Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Aprimoramento de um Método de Detecção de Pessoas Quanto a Acurácia e Custo Computacional

Pesquisa Científica

Victor Andrés Caram Guimarães
(2015083485)
William Robson Schwartz
Orientador

2° Semestre de 2019

Conteúdo

1 Resumo 2
2 Introdução 2
3 Referencial Teórico 2
4 Desenvolvimento do Trabalho 3
4.1 Detecção apenas de pessoas 3
4.2 Retreinamento da rede 3
4.3 Execução em um sistema embarcado 4
5 Resultados Obtidos 4
6 Conclusão 5

1 Resumo

A detecção de pessoas em vídeo ainda é um problema em aberto na literatura. Conciliar a detecção em tempo real com a acurácia é um dos principais desafios atualmente. Nesse trabalho, utilizamos resultados de estudos anteriores, que revelaram o algoritmo YOLO sendo aquele mais versátil e com o melhor custo benefício em acurácia e média de quados por segundo, e tentaremos aprimorar as suas detecções focando apenas em pessoas e reestruturando toda a sua rede para deixá-lo mais leve afim de executá-lo em um sistema embarcado.

2 Introdução

Diante do mesmo problema de reconhecer pessoas em vídeos com um custo computacional que seja aceitável para a aplicação desejada, ou seja, identificar padrões de movimentos, localizar e descrever como elas se encontram em tempo real. A partir dos resultados obtidos perante a primeira parte do Projeto Orientado em Computação, podemos definir aquele algoritmo que é o mais versátil e obteve bons resultados perante os testes realizados.

Recapitulando, primeiramente foram levantados os principais métodos de detecção da atualidade, assim como uma linha do tempo de sua evolução perante os estudos e anos. A partir da coleta desses algoritmos, foi preciso definir padrões para que pudessem ser testados nos mesmos ambientes para que não houvesse um certo viés nos resultados. A partir disso, métricas foram estipuladas e assim pode-se avaliar os algoritmos previamente coletados e montar uma tabela que compara a acurácia e a média de quadros por segundo de cada um desses métodos.

Com esse resultado, podemos escolher o algoritmo para que é um dos mais versáteis daqueles testados: obteve uma acurácia boa em relação ao tempo de execução. Esse algoritmo foi o **YOLO** (*You Only Look Once: Unified, Real-Time Object Detection*). E, a partir dessa escolha, o objetivo desse trabalho é fazer com que esse detector de objetos se torne cada vez mais específico um detector de pessoas e tão mais leve que o original que seja possível executá-lo em um sistema embarcado, visto que não possuem tantos recursos de processamento.

3 Referencial Teórico

Reconhecer pessoas em vídeos possui uma grande importância para a sociedade moderna: identificar anomalias no comportamento, questões de segurança e até mesmo para reconhecer algum caso de saúde grave (como por exemplo uma pessoa passando por algum acidente) que precisa de um atendimento médico imediato. São muita as funcionalidades possíveis, mas, ainda atualmente, existe a dificuldade de reconhecimento de pessoas em tempo real, além de outros empecilhos como a qualidade da câmera, qualidade do processamento em baixo custo, assim por diante.

Com o avanço da tecnologia e o barateamento de hardware, foi possível com a evolução do processamento de vídeo. Atrelado a isso, foram estudados uma diversidade de métodos de extração de padrões, o que possibilitou que os algoritmos se tornassem mais eficazes e precisos.

Diante dessa evolução e dos testes previamente realizados, o algoritmo que se destacou foi o "*You Only Look Once: Unified, Real-Time Object Detection*"[1]. Como agora o trabalho se trata de uma melhora desse algoritmo para uma determinada situação (no nosso caso, a detecção de pessoas), foram-se procurados alguns referenciais (tanto em formas científicas quanto em fóruns da internet) para saber como outras pessoas realizaram essa melhora. Encontrou-se as seguintes referências: "*How to Fine-tune*"[2], "*YOLO Fine-tuning*"[3], "*Training YOLO*"[4], além de outras que ainda serão analisadas conforme a progressão do trabalho.

A partir desses referenciais, é possível ter uma noção de como fazer a melhora do algoritmo proposto e fazer com que ele execute até mesmo em um sistema com poucos recursos computacionais. Além disso, é preciso também especificar os dados em que serão treinados e quais os resultados são esperados, visto que cada tipo de vídeo e de câmera afeta no objetivo final de detecção.

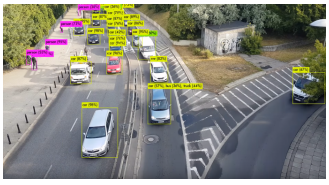
4 Desenvolvimento do Trabalho

O trabalho foi dividido em três grandes etapas, primeiro devemos fazer **YOLO** detectar apenas de pessoas. Depois, foi preciso retreinar a rede que nos foi disponibilizada previamente treinada e, após isso, finalmente executar e testar em um sistema embarcado. Portanto, a seguir falaremos mais de cada etapa do projeto.

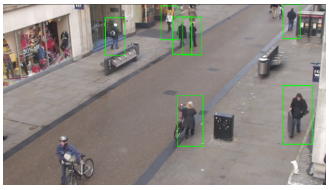
4.1 Detecção apenas de pessoas

O **YOLO** é um dos algoritmos detectores de objetos que são considerados estados da arte atualmente. Além de pessoas, há a detecção de carros, cadeiras, animais, diversas outras categorias de objetos. Como nosso objetivo é apenas detectar pessoas, é necessário transformar esse algoritmo em detector único sem perder a sua acurácia e custo computacional. Por isso, a abordagem de apenas filtrar as detecções, após elas serem feitas, não será vista nesse trabalho, visto que não traz nenhum benefício em questão de processamento e ele continuará detectando objetos, apenas não o mostrará.

Portanto, por alto, é preciso fazer a seguinte operação:



(a) Antes: YOLO como um detector de objetos.



(b) Depois: YOLO detecção apenas de pessoas.

Figura 1: Representação do antes e depois do tratamento.

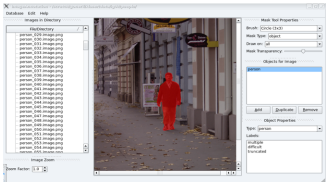
Para isso, foi preciso entender como o código do **YOLO** foi estruturado e o que poderia ser retirado. Foram utilizados diversos exemplos práticos de outras pessoas que fizeram o mesmo método, como: "*Yolo detecting only specific class*"[5], "*Yolo for Snowman*"[6], "*Retraining Yolo*"[7]. Nela, é visto como mudar os pesos das camadas de detecção, que são disponibilizadas no próprio site do **YOLO**.

4.2 Retreinamento da rede

Após a modificação no código fonte do algoritmo, perdemos o modelo pré-treinado que nos foi disponibilizado e, portanto, é preciso fazer um retreino da rede com as informações que se devem ser detectadas.

Para isso, foi pego diversos datasets disponíveis na literatura e foi feito um treinamento geral com pessoas de diversos ângulos e câmeras diferentes. Exemplos públicos como "*COCO Dataset*"[7], Caltech Pedestrian Detection, TownCentre dentre outros. Além disso, foi feita também uma anotação pessoalmente de imagens diferentes com câmeras voltadas para a vigilância, onde são instaladas em um ponto alto com uma vista geral de um certo local.

(a) Exemplos do COCO dataset.



(b) Software para a criação de anotações

Figura 2: Retreino da rede.

Com essas anotações, houve o retreino da rede para a detecção de pessoas utilizando os mesmos procedimentos que o **YOLO** realiza com objetos, afim de não perder a sua qualidade e acurácia.

4.3 Execução em um sistema embarcado

Por fim, após ter retreinado a rede e deixar o algoritmo como apenas um detector de pessoas, foi realizado a execução em dois sistemas embarcados. São eles:

(a) Raspberry Pi 3 Model B

(b) Raspberry Pi 4 Model B

Figura 3: Placas Raspberry Pi.

A terceira geração do Raspberry, lançada no ano de 2016, possui um CPU de 1.2GHz, Quad Core Broadcom BCM2837 64bit. A versão utilizada possuía apenas 1 GB de RAM então os resultados não vao ser dos melhores. Já a quarta geração do Raspberry utilizada, lançada no ano de 2019, possui um CPU de 1.5GHz Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC. Além disso, possui 4GB LPDDR4-3200 SDRAM, ou seja, terá um poder de processamento melhor e mais rápido do que o anterior. Ambos serão utilizados para a demonstração dos resultados e serão apenas denominados como "Raspberry Pi 3 Model B"e "Raspberry Pi 4 Model B"respectivamente.

5 Resultados Obtidos

Após toda a metodologia e os procedimentos realizados, além do treinamento da rede com diversos datasets da literatura juntos, foi feita uma separação de 70% de treino e 30% de teste para realizar os testes. Como se tratam de vídeos, foi preciso simular uma *stream* onde se é passado os vídeos. A ideia é simular uma câmera real que fosse passar os dados conforme ia capturando.

Tabela 1: Raspberry Pi 3 Model B

Método	Resolução	Média de FPS	mAP (%)	Average IoU (%)	Average Loss
Yolo V3	416x416	0,1	57,9	62,4	0,35
Yolo V3	608x608	0,06	61,2	70,3	0,30
Yolo V3	832x832	0,01	64,3	75,2	0,26
Tiny Yolo V3	416x416	1,6	55,4	58,4	0,41
Tiny Yolo V3	608x608	1,02	58,6	60,3	0,36
Tiny Yolo V3	832x832	0,84	60,1	62,1	0,31
Yolo Modificado	416x416	2,1	56,4	60,7	0,33
Yolo Modificado	608x608	1,2	60,7	66,9	0,28
Yolo Modificado	832x832	0,84	61,2	70,5	0,24

Tabela 2: Raspberry Pi 4 Model B

Método	Resolução	Média de FPS	mAP (%)	Average IoU (%)	Average Loss
Yolo V3	416x416	0,2	59,4	65,2	0,28
Yolo V3	608x608	0,1	63,4	69,3	0,24
Yolo V3	832x832	0,05	66,8	76,4	0,21
Tiny Yolo V3	416x416	9,21	57,6	60,4	0,39
Tiny Yolo V3	608x608	6,67	60,1	62,8	0,34
Tiny Yolo V3	832x832	4,2	62,4	66,3	0,30
Yolo Modificado	416x416	12,4	60,2	61,9	0,29
Yolo Modificado	608x608	9,35	65,3	67,8	0,24
Yolo Modificado	832x832	7,2	69,8	72,5	0,19

A partir dos resultados obtidos, foi possível ver uma diferença grande entre, primeiramente, o Yolo V3 e o Tiny Yolo V3. Esse último, é uma versão da arquitetura modificada para a execução em sistemas com menor processamento. Visto que o Yolo V3 é um algoritmo focado em execução de GPU, era previsto que suas métricas de Média de FPS fossem muito baixas, mas com uma acurácia melhor do que os outros.

Já comparando o Tiny Yolo V3 e a nossa versão do Yolo que detecta apenas pessoas, vemos uma melhora significativa em questões de acurácia e média de FPS. Em várias resoluções, tivemos o aumento dessas métricas. Embora não sejam os melhores resultados, se pensado que se trata de um sistema embarcado de baixo poder de processamento, tivemos bons resultados.

6 Conclusão

Como visto nos resultados mostrados nas tabelas, houve uma melhora na média de FPS e até mesmo na acurácia. Este ultimo, entretanto, provavelmente se deu por alguma modificação no CPU entre as duas versões do Raspberry. Porém, conclui-se que foi possível ter uma melhora com a modificação do YOLO e executá-lo em um sistema embarcado. Embora uma GPU seja extremamente importânte para a melhora geral do algoritmo, ainda sim foi possível obter bons resultados com um baixo custo, visto que normalmente sistemas embarcados não possuem muito poder de processamento, além disso outros sistemas estão sendo desenvolvidos, como é o caso do NVIDIA Nano que promete uma placa de vídeo integrada no sistema.

Com isso, foi possível realizar diversos experimentos e analisar as métricas definidas satisfatoriamente, aprender como se comporta um algoritmo de detecção de pessoas e como fazer para especificar o que estaria sendo detectado. Ademais, há a conclusão do meu Projeto Orientado em Computação, onde analisei diversos algoritmos considerados estado da arte da literatura, estipulei aquele que seria o mais versátil e com o melhor custo benefício e, por fim, o aprimorei para uma detecção específica ao ponto de executá-lo em um sistema embarcado.

Referências

[1] YOLO: https://pjreddie.com/media/files/papers/yolo.pdf

[2] How to Fine-tune: https://bit.ly/2lBtpVL

[3] YOLO Fine-tuning: https://bit.ly/2kB1Z21

[4] Training YOLO: http://guanghan.info/blog/en/my-works/train-yolo/

[5] Tutorial StackOverflow: https://stackoverflow.com/questions/44674517/yolo-darknet-detecting-only-specific-class-like-person-cat-dog-etc

[6] Yolo for Snowman: https://github.com/spmallick/learnopencv/tree/master/YOLOv3-Training-Snowman-Detector

[7] Retraining Yolo: https://github.com/YunYang1994/tensorflow-yolov3

[8] COCO Dataset: http://cocodataset.org/#home

[9] mAP: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173