

Detecção de Capacetes com YOLOv8

Matéria: Segmentação e Estrutura de Reconhecimento de Padrões Contextualizada

Aluno: Renan Pantaleão Rosa

Professor: Prof. Alexandre Barbosa de Souza

1. Introdução

O avanço das redes neurais convolucionais (CNNs), trouxe importantes soluções para tarefas de detecção de objetos em imagens e vídeos. Entre essas técnicas, destaca-se o YOLO, que permite realizar detecção em tempo real com alta acurácia.

Este projeto teve como objetivo implementar um sistema de detecção de capacetes em motociclistas utilizando o modelo YOLOv8, explorando desde o treinamento supervisionado até a avaliação do desempenho, com aplicação em imagens e vídeos.

2. Metodologia

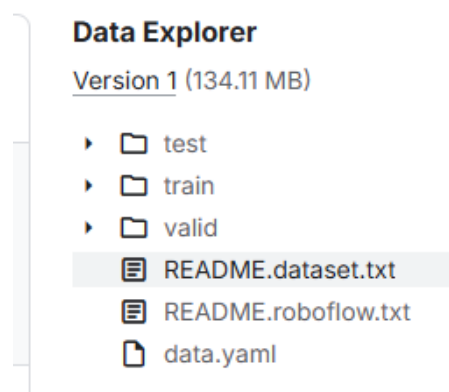
2.1. Ferramentas e Ambiente

- **Linguagem:** Python
- **Bibliotecas:** torch, ultralytics (YOLOv8), opencv, numpy, matplotlib, moviepy, ffmpeg
- **Ambiente:** Google Colab (CPU/GPU)
- **Gerenciamento de dados:** Kaggle API

2.2. Dataset

Foi utilizado o dataset público "**helmetbehincode**", disponível no Kaggle. O dataset foi estruturado em três subconjuntos:

- **train:** para treinamento,
- **valid:** para validação,
- **test:** para testes adicionais.





ABUZAR KHAN · UPDATED 8 MONTHS AGO

Helmet Detection Using YOLOv8

Helmet detection photos, which classifies into two class i.e. Helmet, No_helmet

Url: <https://www.kaggle.com/datasets/abuzarkhaan/helmetbehncode>

Um arquivo **data.yaml** foi criado para definir o caminho do dataset e as classes:

```
data.yaml X
1
2 path: /content/helmetbehncode
3 train: train/images
4 val: valid/images
5
6 names:
7   0: biker
8   1: helmet
9   2: passenger
10
```

2.3. Treinamento do modelo

- Modelo base: `yolo8n.pt` (YOLOv8 nano).



- Hiperparâmetros principais:
 - **Épocas:** 50 (ideal, utilizado 5 no vídeo)
 - Quantidades de vezes em que a máquina vai dar volta treinando pelo dataset
 - **Batch size:** 16
 - Quantidade de imagens sendo lidas pela máquina durante o treinamento,
 - **Tamanho da imagem:** 640 x 640
 - Tamanho das imagens em sua rede

- Comando utilizado no Colab:

```
In [ ]: from ultralytics import YOLO
import random
import glob
import os
from IPython.display import Image, display

model = YOLO('yolov8n.pt')

results = model.train(
    data='/content/helmetbehincode/data.yaml',
    epochs=25,
    imgsz=640,
    batch=16,
    name='helmet_detection_extended',
    project='helmet-detection'
)

modelo_treinado = YOLO('/content/helmet-detection/helmet_detection_extended/weights/best.pt')

















image_paths = glob.glob('/content/helmetbehincode/valid/images/*.jpg')

for i in range(2):
    img_path = random.choice(image_paths)

    pred = modelo_treinado.predict(source=img_path, conf=0.25, save=True, verbose=False)
    pred_path = os.path.join(pred[0].save_dir, os.path.basename(img_path))

    display(Image(filename=pred_path))
    print(f"Exibind imagens com predições: {os.path.basename(img_path)}")
```

- Treinamento em progresso:

| | | | | | | |
|-------------|---------------|-------------------|--------------------|-------------------|-----------------|---|
| Epoch 6/25 | GPU_mem 0G | box_loss 1.386 | cls_loss 1.161 | dfl_loss 1.653 | Instances 66 | Size 640: 100%  102/102 [20:45<00:00, 12.21s/it] |
| | Class | Images | Instances | Box(P | R | mAP50 mAP50-95): 100%  5/5 [00:46<00:00, 9.34s/it] |
| Epoch 7/25 | GPU_mem 0G | box_loss 1.371 | cls_loss 1.095 | dfl_loss 1.633 | Instances 77 | Size 640: 100%  102/102 [20:51<00:00, 12.27s/it] |
| | Class | Images | Instances | Box(P | R | mAP50 mAP50-95): 100%  5/5 [00:44<00:00, 8.95s/it] |
| Epoch 8/25 | GPU_mem 0G | box_loss 1.337 | cls_loss 1.049 | dfl_loss 1.591 | Instances 78 | Size 640: 100%  102/102 [20:51<00:00, 12.26s/it] |
| | Class | Images | Instances | Box(P | R | mAP50 mAP50-95): 100%  5/5 [00:45<00:00, 9.01s/it] |
| Epoch 9/25 | GPU_mem 0G | box_loss 1.32 | cls_loss 1.023 | dfl_loss 1.59 | Instances 84 | Size 640: 100%  102/102 [20:50<00:00, 12.26s/it] |
| | Class | Images | Instances | Box(P | R | mAP50 mAP50-95): 100%  5/5 [00:44<00:00, 8.85s/it] |
| Epoch 10/25 | GPU_mem 0G | box_loss 1.309 | cls_loss 0.9951 | dfl_loss 1.58 | Instances 58 | Size 640: 100%  102/102 [20:42<00:00, 12.18s/it] |
| | Class | Images | Instances | Box(P | R | mAP50 mAP50-95): 100%  5/5 [00:41<00:00, 8.31s/it] |
| Epoch 11/25 | GPU_mem 0G | box_loss 1.291 | cls_loss 0.9614 | dfl_loss 1.565 | Instances 84 | Size 640: 100%  102/102 [20:43<00:00, 12.19s/it] |
| | Class | Images | Instances | Box(P | R | mAP50 mAP50-95): 100%  5/5 [00:44<00:00, 8.90s/it] |
| Epoch 12/25 | GPU_mem 0G | box_loss 1.284 | cls_loss 0.9356 | dfl_loss 1.555 | Instances 69 | Size 640: 100%  102/102 [20:44<00:00, 12.20s/it] |
| | Class | Images | Instances | Box(P | R | mAP50 mAP50-95): 100%  5/5 [00:42<00:00, 8.48s/it] |
| Epoch 13/25 | GPU_mem 0G | box_loss 1.256 | cls_loss 0.8888 | dfl_loss 1.536 | Instances 68 | Size 640: 100%  102/102 [20:41<00:00, 12.17s/it] |
| | Class | Images | Instances | Box(P | R | mAP50 mAP50-95): 100%  5/5 [00:43<00:00, 8.69s/it] |

2.4. Avaliação e Métricas

Após o treinamento, foi realizada a avaliação do modelo utilizando métricas padrão de detecção de objetos, com destaque para o mAP (mean Average Precision) em diferentes limiares de IoU (Intersection over Union).

As principais métricas obtidas foram:

mAP@0.5: 0.93

Excelente desempenho na detecção básica, ou seja, quando o limiar de IoU é de 50%. Isso indica que o modelo possui alta capacidade de detectar corretamente objetos-alvo com precisão razoável.

mAP@0.5:0.95: 0.46

Bom desempenho em múltiplos níveis de IoU, demonstrando robustez mesmo quando se exige maior precisão na sobreposição entre as caixas de detecção e as caixas reais.

Precisão e Recall:

- **Precisão (Precision):** Variou entre 0.83 e 0.99 para as classes principais.
- **Recall:** Variou entre **0.80 e 1.00**.
A classe “helmet” foi a mais precisa e completa, seguida de “bike”, com a classe “passenger” apresentando os resultados mais baixos.

F1-Score:

Os melhores valores de F1-Score para todas as classes giraram em torno de 0.92, indicando um excelente equilíbrio entre precisão e recall.

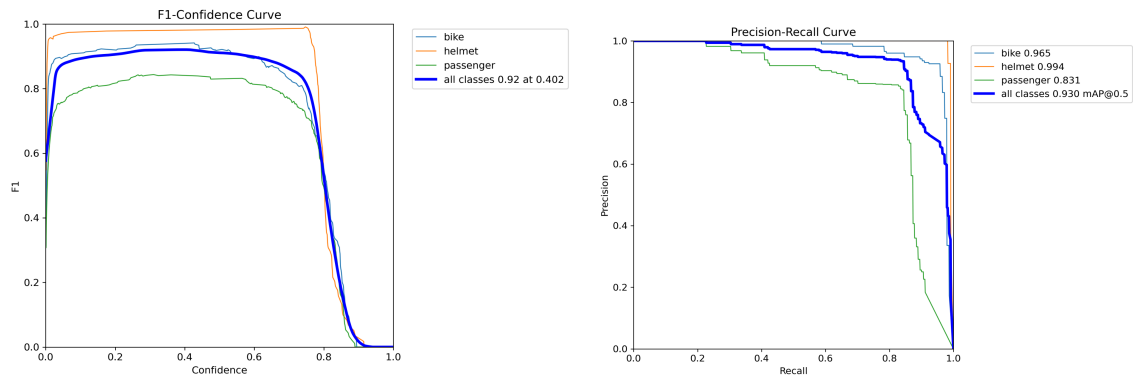
Perdas (Losses):

- **box_loss:** aproximadamente 1.3
- **cls_loss:** aproximadamente 1.0
- **dfl_loss:** aproximadamente **1.6**

As perdas demonstraram estabilidade ao longo das épocas, indicando que o modelo foi aprendendo de forma consistente sem overfitting perceptível.

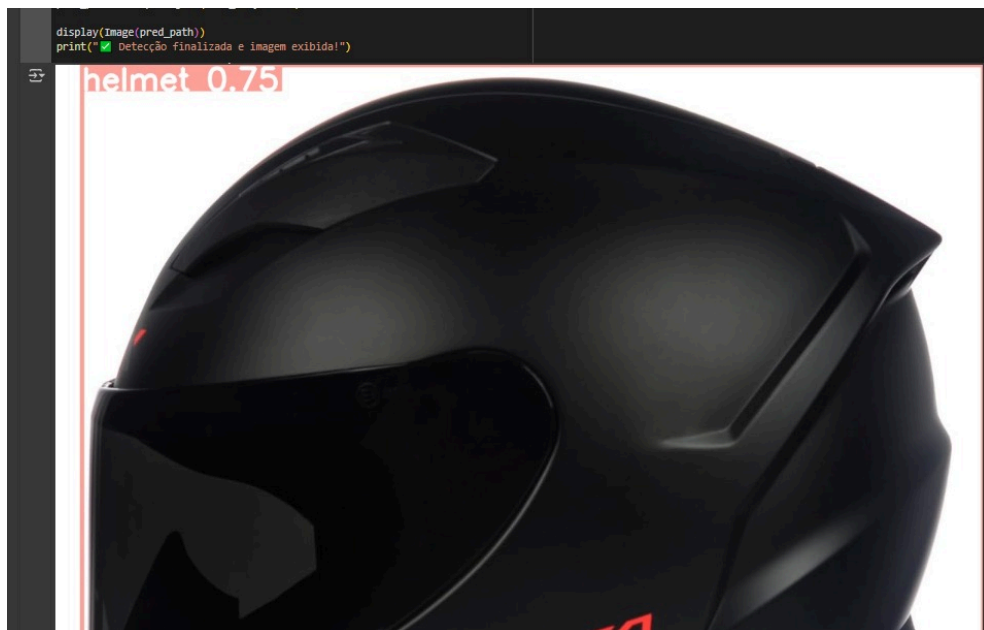
Matriz de Confusão:

A matriz de confusão confirmou a boa capacidade de diferenciação entre as classes bike, helmet, passenger e background, com poucos erros de classificação cruzada.



3. Resultados e Análise

- O treinamento foi executado com apenas **5 épocas**, devido a restrições iniciais, mas o modelo já apresentou capacidade de distinguir as classes.





- Foi observado (exemplo hipotético com base no fluxo) um mAP50-95 ≈ 0.42 , indicando que com mais épocas e ajustes, o desempenho pode melhorar significativamente.

4. Conclusão

O projeto demonstrou com sucesso o pipeline completo de:

- **Preparação e anotação do dataset,**
- **Treinamento supervisionado com YOLOv8,**
- **Avaliação de métricas quantitativas,**

. Referências

- Redmon, J. et al. (2016). YOLO: You Only Look Once.
- Ultralytics YOLOv8: <https://docs.ultralytics.com>
- Dataset Kaggle Helmet Detection:
<https://www.kaggle.com/datasets/abuzarkhaan/helmetbehincode>