Descarte de Quadros para Redução do Atraso na Detecção de Objetos em Vídeos

Hugo Antunes[1], **Rodrigo S. Couto**[1], **Pedro Cruz**[1]

[1]GTA/DEL-POLI/PEE-COPPE
Universidade Federal do Rio de Janeiro (UFRJ)

{hugo,rodrigo,cruz}@gta.ufrj.br

Resumo. *A detecção de objetos em tempo real é um desafio comum a diferentes aplicações, de carros autônomos à vigilância. Entretanto, o processamento de vídeos em tempo real exige um alto poder computacional, tornando comum a ocorrência de atrasos. Algumas dessas aplicações podem ser sensíveis a atrasos, tendo seu funcionamento prejudicado. Assim, este artigo propõe uma comparação de quadros sequenciais por meio da utilização dos valores RGB de cada pixel. Aqueles quadros que forem julgados semelhantes não serão enviados para processamento, o que diminui significativamente o tempo de processamento. Com os experimentos deste trabalho, pode-se observar uma redução no tempo de processamento de 41,5% com uma perda de precisão inferior a 13%.*

Abstract. *Real-time object detection is a common challenge in different contexts, such as in autonomous cars and surveillance cameras. However, the processing of videos in real-time needs a high computational power, and this requirement causes the occurrence of delays. Some of these applications may be sensitive to delays, causing their operation impaired. In this way, this paper proposes a comparison between sequential frames using the RGB values of each pixel. Those frames deemed similar will not be processed, which decrease the processing time. The experiments that were done throughout this work, lead us to the conclusion that there's a certain processing time reduction of 41,5% with the loss of precision inferior to 13%.*

1. Introdução

A detecção de objetos é um grande desafio de visão computacional, com aplicações em diversas áreas. Por exemplo, pode ser utilizada em cidades inteligentes, coletando dados sobre a mobilidade urbana [Cristiani et al. 2018], em aplicações de Internet das Coisas, na forma de câmeras inteligentes [Nguyen et al. 2021], ou até mesmo em problemas de segurança pública [Narejo et al. 2021]. Uma das técnicas mais frequentes para detecção de objetos envolve a combinação de aprendizado profundo (*Deep Learning*) e Redes Neurais Convolucionais (*Convolutional Neural Networks* – CNNs). Essa técnica é empregada por ferramentas como o Mask R-CNN [He et al. 2017], o Faster R-CNN [Girshick 2015] e o YOLO [Redmon et al. 2016]. Essas ferramentas recebem imagens e marcam essas imagens com contornos retangulares na forma de caixas delimitadoras (*bounding boxes*) de objetos de interesse.

A detecção de objetos em vídeos pode ser tratada como a detecção de objetos em imagens estáticas para cada um dos quadros de um vídeo. Porém, a detecção de

objetos em uma imagem é um procedimento computacionalmente intensivo. Portanto, é possível que o tempo de processamento de um único quadro demande mais tempo do que o tempo representado pelo quadro. Essa situação pode criar um problema para a detecção de objetos em tempo real [Ćorović et al. 2018].

Neste trabalho, o tempo de detecção de objetos em um vídeo é reduzido por meio da escolha de quadros para não serem processados pelo algoritmo de detecção. Esses quadros são ditos *desviados*. O desvio de quadros pode ser prejudicial ao desempenho da detecção de objetos, uma vez que objetos em um quadro desviado não podem ser recuperados pelo algoritmo de detecção. Assim, para reduzir a perda de desempenho de detecção, são utilizadas duas heurísticas. A primeira é que, no caso de vídeos, os quadros sequenciais possuem um contexto [Shang et al. 2017] e, em diversas situações, os quadros em sequência são muito semelhantes. A segunda é de que quadros similares devem possuir os mesmos objetos. Dessa forma, na proposta deste trabalho, é estabelecida uma métrica de similaridade entre os quadros de um vídeo. Também é estabelecido um limiar, que considera que dois quadros são similares quando a similaridade entre eles atinge tal limiar. Os quadros são comparados ao último quadro que já foi processado. Caso o quadro atual seja considerado similar ao último quadro processado, o quadro atual é desviado e assume-se que os objetos encontrados no quadro já processado continuam enquadrados. São realizados experimentos com 500 vídeos do conjunto de dados ImageNet [Shang et al. 2017], comparando o fluxo proposto com o fluxo tradicional de processamento de imagens. Como resultado, os limiares de 70% e 90% tiveram bom desempenho no equilíbrio do compromisso estabelecido entre a precisão da detecção e o tempo de processamento. No limiar de 50%, a redução no tempo de processamento foi de 38,6% quando comparado ao tempo requerido pelo fluxo tradicional, mas com uma perda na precisão inferior a 13%. Já no limiar de 70%, a redução do desempenho atinge números próximos a 3%, tendo um ganho no tempo de processamento de quase 10%.

A literatura científica possui alguns trabalhos que propõem reduzir o atraso na identificação de objetos de interesse em vídeos. Esses trabalhos seguem estratégias que buscam melhorar a velocidade da detecção de objetos [Shafiee et al. 2017a, Huang et al. 2018, Fang et al. 2019, Lu et al. 2020, Liang et al. 2022] ou a eliminação de quadros a serem processados, sem considerar o desempenho da detecção de objetos [Lee and Hwang 2022]. Não foi encontrado na literatura científica um trabalho que utilize o descarte de quadros levando em consideração uma métrica de similaridade com o último quadro processado, procurando manter o desempenho de detecção de objetos.

A estrutura deste artigo está organizada como se segue. Na Seção 2, são abordados outros estudos já realizados sobre detecção de objetos e seu atraso de processamento. Posteriormente, na Seção 3, discute-se o problema do atraso e apresenta-se o modelo utilizado. Já na Seção 4, é explicada a implementação da comparação dos quadros e sua aplicação. Em seguida, na Seção 5, é apresentado um experimento para validar a proposta, com o detalhamento da metodologia experimental e, em sequência, seus resultados na Seção 6. Por último, são desenvolvidas as conclusões na Seção 7.

2. Trabalhos Relacionados

A detecção de objetos em tempo real é útil para diversas aplicações. Os principais trabalhos de detecção de objetos em imagens utilizam CNNs que são treinadas para

reconhecer objetos de interesse. É o caso do Mask R-CNN [He et al. 2017], do Faster R-CNN [Girshick 2015] e do YOLO [Redmon et al. 2016]. O presente trabalho utiliza especificamente o YOLO, devido à sua popularidade ao grande número de aplicações que o utilizam. Para citar alguns exemplos, têm-se aplicações para o trânsito inteligente, como a detecção de pedestres [Lan et al. 2018] e de veículos [Zuraimi and Zaman 2021] ou aplicações em segurança pública, como a identificação de armas [Ashraf et al. 2022, Narejo et al. 2021] e de ações relevantes [Shinde et al. 2018]. No momento de escrita deste artigo, o YOLO se encontra na versão YOLOv8 [Jocher et al. 2023].

No caso do processamento de vídeo em tempo real, estudos procuram soluções com uma grande gama de metodologias e estratégias para reduzir o tempo entre a captura de uma imagem e a detecção de objetos de interesse. Esses diferentes trabalhos podem ser separados em duas categorias. A primeira busca reduzir o tempo de execução do algoritmo de detecção e a segunda procura modificar a maneira com a qual o fluxo de vídeo é enviado para o algoritmo de detecção.

Na primeira categoria, é possível encontrar dezenas de trabalhos desse tipo, como o artigo [Huang et al. 2018], que propõe o YOLO-LITE, um algoritmo otimizado para detecção de objetos em tempo real para máquinas sem unidade de processamento gráfico (GPU). O algoritmo proposto se baseia na ideia de eliminar a normalização do lote, uma vez que a normalização aumenta consideravelmente o tempo de processamento da rede e não é tão eficiente em redes superficiais. Uma rede é dita superficial quando não possui muitas camadas, uma das características do YOLO-LITE. O YOLO-NAS [Aharon et al. 2021], criado pela equipe SuperGradients, otimizou o YOLOv8 considerando não apenas os compiladores, mas também a quantização. O NAS utiliza blocos sensíveis à quantização e quantização seletiva para melhorar o tempo de execução, sem redução da precisão do modelo. Já [Shafiee et al. 2017a] propõem o Fast YOLO. Ele reduz o número de parâmetros utilizados pelo YOLOv2 aplicando o conceito de inteligência evolutiva profunda [Shafiee et al. 2017b], o que torna o processamento do YOLOv2. Frankle e Carbin aprimoram a redução de parâmetros de uma rede neural buscando uma sub-rede que, quando treinada, obtém resultados similares à rede completa, num conceito denominado de bilhete de loteria (lottery ticket) [Frankle and Carbin 2019].

Os trabalhos Edge YOLO [Liang et al. 2022], [Pacheco and Couto 2021], YOLO-compact [Lu et al. 2020] e Tinier-YOLO [Fang et al. 2019] também pertencem à categoria dos trabalhos que buscam otimizar o tempo de processamento dos algoritmos de detecção. Os quatro procuram reduzir o tamanho da rede neural para diminuir o tempo de processamento dos quadros, uma vez que essa redução faz com que o caminho dos dados obtidos da imagem seja menor. O primeiro deles propõe o Edge YOLO, que realiza detecção de objetos em carros autônomos. O Edge YOLO realiza o processamento dos dados, que são enviados pelo automóvel, na nuvem. Além disso, a redução do número de camadas convolucionais presentes no YOLO é consolidada pela fusão de estruturas internas, de modo que diferentes partes da rede neural, como a cabeça e a espinha dorsal, tiveram suas estruturas integradas. Nessa mesma perspectiva de processamento na nuvem, Pacheco e Couto [Pacheco and Couto 2021] propõem um particionamento da rede neural que, caracterizada por saídas antecipadas, se destina à detecção de objetos. Essa divisão é realizada entre a borda e a nuvem, ou seja, parte da rede será processada na borda até uma certa camada, e o restante será responsabilidade da nuvem. A escolha da

camada convolucional que dividirá onde será o processamento, na borda ou na nuvem, engloba um problema de otimização. Sendo assim, esse trabalho determina uma camada que minimize o tempo de inferência da imagem. O YOLO-compact redesenha o YOLO para detectar apenas uma única classe por vez. Tal situação reduz a rede pela redução do número de camadas convolucionais. Assim, o tempo de processamento também se torna menor. O último trabalho, Tinier-YOLO, utiliza o já consolidado Tiny-YOLO, realizando a redução do número de parâmetros utilizados pelo modelo original. No Tinier-YOLO, camadas convolucionais tradicionais são substituídas por outras designadas como *fire modules*. Os *fire modules* possuem conexões mais densas do que as camadas tradicionais, o que viabiliza um desempenho melhor na detecção de objetos.

Na segunda categoria, que busca reduzir o número de quadros enviados ao algoritmo de detecção de objetos, é possível encontrar o trabalho de Lee e Huang [Lee and Hwang 2022]. O trabalho propõe um controle adaptativo dos quadros que são enviados para a rede neural. O trabalho leva em consideração que o YOLO possui uma fila para o processamento de quadros de um vídeo. Assim, o algoritmo monitora o tamanho da fila e, caso seu tamanho ultrapasse um limiar, os quadros mais antigos são descartados para dar lugar aos quadros mais recentes. Assim, o descarte é realizado de maneira adaptativa conforme a velocidade de processamento da rede neural, que varia consideravelmente de quadro para quadro. O trabalho de Lee e Huang não considera a possível redução do desempenho de detecção com o descarte dos quadros.

A maior parte dos trabalhos, para resolver o problema do atraso, desenvolvem diferentes versões da rede neural utilizada. Nesse aspecto, há uma carência de pesquisas na literatura que busca solucionar o atraso sem necessariamente alterar a rede neural da detecção. Assim, o presente trabalho procura reduzir o atraso da detecção de objetos em tempo real conservando o desempenho da detecção. Adicionalmente, os resultados deste trabalho podem ser aplicado em conjunto com técnicas que otimizam a rede neural.

Uma observação importante é a de que existem poucos conjuntos de dados rotulados para vídeos. A maior parte dos bancos de dados de imagens referem-se exclusivamente a imagens estáticas. O principal conjunto de dados de vídeos, o ImageNet-VID [Deng et al. 2009], encontra-se fora do ar. Dessa forma, os vídeos utilizados neste trabalho foram obtidos de um outro trabalho que utiliza o ImageNet [Shang et al. 2017], mas que trabalha com a contextualização dos quadros dos vídeos e, portanto, não contém a marcação das classes em cada quadro. Apesar disso, conjuntos de dados de vídeos são de extrema importância para a viabilização de pesquisas futuras.

3. Atrasos na Detecção de Objetos de Interesse em Vídeos

Um vídeo pode ser definido como um fluxo de quadros no tempo. A detecção em tempo real de objetos requer o processamento das imagens dos quadros. Para as aplicações de detecção de objetos em vídeos, o atraso é o tempo decorrido desde que uma imagem é capturada até o instante no qual um usuário recebe como resultado quais objetos de interesse estão presentes na imagem. A Figura 1[1] ilustra a sequência de processamento considerada neste trabalho. No caso geral, os algoritmos de detecção de objetos em imagens, como o YOLOv8 [Jocher et al. 2023], recebem um vídeo quadro por quadro

[1]Figura com ícones de Freepik, de Flaticon.com.

e utilizam uma fila para acumular os quadros recebidos. Então, os quadros são processados e uma rede neural profunda é utilizada para identificar os objetos em cada quadro. Um módulo realiza a marcação dos objetos utilizando caixas delimitadoras. Ao final, o vídeo é disponibilizado para usuários finais.

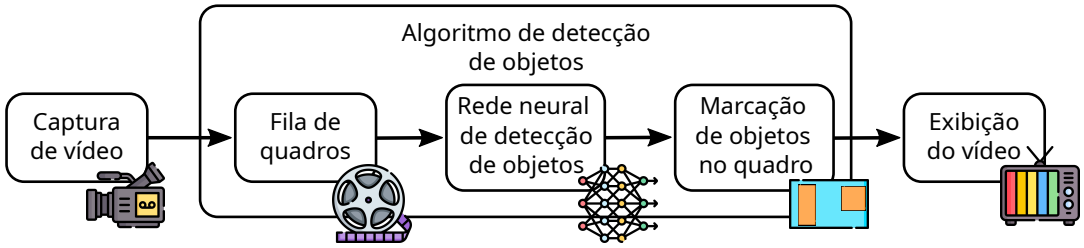

Figura 1. Sequência de processamento considerada neste trabalho.

A taxa de quadros por segundo (*Frames per second* - FPS) exigida pelas aplicações é definida com base na velocidade dos objetos a serem detectados, além do posicionamento da captura de vídeo com relação aos objetos. Por exemplo, aplicações de contagem de pedestres podem necessitar de taxas de quadros de 2 ou 3 FPS, enquanto carros autônomos podem necessitar de taxas de 10 FPS ou até maiores [Lee and Hwang 2022]. As câmeras comerciais normalmente possuem taxas de captura constantes, o que significa que o tempo de captura é igual para cada quadro. Por outro lado, o tempo de processamento para detecção de objetos difere para cada quadro. No caso geral, o tempo de captura de um quadro é muito menor do que o tempo necessário para encontrar objetos de interesse no quadro, fazendo com que a fila interna de algoritmos de processamento de imagens possa crescer indefinidamente [Lee and Hwang 2022].

O atraso é um problema comum no processamento de vídeos e se refere ao tempo decorrido entre a ocorrência de um evento cujas imagens foram capturadas e a sua exibição. Vários fatores podem aumentar o tempo entre a captura de uma imagem e sua exibição, como o envio pela rede, a codificação e a decodificação. No caso de aplicações com detecção de objetos em tempo real, o atraso é o intervalo de tempo entre o instante no qual um objeto entra no campo de visão da câmera que faz a captura de imagens e o instante no qual o usuário (ou usuária) recebe o resultado de quais objetos estão enquadrados pela câmera. Neste trabalho, é considerado apenas o atraso causado pelo processamento para a identificação de objetos.

Para estabelecer um modelo geral para o atraso inserido pela identificação de objetos, assume-se que as câmeras possuem taxa tx_{cam} constante de captura de quadros. Assim sendo, o intervalo entre quadros é de $(tx_{cam})^{-1}$. Também se assume que esses quadros são recebidos pelo algoritmo de identificação de objetos com a mesma taxa e com atraso de comunicação desprezível. Os quadros recebidos pelo algoritmo de detecção são colocados em uma fila e processados por ordem de chegada. Ao processar um quadro q_i, o algoritmo de detecção de objetos utiliza o quadro como entrada para uma rede neural profunda, que demora $tp(q_i)$ para processar q_i. Assume-se que o resultado é entregue ao usuário com atraso desprezível.

Quando a fila está vazia, o atraso $d(q_i)$ de um quadro q_i é apenas $tp(q_i)$. Porém, quando a fila possui quadros para serem processados, o atraso $d(q_i)$ se dá pela expressão:

$$d(q_i) = \sum_{n=j}^{i} (tx_{cam})^{-1} - tp(q_n), \tag{1}$$

com q_j sendo o quadro mais antigo da fila.

A Figura 2 ilustra a situação na qual $tp(q_i) > tx_{cam}^{-1}$. Tal diferença gera um aumento da fila e um atraso crescente entre os objetos em cena e o resultado da detecção de tais objetos. Neste trabalho, o atraso é reduzido com a redução do tempo de processamento de alguns quadros, evitando que eles sejam processados pelo algoritmo de detecção. Assim, os quadros não são enfileirados, o que reduz o atraso.

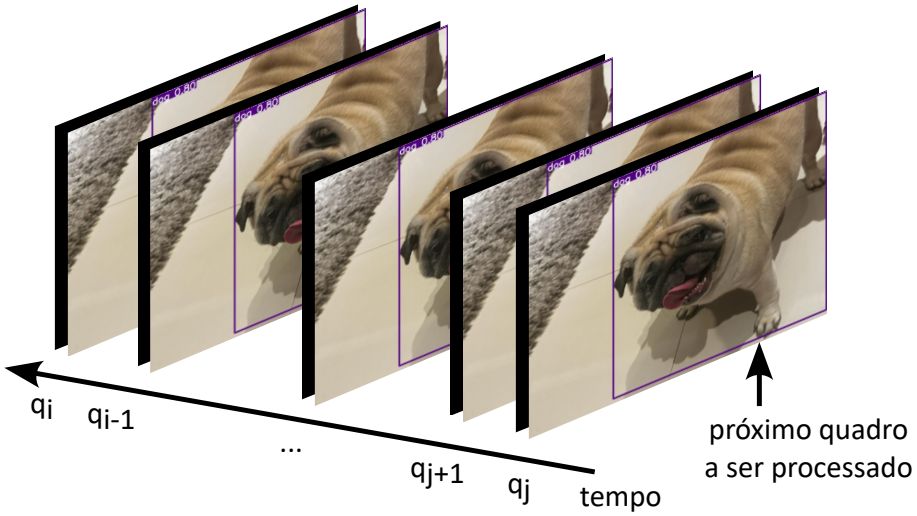

Figura 2. Gráfico esquemático do atraso de vídeos

4. Desvio de Quadros por Similaridade

Os atrasos decorrentes da detecção de objetos em vídeos podem ser um problema para as aplicações que necessitam dessa detecção. A Seção 3 discute o modelo de atrasos considerado neste trabalho. Para reduzir os atrasos entre a captura do vídeo e a entrega do resultado da detecção de objetos, este trabalho se baseia na ideia de que processar quadros semelhantes significa essencialmente extrair informações redundantes. Ou seja, quando duas imagens muito similares são enviadas à rede neural, as saídas de caixas delimitadoras de cada imagem serão também muito próximas, ao custo do processamento de dois quadros. Para decidir se um quadro é semelhante a outro, este trabalho utiliza uma função de similaridade e um limiar. A função de similaridade é detalhada a seguir.

4.1. Função de similaridade

Um quadro pode ser interpretado como uma matriz na qual cada elemento é um pixel. Cada pixel é representado por uma tupla de três elementos r, g e b (RGB), contendo as intensidades das cores vermelho, verde e azul daquele pixel, respectivamente. A medida de similaridade consiste em uma contagem de quantas vezes um elemento r, g, ou b do quadro q_{ref} está suficientemente próximo do elemento respectivo na mesma posição do quadro q_i. O Algoritmo 1, entre as linhas 1 e 9 detalha a função de similaridade adotada neste trabalho.

Algoritmo 1 Função de Similaridade entre Quadros

```
1:  função COMPARAR(q_ref, q_i, l, tol)
2:      x ← q_ref
3:      similaridade ← 0
4:      para x em [0..tamanho(q_ref)] faça
5:          para y em [0..tamanho(q_ref)[x]] faça
6:              delta_pixel ← |q_ref[x][y] − q_i[x][y]|
7:              para cor em delta_pixel faça
8:                  se cor ≤ tol então
9:                      similaridade ← similaridade + 1
10:     se l ≤ similaridade então
11:         Retorne True
12:     senão
13:         Retorne False
```

O Algoritmo 1 recebe o quadro de referência q_{ref}, o quadro atual q_i, um limiar l e uma tolerância tol. As linhas 4 e 5 iteram sobre cada pixel de ambos os quadros, de forma que as funções $tamanho(q_{ref})$ e $tamanho(q_{ref})[x]$ retornam o número de colunas da matriz e o número de elementos em uma coluna, respectivamente. A linha 6 encontra a diferença entre o pixel x, y de q_i pelo pixel x, y de q_{ref} e armazena na variável $delta_{pixel}$. Como um pixel é formado por três inteiros representando as cores, o laço na linha 7 incrementa a variável $similaridade$ para todo valor de $delta_{pixel}$ maior do que a tolerância tol. Dessa maneira, o valor máximo que a $similaridade$ pode alcançar é o número total de pixels multiplicado por 3. A variável l representa a similaridade mínima que dois quadros precisam alcançar para serem julgados como semelhantes. Quando $similaridade \geq l$, os quadros são considerados similares e, caso contrário, como diferentes. A Seção 6 analisa experimentalmente diferentes valores para l. Neste trabalho é utilizado um valor de 10 para tol, uma vez que os valores de cor podem variar de 0 a 255.

4.2. Desvio de Quadros Baseado em Similaridade

A proposta deste trabalho, ilustrada na Figura 3[1], recebe o quadro q_i de um vídeo e o compara ao último quadro já processado, denominado q_{ref}. Caso q_i seja considerado semelhante a q_{ref}, então q_i é marcado com as mesmas caixas delimitadoras obtidas no processamento de q_{ref} e enviado para a saída. Caso q_i não seja considerado semelhante o suficiente a q_{ref}, ele é enviado para o processamento pela rede neural e torna-se o novo q_{ref}. A semelhança é obtida a partir da função detalhada na Seção 4.1.

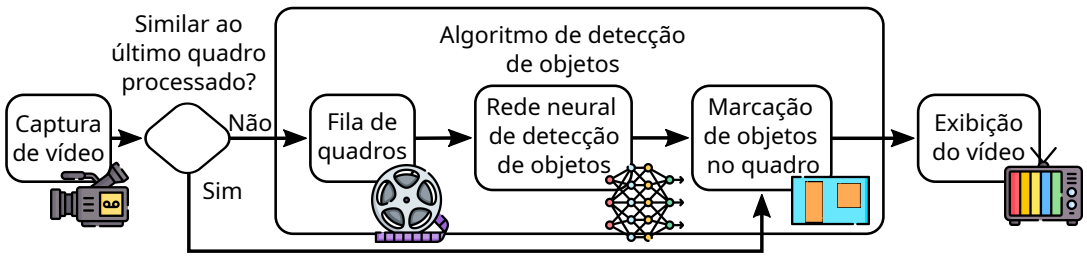

Figura 3. Sequência de processamento considerada neste trabalho.

A análise de um quadro só pode acontecer depois da exibição do quadro anterior a ele, ou seja, a fila do YOLO só pode se comportar de duas formas. A primeira delas é

quando q_i for similar ao q_{ref} e, portanto, a fila estará vazia. Já a segunda, é quando o q_i for considerado diferente, então a fila estará apenas com o quadro q_i. Dessa maneira, a função do atraso expressa na Equação 1 tende a ser reduzida, uma vez que a demora $tp(q_i)$ não acontecerá em todos os quadros, por conta do desvio. Sendo assim, aqueles que seriam enfileirados são mais rapidamente processados com a execução do Algoritmo 1.

5. Metodologia Experimental

Os experimentos implementam o fluxo de processamento exibido na Figura 1 e o fluxo de processamento exibido na Figura 3. O objetivo é comparar o cenário tradicional com o cenário proposto, analisando o compromisso entre precisão de detecção e tempo de processamento. Para a implementação do código[2] da comparação, são utilizadas as bibliotecas OpenCV [Itseez 2015], para capturar os quadros do vídeo, e Numpy [Van Der Walt et al. 2011], para a realização dos cálculos de maneira mais eficiente. O OpenCV captura cada quadro de um vídeo como uma matriz no formato NumPy. Ela representa os pixels do quadro com os seus respectivos valores RGB. A ferramenta de detecção de objetos utilizada no experimento é o YOLOv8, treinado com os dados do COCO [Lin et al. 2014], disponibilizada pela própria equipe do YOLOv8 [Jocher et al. 2023]. Além disso, a máquina utilizada nos experimentos é equipada com uma CPU Intel(R) Core(TM) i7-10700T CPU @ 2.00GHz, 16 GB de memória RAM e sistema operacional Ubuntu 22.04.2. A fim de reproduzir um cenário com escassez de recursos computacionais, no qual o atraso pode ser proibitivo para aplicações em tempo real, a máquina empregada nos experimentos não possui GPU.

O conjunto escolhido para os experimentos é o ImageNet VID 2016 [Deng et al. 2009]. O repositório oficial do ImageNet VID está fora do ar no momento da escrita deste trabalho[3]. Assim, é utilizado o ImageNet VID disponibilizado por Shang *et al.* [Shang et al. 2017][4]. Além disso, o ImageNet VID faz a separação do conjunto de vídeos em duas partes. Para os experimentos deste trabalho, é aproveitado a primeira parte do conjunto de dados em questão, o que contabiliza um total de 500 vídeos, com 127150 quadros e mais de 1 hora e 20 minutos de vídeo. Além disso, o limiar utilizado nos experimentos é ajustado de maneira proporcional ao total de pixels em cada vídeo, uma vez que os vídeos têm resoluções distintas.

Nos experimentos, as caixas delimitadoras obtidas pelo YOLO no fluxo tradicional são consideradas corretas (*Ground-Truth*). Assim, a precisão média (*mean Average Precision* – mAP) é obtida para o fluxo de processamento proposto, para diferentes limiares. O mAP é obtido comparando as classes detectadas pela ferramenta de detecção e o *Ground-Truth*, essa comparação é realizada por meio da Interseção de Uniões (*Intersection over Union – IoU*). A IoU é obtida por uma proporção das áreas das caixas delimitadoras do *Ground-Truth* e da ferramenta de detecção utilizada. Quando o IoU atinge um certo limiar, a classe detectada pela ferramenta é considerada uma marcação verdadeira e, caso contrário, como falsa. Por meio da contabilização das marcações verdadeiras e falsas, é realizado o cálculo da precisão média [Padilla et al. 2020]. O mAP é amplamente utilizado na literatura a respeito da precisão de modelos para a detecção de

[2]Os códigos deste trabalho podem ser encontrados em https://github.com/GTA-UFRJ/BetterYolo.

[3]https://image-net.org/challenges/LSVRC/2016/

[4]https://xdshang.github.io/docs/imagenet-vidvrd.html

objetos e, quanto maior o seu valor, melhor é considerado o modelo de detecção. Uma vez que o mAP é obtido tendo as marcações do YOLO como corretas, este será definido como mAP relativo. Ademais, este trabalho utiliza 50% como limiar da Interseção de Uniões para o mAP, visto que é um valor utilizado extensivamente na literatura. Assim, considera-se que se a interseção entre duas caixas delimitadoras for 50% ou maior, existe um verdadeiro positivo. Também são obtidas medidas para o tempo de processamento de cada quadro e o tempo de processamento de cada vídeo.

6. Resultados

A Tabela 1 apresenta os resultados dos experimentos realizados neste trabalho. Na primeira coluna, tem-se os limiares utilizados e, na última linha, o funcionamento com o YOLO, sem desvio de quadros. Com exceção da última, cada linha faz referência às características obtidas com o processamento com desvio pelo limiar indicado. Na segunda coluna é exibido o mAP relativo, como descrito na Seção 5. Já na terceira coluna, há a quantidade de quadros desviados em relação aos quadros totais, enquanto na quarta coluna há a quantidade de quadros que foram processados pelo YOLO. É esperado que os valores das terceira e quarta colunas de uma mesma linha somem 100%, uma vez que os quadros ou são desviados pelo algoritmo de comparação ou são processados pelo YOLO. A quinta coluna exibe o tempo necessário, em média, para processar cada vídeo do conjunto de dados e, finalmente, a sexta coluna mostra a diferença relativa do tempo de processamento por vídeo, ou seja, o ganho em tempo de processamento em relação ao tempo do YOLO sem desvios. Seu valor é calculado subtraindo o tempo de processamento do YOLO com desvios de quadros do tempo de processamento do YOLO puro e depois normalizando pelo tempo do YOLO puro. Além disso, vale ressaltar que tanto o mAP quanto o tempo de processamento são apresentados com um intervalo de confiança de 95%.

Como pode ser visto na segunda coluna, o mAP relativo cresce conforme o limiar aumenta, até chegar aos 100%, quando é utilizado apenas o YOLO. Ademais, é importante destacar que o mAP em relação ao limiar de 0% não é 0%. Isso acontece porque o primeiro quadro de cada vídeo sempre é enviado para o processamento, uma vez que não há quadro anterior a ser comparado e, então, esse é obrigatoriamente processado. A comparação da segunda coluna com a sexta coluna exibe o compromisso entre precisão e tempo de processamento. O limiar de 50% apresenta bom desempenho, uma vez que têm um mAP relativo próximo de 90% e uma redução do processamento perto de 40%. Também pode-se observar que o limiar de 70% obtém uma perda de desempenho de detecção inferior a 3% e um ganho no tempo de processamento superior a 12%. Os resultados a seguir analisam de maneira mais detalhada o compromisso entre precisão de detecção e tempo de processamento.

Os vídeos utilizados possuem diferenças entre si, como o tempo de duração, cenário, resolução e contexto. Assim, para uma melhor análise da proposta deste artigo, apresentam-se boxplots com o mAP relativo e a diferença relativa de tempos de processamento para cada vídeo. A Figura 4 exibe um boxplot relacionando cada limiar de similaridade dos experimentos com o mAP relativo resultante. Por meio desse gráfico, é possível confirmar a tendência de crescimento da precisão ao aumentar o limiar de similaridade para desvio dos quadros. A Figura 5 exibe um boxplot relacionando os diferentes limiares de similaridade ao tempo de processamento de cada vídeo, além de ilustrar o aumento do tempo de execução conforme há um aumento no limiar de similaridade. Ade-

Tabela 1. Resultados dos experimentos.

Limiar (%)	mAP relativo (%)	Quadros desviados (%)	Quadros processados pelo YOLO (%)	Tempo médio de processamento por vídeo (s)	Diferença relativa do tempo de processamento por vídeo
0	4,7 ± 2,3	99,76	0,24	0,631 ± 0,080	0,900 ± 0,006
10	30,1 ± 5,6	98,24	1,76	0,766 ± 0,090	0,878 ± 0,007
20	48,4 ± 5,5	90,54	9,46	1,405 ± 0,190	0,793 ± 0,016
30	64,5 ± 4,5	79,47	20,53	2,297 ± 0,303	0,671 ± 0,024
40	78,9 ± 3,0	66,34	33,66	3,354 ± 0,409	0,535 ± 0,028
50	88,0 ± 2,4	50,69	49,31	4,632 ±0,544	0,386± 0,029
60	93,5 ± 1,8	35,41	64,59	5,885 ±0,668	0,243 ± 0,028
70	97,1 ± 5,9	23,37	76,63	6,902 ± 0,809	0,128 ± 0,024
80	98,6 ± 0,5	14,28	85,72	7,495 ± 0,904	0,061 ± 0,019
90	99,6 ± 0,2	6,78	93,22	8,094 ± 1,034	0,003 ± 0,013
YOLO	100 ± 0,0	0,00	100,00	8,296 ±1,167	0,000 ± 0,000

mais, é possível encontrar valores negativos na Figura 5, uma vez que, quando os quadros não são desviados, o tempo que esses levam para serem processados pela rede neural é maior, pois leva em consideração o tempo requerido pela comparação.

As figuras 4 e 5 mostram que o limiar de similaridade controla o compromisso entre o tempo de processamento de cada vídeo e a precisão de detecção de objetos. Os limiares de 60% e 70% apresentaram resultados interessantes no balanceamento do compromisso entre precisão e tempo de processamento. O primeiro alcançou um mAP de 93,5% e uma redução do tempo de processamento perto de 24%. Já o segundo atinge uma precisão próxima a 97% e uma redução do processamento de aproximadamente 13%.

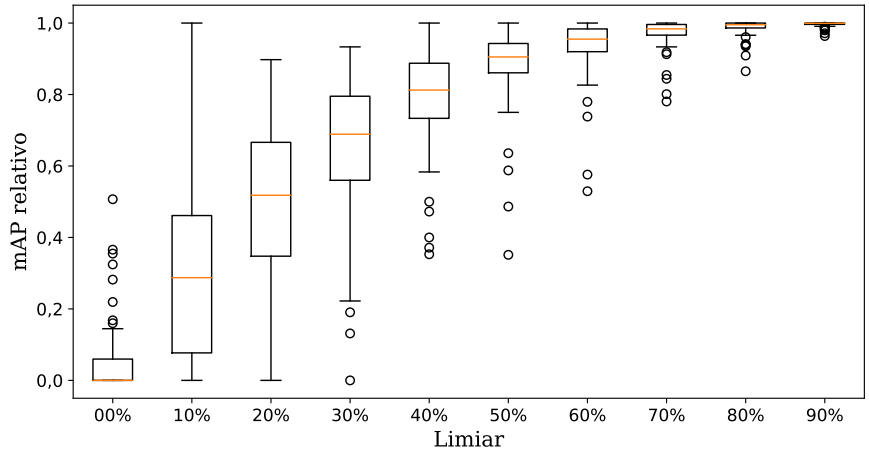

Figura 4. Boxplot do mAP de acordo com o limiar.

A Figura 5 e a Tabela 1 exibem o tempo de processamento de cada vídeo, o que não possibilita visualizar claramente a diferença do custo de processamento entre unicamente a ferramenta de detecção e o algoritmo de comparação. Por conta disso, a Tabela 2 explicita o tempo médio de processamento da função de similaridade para um único qua-

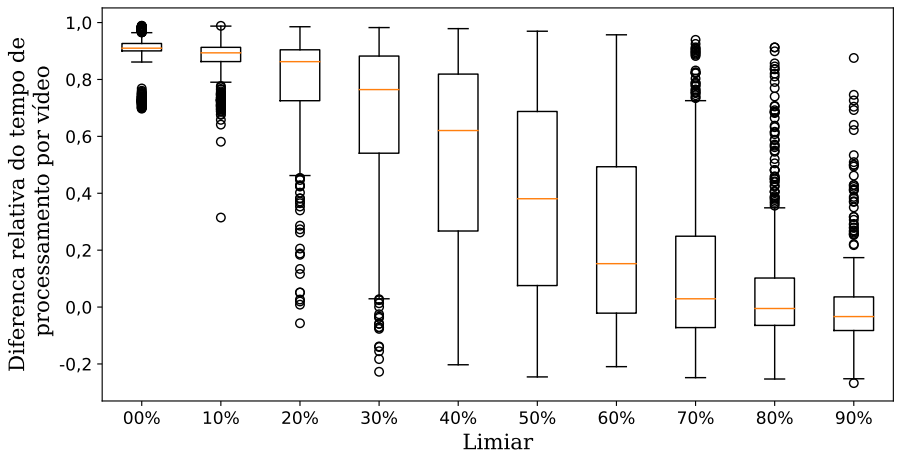

Figura 5. Boxplot do tempo de processamento por vídeo de acordo com o limiar.

Tabela 2. Tempo médio gasto para processar cada quadro.

Meio de processamento	Tempo de processamento por quadro
Função de similaridade	2,575 $\pm$ 0,021 (ms)
Detecção de objetos pelo YOLO	32,288 $\pm$ 0,034 (ms)

dro e o tempo médio de processamento de um único quadro pelo YOLO sem a utilização da comparação, isto é, utilizando somente a ferramenta de detecção. Por meio da segunda tabela, é possível observar que a métrica de comparação possui um custo inferior a 10% do custo do processamento de detecção de objetos.

A Figura 6 detalha as informações de tempo de execução dos experimentos, ilustrando a função de distribuição acumulada empírica (CDF) do tempo de processamento de cada quadro para diferentes limiares do fluxo proposto. Também é exibida a CDF do tempo de processamento dos quadros para o YOLO sem desvios. É possível visualizar que todas as curvas, com exceção da relacionada com o próprio YOLO, possuem duas regiões nitidamente muito inclinadas, sendo essas destacadas em azul e vermelho na Figura 6. Essas regiões significam um crescimento no número de amostras que têm o tempo para processar um único quadro indicado no eixo horizontal. Sendo assim, a primeira região representa os quadros que não são enviados ao YOLO, levando entre 0 e 10 milissegundos para serem processados, sendo muito coerente com o que foi encontrado na Tabela 2. Esse é o tempo necessário para executar o algoritmo de comparação, o que fica ainda mais evidente quando se observa que a única curva que não possui essa inclinação é a do próprio YOLO. Já a segunda região inclinada representa os quadros que são comparados pelo algoritmo proposto e processados pela rede neural, que demandam, na maior parte dos casos, entre 20 e 40 milissegundos para serem processados. Ademais, a curva relacionada ao processamento do YOLO, sem o algoritmo de comparação, tem a sua segunda região de inclinação encontrada um pouco antes das demais curvas, uma vez que não há o tempo requerido pelo algoritmo de comparação.

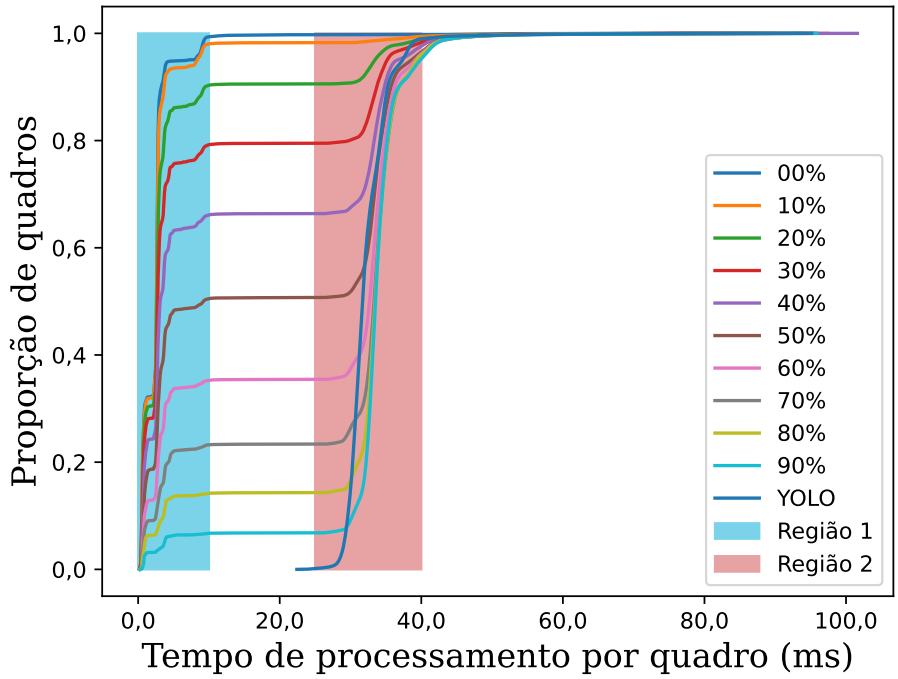

Figura 6. Gráfico CDF do tempo de processamento de cada quadro.

7. Conclusões e Trabalhos Futuros

A detecção de objetos em vídeos em tempo real é um grande desafio da visão computacional. Uma vez que o processamento de cada quadro requer tempo, pode acontecer do atraso entre a captura de uma imagem e o reconhecimento de objetos nela ser grande o suficiente para impossibilitar as aplicações desejadas. Por exemplo, se um algoritmo demorar demais a reconhecer um pedestre, um carro autônomo pode causar um acidente. No caso de câmeras de segurança, a identificação tardia de uma arma pode deixar de prevenir um roubo. Assim, ambas aplicações tornam-se inviáveis em caso de atraso, com consequências graves para seus usuários.

Neste artigo o problema do atraso em detecção de objetos em vídeos é tratado através do desvio de quadros. Os quadros recebidos da captura são comparados com o último quadro já processado e, se forem considerados similares, recebem as mesmas marcações do último quadro processado. A decisão é feita a partir de um limiar de similaridade entre os quadros. O método proposto foi aplicado a vídeos do ImageNet-VID [Deng et al. 2009], um conjunto de dados consolidado na literatura. Os experimentos mostraram que existe um compromisso entre a acurácia do método e o tempo de processamento. Além disso, também é possível mostrar que tal compromisso é controlado pelo número de quadros desviados, que é ligado ao limiar de similaridade. Assim, mostra-se que, com um limiar de similaridade de 70%, é possível obter um ganho no tempo de processamento de 12,8% a um custo de 2,9% da precisão média, com relação ao algoritmo tradicional de detecção de objetos.

Em trabalhos futuros, pretende-se comparar o método de similaridade utilizado com outros métodos divulgados na literatura, além de realizar aplicações em tempo real

para medir o desempenho. Ademais, pensa-se em desenvolver um algoritmo adaptável, que se molde de acordo com a precisão de detecção e os recursos de computação e comunicação disponíveis.

Referências

Aharon, S., Louis-Dupont, Ofri Masad, Yurkova, K., Lotem Fridman, Lkdci, Khvedchenya, E., Rubin, R., Bagrov, N., Tymchenko, B., Keren, T., Zhilko, A., and Eran-Deci (2021). Super-gradients.

Ashraf, A. H., Imran, M., Qahtani, A. M., Alsufyani, A., Almutiry, O., Mahmood, A., Attique, M., and Habib, M. (2022). Weapons detection for security and video surveillance using cnn and yolo-v5s. *CMC-Comput. Mater. Contin*, 70:2761–2775.

Ćorović, A., Ilić, V., Đurić, S., Marijan, M., and Pavković, B. (2018). The real-time detection of traffic participants using yolo algorithm. In *2018 26th Telecommunications Forum (TELFOR)*, pages 1–4. IEEE.

Cristiani, A. L., Nespolo, R. G., Maschi, L. F. C., Nakamura, L., Ueyama, J., and Meneguette, R. I. (2018). Uma nova arquitetura para classificação de tráfego de véıculos baseado em processamento de imagens. In *Anais do II Workshop de Computação Urbana*, Porto Alegre, RS, Brasil. SBC.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Fang, W., Wang, L., and Ren, P. (2019). Tinier-yolo: A real-time object detection method for constrained environments. *IEEE Access*, 8:1935–1944.

Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.

Huang, R., Pedoeem, J., and Chen, C. (2018). Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers. In *2018 IEEE international conference on big data (big data)*, pages 2503–2510. IEEE.

Itseez (2015). Open source computer vision library. `https://github.com/itseez/opencv`.

Jocher, G., Chaurasia, A., and Qiu, J. (2023). YOLO by Ultralytics.

Lan, W., Dang, J., Wang, Y., and Wang, S. (2018). Pedestrian detection based on yolo network model. In *2018 IEEE international conference on mechatronics and automation (ICMA)*, pages 1547–1551. IEEE.

Lee, J. and Hwang, K.-i. (2022). Yolo with adaptive frame control for real-time object detection applications. *Multimedia Tools and Applications*, 81(25):36375–36396.

Liang, S., Wu, H., Zhen, L., Hua, Q., Garg, S., Kaddoum, G., Hassan, M. M., and Yu, K. (2022). Edge yolo: Real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):25345–25360.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer.

Lu, Y., Zhang, L., and Xie, W. (2020). Yolo-compact: an efficient yolo network for single category real-time object detection. In *2020 Chinese control and decision conference (CCDC)*, pages 1931–1936. IEEE.

Narejo, S., Pandey, B., Esenarro Vargas, D., Rodriguez, C., and Anjum, M. R. (2021). Weapon detection using yolo v3 for smart surveillance system. *Mathematical Problems in Engineering*, 2021:1–9.

Nguyen, H. H., Ta, T. N., Nguyen, N. C., Pham, H. M., Nguyen, D. M., et al. (2021). Yolo based real-time human detection for smart video surveillance at the edge. In *2020 IEEE Eighth International Conference on Communications and Electronics (ICCE)*, pages 439–444. IEEE.

Pacheco, R. and Couto, R. (2021). Particionamento de redes neurais profundas com saídas antecipadas. In *Anais Estendidos do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 169–176, Porto Alegre, RS, Brasil. SBC.

Padilla, R., Netto, S. L., and da Silva, E. A. B. (2020). A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Shafiee, M. J., Chywl, B., Li, F., and Wong, A. (2017a). Fast yolo: A fast you only look once system for real-time embedded object detection in video. *arXiv preprint arXiv:1709.05943*.

Shafiee, M. J., Mishra, A., and Wong, A. (2017b). Deep learning with darwin: Evolutionary synthesis of deep neural networks.

Shang, X., Ren, T., Guo, J., Zhang, H., and Chua, T.-S. (2017). Video visual relation detection. In *ACM International Conference on Multimedia*, Mountain View, CA USA.

Shinde, S., Kothari, A., and Gupta, V. (2018). Yolo based human action recognition and localization. *Procedia computer science*, 133:831–838.

Van Der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in science & engineering*, 13(2):22–30.

Zuraimi, M. A. B. and Zaman, F. H. K. (2021). Vehicle detection and tracking using yolo and deepsort. In *2021 IEEE 11th IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pages 23–29. IEEE.