uma camada de agrupamento rápido de pirâmide espacial (SPPF) acelera a computação agrupando recursos em um mapa de tamanho fixo. Cada convolução possui normalização em lote e ativação SiLU. O cabeçote é desacoplado para processar tarefas de “*objetividade*”, classificação e regressão de forma independente.

Por definição, “*objetividade*” refere-se à qualidade ou probabilidade de que uma região em uma imagem contenha um objeto.

YOLOv8 também fornece um modelo de segmentação semântica denominado modelo YOLOv8-Seg. O backbone é um extrator de recursos CSPDarknet53, seguido por um módulo C2f em vez da arquitetura tradicional de pescoço YOLO. O módulo C2f é seguido por dois cabeçotes de segmentação, que aprendem a prever as máscaras de segmentação semântica para a imagem de entrada. O modelo possui cabeçotes de detecção semelhantes ao YOLOv8, consistindo em cinco módulos de detecção e uma camada de predição. O modelo YOLOv8-Seg alcançou resultados de

última geração em vários benchmarks de detecção de objetos e segmentação semântica, mantendo alta velocidade e eficiência[9].

YOLOv8 possui cinco modelos diferenciados pelo número de parâmetros – nano(n), pequeno(s), médio(m), grande(l) e extra grande(x). Cada variante deve ser utilizada para uma tarefa específica, seja detecção, classificação ou segmentação de objetos.

▼ Detection

Model	size (pixels)	mAP _{val} 50-95	Speed CPU (ms)	Speed T4 GPU (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	-	-	3.2	8.7
YOLOv8s	640	44.9	-	-	11.2	28.6
YOLOv8m	640	50.2	-	-	25.9	78.9
YOLOv8l	640	52.9	-	-	43.7	165.2
YOLOv8x	640	53.9	-	-	68.2	257.8

Fonte: <https://blog.roboflow.com/whats-new-in-yolov8/>

Neste projeto, será utilizado o modelo pré-treinado *yolov8n.pt*, para detecção de objetos, que prioriza a velocidade de processamento em detrimento da precisão.

Arquivo de entrada

Para teste, utilizaremos um arquivo baixado do portal da Prefeitura Municipal de Juiz de Fora, com a captura do vídeo da câmara 16 (<http://camerasjf.gctnet.com.br:8880/cameras/?cam=16>), do cruzamento da Avenida Brasil com a Avenida Rio Branco, do dia 16/10/2023, com início às 10:49:32, com duração de 60s, resolução 1920x1080, Codec H.265, 29,97 fps, 448 kbps, com a extensão .mkv.

O arquivo de vídeo é fornecido em um *container* **mkv**, que é um envelope que incorpora múltiplos formatos multimídia[10].

Pré-processamento

Convertemos o arquivo de vídeo disponível para o formato *mp4*, utilizando o seguinte *script*:

```
import os, sys, re

start_time = "00:00:00" # Substitua pelo tempo de início desejado
end_time = "00:01:00" # Substitua pelo tempo de término desejado

output_file_path = re.search("^[/\\].+[/\\]", video_file_path)
output_file_path_raw = output_file_path.group(0)
delsplit = re.search("\\\\(?:.?(?!\\/))+", video_file_path)
filename = re.sub("^[/\\]", "", delsplit.group(0))
filename_raw = re.sub("\\.+", "", filename)
file_extension = re.search("\\.(\\w+)$", filename)
file_extension_raw = file_extension.group(0)

os.environ['inputFile'] = video_file_path
os.environ['outputPath'] = output_file_path_raw
os.environ['startTime'] = start_time
os.environ['endTime'] = end_time
os.environ['fileName'] = filename_raw
os.environ['fileExtension'] = file_extension_raw

ffmpeg -hide_banner -n -i "$inputFile" -c copy -strict -2 \
"$outputPath"/temp.mkv
ffmpeg -hide_banner -n -i "$outputPath"/temp.mkv -c copy -strict -2 \
"$outputPath"/"$fileName".mp4
```

Implementação

Instalamos o pacote **Ultralytics**, que contém a classe YOLO:

```
# Instalação do pacote Ultralytics
!pip install ultralytics
```

Importamos as bibliotecas necessárias:

```
# Importação das bibliotecas
import cv2 as cv
from glob import glob
import os, sys, re
from ultralytics import YOLO
import math
```

Uma variável recebe o arquivo de entrada:

```
# Carregamos os vídeos em uma variável
videos = glob('/content/drive/MyDrive/ELT578/Projeto_Final/Videos/*.mp4')
```

Criamos um modelo de rede neural(YOLOv8 é um grupo de modelos de rede neural, criados e treinados com a biblioteca **Pytorch**, exportados para arquivos com a extensão *.pt*):

```
# Definição do modelo
model_pretrained = YOLO('yolov8n.pt')
```

Lemos o arquivo de entrada:

```
# Leitura do arquivo de vídeo
video = cv.VideoCapture(videos[0])
```

Utilizamos a classe *VideoWriter* do **opencv** para criar o objeto de saída, no formato **mp4**:

```
# Definimos o codec e criamos o objeto VideoWriter
fourcc = cv.VideoWriter_fourcc(*'mp4v')
out = cv.VideoWriter('/content/drive/MyDrive/ELT578/Projeto_Final/Videos/teste 2.mp4',
                    fourcc, 20.0, size)
```

Listamos as classes do dataset COCO (<https://cocodataset.org/#home>)[11]:

```
classNames = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus",
"train", "truck", "boat", "traffic light", "fire hydrant",
"stop sign", "parking meter", "bench", "bird", "cat",
"dog", "horse", "sheep", "cow", "elephant", "bear", "zebra",
"giraffe", "backpack", "umbrella", "handbag", "tie", "suitcase",
"frisbee", "skis", "snowboard", "sports ball", "kite",
"baseball bat", "baseball glove", "skateboard", "surfboard",
"tennis racket", "bottle", "wine glass", "cup", "fork", "knife",
"spoon", "bowl", "banana", "apple", "sandwich", "orange",
"broccoli", "carrot", "hot dog", "pizza", "donut", "cake",
"chair", "sofa", "pottedplant", "bed", "diningtable", "toilet",
"tvmonitor", "laptop", "mouse", "remote", "keyboard",
"cell phone", "microwave", "oven", "toaster", "sink",
"refrigerator", "book", "clock", "vase", "scissors",
"teddy bear", "hair drier", "toothbrush"]
```

Criamos dicionários auxiliares:

```
# Dicionário para rastrear IDs únicos por tipo de veículo
unique_ids = {'car': set(), 'bus': set(), 'motorbike': set(), 'truck': set()}

# Mapeamento de códigos de classe para nomes de classe
class_names = {2: 'car', 5: 'bus', 3: 'motorbike', 7: 'truck'}
```

Realizamos a leitura dos *frames*:

```
# Leitura dos frames
ret, frame = video.read()
```

Ao processar a entrada, o modelo retorna um *array* de resultados para cada *frame*:

```
# Detecção e rastreamento de objetos
results = model_pretrained.track(frame, persist=True)
```

Mapeamos algumas propriedades da caixa delimitadora:

```
for detection in results[0]:
    boxes = detection.boxes
    for box in boxes:
        # Bounding Box
        x1, y1, x2, y2 = map(int, box.xyxy[0])
        cv.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 255), 3)

        # Confidence
        conf = math.ceil((box.conf[0] * 100)) / 100

        # Class Name
        cls = int(box.cls[0])
        current_class = classNames[cls]

        # id Number
        if box.id is not None:
            vehicle_id = int(box.id[0])

        # Adiciona o ID ao conjunto correspondente se ainda não estiver presente
        if cls in class_names:
            unique_ids[class_names[cls]].add(vehicle_id)
```

Adicionamos rótulos às caixas delimitadoras:

```
# Verifica se a classe é relevante e a confiança suficiente
if current_class in class_names.values() and conf > 0.3:
    cv.putText(frame, f'{current_class} {conf}',
               (max(0, x1), max(35, y1)),
               fontFace=cv.FONT_HERSHEY_SIMPLEX,
               fontScale=0.6, color=(255, 255, 255),
               thickness=2)
```

Inserimos na tela a contagem de veículos por tipo:

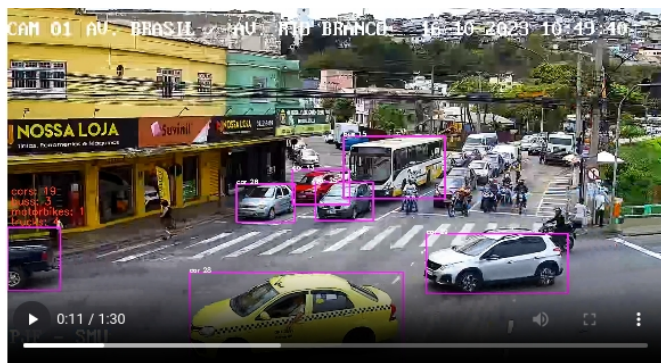
```
# Adiciona texto com a quantidade total de veículos por tipo |
y_offset = frame_height // 2 + 30
for class_code, class_name in class_names.items():
    if class_name in unique_ids:
        text = f'{class_name}s: {len(unique_ids[class_name])}'
        cv.putText(frame, text, (10, y_offset),
                   cv.FONT_HERSHEY_SIMPLEX, 1, (50, 50, 255), 2)
        y_offset += 30
```

Salvamos o arquivo de saída, liberamos os objetos de captura e gravação e fechamos todos os frames:

```
# save video
out.write(frame)

out.release()
video.release()

# Closes all the frames
cv.destroyAllWindows()
```

V – CONSIDERAÇÕES FINAIS

Neste trabalho, realizamos uma demonstração básica a respeito de como a visão computacional pode ser utilizada como ferramenta de gestão inteligente do trânsito nas grandes cidades, com impacto na qualidade de vida dos moradores e motoristas.

As câmeras de monitoramento existentes podem auxiliar na extração de dados adequados para análise do fluxo de veículos, que permita a temporização de semáforos de acordo com a demanda de determinado horário ou dia da semana, por exemplo.

Para a realização de inferências e tomada de decisão, um projeto robusto deve levar em consideração diferentes horários do dia, diversos dias da semana, em variadas condições de clima e épocas do ano. Ademais, levando-se em conta que as câmeras rotacionam 360°, deve-se elaborar rotina que avalie o fluxo dos veículos em diferentes direções.

Em nosso código, não foram utilizadas máscaras para detectar veículos estacionados, de modo que a contagem de veículos ficou prejudicada em alguns momentos, quanto à precisão.

Contudo, como protótipo, acreditamos que conseguimos demonstrar as funcionalidades básicas da ferramenta que é considerada o *estado-da-arte* no campo da visão computacional.

Exibimos as funcionalidades do modelo *yolov8n.pt*, capazes de realizar a detecção, classificação e rastreamento de objetos.

Finalizando, adicionamos rotinas para extração das propriedades das caixas delimitadoras, com exibição de características como *id*, *nível de confiança (probabilidade)* da predição e *classe* dos objetos detectados, cujas trajetórias puderam ser acompanhadas através das imagens em movimento (*tracking*).

AGRADECIMENTOS

Agradeço à Professora Talita Stéfani Zunino Santana, que ministrou a disciplina ELT 578 – Análise de Imagens e vVsão Computacional, no curso de Pós-Graduação em Inteligência Artificial e Computacional da Universidade Federal de Viçosa, pela dedicação e empatia.

REFERÊNCIAS

- [1] Kutlimuratov, A.; Khamzaev, J.; Kuchkorov, T.; Anwar, M.S.; Choi, A. Applying Enhanced Real-Time Monitoring and Counting Method for Effective Traffic Management in Tashkent. *Sensors* 2023, 23, 5007. <https://doi.org/10.3390/s23115007>
- [2] Lin, Cheng-Jian; Jeng, Shiou-Yun and Lioa, Hong-Wei, *AA Real-Time Vehicle Counting, Speed Estimation, and Classification System Based on Virtual Detection Zone and YOLO*, Hindawi-Mathematical Problems in Engineering Volume 2021, Article ID 1577614, 10 pages. <https://doi.org/10.1155/2021/1577614>
- [3] Ministério da Infraestrutura, SENATRAN - Secretaria Nacional de Trânsito – 2022: <https://cidades.ibge.gov.br/brasil/pesquisa/22/28120>. Último acesso realizado em 13/12/2023.
- [4] Szeliski, Richard: “Computer Vision Algorithms and Applications “. 2nd Ed. 2022, Springer Nature Switzerland AG.
- [5] Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
- [6] Chai, Junyi; Zeng, Hao; Li, Anming; Ngai, Eric W.T., “Deep learning in computer vision: A critical review of emergin techniques and application scenarios”, *Machine Learning with Applications*, Vol. 6, 15 Dec 2021, 100134, ScienceDirect Elsevier.
- [7] Padilla, R.; Passos, W.L.; Dias, T.L.B.; Netto, S.L.; da Silva, E.A.B. A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. *Electronics* 2021, 10, 279. <https://doi.org/10.3390/electronics10030279>
- [8] Terven, J.; Córdova-Esparza, D.-M.; Romero-González, J.-A. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Mach. Learn. Knowl. Extr.* 2023, 5, 1680-1716. <https://doi.org/10.3390/make5040083>
- [9] Lhomme, S., Bunkus, M., and D. Rice, "Media Container Specifications", Work in Progress, Internet-Draft, draftietf-cellarmatroska-10, 1 May 2022. <https://datatracker.ietf.org/doc/html/draft-ietf-cellarmatroska-10>.
- [10] Tsung-Yi Lin et al., 2014. Microsoft COCO: Common Objects in Context. CoRR, abs/1405.0312. Available at: <http://arxiv.org/abs/1405.0312>.