UNIVERSIDADE FEDERAL DE GOIÁS / INSTITUTO DE INFORMÁTICA

Redes Neurais Artificiais aplicadas à Visão Computacional

Um review sobre as principais tarefas e arquiteturas da área

Marcos Vinicius Satil Medeiros

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Imagem de capa gerada por IA

UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
INSTITUTO DE INFORMÁTICA (INF)

MARCOS VINICIUS SATIL MEDEIROS

REDES NEURAIS ARTIFICIAIS APLICADAS À VISÃO COMPUTACIONAL

Um review sobre as principais tarefas e arquiteturas da área

Goiânia
2024

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): **MARCOS VINICIUS SATIL MEDEIROS**

Título do trabalho:

REDES NEURAIS ARTIFICIAIS APLICADAS À VISÃO COMPUTACIONAL

Um review sobre as principais tarefas e arquiteturas da área

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [**X**] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Marcos Vinicius Satil Medeiros**, **Discente**, em 15/02/2024, às 19:19, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Fernando Marques Federson**, **Professor do Magistério Superior**, em 12/09/2024, às 11:09, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4383418** e o código CRC **B1067AE2**.

Referência: Processo nº 23070.008395/2024-46 SEI nº 4383418

MARCOS VINICIUS SATIL MEDEIROS

REDES NEURAIS ARTIFICIAIS APLICADAS À VISÃO COMPUTACIONAL

Um review sobre as principais tarefas e arquiteturas da área

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.
Orientador: Prof. Dr. Fernando Marques Federson

Goiânia
2024

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

MEDEIROS, MARCOS VINICIUS SATIL
REDES NEURAIS ARTIFICIAIS APLICADAS À VISÃO COMPUTACIONAL [manuscrito] : Um review sobre as principais tarefas e arquiteturas da área / MARCOS VINICIUS SATIL MEDEIROS. - 2024.
178 f.

Orientador: Prof. Dr. FERNANDO MARQUES FEDERSON.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Instituto de Informática (INF), Inteligência Artificial, Goiânia, 2024.

1. inteligência artificial. 2. redes neurais. 3. visão computacional. I. FEDERSON, FERNANDO MARQUES, orient. II. Título.

CDU 004

MARCOS VINICIUS SATIL MEDEIROS

REDES NEURAIS ARTIFICIAIS APLICADAS À VISÃO COMPUTACIONAL

Um review sobre as principais tarefas e arquiteturas da área

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Data da Aprovação: 08 de fevereiro de 2024.

Prof. Dr. Fernando Marques Federson
Orientador (INF-UFG)

Prof. Dr. Aldo André Díaz Salazar
Coordenador de TCC do BIA (INF-UFG)

Prof. Dr. Vinícius Sebba Patto
Coordenador do BIA (INF-UFG)

Prof. Dr. Iwens Gervasio Sene Junior
(INF-UFG)

MARCOS VINICIUS SATIL MEDEIROS

REDES NEURAIS ARTIFICIAIS APLICADAS À VISÃO COMPUTACIONAL

Um review sobre as principais tarefas e arquiteturas da área

RESUMO

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Visão Computacional**. Uma ilustração e sua narrativa descrevem os períodos de trabalho. Os Apêndices contêm os Termos de Aceite de Entrega e os resultados obtidos durante cada período de trabalho.

Palavras-chave: inteligência artificial, redes neurais, visão computacional.

ABSTRACT

This Course Completion Report aims to bring together the results of my journey to become an expert in **Computer Vision**. An illustration and its narrative describe the work periods. The Appendices contain the Delivery Acceptance Terms and the results obtained during each work period.

Keywords: artificial intelligence, neural networks, computer vision.

Goiânia

2024

Minha Jornada

Marcos Vinicius Satil Medeiros

Especialista em: Visão Computacional

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

MINHA JORNADA

Nome: Marcos Vinicius Satil Medeiros

Especialidade: Visão Computacional

Objetivo deste documento

Durante o processo da disciplina Residência em IA[1], foram gerados diversos resultados na construção da minha especialização. A cada semana, um conjunto de resultados foi formalizado por um Termo de Aceite de Entrega e avaliado por uma banca, considerando o planejado e o realizado para o período. Este documento tem como objetivo descrever esses resultados obtidos, fazendo referência aos Termos de Aceite de Entrega e seus documentos associados.

Minha Jornada

Minha jornada teve início na **Semana 1** com a definição da área de conhecimento para a minha especialização. Decidi que seria algo envolvendo Visão Computacional, visto que é o ramo da Inteligência Artificial que mais me atrai e também por já possuir conhecimento prévio através de disciplinas do Bacharelado em Inteligência Artificial. Além disso, minha participação em projetos de pesquisa correlatos contribuiu significativamente para minha decisão.

Nessa fase inicial, dediquei-me a uma investigação nos principais congressos de tecnologia. O objetivo era classificar os estudos de acordo com a metodologia adotada pelo *Conference on Computational Science and Computational Intelligence"* (CSCI), uma conferência reconhecida por divulgar trabalhos científicos em diversas áreas tecnológicas. Assim sendo, este trabalho alinha-se aos temas discutidos no CSCI'23, abordando aspectos como *Neural networks and applications, Emerging technologies; and Applications (including: computer vision, signal processing, Large language models, emerging applications)*. Essa imersão não apenas orientou minha pesquisa, mas também contribuiu para o

[1] Dez semanas, entre setembro de 2023 e janeiro de 2024.

estabelecimento de objetivos, refletidos em um cronograma inicial para as entregas semanais.

Na sequência, durante a **Semana 2**, foi feita uma revisão bibliográfica, abarcando desde os primórdios até os modelos mais avançados. O foco foi procurar sobre os principais artigos científicos e *surveys*, e com base nesse estudo anotar os pontos encontrados que são necessários para um especialista na área conhecer. Progressivamente, adentrei os avanços mais recentes, examinando os modelos de última geração, os chamados SOTA (*State-of-the-Art*). Além disso, durante essa semana, concentrei-me na identificação e elencagem das principais tarefas relacionadas a redes neurais. Desde tarefas mais fundamentais, como classificação e reconhecimento de padrões, até desafios mais avançados, como detecção de objetos, segmentação semântica e geração de imagens. Esse mapeamento proporcionou uma compreensão das diversas aplicações e das complexidades inerentes. As referências consultadas e as observações pertinentes foram registradas detalhadamente no **Apêndice 1**.

Na **Semana 3**, concentrei meus esforços no estudo sobre dados e o no *pipeline* dos modelos de Visão Computacional, visando compreender cada etapa da preparação e do processamento de imagens, elementos fundamentais para o sucesso dos algoritmos empregados. No estágio inicial do pré-processamento de dados, destaco a importância da limpeza e do redimensionamento de dados, que são fases realizadas com o propósito de economizar tempo e recursos. Ademais, ressalto abordagens como *Transfer Learning*, *Self-Supervised Learning* e *Active Learning*. Essas estratégias proporcionam eficiência, reduzindo a dependência de anotações manuais e otimizando o uso de conjuntos de dados. Em seguida, é descrita uma das etapas mais importantes no treinamento de modelos: a anotação de dados. A importância das plataformas de anotação também foi enfatizada, destacando sua relevância na automação, gerenciamento e aceleração do processo. Essas ferramentas são essenciais para o sucesso de projetos, e oferecem uma base sólida para lidar com a complexidade da anotação de dados em larga escala. O detalhamento desse estudo pode ser encontrado no **Apêndice 2**.

As **Semanas 4 e 5** foram dedicadas à pesquisa exploratória e ao levantamento de frameworks/bibliotecas. Esse período permitiu a identificação da forma de implementação

das técnicas previamente descritas e de artigos mais específicos, contribuindo para a fundamentação do trabalho.

Durante a **Semana 4**, dediquei-me na pesquisa sobre as aplicações da Visão Computacional. O objetivo era identificar um projeto que abordasse todo o *pipeline* de dados, com foco na classificação e controle de qualidade de culturas agrícolas. A inspiração para esse projeto veio da leitura do paper *"Foundation Models in Smart Agriculture: Basics, Opportunities, and Challenges"*. Este artigo aborda os fundamentos básicos e oportunidades da agricultura inteligente, embora seu foco principal seja em modelos de linguagem, têm-se diversas ideias que podem ser aplicadas.

Na semana seguinte, a **Semana 5**, realizei uma pesquisa mais focada, onde concentrei esforços na busca de conjuntos de dados públicos que fossem condizentes com o tema proposto. Antecipando as próximas etapas, conduzi uma análise dos principais frameworks da área, delineando as escolhas e configurações necessárias para a construção de uma solução. Essa avaliação, fundamentada em artigos acadêmicos, aplicações estudadas previamente e na documentação oficial dessas bibliotecas, teve como propósito preparar o terreno para as experimentações. Os resultados obtidos nessa etapa estão detalhados no **Apêndice 3**.

As **Semanas 6 e 7** marcaram o início do desenvolvimento e testes de modelos de classificação. O objetivo principal era avaliar a eficácia desses modelos na identificação precisa das classes presentes no conjunto de dados selecionado. Após uma revisão, explorei arquiteturas renomadas, como ResNet, DenseNet, VGG, Inception, EfficientNet, HRNet e Vision Transformers (ViT). A implementação e teste desses modelos foram conduzidos utilizando bibliotecas como PyTorch e Fast.ai. O treinamento e a avaliação dos modelos foram fundamentados nas métricas acurácia, precisão, recall e F1-score. Os resultados obtidos proporcionaram uma análise, destacando variações significativas no desempenho entre as diferentes arquiteturas.

Ao realizar testes dos modelos em dados reais, ampliei a aplicabilidade prática desse tipo de solução. Os experimentos finais revelaram que as redes neurais convolucionais apresentaram desempenhos satisfatórios, enquanto os Vision Transformers mostraram um desempenho relativamente inferior. Esses resultados apontam para a importância de escolher arquiteturas adequadas em função do problema, abrindo caminho para futuras

pesquisas e otimizações na busca por uma eficácia melhor na classificação. A documentação dos experimentos e os códigos produzidos estão presentes no **Apêndice 4**.

Em seguida, nas **Semanas 8 e 9**, parti para o desenvolvimento e testes de modelos de detecção, visando a identificação e localização de objetos em imagens. O processo envolveu desde a seleção de frameworks e conjuntos de dados até a compreensão detalhada de arquiteturas e técnicas de treinamento. O ponto de partida foi uma análise da tarefa como um todo, destacando os métodos de um único estágio e os métodos de dois estágios.

Para a implementação desses conceitos, optei por utilizar bibliotecas prontas amplamente reconhecidas, como Ultralytics, Detectron2 e Super Gradients. Ao longo desse processo, explorei diversas arquiteturas de modelos renomadas, com especial ênfase na arquitetura YOLO. Aprofundei-me nos detalhes do funcionamento desta arquitetura, compreendendo suas camadas principais e suas configurações de treinamento, destacando a aplicação de *Transfer Learning* e o uso de hiperparâmetros. A condução dos experimentos envolveu não apenas o treinamento dos modelos, mas também suas avaliações, empregando diversas métricas de detecção de objetos, como Precisão, Revocação, F1-Score, IoU, Average Precision e mAP. Além disso, o tempo de inferência também foi considerado, reconhecendo a importância da eficiência temporal em cenários reais de aplicação. Os resultados relativos aos experimentos realizados, os códigos e a documentação estão disponíveis no **Apêndice 5**.

Por fim, a **Semanas 10** foi dedicada ao estudo de caso de modelos de segmentação, onde foram exploradas arquiteturas populares, como VGG-16, DeepLab e U-Net, e uma comparação entre os métodos de segmentação. Na análise comparativa, destaco as diferenças entre a Segmentação Semântica e de Instância, ressaltando suas vantagens e desvantagens em diferentes cenários. A precisão na identificação e o tratamento de objetos sobrepostos foram considerados, juntamente com as capacidades de processamento em tempo real, evidenciando a adequação de cada técnica a depender da aplicação. Explorando avanços na tarefa de segmentação, destaco a eficácia das abordagens híbridas, que combinam aprendizagem profunda com outros métodos.

Além disso, foi abordada a evolução na aprendizagem auto-supervisionada e o impacto do mecanismo de atenção, especialmente em modelos baseados em *Transformers*,

onde também introduzimos o *Segment Anything Model* (SAM), uma inovação significativa, com capacidade única de realizar tarefas complexas de segmentação de imagens com precisão e versatilidade sem precedentes. Detalho a arquitetura do SAM, enfatizando os componentes principais. Nos resultados obtidos com o SAM, a capacidade *zero-shot* do modelo ficou clara em suportar diferentes modos de segmentação, desde a totalmente automática até a semi-automática com entrada de *bounding box*, ponto ou texto. Por fim, trabalhei na automação da criação de datasets, em especial, como a combinação eficiente de modelos de detecção e segmentação pode acelerar e simplificar o processo de anotação de conjuntos de dados, reduzindo o esforço humano. Os resultados dos experimentos podem ser encontrados no **Apêndice 6**.

Ao final dessas **Semanas** dedicadas à Visão Computacional, percebo o avanço em aprofundar meus conhecimentos na área. No entanto, reconheço que o aprendizado não se encerra com o término da décima **Semana** da Disciplina Residência em IA. A Visão Computacional é um campo em constante evolução, com novas descobertas, técnicas e aplicações emergindo regularmente. Pretendo continuar minha especialização, a fim de aplicar os conhecimentos adquiridos para resolver problemas relevantes.

APÊNDICE 1

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 19 de out. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Esta primeira entrega consiste na pesquisa do tema que será abordado no decorrer do projeto e definição dos objetivos que serão explorados.

Os requisitos para a entrega são:

- ☑ Classificação dos estudos segundo a metodologia da Conference on Computational Science and Computational Intelligence (CSCI):

Conforme solicitado, em relação aos termos que aparecem no Conference on Computational Science and Computational Intelligence (CSCI'23), esse trabalho se enquadra nos temas: Neural networks and applications, Emerging technologies; and Applications (including: computer vision, signal processing, Large language models, emerging applications).

- ☑ Definição dos objetivos e início da revisão de literatura.

O objetivo geral deste trabalho será explorar a área de Visão Computacional, com ênfase nas principais tarefas da área como classificação, localização e detecção de objetos em imagens. Este estudo tem como meta adquirir um profundo entendimento das técnicas, algoritmos e modelos envolvidos na Visão Computacional e aplicá-los em um contexto prático, a fim de contribuir para a resolução de desafios do mundo real. Adicionalmente, o objetivo é avaliar e aprimorar o desempenho de tais abordagens, considerando a evolução dos estados da arte na área, e explorar a Inteligência Artificial Generativa para gerar conteúdo visual e anotação dos dados. Ao final, pretende-se não apenas adquirir expertise em Visão Computacional, mas também apresentar resultados e conclusões que possam beneficiar aplicações práticas em diferentes domínios.

Ademais, foi iniciada a busca por materiais relacionados às tarefas e aos principais métodos utilizados, bem como os estados da arte. O documento encontra-se no seguinte link: Entrega 19/10 - MARCOS VINICIUS SATIL MEDEIROS

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega teremos os seguinte objetivos:

- Criação de um repositório (git, ou similar) para colocar a documentação e os códigos gerados.
- Continuar a busca por artigos para o repositório.
- Estudo de tópicos clássicos de visão e tópicos mais avançados como os estados da arte para cada tarefa.
- Resumos e exemplificação dos trabalhos considerados mais relevantes.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

LUANA GUEDES BARROS MARTINS: Go!

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 26 de out. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para esta entrega, conforme o planejamento, temos os seguinte objetivos:

- ☑ Criação de um repositório (git, ou similar) para colocar a documentação e os códigos gerados.

O repositório foi criado no seguinte link: https://github.com/marcosvinism/Residencia-em-IA

- ☑ Continuar a busca por artigos para o repositório.
- ☑ Estudo de tópicos clássicos de visão e tópicos mais avançados como os estados da arte para cada tarefa.
- ☑ Resumos e exemplificação dos trabalhos considerados mais relevantes.

Os objetivos listados foram incluídos em um documento no seguinte link: Entrega 26/10

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega teremos os seguinte objetivos:

- Continuar a pesquisa de artigos para inclusão no repositório.
- Investigar técnicas de pré-processamento, abrangendo a limpeza, redimensionamento e anotação dos dados.
- Explorar técnicas para o uso eficiente de um conjunto menor de dados anotados, como o aprendizado ativo (active learning) e o aprendizado auto-supervisionado (self-supervised learning).
- Identificar conjuntos de dados relevantes, incluindo aqueles que contenham descrições textuais.
- Definir o escopo da parte prática do projeto.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

LUANA GUEDES BARROS MARTINS: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

OBJETIVOS DO PROJETO

Objetivo geral:

O objetivo geral deste trabalho será explorar a área de Visão Computacional, com ênfase nas principais tarefas da área como classificação, localização e detecção de objetos em imagens. Este estudo tem como meta adquirir um profundo entendimento das técnicas, algoritmos e modelos envolvidos na Visão Computacional e aplicá-los em um contexto prático, a fim de contribuir para a resolução de desafios do mundo real. Adicionalmente, o objetivo é avaliar e aprimorar o desempenho de tais abordagens, considerando a evolução dos estados da arte na área, e explorar a Inteligência Artificial Generativa para gerar conteúdo visual e anotação dos dados. Ao final, pretende-se não apenas adquirir expertise em Visão Computacional, mas também apresentar resultados e conclusões que possam beneficiar aplicações práticas em diferentes domínios.

Objetivos Específicos:

Temos como objetivos específicos:

- Realizar uma pesquisa inicial sobre Visão Computacional, Redes Neurais.
- Identificar as principais aplicações de Visão Computacional.
- Selecionar conjuntos de dados adequados para os projetos desenvolvidos.
- Realizar a coleta e o pré-processamento dos dados, incluindo limpeza, redimensionamento e anotação, quando necessário.
- Configurar e treinar modelos.
- Comparar o desempenho de diferentes modelos.
- Realizar avaliações abrangentes do desempenho do sistema em tarefas de detecção, classificação de objetos e segmentação.
- Documentar todos os experimentos, resultados e conclusões em um relatório.
- Preparar uma apresentação para a defesa do TCC.

Metodologia:

Para alcançar os objetivos propostos, seguiremos as seguintes etapas:

(Semanas 1-2) Revisão da Literatura:

Objetivo: Realizar uma revisão detalhada da literatura sobre Visão Computacional, Redes Neurais Convolucionais (CNNs), e suas aplicações conjuntas, bem como os estados da arte em cada uma dessas aplicações.

Atividades:

- ☐ Estudar e compreender os fundamentos teóricos da Visão Computacional.

(Semanas 3-4) Estudo sobre os dados:

Objetivo: Estudar, adquirir e preparar dados para alimentar modelos de Visão Computacional.

Atividades:

- ☐ Pesquisa de conjunto de dados relacionados ao projeto.
- ☐ Pesquisa de técnicas sobre o uso de uma quantidade menor de dados anotados, como activate learning e self-supervised.
- ☐ Pré-processamento, limpeza, redimensionamento e anotação dos dados, se necessário.

(Semanas 5-10) Estudo dos modelos:

Objetivo: Estudar, configurar, treinar e testar modelos de Visão Computacional: classificação, localização e detecção de objetos e segmentação.

Atividades:

- ☐ Treinamento de modelos.
- ☐ Avaliação do desempenho dos modelos.
- ☐ Realizar avaliações abrangentes em tarefas de detecção, classificação e segmentação nas imagens.
- ☐ Documentar todos os experimentos, resultados, desafios e conclusões em um relatório detalhado.
- ☐ Preparar uma apresentação para a defesa do TCC.

Referências:

As referências de papers para as tarefas de visão foram retiradas de https://openaccess.thecvf.com/ICCV2023?day=all e serão exploradas ativamente conforme forem necessárias.

Pan, Jun. (2023). Recent Deep Neural Networks for Object Detection. Highlights in Science, Engineering and Technology. 31. 268-273. 10.54097/hset.v31i.5153.

ZOU, Zhengxia et al. Object detection in 20 years: A survey. arXiv preprint arXiv:1905.05055, 2019.

ZHAO, Zhong-Qiu et al. Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems, v. 30, n. 11, p. 3212-3232, 2019.

GPT-4V(ision) System Card https://cdn.openai.com/papers/GPTV_System_Card.pdf

Livros de referência:

Computer Vision: Algorithm e Applications, 2nd Edition, Richard Szeliski. https://szeliski.org/Book/

Computer & Machine Vision – Theory, Algorithms, Practicalities. Fourth Edition, 2012. Elsevier Inc. E. Roy Davies.

Vídeos interessantes:

TedTalks: Fei-Fei Li | How we're teaching computers to understand pictures: https://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures?language=en&subtitle=pt-br

GPT-4 Vision: 10 Amazing Use Cases - This is HUGE!: https://www.youtube.com/watch?v=Yq2VOWDFpNA

GPT-4 Vision da OpenAI: A Revolução da IA que "Vê" Imagens!: https://www.youtube.com/watch?v=R24Y1T6Mxbc

INTRODUÇÃO

A Visão Computacional, como campo de pesquisa e aplicação, abarca uma ampla gama de tarefas, cada qual com sua própria história evolutiva de métodos e técnicas. Este domínio multidisciplinar permite que máquinas interpretem e compreendam informações visuais, estendendo nossa compreensão do mundo digital e físico. Neste contexto, forneceremos uma visão panorâmica das principais tarefas, desde os métodos clássicos que lançaram as bases até os estados da arte que impulsionam a evolução constante desta área. A seguir, abordaremos estas tarefas em detalhes, respondendo algumas dúvidas iniciais, oferecendo uma perspectiva abrangente e um resumo de trabalhos relevantes.

Começando pela duvidas:

O que é estado da arte (SOTA) em Visão Computacional ?

O estado da arte são os melhores modelos / arquiteturas que podemos empregar para uma tarefa específica. Estes modelos podem ganhar o rótulo SOTA (State of the Art) com base em métricas relevantes para a tarefa em que será avaliada. No entanto, em muitos domínios, existe uma compensação entre essas métricas. Em outras palavras, podemos ter um modelo muito rápido, mas sua precisão é insuficiente. Por outro lado, existem modelos com métricas de precisão impressionantes que não possuem a latência ou a taxa de transferência necessárias em várias tarefas, como classificação de imagens e detecção de objetos. Nestes domínios, será considerado SOTA se proporcionar um compromisso ideal entre as métricas relevantes.

As métricas mais conhecidas e mais utilizadas para comparar e avaliar esses modelos são acurácia, precisão, recall, F1-score, IoU e mAP. Um modelo será declarado de última geração se oferecer a melhor compensação entre essas métricas e dado o desempenho em métricas adicionais de interesse, como FLOPS, latência, taxa de transferência, etc.

PRINCIPAIS TAREFAS DA ÁREA DE VISÃO COMPUTACIONAL

CLASSIFICAÇÃO DE IMAGENS

A classificação de imagens envolve o treinamento de algoritmos para categorizar automaticamente as imagens em grupos ou “classes” predefinidos. Por exemplo, quando apresentadas a várias imagens de animais, o algoritmo deve ser capaz de categorizá-los nas respectivas classes. Neste processo, um modelo recebe uma imagem como entrada e tem a tarefa de determinar com precisão sua classe correspondente.

- **Métodos Clássicos para classificação:**

  Os métodos clássicos para classificação de imagens envolviam o uso de descritores de características, como Histogramas de Gradientes Orientados (HOG) e Descritores de Cor. Essas características eram então usadas em classificadores tradicionais, como Máquinas de Vetores de Suporte (SVM), regressão logística ou até mesmo redes neurais artificiais.

- **SOTA para classificação:**

  Atualmente, os principais métodos para classificação de imagens são os baseados em redes neurais convolucionais (CNNs) e em transformers. Esses modelos estão na vanguarda, apresentando recursos exclusivos que otimizam o desempenho em diversas tarefas. Lembrando que existem benchmarkings para datasets específicos e que há uma variação entre os melhores modelos para determinadas tarefas. A referência principal utilizada para avaliação e comparação de desempenho é o conjunto de dados ImageNet. Para acessar os trabalhos acadêmicos e códigos relacionados, é só acessar o seguinte link: https://paperswithcode.com/sota/image-classification-on-imagenet

Resumo das arquiteturas básicas utilizadas em Visão Computacional

Convolutional Neural Networks (CNNs) e Vision Transformers (ViTs) são arquiteturas comumente usadas em visão computacional, cada uma usando um método exclusivo de processamento de dados.

As CNNs têm sido a base para tarefas de processamento de imagens. Eles empregam camadas convolucionais para digitalizar imagens sistematicamente, detectando

inicialmente recursos básicos como bordas e identificando progressivamente padrões mais complexos mais profundos na rede. Devido à sua abordagem estruturada, as CNNs têm se destacado em tarefas como classificação de imagens e detecção de objetos.

Já o ViTs introduz uma nova abordagem para análise de imagens. Originados de arquiteturas de transformers inicialmente desenvolvidas para processamento de linguagem natural, os ViTs segmentam imagens em fragmentos de tamanho fixo e as processam como sequências, não como grades. Seus mecanismos de atenção inerentes permitem-lhes discernir as relações entre vários patches, capturando o contexto e oferecendo uma interpretação distinta das CNNs. Esta perspectiva inovadora dos ViTs enriqueceu o domínio da visão computacional, desencadeando extensas pesquisas sobre sinergias entre eles e as CNNs.

Como funcionam as CNNs na classificação de imagens ?

Para compreender o funcionamento das Redes Neurais Convolucionais (CNNs) na tarefa de classificação de imagens, pode-se imaginar o processo como a análise sistemática de uma imagem por meio de uma série de pequenas janelas sobrepostas, conhecidas como filtros. Esses filtros deslizam pela imagem, examinando áreas específicas em busca de padrões visuais, que podem variar desde simples como linhas e curvas até estruturas mais complexas, como características faciais ou texturas.

Conforme a imagem passa por várias camadas da rede, esses padrões visuais tornam-se progressivamente mais elaborados, evoluindo de características simples, como bordas, para características mais complexas, como objetos e partes de objetos.

Uma vez que a rede tenha identificado esses padrões, camadas de pooling são introduzidas para reduzir a resolução da imagem. Essencialmente, o pooling preserva as características mais significativas, descartando detalhes redundantes. Isso simplifica a representação da imagem e reduz a carga computacional.

Após a etapa de convolução e pooling, os dados resultantes são nivelados em uma única matriz unidimensional e passados para camadas totalmente conectadas. Estas camadas funcionam como "tomadores de decisão" da rede, pesando as características identificadas para determinar a classe à qual a imagem pertence.

Em resumo, as CNNs examinam imagens de forma sistemática, identificam padrões hierárquicos por meio de convoluções,destilam informações essenciais e empregam camadas densas para tirar conclusões sobre o conteúdo da imagem.

Como funcionam os ViTs na classificação de imagens ?

Para compreender o funcionamento dos Vision Transformers (ViTs), podemos visualizar o processo da seguinte maneira: imagine uma imagem sendo dividida em vários pequenos quadrados, semelhantes a um mosaico ou uma grade. Cada um desses quadrados é convertido em uma representação numérica que captura todas as informações de cor (valores RGB) contidas nele. Essas sequências numéricas, conhecidas como "patch embeddings", lembram o modo como os transformers representam palavras em uma frase.
No entanto, a arquitetura original do transformer não mantém a ordem sequencial desses embeddings de maneira inerente. Para solucionar esse problema, os ViTs incorporam uma representação posicional em cada um desses embeddings de patch, assegurando que o modelo mantenha a consciência espacial da origem de cada segmento na imagem.

Uma vez que essas informações foram enriquecidas, os embeddings dos patches são processados pelo transformer. Através do uso de mecanismos de atenção e múltiplas camadas, o transformer analisa minuciosamente o conteúdo da imagem, o que eventualmente leva à sua classificação em uma categoria específica.

Em resumo, os Vision Transformers (ViTs) desmontam uma imagem em representações numéricas, acrescentam contexto espacial a elas e fazem uso das capacidades do transformer para analisar e classificar os dados visuais.

Resumo de trabalhos relevantes na tarefa de classificação de imagens:

ConvNeXt

ConvNeXt rejuvenesce as CNNs tradicionais, ou “ConvNets”, conforme destacado em “A ConvNet for the 2020s“ por Zhuang Liu et al. Este modelo inovador renova as arquiteturas ConvNet padrão para competir com a capacidade dos Vision Transformers (ViTs) em uma variedade de tarefas de visão computacional. Notavelmente, o ConvNeXt é construído exclusivamente a partir de módulos ConvNet clássicos, mesclando simplicidade e

eficiência sem comprometer o desempenho. Ele atinge uma precisão impressionante de 87,8% top-1 no ImageNet e supera os Swin Transformers em tarefas como detecção de objetos COCO e segmentação semântica ADE20K.

Sua evolução arquitetônica começa a partir de um ResNet básico e integra elementos que lembram um Vision Transformer, incorporando princípios de design do ResNeXt, estruturas de gargalo invertidas e empregando tamanhos de kernel ampliados. Essas modificações preenchem a lacuna de desempenho entre ConvNets tradicionais e Transformers de última geração. Como resultado, a ConvNeXt apresenta métricas de desempenho comparáveis ou até mesmo superiores às dos Transformers, mantendo-se fiel à sua herança ConvNet.

O que diferencia o ConvNeXt é sua capacidade de alcançar um desempenho estelar sem os mecanismos de atenção característicos dos Transformers. Em vez disso, ele navega de forma inteligente pelos caminhos de design que sustentam o sucesso do Transformer, aproveitando meios puramente convolucionais. Além de sua notável precisão no ImageNet, o ConvNeXt se destaca em diversas tarefas de visão, superando até mesmo modelos estabelecidos como Swin Transformers. Tudo isso, preservando a simplicidade e a eficiência computacional características dos ConvNets.

EfficientFormer e EfficientFormer v2

EfficientFormer e seu sucessor, EfficientFormer v2, representam avanços em redes neurais, mostrando o potencial dos Vision Transformers (ViTs) otimizados para dispositivos móveis. Esses modelos combinam habilmente a robustez dos ViTs com a eficiência essencial necessária para a implantação de dispositivos móveis.

O EfficientFormer original adota um design exclusivo com dimensões consistentes, eliminando operações frequentes de remodelagem, o que reduz os custos computacionais. O EfficientFormer v2 aprimora ainda mais essa abordagem, aproveitando uma super-rede e uma estratégia de pesquisa conjunta refinada para atingir um equilíbrio ideal entre latência e eficiência de parâmetros. Os modelos abrangem vários tamanhos: o EfficientFormer varia de L1 a L7, enquanto o EfficientFormer v2 oferece variantes como S0 e S1, cada uma meticulosamente projetada para uma combinação específica de precisão e eficiência.

O que diferencia o EfficientFormer é seu design inovador que prioriza a velocidade do dispositivo, enquanto o EfficientFormerV2 otimiza a arquitetura para maior precisão sem latência adicional. Isso é feito unificando a Rede Feed Forward (FFN) e inovando o mecanismo Multi Head Self Attention (MHSA).

Em termos de desempenho, ambas as versões alcançam um equilíbrio admirável entre velocidade e precisão, crucial para a implantação móvel. O modelo L1 do EfficientFormer possui uma precisão top-1 de 79,2% no ImageNet-1K com um tempo de inferência rápido de 1,6 ms em um iPhone 12. O modelo S0 do EfficientFormerV2 supera o MobileNetV2, alcançando uma precisão top-1 3,5% maior no ImageNet-1K com latência comparável e parâmetros. Esses avanços ressaltam a viabilidade de integração de transformadores de visão eficientes e de alto desempenho em plataformas móveis sem comprometer o desempenho.

Mobile-Former

Mobile-Former, apresentado em “Mobile-Former: Bridging MobileNet and Transformer”, combina de forma inovadora os pontos fortes dos modelos MobileNet e Transformer para produzir uma arquitetura que se destaca em eficiência e desempenho. Ao utilizar o MobileNet para processamento local adequado e aproveitar um Transformer para interação global diferenciada, essa arquitetura se beneficia de uma ponte bidirecional exclusiva que garante uma fusão perfeita de recursos locais e globais. O que diferencia o aspecto Transformer deste design é sua abordagem minimalista, usando um número muito limitado de tokens, o que reduz drasticamente as despesas gerais computacionais.

O ponto central para a inovação da Mobile-Former é a sua capacidade de aproveitar eficazmente os recursos locais e globais. Enquanto o MobileNet se concentra na extração eficiente de recursos locais, o Transformer captura interações mais amplas. A sua ligação através de uma ponte bidirecional facilita a troca e o enriquecimento destas funcionalidades, reforçadas ainda por um mecanismo leve de atenção cruzada.

As métricas de desempenho do Mobile-Former são impressionantes. No domínio da classificação de imagens, ele atinge uma precisão top-1 de 77,9% no conjunto de dados ImageNet em 294 milhões de FLOPs, superando os benchmarks definidos pelo MobileNetV3. Quando avaliado em tarefas de detecção de objetos, o Mobile-Former

demonstra capacidade superior, superando o MobileNetV3 na estrutura RetinaNet por uma margem de precisão média (AP) de 8,6. Notavelmente, quando incorporado à estrutura do DETR como backbone, codificador e decodificador, ele não apenas supera o DETR com uma vantagem de 1,1 AP, mas também atinge uma redução de 52% no custo computacional e uma redução de 36% no número de parâmetros.

DETECÇÃO DE OBJETOS

A detecção de objetos é outra tarefa essencial em visão computacional, tendo por objetivo identificar e localizar objetos nas imagens. Indo além do mero reconhecimento de objetos, e assim oferecendo uma perspectiva dupla ao identificar tanto o tipo de objeto (“o quê”) como a sua localização específica dentro da imagem (“onde”), muitas vezes representada por bounding boxes. Duas metodologias predominantes são centrais para tarefas de detecção de objetos: detecção de disparo único (SSD) e redes de dois estágios.

- Detecção de disparo único (SSD): Projetado com a eficiência em mente, o SSD executa a detecção de objetos em apenas uma passagem direta pela rede. Em vez de tratar a localização e classificação de objetos como tarefas separadas, o SSD as íntegra, gerando um processo de detecção rápido e simplificado. Aqui, as imagens são divididas em grades, com bounding boxes e com as probabilidades de classe previstas simultaneamente. Deixando a detecção de objetos mais rápida e mais suave com os recursos computacionais.

- Redes em Dois Estágios: Um pouco mais complexa, esta abordagem é dividida em duas etapas principais. Inicialmente, uma Rede de Proposta de Região (RPN) examina a imagem para sugerir possíveis bounding boxes, destacando áreas que provavelmente contêm objetos. Depois disso, uma segunda CNN separada refina essas propostas e assume a tarefa de classificação de objetos. Embora este método tende a ser mais preciso, sua natureza bifurcada o torna mais intensivo computacionalmente em comparação ao SSD.

A escolha entre SSD e redes de dois estágios geralmente se resume às necessidades específicas do projeto. Embora os SSDs sejam preferidos em cenários onde o processamento em tempo real e a baixa latência são fundamentais, as redes de dois estágios são procuradas quando a precisão tem precedência. No entanto, os avanços no SSD permitiram que ele fornecesse métricas de desempenho competitivas enquanto consumia menos recursos computacionais, posicionando-o como uma escolha preferencial para muitas aplicações com recursos limitados.

- Métodos Clássicos: O método clássico para detecção de objetos é o uso de recursos baseados em bordas, como Cascades de Haar e Métodos de Viola-Jones. Eles são frequentemente combinados com classificadores, como SVMs, para localizar objetos em imagens.

- SOTA para detecção de objetos
  Entre os modelos de detecção de objetos, YOLOv8 e YOLO-NAS se destacam como estado da arte atualmente por conta do seu desempenho em tempo real. Cada modelo traz seu próprio conjunto de avanços arquitetônicos e melhorias de desempenho. A seguir, oferecemos um exame conciso desses modelos de destaque, destacando sua arquitetura, principais recursos e benchmarks de desempenho. O principal benchmarking é do conjunto de dados COCO, e é possível acessar os artigos e códigos de referência em:
  https://paperswithcode.com/sota/real-time-object-detection-on-coco
  https://paperswithcode.com/sota/object-detection-on-coco

Resumo de trabalhos recentes na tarefa de detecção de objeto:

YOLOv8

Introduzido pela Ultralytics em janeiro de 2023, a YOLOv8 se destaca como um modelo de detecção de objetos versátil e escalável. Em termos de arquitetura, o YOLOv8 utiliza um backbone que lembra o YOLOv5, mas é enriquecido pelo módulo C2f. Este

módulo aumenta a precisão da detecção ao combinar recursos de alto nível com insights contextuais. A abordagem livre de âncoras do modelo, combinada com um cabeçote desacoplado, otimiza o tratamento de tarefas de objetividade, classificação e regressão. Com a integração de funções de perda CIoU e DFL adaptadas para detecção de objetos menores, bem como entropia cruzada binária para perda de classificação, o YOLOv8 reflete um avanço notável nas capacidades de detecção de objetos.

Está disponível em cinco variantes: YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l e YOLOv8x, projetadas para atender a uma variedade de demandas computacionais e cenários de aplicação. Demonstrando sua habilidade, YOLOv8x (maior modelo) alcançou um notável AP de 53,9% no conjunto de dados MS COCO com um tamanho de imagem de 640 pixels, operando a rápidos 280 FPS em um NVIDIA A100 com TensorRT. Isso enfatiza a proficiência do YOLOv8 em fornecer precisão e velocidade.

YOLO-NAS

Lançado em maio de 2023 pela Deci, YOLO-NAS é uma arquitetura pioneira no domínio da detecção de objetos, estabelecendo padrões incomparáveis no equilíbrio de precisão e latência. Ele incorpora blocos QSP e QCI, que combinam os benefícios da re-parametrização e da quantização de 8 bits, garantindo degradação mínima da precisão durante a quantização pós-treinamento. A tecnologia proprietária de pesquisa de arquitetura neural (NAS) da Deci, AutoNAC, foi fundamental na geração da arquitetura YOLO-NAS. Ele determinou os tamanhos ideais, tipos de blocos, número de blocos e contagens de canais em cada estágio.

Projetado com a produção em mente, o YOLO-NAS é adaptado para integração perfeita com mecanismos de inferência de ponta, como NVIDIA® TensorRT™, e suporta quantização INT8 para desempenho de tempo de execução incomparável. Sua adaptabilidade o posiciona como uma escolha ideal para aplicações do mundo real que exigem baixa latência e processamento robusto, como veículos autônomos, robótica e análise de vídeo.

A arquitetura inova ainda mais ao empregar técnicas de última geração, como mecanismos de atenção, blocos com reconhecimento de quantização e reparametrização durante a inferência. Esses elementos capacitam o YOLO-NAS a detectar objetos com

habilidade, independentemente de seu tamanho ou complexidade, redefinindo a excelência no cenário de detecção de objetos e sua aplicabilidade em vários setores.

SEGMENTAÇÃO

Segmentação é a tarefa da visão computacional de identificar e localizar um objeto (ou objetos) em uma imagem, dividindo-o em regiões de formato irregular, chamadas segmentos, que correspondem às formas de diferentes objetos ou partes da cena.
É semelhante à detecção de objetos, mas enquanto a detecção é satisfeita com bounding boxes ao redor dos objetos, a segmentação traça os contornos dos objetos no nível do pixel. Existem muitas tarefas de segmentação, dependendo da natureza de sua entrada e das informações que desejamos obter delas. Com base em sua funcionalidade, os três principais tipos de segmentação são semântica, por instância e panóptica.

Segmentação semântica

A segmentação semântica está interessada apenas nas classes de objetos, não em objetos individuais. Cada segmento corresponde a uma classe que cobre os objetos dessa classe, mesmo que estejam distantes uns dos outros na cena. Ele atribui um rótulo de classe a cada pixel e usa o mesmo rótulo de classe para os pixels de qualquer objeto dessa classe. Cada segmento é gerado como uma máscara de sobreposição do tamanho de uma imagem cobrindo todos os pixels daquela classe.

Segmentação por instância

A segmentação por instâncias ou objetos não apenas identifica as classes, mas também diferencia cada objeto dentro de uma classe. Cada segmento corresponde a uma instância de objeto individual. Ele atribui um rótulo de classe e um identificador de objeto exclusivo para cada pixel. Inúmeros elementos de fundo, como o céu ou o solo, geralmente são ignorados.

Segmentação panóptica = semântica + instância

A segmentação panóptica combina apenas segmentação semântica e de instância. Assim como a segmentação de instâncias, ela diferencia cada objeto dentro de uma classe. Assim como a segmentação semântica, ela também rotula os elementos de fundo.

Outros métodos de segmentação

Com base na natureza das suas entradas, outros tipos de tarefas de segmentação são:

- Segmentação de referência ou guiada por texto: utiliza uma descrição de texto como entrada adicional e segmenta os objetos ou plano de fundo que correspondem a essa descrição. Se a descrição vier de um conjunto fixo aprendido durante o treinamento, isso é chamado de segmentação de conjunto fechado. Mas se for qualquer texto arbitrário, é chamado de conjunto aberto ou vocabulário aberto.
- Segmentação zero-shot: Normalmente, apenas as classes vistas durante o treinamento são segmentadas. Mas o objetivo da segmentação zero-shot é segmentar até mesmo novas classes inéditas sem retreinar um modelo. Ele faz isso relacionando as classes invisíveis com as classes conhecidas.
- Segmentação one-shot: A segmentação one-shot é um tipo de segmentação zero-shot em que cada classe não vista é fornecida por meio de uma imagem de exemplo. Relaciona a imagem de exemplo a classes conhecidas com base em semelhanças visuais.
- Segmentação de vídeo: a segmentação normalmente funciona em imagens. Mas quando a entrada é um vídeo, isso se chama segmentação de vídeo. Como não pensamos em um objeto em um quadro como diferente do mesmo objeto no quadro seguinte, a segmentação de vídeo deve não apenas segmentar objetos, mas também rastreá-los com os mesmos identificadores entre os quadros.

SOTA para segmentação:

A inovação recente mais impactante em visão computacional é o uso de linguagem natural para orientar tarefas de visão, iniciada pelo CLIP e DALL-E da OpenAI. A

segmentação também se beneficiou na forma de segmentação de referência, onde a linguagem natural é usada para orientar a segmentação. Alguns modelos de linguagem-imagem notáveis para segmentação são:

CLIPSeg

CLIPSeg é um modelo de segmentação semântica linguagem-imagem que segmenta imagens com base em prompts de texto ou imagens de exemplo. Isso significa que ele é capaz de segmentação única e de referência. Além disso, o uso do poderoso modelo de linguagem CLIP também o equipa para segmentação zero-shot. É um modelo simples e leve, o que o torna um bom exemplo da abordagem linguagem-imagem.

Segment Anything Model (SAM)

O SAM (Segment Anything Model) é um modelo de segmentação de imagens promptable desenvolvido pelo time FAIR da Meta AI. Ele é projetado para ser usado em tarefas de segmentação baseadas em prompt e pode transferir zero-shot para novas distribuições e tarefas de imagem. O SAM foi treinado em um conjunto de dados de mais de 1 bilhão de máscaras em 11 milhões de imagens, e sua capacidade de desempenho zero-shot é impressionante, muitas vezes competitiva ou superior aos resultados totalmente supervisionados anteriores. O SAM está disponível sob uma licença aberta permissiva (Apache 2.0) e pode ser encontrado no repositório do GitHub do Facebook Research. O modelo é uma ferramenta poderosa para a segmentação de imagens e pode ser aplicado em diversas áreas, como detecção de objetos e detecção de bordas.

Referências:

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, Saining Xie. “A ConvNet for the 2020s” arXiv:2201.03545 [cs.CV]. https://arxiv.org/abs/2201.03545

Yanyu Li, Geng Yuan, Yang Wen, Ju Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, Jian Ren (2022). "EfficientFormer: Vision Transformers at MobileNet Speed" arXiv:2206.01191 [cs.CV]. https://arxiv.org/abs/2206.01191

Mingxing Tan, Quoc V. Le (2021). "EfficientNetV2: Smaller Models and Faster Training". arXiv:2104.00298 [cs.CV]. https://arxiv.org/abs/2104.00298

Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, Zicheng Liu (2022). "Mobile-Former: Bridging MobileNet and Transformer". arXiv:2108.05895 [cs.CV] https://arxiv.org/abs/2108.05895

Timo Lüddecke, Alexander S. Ecker (2021). "Image Segmentation Using Text and Image Prompts." arXiv:2112.10003 [cs.CV]. https://arxiv.org/abs/2112.10003

Adam Botach, Evgenii Zheltonozhskii, Chaim Baskin (2021). "End-to-End Referring Video Object Segmentation with Multimodal Transformers." arXiv:2111.14821 [cs.CV]. https://arxiv.org/abs/2111.14821v2

Renjie Pi, Jiahui Gao, Shizhe Diao, Rui Pan, Hanze Dong, Jipeng Zhang, Lewei Yao, Jianhua Han, Hang Xu, Lingpeng Kong, Tong Zhang (2023). "DetGPT: Detect What You Need via Reasoning" arXiv:2305.14167 [cs.CV]. https://arxiv.org/abs/2305.14167

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, Ross Girshick (2023). "Segment Anything". arXiv:2304.02643 [cs.CV] https://arxiv.org/abs/2304.02643

APÊNDICE 2

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 9 de nov. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para esta entrega, conforme o planejamento, temos os seguinte objetivos:

- ☑ Continuar a pesquisa de artigos para inclusão no repositório.
- ☑ Investigar técnicas de pré-processamento, abrangendo a limpeza, redimensionamento e anotação dos dados.
- ☑ Explorar técnicas para o uso eficiente de um conjunto menor de dados anotados, como o aprendizado ativo (active learning) e o aprendizado auto-supervisionado (self-supervised learning).
- ☑ Identificar conjuntos de dados relevantes, incluindo aqueles que contenham descrições textuais.

Os objetivos listados foram incluídos em um documento no seguinte link: Entrega 09/11 - MARCOS VINICIUS SATIL MEDEIROS

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega teremos os seguinte objetivos:

- Terminar de estudar artigos listados.
- Aprofundar em artigos recentes de modelos de linguagem.
- Iniciar escopo da aplicação prática, visando definir a viabilidade.
- Levantamento das principais ferramentas/frameworks que podem ser utilizadas no pipeline.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

LUANA GUEDES BARROS MARTINS: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

PRÉ-PROCESSAMENTO DE DADOS

Nesta etapa, exploramos um componente essencial no pipeline de processamento de imagens em visão computacional: o pré-processamento. O pré-processamento de dados é uma das etapas iniciais, ocorrendo após a aquisição dos dados, mas que desempenha um papel de extrema importância na qualidade e eficácia dos algoritmos que serão utilizados posteriormente. Trata-se de uma fase onde as imagens são preparadas e aprimoradas, a fim de torná-las mais adequadas para as tarefas subsequentes de análise, classificação, detecção, rastreamento e reconhecimento de objetos, entre outras.

Limpeza de dados

A limpeza de dados representa uma etapa fundamental do processo e deve ser conduzida antes da anotação e do treinamento do modelo. A importância desta etapa reside na economia de tempo e recursos que a obtenção de dados limpos proporciona. De acordo com uma estimativa da IBM, publicada pela Harvard Business Review (HBR), o custo associado a dados impuros, duplicados e de baixa qualidade atinge a marca de 3,1 bilhões de dólares.

No contexto de conjuntos de dados baseados em imagens e vídeos, o trabalho de limpeza é ainda mais significativo e, consequentemente, ainda mais complicado. Essa etapa, que representa o primeiro passo no processo de pré-processamento, visa assegurar que os dados estejam limpos, permitindo que os anotadores se concentrem integralmente na anotação, sem a necessidade de gerenciar questões relacionadas à integridade dos dados.

Contudo, em conjuntos de dados de imagens, surgem vários desafios no processo de limpeza de dados. Esses desafios podem incluir a presença de imagens excessivamente claras ou escuras, duplicatas, arquivos corrompidos e outras anomalias. Tais imperfeições têm o potencial de impactar significativamente os resultados dos modelos. Além disso, questões como diferentes formatos de arquivos, incompatibilidades entre os formatos podem criar obstáculos adicionais.

A essência do treinamento de um modelo de visão computacional está na inserção da mais alta qualidade e quantidade de imagens anotadas. Isso se torna uma meta inatingível quando um percentual substancial de imagens em um conjunto de dados apresenta problemas de qualidade.

No que diz respeito ao processo de limpeza diretamente nas imagens, este envolve a identificação e correção de imperfeições nos dados. Existem técnicas comuns de limpeza de dados em visão computacional que merecem destaque como:

- Remoção de Ruído: Essa técnica é frequentemente necessária para eliminar informações indesejadas em uma imagem, como granulação ou distorção causada por condições de captura.
- Preenchimento de Valores Ausentes: Quando há regiões com informações ausentes nas imagens, como áreas não detectadas ou corrompidas, técnicas de preenchimento podem ser aplicadas para estimar ou interpolar os valores ausentes.
- Correção de Iluminação: A iluminação irregular ou inadequada pode impactar negativamente a qualidade das imagens. Técnicas de correção de iluminação, como a correção gamma e equalização de histograma, são empregadas para melhorar a exposição e o contraste das imagens, tornando-as mais adequadas para análise.
- Transformações nas imagens: Em alguns casos, as imagens podem sofrer distorções geométricas devido a ângulos de visão, inclinação ou outros fatores. Portanto, a aplicação de transformações geométricas, como rotação, escala e cisalhamento, pode ser necessária para corrigir essas distorções e alinhar as imagens adequadamente.

Redimensionamento de Dados

O redimensionamento de dados é uma outra etapa comum, em que as imagens são ajustadas para tamanhos específicos ou resoluções, a fim de atender aos requisitos de um determinado algoritmo ou aplicação. Existem diversas técnicas de redimensionamento que podem ser utilizadas como:

- Normalização: Normalizar as intensidades de pixel para uma faixa específica (por exemplo, 0-255) é comum para garantir consistência nos dados de entrada.
- Redimensionamento linear: Esta técnica altera o tamanho da imagem de acordo com uma escala específica, mantendo a relação de aspecto.

- Redimensionamento não linear: Em certos casos, é preferível aplicar uma transformação não linear para ajustar o tamanho da imagem de maneira adaptativa.

Anotação de dados

A anotação de dados é uma parte crucial em tarefas de visão computacional que envolve a marcação de objetos de interesse em imagens. Ela é essencial na preparação dos dados para treinamento de modelos. Nesta etapa, as imagens são enriquecidas com informações relevantes que serão utilizadas para o desenvolvimento de algoritmos capazes de entender e interagir com o ambiente visual.

De uma forma geral, a anotação de imagem é um subconjunto de rotulagem de dados em que envolve humanos no back-end, marcando incansavelmente imagens com informações de metadados e atributos que ajudarão as máquinas a identificar melhor os objetos. As anotações em imagens podem ser feitas na seguinte forma:

Dados de imagem:
- Imagens 2-D
- Imagens 3-D

Tipos de anotação:
- Classificação de imagens
- Detecção de Objetos
- Segmentação de imagem
- Rastreamento de Objeto
- Transcrição da imagem

Técnicas de Anotação:
- Caixa delimitadora
- Polyline
- Polygon
- Anotação de ponto de referência

Além das técnicas tradicionais de anotação de dados, é importante destacar que a constante evolução tem dado origem a abordagens inovadoras que visam otimizar significativamente o processo de anotação. Essas abordagens estão transformando a maneira como lidamos com dados em projetos de visão computacional, proporcionando ganhos notáveis em eficiência e qualidade como:

Transfer Learning:

Técnicas de transferência de conhecimento podem ser aplicadas para otimizar o processo de anotação. Isso envolve a alavancagem de modelos pré-treinados em tarefas relacionadas. Em vez de iniciar do zero, um modelo pré-treinado pode ser ajustado para a tarefa de interesse, transferindo as anotações de um conjunto de dados relacionado. Isso economiza tempo e recursos, uma vez que parte do trabalho de anotação já foi realizado em outros contextos.

Self-Supervised Learning:

Essa técnica envolve treinar modelos em tarefas auxiliares, nas quais os rótulos são gerados automaticamente a partir dos próprios dados, sem a necessidade de anotação manual. Posteriormente, esses modelos pré-treinados são ajustados para a tarefa principal. Por exemplo, um modelo pode ser treinado para reconstruir imagens a partir de partes cortadas, aprendendo representações úteis no processo. Isso reduz a dependência de anotações manuais e amplia o escopo de aplicação do aprendizado supervisionado.

Active learning:

Técnicas de aprendizado ativo oferecem uma abordagem inteligente para a anotação de dados. Em vez de anotar aleatoriamente todas as imagens, algoritmos de aprendizado ativo selecionam amostras estrategicamente para serem anotadas. Isso é feito com base na incerteza do modelo em relação a essas amostras, economizando tempo e recursos ao priorizar as imagens que mais beneficiarão o treinamento do modelo. O aprendizado ativo é especialmente útil em situações em que o conjunto de dados é muito grande, e a anotação manual de todas as imagens é inviável.

Plataformas de Anotação

Para a anotação de dados em projetos de visão computacional, várias plataformas e ferramentas estão disponíveis, ajudando a automatizar, gerenciar e acelerar o processo. Entre as principais plataformas de anotação de imagens podemos listar:

Referências:

Active learning:

Chenhongyi Yang, Lichao Huang, Elliot J. Crowley (2022). Plug and Play Active Learning for Object Detection. arXiv:2211.11612 [cs.CV] *https://doi.org/10.48550/arXiv.2211.11612*

Feixiang Tan & Guansheng Zheng (2023) Active learning for deep object detection by fully exploiting unlabeled data, Connection Science, 35:1, *https://doi.org/10.1080/09540091.2023.2195596*

Yu, W., Zhu, S., Yang, T., & Chen, C. (2022). Consistency-based Active Learning for Object Detection (arXiv:2103.10374). arXiv. http://arxiv.org/abs/2103.10374.

Self-Supervised Learning:

Gabriel Huang, Issam Laradji, David Vazquez, Simon Lacoste-Julien, Pau Rodriguez (2022). A Survey of Self-Supervised and Few-Shot Object Detection. (arXiv:2110.14711) [cs.CV] https://doi.org/10.48550/arXiv.2110.14711

Yi-Syuan Chen, Yun-Zhu Song, Cheng Yu Yeo, Bei Liu, Jianlong Fu, Hong-Han Shuai; SINC: Self-Supervised In-Context Learning for Vision-Language Tasks. Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2023, pp. 15430-15442.

APÊNDICE 3

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 16 de nov. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para esta entrega, conforme o planejamento, temos os seguinte objetivos:

- [x] Terminar de estudar artigos listados.
- [x] Aprofundar em artigos recentes de modelos de linguagem para visão.
- [x] Iniciar escopo da aplicação prática, visando definir a viabilidade.
- [ ] Levantamento das principais ferramentas/frameworks que podem ser utilizadas no pipeline.

Os objetivos listados foram incluídos em um documento no seguinte link: Entrega 16/11 - MARCOS VINICIUS SATIL MEDEIROS

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega teremos os seguinte objetivos:

- Terminar objetivos anteriores que ficaram parcialmente completos, como levantamento das ferramentas/frameworks.
- Definição de datasets a serem utilizados.
- Pesquisa qualitativa e exploratória, visando proporcionar maior familiaridade com os métodos que serão utilizados.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

LUANA GUEDES BARROS MARTINS: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 23 de nov. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para esta entrega, conforme o planejamento, temos os seguinte objetivos:

- ☑ Terminar objetivos anteriores que ficaram parcialmente completos, como levantamento das ferramentas/frameworks.
- ☑ Definição de datasets a serem utilizados.
- ☑ Pesquisa exploratória, visando proporcionar maior familiaridade com os métodos que serão utilizados.

Os objetivos listados foram incluídos em um documento no seguinte link: Entrega 23/11- MARCOS VINICIUS SATIL MEDEIROS

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega teremos os seguinte objetivos:

- Início da implementação dos códigos.
- Pré-processamento dos dados.
- Revisão de artigos.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

LUANA GUEDES BARROS MARTINS: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

PESQUISA EXPLORATÓRIA

Durante a semana, realizei diversas pesquisas sobre as aplicações da visão computacional na agricultura, com o objetivo de identificar uma ideia de projeto que atendesse aos requisitos propostos. Após uma análise aprofundada, optei por desenvolver uma aplicação que abordasse todo o pipeline de dados, concentrando-se na classificação e no controle de qualidade de culturas agrícolas.

A ideia surgiu a partir da leitura do paper *"Foundation Models in Smart Agriculture: Basics, Opportunities, and Challenges"*, onde é demonstrando os fundamentos básicos e oportunidades da agricultura inteligente. Entretanto este artigo é focado em modelos básicos (LLMs) de diversos domínios como texto, imagem e multimodal, e em nossa aplicação iremos fazer usos também de outras abordagens mais focadas a áreas específicas de visão.

Aplicação prática: Sistemas de visão computacional para classificação e controle de qualidade

À medida que a busca por padrões de garantia de qualidade de produtos alimentícios aumenta globalmente, a inteligência artificial desempenha um papel crucial na indústria agrícola por meio de suas capacidades de monitoramento. Esses sistemas têm a capacidade precisa de medir características físicas, como tamanho, forma e cor, permitindo a categorização exata de frutas ou vegetais com base em suas propriedades. A automação resultante reduz a dependência do trabalho humano, ao mesmo tempo em que minimiza o desperdício ao longo das cadeias de abastecimento.

A classificação e inspeção visual da qualidade de colheitas na maioria dos casos ainda é realizada manualmente, sendo este um processo lento e suscetível a erros. Desta forma, a visão computacional possibilita a implementação de sistemas automatizados capazes de inspecionar e classificar produtos com base em suas especificações.

Entre as principais aplicações para classificação e inspeção de qualidade podemos citar:

- Classificação de colheitas em diferentes categorias. Um bom exemplo desses sistemas, é a máquina Optyx da Key Technology, que utiliza visão computacional para classificar batatas em diferentes graus.
- Verificação do atendimento a especificações de tamanho, formato e cor em frutas e vegetais, possibilitando produtos deformados ou danificados serem identificados e removidos.
- Detecção de hematomas, manchas incomuns, mofo e outros defeitos nos produtos, inclusive pequenos defeitos invisíveis aos olhos humanos, por meio de imagens hiperespectrais. Isso evita o envio de produtos de qualidade inferior aos consumidores além de reduzir os custos laborais e minimizar o desperdício de alimentos, promovendo assim a sustentabilidade.
- Avaliação do nível de maturação através da análise de cores, possibilitando a classificação de produtos, como morangos, com base nos níveis ideais de vermelhidão para envio ou processamento.
- Garantia da qualidade consistente do produto em todos os lotes e temporadas, atendendo aos requisitos regulatórios como por exemplo em supermercados.

Em suma, a aplicação da visão computacional não se limita à redução do custo de mão de obra vinculado à inspeção manual, mas transcende ao aprimoramento da precisão, prevenindo, assim, as perdas financeiras resultantes. Em uma análise conduzida pela Allied Market Research, a previsão é que o mercado de inteligência artificial na agricultura, especialmente nas aplicações de avaliação de qualidade, atingirá um patamar significativo, projetando-se para alcançar a expressiva cifra de 485 milhões de dólares até o ano de 2027.

Perguntas sobre o artigo “Foundation Models in Smart Agriculture: Basics, Opportunities, and Challenges.”:

1. Qual a principal implicação desta pesquisa ?

Na última década, as metodologias de aprendizado de máquina e aprendizado profundo em sistemas agrícolas se desenvolveram rapidamente. Isto foi demonstrado

através de grande sucesso em diversas aplicações. No entanto, esses modelos tradicionais têm certas limitações, por exemplo, requerem grandes quantidades de dados rotulados para formação, requerem conhecimento especializado para serem desenvolvidos e mantidos e carecem de generalidade porque são específicos da tarefa. Portanto, temos a seguinte uma questão: “Quais são as possibilidades de aplicação de um modelo geral ao campo agrícola?"

2. Qual o propósito desta pesquisa ?

O objetivo deste estudo é explorar o potencial dos Modelos Fundamentais (como é descrito no artigo) no campo da agricultura inteligente. Em particular, são apresentadas ferramentas conceituais e conhecimentos técnicos para facilitar a compreensão do espaço do problema e descobrir novas direções de pesquisa neste campo.

3. Como surgiu a ideia desta pesquisa ?

Os modelos recentes tiveram resultados muito bons em tarefas de linguagem e visuais, o que serviu de inspiração para este estudo. Contudo, apesar disso, pouco progresso foi feito na exploração da aplicação destes modelos no setor agrícola.

4. O que esta pesquisa revelou, como e em que medida?

Primeiro foram revisados os modelos básicos recentes no campo geral da ciência da computação e classificados em quatro categorias. Sendo este: modelos textuais, modelos de visão, modelos multimodais e modelos de aprendizagem por reforço. Em seguida, foi descrito o processo de desenvolvimento de um modelo geral agrícola e discutido suas aplicações potenciais na agricultura inteligente. Além disso, os desafios únicos associados ao desenvolvimento foram discutidos do ponto de vista do treinamento, validação e implantação do modelo.

5. Como foi verificada a eficácia deste estudo?

Este estudo não verificou a eficácia usando conjuntos de dados ou experimentos específicos. No entanto, ao delinear o processo de desenvolvimento foi discutido detalhadamente as oportunidades e desafios dos sistemas de IA agrícola.

Referências:

Jiajia Li, Mingle Xu, Lirong Xiang, Dong Chen, Weichao Zhuang, Xunyuan Yin, Zhaojian Li (2023). Foundation Models in Smart Agriculture: Basics, Opportunities, and Challenges. arXiv:2308.06668 [cs.LG] https://doi.org/10.48550/arXiv.2308.06668

Narendra V. G., Hareesh K. S. (2010). Quality Inspection and Grading of Agricultural and Food Products by Computer Vision - A Review. https://www.ijcaonline.org/volume2/number1/pxc387863.pdf

Mavridou , E. , Vrochidou , E. , Papakostas , G.A. , Pachidis , T. , Kaburlasos , V.G. , Machine vision systems in precision agriculture for crop farming . J. Imaging , 5, (12), 2019 . https://www.mdpi.com/2313-433X/5/12/89

PESQUISA EXPLORATÓRIA [2]

VISANDO PROPORCIONAR MAIOR FAMILIARIDADE COM OS MÉTODOS QUE SERÃO UTILIZADOS

A ideia principal foi fazer um levantamento de artigos, visando proporcionar uma compreensão aprofundada dos métodos envolvidos na classificação e controle de qualidade de frutas por meio da visão computacional. A abordagem adotada não inclui tratamentos estatísticos extensivos, mas busca, em vez disso, uma imersão no universo dessas práticas, a fim de tornar explícita a complexidade do problema.

No levantamento bibliográfico foram realizadas pesquisas nas seguintes bases de dados: Portal de periódicos Capes, IEEE, SciELO, Science Direct, Google Acadêmico e CVPR (principal Conferência sobre Visão Computacional). Para identificar trabalhos relacionados foram consideradas as seguintes palavras-chave: “visão computacional”, “agricultura”, “frutas”, “classificação” e outras no decorrer da pesquisa. Já na revisão da literatura acerca das abordagens para classificação e controle de qualidade foram consideradas também as palavras-chave “controle”, “qualidade”, “inspeção”, “visual”, “automática”, “detecção” as quais foram combinadas, em inglês e português. Nesse segundo levantamento bibliográfico, de maior interesse para esse trabalho, inúmeros artigos foram identificados e, após leitura dos resumos, foram excluídos os que não se referiam ao tema proposto, e mesmo assim ficando uma ampla quantidade. Logo, decidi ir para os mais atuais e identificar aqueles que se aproximasse da ideia do projeto.

Entre os trabalhos recentes encontrados sobre classificação e controle de qualidade de frutas destacam-se os seguintes:

Fruta	Referência	Descrição/técnica usada	Métrica usada
Apple	Gill H.S. and Khehra B.S (2022)	CNN, LSTM, and RNN	Accuracy=70%, RMSE
Banana	Wieme J., Mollazade K., Malounas I., Zude-Sasse M., Zhao M., Gowen A., & Van Beek J. (2022)	Deep CNN	Accuracy=99%

Peach	Sun, H., Huang, X., Chen, T., Zhou, P., Huang, X., Jin, W., & Gao,Z. (2022)	CNN	Accuracy =93%

Além disso, durante a análise dos trabalhos recentes relacionados ao controle de qualidade de frutas, emergiram diversas estratégias inovadoras que evidenciam a dinâmica e a evolução constante da área. Cada abordagem destaca-se por contribuir de maneira singular para o aprimoramento dos métodos de visão computacional aplicados à agricultura. Os principais trabalhos são utilizando redes neurais convolucionais para a análise de imagens, entretanto outros métodos também devem ser considerados, como:

Aplicação de Técnicas de Aprendizado Semi-Supervisionado:

O emprego de técnicas de aprendizado semi-supervisionado tem se destacado como uma resposta inovadora aos desafios associados à disponibilidade limitada de conjuntos de dados rotulados. Essa abordagem visa otimizar a eficiência do processo de treinamento do modelo, tornando-o mais adaptável a diferentes variações nas características das frutas.

Enfoque na Detecção de Defeitos Específicos:

Algumas estratégias concentram-se na detecção específica de defeitos, como manchas, deformidades e irregularidades nas superfícies. Isso não apenas contribui para a avaliação da qualidade geral, mas também possibilita a implementação de medidas corretivas mais direcionadas durante a produção.

Integração de Sensores Espectrais:

Estratégias que envolvem a integração de sensores espectrais, como câmeras multiespectrais, foram identificadas como uma abordagem promissora. Essa técnica permite a captura de informações específicas sobre a composição das frutas, possibilitando uma análise mais abrangente da qualidade, especialmente em relação a parâmetros como maturação e presença de defeitos.

Desenvolvimento de Sistemas Integrados:

Outros estudos destacam a importância do desenvolvimento de sistemas integrados que conectam a análise de imagem com outras variáveis ambientais e de produção. Essa abordagem holística visa proporcionar uma compreensão mais completa dos fatores que impactam a qualidade das frutas.

DEFINIÇÃO DE DATASETS A SEREM UTILIZADOS

O conjunto de dados Fruits 360 foi selecionado inicialmente para este estudo. Este conjunto de dados contém 90.380 imagens de 131 frutas e vegetais diferentes. As imagens coloridas (RGB) possuem resolução de 100 por 100 pixels (portanto, 3 valores para cada pixel). E os dados estão separados na seguinte proporção: 67.692 imagens para treinamento e 22.688 imagens para teste.

Ademais, a escolha deste dataset para esta pesquisa proporciona uma base robusta e diversificada. Contudo, é fundamental ressaltar que a eficácia dos modelos e algoritmos aplicados dependerá, em parte, da representatividade e da natureza específica dos dados.

Ao iniciar os experimentos, será possível realizar testes e avaliações detalhadas sobre a adequação dos dados para os objetivos do estudo. Essa análise envolverá a verificação da diversidade das imagens e a identificação de possíveis desafios.

Durante essa fase experimental, será crucial observar como os modelos respondem a nuances que podem surgir durante a aplicação prática. Caso sejam identificadas limitações significativas ou lacunas nos resultados obtidos com o conjunto de dados, considerações sobre a incorporação de conjuntos de dados adicionais ou a customização do conjunto existente serão avaliadas.

Este processo iterativo de experimentação e análise garantirá uma abordagem fundamentada na escolha dos dados, permitindo adaptações conforme necessário para alcançar os melhores resultados. A finalidade é assegurar que o estudo seja conduzido de maneira abrangente e ajustável, garantindo, assim, uma investigação sólida e eficaz no âmbito da classificação e controle de qualidade de frutas por meio da visão computacional.

LEVANTAMENTO DAS PRINCIPAIS FERRAMENTAS/FRAMEWORKS QUE PODEM SER UTILIZADAS NO PIPELINE

Existem vários frameworks para visão computacional, cada um com suas características e funcionalidades específicas. Aqui estão listados alguns dos principais frameworks utilizados, seja para métodos clássicos ou para aplicações de redes neurais e uma breve descrição deles:

OpenCV (Open Source Computer Vision):

Descrição: O OpenCV é uma biblioteca open-source que fornece ferramentas para desenvolvimento de aplicações de visão computacional. Oferece suporte a uma ampla variedade de algoritmos para processamento de imagem, detecção de objetos, reconhecimento facial, calibração de câmeras, entre outros.

Link: https://opencv.org/

Scikit-image:

Descrição: O scikit-image é um conjunto de ferramentas para processamento de imagem construído sobre o scikit-learn. Ele fornece uma ampla variedade de algoritmos e funções para manipulação e análise de imagens, incluindo segmentação, filtragem, transformações geométricas e extração de características.

Link: https://scikit-image.org/

Pillow:

Descrição: O Pillow é uma biblioteca poderosa para manipulação de imagens em Python. Ele oferece funcionalidades como abertura, manipulação e salvamento de vários formatos de imagem. O Pillow é uma bifurcação do antigo PIL (Python Imaging Library).

Link: https://pillow.readthedocs.io/en/stable/

TensorFlow:

Descrição: Desenvolvido pelo Google, o TensorFlow é um framework de código aberto para aprendizado de máquina e visão computacional. Ele oferece uma arquitetura

flexível que permite a construção de modelos complexos, sendo amplamente utilizado em projetos de deep learning.

Link: https://www.tensorflow.org/?hl=pt-br

PyTorch:

Descrição: PyTorch é outro framework popular para aprendizado de máquina e visão computacional. É conhecido por sua facilidade de uso e oferece uma abordagem dinâmica de construção de gráficos computacionais, o que facilita o processo de experimentação.

Link: https://pytorch.org/

Keras:

Descrição: Originalmente desenvolvido como uma API de alto nível para TensorFlow, o Keras agora é integrado diretamente ao TensorFlow 2.0. Ele simplifica a construção e treinamento de modelos de deep learning, sendo acessível a usuários com diferentes níveis de experiência.

Link: https://keras.io/

Transformers (Hugging Face):

Descrição: A biblioteca transformers, desenvolvida pela Hugging Face, é uma referência em processamento de linguagem natural (PLN) e modelos de aprendizado profundo pré-treinados. Além de PLN, ela inclui modelos para tarefas relacionadas à visão, como classificação, detecção de objetos e geração de imagens.

Link: https://huggingface.co/docs/transformers/index

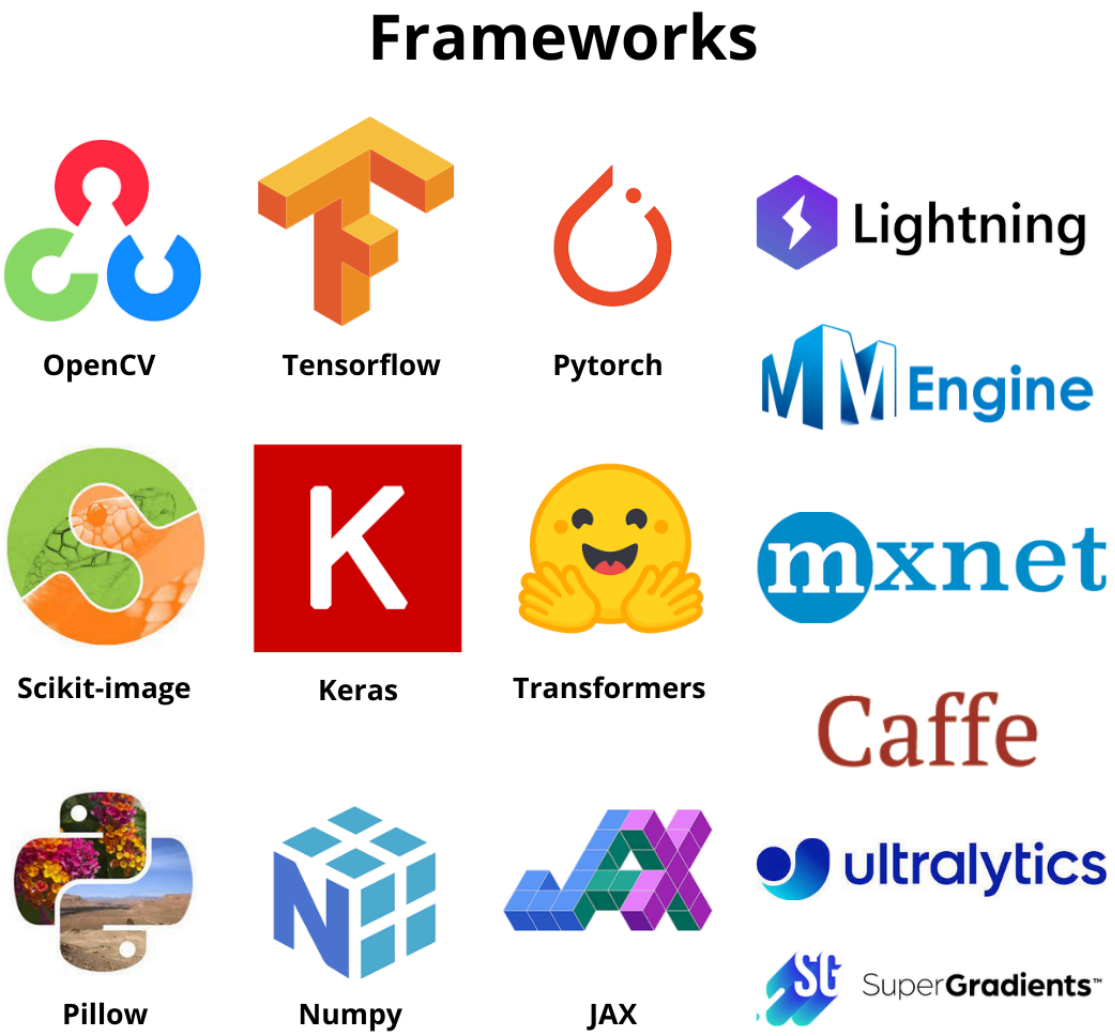


Fig 1: Principais frameworks de Visão Computacional

Obs: Não foram descritos todos frameworks contidos na imagem, apenas os mais utilizados.

Referências:

Gill,H.S., & Khehra, B.S. (2022). Fruit image classification using deep learning.

Wieme, J., Mollazade, K., Malounas, I.,Zude-Sasse, M.,Zhao, M, Gowen, A, & Van Beek, J. (2022). Application of hyperspectral imaging systems and artificial intelligence for quality assessment of fruit, vegetables and mushrooms:A review. Biosystems Engineering, 222, 156-176.

Sun, H., Huang, X., Chen, T., Zhou, P., Huang, X., Jin, W., & Gao,Z. (2022). Fruit Quality Prediction Based On Soil Mineral element content in peach orchard. Food Science & Nutrition.

Arakeri, M. P. (2016). Computer vision based fruit grading system for quality evaluation of tomatoes in the agriculture industry. Procedia Computer Science, 79, 426-433.

Singh, P. M., Maity, D., Saha, S., & Dhal, N. K. (2022). Seaweed utilization and its economy in Indian agriculture. Materials Today: Proceedings.

Mohapatra, D., Das, N., Mohanty, K.K., & Shresth, J. (2022). Automated Visual Inspecting System for Fruit Quality Estimation Using Deep Learning. In Innovation in Electrical Power Engineering, Communication, and Computing Technology (pp. 379-389). Springer, Singapore.

Mohapatra, D., Das, N., & Mohanty, K. K. (2022). Deep neural network-based fruit identification and grading system for precision agriculture. Proceedings of the Indian National Science Academy, 1-12.

Links úteis:

https://expertbeacon.com/computer-vision-agriculture/
https://www.opencv.ai/blog/computer-vision-in-agriculture-challenges-solutions
https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119792109.ch11
https://mindtitan.com/resources/blog/computer-vision-in-agriculture/
https://github.com/RobustMM/Robust-Vision-Language-Model-Literature-Review
https://agtecher.com/agrigpt-llm-2sided/

APÊNDICE 4

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 30 de nov. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para esta entrega, conforme o planejamento, temos os seguinte objetivos:

- ☑ Coleta e pré-processamento dos dados.
- ☑ Implementação inicial dos modelos de classificação de imagens.
- ☑ Revisão de artigos.

Os objetivos listados foram incluídos em um documento no seguinte link:
Entrega 30/11- MARCOS VINICIUS SATIL MEDEIROS

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega seguirei os passos feitos no Gate 6 (30/11), onde foi abordada a tarefa de classificação. Dessa forma, teremos os seguinte objetivos:

- Melhoria do código e documentação.
- Reformatação do repositório, adicionando código feitos no Github: (https://github.com/marcosvinism/Residencia-em-IA).
- Testagem de novas arquiteturas como Vision transformers e diferentes hiperparâmetros nas arquiteturas já utilizadas.
- Realização de experimentos controlados para avaliar o impacto dessas alterações nas métricas de desempenho.
- Revisão do cronograma estabelecido.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

Neste gate, o Professor Aldo André Díaz Salazar esteve na banca avaliadora substituindo a Professora Luana.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

LUANA GUEDES BARROS MARTINS: Em análise!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 7 de dez. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para esta entrega, conforme o planejamento, temos os seguinte objetivos:

- ☑ Melhoria do código e documentação.
- ☑ Reformatação do repositório, adicionando código feitos no Github: (https://github.com/marcosvinism/Residencia-em-IA).
- ☑ Testagem de novas arquiteturas como Vision transformers e diferentes hiperparâmetros nas arquiteturas já utilizadas.
- ☑ Realização de experimentos controlados para avaliar o impacto dessas alterações nas métricas de desempenho.
- ☑ Revisão do cronograma estabelecido.

Os objetivos listados foram incluídos em um documento no seguinte link:
Entrega 07/12- MARCOS VINICIUS SATIL MEDEIROS

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega, diferentemente dos passos anteriores, onde foi abordada a tarefa de classificação, iremos implementar a tarefa de detecção. Dessa forma, teremos os seguinte objetivos:

- Revisão de artigos.
- Início da implementação dos modelos de detecção.
- Obtenção dos dados similares para esta tarefa.
- Testagem de diferentes de arquiteturas.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

Neste gate, o Professor Aldo André Díaz Salazar esteve na banca avaliadora substituindo a Professora Luana.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

LUANA GUEDES BARROS MARTINS: Em análise!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

MODELOS DE CLASSIFICAÇÃO DE IMAGENS

O objetivo principal das entregas referentes aos gates 6 e 7 (30/11-07/12) foi explorar diferentes arquiteturas de modelos de classificação para avaliar sua eficácia na tarefa de identificação das frutas presentes no conjunto de dados selecionado. Dado que já realizamos revisões de literatura em entregas anteriores, neste documento iremos descrever de forma simples alguns conceitos aplicados, assim como as arquiteturas exploradas. O código produzido pode ser encontrado no repositório do github:

https://github.com/marcosvinism/Residencia-em-IA

Este texto é uma versão 2 do documento contido na entrega do dia 30/11: Entrega 30/11- MARCOS VINICIUS SATIL MEDEIROS

Conceitos sobre classificação

Reconhecimento/Classificação de Imagens:

É o processo de prever a classe ou rótulo específico, através da análise de grupos de pixels e vetores no contexto de uma imagem. Esta tarefa desempenha um papel importante em diversas aplicações, desde a identificação de objetos simples até reconhecimento facial em sistemas de segurança.

Existem dois principais tipos de classificação de imagens:

- Não Supervisionada:

A classificação não supervisionada é um método totalmente automatizado, onde algoritmos são empregados para analisar e agrupar dados em conjuntos. Este processo ocorre pela descoberta de padrões implícitos sem a necessidade de intervenção humana. Um exemplo notável desse método é o algoritmo K-means, que efetua a clusterização de dados de forma eficiente, agrupando-os com base em similaridades.

- Supervisionada:

A classificação supervisionada, por outro lado, utiliza amostras previamente classificadas como base para treinar o classificador. Isso significa que o algoritmo é alimentado com dados rotulados, permitindo-lhe aprender padrões e características específicas associadas a cada classe. Posteriormente, o classificador treinado é capaz de

generalizar esse conhecimento para identificar e classificar novos dados. Esse método é comumente empregado em problemas complexos nos quais é necessária uma compreensão mais profunda das características das classes.

Conceitos relacionados a rede neurais

CONVOLUTIONAL NEURAL NETWORK (CNN)

A ideia por trás das CNNs é inspirada no nosso cérebro e em como processamos visualmente as coisas. Assim como nossos neurônios trabalham em conjunto para reconhecer padrões, as camadas da CNN fazem o mesmo. Elas recebem estímulos (imagens) e compartilham informações entre si para entender o que estão vendo. Um aspecto interessante sobre as CNNs é que elas conseguem aprender com a experiência, assim como nós. Elas usam pesos e vieses para ajustar suas respostas com base nos exemplos que veem. Isso ajuda a torná-las muito boas em identificar objetos em diferentes situações.

Desta forma, uma Rede Neural Convolucional (CNN) funciona como uma equipe organizada de detetives para imagens. Cada "detetive" (camada) usa um conjunto especial de ferramentas (kernels convolucionais) para investigar diferentes partes da imagem. Esses detetives trabalham em conjunto, passando informações uns para os outros até que a rede seja capaz de identificar ou distinguir objetos nas imagens.

Essas redes são como mágicos que conseguem ver padrões nas imagens usando filtros especiais. Esses filtros ajudam a reduzir a quantidade de coisas que a CNN precisa analisar e a reutilizar o que ela já aprendeu.

As CNNs têm tipos de camadas e funções, que vamos explicar a seguir:

Camada de Convolução (Convolutional Layer)

A camada de convolução é como um conjunto de pincéis mágicos, cada um com suas configurações especiais (kernels/filtros) e ajustes que são aprimorados durante o treinamento. Durante essa fase, a camada realiza uma operação chamada convolução, onde cada filtro "pinta" sobre a entrada (input), criando assim um mapa de ativação, como se fosse um mapa de destaque ou padrões importantes no espaço da entrada.

Imagine que o input é como uma tela com dimensões (Número de inputs, altura do input, largura do input, canais do input). A camada de convolução, ao fazer sua mágica, retorna um novo mapa de ativação com dimensões (Número de inputs, altura do feature map, largura do feature map, canais do feature map). Em outras palavras, ela destaca as áreas significativas, transformando o input original em algo mais compreensível e útil para as camadas seguintes da rede neural.

Camada de Pooling/Downsampling (Pooling Layer)

A camada de Pooling tem como função a redução das dimensões do tensor. Sua missão é agrupar resultados semelhantes em conjuntos de neurônios, simplificando assim as operações e contribuindo para uma maior generalização, o que ajuda a evitar que a rede se torne excessivamente especializada (overfitting). Existem diferentes técnicas de pooling, tais como:

- Max Pooling
- Average Pooling
- L2 Pooling
- Overlapping Pooling
- Spatial Pyramid Pooling

Cada uma dessas abordagens de pooling possui suas características distintas, proporcionando maneiras únicas de destacar e resumir informações relevantes nos dados, tornando o processo de aprendizado mais eficiente e robusto.

Camada Totalmente Conectada (Fully Connected Layer)

A Camada Totalmente Conectada entra em cena geralmente no final das redes neurais, após o feature map ter sido achatado, com o objetivo de classificação. Essa camada age como um Perceptron Multicamadas (MLP), composta por uma camada oculta, uma função de ativação e uma função de perda (loss function). Desta forma, utilizando essas ferramentas, a imagem é classificada com base nas informações contidas no feature map. Em essência, é como se essa camada final organizasse e interpretasse os dados para produzir a classificação final da imagem.

Pesos

Os pesos são parâmetros ajustáveis cuja função é controlar o fluxo de informações entre neurônios. Eles são responsáveis por determinar a magnitude da influência que cada característica do input terá sobre o output. Em essência, os pesos moldam a contribuição relativa de diferentes características na tomada de decisões da rede neural.

Função de Ativação

A função de ativação é como um "interruptor de luz" para a rede neural, ajudando-a a aprender padrões importantes. A escolha dessa função pode fazer o treinamento ser mais rápido ou mais devagar. Existem várias funções populares para isso, como:

- Sigmoid
- Tangente Hiperbólica
- SWISH
- ReLU e suas variações

Função de Perda (Loss Function)

A função de perda é como um "juiz" que avalia o desempenho do seu modelo, comparando o que era esperado com o que foi realmente produzido. Assim como a função de ativação, existem diferentes tipos de funções de perda, como:

- Cross-entropy
- Exponential Loss
- Mean Square Error (MSE)
- Mean Absolute Error (MAE)

Conceitos e informações sobre a implementação

Frameworks utilizados

Inicialmente, utilizamos diretamente a biblioteca PyTorch, mas por conta da simplicidade iremos também utilizar a biblioteca de alto nível Fast.ai.

Fast.ai é uma biblioteca criada por Jeremy Howard e Rachel Thomas dedicada a tornar o poder do Aprendizado Profundo acessível a todos. Para tanto, argumentam que,

para esse potencial ser atingido, a tecnologia tem de ser muito mais acessível, intuitiva e fácil de usar. A biblioteca tem sido altamente utilizada em visão computacional, processamento de linguagem natural, entre outras. Além disso, a Fastai oferece uma abordagem de Transfer Learning, que permite ao usuário utilizar modelos pré-treinados para acelerar treinamento.

Dataset utilizado - Fruits 360

O dataset Fruits 360 consiste em uma ampla variedade de imagens de frutas, organizadas em diversas categorias. Cada categoria representa uma espécie específica de fruta, tornando-o um conjunto de dados robusto e diversificado para treinar e avaliar modelos de classificação. A obtenção do dataset está disponível em: https://www.kaggle.com/moltean/fruits

Pré-processamento dos Dados

Após a obtenção dos dados do conjunto Fruits 360, realizamos o pré-processamento para preparar as imagens antes de aplicar as arquiteturas de modelos.

Utilizamos transformações específicas para cada conjunto de dados (treinamento, validação e teste) por meio da biblioteca torchvision. Estas transformações foram aplicadas visando melhorar a diversidade e a representatividade do conjunto de treinamento, aumentando assim a robustez do modelo. Para enriquecer essa diversidade, implementamos técnicas de aumento de dados incluindo rotações aleatórias, recortes redimensionados e espelhamentos horizontais, proporcionando ao modelo uma maior capacidade de generalização.

Essas estratégias de pré-processamento são fundamentais para garantir que o modelo seja exposto a uma variedade suficiente de exemplos durante o treinamento, promovendo um desempenho mais robusto e confiável quando confrontado com dados reais.

Exploração de Diferentes Arquiteturas de Modelos

No processo de desenvolvimento do modelo de classificação, exploramos diversas arquiteturas de modelos renomadas, utilizando assim os pesos já pré-carregados das bibliotecas Torch Vision e Timm para fazer o transfer learning. Cada arquitetura foi escolhida com base em suas características específicas e no potencial para a tarefa em questão. Destacamos algumas das arquiteturas investigadas:

ResNet

A ResNet, ou Rede Neural Residual, é um modelo inovador desenvolvido por Kaiming He e apresentado no artigo de 2015. Este modelo marcou um avanço significativo ao conseguir utilizar com sucesso redes neurais feedforward extremamente profundas. A ideia-chave por trás da ResNet foi a introdução do conceito de "Identity shortcut connection", que permite pular uma ou mais camadas durante o processo de aprendizado.

Outro conceito importante implementado na ResNet é o "Pre-activation variant of residual block". Essa abordagem possibilita que os gradientes fluam através de atalhos para qualquer camada anterior, facilitando o treinamento da rede. Embora tenha sido criada a algum tempo, a ResNet continua a ser altamente relevante. Ao longo dos anos, várias versões foram desenvolvidas, variando tanto no número de camadas quanto na arquitetura, demonstrando sua versatilidade e impacto duradouro no campo das redes neurais.

A RestNet é diferente das redes neurais tradicionais no sentido de que pega resíduos de cada camada e os utiliza nas camadas conectadas subsequentes (semelhante às redes neurais residuais usadas para previsão de texto).

DenseNet

DenseNet, ou Densely Connected Convolutional Network, representa uma abordagem revolucionária no cenário de modelos de classificação de imagens. A arquitetura DenseNet161, escolhida para estudo e destaca-se por sua inovação na forma como as camadas estão interconectadas. Diferentemente de abordagens tradicionais, o DenseNet adota uma estratégia densa, conectando cada camada diretamente umas às outras.

A arquitetura DenseNet161, em particular, se destaca por sua profundidade e eficácia na extração de características complexas. Ao contrário de modelos convencionais, o DenseNet permite que cada camada tenha acesso direto às informações geradas por todas

as camadas precedentes, promovendo uma representação mais rica e detalhada dos padrões presentes nas imagens.

VGG

VGG (Very Deep Convolutional Networks for Large-Scale Image Recognition) é uma arquitetura de rede neural convolucional apresentada por Karen Simonyan em seu artigo, publicado em 2014. Esta rede é uma referência em simplicidade combinada com eficácia, tornando-se uma escolha notável para tarefas de classificação de imagens.

Nossa opção de estudo recaiu sobre a variante VGG19, que se caracteriza por suas 19 camadas, que estabelecem um equilíbrio singular entre desempenho e capacidade de generalização.

A notável simplicidade da VGG, com suas camadas convolucionais profundas, oferece uma compreensão aprimorada de padrões complexos nas imagens. A eficácia desta arquitetura foi evidenciada em diversos contextos, consolidando sua reputação como uma escolha sólida para desafios de classificação. Durante a avaliação de desempenho, a VGG19 demonstrou consistência ao traduzir sua arquitetura simples em resultados expressivos. Sua capacidade de generalização, aliada à simplicidade estrutural, oferece uma abordagem robusta para a tarefa de classificação de frutas. Essa escolha da VGG19 se revela como uma opção equilibrada, eficiente e confiável..

Inception

A arquitetura Inception é amplamente reconhecida por sua habilidade única de processar informações em diferentes escalas, conferindo-lhe uma notável eficácia em tarefas de classificação. Inspirada na proposta do paper que introduziu a arquitetura Inception, a qual definiu um novo padrão para classificação e detecção no desafiador ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC 2014), a Inception destaca-se por uma utilização aprimorada dos recursos computacionais.

Nossa escolha foi sobre a versão InceptionV3, uma iteração refinada dessa arquitetura. Ao desativar camadas auxiliares na InceptionV3, buscamos simplificar a estrutura sem comprometer a essência do poder multi-escala que define essa arquitetura. Durante a avaliação, buscamos compreender como a InceptionV3, com suas características

distintivas, se adapta à complexidade intrínseca do conjunto de dados. Essa escolha reforça nossa busca por uma abordagem eficaz e adaptável para a tarefa de classificação.

EfficientNet

A EfficientNet representa uma família de arquiteturas de redes neurais profundas, conforme introduzido no artigo de Tan e Le em 2019. Essa proposta visa maximizar tanto a eficiência computacional quanto a precisão do modelo. A abordagem adotada envolve um método de escalonamento que considera largura, profundidade, resolução e a incorporação de blocos de construção eficientes.

Dessa maneira, o resultado obtido é uma arquitetura altamente eficiente, destacando-se pela capacidade de ser facilmente adaptada para diferentes tamanhos e finalidades. Essa flexibilidade torna o EfficientNet uma escolha versátil para diversas aplicações no campo de aprendizado profundo.

HRNet

HRNet (High Resolution Neural Network) é uma arquitetura de rede neural convolucional apresentada no artigo de Wang e colaboradores em 2019. O principal objetivo desta arquitetura é aprimorar a precisão na compreensão do corpo humano e suas poses, embora também seja eficaz para outras finalidades devido à sua eficiência notável. Diferenciando-se de muitas redes neurais convencionais, a HRNet mantém a resolução mais alta durante todo o processo computacional, graças a múltiplos caminhos de processamento paralelo.

A HRNet segue um design linear e sequencial semelhante às redes neurais VGG do início da era do aprendizado profundo. No entanto, sua singularidade reside na divisão do fluxo de processamento em quatro caminhos paralelos com resoluções diferentes: 1/1, 1/2, 1/4 e 1/8 de resolução. A linearidade é interrompida por conexões verticais que integram os quatro fluxos em intervalos regulares.

Essa rede pode ser adaptada para três tipos de tarefas diferentes, como classificação de imagens, detecção de objetos ou segmentação semântica. Além de suas capacidades funcionais, a HRNet se destaca por sua homogeneidade, estrutura lógica e simplicidade, tornando-a visualmente atraente e fácil de compreender. Essas características

fazem dela uma escolha adequada não apenas para aplicações práticas, mas também para fins educacionais.

Vision Transformers (ViT)

A abordagem Vision Transformers (ViT) de Dosovitskiy e colaboradores em 2020 representa uma aplicação dos modelos Transformers, originalmente utilizados em processamento de linguagem natural (NLP), agora adaptados para visão computacional. Os Transformers são uma forma avançada de aprendizado profundo usada para tradução e sumarização, projetada para processar dados de maneira sequencial, lendo o input de uma vez só.

Na arquitetura do ViT, a imagem é dividida inicialmente em pequenos patches, cada um transformado em um vetor e processado por uma camada de embedding. Em seguida, um token especial de classificação é adicionado à sequência resultante, que é então inserida em uma série de blocos Transformer. Esses blocos utilizam uma atenção de múltiplas cabeças para entender as interações entre os patches. A previsão final do modelo é gerada processando o token de classificação através da função softmax, resultando nas probabilidades de classe.

Há duas variantes do Vision Transformer, conhecidas como "Small Patch" e "Base Patch". A principal diferença entre elas reside no tamanho do modelo (número de parâmetros) e na quantidade de dados necessários para um treinamento eficaz. O "Small Patch" tem menos parâmetros, sendo um modelo menor e geralmente mais rápido para treinar. Contudo, devido ao seu tamanho, pode ter menos capacidade para aprender representações complexas em comparação com o outro modelo.

Treinamento e Avaliação dos Modelos

O treinamento dos modelos foi realizado utilizando tanto a biblioteca PyTorch quanto a Fast.ai. Durante este processo, a avaliação foi fundamentada em métricas como acurácia, precisão, recall e F1-score, proporcionando uma compreensão do desempenho dos modelos.

Antes do treinamento, os parâmetros dos modelos pré-treinados foram congelados para evitar ajustes durante as primeiras etapas. Em seguida, uma rede neural completamente conectada foi anexada ao modelo para a tarefa específica de classificação.

O treinamento foi conduzido por meio de uma função personalizada, que itera sobre o número especificado de épocas. Durante cada época, tanto o conjunto de treinamento quanto o conjunto de validação foram avaliados, ajustando os pesos do modelo com base na função de perda e otimizador escolhidos. A função também monitora o desempenho ao longo do tempo, registrando as métricas de perda e acurácia para análise posterior.

Esse processo não apenas capacitou os modelos a aprenderem padrões relevantes nos dados de treinamento, mas também avaliou sua capacidade de generalização, garantindo um bom desempenho em conjuntos de dados não vistos.

Resultados

Os resultados provenientes da avaliação dos diversos modelos proporcionaram insights valiosos sobre o desempenho global na tarefa de classificação de frutas. Ao destacar as métricas fundamentais, obtivemos uma visão holística do comportamento de cada arquitetura testada. Observamos variações significativas, desde o tempo de treinamento até o desempenho das métricas, entre as diferentes arquiteturas. Aprofundando nossa análise, examinamos padrões emergentes, identificando áreas de destaque e possíveis pontos de melhoria.

Dentro desse cenário, exploramos o desempenho de diversas redes neurais distintas, abrangendo os modelos de redes convolucionais e Vision Transformers. O principal objetivo foi comparar e avaliar a eficácia de cada arquitetura na tarefa específica de classificação de frutas. Essa abordagem possibilitou a exploração de diversas técnicas de Deep Learning, contribuindo para uma compreensão mais aprofundada do tema.

Durante os experimentos, todas as redes neurais foram treinadas ao longo de 10 épocas, utilizando taxas de aprendizagem (learning rates) variáveis com base no localizador da taxa do Fast.ai. A escolha desses valores também foi orientada pela ausência de variação significativa ou pela identificação de um platô nas funções de perda, tanto na validação (Validation Loss) quanto no treinamento (Train Loss). Esses fenômenos foram observados consistentemente na maioria das redes durante o processo de treinamento.

Em resumo, esta fase de avaliação e análise não apenas proporcionou uma compreensão aprofundada dos resultados obtidos, mas também orientou as decisões subsequentes, mesmo considerando que os testes foram conduzidos de maneira controlada.

epoch	train_loss	valid_loss	error_rate	accuracy	f1_score	precision_score	recall_score	time
0	0.085235	0.287775	0.080514	0.919486	0.910301	0.932871	0.922350	02:38
1	0.134511	0.088342	0.028512	0.971488	0.970128	0.977009	0.970437	02:47
2	0.081820	0.104619	0.030950	0.969050	0.965760	0.979856	0.967057	02:39
3	0.047784	0.012780	0.003989	0.996011	0.995666	0.996255	0.995637	02:36
4	0.032178	0.034571	0.007756	0.992244	0.991238	0.993720	0.991814	02:37
5	0.013793	0.002838	0.000960	0.999040	0.999018	0.999081	0.999002	02:36
6	0.007299	0.001547	0.000665	0.999335	0.999197	0.999147	0.999281	02:36
7	0.001012	0.000041	0.000000	1.000000	1.000000	1.000000	1.000000	02:36
8	0.000595	0.000002	0.000000	1.000000	1.000000	1.000000	1.000000	02:36
9	0.000466	0.000001	0.000000	1.000000	1.000000	1.000000	1.000000	02:41

Fig: Exemplo de saída das métricas de treinamento com o modelo Resnet34

Conclusão

Diante dos conhecimentos teóricos consolidados e das explorações realizadas nas arquiteturas de redes neurais convolucionais, torna-se evidente a contribuição significativa proporcionada por esses modelos na tarefa de classificação.

Ao realizar o deploy com o framework Gradio para avaliação em dados reais, além do conjunto de testes, ampliamos a aplicabilidade prática dos modelos. No entanto, os resultados obtidos nos quatro experimentos finais revelaram que as arquiteturas de redes convolucionais apresentaram desempenhos semelhantes e satisfatórios na análise e processamento do conjunto de dados. Os Vision Transformers, reconhecidos por sua eficiência em diversos cenários, por sua vez, tiveram um desempenho inferior para este tipo específico de dados.

Esta constatação sugere a importância de considerar a adequação das arquiteturas para cada problema específico. A hipótese é que o tempo de treinamento pode ter contribuído para o desempenho inferior dos Vision Transformers, pois é respaldada pelo

próprio artigo dos criadores, destacando a exigência de uma quantidade substancialmente maior de dados para treinamento em comparação com redes tradicionais e por mais épocas.

Diante desses resultados, delineamos direções para futuras pesquisas, enfocando a continuidade do treinamento dos modelos, ajustes nos hiperparâmetros, exploração de técnicas avançadas e consideração da possibilidade de integrar outras estratégias de aprendizado. Em suma, este estudo não apenas oferece insights valiosos sobre o desempenho das arquiteturas exploradas, mas também estabelece bases para investigações futuras, visando otimizar a eficácia dos modelos em tarefas específicas de classificação de frutas.

Referências

LECUN, Y. et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, IEEE, v. 86, n. 11, p. 2278–2324, 1998.

HE, K. et al. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015. Disponível em: http://arxiv.org/abs/1512.03385.

HOWARD, J.; GUGGER, S. fastai: A layered api for deep learning. arXiv preprint arXiv:2002.04688, 2020.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

TAN, M.; LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. International Conference on Machine Learning, 2019.

WANG, K. et al. Deep high-resolution representation learning for human pose estimation. IEEE Conference on Computer Vision and Pattern Recognition, 2019.

DOSOVITSKIY, A. et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv, 2020. Disponível em: https://arxiv.org/abs/2010.11929.

Jupyter Notebook: Modelos de classificação.ipynb

Author: Marcos Vinicius Satil Medeiros

Modelos de Classificação

O objetivo deste notebook é treinar modelos de aprendizado profundo para prever o rótulo/nome de uma determinada fruta tendo apenas uma imagem de entrada.

Bibliotecas utilizadas

```
%%capture
!pip install opendatasets
```

```
import os
import time
import copy
import tarfile
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import opendatasets as od
import torch.nn.functional as F
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as tt
from torchvision import datasets,transforms,models
from torchvision.utils import make_grid
from torchvision.datasets.utils import download_url
from torchvision.datasets import ImageFolder
from torch.utils.data import *
from torch import nn,optim
from PIL import Image
```

Baixando dataset do Kaggle com OPENDATASETS

```
od.download('https://www.kaggle.com/moltean/fruits')
```

```
data_dir = '/content/fruits/fruits-360_dataset/fruits-360'
print(os.listdir(data_dir))

len(os.listdir(data_dir + '/Training'))
```

Pré-processamento dos dados

```
train_dir = data_dir + '/Training'
test_dir = data_dir + '/Test'
batch_size = 32

# Aumento e normalização de dados para treinamento
# Apenas normalização para validação
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomRotation(30),
        transforms.RandomResizedCrop(299),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'valid': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(299),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),

    'test': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(299),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
    ]),
}

dataset =
datasets.ImageFolder(train_dir,transform=data_transforms['train'])
# split dos dados
valid_size  = int(0.2 * len(dataset))
train_size = len(dataset) - valid_size
dataset_sizes = {'train': train_size, 'valid': valid_size}

train_dataset, valid_dataset = torch.utils.data.random_split(dataset,
[train_size, valid_size])
```

```
test_dataset =
datasets.ImageFolder(test_dir,transform=data_transforms['test'])

# carregando datasets no dataloader
dataloaders = {'train': DataLoader(train_dataset, batch_size = batch_size,
shuffle = True),
               'valid': DataLoader(valid_dataset, batch_size = batch_size,
shuffle = False),
               'test': DataLoader(test_dataset, batch_size = batch_size,
shuffle = False)}

print("Número total de amostras no dataset: ",len(dataset))
print("Número de amostras de Train: ",len(train_dataset))
print("Número de amostras de Valid: ",len(valid_dataset))
print("Número de amostras de Test: ",len(test_dataset))
print("Número de classes: ",len(dataset.classes))

dataset.classes[:10]
```

Inicialmente, temos imagens de 100 x 100 pixel com 3 canais de cores (rgb), mas aplicando as transformações o tamanho da imagem aumenta.

```
train_dataset[10][0].shape

torch.Size([3, 299, 299])
```

Visualizando imagens

```
def show_example(img, label):
  print('Label: ', dataset.classes[label], "("+str(label)+")")
  plt.imshow(img.permute(1, 2, 0))

show_example(*dataset[100])
```

Visualizando batch de imagens

```
def show_batch(dl):
  for images, labels in dl:
    fig, ax = plt.subplots(figsize = (8, 6))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(images, nrow = 4).permute(1, 2, 0))
    break

show_batch(dataloaders['train'])
```

Arquitetura do modelo

Setando device

```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

device = get_default_device()
device

device(type='cuda')
```

Carregando modelo pré-treinados para Transfer Learning

```
model_name = 'vgg'
if model_name == 'densenet':
    model = models.densenet161(pretrained=True)
    num_in_features = 2208
    print(model)
elif model_name == 'vgg':
    model = models.vgg19(pretrained=True)
    num_in_features = 25088
    print(model.classifier)
elif model_name == 'resnet':
    model = models.resnet34(pretrained=True)
    num_in_features = 512
    print(model.fc)
elif model_name == 'inception':
    model = models.inception_v3(pretrained=True)
    model.aux_logits=False
    num_in_features = 2048
    print(model.fc)
else:
    print("Unknown model")
```

Congelando parâmetros da rede para construir classificador

```
for param in model.parameters():
    param.require_grad = False

# Criando classificador
```

```
def build_classifier(num_in_features, hidden_layers, num_out_features):
    classifier = nn.Sequential()
    if hidden_layers == None:
        classifier.add_module('fc0', nn.Linear(num_in_features, 196))
    else:
        layer_sizes = zip(hidden_layers[:-1], hidden_layers[1:])
        classifier.add_module('fc0', nn.Linear(num_in_features,
hidden_layers[0]))
        classifier.add_module('relu0', nn.ReLU())
        classifier.add_module('drop0', nn.Dropout(.6))

        for i, (h1, h2) in enumerate(layer_sizes):
            classifier.add_module('fc'+str(i+1), nn.Linear(h1, h2))
            classifier.add_module('relu'+str(i+1), nn.ReLU())
            classifier.add_module('drop'+str(i+1), nn.Dropout(.5))
        classifier.add_module('output', nn.Linear(hidden_layers[-1],
num_out_features))

    return classifier

hidden_layers = None
classifier = build_classifier(num_in_features, hidden_layers, 196)
print(classifier)

 # Definindo hiperparametros do modelo
if model_name == 'densenet':
    model.classifier = classifier
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.0001,
weight_decay=0.001, momentum=0.9)
    sched = optim.lr_scheduler.StepLR(optimizer, step_size=4, gamma=0.1)
elif model_name == 'vgg':
    model.classifier = classifier
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.0001,weight_decay=0.001,
momentum=0.9)
    sched = optim.lr_scheduler.StepLR(optimizer, step_size=4, gamma=0.1)
elif model_name == 'resnet':
    model.fc = classifier
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
    sched = optim.lr_scheduler.StepLR(optimizer, step_size=4, gamma=0.1)
elif model_name == 'inception':
    model.fc = classifier
    criterion = nn.CrossEntropyLoss()
```

```
    optimizer = optim.SGD(model.parameters(), lr=0.001,momentum=0.9)
    sched = optim.lr_scheduler.StepLR(optimizer, step_size=4, gamma=0.1)
else:
    pass
```

```
Sequential(
  (fc0): Linear(in_features=25088, out_features=196, bias=True)
)
```

Treinamento do modelo

```
def train_model(model, criterion, optimizer, sched,
num_epochs=5,device='cuda'):
    start = time.time()
    train_results = []
    valid_results = []
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch+1, num_epochs))
        print('-' * 10)
        # Cada época tem uma fase de treinamento e validação
        for phase in ['train', 'valid']:
            if phase == 'train':
                model.train()  # Definindo modelo para modo de treinamento
            else:
                model.eval()   # Definindo modelo para modo de avaliação
            running_loss = 0.0
            running_corrects = 0
            # Iterarando sobre os dados.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)
                # zerar os gradientes dos parâmetros
                optimizer.zero_grad()
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)
                    if phase == 'train':
                        #sched.step()
                        loss.backward()
                        optimizer.step()
                # estatisticas
                running_loss += loss.item() * inputs.size(0)
```

```
                running_corrects += torch.sum(preds == labels.data)
            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            if(phase == 'train'):
              train_results.append([epoch_loss,epoch_acc])
            if(phase == 'valid'):
              valid_results.append([epoch_loss,epoch_acc])

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss,
epoch_acc))

            if phase == 'valid' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
                model_save_name = "FruitVGG.pt"
                path = F"/content/{model_save_name}"
                torch.save(model.state_dict(), path)
        print()

    time_elapsed = time.time() - start
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    #carregando best model weights
    model.load_state_dict(best_model_wts)

    return model,train_results,valid_results

epochs = 3
model.to(device)
model,train_results,valid_results = train_model(model, criterion,
optimizer, sched, epochs)
```

Métricas do modelo

```
# Converter tensores para valores na CPU
train_results_np = [[item[0], item[1].cpu().item()] for item in
train_results]
valid_results_np = [[item[0], item[1].cpu().item()] for item in
valid_results]

# Converter para NumPy para plotagem
train_results_np = np.array(train_results_np)
```

```
valid_results_np = np.array(valid_results_np)

# Plot dos Losses
plt.plot(train_results_np[:, 0], '-o', label='Train Loss')
plt.plot(valid_results_np[:, 0], '-o', label='Valid Loss')
plt.xlabel('Epoch Number')
plt.ylabel('Loss')
plt.ylim(0, 2)  # Ajuste o limite conforme necessário

# Adicionar os valores no gráfico
for i, txt in enumerate(train_results_np[:, 0]):
    plt.annotate(f'{txt:.4f}', (i, txt), textcoords="offset points",
xytext=(0, 5), ha='center')

for i, txt in enumerate(valid_results_np[:, 0]):
    plt.annotate(f'{txt:.4f}', (i, txt), textcoords="offset points",
xytext=(0, 5), ha='center')

plt.legend()
plt.show()

# Plot das Accuracies
plt.plot(train_results_np[:, 1], '-o', label='Train Accuracy')
plt.plot(valid_results_np[:, 1], '-o', label='Valid Accuracy')
plt.xlabel('Epoch Number')
plt.ylabel('Accuracy')
plt.ylim(0, 1)  # Ajuste o limite conforme necessário

# Adicionar os valores no gráfico
for i, txt in enumerate(train_results_np[:, 1]):
    plt.annotate(f'{txt:.4f}', (i, txt), textcoords="offset points",
xytext=(0, 5), ha='center')

for i, txt in enumerate(valid_results_np[:, 1]):
    plt.annotate(f'{txt:.4f}', (i, txt), textcoords="offset points",
xytext=(0, 5), ha='center')

plt.legend()
plt.show()
```

Inferência

Carregando modelo gerado

```
model.load_state_dict(torch.load('/content/FruitVGG.pt'))
model.to(device)
```

Avaliação do modelo

Teste por classe

```
def class_test(model, test_loader, criterion, classes):
    total_class = len(classes)
    test_loss = 0.0
    class_correct = list(0. for i in range(total_class))
    class_total = list(0. for i in range(total_class))
    model.eval()
    for data, target in test_loader:
        data,target = data.to(device), target.to(device)
        output = model(data)
        loss = criterion(output, target)
        test_loss += loss.item() * data.size(0)
        _, pred = torch.max(output, 1)
        correct = np.squeeze(pred.eq(target.data.view_as(pred)))
        for i in range(len(target) - 1):
            label = target.data[i]
            class_correct[label] += correct[i].item()
            class_total[label] += 1
    test_loss = test_loss / len(test_loader.dataset)
    print('Loss de teste: {:.6f}\n'.format(test_loss))

    for i in range(total_class):
        if class_total[i] > 0:
            print('Acuracia de teste %5s: %2d%% (%2d/%2d)' % (
                str(i), 100 * class_correct[i] / class_total[i],
                np.sum(class_correct[i]), np.sum(class_total[i])))
        else:
            print('Acuracia de teste %5s: N/A (no training examples)' %
(classes[i]))

    print('\nAcuracia de teste (Overall): %2d%% (%2d/%2d)' % (
        100. * np.sum(class_correct) / np.sum(class_total),
        np.sum(class_correct), np.sum(class_total)))

class_test(model, dataloaders['test'], criterion, dataset.classes)
```

Inferência com image do conjunto de teste

```
def inf_test(model, file, transform, classes):
    file = Image.open(file).convert('RGB')
    img = transform(file).unsqueeze(0)
    with torch.no_grad():
        out = model(img.to(device))
        ps = torch.exp(out)
        top_p, top_class = ps.topk(1, dim=1)
        value = top_class.item()
        print("Value:", value)
        print(classes[value])
        plt.imshow(np.array(file))
        plt.show()
```

```
inf_test(model,
'/content/fruits/fruits-360_dataset/fruits-360/Test/Cocos/0_100.jpg',
data_transforms['test'], dataset.classes)
```

```
inf_test(model,
'/content/fruits/fruits-360_dataset/fruits-360/Test/Mango/0_100.jpg',
data_transforms['test'], dataset.classes)
```

```
inf_test(model, '/content/fruits/fruits-360_dataset/fruits-360/Test/Tomato
4/10_100.jpg', data_transforms['test'], dataset.classes)
```

Inferência com imagens reais

```
!wget
https://img.freepik.com/fotos-premium/vegetais-de-tomate-isolados-no-branco
-trajeto-de-grampeamento-da-fruta-do-tomate-fresco-foto-macro-de-tomate_299
651-601.jpg?w=826
!mv
/content/vegetais-de-tomate-isolados-no-branco-trajeto-de-grampeamento-da-f
ruta-do-tomate-fresco-foto-macro-de-tomate_299651-601.jpg?w=826 tomate.jpg
```

```
inf_test(model, '/content/tomate.jpg', data_transforms['test'],
dataset.classes)
```

Jupyter Notebook: Vision-transformer.ipynb

Author: Marcos Vinicius Satil Medeiros

Modelos de Classificação - Vision Transformers

O objetivo deste notebook é treinar modelos de aprendizado profundo para prever o rótulo/nome de uma determinada fruta tendo apenas uma imagem de entrada.

Bibliotecas utilizadas

```
%%capture
!pip install opendatasets
```

```
import os
import time
import copy
import pandas as pd
import tarfile
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import opendatasets as od
import torch.nn.functional as F
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as tt
from torchvision import datasets,transforms,models
from torchvision.utils import make_grid
from torchvision.datasets.utils import download_url
from torchvision.datasets import ImageFolder
from torchvision.utils import make_grid
from torch.utils.data import *
from torch import nn,optim
from PIL import Image
from transformers import ViTFeatureExtractor, ViTModel # Build up the
pretrained transformers model
from sklearn.metrics import classification_report
```

Baixando dataset do Kaggle com OPENDATASETS

```
od.download('https://www.kaggle.com/moltean/fruits')

data_dir = '/content/fruits/fruits-360_dataset/fruits-360'
print(os.listdir(data_dir))

len(os.listdir(data_dir + '/Training'))
```

Pré-processamento dos dados

Setando device

```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

device = get_default_device()
device

# Set up device and training hyperparameters
latent_dim = 256
epochs = 1
learning_rate = 5e-4
```

Construindo conjunto de dados e dataloader

```
train_dir = data_dir + '/Training'
test_dir = data_dir + '/Test'
batch_size = 256

# Aumento e normalização de dados para treinamento
# Apenas normalização para validação
data_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((224, 224)),
    transforms.Normalize((0.5,)*3, (0.5,)*3)
])

train_dataset = ImageFolder(train_dir, transform=data_transforms)
val_dataset = ImageFolder(test_dir, transform=data_transforms)

# carregando datasets no dataloader
train_dataloader = DataLoader(train_dataset,batch_size=batch_size,
```

```
shuffle=True)
val_dataloader = DataLoader(valid_dataset,batch_size=batch_size,
shuffle=True)
```

Visualizando imagens

```
image_size = 100
show_images, show_labels = next(iter(train_dataloader))
show_images = show_images[:image_size]
show_labels = show_labels[:image_size]

ncols = 8
nrows = int(image_size / ncols) + 1

plt.figure(figsize=(12, 12))
def inverse_normalized(image):
    image = torch.clamp(input=image * 0.5 + 0.5, min=0.0, max=1.0)
    return image


for idx in range(len(show_images)):
    plt.subplot(nrows, ncols, idx + 1)
    plt.axis("off")
    plt.title(train_dataset.classes[show_labels[idx].item()])
    plt.imshow(inverse_normalized(show_images[idx].permute(1, 2, 0)))
plt.tight_layout()
```

Arquitetura do modelo

```
class CategoryClass(nn.Module):
    def __init__(self, vit, latent_dim, classes_):
        super(CategoryClass, self).__init__()
        self.classes_ = classes_
        self.vit = vit
        self.fc_1 = nn.Linear(768, latent_dim)
        self.fc_out = nn.Linear(latent_dim, self.classes_)
        self.dropout = nn.Dropout(0.2)

    def forward(self, in_data):
        vit_outputs = self.vit(in_data)
        pooler_output = vit_outputs.pooler_output
        outputs = torch.relu(self.fc_1(pooler_output))
        outputs = self.fc_out(self.dropout(outputs))
```

```
        return outputs
```

Carregando modelo pré-treinados para Transfer Learning

```
vit = ViTModel.from_pretrained('google/vit-base-patch16-224')
```

Congelando parâmetros da rede para construir classificador

```
for param in vit.parameters():
    param.requires_grad = False

vit.pooler.dense.weight.requires_grad = True
vit.pooler.dense.bias.requires_grad = True

# Criando o modelo completo
model = CategoryClass(vit, latent_dim,
len(train_dataset.classes)).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(model.parameters(), learning_rate)
```

Treinamento do modelo

```
train_loss_history =[]
train_accuracy_history = []
val_loss_history = []
val_accuracy_history = []
batch_loss_history = []
batch_accuracy_history = []

for epoch in range(epochs):
    model.train()
    train_epoch_loss = 0.0
    train_epoch_accuracy = 0.0
    for idx, (images, labels) in enumerate(train_dataloader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_epoch_loss += loss.item()
```

```
        predict_class = outputs.argmax(dim=-1)
        accuracy = torch.sum(predict_class == labels).item() /
labels.shape[0]
        train_epoch_accuracy += accuracy
        batch_loss_history.append(loss.item())
        batch_accuracy_history.append(accuracy)
        # Print informações
        print(f"Batch {idx + 1} in epoch {epoch + 1}/{epochs} \t Average
loss: {loss.item()} \t Average accuracy {accuracy}")

    # Set to the eval mode
    model.eval()
    val_epoch_loss = 0.0
    val_epoch_accuracy = 0.0

    with torch.no_grad():
        for val_images, val_labels in val_dataloader:
            val_images = val_images.to(device)
            val_labels = val_labels.to(device)
            val_outputs = model(val_images)
            val_epoch_loss += criterion(val_outputs, val_labels)
            val_predict_class = val_outputs.argmax(dim=-1)
            val_epoch_accuracy += torch.sum(val_predict_class ==
val_labels).item() / val_labels.shape[0]
    train_loss_history.append(train_epoch_loss / len(train_dataloader))
    train_accuracy_history.append(train_epoch_accuracy /
len(train_dataloader))
    val_loss_history.append(val_epoch_loss / len(val_dataloader))
    val_accuracy_history.append(val_epoch_accuracy / len(val_dataloader))

    # Print informações
    print(f"Epoch {epoch + 1}")
    print(f"Average training loss: {traing_loss_history[-1]}, Average
validation loss: {val_loss_history[-1]}")
    print(f"Average training accuracy: {train_accuracy_history[-1]},
Average validation accuracy: {val_accuracy_history[-1]}")

# Save os pesos do modelp
torch.save(model.state_dict(), "model_transformer.pt")
```

Métricas do modelo

```
# Plot the loss and accuracy
plt.figure(figsize=(18,6))
```

```
# Loss
plt.subplot(1, 2, 1)
plt.title("Loss for batch")
plt.xlabel("Batch")
plt.ylabel("Loss")
plt.plot(batch_loss_history)

# Accuracy
plt.subplot(1, 2, 2)
plt.title("Accuracy for batch")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.plot(batch_accuracy_history)
```

Inferência

Inferência com image do conjunto de teste

```
prediction = []
true_labels = []

with torch.no_grad():
    for val_images, val_labels in val_dataloader:
        val_images = val_images.to(device)
        val_labels = val_labels.to(device)
        val_outputs = model(val_images)
        val_predict_class = val_outputs.argmax(dim=-1)
        prediction.extend([predict_class.item() for predict_class in
val_predict_class])
        true_labels.extend([val_label.item() for val_label in val_labels])

report = classification_report(true_labels, prediction,
                               output_dict=True,
                               target_names=training_dataset.classes)
report_df = pd.DataFrame(report).transpose()

# Show all columns
pd.set_option("display.max_rows", None)
report_df.head()
```

Jupyter Notebook: Classificação_Fast_ai_.ipynb

Author: Marcos Vinicius Satil Medeiros

Modelos de Classificação

O objetivo deste notebook é treinar modelos de aprendizado profundo para prever o rótulo/nome de uma determinada fruta tendo apenas uma imagem de entrada.

O que é fast.ai?

Em notebooks anteriores utilizamos diretamente a biblioteca PyTorch - https://pytorch.org/. Por conta da simplicidade iremos utilizar a biblioteca de alto nível fast.ai.

Fast.ai é uma biblioteca criada por Jeremy Howard e Rachel Thomas dedicada a tornar o poder do Aprendizado Profundo acessível a todos. Para tanto, argumentam que, para esse potencial ser atingido, a tecnologia tem de ser muito mais acessível, intuitiva e fácil de usar.

Referências

- fast.ai—Making neural nets uncool again https://www.fast.ai/
- Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a Ph - The fastai Book - https://github.com/fastai/fastbook
- https://codigos.ufsc.br/aldo.vw/vision/-/tree/master/jupyter

Bibliotecas utilizadas

```
%%capture
!pip install gradio
!pip install opendatasets
!pip install fastai
!pip install timm
```

```
import os
import time
import copy
import random
import pandas as pd
import numpy as np
import tarfile
```

```
import matplotlib
import matplotlib.pyplot as plt
import opendatasets as od
import timm
import fastai
from fastdownload import download_url
from fastcore.all import *
from fastai.vision.all import *
from fastai.interpret import *
from fastai.vision.widgets import *
from pathlib import Path
from PIL import Image
from sklearn.metrics import classification_report

print(fastai.__version__)
```

Baixando dataset do Kaggle com OPENDATASETS

```
od.download('https://www.kaggle.com/moltean/fruits')

data_dir = '/content/fruits/fruits-360_dataset/fruits-360'
print(os.listdir(data_dir))
```

```
['papers', 'LICENSE', 'readme.md', 'Test', 'test-multiple_fruits',
'Training']
```

```
len(os.listdir(data_dir + '/Training'))
```

Setando device

```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

device = get_default_device()
device
```

Carregando o Dataloader

```
def get_dls(path, bs, size):
    dblock = DataBlock(blocks = (ImageBlock, CategoryBlock),
                       get_items = get_image_files,
                       get_y = parent_label,
```

```
                        splitter = RandomSplitter(),
                        item_tfms = Resize(size),
                        batch_tfms=aug_transforms()
                       )
    return dblock.dataloaders(path, bs = bs)

train_dir = data_dir+'/Training'
test_dir = data_dir+'/Test'
```

Visualizando um batch do conjunto de dados

```
dls = get_dls(train_dir, 64, 100) #Definindo Batch_size e Tamanho da Imagem Respectivamente

dls.show_batch(max_n=6)
```

Arquitetura do modelo

Uma lista completa de modelos disponível em:
https://github.com/fastai/fastai/tree/2.5.3/fastai/vision/models

Para mais modelos:
https://github.com/huggingface/pytorch-image-models/tree/main/timm/models

Selecionando a Rede Neural:

- resnet18, resnet34, resnet50, resnet101, resnet152
- squeezenet1_0, squeezenet1_1
- densenet121, densenet169, densenet201, densenet161
- vgg16_bn, vgg19_bn
- alexnet

Definindo Learner

Carregando as inicializações básicas para o aprendizado e definição dos parâmetros básicos da rede.

```
# Definir o array das métricas
metrics = [
    error_rate,
    accuracy,
    F1Score(average="macro"),
    Precision(average="macro"),
```

```
    Recall(average="macro"),
]

learn = vision_learner(dls, resnet34, loss_func = nn.CrossEntropyLoss(),
metrics=metrics)
```

Localizador de taxa de aprendizagem

O localizador de taxa de aprendizagem do fastai adiciona pontos nos locais recomendados.

O que procuramos é um local lógico no gráfico onde a perda está diminuindo. Os pontos vermelhos no gráfico indicam o valor mínimo do gráfico dividido por 10, bem como o ponto mais íngreme.

```
lr_min = learn.lr_find()
suggested_lr = min(lr_min)
```

Agora vamos ajustar o modelo como uma primeira etapa de treinamento.

```
learn.fine_tune(1, base_lr = suggested_lr)
```

Agora que fizemos algum treinamento, precisaremos executar novamente o localizador de taxa de aprendizagem. À medida que o modelo muda e é treinado, podemos encontrar uma nova "melhor" taxa de aprendizagem.

```
lr_min1 = learn.lr_find()
suggested_lr1 = min(lr_min)

learn.fit_one_cycle(1, suggested_lr1)
```

Treinamento do modelo

O que é fit1cycle?

Fit1cycle é uma política de superconvergência desenvolvida por Leslie N. Smith. Está disponível como a política de treinamento preferida no fast.ai. Veja abaixo os detalhes:

- https://docs.fast.ai/callbacks.one_cycle.html
- A disciplined approach to neural network hyper-parameters: Part 1 — learning rate, batch size, momentum, and weight decay — https://arxiv.org/abs/1803.09820

- Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates — https://arxiv.org/abs/1708.07120

Treinamos utilizando a política de adaptação simultânea de taxa de aprendizado e momento linear da rede para superconvergência 1cycle

O método fit_one_cycle() empregado pelo fast.ai funciona com taxas e momentos de aprendizado variados e adaptativos, seguindo uma curva em que a taxa é aumentada e depois diminuída, enquanto o momento é tratado de forma oposta;

Se você interromper um treino na epoch 10 de, digamos, 20 epochs e depois recomeçar por mais 9 epochs, você não terá o mesmo resultado de treinar ininterruptamente por 20 epochs, porque um novo treino do zero, mesmo que você carregue os pesos da última época, empregarão uma nova taxa de aprendizado e política de impulso e passarão pelo ciclo novamente.

O que você quer é começar de onde você foi interrompido no ciclo.

```
epochs = 10  # épocas
lr = suggested_lr1 # Incluir aqui a taxa de aprendizado encontrada no passo anterior
wd = 1e-2  # decaimento da rede

# Train
learn.fit_one_cycle(epochs, lr_max=lr) # wd=wd
```

Visualizando resultados

```
learn.show_results()
```

Salvando o modelo

```
learn.export("modelo_fruits.pkl")
```

Encontrando as maiores perdas (top losses)

```
interp = ClassificationInterpretation.from_learner(learn)
```

Utilizando o modelo em dados reais

Fazendo o dowload de uma imagem para teste

```
!wget
https://upload.wikimedia.org/wikipedia/commons/thumb/f/f7/Lemon_-_whole_and
_split.jpg/1920px-Lemon_-_whole_and_split.jpg
```

Testando modelo com imagem baixada anteriormente.

```
photo_type,_,probs =
learn.predict(PILImage.create('1920px-Lemon_-_whole_and_split.jpg'))
print(f"This is a: {photo_type}.")
print(f"Probability is: {max(probs):.4f}")
```

Fazendo Deploy do Modelo

```
import gradio as gr
from fastai.vision.all import *
import skimage

learn = load_learner('/content/modelo_fruits.pkl')

labels = learn.dls.vocab
def predict(img):
    img = PILImage.create(img)
    pred,pred_idx,probs = learn.predict(img)
    return {labels[i]: float(probs[i]) for i in range(len(labels))}

image_input = gr.Image(height=192, width=192)
label_output = gr.Label()

iface = gr.Interface(
    fn=predict,
    inputs=image_input,
    outputs=label_output,
    examples=['1920px-Lemon_-_whole_and_split.jpg'],
)

iface.launch()
```

Jupyter Notebook: Transformers_Classificação_Fast_ai_.ipynb

Author: Marcos Vinicius Satil Medeiros

Modelos de Classificação

O objetivo deste notebook é treinar modelos de aprendizado profundo para prever o rótulo/nome de uma determinada fruta tendo apenas uma imagem de entrada.

O que é fast.ai?

Em notebooks anteriores utilizamos diretamente a biblioteca PyTorch - https://pytorch.org/. Por conta da simplicidade iremos utilizar a biblioteca de alto nível fast.ai.

Fast.ai é uma biblioteca criada por Jeremy Howard e Rachel Thomas dedicada a tornar o poder do Aprendizado Profundo acessível a todos. Para tanto, argumentam que, para esse potencial ser atingido, a tecnologia tem de ser muito mais acessível, intuitiva e fácil de usar.

Referências

- fast.ai—Making neural nets uncool again https://www.fast.ai/
- Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a Ph - The fastai Book - https://github.com/fastai/fastbook
- https://codigos.ufsc.br/aldo.vw/vision/-/tree/master/jupyter

Bibliotecas utilizadas

```
%%capture
!pip install gradio
!pip install opendatasets
!pip install fastai
!pip install timm
```

```
import os
import time
import copy
import random
import pandas as pd
```

```
import numpy as np
import tarfile
import matplotlib
import matplotlib.pyplot as plt
import opendatasets as od
import timm
import fastai
from fastdownload import download_url
from fastcore.all import *
from fastai.vision.all import *
from fastai.interpret import *
from fastai.vision.widgets import *
from pathlib import Path
from PIL import Image
from sklearn.metrics import classification_report

print(fastai.__version__)
```

Baixando dataset do Kaggle com OPENDATASETS

```
od.download('https://www.kaggle.com/moltean/fruits')

data_dir = '/content/fruits/fruits-360_dataset/fruits-360'
print(os.listdir(data_dir))

len(os.listdir(data_dir + '/Training'))
```

Setando device

```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

device = get_default_device()
device
```

Carregando o Dataloader

```
def get_dls(path, bs, size):
    dblock = DataBlock(blocks = (ImageBlock, CategoryBlock),
                       get_items = get_image_files,
                       get_y = parent_label,
                       splitter = RandomSplitter(),
```

```
                        item_tfms = Resize(size),
                        batch_tfms=aug_transforms()
                       )
    return dblock.dataloaders(path, bs = bs)

train_dir = data_dir+'/Training'
test_dir = data_dir+'/Test'
```

Visualizando um batch do conjunto de dados

```
dls = get_dls(train_dir, 64, 100) #Definindo Batch_size e Tamanho da Imagem Respectivamente

dls.show_batch(max_n=6)
```

Arquitetura do modelo

Uma lista completa de modelos disponível em:
https://github.com/fastai/fastai/tree/2.5.3/fastai/vision/models

Para mais modelos:
https://github.com/huggingface/pytorch-image-models/tree/main/timm/models

Selecionando a Rede Neural:

- Hrnet18, Hrnet32, Hrnet48
- VIT_base_patch, VIT_small_patch
- MobiletnetV3

```
def hrnet18(pretrained=True):
    model = timm.create_model("hrnet_w18_small_v2", pretrained=pretrained)
    return model

def hrnet32(pretrained=True):
    model = timm.create_model("hrnet_w32", pretrained=pretrained)
    return model

def hrnet48(pretrained=True):
    model = timm.create_model("hrnet_w48", pretrained=pretrained)
    return model

def vit_base_patch16_224(num_classes=3, pretrained=True):
    model = timm.create_model("vit_base_patch16_224",
pretrained=pretrained, num_classes=num_classes)
```

```
    return model

def vit_small_patch16_224(num_classes=3, pretrained=True):
    model = timm.create_model("vit_small_patch16_224",
pretrained=pretrained, num_classes=num_classes)
    return model

def mobilenetv3(pretrained=True):
    model = timm.create_model("mobilenetv3_large_100.ra_in1k",
pretrained=pretrained)
    return model

model = hrnet18(pretrained=True)
```

Definindo Learner

Carregando as inicializações básicas para o aprendizado e definição dos parâmetros básicos da rede.

```
# Definir o array das métricas
metrics = [
    error_rate,
    accuracy,
    F1Score(average="macro"),
    Precision(average="macro"),
    Recall(average="macro"),
]

learn = vision_learner(dls, resnet34, loss_func = nn.CrossEntropyLoss(),
metrics=metrics)
```

Localizador de taxa de aprendizagem

O localizador de taxa de aprendizagem do fastai adiciona pontos nos locais recomendados.

O que procuramos é um local lógico no gráfico onde a perda está diminuindo. Os pontos vermelhos no gráfico indicam o valor mínimo do gráfico dividido por 10, bem como o ponto mais íngreme.

```
lr_min = learn.lr_find()
suggested_lr = min(lr_min)
```

Agora vamos ajustar o modelo como uma primeira etapa de treinamento.

```
learn.fine_tune(1, base_lr = suggested_lr)
```

Agora que fizemos algum treinamento, precisaremos executar novamente o localizador de taxa de aprendizagem. À medida que o modelo muda e é treinado, podemos encontrar uma nova "melhor" taxa de aprendizagem.

```
lr_min1 = learn.lr_find()
suggested_lr1 = min(lr_min)

learn.fit_one_cycle(1, suggested_lr1)
```

Treinamento do modelo

O que é fit1cycle?

Fit1cycle é uma política de superconvergência desenvolvida por Leslie N. Smith. Está disponível como a política de treinamento preferida no fast.ai. Veja abaixo os detalhes:

- https://docs.fast.ai/callbacks.one_cycle.html
- A disciplined approach to neural network hyper-parameters: Part 1 — learning rate, batch size, momentum, and weight decay — https://arxiv.org/abs/1803.09820
- Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates — https://arxiv.org/abs/1708.07120

Treinamos utilizando a política de adaptação simultânea de taxa de aprendizado e momento linear da rede para superconvergência 1cycle

O método fit_one_cycle() empregado pelo fast.ai funciona com taxas e momentos de aprendizado variados e adaptativos, seguindo uma curva em que a taxa é aumentada e depois diminuída, enquanto o momento é tratado de forma oposta;

Se você interromper um treino na epoch 10 de, digamos, 20 epochs e depois recomeçar por mais 9 epochs, você não terá o mesmo resultado de treinar ininterruptamente por 20 epochs, porque um novo treino do zero, mesmo que você carregue os pesos da última época, empregarão uma nova taxa de aprendizado e política de impulso e passarão pelo ciclo novamente.

O que você quer é começar de onde você foi interrompido no ciclo.

```
epochs = 1  # épocas
lr = suggested_lr1 # Incluir aqui a taxa de aprendizado encontrada no passo anterior
wd = 1e-2  # decaimento da rede
```

```
# Train
learn.fit_one_cycle(epochs, lr_max=lr) # wd=wd
```

Visualizando resultados

```
learn.show_results()
```

Salvando o modelo

```
learn.export("modelo_fruits.pkl")
```

Encontrando as maiores perdas (top losses)

```
interp = ClassificationInterpretation.from_learner(learn)

interp.plot_top_losses(3, nrows=3, figsize=(8,5))
```

Utilizando o modelo em dados reais

Fazendo o dowload de uma imagem para teste

```
!wget
https://upload.wikimedia.org/wikipedia/commons/thumb/f/f7/Lemon_-_whole_and
_split.jpg/1920px-Lemon_-_whole_and_split.jpg
```

Testando modelo com imagem baixada anteriormente.

```
photo_type,_,probs =
learn.predict(PILImage.create('1920px-Lemon_-_whole_and_split.jpg'))
print(f"This is a: {photo_type}.")
print(f"Probability is: {max(probs):.4f}")
```

Fazendo Deploy do Modelo

```
import gradio as gr
from fastai.vision.all import *
import skimage

learn = load_learner('/content/modelo_fruits.pkl')

labels = learn.dls.vocab
def predict(img):
    img = PILImage.create(img)
```

```
    pred,pred_idx,probs = learn.predict(img)
    return {labels[i]: float(probs[i]) for i in range(len(labels))}

image_input = gr.Image(height=192, width=192)
label_output = gr.Label()

iface = gr.Interface(
    fn=predict,
    inputs=image_input,
    outputs=label_output,
    examples=['1920px-Lemon_-_whole_and_split.jpg'],
)

iface.launch()
```

APÊNDICE 5

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 14 de dez. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para esta entrega, conforme o planejamento, temos os seguinte objetivos:

- ☑ Revisão de artigos.
- ☑ Início da implementação dos modelos de detecção.
- ☑ Obtenção dos dados similares para esta tarefa.
- ☑ Testagem de diferentes de arquiteturas e frameworks.

Os objetivos listados foram incluídos em um documento no seguinte link:
Entrega 14/12- MARCOS VINICIUS SATIL MEDEIROS

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega seguirei os passos feitos no Gate 8 (14/12), onde foi abordada a tarefa de detecção. Dessa forma, teremos os seguinte objetivos:

- Finalização do documento sobre detecção de objetos.
- Estudo dos hiperparâmetros;
- Treinamento por mais épocas e alterando os hiperparâmetros.
- Avaliação dos modelos.
- Atualização do repositório.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

LUANA GUEDES BARROS MARTINS: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 21 de dez. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para esta entrega, conforme o planejamento, temos os seguinte objetivos:

- ☑ Finalização do documento sobre detecção de objetos.
- ☑ Estudo dos hiperparâmetros;
- ☑ Treinamento por mais épocas e alterando os hiperparâmetros.
- ☑ Avaliação dos modelos.
- ☑ Atualização do repositório.

Os objetivos listados foram incluídos em um documento no seguinte link:
Entrega 21/12- MARCOS VINICIUS SATIL MEDEIROS

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega teremos os seguinte objetivos:

- Combinação dos resultados para diferentes tarefas apresentadas.
- Atualização final do repositório.
- Correção de eventuais erros.
- Estudo e elaboração de documento sobre a tarefa de segmentação.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

LUANA GUEDES BARROS MARTINS: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

MODELOS DE DETECÇÃO DE OBJETOS

O objetivo principal desta entrega será explorar a tarefa de detecção de objetos. Dado que já realizamos revisões de literatura em entregas anteriores, neste documento iremos descrever de forma simples alguns conceitos aplicados, assim como as arquiteturas exploradas. O código produzido pode ser encontrado no repositório do github:

https://github.com/marcosvinism/Residencia-em-IA

Este texto é uma versão 2 do documento contido na entrega do dia 14/12: Entrega 14/12- MARCOS VINICIUS SATIL MEDEIROS, pois foram adicionados textos e questões de implementação que não haviam sido feitas/colocadas.

Conceitos sobre detecção de objetos

A detecção de objetos é uma área da Visão Computacional encarregada de determinar a localização de um objeto em uma imagem, juntamente com sua classe correspondente [Zhao et al. 2018]. Em termos gerais, os métodos que facilitam essa tarefa podem ser divididos em dois tipos principais: métodos de disparo único e métodos de dois estágios.

Os métodos de disparo único têm como prioridade o tempo de resposta da inferência. Exemplos desse método incluem os detectores YOLO [Li et al. 2020], SSD [Liu et al. 2015] e RetinaNet [Lin et al. 2017]. Por outro lado, os métodos de dois estágios priorizam a precisão da detecção. Algoritmos notáveis para esse método incluem Fast R-CNN [Girshick 2015], Faster R-CNN [Ren et al. 2015], Mask-RCNN [He et al. 2017], entre outros.

Conceitos e informações sobre a implementação

Frameworks utilizados

A utilização de frameworks é essencial na implementação e desenvolvimento de sistemas de Visão Computacional, proporcionando estruturas robustas e eficientes para a construção de modelos de detecção de objetos. Durante nossos experimentos, foram utilizados três frameworks amplamente reconhecidos e empregados:

Ultralytics:

O Ultralytics é um framework de Visão Computacional que ganhou destaque devido à sua flexibilidade e facilidade de uso. Baseado em PyTorch, o Ultralytics oferece uma ampla gama de funcionalidades para treinamento e inferência de modelos de detecção de objetos. Sua arquitetura modular permite a adaptação a diferentes tarefas e conjuntos de dados, tornando-o uma escolha popular entre pesquisadores e desenvolvedores. Além disso, o Ultralytics é conhecido por sua integração eficiente com algoritmos de última geração, proporcionando uma base sólida para a implementação de soluções inovadoras em Visão Computacional.

Detectron2:

Desenvolvido pelo Facebook AI Research (FAIR), o Detectron2 é um framework de código aberto construído sobre o PyTorch. Ele representa uma evolução significativa em relação ao seu antecessor, o Detectron, oferecendo uma arquitetura mais modular, flexível e de fácil extensão. O Detectron2 fornece implementações eficientes de uma variedade de algoritmos de detecção de objetos, incluindo Faster R-CNN, Mask R-CNN e outros. Sua comunidade ativa e suporte contínuo fazem dele uma escolha sólida para projetos de Visão Computacional em larga escala.

Super Gradients:

O Super Gradients é uma biblioteca de código aberto desenvolvida pelos especialistas em aprendizado profundo da Deci, destinada à comunidade de inteligência artificial para o treinamento de modelos de visão computacional baseados em PyTorch. O SG oferece scripts de treinamento para todas as tarefas mais comumente aplicadas em visão computacional, proporcionando receitas para a rápida e simples reprodução dos resultados de modelos de última geração (SOTA - State-of-the-Art). Além disso, apresenta parâmetros prontos para uso (Plug & Play) para treinamento, conjunto de dados e arquitetura.

Dataset utilizado - Fruit Detection

Para a realização dos nossos experimentos, utilizamos o conjunto de dados Fruit Detection. A obtenção do dataset está disponível em: https://www.kaggle.com/datasets/lakshaytyagi01/fruit-detection/data.

Este conjunto de dados contém um total de 8.479 imagens de 6 frutas diferentes (maçã, uva, abacaxi, laranja, banana e melancia), cada uma com sua contagem total de instâncias listada abaixo.

Class Label	Instance Count
Apple	7049
Grapes	7202
Pineapple	1613
Orange	15549
Banana	3536
Watermelon	1976

Figura 1: Quantidade de instâncias por classe no dataset.

As imagens obtidas estão anotadas no formato YOLO. O seguinte pré-processamento foi aplicado a cada imagem:

- Orientação automática de dados de pixel (com remoção de orientação EXIF)
- Redimensionamento para 640x640

O seguinte aumento foi aplicado para criar 3 versões de cada imagem de origem. As seguintes transformações foram aplicadas às bounding boxes de cada imagem:

- 50% de probabilidade de inversão horizontal

A separação do dados em treino, validação e teste encontra-se no seguinte formato:

Label	Images	Boxes
0	1564	6070
1	1205	2971
2	1453	6027
3	1818	13938
4	601	1372
5	746	1683
Total	7108	32061

Figura 2: Dados de treinamento.

Label	Images	Boxes
0	188	557
1	167	391
2	199	809
3	197	1100
4	77	154
5	107	217
Total	914	3228

Figura 3: Dados de validação.

Label	Images	Boxes
0	113	422
1	88	174
2	94	366
3	92	511
4	40	87
5	47	76
Total	457	1636

Figura 4: Dados de teste.

Exploração de Diferentes Arquiteturas de Modelos

No processo de desenvolvimento dos modelos de detecção, exploramos diversas arquiteturas de modelos renomadas, utilizando as bibliotecas listadas: Ultralytics, Detectron2 e Super Gradients.

Cada arquitetura foi escolhida com base em suas características específicas e no potencial para a tarefa em questão. Destacamos a seguir algumas das arquiteturas investigadas com ênfase para arquitetura YOLO onde aprofundamos a pesquisa:

YOLO:

Na análise referente à evolução da detecção de objetos, Zou et al. (2023) estabelece que o propósito fundamental dessa tarefa é fornecer a compreensão necessária para as aplicações de Visão Computacional, a fim de identificar a localização dos objetos em uma imagem. Os autores enfatizam que, na era do Aprendizado Profundo, iniciada em 2014, o modelo YOLO, proposto em 2015, foi pioneiro como detector de um único estágio que utiliza uma única rede neural na análise completa da imagem. Dessa forma, adotaram um paradigma completamente distinto dos detectores de dois estágios desenvolvidos na mesma época, os quais começavam a tarefa de maneira mais abrangente para otimizar o recall e progrediram para uma segunda etapa visando aprimorar a localização.

Adicionalmente, os autores sublinham a singularidade de cada aplicação específica das tarefas de detecção, destacando que cada contexto apresenta desafios particulares dependendo de suas características, citando objetos pequenos como um exemplo. Nesse cenário, os detectores de um único estágio demonstram uma eficiência elevada em tempo

real em dispositivos móveis e embarcados, contudo, revelam uma redução de desempenho em situações de uso que envolvem objetos de menor porte.

O termo YOLO é uma abreviação para "You only look once" e foi inicialmente introduzido como uma série de arquiteturas e modelos para detecção de objetos. Esses modelos são desenvolvidos utilizando o framework PyTorch como base e representam a pesquisa de código aberto. Os modelos disponíveis foram pré-treinados com a base de dados COCO, que compila imagens de cenas cotidianas complexas contendo objetos em seus contextos naturais. O dataset original passou por diversas atualizações, e a versão utilizada na calibração dos modelos foi a versão de 2017, contendo 123 mil imagens, distribuídas em 118 mil para treinamento e 5 mil para validação. Na etapa de testes, são empregadas 40 mil imagens, as mesmas exatas utilizadas nas fases de treinamento e validação, sem a adição de novas anotações de detecção ou pontos-chave.

Funcionamento

O conceito fundamental por trás do sistema YOLO, concebe a detecção de objetos como um problema de regressão, visando identificar os objetos presentes na imagem e suas respectivas localizações. Nesse contexto, o sistema deixa de lidar diretamente com pixels e passa a operar com coordenadas de bounding boxes e probabilidades de classe. Essa abordagem permite que o sistema trabalhe com a imagem completa durante as fases de treinamento e teste, capturando informações contextuais sobre as classes e suas aparências, ao mesmo tempo em que aprende representações generalizadas dos objetos.

O sistema divide a imagem de entrada em uma grade de células S×S. Se o centro de um objeto estiver contido em uma dessas células, ela é responsável por detectar esse objeto. Cada célula gera previsões de B bounding boxes, junto com o grau de confiança associado. Se nenhum objeto estiver presente na célula, o grau de confiança é estabelecido como 0, e nenhuma probabilidade de classe é gerada. No entanto, caso haja a presença de um objeto, são geradas previsões de C probabilidades de classe, sendo que cada célula representa uma única classe, independentemente do número de bounding boxes presentes.

Visão geral da arquitetura YOLO

A arquitetura deste modelo consiste em três camadas principais: backbone, neck e head. Antes de nos aprofundarmos, é importante compreender as características e responsabilidades de cada uma dessas camadas, as quais estão explicadas a seguir.

A camada de Backbone é encarregada da extração de características das imagens de entrada durante o treinamento. Dado que essa fase pode ser computacionalmente intensiva, é possível utilizar modelos pré-treinados em conjuntos de dados abrangentes e diversos para facilitar a transferência de aprendizado. Essa transferência, por analogia, instrui a rede neural a reconhecer objetos de maneira geral, mesmo a partir de imagens que representam classes genéricas. A tarefa da etapa de treinamento, então, é ensinar à rede neural como aplicar essa habilidade ao contexto específico.

A camada Neck é uma camada intermediária responsável pela coleta de mapas de características em diferentes estágios de modelos de detecção de objetos. Geralmente, esses mapas são compostos por caminhos bottom-up e top-down, otimizando o desempenho e a propagação de informações semânticas mais eficazes. Isso é realizado aproveitando a capacidade dos neurônios de camada para serem ativados por percepções distintas, desde objetos completos até a identificação de texturas e padrões locais.

Quanto à camada Head, ela é responsável pela saída final do modelo, relacionada à classificação das classes. No contexto da detecção de objetos, os autores ressaltam que a qualidade e distribuição das previsões da rede neural são tipicamente consideradas. Essas previsões são combinadas com outras fontes de verdade para calcular e otimizar a definição das classes, em linha com a abordagem do Yolo, que utiliza métricas de regressão nas previsões das bounding boxes para determinar as classes dos objetos.

Configurações de Treinamento

A YOLO disponibiliza alguns modelos pré-treinados, mas também oferece a flexibilidade de ser utilizado com outros métodos de treinamento e configurações personalizadas. A abordagem padrão e recomendada para bases de dados pequenas e médias é realizar o que é conhecido como Transfer Learning, utilizando os pesos dos modelos pré-treinados.

O Transfer Learning busca aprimorar o desempenho de novos modelos em treinamento ou reduzir a necessidade de exemplos rotulados por meio da transferência de

conhecimento entre domínios. Esse conceito está possivelmente vinculado à teoria da generalização da transferência, proposta pelo psicólogo C.H. Judd. A teoria sugere que a transferência do aprendizado resulta da generalização da experiência, contanto que exista uma conexão entre as duas atividades. Embora pareça promissor, nem sempre proporciona benefícios positivos e pode resultar em transferência negativa. Evitar esse cenário depende da relevância entre os domínios e da capacidade do modelo de selecionar a parte do conhecimento que será benéfica.

O primeiro método recomendado pelos autores é utilizar os pesos dos modelos pré-treinados em bases de origem, como COCO, por exemplo, e carregar esses pesos e as características aprendidas anteriormente para o processo de treinamento em um novo domínio. Para implementar essa abordagem, define-se o parâmetro "weights" com o nome do modelo pré-treinado. O segundo método, indicado para bases de dados maiores, envolve um treinamento do zero, sem nenhuma etapa de aprendizado de localização e classificação. Para essa abordagem, o parâmetro "weights" é configurado com um valor vazio.

Além das duas alternativas mencionadas anteriormente, uma terceira opção de treinamento envolve o congelamento de camadas na rede neural. Nessa abordagem, os pesos previamente treinados na base de origem permanecem inalterados, não sendo reajustados, e são generalizados para o novo domínio. Apenas as demais camadas e classificadores finais são otimizados conforme as funções de perda. Os experimentos conduzidos pelo autor sugerem que o congelamento parcial resulta em uma redução significativa no tempo de treinamento, com uma leve diminuição no desempenho do modelo. Essa estratégia é adequada para consumir menos recursos computacionais, especialmente em modelos de maior escala ou com imagens maiores.

Hiperparâmetros

Os principais hiperparâmetros para a inicialização do treinamento são os parâmetros lr0, lrf, momentum e weight decay que controlam a taxa de aprendizagem do SGD (Stochastic Gradient Descent), que é o otimizador padrão para bases de dados personalizadas. Os parâmetros de aquecimento são empregados para gradualmente aumentar os valores iniciais e estabilizar o treinamento. No contexto do treinamento com BCE (Binary Cross Entropy) para classificação e detecção de objetos, são utilizados os

parâmetros de taxa de perda, enquanto iou_t e anchor_t são empregados para a localização dos objetos existentes.

Também são incluídos parâmetros associados à aplicação de técnicas de aumento de dados. Essas técnicas são aplicadas durante o carregamento de imagens pelo modelo durante o treinamento. Os primeiros a serem ativados por padrão incluem hsv_h, hsv_s e hsv_v, que ajustam, respectivamente, um percentual da matiz (H), saturação (S) e valor ou brilho (V) da imagem. Outros parâmetros ativos por padrão são: translate, que desloca uma porção de pixels da imagem; scale, que aplica um percentual de aumento ou diminuição da imagem; fliplr, que aplica inversões horizontais; e mosaic, que aplica uma probabilidade de criar um painel com quatro imagens.

Os demais parâmetros, listados na mesma categoria, estão desativados por padrão e incorporam outras técnicas, tais como: degrees, que aplica um grau de rotação à imagem; shear, que aplica um grau de cisalhamento à imagem; perspective, que aplica um grau de angulação à imagem; flipud, que ativa uma probabilidade de aplicar inversões verticais; mixup, que ativa uma probabilidade de mesclar duas imagens sobrepostas; copy_paste, que ativa uma probabilidade de incluir partes recortadas de imagens em outras imagens aleatórias, gerando novas amostras.

Hiperparâmetro	Valor	Descrição
lr0	0,01	Taxa de aprendizagem inicial (SGD=1e −2)
lrf	0,01	Taxa de aprendizagem final OneCycleLR (lr0 × lrf)
momentum	0,937	SGD momentum
weight_decay	0,0005	Taxa de queda dos pesos do otimizador 5e −4
warmup_epochs	3,0	Épocas de aquecimento (permite valores fracionados)
warmup_momentum	0,8	Momentum de aquecimento inicial
warmup_bias_lr	0,1	Viés de taxa de aprendizado aquecimento inicial
box	0,05	Taxa de perda dos bounding boxes
cls	0,5	Taxa de perda das classes

cls_pw	1,0	Taxa de perda BCE das classes (peso positivo)
obj	1,0	Taxa de perda dos objetos (escala com pixels)
obj_pw	1,0	Taxa de perda BCE dos objetos (peso positivo)
iou_t	0,2	Limiar de treinamento IoU
anchor_t	4,0	Limiar Multiplicador de âncoras

Tabela 1: Hiperparâmetros de inicialização utilizados nos modelos.

Parâmetro	Valor	Descrição
hsv_h	0,015	Ajuste parcial de Matiz (Hue)
hsv_s	0,7	Ajuste parcial de saturação (Saturation)
hsv_v	0,4	Ajuste parcial de Valor (brilho) (Value)
translate	0,1	Porção de pixels de deslocamento (+/-)
scale	0,5	Porção de escala (+/-)
fliplr	0,5	Probabilidade de inversão (Horizontal)
mosaic	1,0	Probabilidade de criação de mosaicos (4 imagens)
degrees	0,0	Grau de rotação (+/-)
shear	0,0	Grau de cisalhamento (+/-)
perspective	0,0	Grau de angulação de perspectiva (+/-)
flipud	0,0	Probabilidade de inversão (Vertical)
mixup	0,0	Probabilidade de mesclar duas imagens sobrepostas
copy_paste	0,0	Probabilidade de inclusão de partes de imagens em outras

Tabela 2: Parâmetros de data augmentation.

Novas Versões da Família YOLO

A família YOLO continua a ser objeto de estudo por diversos pesquisadores, resultando em novas publicações que introduzem métodos e abordagens inovadoras. Estas contribuições foram a base para a revisão realizada e merecem destaque, incluindo o YOLOR, YOLOX , DAMO-YOLO, PP-YOLO e, mais recentemente, em maio de 2023, o YOLO-NAS. Devido às restrições de tempo para a condução de experimentos, não foi possível incluir todas essas versões neste trabalho. No entanto, é relevante mencionar as versões YOLOv5, YOLOv6 e YOLOv7, que continuam a evoluir e incorporam novas funcionalidades promissoras que também influenciam outras variantes.

Figura 5: Linha do tempo da família YOLO.

YOLOv8:

O YOLOv8, versão mais recente desses modelos, foi lançado oficialmente em janeiro de 2023. Representando um modelo State-of-the-Art (SOTA) de código aberto, o YOLOv8 é mantido pela equipe da Ultralytics, distribuído sob a Licença Pública Geral GNU, permitindo a compartilhamento, modificação e distribuição do software.

Diferentemente dos modelos anteriores, o YOLOv8 demonstra um desempenho significativamente superior, superando não apenas o YOLOv5, mas também os modelos YOLOv7 e YOLOv6 (Glenn Jocher, A. Chaurasia e J. Qiu, 2023). Com cinco versões

escalonadas - YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large) e YOLOv8x (extra large) - o YOLOv8 oferece suporte a diversas tarefas de visão, incluindo detecção de objetos, segmentação, estimativa de pose, rastreamento e classificação.

Ao comparar os modelos YOLOv8 com os YOLOv5 da Ultralytics, observamos melhorias significativas de desempenho. Com um desempenho excepcionalmente rápido, arquitetura simplificada e abordagem única de regressão e predição de máscaras, o YOLOv8 continua a ser uma referência na área de visão computacional, demonstrando uma evolução notável em relação às iterações anteriores da família YOLO.

YOLO-NAS

Lançado em maio de 2023 pela Deci, YOLO-NAS é uma arquitetura pioneira no domínio da detecção de objetos, estabelecendo padrões incomparáveis no equilíbrio de precisão e latência. Ele incorpora blocos QSP e QCI, que combinam os benefícios da re-parametrização e da quantização de 8 bits, garantindo degradação mínima da precisão durante a quantização pós-treinamento. A tecnologia proprietária de pesquisa de arquitetura neural (NAS) da Deci, AutoNAC, foi fundamental na geração da arquitetura YOLO-NAS. Ele determinou os tamanhos ideais, tipos de blocos, número de blocos e contagens de canais em cada estágio.

Projetado com a produção em mente, o YOLO-NAS é adaptado para integração perfeita com mecanismos de inferência de ponta, como NVIDIA® TensorRT™, e suporta quantização INT8 para desempenho de tempo de execução incomparável. Sua adaptabilidade o posiciona como uma escolha ideal para aplicações do mundo real que exigem baixa latência e processamento robusto, como veículos autônomos, robótica e análise de vídeo.

A arquitetura inova ainda mais ao empregar técnicas de última geração, como mecanismos de atenção, blocos com reconhecimento de quantização e reparametrização durante a inferência. Esses elementos capacitam o YOLO-NAS a detectar objetos com habilidade, independentemente de seu tamanho ou complexidade, redefinindo a excelência no cenário de detecção de objetos e sua aplicabilidade em vários setores.

ALTERNATIVAS A YOLO

Nos últimos anos, tem-se observado um aumento exponencial na publicação de estudos relacionados à detecção de objetos, resultando em significativo progresso na área de visão computacional. Ao longo da última década, identificam-se marcos relevantes, como transição dos métodos tradicionais para a adoção de abordagens com Deep Learning e os métodos de dois estágios, mas também destaca-se outro marco: a detecção de objetos sem o uso de âncoras, que evoluiu em duas fases distintas. Inicialmente, empregou-se uma estratégia de regressão para prever as coordenadas dos bounding boxes com base nas características identificadas, sendo o YOLO um dos protagonistas desse período. A segunda fase trouxe a utilização de transformers, superando as limitações das CNNs e alcançando um campo de receptividade em escala global. Com isso temos as seguintes alternativas ao uso de redes em um único estágio:

Faster R-CNN

Faster R-CNN é um algoritmo de detecção de objetos proposto por Shaoqing Ren, Kaiming He, Ross Girshick e Jian Sun em 2015. A R-CNN mais rápida baseia-se em trabalhos anteriores para classificar com eficiência propostas de objeto usando redes convolucionais profundas..

A principal inovação do Faster R-CNN reside na introdução de uma rede neural totalmente convolucional chamada Região de Proposição (Region Proposal Network - RPN), que eficientemente propõe regiões candidatas para objetos dentro de uma imagem. Essas regiões propostas são então avaliadas por uma rede convolucional para determinar a presença de objetos e classificá-los.

A arquitetura do Faster R-CNN é composta por três etapas principais:

- Extração de Características: Uma rede neural convolucional é empregada para extrair características da imagem de entrada. Essas características são essenciais para a subsequente detecção de objetos.

- Region Proposal Network (RPN): Uma rede específica, o RPN, opera sobre as características extraídas e propõe regiões que possivelmente contêm objetos. Essas regiões são sugeridas com base em ancoragens (anchors),

que são caixas delimitadoras predefinidas de diferentes escalas e razões de aspecto.

- Detecção de Objetos: As regiões propostas são então refinadas e classificadas por meio de duas cabeças diferentes da rede: uma responsável por ajustar as coordenadas da caixa delimitadora (bounding box regression) e outra para atribuir uma classe específica a cada região proposta.

Essa abordagem modular do Faster R-CNN, com o uso do RPN, supera as limitações de métodos anteriores que dependiam de algoritmos externos para gerar propostas de regiões. O RPN é treinado em conjunto com o restante da rede, permitindo uma aprendizagem mais integrada e end-to-end. O Faster R-CNN demonstrou ser altamente preciso e eficiente em tarefas de detecção de objetos, tornando-se uma referência na área.

DETR e Deformable DETR

Na era mais recente dos detectores sem a necessidade de âncoras, dois estudos destacados são os modelos DETR e Deformable DETR. O DETR aborda a detecção de objetos como um problema de conjunto de previsões de ponta a ponta, combinando uma CNN com uma arquitetura de transformers encoder-decoder para gerar o conjunto de previsões dos bounding boxes. Na última fase do treinamento, emprega-se a técnica bipartite matching loss para ajustar as previsões aos bounding boxes reais. Esse modelo alcança resultados notáveis em bases de dados desafiadoras, como a COCO, utilizando uma arquitetura simplificada, eliminando a necessidade de acoplar vários componentes de maneira manual.

Em seguida, o Deformable DETR introduz melhorias relacionadas às deficiências do Transformer na utilização dos mapas de atenção. Essa abordagem compromete o desempenho na detecção de objetos menores e requer uma verificação de todas as localizações espaciais, aumentando o tempo de convergência e a complexidade espacial com os dados de entrada dos mapas de características. O Deformable Attention Module

concentra-se em um pequeno grupo de amostras de pontos-chave em torno de um ponto de referência, independentemente do tamanho dos mapas de características.

Treinamento e Avaliação dos Modelos

O treinamento dos modelos foi conduzido utilizando as bibliotecas previamente mencionadas. Após a conclusão deste processo, procedemos à avaliação dos modelos, fundamentando-nos em métricas de detecção de objetos. Essa abordagem visa proporcionar uma compreensão mais profunda do desempenho alcançado, permitindo uma análise de sua eficácia.

Métricas utilizadas

Diversas métricas são empregadas para medir a precisão, a robustez e a generalização desses modelos. Entre as principais métricas de detecção de objetos, destacam-se também algumas das métricas amplamente reconhecidas para problemas de classificação:

Precisão (Precision): A precisão é uma métrica que avalia a proporção de detecções corretas em relação ao total de detecções feitas pelo modelo. Ela é calculada pela divisão do número de verdadeiros positivos pelo somatório dos verdadeiros positivos e falsos positivos. A precisão fornece uma medida da confiabilidade das detecções realizadas.

Revocação (Recall): A revocação, também conhecida como sensibilidade, expressa a capacidade do modelo em identificar todos os objetos de uma determinada classe presentes na imagem. É calculada pela divisão do número de verdadeiros positivos pelo somatório dos verdadeiros positivos e falsos negativos. A revocação é especialmente importante quando se busca um equilíbrio entre a detecção de todos os objetos de interesse.

F1-Score: O F1-Score é uma métrica que combina precisão e revocação, proporcionando uma visão equilibrada do desempenho do modelo. Calculado a partir da média harmônica entre precisão e revocação, o F1-Score é particularmente útil quando se

deseja avaliar um modelo de detecção em que tanto falsos positivos quanto falsos negativos têm impacto significativo.

IoU (Intersection over Union): A IoU é uma métrica que mede a sobreposição entre a bounding box predita e a bounding box verdadeira do objeto. É calculada como a área da interseção entre as duas bounding boxes dividida pela área da sua união. A IoU é frequentemente utilizada para definir se uma detecção é considerada verdadeira ou falsa, sendo um critério crucial em avaliações de detecção de objetos.

AP (Average Precision): A Average Precision é uma métrica que avalia a qualidade das detecções em várias classes. Calcula a área sob a curva de precisão em função da revocação para diferentes valores de threshold de confiança. O AP é frequentemente utilizado para comparar o desempenho de diferentes modelos de detecção.

mAP (Mean Average Precision): O mAP é a média das Average Precisions calculadas para cada classe. Essa métrica proporciona uma avaliação global do desempenho do modelo em detecções multiclasse. O mAP é amplamente utilizado em competições e benchmarks de visão computacional.

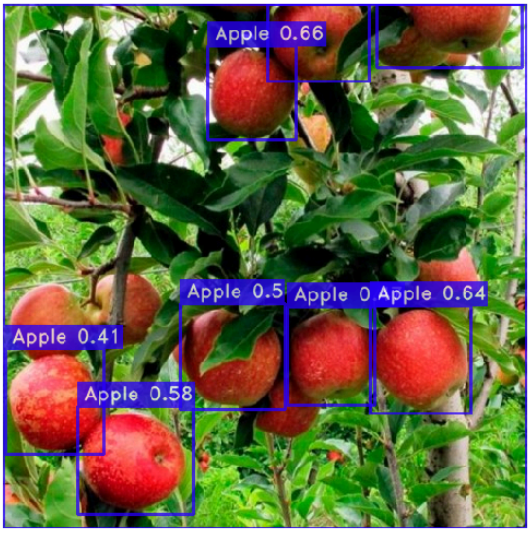
Tempo de Inferência: Além das métricas tradicionais, o tempo de inferência é uma métrica prática que avalia o desempenho temporal do modelo durante o processo de detecção. Em aplicações em tempo real, a eficiência temporal é crucial, e modelos mais rápidos são preferidos.

Resultados e discussões

O principal objetivo foi comparar e avaliar a eficácia de cada arquitetura na tarefa de detecção de objetos em um dataset para detecção de frutas. Essa abordagem possibilitou a exploração de diversas técnicas de Deep Learning, contribuindo para uma compreensão mais aprofundada do tema.

A seguir temos os resultados gerados ao fazer a inferência em imagens reais coletados do Google Images:

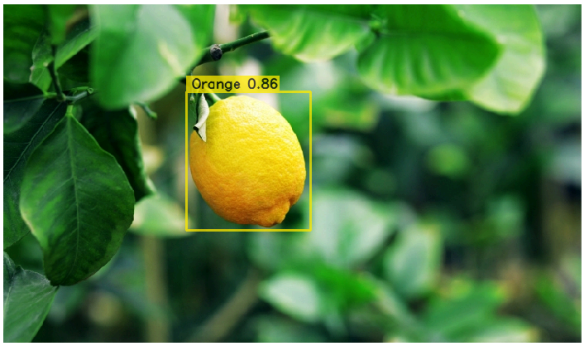
Apple

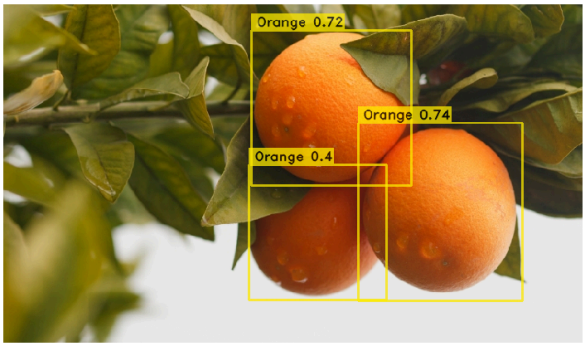


Banana

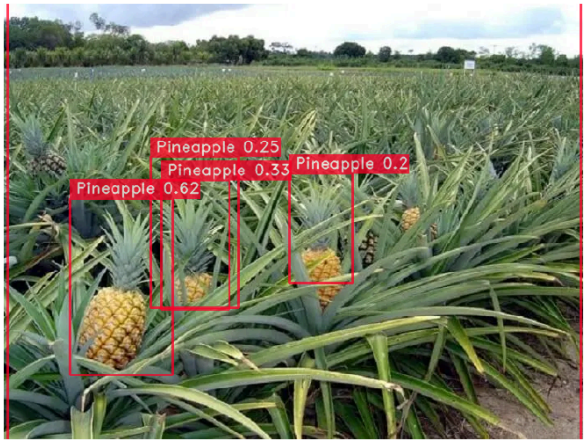
Grape

Orange

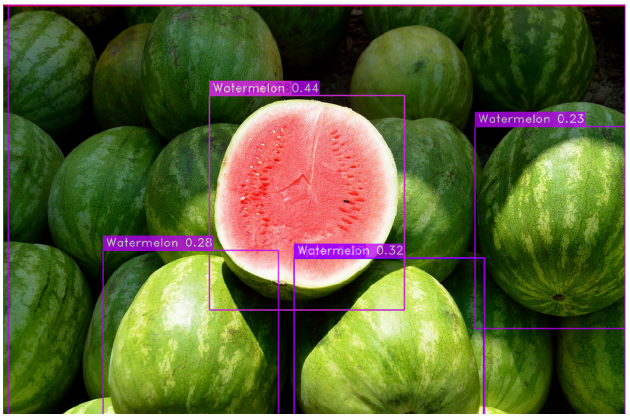




Pineapple



Watermelon



De acordo com a abordagem proposta, diversas técnicas e ajustes foram implementados em diferentes cenários, visando melhorar o desempenho do conjunto de dados e do modelo, tornando-os mais adequados para uma aplicação prática no reconhecimento e detecção de frutas. O processo experimental foi progressivo e inicialmente compreendeu três fases, incorporando aprimoramentos ao modelo-base, sendo testadas as versões n, m, l da Yolov8 com diferentes épocas e hiperparâmetros.

À medida que os estudos sobre o modelo e o domínio de detecção de objetos foram aprofundados, novos experimentos foram conduzidos para explorar as características e possibilidades de outras versões. Apesar de resultados ainda ruins, é possível obter muitos insights e possíveis pontos de melhorias, pois o foco deste trabalho foi o aprendizado do

processo end-to-end para a tarefa de detecção, indo desde a obtenção dos dados até a avaliação dos modelos gerados.

Referências

Zhao, Z., Zheng, P., Xu, S., and Wu, X. (2018). Object detection with deep learning: A review. CoRR, abs/1807.05511. Disponível em: https://arxiv.org/abs/1807.05511

Li, X., Tian, M., Kong, S., Wu, L., and Yu, J. (2020). A modified yolov3 detection method for vision-based water surface garbage capture robot. International Journal of Advanced Robotic Systems, 17(3):1729881420932715.

Lin, T., Goyal, P., Girshick, R. B., He, K., and Dollar, P. (2017). Focal loss for dense object detection. CoRR, abs/1708.02002. Disponível em: https://arxiv.org/abs/1708.02002

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. (2015). SSD: single shot multibox detector. CoRR, abs/1512.02325. Disponível em: https://arxiv.org/abs/1512.02325

Girshick, R. B. (2015). Fast R-CNN. CoRR, abs/1504.08083. Disponível em: https://arxiv.org/abs/1504.08083

He, K., Gkioxari, G., Dollar, P., and Girshick, R. B. (2017). Mask R-CNN. CoRR, abs/1703.06870. Disponível em: https://arxiv.org/abs/1703.06870

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. Advances in Neural Information Processing Systems, 28. Disponivel em: https://arxiv.org/abs/1506.01497

REDMON, J. et al. You only look once: Unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society, 2016. p. 779–788. ISSN 1063-6919. Disponível em: https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.91

Sovit Rath. YOLOv8 Ultralytics: State-of-the-Art YOLO Models. LearnOpenCV, 2023. Acesso em junho de 2015. Disponível em: https://learnopencv.com/ultralytics-yolov8/

Jupyter Notebook: Yolov8.ipynb

Author: Marcos Vinicius Satil Medeiros

Modelos de Detecção

O objetivo deste notebook é treinar modelos de aprendizado profundo para identificar e localizar a presença de uma determinada fruta em uma imagem de entrada.

Bibliotecas utilizadas

```
%%capture
!pip install ultralytics
!pip install opendatasets
```

```
import os
import cv2
import yaml
import json
import shutil
import requests
import numpy as np
import torch
import random
import pathlib
import ultralytics
import matplotlib.pyplot as plt
from PIL import Image
from torch.utils.data import Dataset
from torchvision import transforms, utils
from pathlib import Path
from ultralytics import YOLO

ultralytics.checks()
```

Baixando dataset do Kaggle com OPENDATASETS

```
import opendatasets as od

od.download('https://www.kaggle.com/datasets/lakshaytyagi01/fruit-detection
')
```

Configurações

```
class config:
    #trainer params
    CHECKPOINT_DIR = 'checkpoints' #specify the path you want to save
checkpoints to
    EXPERIMENT_NAME = 'fruit-detector' #specify the experiment name
    #dataset params
    DATA_DIR = '/content/fruit-detection/Fruits-detection/' #parent
directory to where data lives
    TRAIN_IMAGES_DIR = 'train/images' #child dir of DATA_DIR where train
images are
    TRAIN_LABELS_DIR = 'train/labels' #child dir of DATA_DIR where train
labels are
    VAL_IMAGES_DIR = 'valid/images' #child dir of DATA_DIR where validation
images are
    VAL_LABELS_DIR = 'valid/labels' #child dir of DATA_DIR where validation
labels are
    # if you have a test set
    TEST_IMAGES_DIR = 'test/images' #child dir of DATA_DIR where test
images are
    TEST_LABELS_DIR = 'test/labels' #child dir of DATA_DIR where test
labels are
    CLASSES = ['Apple', 'Banana', 'Grape', 'Orange', 'Pineapple',
'Watermelon'] #what class names do you have
    NUM_CLASSES = len(CLASSES)
    # model params
    MODEL_NAME = 'yolov8x' # choose from yolov8n, yolov8m, yolov8x


def arq_yaml():
    #define YAML file information
    yml = {"names": config.CLASSES,
           "nc": len(config.CLASSES),
           "path": config.DATA_DIR,
           "test": config.TEST_IMAGES_DIR,
           "train": config.TRAIN_IMAGES_DIR,
           "val": config.VAL_IMAGES_DIR}
    #generate YAML file
    yaml_path = os.path.join(config.DATA_DIR, 'data.yaml')  #path to the
YAML file
    with open(yaml_path, "w") as file:
        yaml.dump(yml, file)

arq_yaml()
```

Explorando imagens

```
train_images_dir= os.path.join(config.DATA_DIR, config.TRAIN_IMAGES_DIR)
train_labels_dir= os.path.join(config.DATA_DIR, config.TRAIN_LABELS_DIR)
# Number of images to randomly select
num_images = 5
# Get the list of all image files in the 'images' directory
image_files = [f for f in pathlib.Path(train_images_dir).iterdir() if
f.is_file()]
# Shuffle the list of image files
random.shuffle(image_files)

# Select the specified number of image files
selected_image_files = image_files[:num_images]

for selected_image_file in selected_image_files:
    demo_image = selected_image_file
    # Get the corresponding label file
    demo_label = pathlib.Path(train_labels_dir) /
f"{selected_image_file.stem}.txt"
    # Load the image using OpenCV's imread function
    image = cv2.imread(str(demo_image))
    # Get the list of class names from the 'data' dictionary
    class_list = config.CLASSES
    # Define a list of colors to be used to draw bounding boxes
    colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (0,
255, 255), (255, 0, 255)]
    # Get the height and width of the image
    height, width, _ = image.shape
    # Create an empty list T
    T = []
    # Open the label file 'demo_label' in read mode and process each line
    with open(demo_label, "r") as file1:
        for line in file1.readlines():
            # Split the line into a list of strings
            split = line.split(" ")
            # Get the class id from the first element of the split list
            class_id = int(split[0])
            # Get the color corresponding to the class id from the 'colors'
list
            color = colors[class_id]
            clazz = class_list[class_id]
            # Get the x, y, w, h bounding box coordinates from the split
list
```

```
            x, y, w, h = float(split[1]), float(split[2]), float(split[3]),
float(split[4])
            # Rescale the x, y, w, h values to the size of the image
            box = [int((x - 0.5*w)* width), int((y - 0.5*h) * height),
int(w*width), int(h*height)]
            # Draw a rectangle on the image using the 'box' and 'color'
values
            cv2.rectangle(image, box, color, 2)
            # Draw a filled rectangle for the class label on the image
            cv2.rectangle(image, (box[0], box[1] - 20), (box[0] + box[2],
box[1]), color, -1)
            # Write the class label on the image
            cv2.putText(image, class_list[class_id], (box[0], box[1] - 5),
cv2.FONT_HERSHEY_SIMPLEX, .5, (0,0,0))
        # Show the image using matplotlib
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        # Optionally resize the image (commented out in code)
        image = cv2.resize(image, (600, 600))
        plt.show()
```

Fine-tuning YOLOv8

Treinamento do modelo

```
config_dict = dict(data = os.path.join(config.DATA_DIR, 'data.yaml'),
                   epochs = 10,
                   batch = 16,
                   workers = 4,
                   imgsz = 640,
                   task = 'detect')

def train(model_variant, config_dict):
    #define model
    model = YOLO(f"{model_variant}.pt")
    #train
    results = model.train(**config_dict)

train(config.MODEL_NAME, config_dict)
```

Avaliando modelos treinado no conjunto de teste

```
path_model = "/content/runs/detect/train4/weights/best.pt"
best_model = YOLO(path_model)
```

```
path_test = '/content/fruit-detection/Fruits-detection/test/images'
evaluate = best_model.predict(path_test, save=True, imgsz=640, conf=0.4)
```

Inferência

```
def show_result(result):
    boxes = result[0].boxes.xyxy.cpu().numpy()
    names = result[0].names
    confidences = result[0].boxes.conf

    # Convertendo a imagem para o formato RGB (se estiver em BGR)
    img = result[0].orig_img
    if img.shape[-1] == 3:  # Verifica se a imagem tem 3 canais (BGR)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Plotar as caixas delimitadoras na imagem
    for box, name, confidence in zip(boxes, names, confidences):
        if confidence >= 0.4:  # Exibir apenas caixas com confiança maior
que 0.4
            box = [int(coord) for coord in box]
            plt.plot([box[0], box[2], box[2], box[0], box[0]], [box[1],
box[1], box[3], box[3], box[1]], label=f'{name} ({confidence:.2f})',
linewidth=2)

    # Mostrar a imagem com as caixas delimitadoras
    plt.imshow(img)
    plt.axis('off')
    plt.show()
```

Apple

```
url =
'https://s2-g1.glbimg.com/hWMKfZrVwHIPJJMgslhvGEuRKGw=/0x0:1920x1080/984x0/
smart/filters:strip_icc()/i.s3.glbimg.com/v1/AUTH_59edd422c0c84a879bd37670a
e4f538a/internal_photos/bs/2018/i/S/DJzZRhQWyub4r7FnBZrA/nc-maca-safra-1811
18.jpg'
result = best_model.predict(source=url,  conf=0.6)

show_result(result)
```

Banana

```
url =
'https://blog4.mfrural.com.br/wp-content/uploads/2020/06/plantar-banana.jpg
'
result = best_model.predict(source=url,  conf=0.4)

show_result(result)
```

Grape

```
url =
'https://ocontadordecervejas.com.br/wp-content/uploads/thumbnails/quanto-te
mpo-leva-para-um-pe-de-uva-dar-frutos.jpg'
result = best_model.predict(source=url,  conf=0.4)

show_result(result)
```

Orange

```
url =
'https://www.nt4gbi.com/wp-content/uploads/2023/04/Como-saber-se-e-pe-de-la
ranja-ou-limao.jpg'
result = best_model.predict(source=url,  conf=0.6)

show_result(result)

url =
'https://www.nt4gbi.com/wp-content/uploads/2023/04/Quanto-tempo-leva-um-pe-
de-laranja-para-dar-frutos.jpg'
result = best_model.predict(source=url,  conf=0.6)

show_result(result)
```

Pineapple

```
url =
'https://imagens-revista.vivadecora.com.br/uploads/2021/08/Aprenda-dicas-e-
truques-de-como-plantar-abacaxi-em-casa.-Foto-MF-Magazine-MF-Rural.jpg'
result = best_model.predict(source=url,  conf=0.6)

show_result(result)
```

Watermelon

```
url =
'https://alavoura.com.br/wp-content/uploads/2020/09/Agricultores-investem-no-cultivo-irrigado-de-melancia-como-alternativa-de-renda-no-Norte-de-MT-1-e1599056081966.jpg'
result = best_model.predict(source=url,  conf=0.6)

show_result(result)
```

Salvando modelo no drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# caminho para o modelo no Colab
colab_model_path = '/content/runs' #ckpt_best, ckpt_latest, average_model
# caminho para salvar o modelo no Google Drive
drive_model_path = '/content/drive/MyDrive/TCC/Pesos/Yolov8'
# copiando o modelo do Colab para o Google Drive
shutil.copytree(colab_model_path, drive_model_path)

print(f'Modelo salvo com sucesso em {drive_model_path}')
```

```
Modelo salvo com sucesso em /content/drive/MyDrive/New
```

Jupyter Notebook: Yolonas.ipynb

Author: Marcos Vinicius Satil Medeiros

Modelos de Detecção

O objetivo deste notebook é treinar modelos de aprendizado profundo para identificar e localizar a presença de uma determinada fruta em uma imagem de entrada.

Bibliotecas utilizadas

```
%%capture
!pip install super-gradients
!pip install opendatasets
```

```
import os
import cv2
import json
import shutil
import requests
import numpy as np
import torch
import random
import pathlib
import matplotlib.pyplot as plt
from PIL import Image
from torch.utils.data import Dataset
from torchvision import transforms, utils
from pathlib import Path
from super_gradients.training import Trainer, dataloaders, models
from super_gradients.training.dataloaders.dataloaders import (
    coco_detection_yolo_format_train, coco_detection_yolo_format_val)
from super_gradients.training.losses import PPYoloELoss
from super_gradients.training.metrics import DetectionMetrics_050
from super_gradients.training.models.detection_models.pp_yolo_e import (
    PPYoloEPostPredictionCallback)
```

Baixando dataset do Kaggle com OPENDATASETS

```
import opendatasets as od
```

```
od.download('https://www.kaggle.com/datasets/lakshaytyagi01/fruit-detection
')
```

Configurações

```
class config:
    #trainer params
    CHECKPOINT_DIR = 'checkpoints' #specify the path you want to save
checkpoints to
    EXPERIMENT_NAME = 'fruit-detector' #specify the experiment name
    #dataset params
    DATA_DIR = '/content/fruit-detection/Fruits-detection' #parent
directory to where data lives
    TRAIN_IMAGES_DIR = 'train/images' #child dir of DATA_DIR where train
images are
    TRAIN_LABELS_DIR = 'train/labels' #child dir of DATA_DIR where train
labels are
    VAL_IMAGES_DIR = 'valid/images' #child dir of DATA_DIR where validation
images are
    VAL_LABELS_DIR = 'valid/labels' #child dir of DATA_DIR where validation
labels are
    # if you have a test set
    TEST_IMAGES_DIR = 'test/images' #child dir of DATA_DIR where test
images are
    TEST_LABELS_DIR = 'test/labels' #child dir of DATA_DIR where test
labels are
    CLASSES = ['Apple', 'Banana', 'Grape', 'Orange', 'Pineapple',
'Watermelon'] #what class names do you have
    NUM_CLASSES = len(CLASSES)
    #dataloader params - you can add whatever PyTorch dataloader params you
have
    #could be different across train, val, and test
    DATALOADER_PARAMS={
    'batch_size':16,
    'num_workers':2
    }
    # model params
    MODEL_NAME = 'yolo_nas_s' # choose from yolo_nas_s, yolo_nas_m,
yolo_nas_l
    PRETRAINED_WEIGHTS = 'coco' #only one option here: coco
```

Explorando imagens

```
train_images_dir= os.path.join(config.DATA_DIR, config.TRAIN_IMAGES_DIR)
train_labels_dir= os.path.join(config.DATA_DIR, config.TRAIN_LABELS_DIR)
# Number of images to randomly select
num_images = 5
# Get the list of all image files in the 'images' directory
image_files = [f for f in pathlib.Path(train_images_dir).iterdir() if
f.is_file()]
# Shuffle the list of image files
random.shuffle(image_files)

# Select the specified number of image files
selected_image_files = image_files[:num_images]

for selected_image_file in selected_image_files:
    demo_image = selected_image_file
    # Get the corresponding label file
    demo_label = pathlib.Path(train_labels_dir) /
f"{selected_image_file.stem}.txt"
    # Load the image using OpenCV's imread function
    image = cv2.imread(str(demo_image))
    # Get the list of class names from the 'data' dictionary
    class_list = config.CLASSES
    # Define a list of colors to be used to draw bounding boxes
    colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (0,
255, 255), (255, 0, 255)]
    # Get the height and width of the image
    height, width, _ = image.shape
    # Create an empty list T
    T = []
    # Open the label file 'demo_label' in read mode and process each line
    with open(demo_label, "r") as file1:
        for line in file1.readlines():
            # Split the line into a list of strings
            split = line.split(" ")
            # Get the class id from the first element of the split list
            class_id = int(split[0])
            # Get the color corresponding to the class id from the 'colors'
list
            color = colors[class_id]
            clazz = class_list[class_id]
            # Get the x, y, w, h bounding box coordinates from the split
list
            x, y, w, h = float(split[1]), float(split[2]), float(split[3]),
float(split[4])
```

```
            # Rescale the x, y, w, h values to the size of the image
            box = [int((x - 0.5*w)* width), int((y - 0.5*h) * height),
int(w*width), int(h*height)]
            # Draw a rectangle on the image using the 'box' and 'color'
values
            cv2.rectangle(image, box, color, 2)
            # Draw a filled rectangle for the class label on the image
            cv2.rectangle(image, (box[0], box[1] - 20), (box[0] + box[2],
box[1]), color, -1)
            # Write the class label on the image
            cv2.putText(image, class_list[class_id], (box[0], box[1] - 5),
cv2.FONT_HERSHEY_SIMPLEX, .5, (0,0,0))
        # Show the image using matplotlib
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        # Optionally resize the image (commented out in code)
        image = cv2.resize(image, (600, 600))
        plt.show()
```

Fine-tuning YOLONAS

Trainer

A primeira coisa que você precisa definir no SuperGradients é o trainer.

O trainer é responsável pelo treinamento, avaliação, salvamento de checkpoints, etc. Há dois argumentos importantes para o trainer:

1) `ckpt_root_dir` - este é o diretório onde os resultados de todos os seus experimentos serão salvos.

2)`experiment_name` - todos os pontos de verificação, logs e tensorboards serão salvos em um diretório com o nome que você especificar aqui.

```
trainer = Trainer(experiment_name=config.EXPERIMENT_NAME,
ckpt_root_dir=config.CHECKPOINT_DIR)
```

Datasets e DataLoaders

SuperGradients é totalmente compatível com Datasets e Dataloaders do PyTorch, então você pode usar seus dataloaders como estão.

Existem vários conjuntos de dados bem conhecidos para detecção de objetos, como:

- COCO

- Pascal
- YOLODarkNet
- YOLOv5

```
train_data = coco_detection_yolo_format_train(
    dataset_params={
        'data_dir': config.DATA_DIR,
        'images_dir': config.TRAIN_IMAGES_DIR,
        'labels_dir': config.TRAIN_LABELS_DIR,
        'classes': config.CLASSES
    },
    dataloader_params=config.DATALOADER_PARAMS
)

val_data = coco_detection_yolo_format_val(
    dataset_params={
        'data_dir': config.DATA_DIR,
        'images_dir': config.VAL_IMAGES_DIR,
        'labels_dir': config.VAL_LABELS_DIR,
        'classes': config.CLASSES
    },
    dataloader_params=config.DATALOADER_PARAMS
)

test_data = coco_detection_yolo_format_val(
    dataset_params={
        'data_dir': config.DATA_DIR,
        'images_dir': config.TEST_IMAGES_DIR,
        'labels_dir': config.TEST_LABELS_DIR,
        'classes': config.CLASSES
    },
    dataloader_params=config.DATALOADER_PARAMS
)
```

Visualizando como ficam algumas imagens após os aumentos

```
train_data.dataset.plot()
```

Instanciando o modelo

Abaixo está como instanciar o modelo para ajuste fino. Observe que você precisa adicionar o argumento `num_classes` aqui.

Lembre-se, para este tutorial, você está usando `yolo_nas_l`, mas o SuperGradients tem mais duas variações do YOLONAS disponíveis para você: `yolo_nas_s` e `yolo_nas_m`.

```
model = models.get(config.MODEL_NAME,
                   num_classes=config.NUM_CLASSES,
                   pretrained_weights=config.PRETRAINED_WEIGHTS
                   )
```

Definindo métricas e parâmetros de treinamento

Há alguns argumentos obrigatórios que você deve definir para os parâmetros de treinamento

- `max_epochs` - Número máximo de épocas de treinamento
- `loss` - a função de perda
- `optimizer` - o otimizador
- `train_metrics_list` - Métricas para registrar durante o treinamento
- `valid_metrics_list` - Métricas para registrar com o conjunto de validação
- `metric_to_watch` - Métrica pela qual o checkpoint do modelo será salvo

Pode escolher entre uma variedade de otimizadores, como: Adam, AdamW, SGD, Lion ou RMSProps. Se você optar por alterar os parâmetros padrão desses otimizadores, passe-os para optimizer_params.

```
train_params = {
    # ENABLING SILENT MODE
    "average_best_models":True,
    "warmup_mode": "linear_epoch_step",
    "warmup_initial_lr": 1e-5,
    "lr_warmup_epochs": 5,
    "initial_lr": 3e-4,
    "lr_mode": "cosine",
    "cosine_final_lr_ratio": 0.1,
    "optimizer": "Adam",
    "optimizer_params": {"weight_decay": 0.0001},
    "zero_weight_decay_on_bias_and_bn": True,
    "ema": True,
    "ema_params": {"decay": 0.9, "decay_type": "threshold"},
    # ONLY TRAINING FOR 10 EPOCHS FOR THIS EXAMPLE NOTEBOOK
    "max_epochs": 10,
    "mixed_precision": True,
```

```
    "loss": PPYoloELoss(
        use_static_assigner=False,
        # NOTE: num_classes needs to be defined here
        num_classes=config.NUM_CLASSES,
        reg_max=16
    ),
    "valid_metrics_list": [
        DetectionMetrics_050(
            score_thres=0.1,
            top_k_predictions=300,
            # NOTE: num_classes needs to be defined here
            num_cls=config.NUM_CLASSES,
            normalize_targets=True,
            post_prediction_callback=PPYoloEPostPredictionCallback(
                score_threshold=0.01,
                nms_top_k=1000,
                max_predictions=300,
                nms_threshold=0.7
            )
        )
    ],
    "metric_to_watch": 'mAP@0.50'
}
```

Treinamento do modelo

Depois que:

- Instanciamos o `trainer`
- Definimos os parâmetros do seu conjunto de dados e dataloaders
- Instanciamos o modelo
- Configuramos os parâmetros de treinamento

Agora iremos iniciar o treinamento de um modelo usando o SuperGradients.

```
trainer.train(model=model,
              training_params=train_params,
              train_loader=train_data,
              valid_loader=val_data)
```

Obtendo os modelos treinados

Depois que o treinamento está completo, precisamos obter os modelos treinados.

Podemos usar diferentes checkpoints, como por exemplo podemos usar os melhores pesos, pesos da última época, ou os pesos médios, utilizando os seguintes comando:

- pesos médios: `checkpoint_path = os.path.join(config.CHECKPOINT_DIR, config.EXPERIMENT_NAME, 'average_model.pth')`
- melhores pesos: `checkpoint_path = os.path.join(config.CHECKPOINT_DIR, config.EXPERIMENT_NAME, 'ckpt_best.pth')`
- últimos pesos: `checkpoint_path = os.path.join(config.CHECKPOINT_DIR, config.EXPERIMENT_NAME, 'ckpt_latest.pth')`

```
best_model = models.get(config.MODEL_NAME,
                        num_classes=config.NUM_CLASSES,
                        checkpoint_path=os.path.join(config.CHECKPOINT_DIR,
config.EXPERIMENT_NAME,
'/content/checkpoints/fruit-detector/RUN_20231209_222601_547992/ckpt_best.p
th'))
```

Avaliando modelos treinado no conjunto de teste

```
trainer.test(model=best_model,
             test_loader=test_data,
             test_metrics_list=DetectionMetrics_050(score_thres=0.1,
                                                    top_k_predictions=300,

num_cls=config.NUM_CLASSES,
                                                    normalize_targets=True,

post_prediction_callback=PPYoloEPostPredictionCallback(score_threshold=0.01
,

nms_top_k=1000,

max_predictions=300,

nms_threshold=0.7)
                                                   ))
```

Inferência

Apple

```
url =
'https://i0.wp.com/viveiroflorabrasil.com.br/wp-content/uploads/2020/04/maca01.jpg?fit=510%2C510&ssl=1'
best_model.predict(url,  conf=0.4).show()
```

Banana

```
url =
'https://blog4.mfrural.com.br/wp-content/uploads/2020/06/plantar-banana.jpg
'
best_model.predict(url,  conf=0.51).show()
```

Grape

```
url =
'https://ocontadordecervejas.com.br/wp-content/uploads/thumbnails/quanto-tempo-leva-para-um-pe-de-uva-dar-frutos.jpg'
best_model.predict(url,  conf=0.4).show()
```

Orange

```
url =
'https://www.nt4gbi.com/wp-content/uploads/2023/04/Como-saber-se-e-pe-de-laranja-ou-limao.jpg'
best_model.predict(url,  conf=0.51).show()
```

```
url =
'https://www.nt4gbi.com/wp-content/uploads/2023/04/Quanto-tempo-leva-um-pe-de-laranja-para-dar-frutos.jpg'
best_model.predict(url, conf=0.4).show()
```

Pineapple

```
url =
'https://imagens-revista.vivadecora.com.br/uploads/2021/08/Separe-um-espa%C3%A7o-no-seu-terreno-para-plantar-abacaxi.-Foto-HypeScience.jpg'
best_model.predict(url,  conf=0.2).show()
```

```
url =
'https://imagens-revista.vivadecora.com.br/uploads/2021/08/Aprenda-dicas-e-truques-de-como-plantar-abacaxi-em-casa.-Foto-MF-Magazine-MF-Rural.jpg'
best_model.predict(url,  conf=0.51).show()
```

Watermelon

```
url =
'https://s2.glbimg.com/6qQci7OUTmvcCtPU5z-6KP_SAMQ=/e.glbimg.com/og/ed/f/original/2018/05/30/water-melon-1652093_1280.jpg'
best_model.predict(url,  conf=0.2).show()
```

```
url =
'https://alavoura.com.br/wp-content/uploads/2020/09/Agricultores-investem-no-cultivo-irrigado-de-melancia-como-alternativa-de-renda-no-Norte-de-MT-1-e1599056081966.jpg'
best_model.predict(url,  conf=0.7).show()
```

Salvando modelo no drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

```
# caminho para o modelo no Colab
colab_model_path =
'/content/checkpoints/fruit-detector/RUN_20231209_222601_547992/ckpt_latest.pth' #ckpt_best, ckpt_latest, average_model
# caminho para salvar o modelo no Google Drive
drive_model_path =
'/content/drive/MyDrive/TCC/Pesos/Yolonas/ckpt_latest.pth'
# copiando o modelo do Colab para o Google Drive
shutil.copyfile(colab_model_path, drive_model_path)

print(f'Modelo salvo com sucesso em {drive_model_path}')
```

```
Modelo salvo com sucesso em
/content/drive/MyDrive/TCC/Pesos/Yolonas/ckpt_latest.pth
```

Jupyter Notebook: Detectron2.ipynb

Author: Marcos Vinicius Satil Medeiros

Modelos de Detecção

O objetivo deste notebook é treinar modelos de aprendizado profundo para identificar e localizar a presença de uma determinada fruta em uma imagem de entrada.

Bibliotecas utilizadas

```
%%capture
!python -m pip install
'git+https://github.com/facebookresearch/detectron2.git'
!pip install opendatasets
```

```
import os
import cv2
import yaml
import json
import shutil
import requests
import time as t
import numpy as np
import random
import pathlib
import matplotlib.pyplot as plt
from PIL import Image
from pathlib import Path
# detectron2 imports
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
from detectron2 import model_zoo
from detectron2.engine import DefaultTrainer
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.structures import BoxMode
from detectron2.data import DatasetCatalog, MetadataCatalog
from detectron2.evaluation import COCOEvaluator, inference_on_dataset,
LVISEvaluator
```

```
from detectron2.data import build_detection_test_loader
from detectron2.utils.visualizer import ColorMode
```

Baixando dataset do Kaggle com OPENDATASETS

```
import opendatasets as od

od.download('https://www.kaggle.com/datasets/lakshaytyagi01/fruit-detection
')
```

Configurações

Criação de pares de dados (img + ann) para conversão de arquivos de anotação formatados em YOLO para o formato COCO.

```
def pairs_img_ann(input, dir_type = 'train'):
    img_paths = Path(input + dir_type + '/images/').glob('*.jpg')
    pairs = []
    for img_path in img_paths:
        file_name_tmp = str(img_path).split('/')[-1].split('.')
        file_name_tmp.pop(-1)
        file_name = '.'.join((file_name_tmp))
        label_path = Path(input + dir_type + '/labels/' + file_name +
'.txt')
        if label_path.is_file():
            line_img = input + dir_type+'/images/'+ file_name + '.jpg'
            line_annot = input + dir_type+'/labels/' + file_name + '.txt'
            pairs.append([line_img, line_annot])
    return pairs

input = '/content/fruit-detection/Fruits-detection/'

train = pairs_img_ann(input, 'train')
val = pairs_img_ann(input, 'valid')

def coco_format(data_pairs):
    data_list = []
    for i, path in enumerate(data_pairs):
        filename = path[0]
        img_h, img_w = cv2.imread(filename).shape[:2]
        img_item = {}
        img_item['file_name'] = filename
        img_item['image_id'] = i
        img_item['height']= img_h
```

```
            img_item['width']= img_w
            #print(str(i), filename)
            annotations = []
            with open(path[1]) as annot_file:
                lines = annot_file.readlines()
                for line in lines:
                    if line[-1]=="\n":
                      box = line[:-1].split(' ')
                    else:
                      box = line.split(' ')
                    class_id = box[0]
                    x_c = float(box[1])
                    y_c = float(box[2])
                    width = float(box[3])
                    height = float(box[4])
                    x1 = (x_c - (width/2)) * img_w
                    y1 = (y_c - (height/2)) * img_h
                    x2 = (x_c + (width/2)) * img_w
                    y2 = (y_c + (height/2)) * img_h
                    annotation = {
                        "bbox": list(map(float,[x1, y1, x2, y2])),
                        "bbox_mode": BoxMode.XYXY_ABS,
                        "category_id": int(class_id),
                        "iscrowd": 0}
                    annotations.append(annotation)
                img_item["annotations"] = annotations
            data_list.append(img_item)
        return data_list

train_list = coco_format(train)
val_list = coco_format(val)

for catalog_name, file_annots in [("train", train_list), ("val",
val_list)]:
    DatasetCatalog.register(catalog_name, lambda file_annots = file_annots:
file_annots)
    MetadataCatalog.get(catalog_name).set(thing_classes=['Apple', 'Banana',
'Grape', 'Orange', 'Pineapple', 'Watermelon'])

metadata = MetadataCatalog.get("train")

MetadataCatalog.get("val")

namespace(name='val',
          thing_classes=['Apple',
                         'Banana',
                         'Grape',
```

```
                        'Orange',
                        'Pineapple',
                        'Watermelon'])

max_iter = (int(len(train_list)/2)) * 100
print(max_iter)
```

Treinamento do modelo com Detectron2: Faster R-CNN R50 FPN

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R
_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("train",)
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.DEVICE = 'cuda' # cpu
cfg.MODEL.WEIGHTS =
model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 16  # 2
cfg.SOLVER.CHECKPOINT_PERIOD = 500
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 300 # max_iter or (train_size / batch_size) * 100
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128  #
cfg.MODEL.ROI_HEADS.NUM_CLASSES =
len(MetadataCatalog.get("train").thing_classes)
cfg.SOLVER.STEPS = [] # do not decay learning rate

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)

s1 = t.time()
try:
  trainer.train()
except:
  None
s2 = t.time()
# Tempo de treinamento
print(s2 - s1)
```

Inferência

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R
_50_FPN_3x.yaml"))
```

```
cfg.MODEL.DEVICE = 'cuda' # cpu
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 6
cfg.MODEL.WEIGHTS = "/content/output/model_final.pth"
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5   # set the testing threshold
for this model
predictor = DefaultPredictor(cfg)

im =
cv2.imread("/content/fruit-detection/Fruits-detection/valid/images/023a80af
882eba30_jpg.rf.5b5cbadda08d96361527fb5344d7203f.jpg")
outputs = predictor(im)
v = Visualizer(im, metadata=metadata, scale=1., instance_mode =
ColorMode.IMAGE)
v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
img = v.get_image()[:,:,[2,1,0]]
img = Image.fromarray(img)
plt.figure(figsize=(10, 10))
plt.imshow(img)
```

Avaliando modelos treinado no conjunto de teste

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R
_50_FPN_3x.yaml"))
cfg.MODEL.DEVICE = 'cuda' # cpu
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 6
cfg.MODEL.WEIGHTS = "/content/output/model_final.pth"
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5   # set the testing threshold
for this model

predictor = DefaultPredictor(cfg)

evaluator = COCOEvaluator("val", cfg, False, output_dir="./output/")
val_loader = build_detection_test_loader(cfg, "val")
inference_on_dataset(trainer.model, val_loader, evaluator)
```

Jupyter Notebook: benchmarking_models.ipynb

Author: Marcos Vinicius Satil Medeiros

Benchmarking dos modelos

A avaliação em modelos de detecção de objetos é fundamental para comparar e avaliar o desempenho destes. Permitindo selecionar o melhor modelo para uma tarefa, melhorar abordagens existentes e estabelecer padrões de referência. Além disso, o benchmarking auxilia na compreensão das capacidades e limitações de um modelo, impulsionando a inovação e validação experimental. Ao medir métricas quantitativas e identificar lacunas, o benchmark contribui para o avanço contínuo, promovendo eficiência e confiabilidade em cenários diversos.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Bibliotecas

```
%%capture
!pip install opendatasets
!pip install ultralytics
!pip install globox
```

```
import os
import cv2
import glob
import torch
import random
import ultralytics
import numpy as np
from collections import defaultdict
import matplotlib.pyplot as plt
from ultralytics import YOLO
from globox import AnnotationSet, COCOEvaluator
```

Dataset

```
import opendatasets as od
```

```
od.download('https://www.kaggle.com/datasets/lakshaytyagi01/fruit-detection
')
```

```
DATA_DIR = '/content/fruit-detection/Fruits-detection'
data_valid = DATA_DIR+'/valid/images'
data_test = DATA_DIR+'/test/images'
```

Dados de treinamento

```
data_train = AnnotationSet.from_yolo_v5(
    folder=DATA_DIR+"/train/labels/",
    image_folder=DATA_DIR+"/train/images/"
)
data_train.show_stats()
```

Dados de validação

```
val_gts = AnnotationSet.from_yolo_v5(
    folder=DATA_DIR+"/valid/labels/",
    image_folder=DATA_DIR+"/valid/images/"
)
val_gts.show_stats()
```

Dados de teste

```
test_gts = AnnotationSet.from_yolo_v5(
    folder=DATA_DIR+"/test/labels/",
    image_folder=DATA_DIR+"/test/images/"
)
test_gts.show_stats()
```

Benchmarking - Detecções dos modelos

Inferência Yolov8 - NANO

```
model_nano =
"/content/drive/MyDrive/TCC/Pesos/Yolov8/yolov8n/weights/best.pt"
```

```
%%capture
!yolo task=detect \
mode=predict model=$model_nano \
```

```
source=$data_valid\
save_txt=True save_conf=True name='Nano_valid'
```

```
%%capture
!yolo task=detect \
mode=predict model=$model_nano \
source=$data_test\


save_txt=True save_conf=True name='Nano_test'
```

Detecções - Valid

```
dets_nano_v = AnnotationSet.from_yolo_v5(
    folder="./runs/detect/Nano_valid/labels/",
    image_folder="./runs/detect/Nano_valid/"
)
dets_nano_v.show_stats()
```

Métricas do modelo

```
evaluator = COCOEvaluator(
    ground_truths=val_gts,
    predictions=dets_nano_v
)

ap = evaluator.ap()
ar_100 = evaluator.ar_100()
ap_75 = evaluator.ap_75()
ap_small = evaluator.ap_small()

evaluator.show_summary()
```

Detecções - Test

```
dets_nano_t = AnnotationSet.from_yolo_v5(
    folder="./runs/detect/Nano_test/labels/",
    image_folder="./runs/detect/Nano_test/"
)
dets_nano_t.show_stats()
```

Métricas do modelo

```
evaluator = COCOEvaluator(
    ground_truths=test_gts,
    predictions=dets_nano_t
```

```
)
evaluator.show_summary()
```

Inferência Yolov8 - Medium

```
model_medium =
"/content/drive/MyDrive/TCC/Pesos/Yolov8/yolov8m/weights/best.pt"

%%capture
!yolo task=detect \
mode=predict model=$model_medium \
source=$data_valid\
save_txt=True save_conf=True name='Medium_valid'

%%capture
!yolo task=detect \
mode=predict model=$model_medium \
source=$data_test\
save_txt=True save_conf=True name='Medium_test'
```

Detecções - Valid

```
dets_medium_v = AnnotationSet.from_yolo_v5(
    folder="./runs/detect/Medium_valid/labels/",
    image_folder="./runs/detect/Medium_valid/"
)
dets_medium_v.show_stats()
```

Métricas do modelo

```
evaluator = COCOEvaluator(
    ground_truths=val_gts,
    predictions=dets_medium_v
)
evaluator.show_summary()
```

Detecções - Test

```
dets_medium_t = AnnotationSet.from_yolo_v5(
    folder="./runs/detect/Medium_test/labels/",
    image_folder="./runs/detect/Medium_test/"
)

dets_medium_t.show_stats()
```

Métricas do modelo

```
evaluator = COCOEvaluator(
    ground_truths=test_gts,
    predictions=dets_medium_t
)
evaluator.show_summary()
```

Inferência Yolov8 - Large

```
model_large =
"/content/drive/MyDrive/TCC/Pesos/Yolov8/yolov8l/weights/best.pt"
```

```
%%capture
!yolo task=detect \
mode=predict model=$model_large \
source=$data_valid\
save_txt=True save_conf=True name='Large_valid'
```

```
%%capture
!yolo task=detect \
mode=predict model=$model_large \
source=$data_test\
save_txt=True save_conf=True name='Large_teste'
```

Detecções - Valid

```
dets_large_v = AnnotationSet.from_yolo_v5(
    folder="./runs/detect/Large_valid/labels/",
    image_folder="./runs/detect/Large_valid/"
)
dets_large_v.show_stats()
```

Métricas do modelo

```
evaluator = COCOEvaluator(
    ground_truths=val_gts,
    predictions=dets_large_v
)
evaluator.show_summary()
```

Detecções - Test

```
dets_large_t = AnnotationSet.from_yolo_v5(
    folder="./runs/detect/Large_teste/labels/",
    image_folder="./runs/detect/Large_teste/"
```

```
)
dets_large_t.show_stats()
```

Métricas do modelo

```
evaluator = COCOEvaluator(
    ground_truths=test_gts,
    predictions=dets_large_t
)
evaluator.show_summary()
```

APÊNDICE 6

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 11 de jan. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

Marcos Vinicius Satil Medeiros

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para a última entrega, fiz os objetivos listados no gate anterior:

- ☑ Combinação dos resultados para diferentes tarefas apresentadas.
- ☑ Estudo e elaboração de documento sobre a tarefa de segmentação.
- ☑ Atualização do repositório.

Os objetivos listados foram incluídos em um documento no seguinte link:
Entrega 11/01- MARCOS VINICIUS SATIL MEDEIROS

Além disso, organizei todas as entregas (Gates) em uma pasta no drive TCC, onde estão as pastas contendo os documentos entregues separados por semana (1 a 10), códigos (colabs) separados por tarefas e pesos dos modelos produzidos.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Organização e apresentação do TCC.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

LUANA GUEDES BARROS MARTINS: Go!

MODELOS DE SEGMENTAÇÃO

O objetivo principal desta entrega será explorar a tarefa de segmentação. Dado que já realizamos revisões de literatura em entregas anteriores, neste documento iremos descrever conceitos importantes, assim como as arquiteturas exploradas. O código produzido pode ser encontrado no repositório do github: https://github.com/marcosvinism/Residencia-em-IA

Conceitos sobre segmentação

No contexto da Visão Computacional, cinco tarefas principais podem ser abordadas de maneiras distintas:

I) Categorização de Imagens: Consiste na classificação de imagens com base em um conjunto limitado de classes.

II) Detecção de Objetos: Envolve a criação de caixas delimitadoras ao redor de itens específicos em imagens, atribuindo-os a categorias específicas.

III) Segmentação Semântica: Busca categorizar áreas dentro das imagens com base em textura, cor e distribuição espacial.

IV) Segmentação de Instâncias: Combina reconhecimento de objetos e segmentação semântica, identificando diversas instâncias na imagem, juntamente com suas bordas.

V) Segmentação Panóptica: Uma combinação de segmentação semântica e de instância, a segmentação panóptica atribui um rótulo exclusivo a cada pixel, seja ele pertencente a uma instância ou a uma classe semântica.

Dentre essas tarefas, a segmentação de objetos como um todo destaca-se como aquela que visa à identificação precisa, em nível de pixel, de todos os objetos presentes em uma imagem. Essa técnica é utilizada em diversas aplicações do mundo real, especialmente em sistemas de visão para carros autônomos, sistemas de monitoramento e, também no nosso exemplo prático, na agricultura.

No contexto agrícola, enfrentamos desafios significativos, destacando-se a necessidade de identificação e mapeamento precisos das características e condições das culturas, tais como diferenciação, doenças, altura das plantas, índice de área foliar e estágio

de crescimento. As abordagens fundamentadas em visão computacional têm sido um impulso significativo para a inovação na indústria, sendo aplicadas em diversas outras áreas.

A implementação dessas tarefas tem evoluído ao longo do tempo, adaptando-se às variáveis condições de iluminação e nitidez das bordas. Diversas técnicas têm sido empregadas, incluindo a conversão do espaço de cores e a combinação de canais de cores. Além disso, técnicas clássicas baseadas em aprendizado de máquina têm sido utilizadas para superar desafios específicos., contudo uma abordagem que tem se destacado é o uso de Deep Learning onde as principais ferramentas são as redes neurais.

A popularidade desse método deve-se, em parte, à disponibilidade pública de modelos como VGG, U-Net, SegNet e DeepLab, que são redes neurais profundas treinadas em conjuntos de dados extensos. Essa evolução na segmentação de objetos destaca não apenas a importância da abordagem Deep Learning, mas também a relevância de conjuntos de dados abrangentes para o treinamento desses modelos, impulsionando avanços significativos na área.

Modelos mais populares e seus conjuntos de dados relacionados

Modelo	Conjuntos de dados usados para treinamento	Tarefa de Visão Computacional
AlexNet (5 camadas de convolução)	ImageNet: 14.197.122 imagens	Classificação de imagens
VGG-16 (16 camadas de convolução)	ImageNet	Segmentação semântica de imagens
ResNet-50 (48 camadas de convolução)	ImageNet	Classificação de imagens, detecção de objetos
U-Net (23 camadas de convolução)	Vários conjuntos de dados de imagens médicas: - MICCAI - ISIC	Segmentação semântica de imagens
DeepLab (4 camadas de convolução)	- PASCAL VOC - Cityscapes: 5000 images - ADE20K: 20210	Segmentação semântica de imagens

	images - COCO: 118000 images	
SegNet (13 camadas de convolução)	- CamVid - Cityscapes - SUN RGB-D - ADE20K	Segmentação semântica de imagens
EfficientNet-B0 (24 camadas de convolução)	ImageNet	Segmentação semântica de imagens

Resumo de técnicas populares de Segmentação de Imagens

- **Limiar:** Este método envolve definir um valor limite para criar imagens binárias com base na intensidade do pixel. Os pixels acima ou abaixo deste limite são categorizados de acordo.
- **Algoritmos de agrupamento:** técnicas como K-means agrupam pixels com base em semelhanças de cor, intensidade ou outros recursos, dividindo a imagem em segmentos.
- **Segmentação baseada em bordas:** identificação de limites entre diferentes regiões, detectando descontinuidades em uma imagem, geralmente por meio de algoritmos de detecção de bordas, como a detecção de bordas Canny.
- **Abordagens de aprendizagem profunda:** Redes Neurais Convolucionais (CNNs) revolucionaram a segmentação de imagens. Arquiteturas como U-Net, FCN (Fully Convolutional Networks) e DeepLab empregam CNNs para segmentar imagens com mais precisão, aprendendo recursos em diferentes escalas.

Análise Comparativa: Segmentação Semântica vs Segmentação de Instância

A segmentação semântica e de instância são técnicas de análise de imagens em visão computacional. Fundamentalmente, a diferença entre as duas técnicas reside na profundidade dos seus modelos de classificação, bem como na sua complexidade. Assim

sendo, ambos têm suas vantagens e desvantagens, tornando-os mais adequados para diferentes casos de uso.

Precisão na identificação de objetos

A segmentação semântica é excelente em cenários onde o objetivo principal é compreender a composição geral de uma imagem. Por exemplo, na monitorização ambiental, a segmentação semântica pode classificar diferentes tipos de cobertura do solo (ou seja, aquática, florestal, urbana) em imagens de satélite.

Já a segmentação de instâncias oferece precisão superior em cenários que exigem identificação e contagem de objetos individuais. No varejo , por exemplo, a segmentação de instâncias é aplicada para análise de prateleira – identificando e contando produtos específicos, uma aplicação onde a segmentação semântica seria insuficiente.

Tratamento de objetos sobrepostos

A segmentação semântica pode ter dificuldades com objetos sobrepostos da mesma classe, pois não consegue distinguir entre diferentes instâncias. Esta limitação é significativa em imagens médicas ao segmentar células ou tecidos que se sobrepõem. A segmentação de instâncias é excelente no tratamento de objetos sobrepostos. Na análise de multidões, como na vigilância ou no gerenciamento de eventos, a segmentação de instâncias pode identificar e rastrear individualmente cada pessoa, mesmo em um quadro densamente povoado.

Capacidades de processamento em tempo real

A segmentação semântica é mais adequada para aplicações em tempo real devido aos seus requisitos computacionais relativamente mais baixos. Os sistemas de direção autônoma geralmente empregam segmentação semântica para detecção de estradas e obstáculos em tempo real. Neste caso, a detecção e classificação rápidas são muito mais importantes do que manter a contagem ou distinguir entre diferentes objetos do mesmo tipo. Devido à sua intensidade computacional, a segmentação de instâncias é usada com menos frequência em cenários de tempo real. No entanto, é indispensável na análise pós-evento ou

em situações onde a alta precisão e a identificação de objetos individuais são críticas, como na análise detalhada da cena pós-acidente em investigações forenses.

Avanços na tarefa de segmentação de imagens

A constante evolução na área de segmentação de imagens tem sido marcada por inovações que transcendem os limites convencionais. Neste cenário dinâmico, destaca-se a eficácia das abordagens híbridas, que combinam aprendizagem profunda com outros métodos. A sinergia entre diferentes técnicas revela-se como um catalisador essencial para alcançar resultados mais robustos e refinados.

Outro ponto de destaque são os notáveis avanços na aprendizagem auto-supervisionada, que desencadeiam uma redução significativa na dependência de dados anotados. Esta conquista permite que os modelos absorvam representações valiosas a partir de dados não rotulados, ampliando consideravelmente as possibilidades de treinamento e aplicação em ambientes nos quais a disponibilidade de dados rotulados é limitada.

Adicionalmente, a ascensão do mecanismo de atenção, especialmente nos modelos baseados em transformers, apresenta uma revolução na forma como a segmentação de imagens é abordada. Ao concentrar-se de maneira inteligente em regiões relevantes da imagem, esses mecanismos conferem maior peso às partes informativas, resultando em um aprimoramento no desempenho da segmentação.

Destacamos ainda modelos zero-shot como o Segment Anything Model (SAM), uma inovação recente que representa um salto significativo na capacidade de segmentação, incorporando uma abordagem mais adaptável. A presença destes modelos nesse contexto reforça o impacto das inovações e seu papel no futuro desta área.

Segment Anything Model (SAM)

O Segment Anything Model (SAM) ou segmente qualquer coisa, uma recente inovação proveniente do laboratório FAIR (Fundamental AI Research) da Meta, representa uma mudança importante na área de visão computacional. Este modelo de segmentação de

instâncias de última geração demonstra uma notável capacidade de realizar tarefas complexas de segmentação de imagens com precisão e versatilidade sem precedentes.

Ao contrário dos modelos convencionais que demandam extenso treinamento em tarefas específicas, o design do Segment Anything Model adota uma abordagem mais adaptável, inspirando-se nos avanços recentes em processamento de linguagem natural (PLN ou NLP). O impacto revolucionário do SAM reside em suas capacidades de inferência zero-shot. Isso implica que o SAM consegue segmentar imagens com precisão sem a necessidade de treinamento prévio específico, uma tarefa que normalmente exige modelos personalizados. Esse salto de eficiência é atribuído à influência de modelos fundamentais de NLP, como os modelos GPT e BERT, que revolucionaram a compreensão e geração da linguagem humana pelas máquinas, aprendendo com vastos conjuntos de dados e generalizando para diversas tarefas. O SAM aplica uma filosofia semelhante à visão, utilizando um extenso conjunto de dados para compreender e segmentar uma ampla variedade de imagens.

Arquitetura SAM

As capacidades revolucionárias do SAM baseiam-se principalmente em sua arquitetura, que consiste em três componentes principais: o codificador de imagem, o codificador de prompt e o decodificador de máscara.

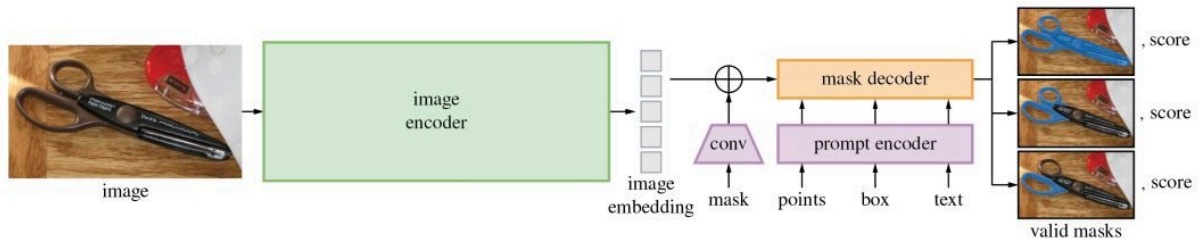


Figura 1: Arquitetura SAM.

Codificador de imagem

- O codificador de imagem está no centro da arquitetura do SAM, um componente responsável por processar e transformar imagens de entrada em um conjunto abrangente de recursos.

- Usando uma abordagem baseada em transformers, como visto em modelos avançados de NLP, esse codificador compacta imagens em uma matriz de recursos densa. Esta matriz forma o entendimento fundamental a partir do qual o modelo identifica vários elementos da imagem.

Codificador de prompt

- O codificador de prompt é um aspecto único do SAM que o diferencia dos modelos tradicionais de segmentação de imagens.
- Ele interpreta várias formas de prompts de entrada, sejam eles baseados em texto, pontos, máscaras aproximadas ou uma combinação destes.
- Este codificador traduz esses prompts em uma incorporação que orienta o processo de segmentação. Isso permite que o modelo foque em áreas ou objetos específicos dentro de uma imagem conforme a entrada determina.

Decodificador de máscara

- O decodificador de máscara é onde acontece a mágica da segmentação. Ele sintetiza as informações dos codificadores de imagem e de prompt para produzir máscaras de segmentação precisas.
- Este componente é responsável pelo resultado final, determinando os contornos e áreas precisas de cada segmento da imagem.
- A forma como esses componentes interagem entre si é igualmente vital para uma segmentação de imagem eficaz, assim como suas capacidades:
- O codificador de imagem primeiro cria uma compreensão detalhada de toda a imagem, dividindo-a em recursos que o mecanismo pode analisar.
- O codificador de prompt então adiciona contexto, concentrando a atenção do modelo com base na entrada fornecida, seja um ponto simples ou uma descrição de texto complexa.
- Finalmente, o decodificador de máscara usa essas informações combinadas para segmentar a imagem com precisão, garantindo que a saída esteja alinhada com a intenção do prompt de entrada.

Fundamentação SAM

As Redes Neurais Convolucionais (CNNs) e as Redes Adversariais Generativas (GANs) constituem a base essencial que viabiliza a segmentação de imagens realizada pelo SAM. Esses modelos de aprendizado profundo são fundamentais pelo avanço no campo de aprendizado de máquina e inteligência artificial, especialmente no âmbito do processamento de imagens.

As CNNs são parte integrante do codificador de imagem da arquitetura. Elas se destacam no reconhecimento de padrões em imagens, aprendendo hierarquias espaciais, desde bordas simples até formas mais complexas. No SAM, as CNNs analisam e interpretam dados visuais, processando pixels de forma eficiente para detectar e compreender vários recursos e objetos dentro de uma imagem.

Já as GANs contribuem para a capacidade do SAM de gerar máscaras de segmentação precisas. Consistindo em duas partes, o gerador e o discriminador, os GANs são adeptos da compreensão e replicação de distribuições de dados complexos. O gerador se concentra na produção de imagens realistas e o discriminador avalia essas imagens para determinar se são reais ou criadas artificialmente. Esta dinâmica aumenta a capacidade do gerador de criar imagens sintéticas altamente realistas.

Essa combinação permite que o SAM entenda uma ampla gama de entradas visuais e responda com alta precisão. Ao integrar estas tecnologias, o SAM demonstra o potencial de combinar diferentes arquiteturas de redes neurais para aplicações avançadas de IA.

CLIP (Contrastive Language-Image Pre-training)

Na parte dos prompts, o SAM se integra ao CLIP, um modelo desenvolvido pela OpenAI que preenche a lacuna entre texto e imagens. Sua capacidade de compreender e interpretar prompts de texto em relação às imagens revela-se inestimável para o desempenho do SAM.

A habilidade do CLIP de processar e interpretar entradas baseadas em texto, como descrições ou rótulos, é altamente eficaz para a boa operação do SAM. Essa integração capacita o SAM a responder e agir com base em instruções textuais, relacionando-as com precisão aos dados visuais. Como resultado, a versatilidade do SAM é ampliada,

possibilitando a segmentação de imagens com base não apenas em pistas visuais, mas também na interpretação inteligente de instruções textuais. Essa colaboração entre estes dois modelos representa uma convergência poderosa entre o entendimento textual e a capacidade de segmentação visual.

Conceitos e Resultados obtidos

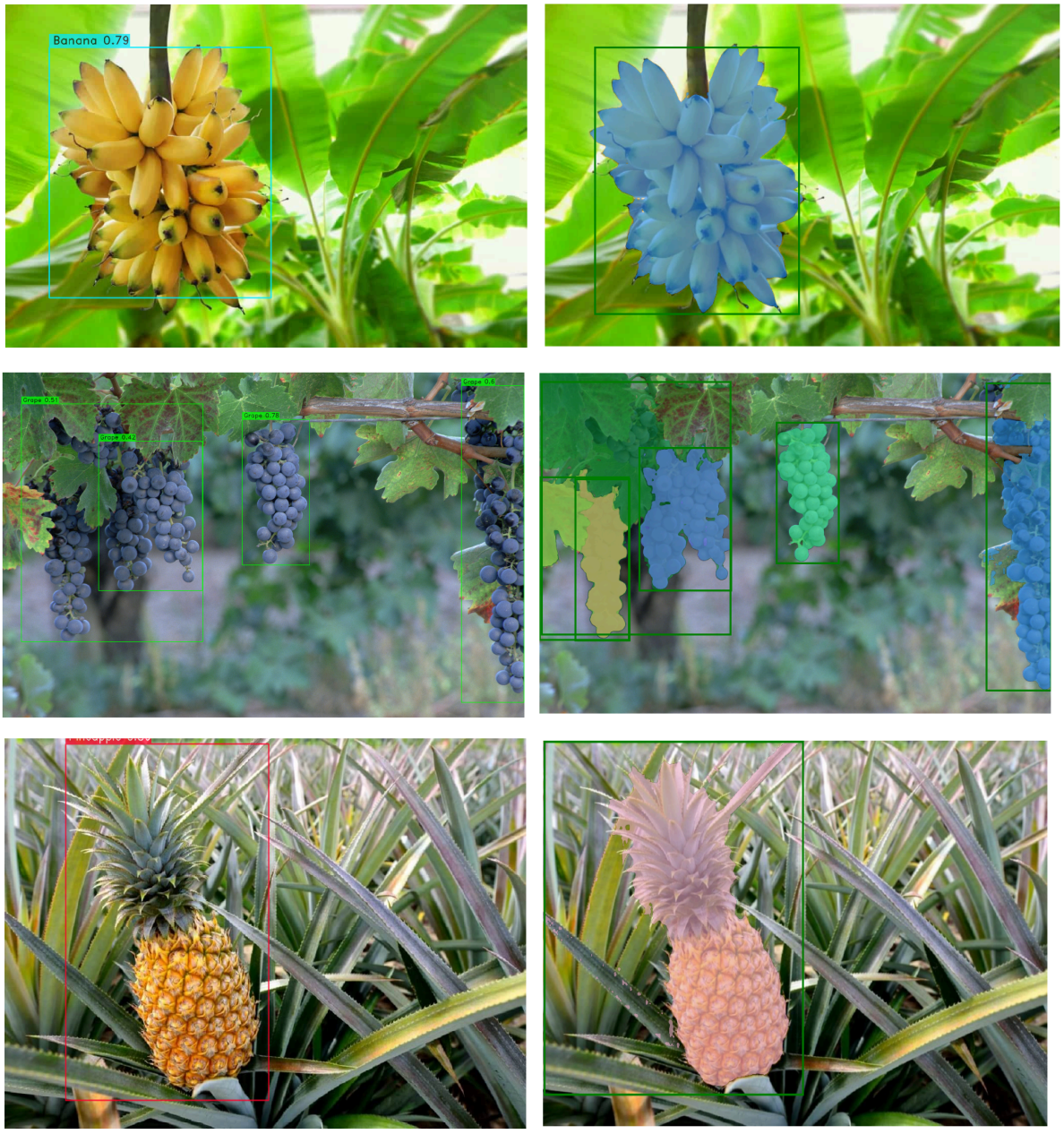
O SAM suporta quatro modos de segmentação:

- segmentar tudo de forma totalmente automática
- com entrada de bounding box
- com entrada de ponto
- com entrada de texto.

Os últimos três modos são semiautomáticos, ou seja, requerem intervenção humana para cada imagem. semiautomático significa que enquanto o usuário vê a imagem, a instância de um objeto chamado SamPredictor (na linguagem de programação Python) aguarda a entrada do usuário (uma bounding box ou um ponto) e então atualiza imediatamente a máscara de segmentação em tempo real. A segmentação totalmente automática usa outro objeto chamado SamAutomaticMaskGenerator, criando uma máscara para cada objeto detectado dentro da imagem.

Considerando as entregas anteriores, onde tínhamos treinado modelos de detecção de objeto, realizei testes com as suas inferências e utilizei o SAM para gerar a máscara de segmentação destas saídas. Os resultados obtidos podem ser vistos a seguir:

Bounding box **Mask segmentation**



Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Os resultados mostrados indicam que a segmentação totalmente automática ou considerando uma maior parte da imagem, ainda não está totalmente pronto para substituir a segmentação de objetos quando anotado por um humano ou automatizar este processo. O SAM tende a super segmentar, ou seja, a detectar mais objetos do que aqueles realmente presentes. Entretanto, quando utilizado em regiões menores, ele possui uma boa performance. Este fato é dedutível pelas duas colunas Precisão e Recall. Baixa Precisão significa que há muitos falsos positivos, enquanto o alto Recall indica que o algoritmo detecta uma alta proporção de positivos.

Automatizando a Criação de Datasets

A geração automatizada de datasets é um avanço impulsionado pela integração de modelos que funcionam bem em um determinado cenário, e podem ser usados para gerar novas anotações em dados não anotados. Essa abordagem simplifica e acelera o processo de criação de conjuntos de dados, proporcionando benefícios substanciais em diversas áreas.

Ao unir um modelo de detecção eficiente com a capacidade sofisticada de segmentação do Segment Anything Model (SAM), é possível automatizar a geração de datasets de maneira abrangente e precisa. O modelo de detecção identifica objetos em uma imagem, enquanto o SAM cria máscaras de segmentação detalhadas desses objetos. A combinação dessas informações resulta em entradas ricas e rotuladas automaticamente, construindo um conjunto de dados diversificado e de alta qualidade.

Essa abordagem tem implicações significativas como:

Treinamento Eficiente de Modelos: A geração automática de datasets proporciona uma fonte contínua e diversificada de dados para treinamento de modelos. Isso é especialmente útil em tarefas de aprendizado supervisionado, onde é necessário a disponibilidade de dados rotulados.

Aplicações em Domínios Específicos: Setores como a medicina, agricultura e manufatura podem se beneficiar da criação automatizada de datasets para treinar modelos específicos. Esses modelos podem ser personalizados para identificar e segmentar elementos específicos dentro de imagens, otimizando processos e oferecendo insights valiosos.

Desenvolvimento de Sistemas Autônomos: Em cenários como veículos autônomos, a geração automática de datasets contribui para a criação de conjuntos de treinamento diversificados, refletindo uma ampla gama de condições e situações de tráfego. Isso melhora a robustez e a capacidade de generalização desses sistemas.

Redução do Esforço Manual: Eliminar ou reduzir a necessidade de rotulação manual de grandes conjuntos de dados economiza tempo e recursos humanos, permitindo que os profissionais concentrem seus esforços em tarefas mais complexas e estratégicas.

Em resumo, a abordagem de detecção e segmentação, aliada à automação da criação de datasets, pode representar um avanço na eficiência e na qualidade dos conjuntos de dados utilizados para treinamento de modelos em visão computacional. Essa inovação não apenas acelera o desenvolvimento de sistemas inteligentes, mas também abre portas para a resolução de desafios complexos em uma variedade de campos de aplicação.

Referências

A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A.C. Berg, W.-Y. Lo, P. Dollár, R. Girshick. Segment Anything (2023).

J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation,"in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15), pp.3431–3440, IEEE, Boston, Mass, USA, June 2015.

Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation. In: Lect. notes comput. sci. (Including subser. lect. notes artif. intell. lect. notes bioinformatics), Vol. 9351. Springer Verlag; 2015, p. 234–41.

Alberto Carraro, Marco Sozzi, Francesco Marinello. The Segment Anything Model (SAM) for accelerating the smart farming revolution (2023), Smart Agricultural Technology. https://doi.org/10.1016/j.atech.2023.100367

Jupyter Notebook: Segmentação SAM.ipynb

Author: Marcos Vinicius Satil Medeiros

Modelos de Segmentação

O objetivo deste notebook é demonstrar a aplicação de modelos de segmentação para geração de anotação de imagens, dado que já temos modelos de detecção de objetos.

- Serão demonstrados diferentes cenários onde temos uma simples detecção, múltiplas detecções e também para uma imagem como um todo.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Instalando dependências

```
%%capture
!pip install ultralytics
!pip install 'git+https://github.com/facebookresearch/segment-anything.git'
```

Download das imagens e pesos dos modelos

```
%%capture
# Pesos
!wget https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth
```

```
%%capture
# Imagens Teste
!wget -P images
'https://raw.githubusercontent.com/facebookresearch/segment-anything/main/notebooks/images/truck.jpg'
!wget -P images
'https://raw.githubusercontent.com/facebookresearch/segment-anything/main/notebooks/images/groceries.jpg'
!wget -P images 'https://wricidades.org/sites/default/files/tr%C3%AAs.jpg'
!wget -P images 'https://wricidades.org/sites/default/files/naldo.jpg'
```

```
%%capture
# Frutas
!wget -P images
```

```
'https://s2-g1.glbimg.com/hWMKfZrVwHIPJJMgslhvGEuRKGw=/0x0:1920x1080/984x0/smart/filters:strip_icc()/i.s3.glbimg.com/v1/AUTH_59edd422c0c84a879bd37670ae4f538a/internal_photos/bs/2018/i/S/DJzZRhQWyub4r7FnBZrA/nc-maca-safra-181118.jpg'
!wget -P images 'https://blog4.mfrural.com.br/wp-content/uploads/2020/06/plantar-banana.jpg' # Banana
!wget -P images 'https://ocontadordecervejas.com.br/wp-content/uploads/thumbnails/quanto-tempo-leva-para-um-pe-de-uva-dar-frutos.jpg' # Uva
!wget -P images 'https://imagens-revista.vivadecora.com.br/uploads/2021/08/Aprenda-dicas-e-truques-de-como-plantar-abacaxi-em-casa.-Foto-MF-Magazine-MF-Rural.jpg'
```

Bibliotecas utilizadas

```
from ultralytics import YOLO
import numpy as np
import cv2
import sys
import torch
from PIL import Image
from segment_anything import sam_model_registry, SamPredictor
import matplotlib.pyplot as plt
```

Funções utilizadas

```
def yolov8_detection(model, img_path):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = model(img, save=True)  # generator of Results objects
    for result in results:
        boxes = result.boxes  # Boxes object for bbox outputs
    bbox=boxes.xyxy.tolist()[0]
    return bbox, img

def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])],
axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
```

```
    ax.imshow(mask_image)

def show_points(coords, labels, ax, marker_size=375):
    pos_points = coords[labels==1]
    neg_points = coords[labels==0]
    ax.scatter(pos_points[:, 0], pos_points[:, 1], color='green',
marker='*', s=marker_size, edgecolor='white', linewidth=1.25)
    ax.scatter(neg_points[:, 0], neg_points[:, 1], color='red', marker='*',
s=marker_size, edgecolor='white', linewidth=1.25)

def show_box(box, ax):
    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    ax.add_patch(plt.Rectangle((x0, y0), w, h, edgecolor='green',
facecolor=(0,0,0,0), lw=2))

image_path = '/content/images/plantar-banana.jpg'
model_path =
'/content/drive/MyDrive/TCC/Pesos/Yolov8/yolov8l/weights/best.pt'
model=YOLO(model_path)
yolov8_boxex, image = yolov8_detection(model, image_path)

sam_checkpoint = "sam_vit_h_4b8939.pth"
model_type = "vit_h"
device = "cuda:0" #cpu

sam = sam_model_registry[model_type](checkpoint=sam_checkpoint)
sam.to(device=device)

predictor = SamPredictor(sam)
predictor.set_image(image)

input_box = np.array(yolov8_boxex)

masks, _, _ = predictor.predict(
    point_coords=None,
    point_labels=None,
    box=input_box[None, :],
    multimask_output=False,
)

for i, mask in enumerate(masks):
    # Convert the mask to a binary image
    #binary_mask = mask.cpu().numpy().squeeze().astype(np.uint8)
    binary_mask =
torch.from_numpy(masks).squeeze().numpy().astype(np.uint8)
```

```
    # Find the contours of the mask
    contours, hierarchy = cv2.findContours(binary_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    # Get the largest contour based on area
    largest_contour = max(contours, key=cv2.contourArea)
    # Get the new bounding box
    bbox = [int(x) for x in cv2.boundingRect(largest_contour)]
    # Get the segmentation mask for object
    segmentation = largest_contour.flatten().tolist()
    # Write bounding boxes to file in YOLO format
    with open('bounding_boxe_NEW_137.txt', 'w') as f:
        for contour in contours:
            # Get the bounding box coordinates of the contour
            x, y, w, h = cv2.boundingRect(contour)
            # Convert the coordinates to YOLO format and write to file
            f.write('0 {:.6f} {:.6f} {:.6f}
{:.6f}\n'.format((x+w/2)/image.shape[1], (y+h/2)/image.shape[0],
w/image.shape[1], h/image.shape[0]))
    mask=segmentation
    # load the image
    #width, height = image_path.size
    img = Image.open(image_path)
    width, height = img.size
    # convert mask to numpy array of shape (N,2)
    mask = np.array(mask).reshape(-1,2)
    # normalize the pixel coordinates
    mask_norm = mask / np.array([width, height])
    # compute the bounding box
    xmin, ymin = mask_norm.min(axis=0)
    xmax, ymax = mask_norm.max(axis=0)
    bbox_norm = np.array([xmin, ymin, xmax, ymax])
    # concatenate bbox and mask to obtain YOLO format
    yolo = np.concatenate([bbox_norm, mask_norm.reshape(-1)])
    # write the yolo values to a text file
    with open('yolo_maskformat.txt', 'w') as f:
        for val in yolo:
            f.write("{:.6f} ".format(val))
    plt.figure(figsize=(10, 10))
    plt.imshow(image)
    show_mask(masks[0], plt.gca())
    show_box(input_box, plt.gca())
    plt.axis('off')
    plt.show()
    # Print the bounding box and segmentation mask
```

```
    print("Bounding box:", bbox)
    print("Segmentation mask:", segmentation)
```

Multi object detection

```
def yolov8_detection(model, img_path):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = model(img, save=True)  # generator of Results objects
    for result in results:
        boxes = result.boxes  # Boxes object for bbox outputs
    bbox = boxes.xyxy.tolist()
    bbox = [[int(i) for i in box] for box in bbox]
    return bbox, iag

image =
'/content/images/quanto-tempo-leva-para-um-pe-de-uva-dar-frutos.jpg'
yolov8_boxex, image = yolov8_detection(model, image)



def inference(image, sam):
    predictor = SamPredictor(sam)
    predictor.set_image(image)
    input_boxes = torch.tensor(yolov8_boxex, device=predictor.device)

    transformed_boxes = predictor.transform.apply_boxes_torch(input_boxes,
image.shape[:2])

    masks, _, _ = predictor.predict_torch(
        point_coords=None,
        point_labels=None,
        boxes=transformed_boxes,
        multimask_output=False,
    )

    for i, mask in enumerate(masks):
        #binary_mask = masks[i].squeeze().numpy().astype(np.uint8)
        binary_mask = masks[i].squeeze().cpu().numpy().astype(np.uint8)
        # Find the contours of the mask
        contours, hierarchy = cv2.findContours(binary_mask,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        largest_contour = max(contours, key=cv2.contourArea)
        # Get the new bounding box
        bbox = [int(x) for x in cv2.boundingRect(largest_contour)]
```

```
        # Get the segmentation mask for object
        segmentation = largest_contour.flatten().tolist()
        # Write bounding boxes to file in YOLO format
        with open('BBOX_yolo.txt', 'w') as f:
            for contour in contours:
                # Get the bounding box coordinates of the contour
                x, y, w, h = cv2.boundingRect(contour)
                # Convert the coordinates to YOLO format and write to file
                f.write('0 {:.6f} {:.6f} {:.6f}
{:.6f}\n'.format((x+w/2)/image.shape[1], (y+h/2)/image.shape[0],
w/image.shape[1], h/image.shape[0]))
                f.write('\n')
        mask=segmentation
        # load the image
        #width, height = image_path.size
        img = Image.open(image_path)
        width, height = img.size
        # convert mask to numpy array of shape (N,2)
        mask = np.array(mask).reshape(-1,2)
        # normalize the pixel coordinates
        mask_norm = mask / np.array([width, height])
        # compute the bounding box
        xmin, ymin = mask_norm.min(axis=0)
        xmax, ymax = mask_norm.max(axis=0)
        bbox_norm = np.array([xmin, ymin, xmax, ymax])
        # concatenate bbox and mask to obtain YOLO format
        yolo = np.concatenate([bbox_norm, mask_norm.reshape(-1)])
        # compute the bounding box
        # write the yolo values to a text file
        with open('yolomask_format.txt', 'w') as f:
            for val in yolo:
                f.write("{:.6f} ".format(val))
            f.write('\n')
        # Print the bounding box and segmentation mask
        print("Bounding box:", bbox)
        #print("Segmentation mask:", segmentation)
        print("yolo",yolo)
    plt.figure(figsize=(10, 10))
    plt.imshow(image)
    for mask in masks:
        show_mask(mask.cpu().numpy(), plt.gca(), random_color=True)
    for box in input_boxes:
        show_box(box.cpu().numpy(), plt.gca())
    plt.axis('off')
    plt.show()
```

```
inference(image, sam)


image = '/content/images/nc-maca-safra-181118.jpg'
yolov8_boxex, image = yolov8_detection(model, image)

image =
'/content/images/Aprenda-dicas-e-truques-de-como-plantar-abacaxi-em-casa.-Foto-MF-Magazine-MF-Rural.jpg'
yolov8_boxex, image = yolov8_detection(model, image)
```