UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

BERNARDO ZUCCO MÜLLER

Desenvolvimento de um Web Service e Extensão do App Inventor para Detecção de Objetos

Trabalho de Conclusão do Curso de Graduação em Sistemas de Informação, do Departamento de Informática e Estatística, do Centro Tecnológico da Universidade Federal de Santa Catarina, requisito parcial à obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Profa. Dr. rer. nat. Christiane Gresse von Wangenheim, PMP

FLORIANÓPOLIS
2024

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A inteligência artificial tem cada vez ganhado mais espaço de utilização em situações cotidianas, agregando melhorias na sociedade. Portanto, é importante incluir o ensino de inteligência artificial na educação, a fim de promover o interesse nessa área em todos os níveis de ensino no Brasil. A IA é um grande campo de conhecimento que possui diversas ramificações, uma delas é o *Machine Learning (ML)*. Uma das tarefas de ML é a detecção de objetos, na qual a capacidade de aprendizagem da máquina permite que sejam identificados instâncias de objetos em uma imagem, tipicamente usando modelos de deep learning como YOLO. A utilização da plataforma App Inventor para implantar aplicações que utilizam esses conceitos na prática, é uma forma de inserir ML na educação básica. Já existem extensões do App Inventor para classificação de imagem e áudio, porém não existe nenhuma extensão que permita criar um aplicativo que realize a detecção de objetos. Com isso, o presente trabalho busca propor uma solução para detecção de objetos que utilize o YOLO como modelo e desenvolver uma extensão que permita utilizar o App Inventor para desenvolver aplicativos que fazem a detecção de objetos. Espera-se que os resultados deste TCC contribuam no contexto de computação na escola.

Palavras-chave: Inteligência Artificial, Machine Learning, YOLO, App Inventor, Detecção de Objetos, ONNX.

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Atualmente a Inteligência Artificial (IA) demonstra um papel econômico-social muito grande no mundo todo por estar presente em grande parte das atividades do nosso dia a dia, desde a produção de manufaturas até atividades médicas (Tavares, 2020). Uma área da IA é o *Machine Learning,* que é a capacidade das máquinas aprenderem com base em conjuntos de dados, de forma similar a capacidade de aprendizagem humana (Mitchell, 1997). Uma das principais tarefas de ML é a detecção de objetos (Zou *et al.*, 2019). Detecção de objetos ajuda a identificação de diversas instâncias de objetos presentes em vídeos ou imagens. Por exemplo, a tecnologia pode ser utilizada em veículos autônomos para permitir ao veículo reconhecer os componentes do ambiente que o cerca (Azevedo, 2022).

A detecção de objetos busca desenvolver modelos computacionais e técnicas de IA que encontrem a localização e classe de um objeto em um vídeo ou imagem (Zou *et al.*, 2019). YOLO é um modelo de *Deep Learning* que divide imagens em um sistema de grades, onde cada célula da grade fica responsável por detectar os objetos dentro dela, permitindo então que a detecção de objetos aconteça em uma única passagem pela rede neural, tornando o algoritmo mais rápido (Redmon & Farhadi, 2015). E ao final, o modelo treinado pode ser exportado no formato ONNX (Redmon & Farhadi, 2015), um formato criado para representar modelos de aprendizado de máquina. Esse arquivo é composto de um modelo de grafos extensível, representando os operadores de entrada e o padrão de cada tipo de dado, com isso é possível exportar um arquivo com formato padrão para ser utilizado por aplicações de IA (ONNX, 2023). Existem ferramentas para realizar as tarefas de treinamento do modelo, como os Jupyter Notebooks que permitem a criação e exportação de modelos treinados (O'hara *et al.*, 2015).

Porém, entender sobre IA ainda é algo muito distante para grande parte da sociedade. Currículos escolares raramente englobam conteúdos sobre como desenvolver sistemas inteligentes. Esse fator impede uma disseminação do conhecimento dessa área na população e acaba afastando possíveis interessados no campo de estudos de IA (Marques *et al*., 2020). Ao encontrar maneiras de popularizar o ensino da computação e IA/ML na escola, portas se abrem para que

sejam inseridos conceitos sobre tecnologia na formação desses alunos. Atualmente já estão surgindo cursos para ensinar IA/ML na educação básica (Marques *et al.*, 2020). A maioria desses cursos adotam a metodologia ativa que leva os estudantes a desenvolver modelos de IA/ML, inclusive de detecção de objetos.

Observando a falta de cursos voltados para o ensino de ML utilizado na detecção de objetos nas escolas brasileiras de ensino médio surgiu o curso “Crie o seu primeiro modelo para detecção de objetos” desenvolvido pela iniciativa Computação na Escola/INCoD/INE/UFSC (Martins, 2022). Nesse curso, o estudante aprende a desenvolver um modelo YOLOv5 (Ultralytics, 2021) de detecção de objetos de casa (mesa, sapato etc.) dentro de um Jupyter Notebook usando ODIN (Santana, 2022), uma camada visual que permite o desenvolvimento de ML nos Jupyter Notebook por meio da *interface* visual. Porém, atualmente esses cursos muitas vezes ainda não abordam a implantação desses modelos de detecção de objetos em sistemas web ou aplicativos móveis (Marques *et al.*, 2020) pela falta de infraestrutura que permita uma implantação simplificada. Uma alternativa de implantação pode ser em aplicativos móveis desenvolvidos com App Inventor. O MIT App Inventor é um ambiente de programação com uma linguagem de blocos visuais que possibilita às pessoas que desejam iniciar na programação de aplicativos, criar aplicações de uma forma interativa e facilitada (Wolber, 2020). A utilização do App Inventor ajuda na introdução de tópicos sobre computação na escola, permitindo que os alunos possam abstrair os conceitos mais profundos sobre codificação e ser inspirados pela possibilidade de produzir um aplicativo Android funcional (Patton, 2019). O App Inventor disponibiliza uma plataforma que permite que extensões sejam criadas para adicionar funcionalidades ao App Inventor. Inclusive já existem extensões para App Inventor, como PIC (Tand *et al.*, 2019) e TMIC (Oliveira, 2022) que permitem criar apps inteligentes voltados à classificação de imagens. Entretanto, ainda não há uma extensão para App Inventor que permita a implantação de modelos de detecção de objetos.

Dessa forma, o presente trabalho busca desenvolver uma extensão que integre um aplicativos criados no App Inventor a um *web service* hospedado no PythonAnywhere por meio de uma API Flask. Essa integração permitirá o uso de modelos YOLOv5 treinados para detecção de objetos com e exportados no formato ONNX.

1.2 OBJETIVOS

Objetivo geral

Dentro do contexto do ensino de ML no ensino médio, visa-se neste TCC o desenvolvimento de uma extensão do App Inventor para possibilitar a integração de aplicativos desenvolvidos na plataforma a um *web service* que realiza a detecção de objetos com base em modelos de *Machine Learning* YOLOv5 treinados com Jupyter Notebook e exportados no formato ONNX.

É desenvolvida uma extensão que permite implantar modelos de ML voltados à tarefa de detecção de objetos. A extensão serve como suporte para evoluir o curso "Crie o seu primeiro modelo para detecção de objetos" (Martins, 2022) abrangendo também a implantação de modelos de ML em aplicativos móveis.

Objetivos Específicos

Os objetivos específicos são:

01. Elaborar a fundamentação teórica em relação ao ensino de ML voltado a detecção de objetos utilizando e desenvolvendo uma extensão para o App Inventor que integre o aplicativo a um *web service* que realiza a inferência utilizando YOLO e o ONNX por meio de uma API Flask;

02. Levantar o estado da arte em relação a extensões do App Inventor semelhantes;

03. Desenvolver a extensão YOLOOD para integrar o App Inventor ao *web service* hospedado no PythonAnywhere;

04. Desenvolver o *web service* com uma API para disponibilizar a detecção de objetos em uma imagem com base de um modelo de ML criado no Jupyter Notebook;

05. Desenvolver o material didático (*slides e* app exemplo) para ensinar o uso da extensão desenvolvida;

1.3 METODOLOGIA DE ESTUDO E TRABALHO

A fim de alcançar os resultados esperados com este trabalho, é adotada uma combinação de metodologias de pesquisa de acordo com o respectivo objetivo a ser buscado. Então, de acordo com os objetivos específicos do projeto são adotadas etapas da seguinte forma:

Etapa 1 – Fundamentação teórica: análise e síntese de conceitos básicos envolvidos no tema.

São abordados conceitos de ML voltado a detecção de objetos. É apresentado o ambiente de programação visual baseado em blocos *App Inventor* (MIT, 2022) e o ambiente para treinamento de modelos de Jupyter Notebook usando YOLOv5 e o formato de exportação ONNX. Este estudo é feito por meio de uma análise e síntese da literatura.

Atividade 1.1: Sintetizar conceitos de *Machine Learning*, detecção de objetos, e YOLOv5 e o formato de exportação ONNX.

Atividade 1.2: Sintetizar conceitos de App Inventor e *framework* de extensões.

Etapa 2 - Levantamento do estado da arte: levantamento sobre trabalhos existentes relacionados à área do projeto.

É realizado um estudo de mapeamento seguindo um processo proposto por Petersen *et al.* (2008) para identificar e analisar extensões do App Inventor para a implantação de modelos de ML.

Atividade 2.1: Definir o protocolo de busca.

Atividade 2.2: Executar a busca.

Atividade 2.3: Extrair e analisar as informações.

Etapa 3 - Desenvolvimento da extensão YOLO OD seguindo um processo de desenvolvimento de *software* (Pressman, 2021), contemplando a análise dos requisitos, modelagem, implementação e teste:

Atividade 3.1: Analisar os requisitos: Levantamento e análise do problema identificando os requisitos funcionais e não-funcionais do *software*.

Atividade 3.2: Modelagem alto nível: Definição da arquitetura do sistema.

Atividade 3.3: Implementar e testar: Desenvolver o sistema baseado nos requisitos e modelagem, e realizar os testes de funcionalidades.

Atividade 3.4: Modelagem detalhada: Detalhar a modelagem do sistema e a implementação.

Etapa 4 - Desenvolvimento do material didático. Para facilitar o uso da extensão e configuração da API, será desenvolvido um material didático incluindo um app exemplo implantando um modelo YOLOv5 e *slides*.

Atividade 4.1: Desenvolvimento do material didático (*slides*)

Atividade 4.2: Desenvolvimento de app exemplo

1.4 ESTRUTURA DO DOCUMENTO

No capítulo 2, é apresentada a fundamentação teórica dos conceitos relacionados à proposta deste TCC. No capítulo 3, apresenta-se o levantamento do estado da arte e quais extensões existem para implementar modelos de *Machine Learning* para detecção de objetos no App Inventor. No capítulo 4, é apresentada a proposta de solução, juntamente aos seus requisitos funcionais e não funcionais. No capítulo 5, é feita a apresentação do aplicativo desenvolvido como exemplo. Já no capítulo 6 é descrita a aula e os slides para ensinar a utilizar a extensão e no capítulo 7 é realizada a conclusão sobre o trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

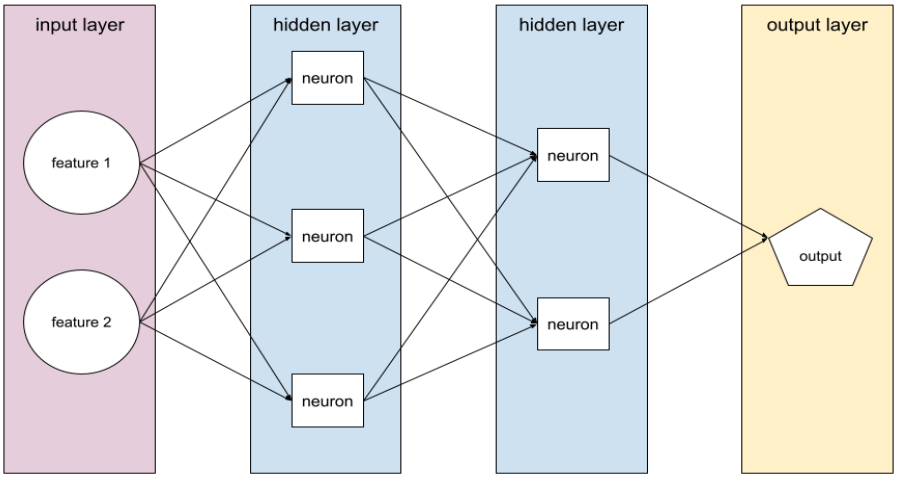
Neste capítulo são apresentados conceitos sobre *Machine Learning, Deep Learning*, Redes Neurais, detecção de objetos, YOLO, ONNX e o App Inventor. Buscando tornar claro os conceitos que envolvem Desenvolvimento de uma Extensão do App Inventor para integrar o aplicativo a uma API com *Machine Learning* para Detecção de Objetos em Aplicativos.

2.1 DETECÇÃO DE OBJETOS COM *DEEP LEARNING*

Inteligência artificial (IA) generaliza vários campos de estudos dentro da ciência da computação, incluindo, *Machine Learning e Deep Learning* (Goodfellow *et al.*, 2016). O *Machine Learning (ML)* é uma vertente no amplo campo da IA que busca representar computacionalmente a maneira como a inteligência humana é desenvolvida por meio da aprendizagem permitindo que os computadores aprendam sem serem explicitamente programados (Mitchell, 1997). Uma das técnicas de ML são redes neurais artificiais, o funcionamento das redes neurais artificiais visa ser parecido ao cérebro humano quando se trata de troca e processamento de informação entre os neurônios, criando uma rede neural artificial (Lecun *et al.*, 2015).

Um subconjunto de ML é o *Deep Learning (DL)*, que utiliza modelos computacionais compostos por múltiplas camadas de processamento para aprender a representar dados com múltiplas camadas de abstração, chamadas de redes neurais profundas (Lecun *et al.*, 2015). Esse tipo de rede neural contém mais de uma camada escondida. Isso pode ser observado na Figura 1, que exemplifica uma rede neural profunda contendo duas camadas escondidas.

Figura 1 - Exemplo de rede neural profunda com duas camadas escondidas



Fonte: Google, 2023

Em uma rede neural, cada neurônio se conecta a todos os neurônios da camada subsequente, associados a um peso e um valor. Na Figura 1, todos os 3 neurônios se conectam aos 2 neurônios da segunda camada (Google, 2022).

O aprendizado de uma rede neural pode ser realizado por aprendizado supervisionado e aprendizado não supervisionado e outros. No aprendizado supervisionado o processo se baseia em comparar o resultado de saída obtido com o resultado de saída esperado, e com isso pode ser calculado o erro e então utilizado esse valor para ajustes a fim de alcançar o resultado esperado (Dey, 2016).

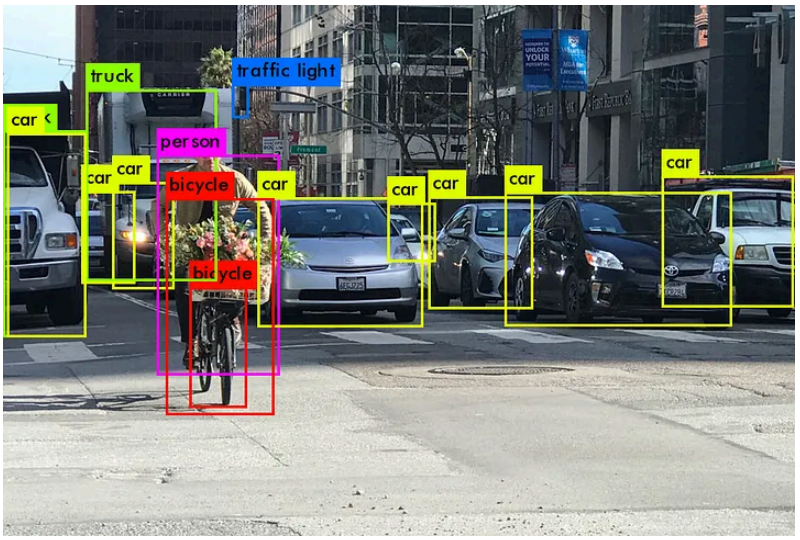
2.2 *Detecção de Objetos*

Detecção de objetos é uma tarefa computacional responsável por identificar visualmente instâncias de objetos de classes pré-definidas, em fotos ou vídeos. O objetivo dos estudos na detecção de objetos é desenvolver modelos capazes de assimilar onde está cada objeto na imagem (Jain *et al.,* 2016).

O processo de detecção de objetos pode ser dividido em (Zhao, 2018): a proposição de região, extração de recursos e classificação de região. Primeiramente, as regiões com possíveis objetos nas imagens são propostas junto a indicação de áreas que possam conter os objetos. Após a primeira etapa ocorre a extração de recursos, onde os relevantes são selecionados para formar um vetor de características, e então esse vetor é classificado para determinar o tipo de objeto.

Por último, depois de realizar as operações pós processamento, como a limitação dos máximos e definição da posição da borda, é obtida a moldura final do objeto.

Figura 2 - Imagem com objetos sendo detectados



Fonte: Shah, 2020

A Figura 2 apresenta um exemplo funcional de detecção de objetos, onde as localizações de carros, caminhões, bicicletas, pessoas e semáforos são identificados na imagem por meio das *bounding boxes*. As *bounding boxes* são utilizadas para representar a localização de objetos em uma imagem, envolvendo todo o objeto em questão e sendo delimitadas por uma posição final e inicial nos eixos x e y (Redmon *et al.*, 2016).

A avaliação de desempenho de um modelo de detecção de objetos costumeiramente é baseada na métrica *Intersection Over Union*, que é baseada no índice Jaccard, que mede a sobreposição de duas *bounding boxes,* entre a *bounding box* predita e a *bounding boxes* real do objeto. Com o valor de 0.5 IOU, que significa que o modelo está predizendo uma *bounding box* com pelo menos 50% de sobreposição sobre a posição da *bounding box* real, geralmente pode-se afirmar que é uma predição correta (Zou *et al.* 2019). Comumente também se mede com 0.9 IOU.

Aplicando a métrica IOU, pode-se classificar as predições como:

- True Positive (TP) — Detecção correta com IOU maior ou igual que o valor mínimo desejado.

- False Positive (FP) — Detecção incorreta com IOU menos que o valor mínimo desejado.
- False Negative (FN) — Detecção perdida, ou seja, *bounding box* real não encontrada.

Para garantir que o modelo tenha alcançado a capacidade de detecção desejada, podem ser utilizadas as seguintes métricas:

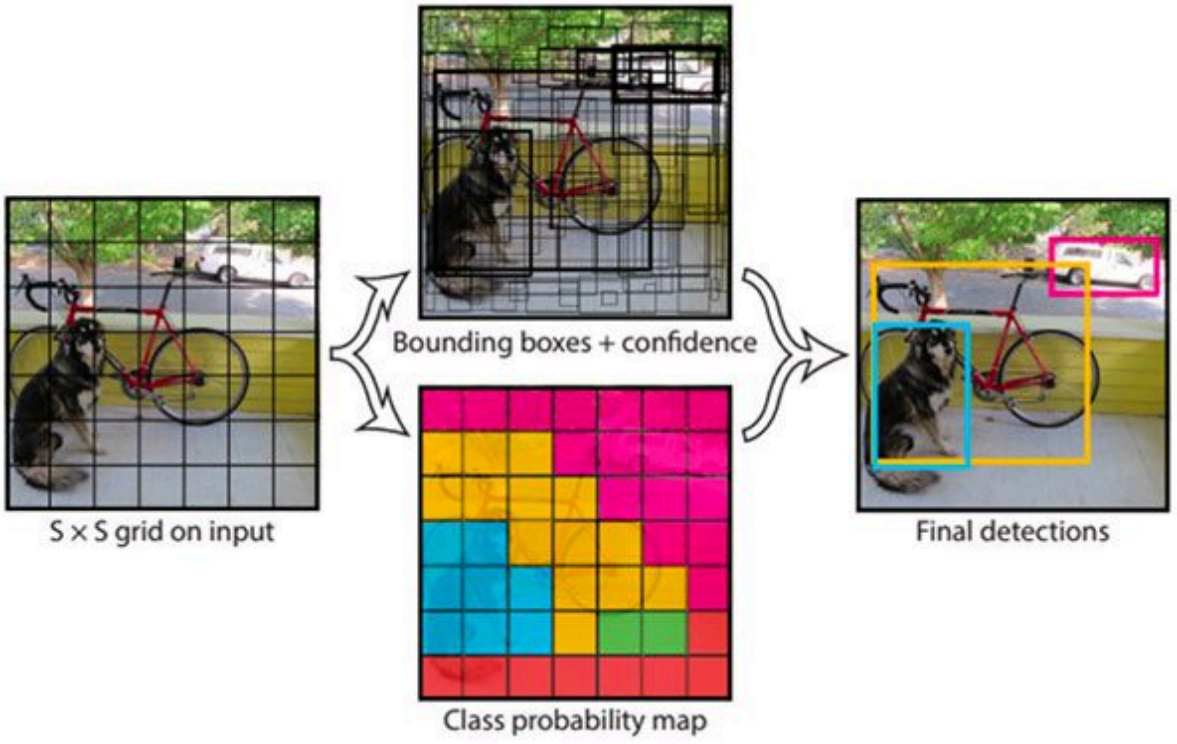
- *Precision and recall*
  - *Precision* proporção de detecções corretas em relação ao número total de predições positivas (TP / TP+FP).
  - *Recall* é a proporção das detecções corretas sobre todos os positivos reais (TP / TP+FN).
- *Average precision (AP)* calcula a precisão média de diversos recalls, atuando somente sobre uma categoria.
- *Mean Average Precision (mAP)* é a *AP* média generalizada para todas as classes do modelo.
  - mAP05 é o valor utilizado para avaliar como corretas as predições que tenham o IOU de pelo menos 0.5. Ou seja, exige que a predição tenha pelo menos 50% de sobreposição sobre o objeto real.
  - mAP09 utiliza IOU de 0.9 como base para avaliar a predição como correta. Exigindo que a predição tenha pelo menos 90% de sobreposição sobre o objeto real.

Existem diversos modelos para detecção de objetos, entre eles o *Region-based Convolutional Neural Network* (R-CNN) (Girshick, 2014), SSD (Liu, 2016), Fast R-CNN (Girshick, 2015) e o YOLO (Ultralytics, 2021). Esses modelos podem ser separados em dois tipos de detectores, *one-stage detector e two-stage detector*. Uma rede que possui módulos separados para gerar as regiões propostas é chamada de *two-stage.* Nesse modelo, inicialmente é feita a procura pela quantidade de objetos para que na segunda etapa os objetos sejam classificados e localizados. Um modelo de detecção de objetos *one-stage* consegue classificar e localizar os objetos em uma só etapa tornando esse tipo de detector usualmente mais rápido (Zaidi, 2022).

Um dos detectores *one-stage* é o YOLO (*You Only Look Once*) (Ultralytics, 2023). Por ser um modelo que trata a classificação e a identificação das *bounding*

boxes na mesma etapa, ele se torna extremamente rápido. Assim, a detecção de objetos utilizando o YOLO é 1000x mais rápida que a detecção utilizando outras arquitetura (Ultralytics, 2021). Desde a criação do yolo diversas versões foram lançadas trazendo novos aspectos e peculiaridades. Em 2015, a primeira versão do YOLO implantou um novo conceito para detecção de objetos em tempo real, utilizando uma única rede neural convolucional (CNN) para dividir a imagem em grades e aplicar as regressões em cada célula para conseguir definir a localização e a classificação de cada objeto (Redmon & Farhadi. 2015), como mostra a Figura 3.

Figura 3 - Demonstração do funcionamento do sistema de grades do YOLO



Fonte: Ultralytics, 2021

Como resultado da detecção é retornado um arquivo JSON incluindo as seguintes informações: cx, cy, w, h, conf, pred_cls. Sendo:

- **cx**- distância do centro da *bounding boxes* em relação ao eixo x do *grid.*
- **cy**- distância do centro da *bounding boxes* em relação ao eixo y do *grid.*
- **w-** largura da *bounding box.*
- **h**- altura da *bounding box.*

- **conf**- valor de precisão da *bounding box.*
- **pred_cls -** probabilidade do objeto predito pertencer a cada uma das classes possíveis do modelo.

Abaixo tem um exemplo de retorno de uma imagem com 3 instâncias de objetos detectados, em um modelo com 85 possíveis classes (Figura 4).

Figura 4 - *Output* de detecção realizada com o YOLO

```
(1, 3, 80, 80, 85) # anchor 0
(1, 3, 40, 40, 85) # anchor 1
(1, 3, 20, 20, 85) # anchor 2
```

Fonte: Cochard, 2021

YOLOv2 lançado em 2016, trouxe melhorias significativas de desempenho e passou a utilizar uma rede neural mais profunda. Com a utilização de técnicas de detecção em múltiplas escalas, ancoragem das caixas e dimensionamento de clusters conseguiu melhorar a precisão da detecção dos objetos (Redmon & Farhadi. 2016).

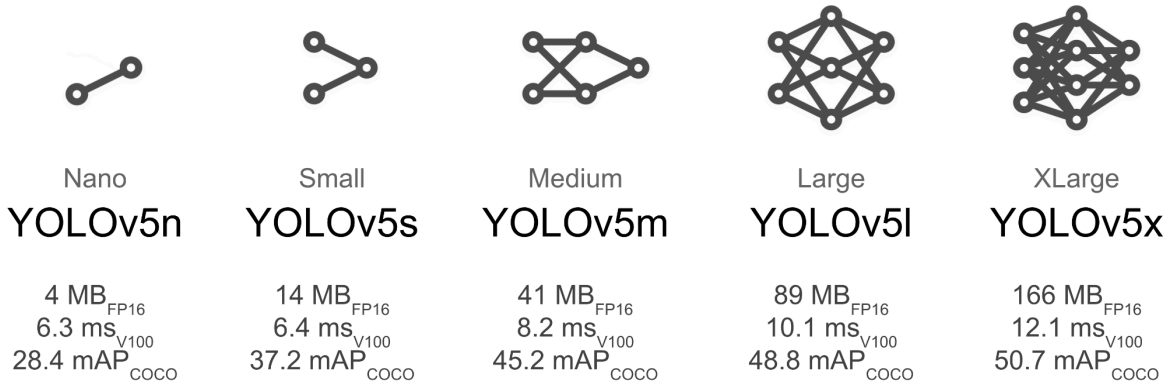
A terceira versão do YOLO, YOLOv3, foi desenvolvida em 2018 apresentando melhorias de desempenho em relação às versões anteriores. Passou a utilizar a Darknet-53, que é uma rede neural mais profunda com 53 camadas convolucionais, apresentando então uma arquitetura mais complexa em relação às 19 camadas anteriores. Nessa versão a detecção em objetos de diferentes tamanhos de imagens e o uso de mapa de característica melhorou mais ainda a precisão

Em 2020, o YOLOv4 foi lançado. Além de melhorias em relação à versão anterior, a versão introduziu novas técnicas avançadas de pós-processamento para a eliminação de falsos positivos e o aumento da precisão da detecção em objetos pequenos (Redmon & Farhadi, A. 2016). Esses avanços foram vistos explicitados nas métricas que apresentaram 10% de melhora no AP (Average Precision) (Bochkovskiy *et al.*, 2020).

A quinta versão do YOLO, o YOLO 5, trouxe ainda mais aprimoramentos do que suas versões anteriores, principalmente por adotar uma arquitetura mais simples, tornando-o mais rápido. O YOLOv5 apresenta algumas versões com

diferentes tamanhos. Consequentemente, a versão com a maior rede neural apresenta uma melhor performance. Porém, a sua maior quantidade de parâmetros torna o treinamento mais demorado e necessitando de mais memória de processamento.

Figura 5 - Versões do YOLO 5

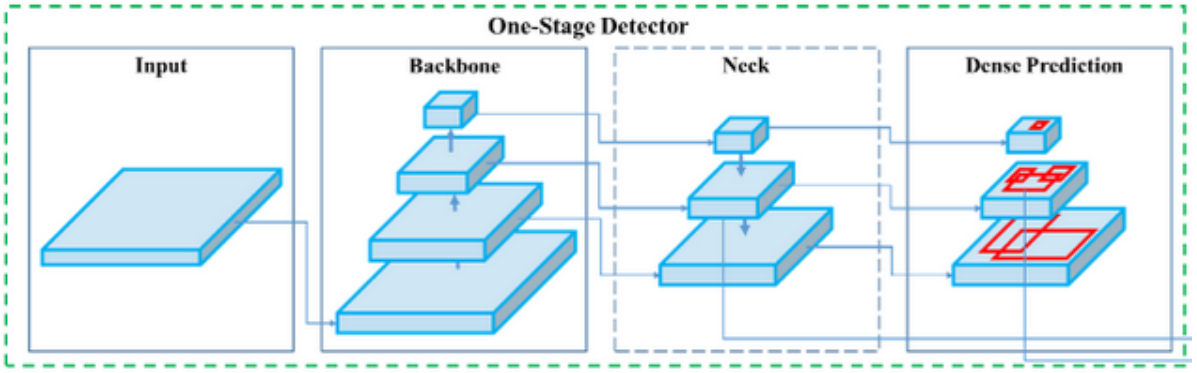


Fonte: Ultralitycs, 2021

Com isso, as versões mais indicadas para implantação em aplicativos mobile são as versões YOLOv5s/m, enquanto as versões YOLOv5l/x são recomendadas para implantação em nuvem.

A arquitetura do YOLOv5 é baseada em uma rede neural convolucional, composta por três componentes: *Backbone*, *Neck* e *Head.*

Figura 6 - Arquitetura do YOLOv5



Fonte: Bochkovskiy, 2020

O *backbone* é uma rede pré-treinada, responsável por extrair as características da imagem. Essa etapa reduz a dimensão das características presentes na imagem e aumenta a resolução das características.

Neck é a camada intermediária que conecta o *Backbone* ao *Head.* É responsável por reduzir ainda mais as dimensões das características da imagem, como por exemplo aumentar a resolução de formas e contornos. Ajudando na generalização do modelo para lidar com objetos de diferentes tamanhos.

E o último modelo é o *Head* que é usado para realizar as operações na etapa final. Produzindo caixas delimitadoras para renderizar a saída com as classes, *scores* e as *bounding boxes.*

O YOLO6, lançado em 2022, manteve cinco tamanhos de modelos, YOLOv6-N, YOLOv6-S, YOLOv6-M, YOLOv6-L e YOLOv6-L6. Essa versão do YOLO passou por um remodelagem na arquitetura de sua rede o que permitiu melhorias significativas na performance do modelo.

A sétima versão do YOLO (YOLOv7) passou por alterações na sua rede e na rotina de treinos. Tiveram 3 pontos principais de melhoria, sendo eles: agregação de uma camada final de extensão para diminuir o tempo do back-propagation, o dimensionamento da profundidade e a largura da rede passaram a ser feitos enquanto acontece a concatenação das camadas e por fim passou a utilizar um novo algoritmo para detectar quais módulos da rede devem utilizar estratégias de reparametrização.

YOLOv8 é mais uma versão desenvolvida do YOLO. Essa versão passou por melhorias na acurácia o que fez com que o YOLOv8 atingisse uma alta taxa de precisão, medida pelos índices COCO e Roboflow 100. Também oferece várias características que facilitam o trabalho dos desenvolvedores, desde uma interface de linha de comando (CLI) fácil de usar, até um pacote Python bem estruturado.

YOLO-NAS (*Neural Architecture Search*) (Ultralitycs, 2023) é uma das mais recentes versões desenvolvidas. Essa versão apresenta melhorias em uma situação que as versões anteriores deixavam a desejar, a falta se suporte a quantização que visa diminuir a memória computacional utilizada pelo modelo. Essa versão também equilibrou a relação entre a precisão e a latência.

2.3 Processo de desenvolvimento de *ML*

O processo de desenvolvimento de ML difere do processo comum de desenvolvimento de software e tem suas peculiaridades. São fases do processo de desenvolvimento (Amershi *et al.*, 2019):

Análise de requisitos. É a fase na qual os objetivos e resultados esperados do modelo de ML são definidos. Para então selecionar o modelo que melhor se encaixa para a solução do problema .

Preparação dos dados. Para essa etapa é importante encontrar um bom conjunto de dados para que possa ser realizado o treinamento e a validação do modelo. É necessário fazer a limpeza dos dados para trazer consistência ao conjunto de dados, identificar e remover dados irrelevantes são tarefas realizadas nessa etapa, caso contrário, terá influência direta no modelo gerado

Rotulação dos dados. Usando aprendizagem supervisionado, as imagens são rotuladas com o objetivo de identificá-los por meio de informações associadas. Dentro do contexto de detecção de objetos, esse processo é destinado para criar *bounding boxes* para um ou mais objetos em uma imagem

Treinamento do modelo. É uma etapa crucial no processo de desenvolvimento de sistemas de *Deep Learning* com o objetivo de alcançar a generalização do modelo treinado e minimizar a diferença entre as saídas previstas e as saídas reais. Na etapa de treinamento, o modelo é alimentado com um conjunto de dados de treinamento e ajustado por meio de um processo iterativo de otimização em que os pesos dos parâmetros do modelo são ajustados para minimizar a função de perda (Redmon *et al.*, 2016).

Avaliação do desempenho. A etapa de avaliação do desempenho é fundamental no processo de desenvolvimento de sistemas de deep learning, pois permite verificar se o modelo desenvolvido atingiu a capacidade de generalização desejada (Géron, 2019). Podem ser utilizadas as métricas *AP e mAP.*

Predição. Nesta etapa, o modelo treinado é utilizado para produzir uma saída para uma entrada desconhecida. Para isso, é utilizado o conjunto de parâmetros aprendidos durante a etapa de treinamento (Goodfellow *et al.*, 2016).

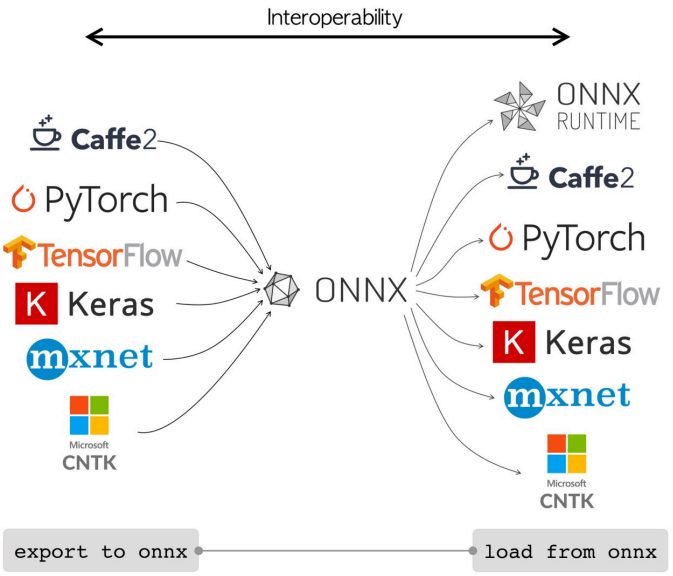
Exportação. É a etapa na qual o modelo é salvo em um formato para que possa ser usado em diversas plataformas e linguagens de programação. *Open Neural Network Exchange (ONNX)* (ONNX, 2023) é um formato que consiste em um modelo serializado em um arquivo *protobuf,* descrevendo a estrutura do modelo

treinado e os parâmetros necessários para executá-lo. A serialização é baseada em grafos, sendo que os nós representam as operações de processamento de dados e as bordas representam a troca de dados entre as operações.

O formato é projetado para ser independente da plataforma. Isso é possível pois a especificação em grafos traz informações sobre os tipos de dados utilizados no modelo e as dimensões dos tensores usados nas operações.

Uma grande vantagem do ONNX é que ele promove a interoperabilidade entre plataformas (Figura 7).

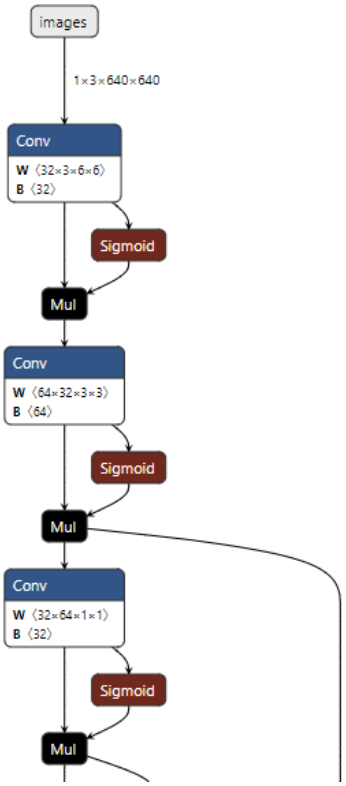
Figura 7 - Interoperabilidade do ONNX



Fonte: López, 2020

Um arquivo ONNX é composto por informações relativas ao modelo treinado (ONNX, 2023). A estrutura do arquivo pode ser dividida em partes. A definição de um modelo de grafos computacional. Definição padrão de tipo de dados. Definição de operadores internos. O ONNX armazena os dados no formato *Protocol Buffer*, fazendo com que o método de serialização dos dados possua sua própria representação e seu próprio compilador. Na Figura 8 é mostrado o início de um grafo gerado a partir de um modelo treinado com YOLOv5 e exportado no formato ONNX.

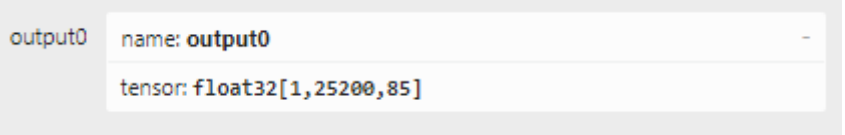
Figura 8 - Grafo de um modelo um arquivo ONNX



Fonte: Autor, 2023

O output de um modelo YOLO exportado em ONNX pode ser visualizado na figura 9. Por padrão, o tensor de saída no formato float32[1, 25200, X], sendo que X varia de acordo com a quantidade de classes que o modelo pode detectar. No modelo abaixo, a primeira (1) dimensão representa o *batch size*, indicando que o modelo processa uma imagem por vez. A segunda dimensão (25200), corresponde ao número total de ancoragens, resultante do produto entre o número de grid cells e o número de âncoras. A terceira dimensão (85), fornece informações detalhadas sobre cada ancoragem: 4 valores para as coordenadas do bounding box (x, y, largura, altura), 1 valor para a confiança da detecção e 80 valores para a probabilidade de cada classe. Assim, para cada uma das 25200 ancoragens, o modelo gera 85 valores que descrevem as propriedades do objeto detectado.

Figura 9 - *Output* de um modelo YOLO exportado no formato ONNX



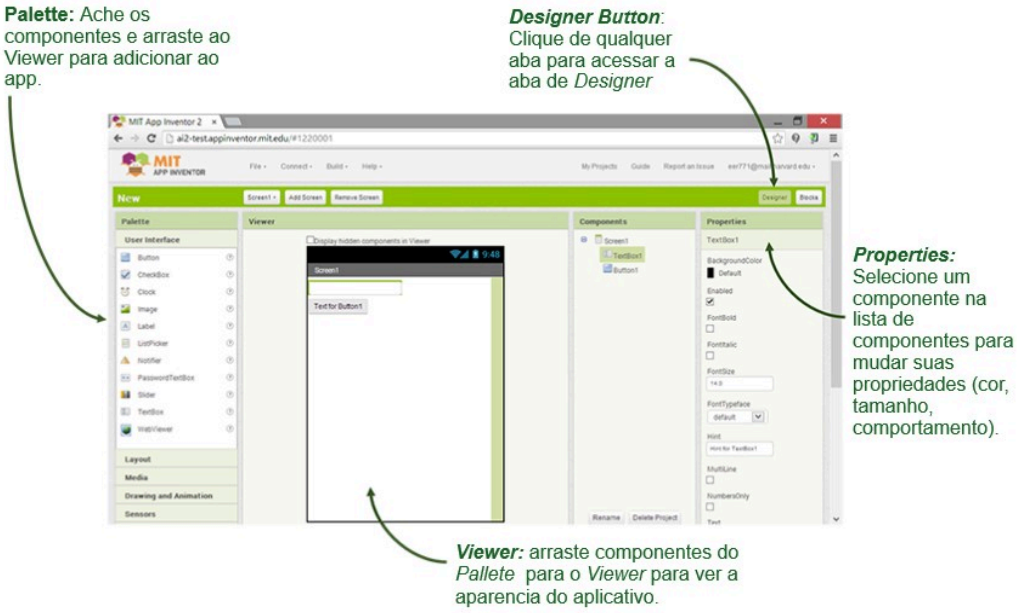
Fonte: Autor, 2024

2.2 APP INVENTOR

O MIT *App Inventor* é uma plataforma de programação visual que permite a qualquer pessoa, incluindo crianças, criar aplicativos completamente funcionais para dispositivos móveis Android, iPhones e *tablets* Android/iOS. Essa plataforma permite aos usuários arrastar e soltar elementos gráficos para criar o aplicativo sem exigir uma base de conhecimentos sobre programação. O App Inventor é comumente utilizado em projetos educacionais (Patton *et al.*, 2019).

A plataforma consiste em uma aba de edição de *Designer* e outra de *Blocks.* Na aba *Designer* encontram-se os componentes que podem ser adicionados ao projeto como botões, labels, imagens, *TextBoxes*. Ainda, cada componente pode ter suas propriedades personalizadas como alteração de cor, tamanho e estilo. Ao adicionar um componentes, as alterações podem ser visualizadas no campo na seção *Viewer* (Figura 10).

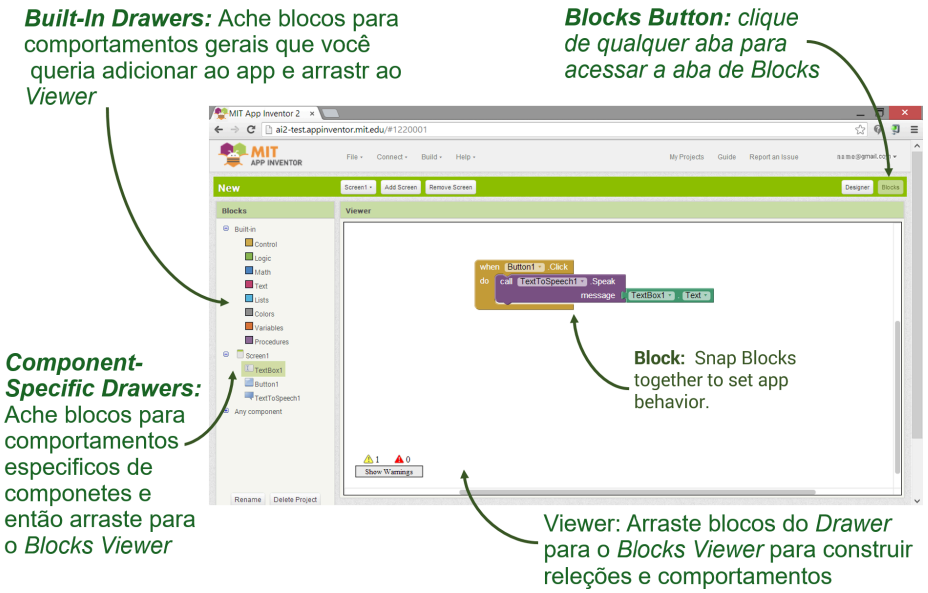
Figura 10 - Aba *designer* do App Inventor



Fonte: MIT, 2022

O *Blocks editor* é a aba onde é definido o comportamento do *app* por meio da associação de blocos que lidam com as ações dos componentes. Os componentes já adicionados ao app estão disponíveis na guia *Blocks* para serem relacionados à lógica do aplicativo. Os blocos representam operações lógicas, funções e eventos que podem interagir com os componentes visuais.

Figura 11 - Aba *blocks* do App Inventor



Fonte: MIT, 2022

Além das funções nativas do App Inventor, também é possível que a comunidade desenvolva e adicione extensões para a plataforma. O App Inventor fornece um *App Inventor extensions* (MIT, 2015) para o desenvolvimento dessas extensões personalizadas. As extensões podem ser escritas nas linguagens Java ou Kotlin. Após o desenvolvimento a extensão pode facilmente ser adicionada ao projeto por meio da própria interface gráfica da plataforma. Também é possível que um desenvolvedor disponibilize sua extensão no repositório de extensões do App Inventor, para que outras pessoas possam utilizá-la. A extensão é constituída pela adição de um ou mais blocos personalizados possibilitando que sejam adicionadas novas funções, integrações com APIs externas, acessos a sensores de dispositivos e novos formatos de leitura e gravação de dados aos aplicativos desenvolvidos com o App Inventor.

Para desenvolver uma extensão para o App Inventor deve-se preparar o ambiente de desenvolvimento, instalando as plataformas necessárias e configurando

as variáveis de ambientes utilizadas pela instalação local do App Inventor (App Inventor, 2021). Inicialmente, o componente deve ser criado para testes no servidor local do App Inventor. Após a verificação do pleno funcionamento do componente, é utilizado o *App Inventor Extension Template* para converter o componente em uma extensão. Com a exportação é gerado um arquivo .aix, sendo esse o arquivo que será disponibilizado para importação no App Inventor a fim de difundir a utilização da extensão.

Existem também outras plataformas que permitem meios de desenvolver aplicativos através da programação visual, como Kodular (Kodular, 2021) e o Thunkable (Thunkable, 2023). Kodular é uma plataforma similar ao App Inventor e teve sua criação baseada em expandir e melhorar as suas funcionalidades. Enquanto o Thunkable é uma plataforma que também permite criar aplicativos de forma intuitiva, permitindo que recursos mais avançados sejam desenvolvidos, como o acesso a APIs externas, acesso a diversos sensores do dispositivo e o suporte para múltiplas plataformas.

3. ESTADO DA ARTE

O levantamento do estado da arte para encontrar soluções existentes para implantar modelos de ML de detecção de objetos em apps com App Inventor foi realizado através de uma revisão sistemática.

3.1 DEFINIÇÃO DO PROTOCOLO DE REVISÃO

O objetivo desta revisão é responder a seguinte questão: Quais soluções existem para implantar modelos de ML de detecção de objetos em apps com App Inventor? Com base no objetivo desta revisão, a pergunta de pesquisa é dividida nas seguintes questões de análise:

AQ1. Quais soluções existem?

AQ2. Em quais ambientes de desenvolvimento os modelos de ML são treinados e exportados?

AQ3. Como os modelos treinados são importados nas soluções?

AQ4. Quais são as características funcionais dessas soluções?

Critérios de inclusão/exclusão. Foram consideradas somente soluções para o App Inventor que importam modelos de ML para detecção de objetos. Extensões do App Inventor voltadas a outras tarefas como, por exemplo, classificação de imagens ou reconhecimento de fala são excluídas.

Critérios de qualidade. Foram considerados apenas artigos ou materiais com informações suficientes e/ou as extensões relacionadas à descrição das extensões ou aplicações do App Inventor.

Fontes dos dados. Foram analisados os materiais e artigos publicados em inglês e disponíveis no Scopus, sendo uma das mais importantes bibliotecas digitais acessíveis por meio do Portal CAPES. Foi também realizada uma busca via Google Scholar, por ser uma grande fonte de conteúdo publicado. Dado o foco de pesquisa do MIT Media Lab (MIT, 2023) nessa área e foco técnico, são buscadas também extensões nas páginas deste grupo, no fórum do App Inventor (MIT, 2023) e na lista das extensões do puraVida (Puravida, 2023).

Definição da *string* de busca. A *string* de busca foi composta de conceitos relacionados à questão de pesquisa, incluindo sinônimos. Dessas palavras chave, a string de busca foi adaptada para cada fonte de dados apresentada na Tabela 1.

Tabela 1 - Termos Chave

Termo chave	**Sinônimos**
“object detection”	"machine learning", "artificial intelligence", "deep learning"
“App Inventor”	kodular, thunkable

A *string* de busca genérico foi definido da seguinte forma:

(“object detection” OR "machine learning" OR "artificial intelligence" OR "deep learning") AND (“App Inventor” OR kodular OR thunkable)

O string de busca genérico foi ajustado para cada uma das fontes conforme apresentado na Tabela 2. A string de busca não foi utilizada nas buscas realizadas nos fóruns de comunidades voltados ao desenvolvimento e distribuição de extensões do App Inventor, pois não existem ferramentas de busca por string nestes ambientes.

Tabela 2 - String de busca para cada fonte

Fonte	**String de busca**
Scopus	SCOPUSTITLE-ABS-KEY (( ) AND ()) AND (LIMITTO (SUBJAREA , "COMP"))
Google Scholar	(“object detection” OR "machine learning" OR "artificial intelligence" OR "deep learning") AND (“App Inventor” OR kodular OR thunkable)
MIT Media Lab (https://www.media.mit.edu/)	“artificial intelligence” “object detection” “extensions
Lista das extensões (puraVida) (http://puravidaapps.com/extensions.php)	–
App Inventor Forum (https://community.appinventor.mit.edu/)	artificial intelligence

3.2 EXECUÇÃO DA BUSCA

A busca foi realizada em maio de 2023 pelo autor e revisada pela orientadora. A quantidade de resultados encontrados estão na (Tabela 3).

Tabela 3 - Quantidade de artigos identificados por repositório e por fase de seleção

	No. de resultados da busca	No. de resultados analisados	No. de resultados potencialmente relevantes	No. dos resultados relevantes
Scopus	28	28	2	0
Google Scholar	2.720	200	17	1
App Inventor Forum	340	200	1	1
Total (sem duplicatas)				2

Na primeira etapa de análise da busca, títulos, resumos e fóruns foram analisados, resultando em 20 artefatos com potencial relevante. Na segunda etapa, os materiais foram lidos por inteiro, para garantir que estivessem dentro dos critérios de relevância definidos para inclusão/exclusão. Como resultado, 2 aplicações que realizam a detecção de objetos e foram implementadas no App Inventor foram consideradas relevantes (Tabela 4).

3.3 RESULTADOS DA REVISÃO

Baseando-se nas perguntas de análise, as informações foram extraídas dos artigos/materiais encontrados para cada uma das soluções. Apenas uma das soluções foi divulgada em formato de artigo, o que dificultou a extração de dados detalhados sobre uma das aplicações.

3.3.1 Quais soluções existem?

Não foi encontrada nenhuma extensão que permitisse integrar o aplicativo a um *web service* por meio de uma API na qual o desenvolvedor faça *upload* do seu próprio modelo treinado. Foram encontrados apenas exemplos de aplicações que tiveram seu *frontend* desenvolvido no *App Inventor* integradas a APIs de servidores que realizavam as tarefas relacionadas à detecção de objetos com modelos padrões do serviço.

Tabela 4 - Soluções encontradas.

Nome da solução	Breve descrição	Extensão para integração	Referências	Link	Licença de uso

		no App Inventor			
EdgeLens	É um *framework* que cria um ambiente de detecção de objetos em um ambiente IoT, Fog and *Cloud Computing*	Detecção de objetos ocorre via webservice	–	https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9036216	--
RecyLink	Realiza a detecção de objetos (garrafas plásticas) por meio de um webservice	Detecção de objetos ocorre via *web service*	–	https://www.youtube.com/watch?v=SBIGcgxxxZk&t=4s	–

Pode-se observar ao analisar os fóruns do MIT relacionados ao App Inventor, que detecção de objetos é um tema de interesse da comunidade, dado que foram encontrados tópicos com questionamentos sobre como utilizar extensões de classificação de imagem para detectar objetos.

Duas soluções foram encontradas. O primeiro aplicativo encontrado foi o EdgeLens, aplicação que integra detecção de objetos a um ambiente de IoT, *Fog* e *Cloud Computing*. O aplicativo foi desenvolvido no App Inventor e permite que o usuário envie imagens para um servidor via *WebService,* onde a detecção de objetos na imagem é processada. A segunda solução, o Recylink, também é um aplicativo desenvolvido no App Inventor que realiza a tarefa de detecção e contagem de garrafas PET. O aplicativo permite ao usuário tirar fotos das garrafas, que são enviadas via API para um servidor que processa a detecção na imagem e retorna os dados sobre os objetos encontrados..

3.3.2 Quais modelos de *ML* são utilizados e como eles exportam os modelos?

Buscou-se identificar quais os modelos de ML são utilizados por cada solução encontrada. Os resultados podem ser visualizados na Tabela 5.

Tabela 5 - Modelos utilizados nas soluções.

Nome da extensão	Modelo utilizado	Formato dos modelos exportação
EdgeLens	YOLOv3	Utiliza um modelo próprio pré-treinado fixo no código que é executado em um módulo em Python com YOLOv3
RecyLink	YOLOv5	Utiliza um modelo pré-treinado fixo no código

Observa-se que predominantemente o YOLO é usado para a tarefa de detecção de objetos. Na primeira solução encontrada (EdgeLens), utiliza-se a versão 3 do YOLO enquanto o RecyLink utiliza o YOLOv5. Ambas utilizam modelos inseridos de forma estática no código da aplicação, não permitindo que os usuários insiram um modelo treinado de forma customizada.

3.3.3 Quais são as características funcionais dessas extensões?

As soluções encontradas possuem a capacidade de realizar a tarefa de detecção de objetos, porém não permitem que o modelo possa ser alterado. Ambas as soluções possuem uma API em que enviam a imagem para ser processada em um *web service* onde então ocorre o processamento da imagem para que possa ser obtida uma resposta com as informações relacionadas aos objetos encontrados.

Ambos os aplicativos dependem da internet para realizar as tarefas relacionadas a detecção de objetos, uma vez que as imagens são enviadas para uma API. Com isso, necessitam hospedar o modelo treinado num servidor web para poder acessar via API .

Por não se enquadrarem propriamente como extensões do App Inventor, as soluções encontradas também não fornecem blocos novos para programação.

Tabela 6 - Características funcionais das extensões.

Nome da extensão	Mecanismo de funcionamento	Quantidade de blocos	Tipos de funções dos blocos	Funciona independente de acesso à internet?
EdgeLens	Roda em servidor acessado via API	–	–	Não
RecyLink	Roda em servidor	–	–	Não

	acessado via API			

3.4 DISCUSSÃO

Com o levantamento do estado da arte, não foi encontrada nenhuma extensão voltada à detecção de objetos, mesmo que existam extensões para outras tarefas de ML como classificação de imagens ou áudio. Somente foram encontradas duas soluções de aplicativos voltadas à detecção de objetos, desenvolvidas com o *App Inventor.* Essas soluções permitem que os usuários realizem tarefas de detecção de objetos, sendo uma delas para detecção e contagem de garrafas plásticas e a outra de objetos de forma geral.

Os aplicativos possuem cada um o seu modelo treinado de forma independente e hospedado de forma fixa em um servidor web, não permitindo que os usuários personalizem o modelo treinado. Também não é especificada a forma como esses modelos foram treinados. Com isso os aplicativos desenvolvidos podem acessar os modelos treinados via webservice/API para implantar a funcionalidade de detecção de objetos. Isto, então possibilita a criação de apps com detecção de objetos. Porém, no contexto educacional este tipo de solução não engloba todo o processo de desenvolvimento e aprendizagem de ML, uma vez que não é possível utilizar um modelo personalizado. Então, uma outra alternativa seria criar um *web service* integrado a uma extensão do App Inventor, permitindo o upload de um modelo treinado, para que a extensão envie as imagens por uma API, para realizar a inferência com base no modelo carregado e retornar a imagem com a detecção de objetos realizada.

Ameaças a validade. Com o objetivo de reduzir as ameaças à validade dos resultados da revisão, foram identificadas possíveis ameaças e implementadas estratégias para minimizá-las. É possível que resultados favoráveis tenham mais probabilidade de serem publicados do que resultados negativos. No entanto, não consideramos isso como uma ameaça significativa, uma vez que o objetivo deste estudo consiste em encontrar e descrever ampliações que possam ser utilizadas na plataforma App Inventor para o desenvolvimento de aplicações móveis de detecção de objetos. Apenas uma das aplicações foi descoberta por meio de trabalhos acadêmicos, enquanto as demais foram encontradas em fóruns especializados na área de desenvolvimento de aplicações. Assim sendo, este estudo não se concentra

no resultado positivo em si, mas sim na mera existência e descrição das funcionalidades das aplicações encontradas.

Para evitar a omissão de artefatos relevantes, foram consultadas várias fontes. Para cada fonte foi utilizada a *string* de busca com a maior abrangência possível, incluindo extensões que pudessem ter sido criadas para o MIT App Inventor, Kodular ou Thunkable. Ameaças à seleção de estudos e extração de dados foram mitigadas através da definição de forma detalhada dos critérios de inclusão e exclusão.

4. YOLOOD: Extensão do App Inventor para Integrar Detecção de Objetos

Neste capítulo, é apresentado o desenvolvimento da extensão **YOLOOD** (*YOLO Object Detection*) para o App Inventor e do *web service.* Essa extensão possibilita a integração entre o aplicativo desenvolvido com o App Inventor e o *web service* que roda a inferência sobre o modelo de ML.

4.1 ANÁLISE DE REQUISITOS

O público alvo da extensão são alunos da educação básica a partir de 14 anos que possuem somente um conhecimento inicial sobre programação, IA/*ML.* Com isso, a extensão busca facilitar o processo de aprendizagem com a prática, permitindo que os estudante possam criar aplicativos que realizam a tarefa de detecção de objetos de imagens capturadas, fixando os conceitos aprendidos no curso “Crie o seu primeiro modelo para detecção de objetos” (Martins,2022) e colocando-os em prática.

Foram identificados requisitos funcionais e requisitos não funcionais. Os requisitos funcionais estão descritos na Tabela 7.

Tabela 7 - Requisitos funcionais

ID	**Requisito**	**Descrição**
RF01	Enviar imagens para a API do PythonAnywhere.	Enviar imagens para a API do PythonAnywhere utilizando um método POST e receber a imagem com os objetos detectados como resposta.
RF02	Abrir a câmera traseira/frontal do celular.	A extensão deve utilizar a câmera do celular do usuário para captura de imagem.
RF03	Solicitar a permissão de utilização da câmera do celular.	A extensão deve solicitar ao usuário a permissão de utilização da câmera do celular antes de abri-la.

RF04	Salvar a imagem tirada com a câmera e então enviar para a detecção.	A imagem tirada pela câmera que foi usada para a detecção deve ser salva no celular.
RF05	Apresentar a imagem selecionada ou capturada.	A extensão deve permitir que a imagem capturada ou selecionada seja apresentada na tela enquanto a detecção está sendo processada.
RF06	Visualizar as *bounding boxes* de objetos detectados na imagem salva de acordo com os resultados da detecção.	A extensão deve permitir a visualização das *bounding boxes* na imagem que foi salva e apresentar os atributos relativos àquele objeto.
RF07	Uso de um modelo treinado e exportado em formato ONNX pelo desenvolvedor do aplicativo.	O serviço deve permitir ao usuário carregar o modelo treinado dentro da sua aplicação no PythonAnywhere.
RF08	Manipulação de resultados.	A extensão deve permitir manipular os resultados da detecção via blocos possibilitando uma integração dos resultados na funcionalidade de um app.
RF09	Acesso a imagens armazenadas no celular.	A extensão deve permitir utilizar imagens armazenadas no celular para realizar a detecção de objetos.
RF10	Câmera continuar funcionando ao sair e voltar para a tela de detecção.	Permitir que o usuário troque de tela e ao voltar para a tela da detecção ainda seja possível capturar fotos com a câmera.

Tabela 8 - Requisitos não funcionais

ID	Requisito	Descrição
RF01	*E*xtensão desenvolvida em Java.	Extensão deve ser desenvolvido utilizando a linguagem de programação Java por meio do framework de desenvolvimento do App Inventor.
RF02	*Web service* desenvolvido em Python.	O *web service* e a API devem ser desenvolvidas utilizando a linguagem Python e o framework Flask.
RF03	Sistema operacional Android.	A extensão deve ser implementada em aplicativos para dispositivos com o sistema operacional Android.
RF04	Internacionalização.	Os blocos devem passar pelo processo de internacionalização, ou seja, estar disponíveis em inglês e em português do Brasil

4.2 Modelo conceitual da solução

Inicialmente buscou-se desenvolver uma extensão que permitisse realizar a inferência sobre modelos YOLOv5 diretamente no dispositivo em que o aplicativo

fosse instalado, sem necessitar de acesso a internet. Porém, foram encontrados obstáculos técnicos significativos durante o desenvolvimento.

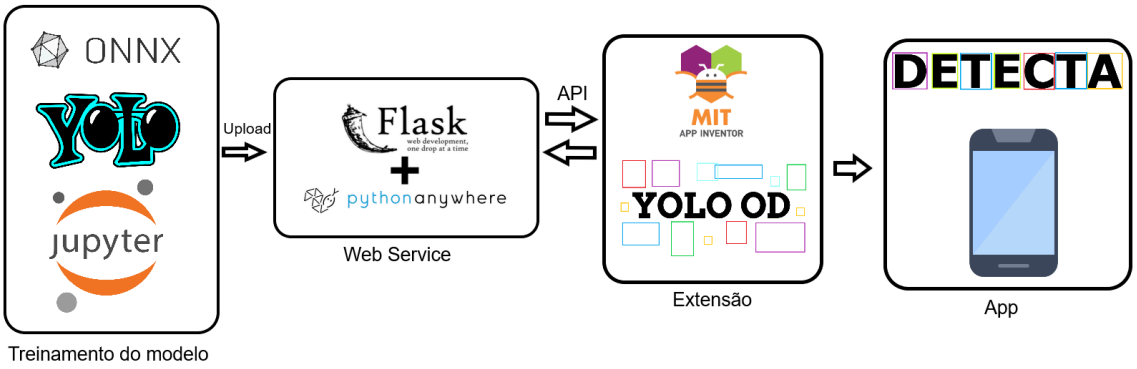
Tentou-se incorporar a biblioteca ONNX Runtime diretamente na extensão do App Inventor. No entanto, a estrutura modular e os requisitos de dependência do ONNX Runtime são complexos, o que dificultou a exportação. O App Inventor utiliza Java para o desenvolvimento de extensões. ONNX Runtime oferece suporte para Java através de JNI (Java Native Interface), mas a integração falhou devido à incompatibilidade entre a versão do SDK usado pelo App Inventor e os requisitos do ONNX Runtime.

Também optou-se por utilizar o Deeplearning4j, sendo uma alternativa ao uso do ONNX Runtime para realizar a inferência. Deeplearning4j é uma biblioteca de código aberto para *deep learning* em Java e Scala, flexível e oferece suporte a múltiplas arquiteturas de redes neurais. Entretanto, é uma biblioteca com diversas dependências e por isso apresentou incompatibilidade com a versão da *Java Virtual Machine (JVM)* utilizada pelo App Inventor.

Contudo, devido aos desafios técnicos relacionados à compatibilidade de bibliotecas como a ONNX Runtime com o ambiente nativo do App Inventor, optou-se por uma abordagem que centraliza a execução da inferência em um servidor externo.

É desenvolvido um web service utilizando Flask hospedado no PythonAnywhere, permitindo que uma extensão do MIT App Inventor envie imagens para detecção de objetos utilizando YOLOv5, retornando a imagem com bounding boxes dos objetos identificados. O modelo YOLOv5 é treinado em um notebook Jupyter, exportado para o formato ONNX, e carregado no PythonAnywhere junto com o arquivo de configuração da API. A extensão do App Inventor permite que desenvolvedores integrem suas aplicações ao web service para realizar detecção de objetos. O usuário do aplicativo pode tirar fotos com a câmera traseira ou selecionar imagens da galeria para a detecção. Após o processamento, a imagem com os objetos identificados é retornada ao usuário. A detecção está disponível apenas para imagens, não suportando vídeos. O aplicativo pode ser exportado como um APK ou utilizado via Companion. A arquitetura geral da solução é mostrada na Figura 12.

Figura 12 - Contexto geral da integração entre modelo, API, extensão e aplicativo

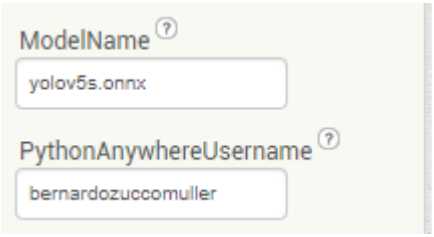
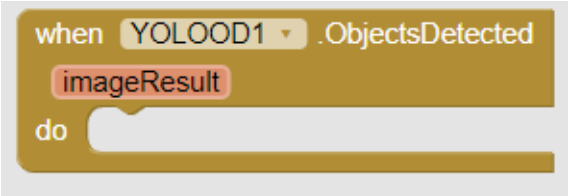


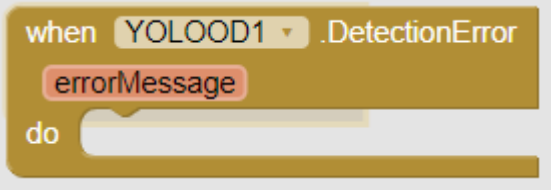
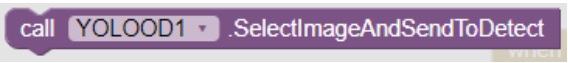
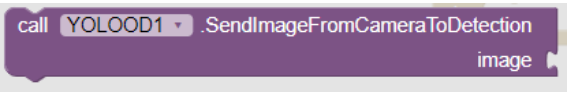
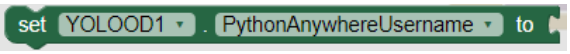
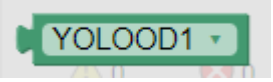
Fonte: Autor, 2024

4.3 Extensão

A extensão YOLOOD possui duas propriedades na aba *designer,* e oito blocos na aba dos blocos. A Tabela 9 mostra como cada bloco e propriedade podem ser vistos no App Inventor.

Tabela 9 - Blocos da extensão

Imagem do componente	**Funcionalidade**
	O *ModelName* é a propriedade que contém o nome do modelo treinado pelo desenvolvedor do aplicativo. *PythonAnywhereUsername* é a propriedade que informa para a extensão qual nome o usuário cadastrou na plataforma do PythonAnywhere, permitindo que a extensão utilize a URL do desenvolvedor.
	O evento *ObjectsDetected* é disparado quando a extensão recebe o arquivo da imagem com a detecção de objetos realizada como resposta da API. O imageResult pode ser passado como parâmetro para o componente de visualização de imagens nativo do App Inventor, que ele irá exibir a imagem, uma vez que o componente busca a imagem a ser exibida pelo caminho do diretório em que ela está salva.
	O evento *ImageSelected* é disparado quando o usuário seleciona uma imagem utilizando a funcionalidade da extensão. Permitindo que o desenvolvedor apresente na tela do aplicativo a imagem escolhida.

	O evento *DetectionError* é disparado quando ocorre algum erro durante o processamento da imagem.
	O bloco *SelectImageAndSentToDetect* abre a galeria de fotos do dispositivo para o usuário selecionar a foto que quer enviar para a API realizar a detecção de objetos. Após a imagem ser selecionada o método envia para a API.
	O bloco *SendImageFromCameraToDetection* permite que o usuário envie para a API uma imagem tirada através do componente de câmera. A imagem deve ser passada como parâmetro da função .
	O bloco *set ModelName* permite ajustar o nome do modelo treinado utilizado na API
	O bloco *set PythonAnywhereUsername* permite ajustar o nome da conta que é utilizada para hospedar a API.
	O bloco retorna uma instância específica da extensão.

A extensão YOLOOD foi desenvolvida utilizando o código fonte do App Inventor, permitindo que a extensão fosse criada e compilada. Também foi construída utilizando a API do App Inventor e várias bibliotecas do Android para manipulação de imagens e requisições HTTP. O objetivo central da extensão é integrar a funcionalidade de detecção de objetos baseada no modelo YOLO utilizando serviços hospedados no PythonAnywhere ao aplicativo do App Inventor, enviando e recebendo uma imagem pela API disponível, código está disponível no repositório do projeto (Disponível em link).

Duas propriedades são definidas para que o desenvolvedor consiga inserir suas variáveis pela interface do App Inventor. Essas propriedades permitem ao desenvolvedor especificar o nome de usuário criado no PythonAnywhere e o modelo de detecção de objetos a ser utilizado.

A classe implementa métodos e eventos que interagem com outros blocos do App Inventor, a seguinte tabela apresenta cada um deles.

Tabela 10 - Métodos da extensão

Nome do método	**Parâmetros**	**Funcionalidade**	**Bloco correspondente**	**Tipo do método**
SelectImageAndSendToDetect	-	Permite selecionar uma imagem da galeria do dispositivo. Ele cria uma intenção para abrir a galeria, espera pela seleção da imagem e regista a intenção para o *resultReturned* capturar o resultado da seleção.	*SelectImageAndSendToDetect*.	*SimpleFunction*
resultReturned	Código da atividade, código do resultado e conteúdo.	Sobrescreve a implementação para tratar o resultado de uma atividade, nesse caso pegando a imagem selecionada para detecção.	-	-
ObjectsDetected	A Uri em String da imagem retornada da API.	Dispara o evento quando a resposta da API é recebida contendo a imagem com as bounding boxes.	*ObjectsDetected*	*SimpleEvent*
DetectionError	Mensagem de erro	Dispara a mensagem quando algum erro acontece durante o processamento da imagem para ser enviada ou da resposta.	DetectionError	SimpleEvent
ImageSelected	A Uri em String da imagem retornada da API	Dispara o evento quando o usuario seleciona uma imagem da galeria para ser enviada para a api processar a detecção.	ImageSelected	SimpleEvent
SaveImage	Bitmap da imagem retornada pela API	Salva na galera do dispositivo a imagem recebida da API.	-	-

A função *SendImageFromCameraToDetection(final String Image)* é a função central da extensão. Este método utiliza a propriedade *Username* para criar a URL de destino da requisição e realiza a operação assíncrona de envio da imagem, nome

do modelo hospedado e nome do usuário cadastrado em um *multipart/form-data* para o servidor PythonAnywhere, onde a detecção de objetos é realizada. A imagem é convertida para um *array* de *bytes*, e uma requisição HTTP POST é feita para o servidor. Caso a resposta da requisição seja recebida com o código de sucesso, a imagem é salva na galeria e disparada em um evento para que o desenvolvedor possa utilizá-la na sua aplicação.

4.4 *Web service*

O *web service* está hospedado no PythonAnywhere e é disponibilizado em uma API construída com o Flask.

O PythonAnywhere é uma plataforma baseada em nuvem que simplifica o processo de programação em Python e a implantação de aplicativos, eliminando a necessidade de configurações complexas ou instalações locais. Esta plataforma oferece um ambiente de desenvolvimento acessível através de navegadores web, onde os usuários podem escrever, editar e executar código Python sem a necessidade de instalar o Python em suas máquinas locais. Além disso, a plataforma permite que os desenvolvedores hospedem suas aplicações de forma gratuita (PythonAnywhere, 2024).

A plataforma foi escolhida, uma vez que ela cumpre com todos os requisitos necessários para o contexto da aplicação. Os alunos treinam os seus próprios modelos utilizando pequenos conjuntos de dados, resultando em modelos que respeitam o limite de 100mb de armazenamento gratuito da plataforma. Também, as aplicações hospedadas tem como objetivo somente a iniciação da educação dos jovens sobre detecção de objetos. Com isso, não se busca desenvolver aplicações que necessitem de grande escalabilidade. Então, o PythonAnywhere serve como uma plataforma acessível e eficaz para o projeto.

O Flask, um framework de aplicativos web em Python, desenvolvido com o objetivo de facilitar a criação de aplicativos *web*. O Flask disponibiliza as ferramentas essenciais para o desenvolvimento de aplicativos web, incluindo roteamento, geração de templates e gestão de solicitações HTTP (FLASK, 2024).

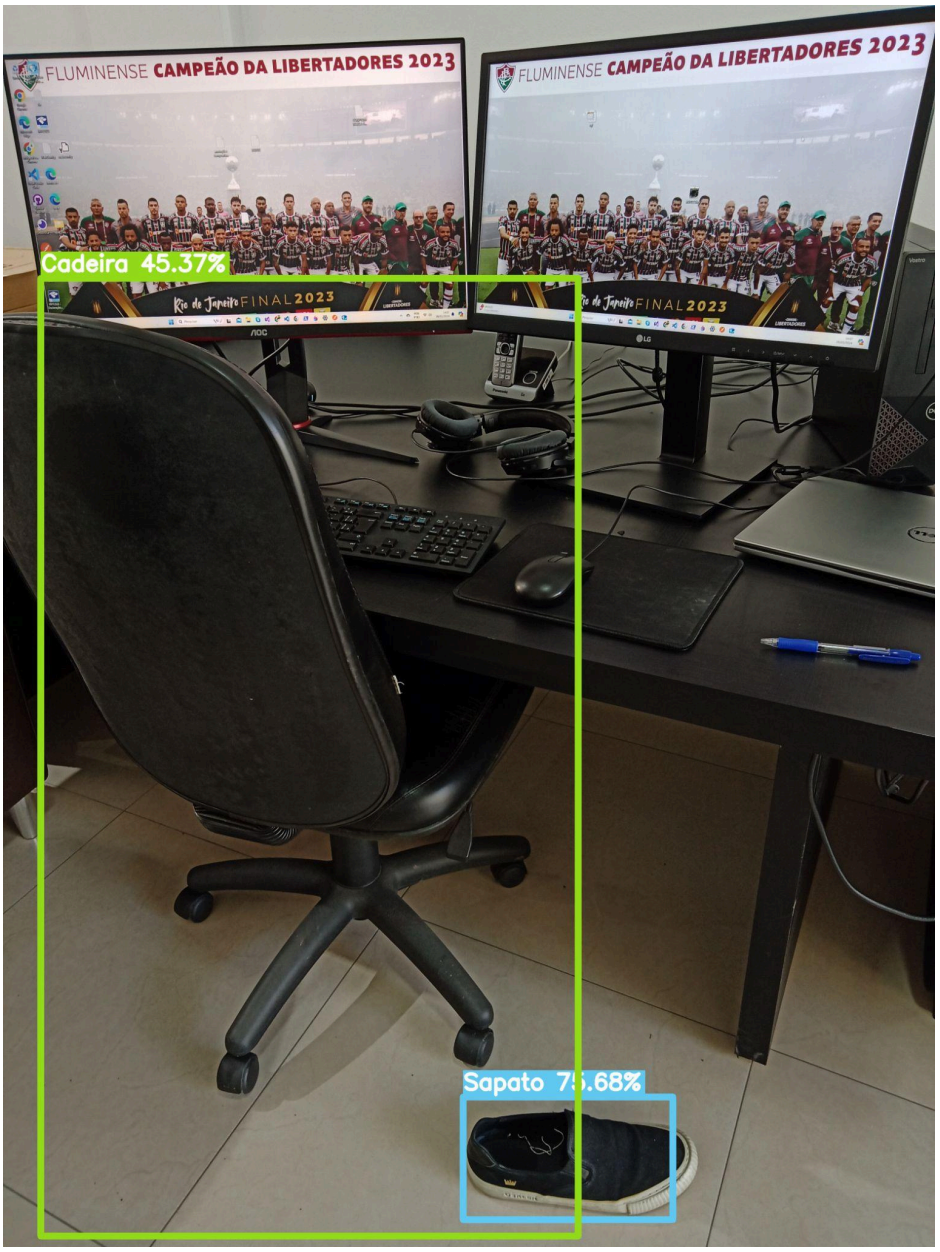
A API define um *endpoint* '/*detect*', acessado por meio de requisições *POST*, em que é recebida a imagem para ser processada. A implementação pode ser vista no repositório do projeto (Disponível em link).

A aplicação recebe a imagem, juntamente com informações adicionais, como o nome do modelo utilizado e o nome de usuário. A imagem passa por um pré-processamento para ser normalizada de forma que se adeque ao tipo de dado esperado pelo modelo de detecção.

O método *detect_objects*, tem papel central na inferência da imagem. Inicialmente, uma sessão de inferência *ONNX* é criada com o modelo especificado, permitindo a execução de inferências. A imagem de entrada é então preparada e passada para o modelo, resultando em saídas que indicam as localizações e classes dos objetos detectados. Após a inferência, as classes detectadas são obtidas a partir dos metadados do modelo e convertidas em um dicionário para serem utilizadas ao desenhar as bounding boxes posteriormente. Além disso, após a detecção inicial, as coordenadas das *bounding boxes* são convertidas de xyxy (canto superior esquerdo e canto inferior direito) para xywh (centro e largura/altura). Em seguida, é aplicado o método *non_max_suppression* para remover detecções redundantes. Esse método avalia a confiança das detecções com base no threshold 0,3 e calcula a sobreposição entre elas usando o índice de IoU 0,5, utilizando os valores da documentação do *ONNX Runtime* como referência . Assim, apenas as detecções mais confiantes e não redundantes são mantidas para serem utilizadas. O método retorna as detecções pós-processadas juntamente com o dicionário de classes, permitindo que as bounding boxes possam ser desenhadas na imagem de entrada.

A Figura 13 apresenta uma imagem retornada pela API após realizar a detecção de objetos.

Figura 13 - Imagem retornada pela API



Fonte: Autor, 2024

5. APLICATIVO DETECTA UTILIZANDO A EXTENSÃO YOLO OD

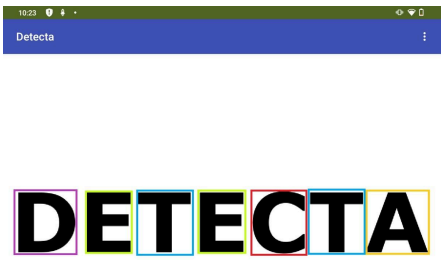
Foi desenvolvido um aplicativo chamado de DETECTA, para apresentar o funcionamento e realizar testes utilizando a extensão YOLOOD. O aplicativo utiliza um modelo desenvolvido com base no curso “Crie o seu primeiro modelo para detecção de objetos” (Martins,2022), o modelo YOLOv5 foi treinado utilizando o ODIN (Disponível em ODIN) (Santana, 2022) e exportado no formato ONNX (Disponível em modelo).

O objetivo do aplicativo é permitir que o usuário selecione fotos da galeria ou tire uma foto utilizando a câmera do dispositivo e receba a foto com a detecção de objetos realizada.

Para fins de teste foi criado uma conta no PythonAnywhere e feito o *upload* do modelo yolov5s.onnx junto ao código do serviço e definição da API. O aplicativo foi desenvolvido utilizando o App Inventor.

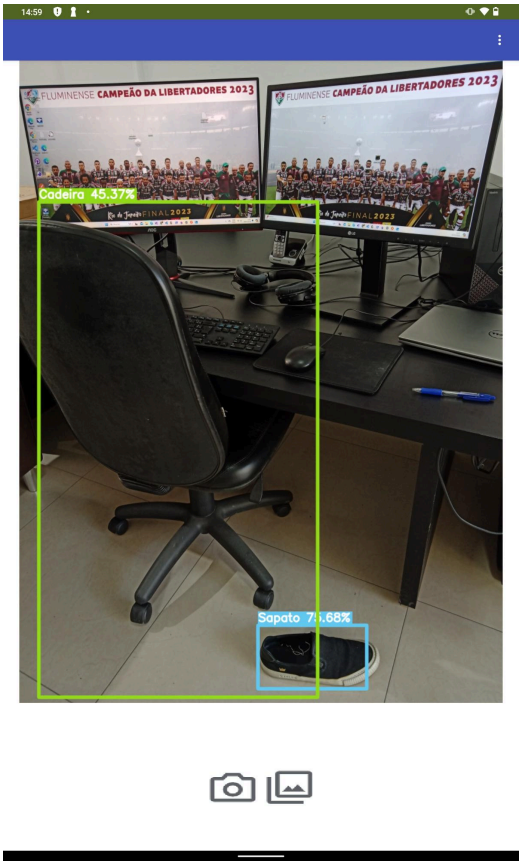
Nas figuras 14 e 15, são apresentadas as telas do aplicativo. Sendo a tela inicial do aplicativo e a tela que possibilita o usuário de selecionar a imagem da galera do dispositivo ou capturar uma foto com a câmera, respectivamente. Nesse caso é exibido o resultado de uma detecção após o usuário tirar uma foto com a câmera do celular.

Figura 14 - Tela inicial do aplicativo DETECTA



Fonte: Autor, 2024

Figura 15 - Tela inicial do aplicativo DETECTA com resultado da detecção



Fonte: Autor, 2024

Durante os testes realizados foi possível avaliar o cumprimento dos requisitos funcionais da extensão e do *web service*. Conclui-se que todos foram cumpridos.

Tabela 11 - Resultados dos testes da extensão

ID	**Requisito**	**Descrição**	**Teste**
RF01	Enviar imagens para a API do PythonAnywhere.	Enviar imagens para a API do PythonAnywhere utilizando um método POST e receber a imagem com os objetos detectados como resposta.	A imagem foi corretamente enviada para a API e foi possível obter imagem com as detecções como resposta.
RF02	Abrir a câmera traseira/frontal do celular.	A extensão deve utilizar a câmera do celular do usuário para captura de imagem.	Durante os testes, foi possível capturar uma imagem pela câmera e enviar para a API.
RF03	Solicitar a permissão de utilização da câmera do celular.	A extensão deve solicitar ao usuário a permissão de utilização da câmera do celular antes de abri-la.	Durante os testes, na primeira execução do AI Companion que utilizou a câmera, foi solicitada a permissão para usá-la.
RF04	Salvar a imagem tirada com a	A imagem tirada pela câmera que foi usada para a detecção	Durante os testes, a imagem recebida como resposta foi salva

	câmera e então enviar para a detecção	deve ser salva no celular .	no dispositivo.
RF05	Apresentar a imagem selecionada ou capturada.	A extensão deve permitir que a imagem capturada ou selecionada seja apresentada na tela enquanto a detecção está sendo processada.	Durante os testes, a imagem pôde ser mostrada na tela para o usuário.
RF07	Visualizar as *bounding boxes* de objetos detectados na imagem salva de acordo com os resultados da detecção.	A extensão deve permitir a visualização das *bounding boxes* na imagem que foi salva e apresentar os atributos relativos àquele objeto.	Durante os testes, a extensão permitiu que o desenvolvedor apresentasse na tela a imagem recebida.
RF08	Uso de um modelo treinado e exportado em formato ONNX pelo desenvolvedor do aplicativo.	O serviço deve permitir ao usuário carregar o modelo treinado dentro da sua aplicação no *PythonAnywhere*.	Durante os testes,o serviço permitiu o carregamento do modelo.
RF09	Manipulação de resultados.	A extensão deve permitir manipular os resultados da detecção via blocos possibilitando uma integração dos resultados na funcionalidade de um app.	Durante os testes, foi possível interagir com os resultados da detecção.
RF10	Acesso a imagens armazenadas no celular.	A extensão deve permitir utilizar imagens armazenadas no celular para realizar a detecção de objetos.	Durante os testes, foi possível selecionar uma imagem previamente existente na galeria do dispositivo.
RF11	Câmera continuar funcionando ao sair e voltar para a tela de detecção.	Permitir que o usuário troque de tela e ao voltar para a tela da detecção ainda seja possível capturar fotos com a câmera.	Durante os testes, ao sair da tela2 e ir para tela home e voltar para tela2 a câmera continuou funcionando.

6. MATERIAL DIDÁTICO

Buscando dar continuidade ao curso "Crie o seu primeiro modelo para detecção de objetos", focalizando na fase de integração do modelo concebido em um aplicativo, foi elaborado um tutorial para auxiliar professores e alunos na utilização da extensão YOLOOD. O material didático está disponível em link para o tutorial contendo os arquivos apresentados na Tabela 12. Visa-se que ao realizar os

passos descritos no tutorial o aluno tenha aprendido a utilizar a extensão YOLOOD e tenha a capacidade de desenvolver seu próprio aplicativo com seu modelo treinado para detectar objetos. O tutorial de implantação do modelo de ML utilizando o YOLOOD é uma continuação do curso "Crie o seu primeiro modelo para detecção de objetos "(Martins, 2022).

Tabela 12 - Conteúdo do material didático

Material	**Tipo**	**Descrição**	**Link**
Tutorial de implantação do modelo de Machine Learning com YOLOOD	Slide	Slides mostrando o passo a passo de como utilizar a extensão e carregar os arquivos no web service.	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/TutorialYOLOOD.pptx
Wireframe do app Detecta.	Arquivo .aia.	Projeto do aplicativo detecta com telas pré-desenvolvidas, para servir como ponto inicial do desenvolvimento.	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/DetectaWireframe.aia
Projeto do aplicativo Detecta, versão final com os blocos.	Arquivo .aia.	Projeto pronto do aplicativo detecta.	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/Detecta.aia
Extensão YOLOOD.	Arquivo.aix.	Extensão YOLOOD desenvolvida neste trabalho	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/YOLOOD.aix
Modelo para detecção de 10 objetos em casa treinado com YOLOv5.	Arquivo .onnx.	Modelo treinando seguindo o curso ML Para Todos	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/yolov5s.onnx
Código python do web service.	Arquivo Python.	Código para definir a API e processar a detecção de objetos.	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/flask_app.py

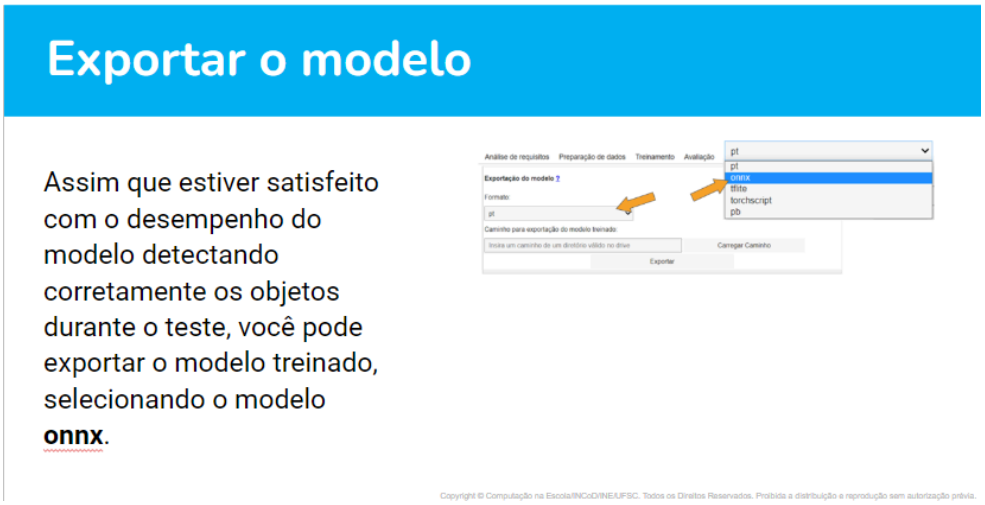
Os slides desenvolvidos buscam descrever o processo de integração do aplicativo à extensão de forma detalhada. O passo a passo mostra ao aluno como exportar o próprio modelo e utilizá-lo na plataforma do App Inventor com a extensão YOLOOD para desenvolver seu próprio aplicativo.

Figura 16 - Capa dos slides do material didático

Fonte: Autor, 2024

Inicialmente mostra-se como exportar o modelo YOLOv5 treinado utilizando o Jupyter Notebook ODIN no formato ONNX.

Figura 17 - Seção dos slides sobre a exportação do modelo



Fonte: Autor, 2024

Na segunda seção da aula é apresentado o PythonAnywhere para o aluno, ensinando-o como criar uma conta e um aplicativo para carregar o seu modelo e disponibilizá-lo em uma API.

Figura 18 - Seção dos slides sobre o PythonAnywhere

Fonte: Autor, 2024

Na sequência é mostrado como utilizar a extensão YOLOOD no App Inventor, é disponibilizado um *wireframe* para que os alunos tenham um ponto inicial para desenvolver o aplicativo. É ensinado como realizar testes no App desenvolvido antes de exportá-lo para que possa ser usado em dispositivos móveis.

Figura 19 - Seção dos slides sobre a extensão YOLOOD

Fonte: Autor, 2024

7. CONCLUSÃO

O objetivo deste trabalho foi implementar a extensão YOLOOD para permitir que alunos da educação básica coloquem em prática conceitos sobre *Machine Learning* aprendidos nos cursos da iniciativa Computação na escola. Primeiramente foi desenvolvida a fundamentação teórica (01). Uma análise do estado da arte foi realizada para encontrar extensões para o App Inventor que permitissem utilizar modelos de *Machine Learning* para detecção de objetos (02). Baseando-se na fundamentação teórica e no estado da arte, foi criada a extensão YOLOOD para implementar a detecção de objetos com modelos YOLOv5 em aplicativos desenvolvidos com o App Inventor, adicionalmente foi criado um *web service* hospedado no PythonAnywhere para processar as imagens e realizar a inferência sobre o modelo(03). Para que os alunos consigam utilizar a extensão, foi elaborado um material didático com slides de um tutorial detalhados sobre como utilizar a extensão desenvolvida e implementar o API para integração com o web service (04).

Deste modo, pretende-se que o trabalho ajude a tornar a aprendizagem de *Machine Learning* mais acessível e atrativa, permitindo que alunos da educação básica desenvolvam aplicativos com essas funcionalidades. Com isso, podemos despertar nos jovens o entusiasmo em aprofundar seus conhecimentos em sistemas de informação e possivelmente desenvolver soluções para a sociedade.

O modelo conceitual da solução foi alterado durante o desenvolvimento da extensão, esperava-se que fosse implementado uma extensão que não necessitasse de internet para realizar a detecção de objetos. Mas foram encontrados problemas técnicos que resultaram na alteração para que a inferência fosse realizada em uma web service.

Para trabalhos futuros, é recomendado que seja revisado o cálculo da posição das classes de cada bounding box na imagem melhorando a acessibilidade da extensão. É interessante que novas alternativas sejam estudadas para realizar o carregamento do modelo e inferência sejam realizados dentro do dispositivo, descartando a necessidade de acesso a internet.

REFERÊNCIAS

APP INVENTOR. App Inventor Extensions. 2021 Disponivel em <http://ai2.appinventor.mit.edu/reference/other/extensions.html>

AYODELE, T. O.. Types of machine learning algorithms. New advances in machine learning, v. 3, p. 19-48, InTech, 2010.

AZEVEDO, P. Object Detection State of the Art 2022. Disponível em:<https://medium.com/@pedroazevedo6/object-detection-state-of-the-art-2022-ad750e0f6003>. Acesso em : 12 julho 2022.

AMERSHI, S. et al. Software engineering for machine learning: A case study. In: Proc. of the IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in

Practice, IEEE, 2019. p. 291-300.

BOCHKOVSKIY, A., WANG, C., LIAO, H. M. YOLOv4: Optimal Speed and Accuracy of Object Detection. arxiv 2004.10934, 2020.

COCHARD, D,. YOLOv5 : The Latest Model for Object Detection 2021. Disponível em: <https://medium.com/axinc-ai/yolov5-the-latest-model-for-object-detection-b13320ec516b#:~:text=The%20output%20from%20YOLOv5,outputs%20the%20following%203%20tensors.&text=The%20breakdown%20of%20the%20output,conf%2C%20pred_cls(80)%5D./> Acesso em: 30 de junho 2022.

DEY, A.. Machine learning algorithms: a review. International Journal of Computer Science and Information Technologies, v. 7, n. 3, p. 1174-1179, 2016.

FLASK, 2024. Disponível em <https://flask.palletsprojects.com/en/3.0.x/>. Acesso em: 1 de maio de 2024

GÉRON, A.. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Inc. 2019.

GOODFELLOW, I., BENGIO, Y., & COURVILLE, A. Deep learning. MIT press. 2016

GOOGLE. Teachable Machine. Disponível em: <https://teachablemachine.withgoogle.com/>.

Acesso em: 20 out. 2022

GOOGLE. Glossário de machine learning. Disponível em: <https://developers.google.com/machine-learning/glossary?hl=pt-br#neural_network/>. 2023 .Acesso em: 7 maio. 2023

JAIN, A., SINGH, A., & KUMAR, A. Object detection: A survey. Journal of Computer Science and Technology, 31(3), 399-417. 2016.

LECUN, Y., BENGIO, Y. & HINTON, G. Deep learning. Nature 521, 436–444 (2015).

LIU, W., ANGUELOV,D., ERHAN,D., SZEGEDY,C., REED,S., FU Y. , BERG, A. SSD: single shot MultiBox detector, Springer International Publishing , pp. 21-37, 2016.

LÓPEZ, F. ONNX: Preventing Framework Lock in. 2020. Disponível em <https://towardsdatascience.com/onnx-preventing-framework-lock-in-9a798fb34c92>

MARQUES, L. S., GRESSE VON WANGENHEIM, C., HAUCK, J. C. R. Ensino de Machine Learning na Educação Básica: um Mapeamento Sistemático do Estado da Arte. In: Anais do Simpósio Brasileiro de Informática na Educação, Natal, Brasil, 2020.

MARTINS. L. Desenvolvimento de um Curso de Machine Learning com Foco em Detecção de Objetos no Ensino Médio. 2022. Trabalho de Conclusão de Curso. (Graduação em Sistemas de Informação) – Universidade Federal de Santa Catarina.

MITCHELL, T. M. Machine Learning. McGraw-Hill Science/Engineering/Math; 1997.

MIT. About Us. Disponível em: <https://appinventor.mit.edu/about-us>. Acesso em: 10 mar. 2022.

MIT. About Us. Disponível em: <https://www.media.mit.edu/>. Acesso em: 8 maio. 2023.
MIT. About Us. Disponível em: <https://community.appinventor.mit.edu//>. Acesso em: 8 maio. 2023.

OLIVEIRA, F. P.. TMIC – Uma Extensão do App Inventor para a Implantação de Modelos de ML voltados a Classificação de Imagens Treinados no Teachable Machine. 2022. Trabalho de Conclusão de Curso. (Graduação em Sistemas de Informação) – Universidade Federal de Santa Catarina.

TANG, D. Empowering Novices to Understand and Use Machine Learning With Personalized Image Classification Models, Intuitive Analysis Tools, and MIT App Inventor. M.Eng thesis, MIT, Cambridge, USA. 2019.

ONNX, 2023 Disponível em <https://onnx.ai/index.html> Acesso em: 22 abril 2022.

O'HARA, K., BLANK, D., MARSHALL, J. Computational Notebooks for AI Education. In: Proc. of the 28th International Florida Artificial Intelligence Research Society Conference, Hollywood, FL, USA, 2015.

PATTON, W., TISSENBAUM, M., HARUNANI, F. MIT app inventor: Objectives, design, and development. Computational thinking education, p. 31-49, 2019.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: Proc. of the 12th International Conference on Evaluation and Assessment in Software Engineering, Bari, Italy, 68–77, 2008.

PYTORCH. 2023 Disponível em <https://pytorch.org/docs/stable/jit.html> Acesso em: 29 de junho de 2023.

PRESSMAN, R. S. Engenharia de Software: Uma Abordagem Profissional. 9ª Edição, AMGH Editora. 2021.

PURAVIDA. 2023 Disponível em <https://puravidaapps.com/> Acesso em: 15 de maio de 2023.

REDMON, J., FARHADI, A. YOLO9000: Better, faster, stronger. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, USA 21-26, 7263-7271. 2016

REDMON, J., DIVVALA, S., GIRSHICK, R., & FARHADI, A. You only look once: Unified, real-time object detection. arxiv 1506.02640. 2016.

GIRSHICK, R., DONAHUE, J., DARRELL, T., MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation . arxiv 1311.2524, 2014.

PYTHONANYWHERE, 2024. Disponivel em <https://www.pythonanywhere.com/> Acesso em 01 de maio de 2024

REDMON, J., FARHADI, A. Yolov3: An incremental improvement. arXiv arXiv:1804.02767, 2018.

SANTANA, J. P. Uma Ferramenta Visual para o Desenvolvimento de Modelos de Detecção de Objetos com Deep Learning no Ensino Superior. 2022. Trabalho de Conclusão de Curso. (Graduação em Ciência da Computação) – Universidade Federal de Santa Catarina.

SHAH, B., Complete guide to Object Detection using Deep Learning. 2020 Disponivel em <https://17bce011.medium.com/complete-guide-to-object-detection-using-deep-learning-23ffc99ab072>

TANG, D. Empowering Novices to Understand and Use Machine Learning With Personalized Image Classification Models, Intuitive Analysis Tools, and MIT App Inventor. M.Eng thesis, MIT, Cambridge, USA. 2019.

TAVARES, L. A. et al. Inteligência Artificial na Educação: Survey". Brazilian Journal of Development, 6(7), 2020.

TENSORFLOW. 2021. Disponível em: <https://www.tensorflow.org/?hl=pt-br>. Acesso em: 8 maio. 2023.

TOURETZKY, D. S., GARDNER-MCCUNE, C., MARTIN, F., SEEHORN, D. K-12 Guidelines for Artificial Intelligence: What Students Should Know. In: Proc. of the ISTE Conference, Philadelphia, PA, USA, 2019.

ULTRALYTICS, 2021. Disponível em: <https://docs.ultralytics.com/> Acesso em: 10 julho 2022.

ULTRALITYCS, 2023. Disponível em: <https://docs.ultralytics.com/> Acesso em: 02 julho 2022.

WOLBER, D., ABELSON H., FRIEDMAN, M. Democratizing Computing with App Inventor. GetMobile: Mobile Computing and Communications. vol. 18, 2014, 53–58.

ZAIDI, S., ANSARI, M., ASLAM, A., KAWALI. N., ASGHAR. M., LEE, B. A survey of modern deep learning based object detection models. arxiv 2104.11892v2, 2020.

ZHAO, Z., ZHENG, P., XU, S., WU., X.Object Detection with Deep Learning: A Review. arxiv 1807.05511, 2018.

ZOU, Z. et al. Object Detection in 20 Years: A Survey. Proceedings of the IEEE, v. 111, n.3, p. 257-276, 2023.

Desenvolvimento de um Web Service e Extensão do App Inventor para Detecção de Objetos

Bernardo Zucco Müller[1], Christiane Gresse Von Wangenheim[1]

[1] Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis / SC, Brasil

bernardo.zucco.muller@grad.ufsc.br, c.wangenheim@ufsc.br

Abstract. *This paper presents the development of the YOLOOD extension for the App Inventor to integrate object detection capabilities using YOLOv5 models. The extension aims to provide basic education students with practical experience in Machine Learning by enabling the creation of applications that perform object detection. The extension, coupled with a web service hosted on PythonAnywhere, processes images and performs inference using trained models. The project includes didactic materials to facilitate the use of the extension. The results indicate the potential of the extension to enhance learning experiences and inspire further exploration in information systems.*

Resumo. *Este artigo apresenta o desenvolvimento da extensão YOLOOD para o App Inventor, integrando a funcionalidade de detecção de objetos utilizando modelos YOLOv5. A extensão visa fornecer aos alunos da educação básica experiência prática em Machine Learning, permitindo a criação de aplicativos que realizam detecção de objetos. A extensão, juntamente com um serviço web hospedado no PythonAnywhere, processa imagens e realiza inferências usando modelos treinados. O projeto inclui materiais didáticos para facilitar o uso da extensão. Os resultados indicam o potencial da extensão para melhorar as experiências de aprendizagem e inspirar uma exploração mais aprofundada em sistemas de informação.*

1. Introdução

Atualmente a Inteligência Artificial (IA) demonstra um papel econômico-social muito grande no mundo todo por estar presente em grande parte das atividades do nosso dia a dia, desde a produção de manufaturas até atividades médicas (Tavares, 2020). Uma área da IA é o Machine Learning (ML), que é a capacidade das máquinas aprenderem com base em conjuntos de dados, de forma similar a capacidade de aprendizagem humana (Mitchell, 1997). Um subconjunto de ML é o Deep Learning (DL), que utiliza modelos computacionais compostos por múltiplas camadas de processamento para aprender a representar dados com múltiplas camadas de abstração, chamadas de redes neurais profundas (Lecun et al., 2015). Uma das principais tarefas de ML é a detecção de objetos (Zou et al., 2019).

Detecção de objetos é uma tarefa computacional responsável por identificar visualmente instâncias de objetos de classes pré-definidas, em fotos ou vídeos. O objetivo dos estudos na detecção de objetos é desenvolver modelos capazes de assimilar onde está cada objeto na imagem (Jain et al., 2016). Existem diversos modelos para detecção de objetos, entre eles o Region-based Convolutional Neural Network (R-CNN) (Girshick, 2014), SSD (Liu, 2016), Fast R-CNN (Girshick, 2015) e o YOLO (Ultralytics, 2021). O modelo YOLO, por ser um modelo que trata a classificação e a identificação das bounding boxes na mesma etapa, ele se torna extremamente rápido.

Assim, a detecção de objetos utilizando o YOLO é 1000x mais rápida que a detecção utilizando outras arquitetura (Ultralytics, 2021).

A quinta versão do YOLO, o YOLO 5, trouxe ainda mais aprimoramentos do que suas versões anteriores, principalmente por adotar uma arquitetura mais simples, tornando-o mais rápido. O YOLOv5 apresenta algumas versões com diferentes tamanhos. Consequentemente, a versão com a maior rede neural apresenta uma melhor performance. Porém, a sua maior quantidade de parâmetros torna o treinamento mais demorado e necessitando de mais memória de processamento.

O processo de desenvolvimento de ML difere do processo comum de desenvolvimento de software e tem suas peculiaridades. São fases do processo de desenvolvimento (Amershi et al., 2019): análise de requisitos, preparação dos dados, rotulação dos dados, treinamento do modelo, avaliação do desempenho, predição e exportação.

A exportação é a etapa na qual o modelo é salvo em um formato para que possa ser usado em diversas plataformas e linguagens de programação. Open Neural Network Exchange (ONNX) (ONNX, 2023) é um formato que consiste em um modelo serializado em um arquivo protobuf, descrevendo a estrutura do modelo treinado e os parâmetros necessários para executá-lo.

Porém, entender sobre IA ainda é algo muito distante para grande parte da sociedade. Currículos escolares raramente englobam conteúdos sobre como desenvolver sistemas inteligentes (Marques et al., 2020). No curso “Crie o seu primeiro modelo para detecção de objetos” desenvolvido pela iniciativa Computação na Escola/INCoD/INE/UFSC (Martins, 2022) é ensinado desenvolver um modelo YOLOv5 (Ultralytics, 2021) de detecção de objetos de casa (mesa, sapato etc.) dentro de um Jupyter Notebook usando ODIN (Santana, 2022), uma camada visual que permite o desenvolvimento de ML nos Jupyter Notebook por meio da interface visual.

Entretanto, atualmente esses cursos muitas vezes ainda não abordam a implantação desses modelos de detecção de objetos em sistemas web ou aplicativos móveis (Marques et al., 2020) pela falta de infraestrutura que permita uma implantação simplificada.

Uma alternativa de implantação pode ser em aplicativos móveis desenvolvidos com App Inventor. O MIT App Inventor é um ambiente de programação com uma linguagem de blocos visuais que possibilita às pessoas que desejam iniciar na programação de aplicativos, criar aplicações de uma forma interativa e facilitada (Wolber, 2020). O App Inventor disponibiliza uma plataforma que permite que extensões sejam criadas para adicionar funcionalidades ao App Inventor.

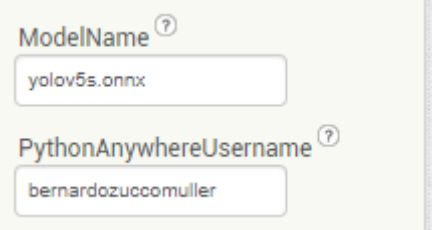
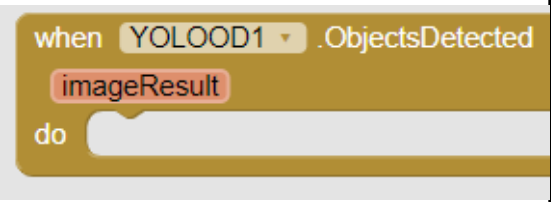
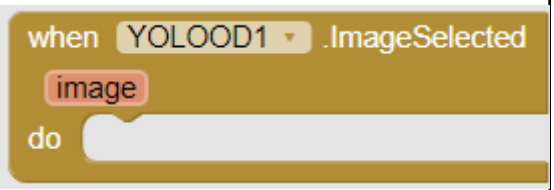
Dessa forma, o presente trabalho busca desenvolver uma extensão que integre um aplicativos criados no App Inventor a um web service hospedado no PythonAnywhere por meio de uma API Flask. Essa integração permitirá o uso de modelos YOLOv5 treinados para detecção de objetos com e exportados no formato ONNX.

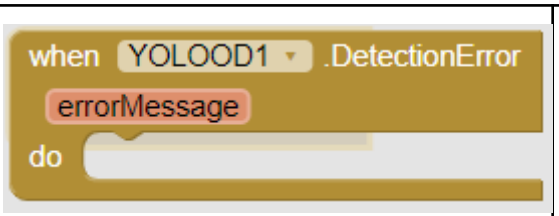
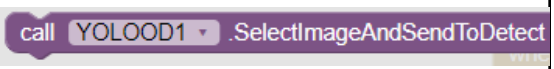
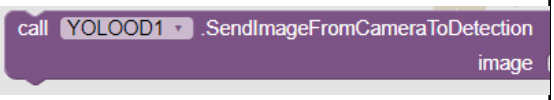
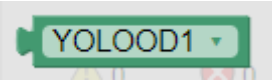
2. Extensão YOLOOD

A extensão YOLOOD foi desenvolvida utilizando o código fonte do App Inventor, permitindo que a extensão fosse criada e compilada. Também foi construída utilizando a API do App Inventor e várias bibliotecas do Android para manipulação de imagens e requisições HTTP. O objetivo central da extensão é integrar a funcionalidade de detecção de objetos baseada no modelo YOLO utilizando serviços hospedados no PythonAnywhere ao aplicativo do App Inventor, enviando e recebendo uma imagem pela API disponível.

Duas propriedades são definidas para que o desenvolvedor consiga inserir suas variáveis pela interface do App Inventor. Essas propriedades permitem ao desenvolvedor especificar o nome de usuário criado no PythonAnywhere e o modelo de detecção de objetos a ser utilizado, o código completo está disponível em repositório.

Tabela 1. Blocos da extensão

Imagem do componente	Funcionalidade
	O ModelName é a propriedade que contém o nome do modelo treinado pelo desenvolvedor do aplicativo. PythonAnywhereUsername é a propriedade que informa para a extensão qual nome o usuário cadastrou na plataforma do PythonAnywhere, permitindo que a extensão utilize a URL do desenvolvedor.
	O evento ObjectsDetected é disparado quando a extensão recebe o arquivo da imagem com a detecção de objetos realizada como resposta da API. O imageResult pode ser passado como parâmetro para o componente de visualização de imagens nativo do App Inventor, que ele irá exibir a imagem, uma vez que o componente busca a imagem a ser exibida pelo caminho do diretório em que ela está salva.
	O evento ImageSelected é disparado quando o usuário seleciona uma imagem utilizando a funcionalidade da extensão. Permitindo que o desenvolvedor apresente na tela do aplicativo a imagem escolhida.

	O evento DetectionError é disparado quando ocorre algum erro durante o processamento da imagem.
	O bloco SelectImageAndSentToDetect abre a galeria de fotos do dispositivo para o usuário selecionar a foto que quer enviar para a API realizar a detecção de objetos. Após a imagem ser selecionada o método envia para a API.
	O bloco SendImageFromCameraToDetection permite que o usuário envie para a API uma imagem tirada através do componente de câmera. A imagem deve ser passada como parâmetro da função .
	O bloco set ModelName permite ajustar o nome do modelo treinado utilizado na API
	O bloco set PythonAnywhereUsername permite ajustar o nome da conta que é utilizada para hospedar a API.
	O bloco retorna uma instância específica da extensão.

A função SendImageFromCameraToDetection(final String Image) é a função central da extensão. Este método utiliza a propriedade Username para criar a URL de destino da requisição e realiza a operação assíncrona de envio da imagem, nome do modelo hospedado e nome do usuário cadastrado em um multipart/form-data para o servidor PythonAnywhere, onde a detecção de objetos é realizada. A imagem é convertida para um array de bytes, e uma requisição HTTP POST é feita para o servidor. Caso a resposta da requisição seja recebida com o código de sucesso, a imagem é salva na galeria e disparada em um evento para que o desenvolvedor possa utilizá-la na sua aplicação.

3. *Web service*

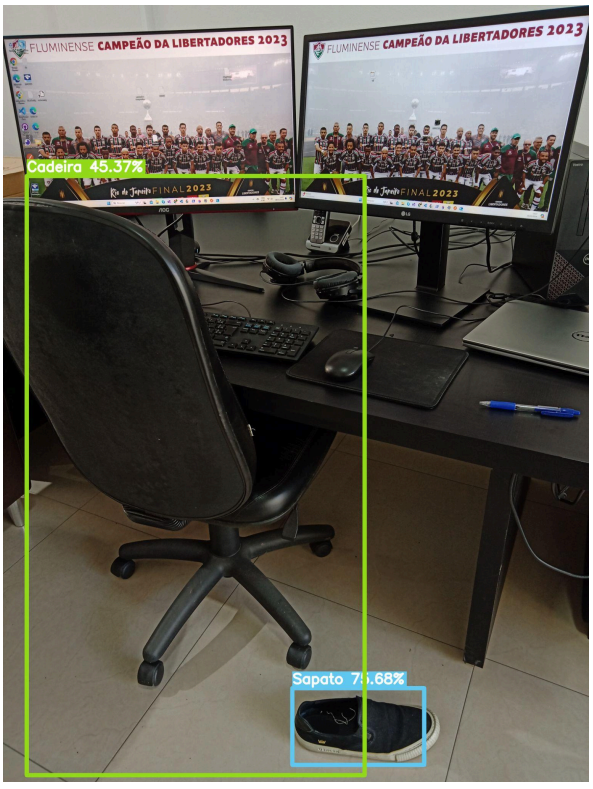
O web service está hospedado no PythonAnywhere e é disponibilizado em uma API construída com o Flask.

O PythonAnywhere é uma plataforma baseada em nuvem que simplifica o processo de programação em Python e a implantação de aplicativos, eliminando a necessidade de configurações complexas ou instalações locais. Esta plataforma oferece um ambiente de desenvolvimento acessível através de navegadores web, onde os usuários podem escrever, editar e executar código Python sem a necessidade de instalar o Python em suas máquinas locais. Além disso, a plataforma permite que os desenvolvedores hospedem suas aplicações de forma gratuita (PythonAnywhere, 2024).

O Flask, um framework de aplicativos web em Python, desenvolvido com o objetivo de facilitar a criação de aplicativos web. O Flask disponibiliza as ferramentas essenciais para o desenvolvimento de aplicativos web, incluindo roteamento, geração de templates e gestão de solicitações HTTP (FLASK, 2024).

A API define um endpoint '/detect', acessado por meio de requisições POST, em que é recebida a imagem para ser processada. A aplicação recebe a imagem, juntamente com informações adicionais, como o nome do modelo utilizado e o nome de usuário. A imagem passa por um pré- processamento para ser normalizada de forma que se adeque ao tipo de dado esperado pelo modelo de detecção. Após a inferência, as classes detectadas são obtidas a partir dos metadados do modelo e convertidas em um dicionário para serem utilizadas ao desenhar as bounding boxes posteriormente. . Em seguida, é aplicado o método non_max_suppression para remover detecções redundantes. Assim, apenas as detecções mais confiantes e não redundantes são mantidas para serem utilizadas.

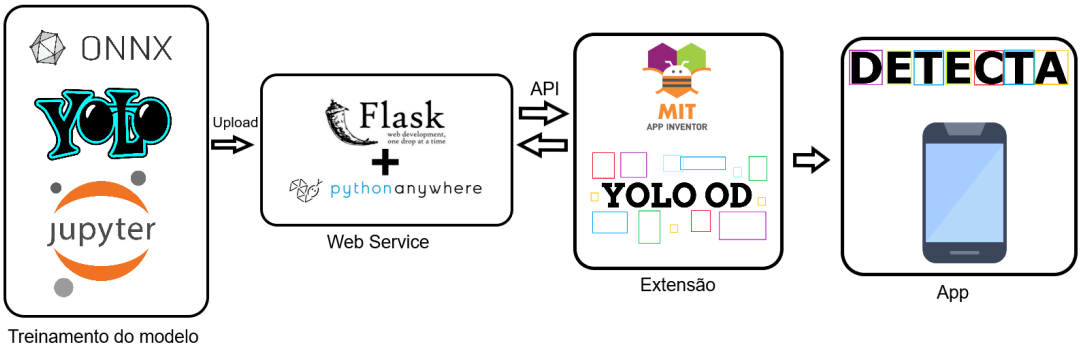
Figure 1. Imagem retornada pela API



3. APLICATIVO DETECTA UTILIZANDO A EXTENSÃO YOLO OD

Foi desenvolvido um aplicativo chamado de DETECTA, para apresentar o funcionamento e realizar testes utilizando a extensão YOLOOD. O aplicativo utiliza um modelo desenvolvido com base no curso “Crie o seu primeiro modelo para detecção de objetos” (Martins,2022), o modelo YOLOv5 foi treinado utilizando o ODIN (Santana, 2022) e exportado no formato ONNX . O objetivo do aplicativo é permitir que o usuário selecione fotos da galeria ou tire uma foto utilizando a câmera do dispositivo e receba a foto com a detecção de objetos realizada. O desenvolvimento do aplicativo é acompanhado da disponibilização de um material didático que contempla todas as etapas de desenvolvimento, para integrar o modelo, API, extensão e aplicativo.

Figura 2. Contexto geral da integração entre modelo, API, extensão e aplicativo



Todos os arquivos necessários para desenvolver o aplicativo estão disponíveis nos links da tabela abaixo.

Tabela 12 - Conteúdo do material didático

Material	Tipo	Descrição	Link
Tutorial de implantação do modelo de Machine Learning com YOLOOD	Slide	Slides mostrando o passo a passo de como utilizar a extensão e carregar os arquivos no web service.	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/TutorialYOLOOD.pptx
Wireframe do app Detecta.	Arquivo .aia.	Projeto do aplicativo detecta com telas pré-desenvolvidas, para servir como ponto inicial do desenvolvimento.	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/DetectaWireframe.aia

Projeto do aplicativo Detecta, versão final com os blocos.	Arquivo .aia.	Projeto pronto do aplicativo detecta.	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/Detecta.aia
Extensão YOLOOD.	Arquivo.aix.	Extensão YOLOOD desenvolvida neste trabalho	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/YOLOOD.aix
Modelo para detecção de 10 objetos em casa treinado com YOLOv5.	Arquivo .onnx.	Modelo treinando seguindo o curso ML Para Todos	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/yolov5s.onnx
Código python do web service.	Arquivo Python.	Código para definir a API e processar a detecção de objetos.	https://codigos.ufsc.br/gqs/extensions-app-inventor-gqs/-/blob/YOLOOD/flask_app.py

4. CONCLUSÃO

O objetivo deste trabalho foi implementar a extensão YOLOOD para permitir que alunos da educação básica coloquem em prática conceitos sobre Machine Learning aprendidos nos cursos da iniciativa Computação na escola. Primeiramente foi desenvolvida a fundamentação teórica (01). Uma análise do estado da arte foi realizada para encontrar extensões para o App Inventor que permitissem utilizar modelos de Machine Learning para detecção de objetos (02). Baseando-se na fundamentação teórica e no estado da arte, foi criada a extensão YOLOOD para implementar a detecção de objetos com modelos YOLOv5 em aplicativos desenvolvidos com o App Inventor, adicionalmente foi criado um web service hospedado no PythonAnywhere para processar as imagens e realizar a inferência sobre o modelo(03). Para que os alunos consigam utilizar a extensão, foi elaborado um material didático com slides de um tutorial detalhados sobre como utilizar a extensão desenvolvida e implementar o API para integração com o web service (04).

Deste modo, pretende-se que o trabalho ajude a tornar a aprendizagem de Machine Learning mais acessível e atrativa, permitindo que alunos da educação básica desenvolvam aplicativos com essas funcionalidades. Com isso, podemos despertar nos jovens o entusiasmo em aprofundar seus conhecimentos em sistemas de informação e possivelmente desenvolver soluções para a sociedade.

O modelo conceitual da solução foi alterado durante o desenvolvimento da extensão, esperava-se que fosse implementado uma extensão que não necessitasse de internet para realizar a detecção de objetos. Mas foram encontrados problemas técnicos que resultaram na alteração para que a inferência fosse realizada em uma web service.

Para trabalhos futuros, é recomendado que seja revisado o cálculo da posição das classes de cada bounding box na imagem melhorando a acessibilidade da extensão. É interessante que novas alternativas sejam estudadas para realizar o carregamento do modelo e inferência sejam realizados dentro do dispositivo, descartando a necessidade de acesso a internet.

Referências

APP INVENTOR. App Inventor Extensions. 2021 Disponivel em <http://ai2.appinventor.mit.edu/reference/other/extensions.html>

AMERSHI, S. et al. Software engineering for machine learning: A case study. In: Proc. of the IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, IEEE, 2019. p. 291-300.

FLASK, 2024. Disponível em <https://flask.palletsprojects.com/en/3.0.x/>. Acesso em: 1 de maio de 2024

GOOGLE. Glossário de machine learning. Disponível em: <https://developers.google.com/machine-learning/glossary?hl=pt-br#neural_network/>. 2023 .Acesso em: 7 maio. 2023

JAIN, A., SINGH, A., & KUMAR, A. Object detection: A survey. Journal of Computer Science and Technology, 31(3), 399-417. 2016.

LECUN, Y., BENGIO, Y. & HINTON, G. Deep learning. Nature 521, 436–444 (2015).

LIU, W., ANGUELOV,D., ERHAN,D., SZEGEDY,C., REED,S., FU Y. , BERG, A. SSD: single shot MultiBox detector, Springer International Publishing , pp. 21-37, 2016.

MARQUES, L. S., GRESSE VON WANGENHEIM, C., HAUCK, J. C. R. Ensino de Machine Learning na Educação Básica: um Mapeamento Sistemático do Estado da Arte. In: Anais do Simpósio Brasileiro de Informática na Educação, Natal, Brasil, 2020.

MARTINS. L. Desenvolvimento de um Curso de Machine Learning com Foco em Detecção de Objetos no Ensino Médio. 2022. Trabalho de Conclusão de Curso. (Graduação em Sistemas de Informação) – Universidade Federal de Santa Catarina.

MITCHELL, T. M. Machine Learning. McGraw-Hill Science/Engineering/Math; 1997.

MIT. About Us. Disponível em: <https://www.media.mit.edu/>. Acesso em: 8 maio. 2023.

TANG, D. Empowering Novices to Understand and Use Machine Learning With Personalized Image Classification Models, Intuitive Analysis Tools, and MIT App Inventor. M.Eng thesis, MIT, Cambridge, USA. 2019.

ONNX, 2023 Disponível em <https://onnx.ai/index.html> Acesso em: 22 abril 2022.

GIRSHICK, R., DONAHUE, J., DARRELL, T., MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation . arxiv 1311.2524, 2014.

PYTHONANYWHERE, 2024. Disponivel em <https://www.pythonanywhere.com/> Acesso em 01 de maio de 2024

SANTANA, J. P. Uma Ferramenta Visual para o Desenvolvimento de Modelos de Detecção de Objetos com Deep Learning no Ensino Superior. 2022. Trabalho de Conclusão de Curso. (Graduação em Ciência da Computação) – Universidade Federal de Santa Catarina.

TANG, D. Empowering Novices to Understand and Use Machine Learning With Personalized Image Classification Models, Intuitive Analysis Tools, and MIT App Inventor. M.Eng thesis, MIT, Cambridge, USA. 2019.

TAVARES, L. A. et al. Inteligência Artificial na Educação: Survey". Brazilian Journal of Development, 6(7), 2020.

ULTRALYTICS, 2021. Disponível em: <https://docs.ultralytics.com/> Acesso em: 10 julho 2022.

ULTRALITYCS, 2023. Disponível em: <https://docs.ultralytics.com/> Acesso em: 02 julho 2022.

WOLBER, D., ABELSON H., FRIEDMAN, M. Democratizing Computing with App Inventor. GetMobile: Mobile Computing and Communications. vol. 18, 2014, 53–58.

ZOU, Z. et al. Object Detection in 20 Years: A Survey. Proceedings of the IEEE, v. 111, n.3, p. 257-276, 2023.