UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

JEFFERSON A. BASTOS JUNIOR

Desenvolvimento de uma Extensão do App Inventor para Implantar Modelos de *Machine Learning* para Classificação de Imagens em Aplicativos

FLORIANÓPOLIS
2024

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

RESUMO

A inteligência artificial está cada vez mais integrada ao nosso dia a dia, contribuindo e transformando a vida das pessoas por meio da tecnologia. A disseminação deste conhecimento é crescentemente relevante, uma vez que a demanda por profissionais que precisarão lidar com IA aumentará significativamente nos próximos anos. Um ramo da inteligência artificial é o aprendizado de máquina (ML), que é a capacidade das máquinas de aprender, a partir do treinamento de um modelo de dados. Tendo em vista o grande crescimento da área, a popularização e a adoção do ML na educação se torna fundamental. Neste contexto, este trabalho visa criar uma extensão que permita a utilização de modelos de ML exportados do *Jupyter Notebooks* na plataforma *App Inventor* para desenvolvimento de aplicativos móveis. O *Jupyter Notebook* é um dos *softwares* mais usados atualmente para ML, o que torna a vivência do aluno mais perto da área atual de trabalho. Além disso, o desempenho das redes residuais (ResNet) a ser exportada/importada ser superior ao das redes neurais importadas em extensões já existentes, indicam a necessidade do desenvolvimento da extensão proposta neste trabalho. Com a extensão, é possível importar modelos de ML exportados no formato ONNX (*Open Neural Network Exchange*), o que traz mais flexibilidade, pois o formato ONNX é um formato aberto e interoperável que permite a portabilidade de modelos de *Machine Learning* entre diferentes *frameworks* e plataformas.

Palavras-chave: Inteligência Artificial, *Machine Learning*, classificação de imagens, *App Inventor*, Extensão, *ResNet*.

SUMÁRIO

LISTA DE ABREVIATURAS

MIT - *Massachusetts Institute of Technology*

ML - *Machine Learning*

TMIC - *Teachable Machine Image Classifier*

PIC - *Personal Image Classifier*

TCC - Trabalho de Conclusão de Curso

ONNX - *Open Neural Network Exchange*

IA - Inteligência Artificial

GTM - *Google Teachable Machine*

CNN - *Convolutional Neural Network*

CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

ONNX-ICE - *ONNX Image Classifier Extension*

LISTA DE TABELAS

LISTA DE FIGURAS

1. INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Hoje, a Inteligência Artificial (IA) está mais profunda em nossas vidas do que pensamos, afetando nossas vidas diárias. Um ramo da IA é o aprendizado de máquina (ML - *Machine Learning*), sendo a capacidade de sistemas de software aplicativos aprenderem com modelos de dados e usá-los para treinar máquinas para executar tarefas como reconhecimento de padrões (Mitchell, 1997). Se usa ML em pesquisas no *Google*, agentes de texto de *e-mail*, assistentes de voz digital, transações bancárias *online*, recomendações de filmes de plataformas de *streaming* e reconhecimento de imagens (Marr, 2020). Esta tecnologia vem tendo um impacto profundo na sociedade, eliminando diversos empregos que podem ser facilmente automatizados e criando novos (Frey et al., 2017). Desta maneira, as pessoas devem estar preparadas para lidar com o ML, e todas as preocupações de ética e segurança que envolvem o seu uso (Patil, 2020).

A demanda por profissionais com essas competências também é crescente (Daws, 2020), levando à necessidade de ensinar esse conhecimento aos jovens e inspirá-los a ingressar em carreiras nessa área (Touretzky et al., 2020). Assim, observando a importância hoje do ML, é importante popularizar este conhecimento desde a educação básica (Touretzky et al., 2020). Conforme o *Big Idea* #3, citado no *K-12 Guidelines for Artificial Intelligence: What Students Should Know (*Touretzky et al*.,* 2019*)*, esse conhecimento de aprendizado de máquina pode ser inserido na educação básica incluindo os principais conceitos de ML, como o que é aprendizado, tipos de algoritmos de aprendizado e arquiteturas de redes neurais, como o treinamento de modelos influenciam o aprendizado e as limitações desta tecnologia (Touretzky et al., 2020).

Atualmente, vários cursos estão sendo criados para o ensino de ML na educação básica (Marques et al., 2020). Alguns levam os alunos a aplicar esse conhecimento utilizando uma abordagem ativa que os orienta a criar modelos de ML por meio de um ciclo de usar-modificar-criar (Lee et al., 2011). Nesse ciclo, os alunos inicialmente seguem um tutorial para criar seu primeiro modelo predefinido e, em

seguida, começam a modificá-lo até eventualmente criarem um novo modelo de ML para resolver um problema identificado em sua comunidade ou vida (Alves et al., 2020). Por exemplo, o curso ML4TEENS (Cardozo, 2022) que ensina o desenvolvimento de um modelo de classificação de imagens usando VULCAN (Franz, 2021), uma camada visual para *Jupyter Notebook* rodando no *Google Colab*. Comumente, utilizam-se ambientes visuais para ensinar ML nesse nível educacional, tais como, o *Jupyter Notebook,* tanto na forma textual com Python quanto utilizando VULCAN (Franz, 2021). Os modelos serão treinados usando redes residuais ResNet (*Residual Network*) (HE et al., 2015), uma rede neural convolucional amplamente utilizada e reconhecida por sua capacidade de aprendizado profundo em tarefas de visão computacional. Uma vez treinados, a exportação dos modelos pode ser realizada nos formatos nativos das redes neurais em que foram treinados ou utilizando formatos de conversão, como ONNX (The Linux Foundation, 2019), que permite a interoperabilidade entre diferentes *frameworks* de aprendizado de máquina.

Porém, observa-se que atualmente existem poucos cursos que incluem a fase de implantação do modelo de ML (Marques et al., 2020). Para tornar esse ensino de ML mais dinâmico, pretendemos dar continuidade ao curso por meio do ensino de implantação na forma de aplicativos móveis, pois uma das formas típicas de ensinar computação, principalmente algoritmos e programação, é por meio do desenvolvimento de aplicativos (Wolber, 2020). Na educação básica, o desenvolvimento de aplicativos geralmente é ensinado usando um ambiente baseado em blocos, como o *App Inventor*. *App Inventor* (MIT, 2022) é uma plataforma de código aberto que permite implementar a parte funcional e o *design* de interface do aplicativo por meio de uma linguagem visual de blocos. Além disso, permite a geração de um arquivo .apk fácil de instalar e compartilhável ao final do projeto, promovendo a disseminação do conhecimento computacional (Lye e Koh, 2014). Sendo assim, ao integrar modelos de ML nesses aplicativos, os alunos podem criar aplicativos “inteligentes”, integrando o ensino de ML com o ensino de programação e algoritmos.

Já existem algumas propostas de evolução do ambiente do *App Inventor*, como o *Personal Image Classifier* (PIC) (Tang, 2019), que permite criar o modelo de

ML por um sistema *web* visual e oferece uma extensão do *App Inventor* para realizar a integração. Outro exemplo é o *Teachable Machine Image Classifier* (TMIC) (Oliveira, 2020), uma extensão criada para importar o modelo de ML exportado do *Google Teachable Machine* no *App Inventor.*

No entanto, é possível notar que o desempenho em termos de acurácia dos modelos de classificação de imagens treinados com o sistema *web* do *Personal Image Classifier* (Mobilenet) e do *Google Teachable Machine* (Mobilenet), utilizado na extensão TMIC, geralmente é inferior ao desempenho dos modelos criados utilizando redes residuais (ResNet) treinado no *Jupyter Notebooks*. Além disso, é importante considerar a necessidade de familiarizar os estudantes com ambientes de desenvolvimento de modelos de ML mais convencionais, como o *Jupyter Notebook* no *Google Colab*, a fim de permitir a utilização de modelos treinados nesse ambiente no *App Inventor*. Atualmente ainda não existe uma extensão que dê suporte a importação de um modelo ResNet exportado do *Jupyter notebook* no *App Inventor*.

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 implantação de modelos de *Machine Learning (*ResNet) treinados no *Jupyter Notebook* e exportados no formato ONNX. A extensão serve como suporte para evoluir o curso de ML4TEENS (Cardozo, 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 modelos de ML para classificação de imagens;

02. Levantar o estado da arte em relação a extensões do *App Inventor* semelhantes;
03. Desenvolver a extensão (ONNX *Image Classifier Extension* - ONNX-ICE) para a implantação do modelo de ML criado no *Jupyter Notebook* para classificação de imagens;
04. Desenvolver o material didático (*slides*) para ensinar o uso da extensão desenvolvida;

1.3 METODOLOGIA DE PESQUISA E TRABALHO

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

Etapa 1 – Fundamentação teórica: análise e síntese de conceitos básicos envolvidos no tema. Conceitos básicos de aprendizagem de máquina. É apresentado o ambiente de programação visual baseado em blocos *App Inventor* (MIT, 2022) 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 ensino de *Machine Learning* na educação básica.
Atividade 1.2: Sintetizar conceitos de exportação de modelos ResNet no formato ONNX.
Atividade 1.3: 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 ONNX - Image Classifier Extension seguindo um processo de desenvolvimento de *software* (Pressman, 2011), contemplando a análise dos requisitos, modelagem, implementação e documentação, em que cada um tem como objetivo desenvolver a qualidade do *software*:
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, será desenvolvido um material didático incluindo um modelo pré-treinado com ResNet no formato ONNX e *slide*.
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, é levantado o estado da arte e quais extensões existem para implementar modelos de *Machine Learning* classificação de imagens no *App Inventor*. No capítulo 4, é apresentada a solução, bem como a arquitetura, implementação, configuração e testes da mesma. Capítulo 5 é apresentado o material didático disponibilizado junto com a extensão. No capítulo 6 é a conclusão do trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos referentes a *Machine Learning*, *Deep Learning* e Redes Neurais. Ao final, é apresentada a ferramenta para desenvolvimento baseado em blocos *App Inventor* e o formato *Open Neural Network Exchange* (ONNX).

2.1 *MACHINE LEARNING*, *DEEP LEARNING* E REDES NEURAIS

Inteligência Artificial (IA) é um campo da ciência da computação que se concentra no desenvolvimento de sistemas capazes de realizar tarefas que exigiriam inteligência humana para serem executadas. Segundo Russell e Norvig (2010), "Inteligência Artificial é a arte de criar máquinas que executam tarefas que requerem inteligência quando executadas por seres humanos". Essas tarefas incluem desde reconhecimento de voz e imagem, até tarefas complexas como diagnóstico médico e direção autônoma de veículos (Russell et al., 2010). Uma das características distintivas da IA é a sua capacidade de aprender e melhorar a partir de experiências passadas, de forma a realizar tarefas com maior eficiência e precisão.

A IA é frequentemente dividida em duas categorias: IA simbólica e IA conexionista (Luger, 2009). A IA simbólica é baseada em programação lógica, em que as regras e conhecimentos são expressos explicitamente pelos programadores. Já a IA conexionista, também conhecida como aprendizado de máquina (*Machine Learning*), é capaz de aprender a partir de exemplos fornecidos, de forma a construir modelos estatísticos capazes de fazer predições sobre novos dados (Alpaydin, 2010). O *Machine Learning* é uma técnica que usa a capacidade dos computadores para complementar a inteligência humana, permitindo a execução de tarefas que seriam impossíveis de serem realizadas apenas com a percepção humana. Por meio da habilidade de examinar e processar grandes quantidades de dados, os programas de ML podem detectar padrões que estão além da capacidade de percepção humana (Shalev-Shwartz; Ben-David, 2014). Os algoritmos de ML são capazes de criar suas próprias representações por meio do processo de treinamento usando dados que

podem ser fornecidos por pessoas ou adquiridos pela própria máquina (Bishop, 2006).

Uma técnica de ML são redes neurais. Redes neurais artificiais são sistemas de computação vagamente inspirados pelas redes neurais biológicas que constituem os cérebros animais (Goodfellow et al., 2016). Tais sistemas aprendem a executar tarefas considerando exemplos, sem serem programados com regras específicas de tarefas. Por exemplo, a empresa *Google* usa uma rede neural convolucional (CNN) em seu aplicativo *Google* Fotos para identificar rostos em fotos e classificar objetos, como animais, paisagens e prédios (Szegedy et al., 2016). Após o treinamento as redes neurais fazem isso sem qualquer conhecimento prévio sobre os objetos a serem classificados, identificando automaticamente as características do material de aprendizagem que processam (Walkarn, 2019).

Deep learning trata-se de um conjunto de técnicas de *Machine Learning* que utilizam redes neurais artificiais profundas, com muitas camadas intermediárias entre a camada de entrada e a de saída (Lecun et al., 2015). Essa técnica tem como diferencial tecnológico os melhores resultados obtidos na resolução de problemas, superando o desempenho dos melhores especialistas em áreas como reconhecimento de características semânticas em imagens (Krizhevsky et al., 2012).

O reconhecimento de imagens é uma técnica fundamental em processamento de imagens e visão computacional (Szeliski, 2010). É uma das áreas mais ativas em aprendizado de máquina e tem uma ampla gama de aplicações em várias indústrias. A técnica de reconhecimento de imagens envolve o uso de algoritmos de aprendizado de máquina, em particular a aprendizagem supervisionada, para identificar objetos em uma imagem. O aprendizado supervisionado é a abordagem mais comum para o reconhecimento de imagens e envolve a construção de um modelo a partir de um conjunto de exemplos de treinamento rotulados (Bishop et al., 2006).

O desenvolvimento é realizado em várias etapas, que podem incluir o pré-processamento das imagens para melhorar a qualidade dos dados, e a escolha de um algoritmo de aprendizagem de máquina adequado (Bishop et al., 2006).

Durante o treinamento, o modelo aprende a identificar padrões nas imagens e a associá-los às categorias correspondentes.

No caso de classificação de imagens com um único objeto na imagem, também conhecida como "*single-label*", o modelo é treinado para identificar apenas uma classe ou categoria de objeto em uma imagem. Já na classificação de imagens com múltiplas categorias, também conhecida como "*multi-label*", o modelo é treinado para identificar várias classes ou categorias de objetos em uma única imagem. Por exemplo, o sistema pode reconhecer um gato e um cachorro na mesma imagem.

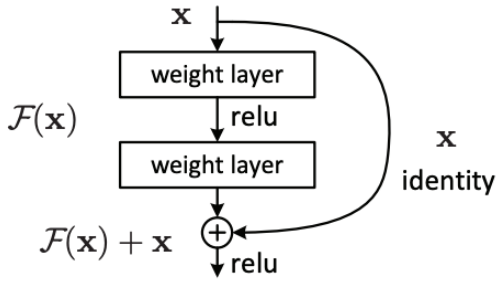
Existem diversas técnicas para realizar o reconhecimento de imagens, e uma das mais eficazes é a utilização de redes neurais convolucionais (CNNs). As CNNs são uma classe de redes neurais artificiais que possuem camadas especiais de convolução, capazes de extrair características relevantes de imagens por meio do uso de filtros (Lecun et al., 1998). Esse tipo de rede é especialmente eficaz no reconhecimento de imagens complexas e em grandes conjuntos de dados de treinamento, uma vez que é capaz de processar informações em paralelo e aprender representações hierárquicas das imagens.

ResNet. ResNet (*Residual Network*) é uma família de arquiteturas de redes neurais convencionais que foi introduzida em 2015 (He et al., 2015). Essas redes são conhecidas por sua capacidade de treinamento profundo e sua eficácia em tarefas de visão computacional, como classificação de imagens e detecção de objetos. A ideia fundamental por trás da ResNet é o uso de blocos residuais, que permitem a passagem de informações não transformadas (ou residual) das camadas anteriores para as camadas posteriores. Isso é alcançado por meio da adição de uma conexão direta (*skip connection*) que permite que a entrada do bloco seja adicionada à saída do bloco, contornando uma ou mais camadas intermediárias. Essa técnica de aprendizado é conhecida como aprendizado profundo residual.

A arquitetura básica do ResNet é composta por camadas convolucionais, camadas de *pooling* e blocos residuais. Os blocos residuais podem ser do tipo básico ou do tipo *bottleneck*, dependendo do número de filtros nas camadas intermediárias. A Figura 1 mostra um exemplo de bloco residual básico da ResNet, que é composto

por duas camadas convolucionais, um elemento de ativação ReLU e uma conexão direta (*skip connection*).

Figura 1: Exemplo de bloco residual básico da ResNet.

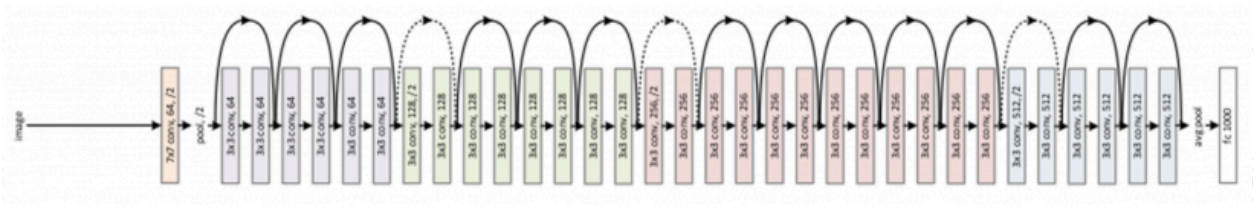


Fonte: Site https://minds.digital/data-science/entendendo-bloco-residual.

Existem diversas variantes de arquiteturas de ResNet, incluindo as versões ResNet-18, ResNet-34, ResNet-50, ResNet-101 e ResNet-152. Cada uma dessas versões difere no número de camadas e na complexidade da rede. As versões mais profundas, como a ResNet-101 e a ResNet-152, geralmente apresentam melhor desempenho em conjuntos de dados mais complexos, como o ImageNet. No entanto, essas redes também são mais pesadas e exigem mais recursos de computação.

Para aplicações em dispositivos móveis, é recomendado o uso de arquiteturas mais leves, como a ResNet-18 e a ResNet-34 (Howard et al., 2017). Essas redes possuem um número menor de camadas e filtros, tornando-as mais eficientes em termos de uso de recursos de memória e processamento e mesmo assim apresentam desempenho competitivo em tarefas de reconhecimento de imagens. A Figura 2 mostra um exemplo de arquitetura ResNet-34, com suas camadas e blocos residuais.

Figura 2: Arquitetura da ResNet-34.



Fonte: Site https://minds.digital/data-science/entendendo-bloco-residual.

Exportação de modelos treinados. Quanto ao processo de exportação de modelos treinados com ResNet, existem diferentes formatos que podem ser utilizados. Isso inclui o formato próprio da biblioteca de *deep learning* utilizada, como PyTorch ou TensorFlow, ou formatos independentes como o ONNX.

O ONNX (*Open Neural Network Exchange*) (https://onnx.ai/) é um formato aberto e interoperável para representação de modelos de *deep learning*. Ele foi desenvolvido por uma colaboração entre a *Microsoft* e o *Facebook* em 2017, e desde então tem sido adotado por diversas bibliotecas e ferramentas de *deep learning*. O ONNX permite que modelos de *deep learning* treinados em diferentes bibliotecas, como PyTorch, TensorFlow e Caffe2, possam ser convertidos para um formato padrão. Dessa forma, um modelo treinado em PyTorch pode ser facilmente convertido para o formato ONNX e ser usado em outras bibliotecas, como TensorFlow, ou até mesmo ser executado em *hardware* especializado.

O formato ONNX é baseado em um modelo de grafo direcionado acíclico, *Directed Acyclic Graph*, que representa a arquitetura do modelo de aprendizado de máquina em uma estrutura hierárquica de nós e arestas. Cada nó no grafo representa uma operação matemática, como uma convolução, uma multiplicação ou uma função de ativação, enquanto as arestas representam os dados de entrada e saída do modelo.

O modelo ONNX é representado por um arquivo com a extensão .onnx, que contém uma descrição JSON do modelo, juntamente com os dados binários do modelo. A descrição JSON do modelo inclui informações sobre a arquitetura do

modelo. Mais especificamente, o arquivo JSON descreve a estrutura do grafo computacional do modelo, incluindo as entradas e saídas do modelo, bem como as camadas e operações que compõem a rede neural. Cada camada da rede neural é descrita como um "nó" no grafo computacional do modelo e inclui informações como o nome da camada, o tipo de operação realizada pela camada (por exemplo, convolução, *pooling*, ativação), os parâmetros da camada (por exemplo, tamanho do filtro, número de neurônios) e os dados de entrada e saída da camada. Além disso, a descrição JSON do modelo ONNX também pode incluir informações adicionais, como os metadados do modelo (por exemplo, nome do autor, versão do modelo, licença), os dados de treinamento usados para gerar o modelo e informações sobre a conversão do modelo de outro formato para o formato ONNX. Segue uma lista com as informações do JSON:

1. Versão do formato ONNX;
2. Metadados do modelo, como nome do autor, versão do modelo e licença;
3. Informações sobre a conversão do modelo de outro formato para ONNX, se aplicável;
4. Informações sobre as entradas e saídas do modelo, incluindo nome, tipo e formato de dados;
5. A estrutura do grafo computacional do modelo, que inclui informações sobre as camadas e operações do modelo, como nome, tipo de operação e parâmetros;
6. Informações sobre as constantes usadas no modelo, como pesos e bias;
7. Informações sobre as funções e operações auxiliares usadas no modelo;
8. Informações sobre o treinamento do modelo, como conjunto de dados usados e configurações de treinamento;
9. Outras informações adicionais, como extensões personalizadas e metadados específicos do *framework*.

2.2 APP INVENTOR

O *App Inventor* (https://appinventor.mit.edu) é uma plataforma de desenvolvimento visual de aplicativos móveis para *Android*, criada pelo *Google* e atualmente mantida pelo Instituto de Tecnologia de Massachusetts (MIT). Ele permite que usuários sem conhecimento em programação possam criar aplicativos para *Android* de forma simples e intuitiva, usando uma interface visual baseada em designer e blocos (https://appinventor.mit.edu).

Além do *App Inventor*, existem outras ferramentas derivadas que também permitem criar aplicativos móveis de forma visual e intuitiva. Duas das ferramentas mais populares são o *Kodular* e o *Thunkable*. O *Kodular* (https://www.kodular.io/) é uma plataforma baseada no *App Inventor* que oferece recursos avançados e ampliados em relação à versão original. Ele oferece uma ampla variedade de componentes e recursos adicionais, como integração com serviços *web*, notificações *push*, autenticação de usuários, armazenamento em nuvem e muito mais. O *Thunkable* (https://thunkable.com/) éoutra plataforma derivada do *App Inventor* que permite criar aplicativos para *iOS* e *Android*. O *Thunkable* oferece uma variedade de componentes e funcionalidades, incluindo integração com serviços populares, como *Google Maps*, *Firebase*, *Airtable* e muito mais. Além disso, ele também suporta a criação de extensões personalizadas e permite exportar seus aplicativos para as lojas de aplicativos *Google Play* e *App Store*.

Uma das principais vantagens do *App Inventor* é sua capacidade de permitir que os usuários integrem novas funcionalidades em seus aplicativos por meio de extensões. As extensões do *App Inventor* são bibliotecas de código que adicionam novas funcionalidades ao ambiente de programação, permitindo que os usuários possam usar recursos de *hardware* específicos, como câmeras e sensores, ou integrar serviços da *web*, como o *Twitter* ou o *Google Maps*, em seus aplicativos. As extensões também podem ser usadas para adicionar componentes personalizados ao designer, permitindo que os usuários criem seus próprios elementos visuais.

Dessa forma, podem ser implantados modelos de ML, como, por exemplo, por meio do uso das extensões *Personal Image Classifier* (PIC) (Tang et al., 2019) e *Teachable Machine Image Classifier* (TMIC) (Oliveira, 2020), que ambos implantam modelos de classificação de imagens. O PIC utiliza redes neurais convolucionais (*Convolutional Neural Networks* - CNNs) treinadas na plataforma do *Personal Image Classifier*, utilizando o modelo *MobileNet*, enquanto o TMIC utiliza modelos treinados no *Google Teachable Machine* (GTM), que também são baseados em CNNs e utilizam MobileNet.

Para criar extensões para o *App Inventor*, é necessário usar o *framework* (https://mit-cml.github.io/appinventor-sources/). O *framework* é uma coleção de bibliotecas e ferramentas que permitem aos desenvolvedores criar e testar extensões. O *framework* inclui uma API Java que define a interface que as extensões devem implementar e fornece suporte para embalar e distribuir as extensões. Com o *framework*, é possível criar extensões para quase qualquer funcionalidade que um usuário possa precisar em um aplicativo, expandindo assim as possibilidades de criação e personalização dos aplicativos desenvolvidos no *App Inventor*.

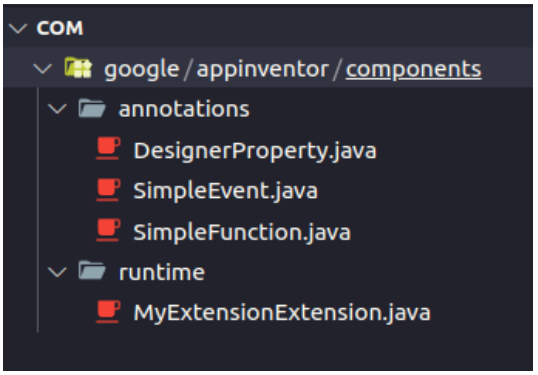
As extensões do *App Inventor* seguem uma estrutura de código específica. Uma visão geral da estrutura do código Java para extensões do *App Inventor*, incluindo um exemplo de hierarquia de arquivos:

- O código-fonte de uma extensão é dividido em dois pacotes principais: "com.google.appinventor.components.runtime" e "com.google.appinventor.components.annotations". O primeiro pacote contém as classes principais que implementam a funcionalidade da extensão, enquanto o segundo pacote contém as anotações usadas para descrever a interface da extensão.
- A classe principal de uma extensão é geralmente nomeada como a própria extensão, seguida pelo sufixo "*Extension*".
- A classe principal da extensão geralmente estende a classe "*AndroidNonvisibleComponent*" e implementa a interface "*Component*", que é fornecida pelo *App Inventor*. Essas classes fornecem a estrutura básica necessária para a extensão funcionar com o *App Inventor*.

- As anotações são usadas para descrever a interface da extensão e indicar ao *App Inventor* quais propriedades, métodos e eventos devem ser expostos aos usuários. As anotações são aplicadas a campos, métodos e classes da extensão.
- O código-fonte de uma extensão geralmente é dividido em vários arquivos Java. Por exemplo, uma extensão pode ter um arquivo Java para a classe principal, outro arquivo para as anotações, e um terceiro arquivo para classes auxiliares.

O esquema na Figura 3 apresenta um exemplo de hierarquia de arquivos para uma extensão fictícia chamada "MyExtension".

Figura 3: Exemplo de hierarquia de arquivos *app inventor.*



Fonte: Autor, 2024.

3. ESTADO DA ARTE

Foi realizado um mapeamento sistemático, seguindo os procedimentos propostos por Petersen et al. (2008), com o objetivo de identificar as extensões disponíveis para a implantação de modelos de *Machine Learning* no *App Inventor*. Esta abordagem foi similar à utilizada por Oliveira (2021), que conduziu um mapeamento sistemático detalhado para levantar o estado da arte sobre extensões de *App Inventor* voltadas para a classificação de imagens com modelos de ML. Oliveira adotou critérios de inclusão e exclusão, bem como fontes de dados variadas para garantir uma revisão abrangente, o que serviu de base metodológica para o presente trabalho.

3.1 DEFINIÇÃO DO PROTOCOLO DE REVISÃO

Com o intuito de identificar as extensões disponíveis para a implementação de modelos de *Machine Learning* (ML) no *App Inventor*, foi realizado um mapeamento sistemático seguindo os procedimentos propostos por Petersen et al. (2008). As seguintes questões de análise foram refinadas a partir da pergunta de pesquisa: Quais são as extensões existentes para a implementação de modelos de classificação de imagens no *App Inventor*? Com base no objetivo desta revisão, a pergunta de pesquisa é refinada nas seguintes questões de análise:

Q1. Quais são as extensões disponíveis?

Q2. Em quais ambientes de desenvolvimento são treinados e exportados os modelos de machine learning?

Q3. Como os modelos treinados são importados nas extensões?

Q4. Quais são as características funcionais das extensões?

Critérios de inclusão/exclusão. A seleção foi restrita a extensões específicas do *App Inventor* que permitem a importação de modelos de ML para classificação de imagens. Extensões do *App Inventor* voltadas para outras tarefas, como reconhecimento de fala, foram excluídas do escopo desta análise.

Critérios de qualidade. Foram selecionados exclusivamente artigos ou materiais que forneciam informações suficientes sobre a descrição das extensões do *App Inventor*.

Fontes dos dados. Foram examinados todos 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*, por indexar um grande conjunto de dados de diferentes fontes (Haddaway et al. 2015), já que nessa área emergente não foram necessariamente publicados artigos científicos das extensões criadas. Dado o foco de pesquisa do MIT *Media Lab* nessa área e foco técnico, são buscadas também extensões nas páginas deste grupo e nos fóruns do *App Inventor*.

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: *String* de busca para cada fonte.

Fonte	**String de busca**
SCOPUS	TITLE-ABS-KEY (("*machine learning*" OR "*artificial intelligence*" OR "*deep learning*") AND ('*App Inventor*' OR '*kodular*' OR '*thunkable*')) AND (SUBJAREA("COMP"))
Google	*"machine learning" “artificial intelligence” "image classification" “App Inventor”*
	"machine learning" “artificial intelligence” "image classification" “kodular”
	"machine learning" “artificial intelligence” "image classification" “thunkable”
MIT media lab (https://appinventor.mit.edu/explore/research)	“image classification”
Lista de extensões do *App Inventor*: Pura vida Apps (https://puravidaapps.com/extensions.php)	“image classification”
MIT *App Inventor* Community (https://community.appinventor.mit.edu/c/extensions)	“image classification”

Para o *MIT Media Lab*, *Pura Vida Apps* e *MIT App Inventor Community*, não foi possível utilizar a *string* de busca, uma vez que esses ambientes não possuem ferramentas de busca por *string*. Em vez disso, utilizaram-se as ferramentas de busca nativas do navegador para pesquisar pelas palavras-chave listadas na tabela referente a essas fontes. Devido ao fato de serem fóruns exclusivos do *MIT App Inventor*, palavras-chave como "*App Inventor*", "*Kodular*" e "*Thunkable*", presentes em outras *strings* de busca, não foram consideradas necessárias nestes casos.

3.2 EXECUÇÃO DA BUSCA

Em abril de 2023, o autor conduziu a busca revisada pela co-orientadora. Apesar da adaptação das *strings* de busca, algumas pesquisas ainda apresentavam uma grande quantidade de resultados. Isso ocorreu devido à semelhança dos termos de busca com artigos que descrevem o uso de IA na educação, como no aprendizado de análise de dados.

Tabela 2: Número de artigos identificados por repositório e por fase de seleção.

Fonte	**Nº de resultados da busca**	**Nº de resultados analisados**	**Nº de resultados potencialmente relevantes**	**Nº de resultados relevantes**
SCOPUS	28	28	4	0
Google Busca 1	2.100	100	7	4
Google Busca 2	5.610	100	4	3
Google Busca 3	4.820	100	6	2
MIT media lab	99	99	23	3
Lista de extensões Pura Vida Apps	23	23	3	3
Fórum da comunidade *App Inventor*	121	121	5	5
Total (sem duplicatas)				**7**

Na primeira fase da análise, foram examinados títulos, resumos e fóruns, resultando em 38 resultados potencialmente relevantes (sem duplicatas). Na segunda etapa, os materiais foram lidos integralmente para garantir sua relevância com relação aos critérios de inclusão/exclusão.

Algumas extensões foram excluídas por não se referirem à classificação de imagens, como por exemplo *Personal Audio Classifier* (Bhatia, 2020) ou de agentes de conversação (Zhu, 2020). Foram encontrados também artigos apresentando *apps* feitos com *App Inventor* utilizando classificação de imagens, como por exemplo para pessoas cegas (Sangeetha et al., 2022), *Date fruit classification* (Vikraman et al., 2022), que propõe um sistema de classificação automática de tâmaras, porém, como usaram extensões já identificadas na versão anterior da revisão (PIC) não trouxeram mais extensões novas para esta versão atual da revisão da literatura.

3.3 RESULTADOS DA REVISÃO

De acordo com as questões de análise, as informações relevantes foram extraídas dos materiais encontrados. A extração de dados foi afetada pelo fato de que apenas duas extensões, *PIC* (Tang, 2019) e TMIC (Oliveira, 2021) foram publicações acadêmicas (artigo e TCC), enquanto as outras extensões foram encontradas em fóruns específicos voltados para plataformas de aprendizado de programação, com poucos detalhes sobre o desenvolvimento das extensões. São apresentados os resultados acrescentando as novas extensões encontradas em relação aos resultados já relatados pelo Oliveira (2021) (marcados em cinza nas tabelas nas próximas seções).

3.3.1 QUAIS EXTENSÕES EXISTEM?

Observa-se que há uma certa motivação e necessidade para esse tipo de extensão, já que algumas foram encontradas e há uma certa demanda por soluções semelhantes nos fóruns pesquisados durante a revisão. No entanto, ainda existe um número pequeno de extensões e limitações quanto ao método de utilização dos modelos treinados. No total, foram encontradas 7 extensões gratuitas.

Tabela 3: Extensões encontradas.

Nome da extensão	**Breve descrição**	**Extensão para integração no *App Inventor***	**Referências**	**Link**	**Licença de uso**
PIC (Personal Image Classifier)	Este sistema consiste em uma plataforma online que permite a criação de modelos de aprendizado de máquina para classificar imagens, juntamente com uma extensão que possibilita a integração desse modelo em um aplicativo desenvolvido com o *App Inventor*.	Arquivo no formato AIX	(Tang, 2019) (Tang et al., 2019)	https://classifier.appinventor.mit.edu/	Apache (http://www.apache.org/licenses/LICENSE-2.0)
Look Extension	Realiza classificação de objetos baseado em um modelo pré-treinado/fixo usando a extensão	Arquivo no formato AIX	--	https://mitcml.github.i o/e	Apache (http://www.apache.org/licenses/LICENSE-2.0)
ML4K Extension	Realiza classificação de imagens, textos e números baseado em um modelo pré-treinado	Arquivo no formato AIX	--	https://github.com/kylecorry31/ML4K-AIExtension	MIT License
Microsoft Image Recognizer (Thunkable)	Realiza classificação de imagens a partir de um modelo pré-treinado hospedado na *Microsoft Azure*	Arquivo no formato AIX	--	https://docs.thunkable.com/imagerecognizer	--
AWS AI Services *App Inventor* Extension	Permite que usuários utilizem serviços de IA da *Amazon Web Services* para converter texto em voz, traduzir textos entre diferentes idiomas e detectar objetos e texto em imagens	Arquivo no formato AIX	--	https://github.com/ceyhunozgun/awsAIServicesAppInventorExtension/wik i	--
Artificial Intelligence Image Classification App 2	Um modelo treinado pelo usuário pode ser integrado num app no *App*	Somente via componente *WebViewer*	--	https://appinventor.tmsoftwareinc.com/en/archives/2437	--

	Inventor mostrando os resultados da classificação via WebViewer				
TMIC - *Teachable Machine Image Classifier*	Uma Extensão do *App Inventor*, que permite a importação de modelos de classificação de imagens *Machine Learning* treinados no *Google Teachable Machin e* exportados para *Google cloud*	Arquivo no formato AIX	OLIVEIRA, 2021		BSD 3 (OPEN SOURCE INITIATIVE, 2022)

Observa-se que há uma busca considerável por parte da comunidade do *App Inventor* em fóruns por extensões que possibilitem o uso de modelos treinados no *Teachable Machine*, assim como tutoriais que instruam como criar essa funcionalidade.

3.3.2 EM QUAIS AMBIENTES DE DESENVOLVIMENTO OS MODELOS DE ML SÃO TREINADOS E COMO ELES EXPORTAM OS MODELOS?

A Tabela 4 apresenta as extensões, o formato dos modelos de exportação e o ambiente de treinamento.

Tabela 4 - Formato de exportação

Nome da extensão	Treinados em que ambiente	Formato dos modelos de exportação	Link do mbiente
PIC (Personal Image Classifier)	Aplicação web do PIC	Utiliza um arquivo na extensão mdl (formato próprio, incluindo 4 arquivos, 3 arquivos no formato JSON que possuem dados do modelo e labels, e um arquivo no formato BIN que contém os pesos do modelo)	https://classifier.appinventor.mit.edu/
Look Extension	--	Possui um modelo próprio já pré-treinado fixo no código que é carregado no *WebViewer* via URL no formato *TensorFlow.js.*	--
ML4K Extension	--	Possui modelos já pré-treinados disponíveis no	--

		site *Machine Learning for Kids* (https://machinelearningforkids.co.uk/#!/pretrained)	
Microsoft Image Recognizer (Thunkable)	--	O modelo é importado do servidor da *Azure* na execução do aplicativo de que formato	--
AWS AI Services *App Inventor* Extension	--	O modelo é importado do servidor da *AWS Web* Services na execução do aplicativo de que formato	--
Artificial Intelligence Image Classification App 2	--	Diretamente em um arquivo index.html	Qualquer ambiente que possa ser utilizado *TensorFlow.js*
TMIC - *Teachable Machine Image Classifier*	*Google Teachable Machine*	Arquivo *Tensorflow.js* a ser armazenado na nuvem. Extensão usa link compartilhável gerado para acessar o modelo treinado.	https://teachablemachine.withgoogle.com/

As extensões listadas oferecem diferentes abordagens para a implementação de modelos de *Machine Learning* voltados para a classificação de imagens no ambiente do *App Inventor*.

A *LookExtension* e a *AWS AI Services App Inventor Extension* não permitem ao usuário treinar e exportar modelos de *Machine Learning*. A *AWS AI Services App Inventor Extension* utiliza o bloco *DetectLabels*, que faz uso do serviço online de classificação de imagens da *Amazon*, o *Amazon Rekognition*. A extensão retorna as etiquetas de classificação da imagem em forma de uma string separada por vírgulas. Por sua vez, a *LookExtension* oferece blocos como o *ClassifyImageData*, que classifica imagens com um modelo de *Machine Learning* próprio, não treinado pelo usuário.

A extensão *ML4K AppInventor Extension* possibilita aos usuários criar modelos de *Machine Learning* através de uma página web dedicada. Nessa plataforma (https://machinelearningforkids.co.uk/), os modelos podem ser treinados utilizando a *webcam* ou importando dados de treinamento. Os modelos são mantidos *online* pelo projeto *Machine Learning for Kids*. A extensão permite a importação dos modelos no *App Inventor* por meio de uma chave de API configurada.

No *Artificial Intelligence Image Classification App* 2, é possível utilizar qualquer modelo treinado, desde que seja possível importá-lo por meio de código *JavaScript* em um arquivo *index*.*html*. No entanto, é necessário importar esse arquivo *HTML* na plataforma *App Inventor* para tornar seu uso possível.

A extensão PIC (Tang, 2019) fornece uma interface *web* própria que permite a criação e o treinamento do modelo de ML. É possível exportar o modelo em uma extensão .mdl, que contém os pesos, rótulos e arquivos do modelo compactados. Em contraste, as outras extensões não possuem um ambiente próprio para o treinamento e exportação de modelos específicos.

A extensão TMIC (Oliveira, 2022) é utilizada para a implantação de modelos de *Machine Learning* treinados com o *Google Teachable Machine* (GTM). Após o treinamento, o modelo pode ser hospedado na nuvem pelo próprio *Teachable Machine*, obtendo um *link* para acessá-lo. No entanto, é importante destacar que o TMIC também apresenta algumas limitações em relação à eficiência e à precisão das redes geradas. As redes treinadas no *Google Teachable Machine* com *MobilenetV1* podem demonstrar um desempenho inferior quando comparadas com outros modelos como ResNet. Portanto, embora o TMIC permita o treinamento de modelos personalizados em uma plataforma independente da extensão, é necessário levar em consideração as limitações relacionadas à precisão e à acurácia das redes geradas pela plataforma.

3.3.3 QUAIS SÃO AS CARACTERÍSTICAS FUNCIONAIS DAS EXTENSÕES?

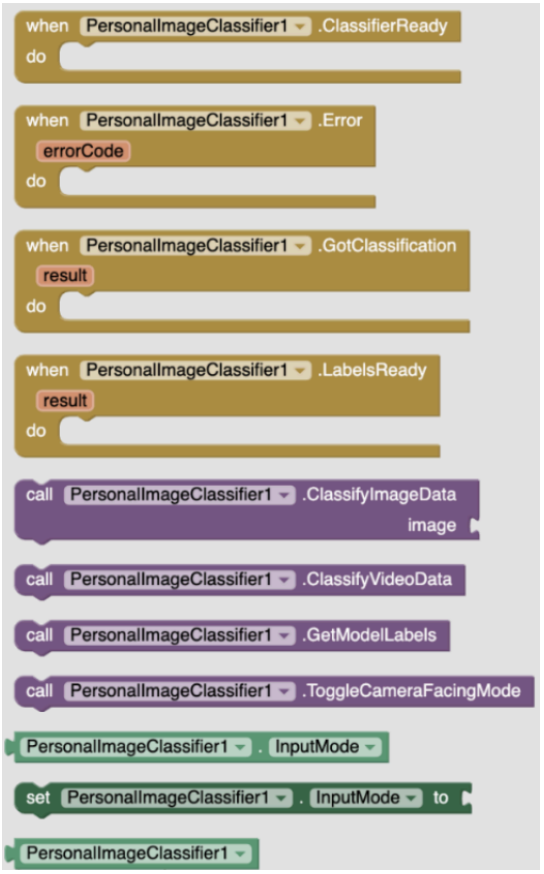
Tabela 5 - Características funcionais.

Extensão	Blocos lógicos	Mecanismo de funcionamento	Funciona sem acesso à internet
AWS AI Services App Inventor Extension	23	Roda diretamente no aplicativo	Não
Artificial Intelligence Image Classification App 2	--	Roda pelo componente WebViewer	sim
Microsoft Image Recognizer (Thunkable)	7	Roda diretamente no aplicativo	Não
ML4K Extension	14	Roda diretamente no aplicativo	Não

Look Extension	9	Roda diretamente no aplicativo	Sim
PIC (Personal Image Classifier)	16	Roda diretamente no aplicativo	Sim
TMIC	6	Roda diretamente no aplicativo	não

Em geral, a maioria das extensões encontradas na pesquisa possui uma ampla variedade de blocos disponíveis para serem utilizados em seus componentes. No entanto, há uma exceção: o *Artificial Intelligence Image Classification App 2*. Esta extensão não possui blocos específicos e executa suas funcionalidades diretamente no navegador do celular, sem a necessidade de blocos adicionais. Os blocos lógicos da extensão PIC são apresentados na Figura 4.

Figura 4: Blocos lógicos da extensão PIC.



Fonte: Autor, 2024.

Exceto pelo *Artificial Intelligence Image Classification App 2*, todas as extensões podem ser executadas diretamente no aplicativo, permitindo que o usuário acesse os resultados das classificações imediatamente, sem se limitar apenas à visualização na tela do dispositivo. Além disso, as extensões TMIC, PIC, *Look Extension Microsoft Image Recognizer* e *AWS AI Services App Inventor Extension* possuem funcionalidades semelhantes, permitindo a captura de imagem pelo celular e sua classificação.

Dentre todas as extensões, três possibilitam a classificação de imagens sem a necessidade de acesso à internet (PIC, *Look Extension* e *Artificial Intelligence Image Classification App* 2). As extensões *Microsoft Image Recognizer* e *AWS AI Services App Inventor Extension* exigem a validação das chaves de acesso e o envio da imagem para os servidores da *Microsoft Azure* e *AWS*, respectivamente, onde os serviços utilizados pelas extensões realizam as classificações. Já a extensão *ML4K Extension* requer uma chave de acesso para o uso de modelos pré-treinados disponíveis em seu site.

3.4 DISCUSSÃO

Durante o levantamento do estado da arte, foram encontradas apenas sete extensões voltadas à classificação de imagens que cumprem os critérios de inclusão/exclusão e de qualidade. As duas mais bem documentadas são o *Personal Image Classifier* (PIC), desenvolvida por Tang (2019) usando uma ferramenta web própria para o treinamento do modelo de ML, e o *Teachable Machine Image Classifier* (TMIC) desenvolvida por Oliveira (2021), que permite importar os modelos treinados no GTM (*Google Teachable Machine)*. Ambas podem ser utilizadas na plataforma *App Inventor* para implantar os modelos de ML treinadas para classificar imagens de fotos, e possuem a vantagem de permitir o desenvolvimento de aplicações personalizadas.

A extensão *Microsoft Image Recognizer* tem como ponto fraco a necessidade de acesso direto ao servidor da *Microsoft Azure*, além de precisar de uma chave de registro gerada no site da empresa. A *AWS AI Services App Inventor Extension*

também precisa acessar o servidor da *AWS Web Services* e verificar a chave de acesso gerada. A *ML4K Extension* exige uma chave de acesso para utilizar um modelo pré-treinado.

Em relação ao ambiente de treinamento, a *Look Extension* não permite que o usuário utilize seu próprio modelo treinado. O mesmo ocorre com as extensões *Microsoft Image Recognizer* e *AWS AI Services App Inventor Extension*, onde a primeira permite que o usuário crie seu próprio servidor e chave de uso do *software* no site da *Microsoft Azure*, e a segunda utiliza AWS Web Services para gerar e checar a chave de acesso e realizar a classificação. A extensão *ML4K Extension* não possui assistência a modelos treinados pelo usuário, sendo possível apenas a utilização de modelos pré-treinados.

A única extensão que fornece um ambiente próprio para a criação do modelo de *machine learning* é o PIC (Tang, 2019). No entanto, é importante mencionar que, apesar de oferecer um ambiente próprio, essa abordagem apresenta desvantagens. A plataforma própria não acompanha os avanços contínuos do *Machine Learning*, resultando em uma acurácia inferior quando comparada a plataformas externas de treinamento que utilizam configurações de redes mais eficientes. Portanto, embora o ambiente próprio do PIC seja uma opção conveniente, é necessário considerar suas limitações em relação ao desempenho e à precisão dos modelos gerados.

Tanto o PIC quanto o TMIC utilizam a arquitetura de rede neural convolucional *MobileNet*, que foi uma das primeiras e projetada para dispositivos móveis com recursos computacionais limitados. Embora tenha sido eficiente em termos de uso de recursos e adequada para dispositivos móveis no momento de seu lançamento, o desempenho relativo da *MobileNetV1* em comparação com as arquiteturas mais recentes e avançadas pode ser considerado inferior. Com o avanço da pesquisa em redes neurais convolucionais, novas redes como ResNet foram desenvolvidas superando a *MobileNetV1* em termos de precisão e eficiência computacional. Portanto, a dependência da arquitetura *MobileNet* é uma desvantagem significativa, uma vez que não há, no momento, uma extensão que possa importar modelos mais eficientes.

Além disso, é importante ressaltar que o TMIC apresenta uma limitação quanto à disponibilidade de modelos. Ele só é capaz de acessar modelos disponibilizados na *Google Cloud*, o que restringe as opções de modelos disponíveis para utilização. Por outro lado, o PIC permite o carregamento dos arquivos do modelo treinado, o que proporciona mais flexibilidade neste aspecto.

Essas informações destacam a necessidade de considerar as limitações da arquitetura utilizada e a disponibilidade de modelos ao avaliar o desempenho e a precisão do PIC e do TMIC. A utilização da arquitetura *MobileNetV1* pode resultar em modelos com desempenho inferior em relação às arquiteturas mais recentes, e a dependência da *Google Cloud* para acessar modelos no TMIC pode restringir as opções disponíveis. Outra desvantagem é a falta de uma versão da interface *web* disponível na língua portuguesa (PIC). Não foi possível encontrar uma extensão que permita treinar/importar outros modelos tipicamente adotados para classificação de imagens, como ResNet*,* treinados em ambientes como *Jupyter Notebook* no *google colab* e exportado como por exemplo no formato ONNX. Com isso, é possível notar a necessidade de uma extensão que possibilite implementar estes modelos dentro de *apps* com *App Inventor*.

4. SOLUÇÃO

Este capítulo apresenta a solução para implementar uma extensão - ONNX *Image Classifier Extension* (ONNX-ICE). Esta extensão permite a integração de modelos de aprendizado de máquina, exportados no formato ONNX, no *App Inventor*, possibilitando que os usuários desenvolvam aplicativos móveis capazes de classificar imagens utilizando esses modelos treinados previamente.

4.1 ANÁLISE DE CONTEXTO E DEFINIÇÃO DA EXTENSÃO

O objetivo principal deste trabalho é desenvolver uma solução que facilite a implantação de modelos de classificação de imagens (ResNet18 e ResNet34) exportado no formato *ONNX* dentro de aplicativos sendo desenvolvidos no *App Inventor* a ser adotado no contexto de ensino de computação na educação básica.

4.2 ANÁLISE DE REQUISITOS

Com o objetivo de desenvolver uma solução integrada ao *App Inventor* que facilita a utilização de modelos de *Machine Learning* para classificação de imagens, foi necessário definir requisitos funcionais e não funcionais que garantam a operação eficiente e prática da solução proposta.

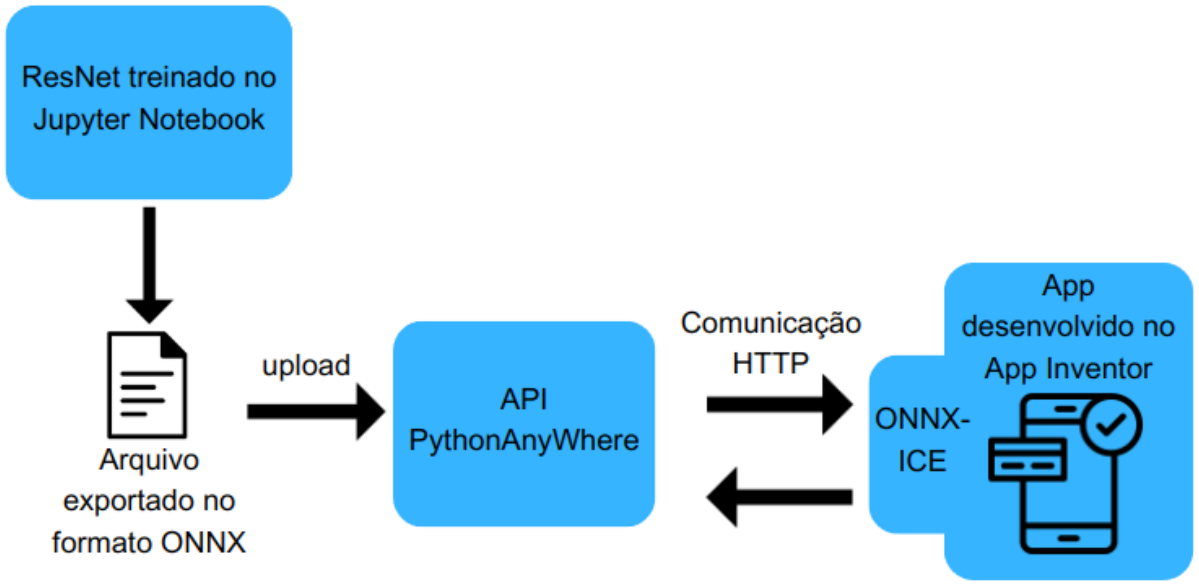
A solução proposta consiste em uma extensão do *App Inventor* que se comunica com uma API hospedada na plataforma *PythonAnywhere* para processar modelos ONNX. O funcionamento conceitual da solução pode ser descrito da seguinte forma:

Os modelos de *Machine Learning*, treinados e exportados no formato ONNX, são enviados para a API hospedada no *PythonAnywhere*. Quando o usuário deseja classificar uma imagem, o *App Inventor* captura a imagem, seja pela câmera do dispositivo ou selecionando uma imagem já salva na galeria, e esta imagem é enviada pelo app para a extensão que então se comunica via HTTP com a API. A API, por sua vez, realiza a inferência utilizando o modelo ONNX e processa a imagem recebida. O resultado da classificação é então retornado para a extensão no

App Inventor, que recebe e envia os dados para que o aplicativo possa lidar e manipular os resultados.

Essa abordagem permite que o processamento de modelos ONNX ocorra em um servidor remoto, utilizando a infraestrutura da plataforma *PythonAnywhere*. A inferência é realizada no servidor, e os resultados são retornados para a extensão no *App Inventor*. Isso facilita a manutenção e atualização dos scripts de inferência, que podem ser modificados diretamente no servidor sem a necessidade de atualizações na extensão. A Figura 5 ilustra o fluxo de dados e as interações entre os componentes do sistema.

Figura 5: Fluxo de dados e interações dos componentes do sistema.



Fonte: Autor, 2024.

4.2.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais da extensão e script a ser desenvolvida estão descritos na Tabela 6.

Tabela 6: Requisitos funcionais da solução.

ID	Requisito	Descrição
RF01	Configurar API no *App Inventor*.	A solução deve permitir que os usuários configurem o nome do modelo, o nome do arquivo de classes e o nome do usuário no *App Inventor*.
RF02	Comunicar o com a *API*.	A solução deve facilitar a comunicação com a API hospedada, enviando imagens e recebendo resultados de classificação.
RF03	Disponibilizar os resultados da classificação de imagem obtidas.	A extensão deve disponibilizar os resultados da classificação de imagem realizada pelo modelo de *Machine Learning importado* (incluindo nomes de todas as categorias e os valores de confiança do resultado em porcentagem), permitindo que o usuário utilize estas informações em outros blocos do aplicativo.
RF04	Gerir as Configurações	Os usuários devem ser capazes de gerenciar e ajustar as configurações da *API* diretamente no *App Inventor*, facilitando o uso e a manutenção da extensão.
RF05	*Endpoint* de Classificação de Imagens	O *script* deve prover um *endpoint* /classificarImagem que aceite imagens via *POST*, realize a inferência usando o modelo especificado e retorne a classificação.
RF06	Utilizar fotos salvas no celular para classificar a imagem.	A extensão deve permitir usar uma foto salva no celular para realizar a classificação, e enviar para api realizar a classificação.
RF07	Gerenciamento de Erros e Respostas	O *script* deve gerenciar e responder adequadamente a erros comuns, como arquivos de imagem não enviados ou problemas de leitura de arquivos.

4.2.2 REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais da extensão e do script a ser desenvolvida estão descritos na Tabela 7.

Tabela 7: Requisitos não funcionais solução.

ID	Requisito	Descrição
RF01	*Back-end* da extensão desenvolvida em Java.	O *back-end* da extensão deve ser desenvolvido na linguagem de programação Java utilizando o *framework* de desenvolvimento do *App Inventor*.
RF03	Sistema operacional Android	A extensão deve ser implementada em aplicativos para o sistema operacional *Android*.
RF04	*Script* desenvolvido em *Python*	O *script* para inferência deve ser desenvolvido em python..
RF05	*Internet* disponível.	Uso do aplicativo com o modelo de ML em ambientes com acesso à *internet* disponível.

4.3 ARQUITETURA DA EXTENSÃO

A arquitetura da extensão ONNX-ICE é composta por vários componentes que interagem para possibilitar a classificação de imagens utilizando modelos de ResNet treinados e exportados em ONNX. Inicialmente, a intenção da extensão era suportar a importação de modelos ResNet no formato ONNX e possibilitar que, de forma *offline*, o aplicativo pudesse fazer a inferência a partir do modelo. No entanto, durante o desenvolvimento da extensão, surgiram problemas de compatibilidade de versão do *Android* e do *Java* utilizados no *App Inventor* em relação à biblioteca ONNX Runtime (https://onnxruntime.ai/), a mesma biblioteca usada no *script* de inferência que será detalhado mais adiante neste trabalho.

Para desenvolver a extensão, é obrigatório o desenvolvimento dentro do ambiente do *App Inventor*, que possui versões de *Android* e *Java* que impossibilitaram a construção (*build*) da extensão. Na Figura 6 está o erro que ocorre ao tentar fazer o *build* da extensão, mesmo sem usar a biblioteca, somente com a importação da mesma:

Figura 6: Erro *onnx runtime* no *app inventor.*

```
dexExtension:
     [java]
     [java] PARSE ERROR:
     [java] InvokeDynamic not supported
     [java] ...while preparsing cst 0003 at offset 00000014
     [java] ...while parsing ai/onnxruntime/providers/OrtCUDAProviderOptions.
class
     [java] Uncaught translation error: com.android.dx.cf.code.SimException:
default or static interface method used without --min-sdk-version >= 24
     [java] 2 errors; aborting

BUILD FAILED
```

Fonte: Autor, 2024.

Foram testadas várias versões do ONNX Runtime, incluindo a versão mais recente até o momento deste trabalho, que é a versão 1.18. Além disso, foram testadas versões específicas para Android e exportadas no formato .aar (*Android Archive*), como a versão *onnxruntime-mobile-1.17.1.aar*, que apresentaram também erros de compatibilidade de ambiente.

4.3.1 MODELO CONCEITUAL

Devido a 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. Para viabilizar essa integração sem exigir conhecimentos avançados de infraestrutura por parte dos alunos, a inferência dos modelos é realizada por meio de uma API hospedada na plataforma *PythonAnywhere* (PythonAnywhere, 2024). *PythonAnywhere* é um serviço de hospedagem baseado em nuvem que permite a execução de *scripts Python* em um ambiente controlado. A plataforma oferece diferentes planos, incluindo opções gratuitas com algumas limitações, porém que atendem as necessidades.

A extensão ONNX-ICE agora funciona como um intermediário que envia imagens capturadas ou selecionadas no dispositivo móvel para a API hospedada no *PythonAnywhere*, onde a inferência é realizada. Os resultados da classificação são então retornados para o aplicativo, onde podem ser utilizados e exibidos.

No entanto, essa solução também apresenta algumas desvantagens que devem ser consideradas. A necessidade de acesso à *internet* para realizar a inferência, que elimina a possibilidade de operação *offline*, pode ser uma limitação em ambientes com conectividade precária. Adicionalmente, a plataforma *PythonAnywhere*, apesar de oferecer um ambiente robusto e acessível, impõe limitações na conta gratuita em termos de processamento, armazenamento de arquivos e disponibilidade do serviço, o que pode restringir o uso a longo prazo e contínuo sem custos. Por fim, a necessidade de configuração inicial por parte do usuário, que inclui o *upload* do modelo e do *script*, além da gestão das *URLs* da *API*, adiciona um passo adicional no processo de utilização da extensão.

A seguir, detalhamos o fluxo de dados e as interações entre os componentes do sistema.

Fluxo de Dados e Interações:

1. **Treinar o Modelo de Classificação De Imagens:** Os alunos treinam modelos de classificação de imagens (ResNet18 ou ResNet34) na plataforma *Jupyter Notebook* e exportam o modelo treinado no formato *ONNX*.
2. ***Upload* do Modelo e *Script*:** O aluno se cadastra na plataforma *PythonAnywhere*, importa o modelo treinado e exportado, as classes e o script de inferência. O arquivo de classes contém os índices necessários para que a inferência possa identificar corretamente a classe prevista.
3. **Configuração da Extensão:** No *App Inventor*, os alunos incluem a extensão ONNX-ICE e configuram as propriedades da extensão ONNX-ICE com as propriedades necessárias (nome de usuário

cadastrado na plataforma *PythonAnywhere*, nome do modelo carregado, nome do arquivo de classes).

4. **Implementação do App:** Os alunos desenvolvem o aplicativo no *App Inventor* utilizando a extensão ONNX-ICE. Configuram os blocos visuais para capturar imagens e enviá-las para a API.
5. **Envio de Imagens para Classificação:** O aplicativo pode usar a câmera ou imagens salvas no dispositivo para enviar para a API. A extensão ONNX-ICE lida com o envio dessas imagens para o servidor.
6. **Visualização dos Resultados:** Os resultados da classificação são recebidos pelo aplicativo e exibidos. O aluno pode decidir como visualizar esses resultados, utilizando-os de diversas maneiras dentro do aplicativo.

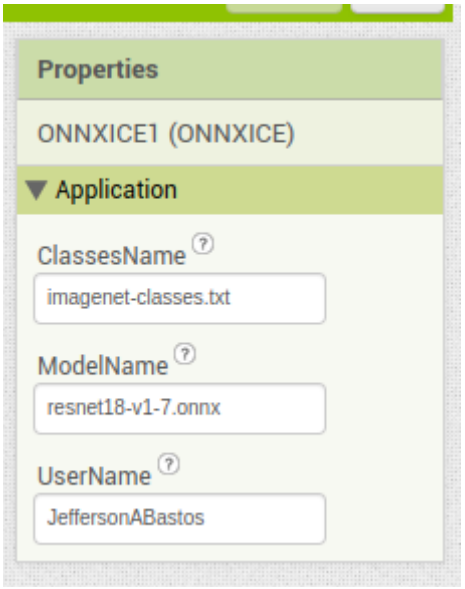
4.3.2 BLOCOS E PROPRIEDADES DA EXTENSÃO

Os componentes da extensão *ONNX-ICE* foram desenvolvidos para facilitar a integração e utilização dos modelos no *App Inventor*. A seguir, apresentam-se as propriedades, funções e eventos disponíveis na extensão.

Propriedades da Extensão:

- **ModelName:** Define o nome do modelo carregado na plataforma *PythonAnywhere*.
- **ClassesName:** Define o nome do arquivo de classes carregado na plataforma *PythonAnywhere*.
- **UserName:** Define o nome de usuário registrado na plataforma *PythonAnywhere*.

Figura 7: Propriedades da extensão ONNX.



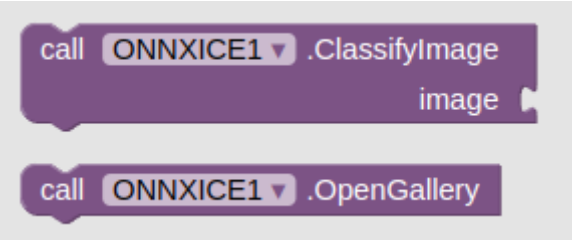
Fonte: Autor, 2024.

Funções da Extensão:

- **classifyImage:** Envia uma imagem para classificação via API. Recebe a imagem que deve ser classificada (*string*).
- **OpenGallery:** Abre a galeria de fotos do dispositivo para selecionar uma imagem e envia para classificação.

Abaixo, a imagem dos blocos no App Inventor.

Figura 8: Blocos de funções.



Fonte: Autor, 2024.

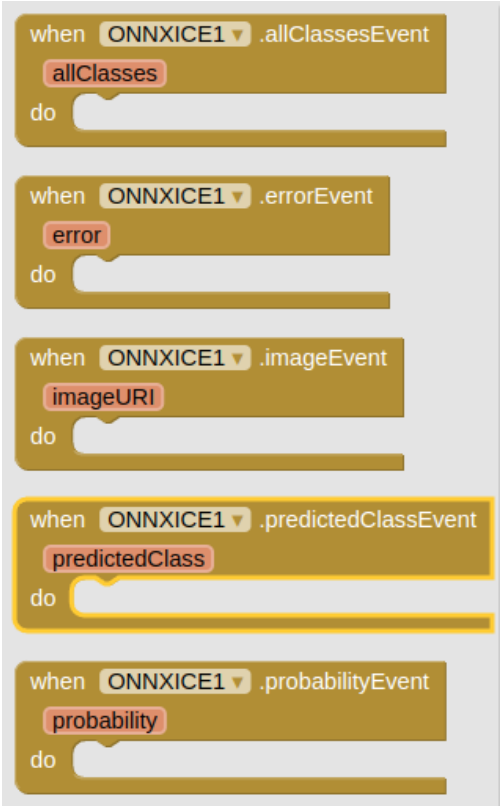
Eventos da Extensão:

- **allClassesEvent:** Disparado quando a resposta da API é retornada, contém todas as classes com as respectivos valores de confiança com todas as classes previstas.

- **imageEvent:** Disparado quando uma imagem é carregada da galeria.
- **errorEvent:** Disparado quando a extensão captura um erro.
- **predictedClassEvent:** Disparado quando a resposta da API é retornada, contém a classe prevista (com maior probabilidade). Exemplo: “cachorro”
- **probabilityEvent:** Disparado quando a resposta da API é retornada, contém a probabilidade da classe prevista, valor formatado com 2 dígitos depois da vírgula. Exemplo: “98.97”

Abaixo, a imagem dos blocos de evento:

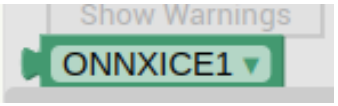
Figura 9: Blocos de evento.



Fonte: Autor, 2024.

O bloco getter ONNX-ICE retorna uma instância específica da extensão.

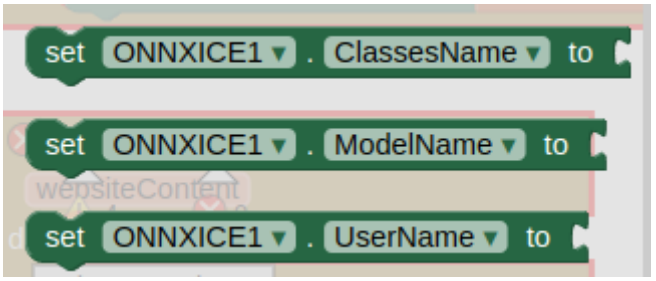
Figura 10: Blocos instância.



Fonte: Autor, 2024.

Outros blocos também são encontrados para atualizar as propriedades da extensão, conforme é possível visualizar na figura abaixo:

Figura 11: Blocos para atualização de propriedades.



Fonte: Autor, 2024.

4.4 IMPLEMENTAÇÃO DA API

A implementação da *API* na plataforma *PythonAnywhere* é um componente essencial deste projeto, permitindo a execução dos modelos de classificação de imagens treinados e exportados no formato ONNX. Esta seção detalha o processo de configuração e execução da API, além de fornecer uma visão técnica sobre o funcionamento da extensão *ONNX-ICE* no *App Inventor*.

A API é construída usando o *framework Flask* (https://flask.palletsprojects.com/), que é adequado para criar aplicações *web* leves e rápidas em *Python*. A *API* recebe imagens enviadas pelo aplicativo desenvolvido no *App Inventor*, processa essas imagens utilizando o modelo *ONNX* e retorna as previsões de classificação. A seguir, é apresentado o código da API com explicações detalhadas sobre suas partes constituintes.

Rota Padrão: Define uma rota básica para verificar se o servidor está funcionando corretamente. Quando a URL raiz é acessada, retorna a mensagem

'Parabéns, você finalizou a etapa de configuração do *pythonAnyWhere* com sucesso!!!'.

Figura 12: Rota padrão.

```
@app.route('/')
def hello_world():
    return 'Parabéns, você finalizou a etapa de configuração do pythonAnyWhere com sucesso!!!'
```

Fonte: Autor, 2024.

Rota de Classificação: A API fornece uma rota ***/classifyImage*** para a classificação de imagens. A seguir, detalha-se a funcionalidade dessa rota, incluindo os parâmetros esperados e o formato da resposta.

Rota: /classifyImage

Método: POST

Formato de Solicitação: A solicitação deve ser do tipo *multipart/form-data*, contendo os parâmetros descritos abaixo.

Parâmetros:

- **user:** Nome de usuário registrado na plataforma PythonAnywhere.
  - **tipo:** texto
  - **exemplo:** “JeffersonBastos”
- **model:** Nome do arquivo do modelo ONNX carregado na plataforma.
  - **tipo:** texto
  - **exemplo:** “modelo.onnx”
- **classes:** Nome do arquivo de classes carregado na plataforma.
  - **tipo:** texto
  - **exemplo:** “classes.txt”
- **image:** Arquivo de imagem enviado para classificação.
  - **tipo:** File (binário)

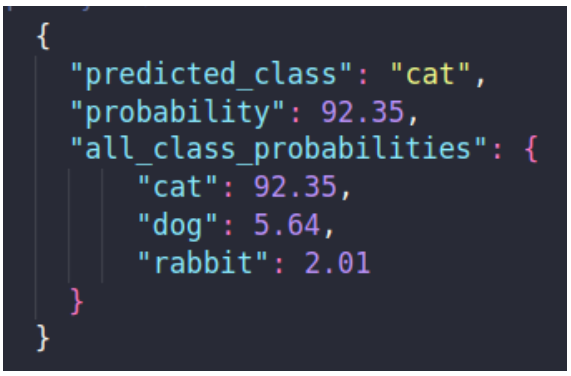
Corpo da Resposta: A resposta da API é um *JSON* contendo:

- **predicted_class:** Classe prevista pelo modelo.
- **probability**: Probabilidade da classe prevista (em porcentagem).

- **all_class_probabilities:** Dicionário contendo todas as classes e suas respectivas probabilidades.

Exemplo de Resposta:

Figura 13: Resposta rota /classifyImage.



Fonte: Autor, 2024.

4.4.1 INFERÊNCIA E NORMALIZAÇÃO DA IMAGEM

A inferência é realizada utilizando a biblioteca *onnxruntime* (https://onnxruntime.ai/), que foi escolhida pelas seguintes razões:

- **Desempenho:** O *ONNX Runtime* é otimizado para executar modelos *ONNX* de forma eficiente em diferentes plataformas (Microsoft, 2022).
- **Compatibilidade:** Suporte abrangente para vários *frameworks* de *Machine Learning*, facilitando a interoperabilidade entre diferentes ferramentas (Microsoft, 2022).
- **Facilidade de Uso:** Interface intuitiva e simples de usar, permitindo uma integração tranquila com aplicativos desenvolvidos em *Python* (Microsoft, 2022).

Normalização da Imagem:

A normalização das imagens é uma prática comum em inferências de classificação de imagens, garantindo que as imagens de entrada estejam no

formato esperado pelo modelo. O pipeline de transformação utilizado é o padrão para tarefas de visão computacional:

- **Resize:** Redimensiona a imagem para 256x256 pixels.
- **CenterCrop**: Recorta o centro da imagem para obter uma imagem de 224x224 pixels.
- **ToTensor:** Converte a imagem para um tensor.
- **Normalize:** Normaliza a imagem com médias e desvios padrão específicos (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) (Pytorch, 2023).

Este pipeline de transformação é amplamente adotado devido à sua eficácia em padronizar as entradas para modelos pré-treinados, melhorando a precisão das predições (Pytorch, 2023). A normalização conforme descrito é essencial para garantir que as imagens estejam no formato adequado, contribuindo para a precisão do modelo durante a inferência. O código da normalização é detalhado na Figura 14.

Figura 14: Normalização da imagem.

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

img_processed = transform(img)
img_processed = np.expand_dims(img_processed, 0).astype(np.float32)
```

Fonte: Autor, 2024.

4.4.2 TRATAMENTO DE ERROS

A *API* pode lançar diferentes erros dependendo das situações encontradas. A seguir, são listados os principais erros e os cenários em que são lançados:

- **400 Bad Request:**

- **"Parameters 'user', 'model', and 'classes' are required.":** Lançado quando algum dos parâmetros obrigatórios (user, model, classes) não é fornecido.
- **"No image file was provided.":** Lançado quando o arquivo de imagem não é fornecido na solicitação.
- **"No file selected for upload.":** Lançado quando o arquivo de imagem não é encontrado no campo image.
- **"Class file '{class_file_path}' not found.":** Lançado quando o arquivo de classes especificado não é encontrado.
- **"Error processing the image: {error_message}":** Lançado quando há um erro no processamento da imagem.
- **"Error loading the model: {error_message}":** Lançado quando há um erro ao carregar o modelo *ONNX*.
- **"Error getting model input/output names: {error_message}":** Lançado quando há um erro ao obter os nomes de entrada/saída do modelo.
- **"Error running inference: {error_message}":** Lançado quando há um erro ao executar a inferência.

- **500 Internal Server Error:**
  - **"Unexpected error: {error_message}":** Lançado quando ocorre um erro inesperado no servidor.

4.4.3 CONFIGURAÇÃO DA API NO PYTHONANYWHERE

Para implementar a solução proposta, foi necessário configurar uma API que processa modelos ONNX, utilizando a plataforma *PythonAnywhere*. Esta etapa é crucial, pois permite a comunicação entre a extensão do *App Inventor* e o serviço de inferência de modelos, possibilitando a classificação de imagens. A seguir, apresenta-se o processo detalhado para a configuração da API no PythonAnywhere, em um estilo técnico e impessoal.

Criação de Conta no *PythonAnywhere*

Para criar uma conta no PythonAnywhere:

1. Acesse o site *PythonAnywhere* e clique em "*Pricing & Signup*".
2. Selecione a opção "*Create a Beginner account*".
3. Insira os dados de nome de usuário, e-mail e senha.
4. Aceite os termos e condições marcando a caixa de seleção.
5. Complete o cadastro clicando em "*Register*".
6. Confirme o e-mail de cadastro clicando no link enviado para o *e-mail* registrado.

Configuração do Ambiente

Para configurar o ambiente:

1. No dashboard do *PythonAnywhere*, vá para a seção "*Web*".
2. Clique em "*Add a new web app*".
3. Selecione "*Flask*" como o *framework*.
4. Escolha a versão do *Python* 3.10 e clique em "*Next*".
5. Aguarde a criação da aplicação *web*.

Upload do Script e Modelos

Para realizar o *upload* dos arquivos necessários:

1. Na seção "*Files*" do *dashboard*, navegue até a pasta "*mysite*".
2. Faça *upload* do arquivo *flask_app.py*, substituindo o existente.
3. Faça upload dos arquivos de classes (.txt) e do modelo treinado (.onnx).

Instalação das Bibliotecas Necessárias

Para instalar as dependências:

1. No dashboard, acesse a seção "Consoles" e abra um novo console Bash.
2. Execute o comando a seguir para instalar as bibliotecas necessárias:

“pip3.10 install onnxruntime pillow torchvision”

Execução da API

Para verificar se a aplicação está funcionando corretamente:

1. Acesse o *link* fornecido pelo *PythonAnywhere*.
2. A mensagem "Parabéns, você finalizou a etapa de configuração do *pythonAnyWhere* com sucesso!!!" deve ser exibida, indicando que a aplicação está rodando corretamente.

Este processo garante a correta configuração da API para realizar a inferência de modelos ONNX no ambiente PythonAnywhere.

4.5 IMPLEMENTAÇÃO DA EXTENSÃO ONNX-ICE

A extensão ONNX-ICE foi desenvolvida utilizando como base o *framework* do *App Inventor* para a criação de extensões. Ela se concentra na comunicação com uma API hospedada na plataforma *PythonAnywhere*, onde a inferência dos modelos *ONNX* é realizada. A seguir, detalham-se os principais componentes e funções da extensão, bem como sua implementação.

A classe principal da extensão, ONNXICE.java, é responsável por definir os blocos e métodos que os usuários utilizarão no *App Inventor*. Cada bloco é definido como um método dentro dessa classe, e esses métodos contêm anotações

específicas que permitem ao *framework* do *App Inventor* identificar suas funções. Por exemplo, a anotação *@SimpleFunction()* é usada para a função *classifyImage*, como mostrado na Figura 14.

Figura 15: Método *ClassifyImage* ONNX-ICE.

```
@SimpleFunction(description = "Send image to classify")   Jefferson Junior *
public void ClassifyImage(String image) {
    try {
        Uri imageUri = Uri.parse(image);
        sendImageToClassification(imageUri);
    } catch (Exception e) {
        Log.e( tag: "ClassifyImage",  msg: "Failed to parse URI or send image", e);
    }
}
```

Fonte: Autor, 2024.

Principais Métodos e Blocos da Extensão

Os métodos da classe ONNXICE.java definem o comportamento dos blocos que os usuários verão no *App Inventor*. Estes métodos incluem funcionalidades para configurar a API, enviar imagens para classificação, e lidar com as respostas da API. A seguir, são apresentados os principais métodos e seus correspondentes blocos no App Inventor (Tabela 8).

Tabela 8: Principais métodos da classe ONNXICE.

Nome do método	Parâmetros	Ação	Bloco correspondente	Eventos disparados
ModelName	*modelName string*	Atualiza ou define o nome do modelo carregado na plataforma PythonAnywhere.	ModelName	-
ClassesName	*classesName string*	Define o nome do arquivo de classes carregado na plataforma PythonAnywhere.	ClassesName	-

UserName	*user string*	O *script* para inferência deve ser desenvolvido em python.sifique imagens sem necessidade de conexão com a internet.	UserName	-
ClassifyImage	*image string*	Converte a URI da imagem em um formato apropriado e chama sendImageToClassification	ClassifyImage	errorEvent, predictedClassEvent, probabilityEvent allClassesEvent
OpenGallery	-	Abre a galeria de fotos do dispositivo para selecionar uma imagem.	OpenGallery	imageEvent, errorEvent, predictedClassEvent, probabilityEvent allClassesEvent
resultReturned	*int requestCode, int resultCode, Intent data*	Lida com o retorno dos resultados após seleção da imagem na galeria e chama a função sendImageToClassication.	-	-
sendImageToClassification	*imageUri Uri*	Envia a imagem selecionada para a API no PythonAnywhere e processa a resposta.	-	-

A função *sendImageToClassification* é central para a operação da extensão. Ela envia a imagem selecionada para a API no *PythonAnywhere* e processa a resposta. A função *sendImageToClassification* utiliza a biblioteca *HttpURLConnection* para enviar a imagem para a API e processar a resposta. A imagem é convertida para um formato adequado, enviada para a *API*, e os resultados são capturados, tratados e disparados usando eventos.

Figura 16: Tratamento do retorno da api.

Fonte: Autor, 2024.

4.6 TESTE

Para validar o funcionamento da extensão ONNX-ICE desenvolvida neste trabalho, foi criado o aplicativo ZooScan. Este aplicativo visa classificar imagens de animais, utilizando um modelo de aprendizado de máquina obtido do repositório ONNX. O ZooScan é baseado em um modelo *ResNet-18* de classificação de imagens, treinado e exportado no formato *ONNX Runtime*, utilizando o *dataset ImageNet*. O *dataset ImageNet* contém mil classes diferentes, abrangendo uma vasta gama de objetos, desde objetos do cotidiano até uma ampla variedade de animais.

Para o desenvolvimento do ZooScan, a intenção era restringir o aplicativo à classificação de imagens de animais, dado que o *dataset ImageNet* é muito extenso e o aplicativo de teste não precisa classificar todas as imagens disponíveis neste *dataset*. As principais classes do *dataset ImageNet* para classificação incluem várias espécies de mamíferos, aves, répteis, peixes e outros animais. A escolha desse tema dá um sentido e um nome ao aplicativo, facilitando a contextualização do teste da extensão.

O modelo *ResNet-18* utilizado no ZooScan é disponibilizado na documentação do *ONNX Runtime*

(https://github.com/onnx/models/tree/main/validated/vision/classification/resnet). Este modelo é reconhecido por seu desempenho robusto em tarefas de classificação de imagens, apresentando uma acurácia de aproximadamente 69.93% no conjunto de validação do ImageNet (ONNX, 2024). Além do *ResNet-18*, também foram realizados testes utilizando o modelo *ResNet-34*, que apresenta uma acurácia de 73.73% (ONNX, 2024).

A escolha desses modelos e do *dataset ImageNet* se dá pelo fato de que, utilizando um *dataset* com um tamanho de arquivo maior e exigindo maior processamento, como o *ImageNet*, podemos validar a extensão ONNX-ICE para seu uso em situações que exigem maior capacidade computacional. Isso demonstra que a extensão pode lidar com modelos robustos, assegurando sua eficácia também para os modelos menores e menos exigentes que serão treinados pelos alunos.

Abaixo o link para o arquivo ONNX:
https://drive.google.com/file/d/1MW2d_B9REr4j7mSSUgcyHZIyQIq31P-i/view?usp=drive_link

Estrutura do Aplicativo. O ZooScan possui duas telas principais:

Tela de Apresentação: A tela inicial do aplicativo apresenta uma interface simples com informações sobre o aplicativo e um botão que direciona o usuário para a tela de classificação.

Figura 17: Tela inicial ZooScan.

Fonte: Autor, 2024.

Tela de Classificação: Esta tela é a interface principal onde ocorre a interação do usuário com o sistema de classificação. Nela, encontram-se quatro botões com as seguintes funções:

- **Botão Voltar:** Localizado no canto superior esquerdo, permite ao usuário retornar à tela de apresentação.
- **Botão Tirar Foto:** Utiliza um componente nativo do App Inventor para capturar uma foto com a câmera do dispositivo.
- **Botão Buscar Imagem na Galeria:** Chama a função *OpenGallery* da extensão ONNX-ICE, permitindo que o usuário selecione uma imagem salva no dispositivo. Esta função já realiza a classificação da imagem selecionada.
- **Botão Classificar Imagem:** Após tirar uma foto, este botão chama a função *ClassifyImage*, passando a imagem capturada como parâmetro para a classificação. Em seguida, os resultados são exibidos na tela. Neste exemplo, optei por mostrar apenas o nome da classe com maior confiança, mas essa foi uma decisão específica para este aplicativo de exemplo. O

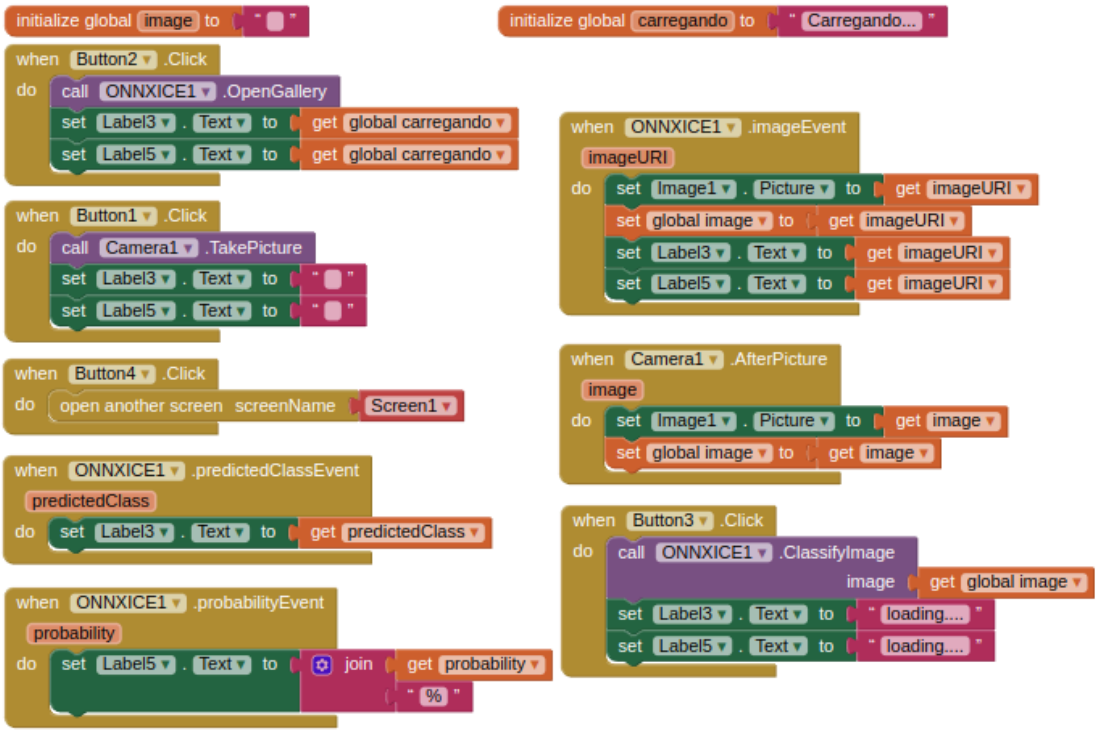
aluno pode lidar com o resultado da forma que achar mais adequada, utilizando os dados fornecidos pela extensão ONNX-ICE.

Figura 18: Tela de classificação do app ZooScan

Fonte: Autor, 2024.

A Figura 19 apresenta todos os blocos correspondentes dessa tela.

Figura 19: Blocos tela de classificação do app ZooScan.



Fonte: Autor, 2024.

Link para o arquivo aia do app:
https://drive.google.com/file/d/1PRtfLNE3VSfrSG_b1L2hxJyEtGkARsW8/view?usp=sharing

Funcionamento Detalhado do Aplicativo Teste - ZooScan

Ao iniciar o aplicativo, o usuário é recebido na tela de apresentação. A partir daí, ele pode navegar para a tela de classificação, onde pode optar por tirar uma nova foto ou selecionar uma imagem existente da galeria. Quando uma imagem é escolhida, a extensão ONNX-ICE processa a imagem enviando-a para a *API* configurada no *PythonAnywhere*.

A função *ClassifyImage* da extensão recebe a *URI* da imagem e a converte em um bitmap. Em seguida, esta imagem é enviada para a API por meio de uma solicitação *HTTP POST*, contendo os parâmetros necessários, como o nome do modelo e do arquivo de classes. A *API* processa a imagem utilizando o modelo

ResNet-18 e retorna a classe prevista e a probabilidade, que são então exibidas no aplicativo.

Avaliação do atendimento dos Requisitos

A Tabela 9 apresenta os resultados da avaliação dos requisitos funcionais definidos anteriormente para a extensão e o script, demonstrando como o aplicativo ZooScan atende a cada um deles.

Tabela 9: Requisitos funcionais avaliados na prática.

ID	Requisito	Descrição	Teste
RF01	Configurar API no *App Inventor*	A solução deve permitir que os usuários configurem o nome do modelo, o nome do arquivo de classes e o nome do usuário no *App Inventor*.	O aplicativo permite a configuração dessas propriedades através da interface do *App Inventor*.
RF02	Comunicação com a *API*	A solução deve facilitar a comunicação com a API hospedada, enviando imagens e recebendo resultados de classificação.	O aplicativo enviou imagens para a API e recebeu os resultados de classificação conforme esperado.
RF03	Disponibilizar os resultados da classificação	A extensão deve disponibilizar os resultados da classificação de imagem realizada pelo modelo de *Machine Learning* importado, permitindo que o usuário utilize estas informações em outros blocos do aplicativo.	Os resultados da classificação, incluindo nomes das categorias e valores de confiança, foram exibidos no aplicativo.
RF04	Gestão de Configurações	Os usuários devem ser capazes de gerenciar e ajustar as configurações da API diretamente no *App Inventor*, facilitando o uso e a manutenção da extensão.	A interface do *App Inventor* permitiu o ajuste das configurações da API conforme necessário.

RF05	Endpoint de Classificação de Imagens	O *script* deve prover um *endpoint /classifyImage* que aceite imagens via *POST*, realize a inferência usando o modelo especificado e retorne a classificação.	O *endpoint* foi testado com sucesso, realizando a classificação das imagens enviadas pelo aplicativo.
RF06	Utilizar fotos salvas no celular	A extensão deve permitir usar uma foto salva no celular para realizar a classificação, e enviar para *API* realizar a classificação.	A funcionalidade de selecionar e classificar imagens da galeria foi validada com sucesso.
RF07	Gerenciamento de Erros e Respostas	O *script* deve gerenciar e responder adequadamente a erros comuns, como arquivos de imagem não enviados ou problemas de leitura de arquivos.	A extensão e a *API* lidaram corretamente com cenários de erro, retornando mensagens apropriadas.

Dessa forma, o aplicativo ZooScan não só valida a funcionalidade da extensão ONNX-ICE como também demonstra a sua aplicabilidade prática, confirmando que todos os requisitos funcionais definidos foram atendidos. A implementação deste aplicativo fornece uma primeira indicação de que a solução desenvolvida pode ser viável e eficaz para integrar modelos de *Machine Learning* em aplicativos móveis utilizando o *App Inventor*.

5. MATERIAL DIDÁTICO

Visando a continuidade do curso ML4TEENS (Cardozo, 2022), que aborda a etapa de implantação de modelos, foi desenvolvido um tutorial para apoiar professores e estudantes na utilização da extensão ONNX-ICE. Para esse fim, disponibilizou-se um material didático contendo os arquivos descritos na Tabela 10. Como objetivos de aprendizagem, espera-se que, ao final do tutorial, o aluno compreenda o funcionamento do ONNX-ICE e seja capaz de desenvolver seu próprio aplicativo no *App Inventor*, utilizando a extensão e o modelo treinado para classificação de imagens.

Tabela 10: Arquivos que compõem o material didático.

Material	**Tipo**	**Descrição**	**Link**
Tutorial de implantação do modelo com ONNX-ICE	Slide	Slides apresentando passo a passo de como usar a extensão ONNX-ICE no app inventor	https://drive.google.com/file/d/1EMcSDwidpiS7KVkCTjZDDfhndd0yz3g7/view?usp=sharing
App ZooScan - versão final com o ONNX-ICE incluído.	Arquivo de extensão AIA	Projeto do aplicativo ZooScan apresentado nos *slides* com ONNX-ICE para ser utilizado no *App Inventor*. Esta versão já possui a extensão ONNX-ICE incluída e todas as funcionalidades programadas.	https://drive.google.com/file/d/1PRtfLNE3VSfrSG_b1L2hxJyEtGkARsW8/view?usp=sharing
Wireframe do app ZooScan	Arquivo de extensão AIA	Projeto do aplicativo ZooScan apresentado nos *slides* com ONNX-ICE a ser utilizado no *App Inventor* apresentando somente um esqueleto básico do app.	https://drive.google.com/file/d/1EtyfKvlLCxpauJZ8fnkNnVKdt6DNoI_B/view?usp=sharing

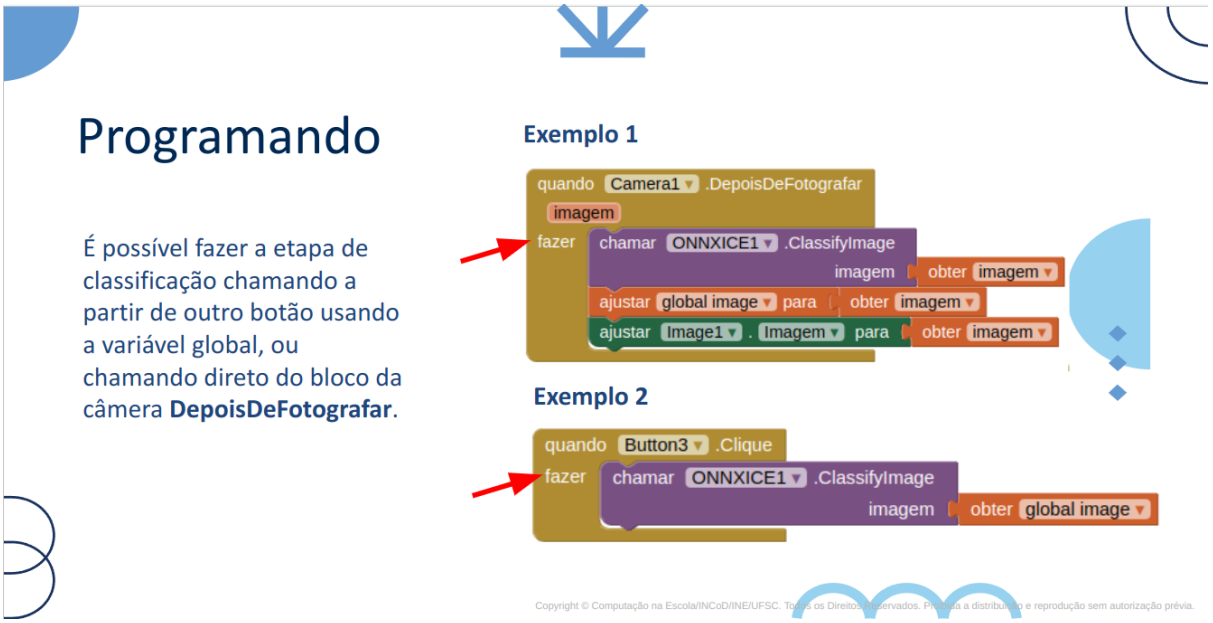
A primeira parte da aula, apresentada em formato de *slides*, consiste em fornecer ao aluno um modelo previamente treinado, acompanhado pelo respectivo arquivo de classes. Este modelo é o mesmo utilizado no teste demonstrado no tópico anterior. Nesta etapa, é explicada a finalidade do arquivo de classes, o qual será utilizado posteriormente na configuração do ambiente *PythonAnyWhere*, conforme ilustrado na Figura 20.

Figura 20: Configuração do ambiente *PythonAnyWhere*

Fonte: Autor, 2024.

Na segunda parte, aborda-se a configuração da API do *PythonAnyWhere*, desde a criação da conta até o *upload* dos arquivos necessários. Em seguida, desenvolve-se o aplicativo *ZooScan* no *App Inventor*, utilizando a extensão ONNX-ICE, detalhando seus blocos e funções.

Figura 21: Screenshot detalhamento do uso da extensão ONNX-ICE



Fonte: Autor, 2024.

Por fim, a última parte da aula ensina o aluno a testar o aplicativo desenvolvido no *App Inventor*. São fornecidas instruções detalhadas sobre como instalar o aplicativo no celular e solucionar possíveis problemas durante os testes.

Adicionalmente, foi criado um *wireframe* de extensão para suporte à aula, sem a inclusão do ONNX-ICE, permitindo que o aluno siga os passos indicados nos slides e crie o aplicativo de forma independente. A versão final do projeto ZooScan já estará com o aplicativo completo, incluindo o ONNX-ICE. A própria extensão ONNX-ICE é parte integrante do material didático, estando disponível para *download* por professores e alunos, que podem utilizar o arquivo AIX em seus projetos no *App Inventor*.

6. CONCLUSÃO

O presente trabalho teve como objetivo principal desenvolver uma solução integrada para a classificação de imagens no ambiente *App Inventor*, utilizando modelos de *Machine Learning* treinados e exportados no formato *ONNX*. Por meio da criação da extensão *ONNX Image Classifier Extension* (ONNX-ICE) e da implementação de uma API hospedada na plataforma *PythonAnywhere*, foi possível proporcionar uma ferramenta prática e eficiente para alunos e desenvolvedores de aplicativos móveis.

A solução proposta apresentou diversas contribuições significativas. Primeiramente, a extensão ONNX-ICE facilitou a integração de modelos ONNX com o *App Inventor*, simplificando o processo de classificação de imagens sem exigir conhecimentos avançados em programação ou infraestrutura. Além disso, a ferramenta desenvolvida mostrou-se potencialmente útil em ambientes educacionais, permitindo que alunos do ensino básico experimentem e compreendam conceitos de *Machine Learning* de forma prática e interativa.

Outro ponto relevante foi a descentralização da computação, que ao utilizar uma *API* externa para a inferência dos modelos, permitiu a execução de modelos mais complexos e robustos sem sobrecarregar o dispositivo móvel, beneficiando-se da capacidade de processamento do servidor. A flexibilidade e escalabilidade proporcionadas pelo uso do *PythonAnywhere* também se destacaram, podendo facilitar a manutenção e atualização da solução.

No entanto, o desenvolvimento deste trabalho encontrou alguns desafios e limitações. A necessidade de uma conexão estável com a internet para a comunicação com a *API* pode ser uma limitação em ambientes com conectividade precária. Além disso, a versão gratuita da plataforma *PythonAnywhere* impõe restrições quanto ao processamento e armazenamento, o que pode limitar o uso contínuo e em larga escala da solução sem custos adicionais. O processo de configuração inicial, que inclui o upload do modelo e do script na plataforma *PythonAnywhere*, também pode ser um obstáculo para usuários com pouca experiência em ferramentas de desenvolvimento web.

Para futuras pesquisas e aprimoramentos, algumas abordagens podem ser exploradas. Melhorar a interface do *App Inventor* para tornar o processo de configuração e utilização da extensão ainda mais intuitivo e amigável para os usuários. Adicionalmente, expandir as capacidades da extensão para incluir outras tarefas de *Machine Learning*, como detecção de objetos e segmentação de imagens, pode ampliar significativamente o escopo de aplicação da ferramenta.

REFERÊNCIAS

ALPAYDIN, Ethem. Introduction to Machine Learning. 2nd ed. Cambridge, MA: MIT Press, 2010.

ALVES, N. da C. et al. Uma Proposta de Avaliação da Originalidade do Produto no Ensino de Algoritmos e Programação na Educação Básica. In: Anais do Simpósio Brasileiro de Informática na Educação, Natal, Brasil, 2020.

BHATIA, N. Using Transfer Learning, Spectrogram Audio Classification, and MIT App Inventor to Facilitate Machine Learning Understanding. Thesis: M. Eng., Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2020.

BISHOP, Christopher. Pattern Recognition and Machine Learning. 1. ed. Springer, 2006.

CARDOZO, J. Desenvolvimento de um Curso On-line para o Ensino de Machine Learning voltado à Classificação de Imagens no Ensino Médio. 2022. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Universidade Federal de Santa Catarina.

DAWS, R. Surprise! Machine learning jobs are high-paying and in-demand. Disponível em: https://artificialintelligence-news.com/2019/03/15/machine-learning-jobs-high-paying-demand/. Acesso em: 12 dez. 2022.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. ImageNet: A large-scale hierarchical image database. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2009. p. 248-255.

FRANZ, AUGUSTO CÉSAR MEDEIROS. Desenvolvimento de uma ferramenta visual de classificação de imagens para o ensino de machine learning no ensino médio. Trabalho de Conclusão de Curso. (Graduação em Sistemas de Informação) – Universidade Federal de Santa Catarina. 2021.

FREY, Carl Benedikt; OSBORNE, Michael A. The Future of Employment: How Susceptible Are Jobs to Computerization? Technological Forecasting and Social Change, v.114, 2017.

GOODFELLOW, Ian et al. Deep Learning. Cambridge, MA: MIT Press, 2016.

HADDAWAY, N. R. et al. The role of Google Scholar in evidence reviews and its applicability to grey literature searching. PloS one, 10(9), 2015.

HE, Kaiming et al. Deep Residual Learning for Image Recognition. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, NV, USA, 2015.

HOWARD, Andrew G. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. Disponível em: https://arxiv.org/abs/1704.04861. Acesso em: 22 mai. 2024.

KRIZHEVSKY, Alex et al. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems. vol25, 2012. p. 1097-1105.

LECUN, Y. et al. Deep learning. Nature, v. 521, n. 7553, p. 436-444, 2015.

LECUN, Yann et al. Gradient-based Learning Applied to Document Recognition. Proceedings of the IEEE, v. 86, n. 11, p. 2278-2324, 1998.

LEE, I. et al. Computational Thinking for Youth in Practice. ACM Inroads, v. 2, n. 1, p. 32-37, 2011.

LUGER, George F. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. 6th ed. Boston: Addison-Wesley, 2009.

LYE, S. Y., KOH, J. H. L. Review on teaching and learning of computational thinking through programming: What is next for K-12?. Computers in Human Behavior, 41, 51–61. 2014.

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.

MICROSOFT. ONNX Runtime. Disponível em: https://onnxruntime.ai/. Acesso em: 22 mai. 2024.

MICROSOFT. ONNX Runtime. Disponível em: https://github.com/microsoft/onnxruntime. Acesso em: 22 mai. 2024.

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

MIT. About Us. Disponível em: https://appinventor.mit.edu/. Acesso em: 20 mar. 2023.

MIT. App Inventor Documentation. Disponível em:https://appinventor.mit.edu/explore/ai2/tutorials. Acesso em: 20 mar. 2023.

MIT. App Inventor Extensions. Disponível em: https://appinventor.mit.edu/extensions/. Acesso em: 20 mar. 2023.

OLIVEIRA, Fabiano Pereira de. 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) - Curso de Sistemas de Informação, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2022.

ONNX (Open Neural Network Exchange). Disponível em: https://onnx.ai/. Acesso em: 19 fev. 2023.

ONNX. ResNet Models. Disponível em: https://github.com/onnx/models/tree/main/validated/vision/classification/resnet. Acesso em: 22 mai. 2024.

PATIL, P. How Machine Learning is Useful In Mobile App Development?

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.

PRESSMAN, R. S. Engenharia de Software: Uma Abordagem Profissional. 7ª Edição, AMGH Editora. 2011.

PYTORCH. Normalization. Disponível em: https://pytorch.org/docs/stable/torchvision/transforms.html#torchvision.transforms.Normalize. Acesso em: 22 mai. 2024.

PYTORCH. Transforms. Disponível em: https://pytorch.org/vision/stable/transforms.html. Acesso em: 22 mai. 2024.

PYTHONANYWHERE. PythonAnywhere: Host, run, and code Python in the cloud. Disponível em: https://www.pythonanywhere.com/. Acesso em: 10 mai. 2024.

RUSSELL, Stuart J.; NORVIG, Peter. Artificial Intelligence: A Modern Approach. Pearson Education, 2010.

SANGEETHA, S. V. T. et al. Currency Detection App for Blind People Using MIT App Inventor. In: 2022 International Conference on Power, Energy, Control and Transmission Systems , Chennai, India, 2022. p. 1-4.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014.

SZEGEDY, C. et al. Inception-v4, Inception-ResNet and the impact of residual connections on learning. In: AAAI Conference on Artificial Intelligence, Phoenix, Arizona, USA, p. 12-17, 2016.

SZELISKI, R. Computer vision: algorithms and applications. 2. ed. London: Springer-Verlag London Limited: Springer Science & Business Media, 2010.

TANG, D. et al. PIC: A Personal Image Classification Webtool for High School Students. In: Proc. of the IJCAI EduAI Workshop, Macao, China, 2019.

TOURETZKY, D. S. et al. K-12 Guidelines for Artificial Intelligence: What Students Should Know. In: Proc. of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 2020.

VIKRAMAN, B. P. et al. Date fruit classification and sorting system using Artificial Intelligence: Application of Transfer Learning. Computers and Electronics in Agriculture, v. 176, 105630, 2020.

WOLBER, D., ABELSON, H., FRIEDMAN, M. Democratizing Computing with App Inventor. GetMobile: Mobile Computing and Communications, v. 18, n. 4, p. 53-58, 2014.

WALKARN, U. A Quick Introduction to Neural Networks. Disponível em: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/. Acesso em: 20 set. 2022.

ZHU, Jessica. Creating Your Own Conversational Artificial Intelligence Agents Using Convo, a Conversational Programming System. M.Eng. thesis, Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, 2021.

Apêndice A – Artigo Monografia

ONNX-ICE: Uma Extensão do App Inventor para Implantação de Modelos de ML para Classificação de Imagens

Jefferson A. Bastos Junior[1], Christiane Gresse Von Wangenheim[1]

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

jefferson.bastos@grad.ufsc.br, c.wangenheim@ufsc.br

Abstract. *This work develops an integrated solution for image classification within the App Inventor environment, using Machine Learning models trained in ResNet 18 or 34 and exported in ONNX format. Through the creation of the ONNX Image Classifier Extension (ONNX-ICE) and the implementation of an API hosted on the PythonAnywhere platform, we provide a practical and efficient tool for students and mobile app developers. The solution facilitated the integration of ONNX models with App Inventor, simplifying the image classification process.*

Keywords: *Machine Learning, Image Classification, App Inventor, ONNX, PythonAnywhere.*

Resumo. *O presente trabalho desenvolve uma solução integrada para a classificação de imagens no ambiente App Inventor, utilizando modelos de Machine Learning treinados em ResNet 18 ou 34 e exportados no formato ONNX. Através da criação da extensão ONNX Image Classifier Extension (ONNX-ICE) e da implementação de uma API hospedada na plataforma PythonAnywhere, proporcionamos uma ferramenta prática e eficiente para alunos e desenvolvedores de aplicativos móveis. A solução facilitou a integração de modelos ONNX com o App Inventor, simplificando o processo de classificação de imagens.*

Palavras-chave: *Machine Learning, Classificação de Imagens, App Inventor, ONNX, PythonAnywhere.*

1. Introdução

A inteligência artificial (IA) está cada vez mais presente em nosso cotidiano, transformando diversos setores através da automação e análise de grandes volumes de dados (Russell e Norvig, 2010). Dentro desse contexto, o aprendizado de máquina (ML) emerge como uma sub-área da IA, permitindo que sistemas aprendam e melhorem a partir de experiências (Mitchell, 1997). O aprendizado de máquina utiliza redes neurais artificiais para realizar tarefas complexas como reconhecimento de imagens, diagnósticos médicos, e direção autônoma de veículos (Goodfellow et al., 2016). Para o ensino desses conceitos na educação básica, ferramentas como o App Inventor se mostram extremamente eficazes, proporcionando um ambiente acessível para a criação de aplicativos (MIT, 2022). Este trabalho visa integrar modelos de ML no App Inventor,

através da extensão ONNX-ICE, para facilitar a classificação de imagens, tornando o processo mais acessível para alunos e desenvolvedores.

2. Metodologia

A seção de metodologia detalha como a solução foi desenvolvida, incluindo arquitetura, desenvolvimento da extensão ONNX-ICE, e a implementação da API no PythonAnywhere.

2.1. Arquitetura da Solução

A arquitetura da solução ONNX-ICE é composta por vários componentes que interagem para possibilitar a classificação de imagens utilizando modelos de ResNet treinados e exportados em ONNX. Inicialmente, a intenção da extensão era suportar a importação de modelos ResNet no formato ONNX e possibilitar que, de forma offline, o aplicativo pudesse fazer a inferência a partir do modelo. No entanto, surgiram problemas de compatibilidade de versão do Android e do Java utilizados no App Inventor em relação à biblioteca ONNX Runtime.

Para superar esses desafios, a solução foi estruturada para realizar a inferência em um servidor remoto hospedado na plataforma PythonAnywhere. Isso permite que o processamento dos modelos ONNX ocorra no servidor, e os resultados sejam retornados para a extensão no App Inventor.

Fluxo de Dados e Interações:

1 - Treinar o Modelo de Classificação de Imagens:

Os alunos treinam modelos de classificação de imagens (ResNet18 ou ResNet34) na plataforma Jupyter Notebook e exportam o modelo treinado no formato ONNX.

2 - Upload do Modelo e Script:

O aluno se cadastra na plataforma PythonAnywhere, importa o modelo treinado e exportado, as classes e o script de inferência.

3 - Configuração da Extensão:

No App Inventor, os alunos incluem a extensão ONNX-ICE e configuram as propriedades da extensão com as propriedades necessárias (nome de usuário cadastrado na plataforma PythonAnywhere, nome do modelo carregado, nome do arquivo de classes).

4 - Implementação do App:

Os alunos desenvolvem o aplicativo no App Inventor utilizando a extensão ONNX-ICE. Configuram os blocos visuais para capturar imagens e enviá-las para a API.

5 - Envio de Imagens para Classificação:

O aplicativo pode usar a câmera ou imagens salvas no dispositivo para enviar para a API. A extensão ONNX-ICE lida com o envio dessas imagens para o servidor.

6 - Visualização dos Resultados:

Os resultados da classificação são recebidos pelo aplicativo e exibidos. O aluno pode decidir como visualizar esses resultados, utilizando-os de diversas maneiras dentro do aplicativo.

2.2. Desenvolvimento da Extensão ONNX-ICE

A extensão ONNX-ICE foi desenvolvida utilizando o framework do App Inventor para a criação de extensões. Ela se concentra na comunicação com uma API hospedada na plataforma PythonAnywhere, onde a inferência dos modelos ONNX é realizada.

Propriedades da Extensão:

ModelName:

Define o nome do modelo carregado na plataforma PythonAnywhere.

ClassesName:

Define o nome do arquivo de classes carregado na plataforma PythonAnywhere.

UserName:

Define o nome de usuário registrado na plataforma PythonAnywhere.

Funções da Extensão:

1 - classifyImage:

Envia uma imagem para classificação via API. Recebe a imagem que deve ser classificada (string).

2 - OpenGallery:

Abre a galeria de fotos do dispositivo para selecionar uma imagem e envia para classificação.

Eventos da Extensão:

1 - allClassesEvent:

Disparado quando a resposta da API é retornada, contendo todas as classes com os respectivos valores de confiança.

2 - imageEvent:

Disparado quando uma imagem é carregada da galeria.

3 - errorEvent:

Disparado quando a extensão captura um erro.

4 - predictedClassEvent:

Disparado quando a resposta da API é retornada, contendo a classe prevista (com maior probabilidade).

5 - probabilityEvent:

Disparado quando a resposta da API é retornada, contendo a probabilidade da classe prevista.

2.3. Implementação da API no PythonAnywhere

A implementação da API no PythonAnywhere é um componente essencial deste projeto, pois permite a execução dos modelos de classificação de imagens treinados e exportados no formato ONNX. PythonAnywhere é uma plataforma de hospedagem baseada em nuvem que oferece um ambiente controlado para a execução de scripts Python, sendo uma escolha adequada para hospedar a API necessária para este trabalho.

O primeiro passo para configurar a API é criar uma conta no PythonAnywhere. Os alunos ou desenvolvedores devem acessar o site da plataforma, registrar-se e criar uma conta. Esse processo é simples e rápido, e a plataforma oferece um plano gratuito que é suficiente para os propósitos deste projeto.

Após a criação da conta, é necessário configurar o ambiente para suportar a execução da API. No painel de controle do PythonAnywhere, o usuário deve acessar a seção "Web" e adicionar um novo aplicativo web. O framework escolhido para a criação da API é o Flask, que é uma ferramenta leve e eficiente para o desenvolvimento de aplicações web em Python. Na configuração do novo aplicativo web, deve-se selecionar o Flask como framework e a versão do Python a ser utilizada, que é a versão 3.10.

Com o ambiente configurado, o próximo passo é fazer o upload dos arquivos necessários. Isso inclui o modelo ONNX treinado, o arquivo de classes e o script de inferência. Esses arquivos devem ser carregados na pasta "mysite" do PythonAnywhere. O modelo ONNX contém a rede neural treinada, o arquivo de classes inclui os índices necessários para identificar corretamente as classes previstas, e o script de inferência é responsável por processar as imagens e retornar as previsões.

Após o upload dos arquivos, é preciso instalar as bibliotecas necessárias para a execução da API. No console Bash do PythonAnywhere, o comando “pip3.10 install onnxruntime pillow torchvision” é utilizado para instalar as bibliotecas onnxruntime, pillow e torchvision. Essas bibliotecas são essenciais para a manipulação de imagens e a execução dos modelos ONNX.

A API é então configurada para receber imagens, processá-las e retornar as classificações. A rota principal “/classifyImage” é configurada para aceitar solicitações POST contendo as imagens a serem classificadas. O script de inferência no Python utiliza a biblioteca onnxruntime para carregar o modelo ONNX e realizar a inferência. As imagens enviadas pelo aplicativo são processadas, e a API retorna um JSON contendo a classe prevista e a probabilidade associada.

Para verificar se a API está funcionando corretamente, pode-se acessar a URL fornecida pelo PythonAnywhere. Se a mensagem de confirmação for exibida, isso indica que a configuração foi bem-sucedida e a API está pronta para receber solicitações de classificação de imagens.

Esta configuração permite que o processamento dos modelos ONNX ocorra no servidor, beneficiando-se da capacidade de processamento do PythonAnywhere. Com isso, os dispositivos móveis não são sobrecarregados com a tarefa de inferência, o que é especialmente útil ao lidar com modelos mais complexos e robustos. Além disso, a centralização do processamento no servidor facilita a manutenção e atualização dos scripts de inferência, que podem ser modificados diretamente no servidor sem a necessidade de atualizações na extensão do App Inventor.

3. Teste com o Aplicativo ZooScan

Para validar o funcionamento da extensão ONNX-ICE, foi criado o aplicativo ZooScan, que classifica imagens de animais utilizando modelos de aprendizado de máquina. O ZooScan é baseado em modelos ResNet-18 e ResNet-34 treinados e exportados no formato ONNX, utilizando o dataset ImageNet. Esse dataset contém mil classes diferentes, abrangendo uma ampla gama de objetos, desde itens do cotidiano até diversas espécies de animais (Deng et al., 2009).

O aplicativo ZooScan possui duas telas principais: a tela de apresentação e a tela de classificação. A tela de apresentação fornece informações sobre o aplicativo e um botão que direciona o usuário para a tela de classificação. A tela de classificação permite que o usuário tire uma foto ou selecione uma imagem da galeria para classificação. A extensão ONNX-ICE processa a imagem enviando-a para a API configurada no PythonAnywhere, que retorna a classe prevista e a probabilidade associada.

4. Material Didático

Visando a continuidade do curso ML4TEENS (Cardozo, 2022), foi desenvolvido um tutorial para apoiar professores e estudantes na utilização da extensão ONNX-ICE. O material didático inclui um tutorial em formato de slides que apresenta passo a passo como usar a extensão no App Inventor, o projeto do aplicativo ZooScan já completo, e um wireframe do aplicativo para que os alunos possam seguir os passos e criar o aplicativo de forma independente.

A primeira parte da aula consiste em fornecer ao aluno um modelo previamente treinado, acompanhado pelo respectivo arquivo de classes. Este modelo é o mesmo utilizado no teste demonstrado anteriormente. Em seguida, é abordada a configuração da API no PythonAnywhere, desde a criação da conta até o upload dos arquivos necessários. Por fim, o tutorial ensina o aluno a desenvolver o aplicativo ZooScan no App Inventor, utilizando a extensão ONNX-ICE, detalhando seus blocos e funções, e a testar o aplicativo desenvolvido.

5. Conclusão

O presente trabalho teve como objetivo principal desenvolver uma solução integrada para a classificação de imagens no ambiente App Inventor, utilizando modelos de Machine Learning treinados e exportados no formato ONNX. Através da criação da extensão ONNX-ICE e da implementação de uma API hospedada na plataforma PythonAnywhere, foi possível proporcionar uma ferramenta prática e eficiente para alunos e desenvolvedores de aplicativos móveis.

A solução proposta apresentou diversas contribuições significativas. Primeiramente, a extensão ONNX-ICE facilitou a integração de modelos ONNX com o App Inventor, simplificando o processo de classificação de imagens sem exigir conhecimentos avançados em programação ou infraestrutura. Além disso, a ferramenta desenvolvida mostrou-se potencialmente útil em ambientes educacionais, permitindo que alunos do ensino básico experimentem e compreendam conceitos de Machine Learning de forma prática e interativa.

Outro ponto relevante foi a descentralização da computação, que ao utilizar uma API externa para a inferência dos modelos, permitiu a execução de modelos mais complexos e robustos sem sobrecarregar o dispositivo móvel, beneficiando-se da capacidade de processamento do servidor. A flexibilidade e escalabilidade proporcionadas pelo uso do PythonAnywhere também se destacaram, podendo facilitar a manutenção e atualização da solução.

No entanto, o desenvolvimento deste trabalho encontrou alguns desafios e limitações. A necessidade de uma conexão estável com a internet para a comunicação com a API pode ser uma limitação em ambientes com conectividade precária. Além disso, a versão gratuita da plataforma PythonAnywhere impõe restrições quanto ao processamento e armazenamento, o que pode limitar o uso contínuo e em larga escala da solução sem custos adicionais. O processo de configuração inicial, que inclui o upload do modelo e do script na plataforma PythonAnywhere, também pode ser um obstáculo para usuários com pouca experiência em ferramentas de desenvolvimento web.

Para futuras pesquisas e aprimoramentos, algumas abordagens podem ser exploradas. Melhorar a interface do App Inventor para tornar o processo de configuração e utilização da extensão ainda mais intuitivo e amigável para os usuários. Adicionalmente, expandir as capacidades da extensão para incluir outras tarefas de Machine Learning, como detecção de objetos e segmentação de imagens, pode ampliar significativamente o escopo de aplicação da ferramenta.

6. References

CARDOZO, J. Curso ML4TEENS, 2022.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09, 2009.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Deep Learning. MIT Press, 2016.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep Residual Learning for Image Recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

MIT. App Inventor. Disponível em: https://appinventor.mit.edu. Acesso em: 22 mai. 2024.

MITCHELL, T. Machine Learning. McGraw-Hill, 1997.

RUSSELL, S.; NORVIG, P. Artificial Intelligence: A Modern Approach. 3rd ed. Upper Saddle River: Prentice Hall, 2010.