



ConBRepro

XIV CONGRESSO BRASILEIRO DE ENGENHARIA DE PRODUÇÃO

I CIGELUBRA

Congresso Internacional de Gestão e Engenharia Luso-Brasileiro

04 a 06 de dezembro de 2024

Energias Limpas nas Engenharias

Comparação de performance do YOLO V8n: um ponto de vista multiobjetivo

João Augusto de Almeida

Escola Politécnica - PUCPR

Tiago Melo da Silva

PPGEPS - PUCPR

Ivan Lucas Reis da Silva

PPGEPS - PUCPR

Gilberto Reynoso Meza

PPGEPS - PUCPR

Resumo: O presente artigo explora a natureza multiobjetiva da função de custo utilizada no modelo YOLO V8n, amplamente empregado em tarefas de visão computacional para detecção e classificação de objetos. A pesquisa propõe uma análise detalhada das interações entre as métricas de desempenho do modelo, buscando verificar se a função de custo, composta por múltiplos componentes ponderados, pode ser enquadrada como um problema multiobjetivo. Para isso, foram realizados experimentos utilizando diferentes configurações de pesos e analisados os efeitos sobre as métricas principais. Os resultados obtidos contribuem para uma melhor compreensão da estrutura da função de custo do YOLO, sugerindo caminhos para futuras investigações em otimização multiobjetiva aplicada a modelos de aprendizado profundo.

Palavras-chave: YOLO, visão computacional, função de custo, otimização multiobjetiva, aprendizado profundo.

Performance comparison of YOLO V8n: a multiobjective point of view

Abstract: This paper explores the multi-objective nature of the loss function used in the YOLO V8n model, widely applied in computer vision tasks for object detection and classification. The research offers a detailed analysis of the interactions between the model's performance metrics, aiming to determine whether the loss function, composed of multiple weighted components, can be framed as a multi-objective problem. Experiments were conducted using different weight configurations, and their effects on key metrics were analyzed. The findings contribute to a deeper understanding of YOLO's loss function structure, suggesting avenues for future research in multi-objective optimization applied to deep learning models.

Keywords: YOLO, computer vision, loss function, multi-objective optimization, deep learning.

1 Introdução

YOLO é o acrônimo para *You Only Look Once* (Você só olha uma vez), que muito além de acrônimo, é um poderoso framework open source que usa inteligência artificial, utilizado em visão computacional.

Como mencionado, o YOLO é uma poderosa ferramenta na visão computacional, sendo seu objetivo lidar com imagens e objetos. Nesse sentido, o YOLO é capaz de identificação, segmentação, pose estimation (estimação de pose; detecção de pontos chave na imagem), rastreamento e classificação de objetos em imagens e/ou vídeos.

O YOLO é baseado em um robusto modelo de deep learning, contendo diversas camadas e passando por algumas redes neurais diferentes para chegar no resultado final. Para auxiliar neste processo, é evidente que o modelo necessita de uma função de custo, que guie o aprendizado do modelo para a direção correta. Em relação a esta função, ela é uma equação composta de 3 expressões diferentes, cada uma considerando um aspecto diferente da entrada da função (valores referentes à imagem ou ao vídeo). Para tornar o aprendizado mais eficiente, cada expressão possui um multiplicador que define o peso que cada expressão tem na função de custo, ou seja, o peso que cada aspecto da análise tem.

Como mencionado anteriormente, foram implementadas estas constantes (pesos) nas expressões para melhor guiar o aprendizado do modelo. Desta forma, devido à existência destes pesos, é possível fazer uma análise multiobjetiva, com o intuito de descobrir se existe uma combinação de valores dos pesos que melhor se adapte ao respectivo problema analisado.

Desta forma, neste artigo, será feita uma análise, do ponto de vista multiobjetivo, da função do custo do YOLO.

2 Referencial teórico

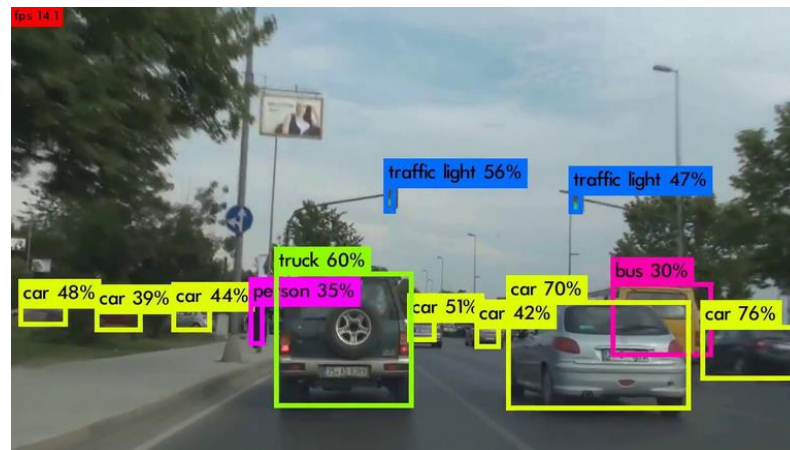
2.1 Classificação e segmentação de dados não estruturados

Em primeiro lugar, é necessário explicar o que são dados não estruturados: dados não estruturados são tipos de dados que não seguem uma organização específica, e podem ser gerados por humanos ou máquinas. Para adicionar um contraste, um tipo de dado estruturado seria aquele visto em um banco de dados, já o não estruturado, pode ser desde um arquivo até um arquivo de áudio, ou imagem. No caso do YOLO, estamos falando de dados não estruturados do tipo imagem ou vídeo.

Além disso, é necessário falar um pouco de machine learning e como este problema de segmentação e classificação está atrelada a esta área da computação. Machine learning é, basicamente, uma subcampo da inteligência artificial, que, como dita pela própria tradução, é fazer com que uma máquina perceba e aprenda padrões em dados (sejam estruturados ou não) a fim de realizar alguma tarefa. Neste caso, a tarefa a ser realizada, é a de identificar e segmentar cada objeto em uma imagem ou vídeo.

Com isso em mente, é possível explicar o que realmente é a classificação e segmentação em imagens e vídeos.

Figura 1 - Exemplo de classificação e segmentação



Fonte: BRANDIZZI (2024)

Observando a figura 1, é possível ver um clássico exemplo de segmentação e classificação: no âmbito da segmentação, é possível ver que o algoritmo segmentou todos os objetos relevantes em análise, desenhando uma caixa ao redor deles; já no âmbito de classificação, é possível ver que cada objeto está sendo classificado conforme sua representação, sendo distintos pelas cores das caixas desenhadas, ainda com a porcentagem de certeza da classificação, dada pelo modelo.

2.2 YOLO e suas versões

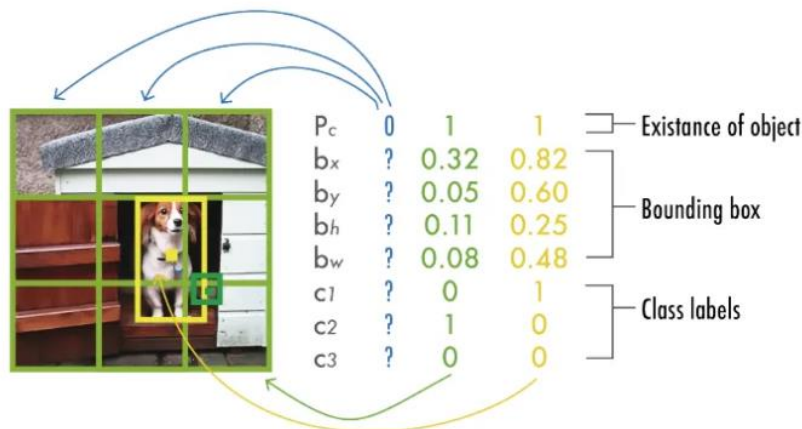
Apesar de o problema e o objetivo serem nítidos, o meio do caminho, isto é, a solução, não é tão nítido assim, pois desenvolver um modelo tão robusto a ponto de conseguir fazer todo este reconhecimento leva tempo e muito estudo. Nesse sentido, conforme foram sendo descobertas formas mais eficazes de tratar este problema, o YOLO foi recebendo atualizações que implementavam algo novo em seu algoritmo. Dessa forma, de acordo com Terven et al. (2023), a evolução das versões do YOLO é descrita conforme:

— YOLOv1

-A primeira versão do YOLO é a base para o entendimento do funcionamento do algoritmo e como as versões seguintes impactaram o modelo.

-Nesse sentido, a primeira versão do YOLO foi capaz de performar melhor que todos os modelos de identificação e segmentação anteriores, utilizando a tecnologia de *bounding boxes* (Caixas de limitação)

Figura 2 - Funcionamento do YOLO



Fonte: TERVERN et al. (2023)

-Este sistema, faz com que seja necessário apenas uma verificação de todos os pixels da imagem, causando uma extrema redução no tempo de inferência, tal motivo pelo qual o YOLO performou muito melhor que qualquer outro modelo na época.

-Por mais que o YOLOv1 tenha performado melhor que qualquer outro sistema de detecção de objetos, ainda é insuficiente em alguns pontos: limitação de detecção de objetos (2) na mesma célula, dificuldade de predição de objetos com resoluções que não estavam presentes nos dados de treinamento e dificuldade de aprendizado de algumas features devido ao *downsampling* de algumas camadas da rede neural.

— YOLOv2

- Aumento da performance de treinamento em resoluções maiores;
- Remoção de camadas “densas”, deixando a arquitetura 100% *convolucional*;
- Utilização de *bounding boxes* âncoras para cada célula (formatos pré-definidos), a fim de facilitar a predição das *bounding boxes*, usando *dimension clustering* para selecionar os 5 mais apropriados formatos;
- Pelo fato de o YOLOv2 não utilizar camadas totalmente conectadas, é possível, desta forma, treinar inputs de diferentes tamanhos (resoluções)

— YOLOv3

- Para melhorar a predição de *anchor bounding boxes*, o YOLOv3 faz a predição de 5 *bounding boxes* usando regressão logística, assim, sendo escolhida a *anchor bounding box* com maior intersecção em relação à caixa verdadeira;
- Também, foi mudado o sistema de classificação, permitindo que um objeto tenha múltiplas classes (por exemplo, um mesmo objeto pode ser um ser humano e um homem)
- Foi mudado drasticamente a arquitetura do YOLO, sendo adicionado um *backbone*, permitindo *multi-scale predictions* e maior eficiência do algoritmo.

— **YOLOv4**

- Melhora no treinamento de dados, utilizando a filosofia "*Bag of Freebies*", que promove uma série de alterações nas estratégias de treinamento, porém, aumentando o tempo de inferência dos dados;
- Implementação de treinamento auto adversário, para deixar o modelo mais robusto, fazendo "ataques" à imagem de input, fazendo com que a imagem passe a impressão de que o objeto não está na imagem;
- A fim de achar os melhores hiper parâmetros para treinamento, é realizado mutações com algoritmos genéticos durante 10% do período de treinamento. Isso faz com que a curva de aprendizado seja diminuída aos poucos, terminando o processo com uma drástica redução.

— **YOLOv5**

- Adição de um algoritmo de pré-treinamento chamado *AutoAnchor*, que basicamente ajusta as *anchor bounding boxes* caso não estejam próprias para o *dataset* e configurações de treinamento;
- Melhorias na arquitetura, incluindo modificação do *backbone* e adição de uma camada SPPF (*Spatial Pyramid Pooling Fast*), para acelerar a computação do algoritmo.

— **YOLOv6**

- Novo *backbone* e neck no algoritmo;
- Novo método de rotulação, utilizando a filosofia do TOOD;
- Novos erros de classificação e regressão. Para classificação foi utilizado o *VariFocal loss* e para regressão foram utilizados o SloU e GloU.

— **YOLOv7**

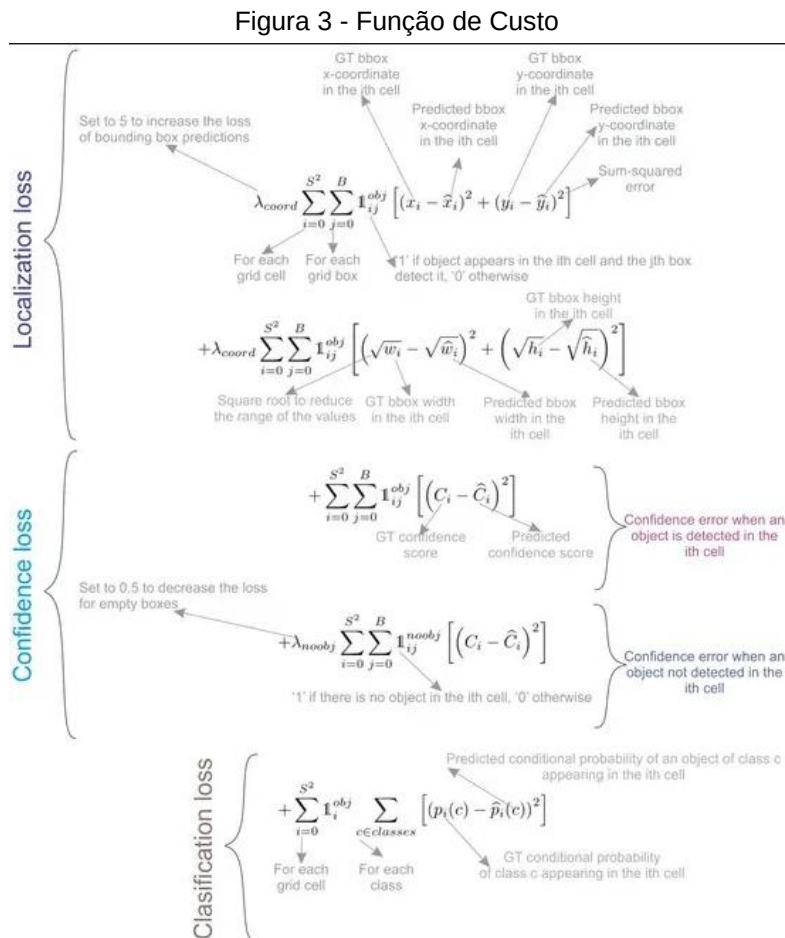
- Diversas melhorias no algoritmo e arquitetura já presentes, aumentando a precisão do modelo sem afetar o tempo de inferência, apenas o tempo de treinamento do modelo.

— **YOLOv8**

- Modificações na arquitetura;
- Utilização de loss functions do CloU e DFL para *bounding-box loss* e *binary-cross-entropy* para *classification loss*.

2.3 Função de custo do YOLO

Como dito anteriormente, assim como qualquer modelo de machine learning, o YOLO depende de uma função de custo para ditar seu aprendizado, sendo uma das etapas mais cruciais de todo o algoritmo, pois, novamente, é ela que orienta o modelo a tomar as decisões “menos erradas”. Com isso em mente, reforça Terven et al., a função de custo é composta de 3 partes: localização, confiança e classificação, como é possível identificar abaixo, na figura 3.



Fonte: TERVEN et al. (2023)

— LOCALIZATION

-A *localization loss* basicamente computa erros de posicionamento das *bounding boxes* de cada célula em relação ao objeto detectado, ou seja, basicamente é obtido o valor de posicionamento previsto daquela caixa e é comparado com o que realmente se observa na análise. É válido notar que a *localization loss* só é computada em caixas que POSSUEM objetos, ou seja, só acrescentando ao *total loss* caso haja um objeto presente naquela célula.

-O primeiro termo do *localization loss* computa o erro da centralização da *bounding box* em relação ao objeto detectado por ela. Neste caso os valores estão elevados ao quadrado para evitar erros negativos.

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

-O segundo termo computa o erro em relação ao tamanho da *bounding box*, utilizando os valores de largura e comprimento de cada caixa. Lembrando que, os valores estão dentro de raízes quadradas para reduzir a variação dos valores, e estão ao quadrado para evitar valores negativos.

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

— CONFIDENCE

-A *confidence loss* computa os erros de intervalo de confiança de um objeto de fato ser um objeto ou não.

-O primeiro termo da *confidence loss* computa o erro para cada *bounding box* de cada célula do intervalo de confiança previsto e real quando um objeto é detectado na célula.

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

-O segundo termo computa o mesmo erro do primeiro termo, porém para quando um objeto NÃO é detectado naquela célula.

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

— CLASSIFICATION

-A *classification loss* computa o erro da probabilidade condicional do objeto da classe C aparecer na célula i. Lembrando que a *classification loss* só é calculada quando a célula i conter um objeto.

$$+ \sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in classes} [(p_i(c) - \hat{p}_i(c))^2]$$

3 Métodos e ferramentas

3.1 Otimização Multiobjetiva e frente de Pareto

Em geral, um problema multiobjetivo pode ser definido da seguinte forma:

$$(\min_{\theta}) J(\theta) = [J_1(\theta), \dots, J_m(\theta)]$$

Mediante a:

$$g(\theta) \leq 0$$

$$h(\theta) = 0$$

$$\theta_{\max} \leq \theta_i \leq \theta_{\min}, i = [1, \dots, n]$$

onde θ é o vetor de decisão, $J(\theta)$ é o vetor objetivo, $g(\theta)$ e $h(\theta)$ são, respectivamente, os vetores de desigualdade e igualdade e $\theta_{\max}$, $\theta_{\min}$ sendo os limites para a decisão espaço para a variável θ_i .

Uma solução θ^* é Pareto-ótima se não existe outra solução θ' tal que: que melhore pelo menos um objetivo sem piorar outro. Existem várias alternativas para visualizar a frente de Pareto em múltiplas dimensões. Uma das mais simples e compactas é a de coordenadas paralelas. A visualização por coordenadas paralelas consiste em representar cada objetivo $J_i(\theta)$ como um eixo paralelo. Cada linha que cruza esses eixos representa uma solução θ .

3.2 Método para criação das soluções

Para criar as soluções, primeiramente foi usado o Python para gerar todos os resultados. Nele, foi fundamental o uso do framework da ultralytics, que contém o YOLO e todas as suas versões. Além disso, bibliotecas acessórias como pandas, numpy e as dependências do YOLO também foram usadas.

Em seguida, foi utilizado o modelo YOLOv8n com a base de dados (dados) (que será explicada no próximo tópico). Para abordar diversas combinações diferentes do peso da função de custo (que configura a frente de Pareto) foram utilizados 3 laços de repetições,

cada um contendo um array com 5 itens para cada peso (totalizando 125 combinações diferentes). Nas configurações iniciais do YOLO, as únicas configurações relevantes foram a de utilizar 50 epochs para cada combinação, juntamente com os drivers CUDA, para permitir o uso de aceleração de GPU para deixar o processo mais rápido.

Depois que os resultados foram salvos em pastas, foi utilizado outro script em Python para transferir todos os pesos para uma planilha do excel, de lá gerando o gráfico que será discutido na seção Discussão de resultados.

Subsection sobre base de dados utilizadas

3.3 Base de dados utilizada para treinamento, validação e teste

Em um estudo recente, conduzido por bla bla, foi identificado que os conjuntos de dados públicos utilizados para detecção de células sanguíneas, como o BCCD e o BCD, apresentam problemas significativos, incluindo erros de rotulagem, tamanho insuficiente de amostras e baixa qualidade dos dados, prejudicando a eficácia dos modelos de aprendizado de máquina.

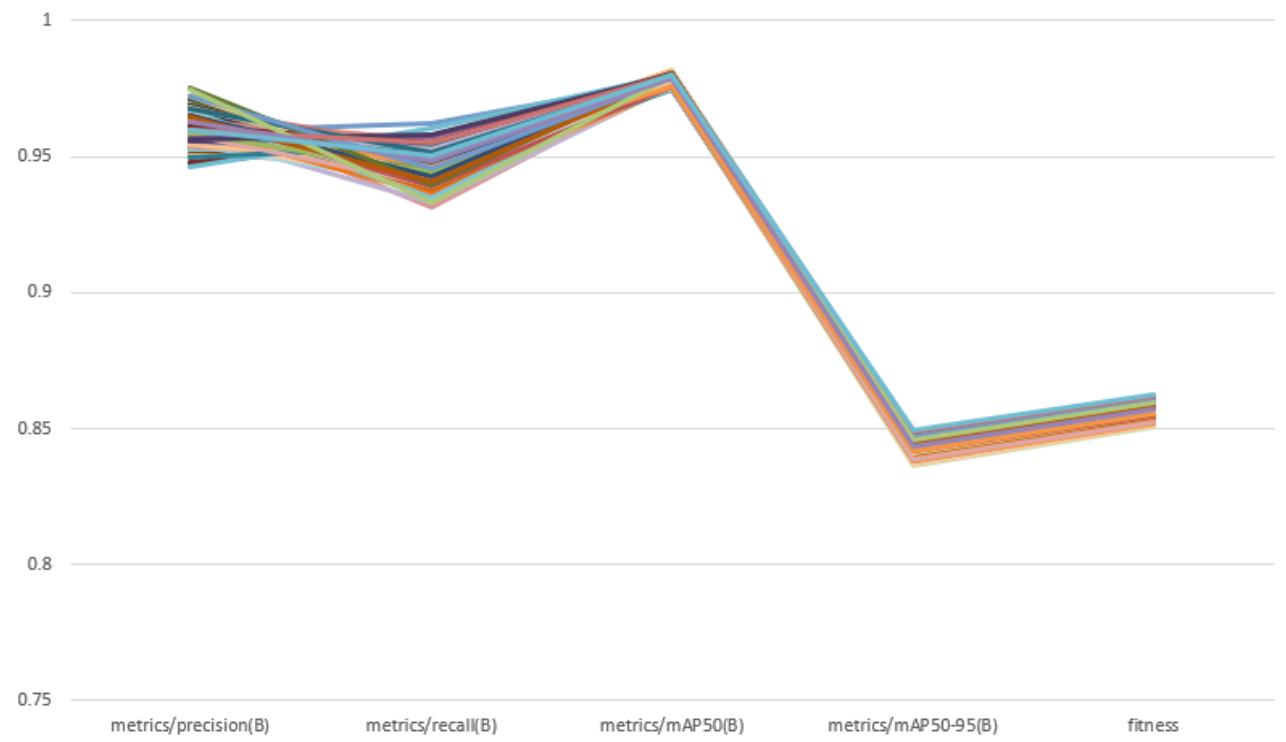
Para superar essas limitações, foi conduzido um processo abrangente de limpeza, re-rotulagem e integração desses datasets, incorporando também informações dos conjuntos PBC e Raabin-WBC. O resultado é o TXL-PBC, um novo conjunto de dados de alta qualidade e balanceado, contendo 1008 imagens para treinamento, 288 para validação e 144 para teste.

O TXL-PBC passou por um rigoroso processo de anotação, combinando técnicas manuais e automáticas com o modelo YOLOv8n, seguido de auditoria manual, garantindo a precisão e consistência das anotações. Este dataset aborda os problemas de rotulagem incorreta presentes nos datasets originais e melhora significativamente a qualidade dos dados disponíveis para treinamento de modelos de detecção de células sanguíneas (LU; XI, 2024, p. 2407.13214).

4 Discussão de resultados

Baseado no problema apresentado e da definição de o que é um problema multiobjetivo e como funciona sua otimização, foram testadas 125 combinações de pesos diferentes para os componentes da função de custo, sendo avaliados seguintes critérios: precisão, recall, mAP50, mAP95 e fitness.

Figura 4 - Coordenadas Paralelas
Visualização de Coordenadas Paralelas



Fonte: o autor (2024)

Após visualizar a figura 4 (gráfico de visualização de coordenadas paralelas), é possível enxergar que, de fato, existe uma relação multiobjetiva entre os pesos da função de custo. No entanto, é necessário pontuar que os valores de todas as métricas possuem influência uns nos outros, de tal forma que as métricas 'mAP50-95' e 'fitness' não possuem nenhuma influência nas outras 3 métricas, desta forma, não fazendo parte do problema multiobjetivo. Vale ressaltar, também, que foram escolhidas estas métricas apenas para a comprovação de um problema multiobjetivo, mas que, na realidade, podem existir outras métricas de avaliação de performance (não citadas/consideradas nesta análise) que podem entrar neste problema específico de otimização multiobjetiva, e que são tão importantes para problemas de machine learning quanto os indicadores descritos.

5 Conclusões

Levando em conta as discussões previamente levantadas, é nítido que existe um problema de otimização multiobjetiva com os pesos da função de custo do YOLO, e que, dependendo da otimização, diferentes resultados podem ser alcançados por meio desta otimização, sendo necessária a avaliação do problema e contexto para que uma possa se encaixar com as necessidades deste.

Referências

BRANDIZZI, L. **Visão computacional: o que é e como funciona**. Disponível em: <<https://www.serpro.gov.br/menu/noticias/noticias-2020/visao-computacional-oqueeh-comofunciona>> Acesso em: 22 out. 2024.

LU, G. XI, L. TXL-PBC: a freely accessible labeled peripheral blood cell dataset. **arXiv**. Flagstaff, p. 2407.13214, 2024.

TERVEN et al. Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. **Machine Learning and Knowledge Extraction**, Queretaro, v.5, n.4, p. 1680-1716, 2023.