Serviço Público Federal
Ministério da Educação
Fundação Universidade Federal de Mato Grosso do Sul

CIDADE UNIVERSITÁRIA
FACULDADE DE COMPUTAÇÃO

VITOR YUSKE WATANABE - 2020.1905.058-4
MATHEUS LOPO BORGES - 2017.1904.081-8

TRABALHO FINAL:
Detecção de capacetes de segurança

CAMPO GRANDE - MS
2023

Sumário

1 Objetivo

O objetivo deste trabalho foi treinar dois modelos para detecção de pessoas que estão usando capacetes de segurança e não estão usando. Em seguida, avaliar o desempenho dos modelos obtidos e testar eles em uma aplicação em tempo real.

2 Materiais e Métodos

A seção atual tem a finalidade de informar os equipamentos, ferramentas utilizadas e descrever o processo utilizado para obter os modelos desejados.

2.1 Materiais

O código usado para lidar com os dados pode ser acessado na pasta do trabalho em [5], com o nome **TPI-Trabalho.ipynb** e o código do aplicativo no repositório do GitHub [6]. Como a implementação do arquivo **TPI-Trabalho.ipynb** foi feita localmente, então não será possível executar o código via Google Colab, sendo necessários ajustes de dependências e diretórios para a execução devida do código que converte o dataset no formato COCO, inferência com uma imagem qualquer e impressão das imagens de teste, que estão nas pastas **faster_rcnn_test_images** e **rtmdet_tiny_test_images** do Google Drive.

O *Dataset* utilizado foi extraído do Kaggle [1]. Além disso, ele pode ser acessado na pasta do trabalho [5] na pasta **data**. Os resultados do processo de treinamento foram armazenadas na pasta **work_dirs** do diretorio do trabalho [5]. As imagens foram separadas em 15887 de treino, 2261 de teste e 4641 de validação.

O hardware utilizado foi:

- Notebook Dell Inc. Dell G15 5530
  - CPU: 13th Gen Intel® Core™ i5-13450HX × 16
  - RAM: 16 GB
  - GPU: RTX 3050
  - VRAM: 6 GB
  - Câmera: 1280 x 720 (HD), 30 fps

Quanto a parte de linguagem de programação e software utilizada foi:

- Python 3.11.6
- Torch 2.0.1
- mmdet 3.2.0
- mmcv 2.0.1
- CUDA compiler 11.8
- GCC 9.3
- cv2 4.7.0

2.2 Métodos

Como o dataset já estava pronto, com as pastas de treino, validação e teste, então foi necessário apenas organizar as anotações no formato COCO e treinar utilizando as bibliotecas do MMDetection [2].

Os modelos Faster R-CNN (R-50-FPN) [4] e RTMDet (tiny) [3] foram escolhidos para a realização do ajuste fino (*fine tuning*), a fim de condicionar eles para o problema desejado. O modelo Faster R-CNN foi escolhido por ser bom na tarefa de detecção de objetos e uma das suas propostas é ter um tempo de predição baixo, mas como é um modelo de 2017 o seu desempenho foi comparado com um modelo mais atual, como o RTMDet, que foi proposto em 2022 e sua proposta é ser utilizado em sistemas de predição em tempo real.

Além disso, foram utilizadas as configurações e algoritmos de treino padrão do MMDetection para definir o processo de treinamento dos modelos, apenas reduzindo a quantidade de épocas de treinamento e o tamanho do *batch*, para reduzir o tempo de treinamento e devido às limitações de hardware.

Para o treinamento do modelo Faster R-CNN, foram utilizadas os parâmetros:

- Épocas: 12
- *Optimizer*: *Stochastic Gradient Descent*
  - *Learning rate*: 0.002
  - *momentum*: 0.9
  - *weight decay*: 0.0001
- *Batch size*: 1 image

Para o treinamento do modelo RTMDet, foram utilizadas os parâmetros:

- Épocas: 150
- *Optimizer*: AdamW
  - *Learning rate*: 0.004
  - *weight decay*: 0.05
- *Batch size*: 10 images

Durante o processo de treinamento o modelo que possuía o maior valor de *mean Average Precision* (bbox_mAP) era salvo e escolhido como o melhor para ser usado na aplicação, ao invés de apenas utilizar o último modelo.

Após a obtenção dos modelos finais, eles foram utilizados em uma aplicação com a câmera no Notebook, com a finalidade de testar a qualidade das predições e a taxa de quadros por segundo (*fps*) obtida com os modelos em execução.

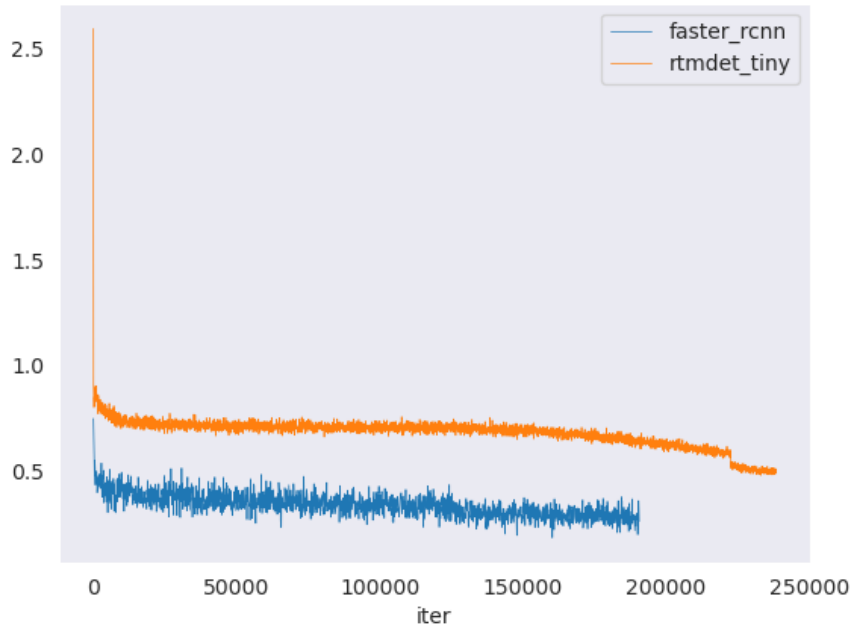
3 Resultados

Em seguida, foram discutidos os desempenhos dos modelos nos aspectos quantitativos e qualitativos, bem como os testes e experimentos realizados com eles.

3.1 Resultados Quantitativos

O processo de treinamento do modelos Faster R-CNN e RTMDet levaram 15 horas e 23 horas, aproximadamente. Avaliando o processo de treinamento dos dois modelos na Figura 1 foi possível observar que o modelo Faster R-CNN oscilava bastante, devido ao seu *batch size* pequeno com 1 imagem, em comparação ao RTMDet, que processava 10 imagens por *batch*, pois é um modelo mais leve, com menos parâmetros, permitindo que mais imagens pudessem ser armazenadas na VRAM. Além disso, o modelo Faster R-CNN apresentou loss de treino menor do que o RTMDet.

Figura 1: Gráfico de *loss* no treinamento dos modelos



Fonte: Autores

O modelo final obtido para o Faster R-CNN foi na época 9, enquanto para o RTMDet foi na última época, de número 150. Avaliando o desempenho durante a validação na Tabela 1 foi possível observar que ambos os modelos tiveram desempenhos similares, mostrando que para a RTMDet obter boa performance é necessário mais iterações do que o modelo Faster. Na seção 3.3 foi abordado o desempenho dos modelos na fase teste.

Tabela 1: Métricas de validação

	Faster	RTMDet
bbox_mAP	0.5640	0.5690
bbox_mAP_50	0.9440	0.9340
bbox_mAP_75	0.6050	0.6170
bbox_mAP_s	0.4830	0.4790
bbox_mAP_m	0.6420	0.6600
bbox_mAP_l:	0.6960	0.7120

Fonte: Autores

3.2 Resultados Qualitativos

Alguns exemplos de detecções realizadas no conjunto de teste estão presentes na Figura 2, utilizando um limiar de 0.5 no *threshold*, em que os retângulos envolventes azuis são os capacetes de segurança e vermelho as pessoas sem capacete. Analisando a Figura mencionada, foi possível observar que o modelo Faster R-CNN tinha maior facilidade na detecção de capacetes que estão próximos da câmera, mas confundiu pessoas com outras vestimentas, como o boné, na quarto exemplo da Figura. E o modelo RTMDet tinha maior dificuldade para detectar capacetes próximos da câmera, mas não confundiu o boné com o capacete no quarto exemplo. Ambos modelos conseguiram apresentar boa performance na detecção de pessoas de maneira geral. Além disso, foi possível observar que os modelos conseguiam reconhecer capacetes que estavam parcialmente na imagem, como no sexto exemplo da Figura 2.

Figura 2: Inferência nas imagens de teste (Faster e RTMDet, respectivamente)

Fonte: Autores

3.3 Experimentos

Na pasta **faster_rcnn_test_images** e **rtmdet_tiny_test_images** do Google Drive [5], estão as imagens usadas para testar os modelos, que foi uma das maneiras para verificar o desempenho do modelo em condições que eles não viram previamente.

Outra forma de verificar o desempenho do modelo, foi através de uma aplicação usando a câmera do notebook, em que o seu código fonte está presente no repositório [6]. A Figura 3 apresenta um quadro do teste do modelo no aplicativo em execução. Nesta versão foi utilizado o modelo RTMDet, por ser mais leve, mais rápido para realizar inferências. A principal vantagem de se utilizar ele foi que era possível realizar a inferência com a CPU em um tempo razoável, apresentando um apenas um pouco de atraso na inferência. No modelo Faster R-CNN não era viável utilizar a CPU para a inferência, devido ao *delay* elevado.

Figura 3: Exemplo de teste do aplicativo em execução

Fonte: Autores

4 Conclusão

Portanto, mesmo com restrições de hardware foi possível treinar os dois modelos e obter desempenho satisfatório. Ademais, foi possível utilizar os modelos treinados em uma aplicação com a câmera do notebook, com destaque para o modelo RTMDet (tiny) apresentou um bom tempo de resposta e boa acurácia.

Referências

[1] Alexander. Yolo helmet/head. Link: `https://www.kaggle.com/datasets/vodan37/yolo-helmethead`.

[2] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[3] C. Lyu, W. Zhang, H. Huang, Y. Zhou, Y. Wang, Y. Liu, S. Zhang, and K. Chen. Rtmdet: An empirical study of designing real-time object detectors, 2022.

[4] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Jun 2017.

[5] V. Y. Watanabe. Arquivos do trabalho final. Link: `https://drive.google.com/drive/folders/1OVyjaviRZ1_WVJ6eHqZ0fj3UQd87N7ug?usp=sharing`.

[6] V. Y. Watanabe. camera-helmet-detection. Link: `https://github.com/TuskiNinjaa/camera-helmet-detection.git`.