UFPA – FCT – 10 de fevereiro de 2025
Disciplina: Inteligência Computacional

Monitoramento de idosos através de câmeras e detecção de objetos usando redes da “família” Yolo

Pré-requisitos: não é mandatório, mas é útil se ter ideia básica do que é uma rede neural, camadas “densas” e “convolucionais”. A rede neural que usa camadas convolucionais é chamada CNN (“convolutional neural net”). Vídeos que ajudam nestes assuntos são:

1) v10-2020-12-14 at 06_23 GMT-8_Neural_Nets.mp4
2) (só caso haja tempo: v10b-chapter_4_geron-2022-10-11_11-20-17.mp4)
3) v11-2021-02-01 at 06_46 GMT-8_Deep_Nets.mp4
4) v12_object_detection_yolo_2022-06-13.mp4

Tarefa: O sistema de monitoramento deve ser pautado em: 1) software de uma das redes da família Yolo (versões da Yolo), ou uma outra rede que execute “detecção de objetos”, 2) modelo do estilo “foundation model” associado ao software de rede neural (NN) escolhido, 3) modelo “custom”, adaptado para a tarefa em questão usando “fine tuning”, 4) OpenCV em Python para processamento de vídeos e imagens e 5) módulo de pós-processamento em Python.

IMPORTANTE: Mesmo que o software/modelo de NN que a equipe escolha dê suporte a tarefas como “pose”, “tracking”, etc., é mandatório usar o modo “object detection”.

O objetivo do sistema é: dado um vídeo pré-gravado em MP4 ou oriundo de uma câmera, após uma dada duração (por exemplo 5 minutos), indicar o tempo em que a pessoa monitorada ficou de pé, deitada ou sentada, acompanhada da porcentagem de cada uma dessas durações em relação à duração total. A saída do sistema deve ser algo como:

Tempo em pé: 2 minutos (20% do tempo total)
Tempo em sentado: 4 minutos (40% do tempo total)
Tempo deitado: 2 minutos (20% do tempo total)
Tempo com usuário não detectado: 4 minutos (40% do tempo total)
Duração total: 10 minutos

As durações de tempo são obtidas através de pós-processamento em Python dos resultados de uma rede neural da família Yolo. As classes devem ser definidas anteriormente por cada equipe e informadas. O número delas pode ser escolhido pela equipe, mas deve ser justificado, não devendo haver classes que não são relacionadas ao problema em questão.

A equipe deve obrigatoriamente: fazer “fine tuning” de um “foundation model” a partir de um dataset próprio, obtido sem usar imagens ou vídeos já disponíveis na Internet. Esse dataset próprio deve ter no mínimo 30 exemplos de imagens para serem usadas em treino de modelos de NN. Essas imagens podem ser extraídas de vídeos, por exemplo, usando-se OpenCV. Mas a equipe deve almejar alguma diversidade das imagens. Por exemplo, um único vídeo pode gerar milhares de imagens (“frames”), mas a variação entre essas imagens pode ser baixa e comprometer a robustez do sistema. para complementar os existentes.

Para desenvolver o modelo, é útil haver imagens para validação (achar os hiperparâmetros) e teste (estimar capacidade de generalização do modelo). O número de imagens nesses conjuntos de validação e teste pode ser escolhido pela equipe. Sugere-se que a equipe use imagens relativamente distintas nos conjuntos de treino, validação e teste, caso efetivamente usem validação e teste (o que não é obrigatório, mas desejado).

Como o problema é de detecção de objeto, os datasets devem ter rótulos (“labeling”) adequados. Para rotular, as equipes podem usar o Roboflow ou qualquer outro site ou ferramenta. A metodologia e estratégia para tirar as fotografias (isso é opcional) pode ser a mesma adotada na referência [8]. O objetivo maior da competição é “acurácia” e, em segundo lugar, o custo computacional do sistema. No dia da apresentação em sala, serão feitas perguntas para todos os participantes da equipe. A nota será individual, e não a mesma nota para a equipe toda.

O que deve ser entregue / reportado:

- Fazer upload no Classroom de arquivo PDF com no máximo 5 slides descrevendo nome/matrícula de todos da equipe, o que foi feito e os principais resultados. Não é preciso incluir informação sobre conceitos básicos (exemplo: o que é uma CNN, detecção de objeto, etc.). Deve-se assumir que a plateia conhece o assunto. Deve constar neste conjunto de slides as instruções para instalar e executar o sistema, por exemplo: o URL com os pesos da rede e arquivos auxiliares para executar o sistema, incluindo o sistema de pós-processamento. Uma possibilidade é o código estar no github, mas não é obrigatório.

- A competição será em sala de aula, de forma presencial. Uma aula antes da competição será dedicada à instalação do software de cada equipe no laptop que será usado no dia da competição, a partir das instruções listadas no PDF citado. Neste dia da "instalação", antes de ir para sala de aula, os monitores já baixaram o que precisam e com ajuda da equipe, só vão instalar, configurar e testar o software da equipe no laptop da competição.

Versões da Rede Yolo

O paper original da Yolo é https://arxiv.org/abs/1506.02640. O primeiro autor, Joseph Redmon criou o site https://pjreddie.com/ e iniciou o framework "Darknet" em https://pjreddie.com/darknet/yolo/. Ele também fez um currículo artístico: https://pjreddie.com/static/Redmon%20Resume.pdf. Mais recentemente, ele mostrou preocupação com o impacto da tecnologia: https://syncedreview.com/2020/02/24/yolo-creator-says-he-stopped-cv-research-due-to-ethical-concerns/. Para ouvir o Joseph apresentando a Yolo, pode-se usar:
https://www.youtube.com/watch?v=NM6lrxy0bxs&ab_channel=ComputerVisionFoundationVideos
e para aprender sobre uma versão mais avançada da Yolo original:
https://www.youtube.com/watch?v=GBu2jofRJtk&ab_channel=DanielGordon

Atualmente, a Yolo está disponível em vários "frameworks" e versões. Alguns dos frameworks mais tradicionais são:

1) Darknet: https://github.com/AlexeyAB/darknet: Yolov2 até Yolov4, linguagens C e C#. Essa versão foi um fork feito pelo alemão AlexeyAB no código do Joseph Redmon.
2) Ultralytics: https://github.com/ultralytics/ultralytics - Yolov11
3) Yolo v7: https://viso.ai/deep-learning/yolov7-guide/
4) Comparações: https://blog.roboflow.com/yolov4-versus-yolov5/

Configurar ambientes é sempre trabalhoso. No caso da Yolo, para quem não tiver GPU, pode-se tentar:

1) rodar em CPU: https://afetulhak.medium.com/running-darknet-yolov4-from-my-laptop-only-with-cpu-78553a594406 .
2) rodar em cloud, tipo no Google Colab: https://colab.research.google.com/github/ultralytics/yolov5/blob/master/tutorial.ipynb

Referências:

[1] data augmentation for the image and also for its bounding boxes. See https://blog.paperspace.com/data-augmentation-for-bounding-boxes/

[2] Dicas interessantes: https://people.cs.pitt.edu/~kovashka/cs2770_sp19/hw3/index.html

[3] Boas referências:
http://slazebni.cs.illinois.edu/fall18/assignment3_part2.html

[4] Blog que busca distinguir Yolo, Yolov2 e Yolov3, e explica o conceito de "anchor" points:
https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[5] Yolo para detecção de carro. Notebook para tarefa no Coursera:
https://nbviewer.org/github/amanchadha/coursera-deep-learning-specialization/blob/master/C4%20-%20Convolutional%20Neural%20Networks/Week%203/Car%20detection%20for%20Autonomous%20Driving/Autonomous_driving_application_Car_detection.ipynb

[6] Roboflow – mesa redonda sobre Yolo: https://www.youtube.com/watch?v=Xgrg8_936pk

[7] Wan, F., Sun, C., He, H. *et al.* YOLO-LRDD: a lightweight method for road damage detection based on improved YOLOv5s. *EURASIP J. Adv. Signal Process.* **2022**, 98 (2022).
https://doi.org/10.1186/s13634-022-00931-x
https://asp-eurasipjournals.springeropen.com/articles/10.1186/s13634-022-00931-x

[8] Sonali Bhutad, Kailas Patil, Dataset of road surface images with seasons for machine learning applications, Data in Brief, Volume 42, 2022, ISSN 2352-3409,
https://doi.org/10.1016/j.dib.2022.108023.
https://www.sciencedirect.com/science/article/pii/S2352340922002347

[9] Implementing YOLOV1 from scratch using Keras Tensorflow 2.0 - O URL https://www.maskaravivek.com/post/yolov1/ é útil para entender a Yolo original

[10] https://pyimagesearch.com/2022/04/11/understanding-a-real-time-object-detection-network-you-only-look-once-yolov1/

[11] https://github.com/hizhangp/yolo_tensorflow

[12] Um dos muitos sites com cursos sobre OpenCV: https://opencv.org/university/mastering-opencv-with-python/ , inclusive "Object Detection & Tracking".

[13] Yolo GitHub: https://github.com/ultralytics/ultralytics

[14] Yolo doc: https://docs.ultralytics.com/fr/models/yolo11

[15] Roboflow: https://roboflow.com

Apêndices: Detalhes técnicos

Parcelas da “Loss Function” usadas por versões da rede Yolo

Discutamos como a loss function variou na evolução da Yolo e as parcelas que as compõem. A "loss function" da rede Yolo pode ser dividida em 3 parcelas: box, obj, cls, chamadas aqui: boxLoss, objLoss e clsLoss, respectivamente.
A loss total é dada pela soma das parcelas:

totalLoss = boxLoss + objLoss + clsLoss

Yolo v1 (original)

boxLoss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2$$

objLoss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

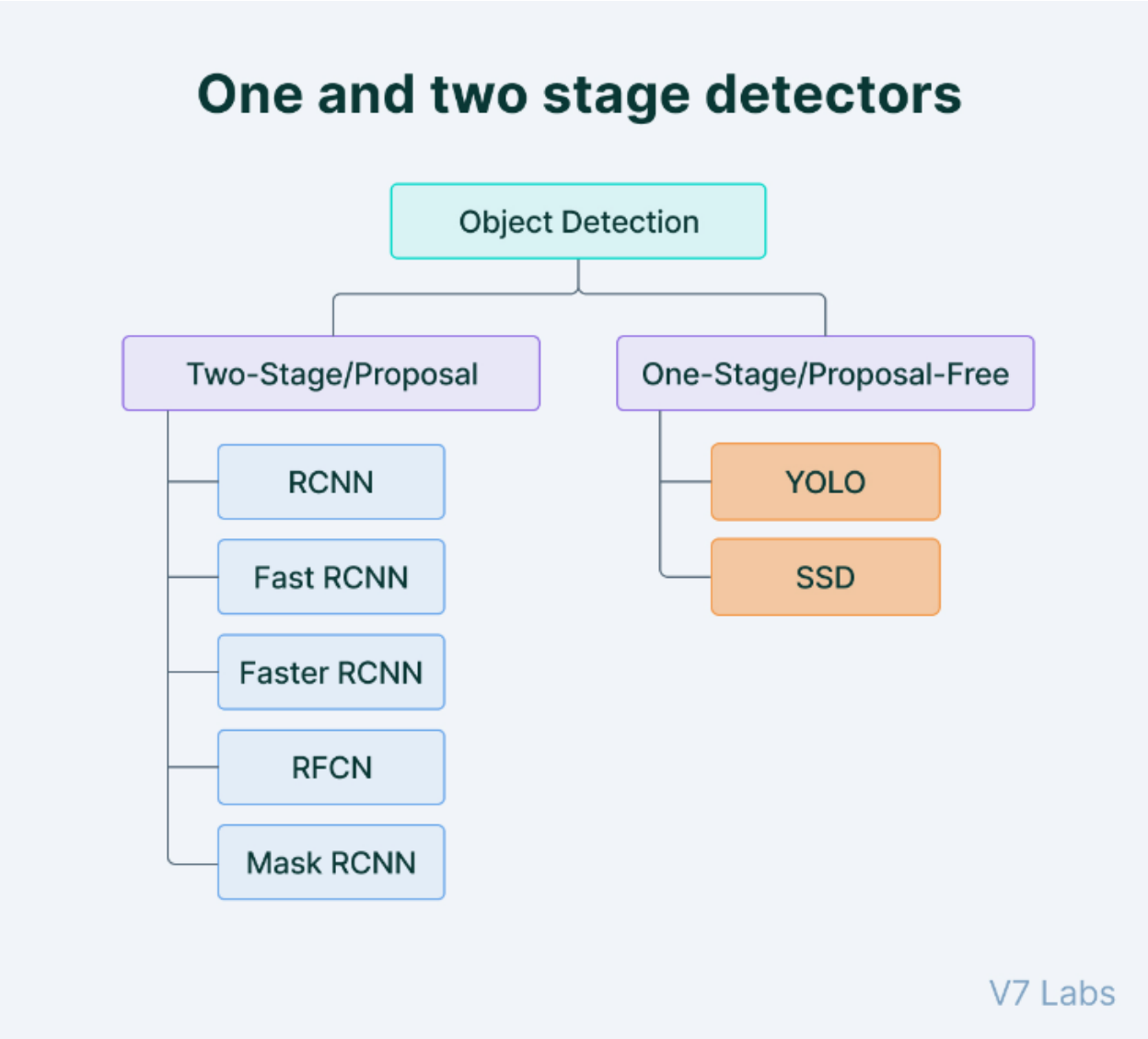
clsLoss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2 \quad (3)$$

No artigo original da Yolo v1, por simplicidade, usou-se o MSE (“mean squared error”) para as 3 parcelas. Na Yolo v2, a loss teve que ser alterada (vide https://fairyonice.github.io/Part_4_Object_Detection_with_Yolo_using_VOC_2012_data_loss.html) e as implementações mais modernas (como a da Mathworks - https://www.mathworks.com/help/vision/ug/object-detection-using-yolo-v3-deep-learning.html) usam binary cross-entropy (BCE) para a objLoss e clsLoss. Neste caso, cada parcela corresponde a:
- boxLoss: Calculates the mean squared error of the predicted bounding box coordinates with target boxes.

- objLoss: Determines the binary cross-entropy of the predicted object confidence score with target (or ground truth from the label) object confidence score
- clsLoss: Determines the binary cross-entropy of the predicted class of object with the target.

Mais informações sobre redes Yolo



Recall that Redmond, author of the original Yolo, calls “darknet” his networks from https://pjreddie.com/darknet/yolo/. For instance, Darknet-53 is the feature extractor with 53 layers used in Yolo v3. When used for image classification (not object detection) it provides:

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Table 2. **Comparison of backbones.** Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

Note also that Alexey Bochkovskiy forked Redmond's code and created Yolo v4 (https://github.com/AlexeyAB/darknet).

Yolo v1 (original)

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2 \qquad (3)$$

where $\mathbb{1}_i^{\text{obj}}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

Note that the loss function only penalizes classification error if an object is present in that grid cell (hence the conditional class probability discussed earlier). It also only penalizes bounding box coordinate error if that predictor is "responsible" for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell).

Observações:

a) Por mais que p_i(c) na última parcela sejam probabilidades, é adotado o MSE e não uma cross-entropy
b) Quando não há objeto (de acordo com as funções indicadores "1", que são baseadas nos rótulos - "labels"), a única parcela que compõe a loss é a baseada na "confidence" de que não há objeto (penúltima linha da loss).

Yolo v2 and Yolo9000

Observações:

a) Yolo v2 é customizada para reconhecer 9000 classes e essa versão customizada é chamada Yolo9000

b) Na terminologia da Yolo, prever “objectness”, é prever a “confidence” de haver um objeto no respectivo bounding box: “he objectness prediction still predicts the IOU of the ground truth and the proposed box and the class predictions predict the conditional probability of that class given that there is an object.” Na ref. [2] é informado: ““Objectness” measures membership to a set of object classes vs. background”.

c) O conceito de “anchor box”, também chamado de “prior box”, é adotado de forma modificada na Yolo v2, onde é chamado de “dimension cluster as anchor boxes”. O anchor box é explicado em [2] e blogs na Internet como [4]. E o dimension cluster é, em alto nível, equivalente a “ancorar” o bounding box no grid que divide a imagem.

d) Para não ter que escolher os anchor boxes manualmente, Yolo v2 executa K-means. Mas não usa a distância Euclidiana, e sim a IOU, como abaixo:

Euclidean distance larger boxes generate more error than smaller boxes. However, what we really want are priors that lead to good IOU scores, which is independent of the size of the box. Thus for our distance metric we use:

$$d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$$

Yolo v3

Yolo v4

$$\mathcal{L}(\hat{z}, z) = \mathcal{L}_{CIoU}$$
$$- \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} \left[\hat{C}_i \log\left(C_i\right) + \left(1 - \hat{C}_i\right) \log\left(1 - C_i\right) \right]$$
$$- \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{noobj}} \left[\hat{C}_i \log\left(C_i\right) + \left(1 - \hat{C}_i\right) \log\left(1 - C_i\right) \right]$$
$$- \sum_{i=0}^{S^2} I_{ij}^{obj} \sum_{c \in \text{ classes}} \left[\hat{p}_i(c) \log\left(p_i(c)\right) + \left(1 - \hat{p}_i(c)\right) \log\left(1 - p_i(c)\right) \right]$$

References (lista 2)

[1] - Boa explicação sobre dois tipos de loss em detecção de objetos
https://stackoverflow.com/questions/54977311/what-is-loss-cls-and-loss-bbox-and-why-are-they-always-zero-in-training
[2] Paper da Faster R-CNN em 2016, que propõe conceito de “anchor” boxes usada na Yolo v2
https://arxiv.org/pdf/1506.01497.pdf
[3] - 1 x 1 convolution e FCN (fully convolutional net) usado na R-CNN
https://ai.stackexchange.com/questions/21810/what-is-a-fully-convolution-network
[4] Anchor box - https://towardsdatascience.com/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9