



UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CENTRO TECNOLÓGICO  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA  
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Rodrigo de Mello Garcia

**EVOLUÇÃO DO DESENVOLVIMENTO DE UMA EXTENSÃO PARA A  
IMPLANTAÇÃO DE MODELOS DE MACHINE LEARNING EM APLICATIVOS COM  
O APP INVENTOR**

Florianópolis  
2022

Rodrigo de Mello Garcia

**EVOLUÇÃO DO DESENVOLVIMENTO DE UMA EXTENSÃO PARA A  
IMPLANTAÇÃO DE MODELOS DE MACHINE LEARNING EM APLICATIVOS COM  
O APP INVENTOR**

Trabalho de Conclusão do Curso do Curso de Graduação em Sistemas de Informação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Sistemas de Informação.

Orientadora: Christiane Gresse von Wangenheim, Dra.

Florianópolis  
2022

Rodrigo de Mello Garcia

**EVOLUÇÃO DO DESENVOLVIMENTO DE UMA EXTENSÃO PARA A  
IMPLANTAÇÃO DE MODELOS DE MACHINE LEARNING EM APLICATIVOS COM  
O APP INVENTOR**

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel pelo Curso de Graduação em Sistemas de Informação

Florianópolis, junho de 2023.

Prof. Álvaro Junio Pereira Franco, Dr.  
Coordenador do Curso

**Banca Examinadora:**

Prof<sup>a</sup>. Christiane Gresse von Wangenheim, Dra.  
Orientadora  
UFSC

Prof. Jean C. R. Hauck, Dr.  
Coorientador  
UFSC

Prof. Ramon Mayor Martins, M.Sc.  
Avaliador  
IFSC

## RESUMO

A Inteligência Artificial (IA) é uma área da computação cada vez mais presente no cotidiano das pessoas, prometendo revolucionar diversos setores da sociedade. Neste contexto, surge também a necessidade de difundir e popularizar conhecimentos nessa área, tanto para conscientização da população quanto para a exploração do potencial da IA para a sociedade, formando novos profissionais capazes de trabalhar com esta tecnologia. Existem diversas iniciativas que visam difundir esse conhecimento entre os alunos do ensino fundamental e médio do Brasil, uma delas sendo a iniciativa Computação na Escola, da Universidade Federal de Santa Catarina (UFSC), ensinando conceitos de *Machine Learning*, algoritmos de aprendizagem e fundamentos de redes neurais para estudantes entre 12 e 18 anos. Uma das ferramentas utilizadas neste curso é o *Teachable Machine Image Classifier* (TMIC), uma extensão para a plataforma de desenvolvimento de aplicativos móveis MIT App Inventor que permite a desenvolvedores importar modelos de classificação de imagem treinados no *framework* Google Teachable Machine. Embora, atualmente, a extensão seja a única capaz de fazer isso, mostrando sua relevância, algumas limitações ainda presentes, como a dependência de conexão à internet ou a impossibilidade de classificar imagens obtidas previamente, limitam as situações em que o TMIC pode ser utilizado. Assim, o presente trabalho visa evoluir a extensão TMIC, reduzindo estas limitações e ampliando a possibilidade do uso da mesma em mais cenários. Como resultados, foram desenvolvidas uma nova versão da extensão TMIC, além de novos materiais didáticos para a iniciativa Computação na Escola, ensinando alunos a usarem as novas funcionalidades da extensão resultantes deste trabalho.

**Palavras-chave:** *Machine Learning. Google Teachable Machine. MIT App Inventor. Extensão.*

## LISTA DE FIGURAS

Figura 1 – Representação do neurônio biológico proposto por McCulloch e Pitts.	18
Figura 2 – Classe de um projeto de classificação de imagens do Google Teachable Machine alimentada com imagens de objetos de papel. . . .	20
Figura 3 – Componente para teste do modelo de classificação de imagens treinado no Google Teachable Machine. . . . .	21
Figura 4 – Componente para teste do modelo de classificação de imagens treinado no Google Teachable Machine. . . . .	22
Figura 5 – Exemplo de tela construída com componentes do App Inventor Designer. . . . .	24
Figura 6 – Exemplo de instrução montada com blocos lógicos de componentes adicionados pelo do App Inventor Designer. . . . .	25
Figura 7 – Importação de arquivos de extensão aix para o projeto do App Inventor.	26
Figura 8 – Bloco lógico da extensão ML4K AppInventor Extension. . . . .	33
Figura 9 – Instruções para importação da extensão ML4K AppInventor Extension para o MIT App Inventor com a API Key configurada. . . . .	34
Figura 10 – Blocos lógicos da extensão <i>Personal Image Classifier</i> (PIC), à esquerda, e TMIC, à direita. . . . .	36
Figura 11 – Arquitetura da extensão original TMIC . . . . .	40
Figura 12 – Arquitetura da evolução da extensão . . . . .	42
Figura 13 – Blocos lógicos da extensão TMIC. . . . .	43
Figura 14 – Propriedades da evolução da extensão TMIC, destacando a nova propriedade <i>UseFrontalCamera</i> . . . . .	44
Figura 15 – Blocos lógicos da evolução da extensão TMIC. . . . .	44
Figura 16 – Passos para alterar o <i>Path</i> nas variáveis de ambiente do Windows. .	46
Figura 17 – <i>Logcat</i> mostrando mensagens do código com a classe <i>Log</i> . . . . .	47
Figura 18 – Função <i>UrlModel</i> na classe <i>TeachableMachinelImageClassifier.java</i>	48
Figura 19 – Painel para <i>download</i> dos arquivos do modelo de classificação de imagem, disponível na página do Google Teachable Machine do modelo. . . . .	49
Figura 20 – Função <i>readModelFile</i> da classe <i>JsonObject</i> . . . . .	50
Figura 21 – Função <i>loadFromFiles</i> carregando o modelo a partir de arquivos criados. . . . .	50
Figura 22 – Copiando os arquivos locais do armazenamento interno para o diretório específico do aplicativo com o <i>ASDDDownloader</i> . . . . .	51
Figura 23 – Nodo de decisão do método de carregamento do modelo de classificação de imagem. . . . .	52

Figura 24 – Bloco <i>SetLocalFilePath</i> recebendo como parâmetro o bloco <i>GetAI-CompanionLocalFilePath</i> . . . . .	52
Figura 25 – Criando uma imagem para passar para a função <i>model.predict</i> . . . . .	53
Figura 26 – Função <i>readImageFiles</i> . . . . .	53
Figura 27 – Blocos lógicos do <i>ImagePicker</i> e <i>classifyImageData</i> . . . . .	54
Figura 28 – Atribuição da variável <i>isClassifyingFromGallery</i> . . . . .	54
Figura 29 – Nodo de decisão da imagem a ser classificada. . . . .	55
Figura 30 – Propriedade <i>UseFrontalCamera</i> . . . . .	55
Figura 31 – configuração da câmera do dispositivo. . . . .	56
Figura 32 – Slide do tutorial atualizado para mencionar a nova propriedade <i>UseFrontalCamera</i> . . . . .	58
Figura 33 – Slide do tutorial ensinando a importar arquivos para o projeto pelo painel Mídia . . . . .	58
Figura 34 – Slide do tutorial ensinando a classificar a imagem escolhida da galeria	59
Figura 35 – Telas do aplicativo de classificação de lixo (primeira tela à esquerda, segunda tela à direita) . . . . .	60
Figura 36 – Arquivos do modelo de classificação de imagem carregados no painel Mídia do MIT App Inventor. . . . .	61
Figura 37 – Blocos lógicos para a classificação de imagens da galeria do dispositivo	62
Figura 38 – Telas do aplicativo exemplo mostrando o fluxo de classificação de imagens da galeria . . . . .	62
Figura 39 – Seleção da propriedade <i>UseFrontalCamera</i> da extensão TMIC . . . . .	63
Figura 40 – Fluxograma do aplicativo exemplo . . . . .	63

## LISTA DE QUADROS

Quadro 1 – Classificação das definições de IA . . . . .	16
Quadro 2 – Fontes de busca e <i>strings</i> de busca correspondentes. . . . .	28
Quadro 3 – Extensões existentes. . . . .	30
Quadro 4 – Ambientes de treinamento e formato de exportação dos modelos de classificação de imagens para as extensões encontradas. . . . .	31
Quadro 5 – Métodos de importação do modelo de classificação de imagem de cada extensão. . . . .	32
Quadro 6 – Requisitos funcionais. . . . .	39
Quadro 7 – Requisitos não funcionais. . . . .	39
Quadro 8 – Blocos e propriedades da extensão TMIC. . . . .	43
Quadro 9 – Novos blocos e propriedades da extensão TMIC. . . . .	45
Quadro 10 – Variáveis de ambiente. . . . .	46
Quadro 11 – Componentes do material didático desenvolvido . . . . .	57

## LISTA DE TABELAS

Tabela 1 – Formatos de exportação disponíveis pelo Google Teachable Machine e o tamanho de cada arquivo resultante. . . . .	23
Tabela 2 – Resultados da execução da busca. . . . .	29
Tabela 3 – Fontes de busca e <i>strings</i> de busca correspondentes. . . . .	35



## LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência Artificial
PIC	<i>Personal Image Classifier</i>
TMIC	<i>Teachable Machine Image Classifier</i>

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>11</b>
1.1	OBJETIVOS	12
1.2	METODOLOGIA DE PESQUISA	13
1.3	ESTRUTURA DO DOCUMENTO	14
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>16</b>
2.1	INTELIGÊNCIA ARTIFICIAL, MACHINE LEARNING, REDES NEURAIS ARTIFICIAIS E <i>DEEP LEARNING</i>	16
<b>2.1.1</b>	<b>Inteligência Artificial</b>	<b>16</b>
<b>2.1.2</b>	<b><i>Machine Learning, Redes Neurais e Deep Learning</i></b>	<b>17</b>
2.2	CLASSIFICAÇÃO DE IMAGENS COM <i>DEEP LEARNING</i> USANDO O GOOGLE TEACHABLE MACHINE	18
2.3	MIT APP INVENTOR	23
<b>2.3.1</b>	<b>Criação de extensões para o App Inventor</b>	<b>25</b>
<b>3</b>	<b>ESTADO DA ARTE</b>	<b>27</b>
3.1	DEFINIÇÃO DO PROTOCOLO DE REVISÃO	27
3.2	EXECUÇÃO DA BUSCA	28
3.3	RESULTADOS DA REVISÃO	30
<b>3.3.1</b>	<b>Quais extensões existem?</b>	<b>30</b>
<b>3.3.2</b>	<b>Em quais ambientes de desenvolvimento os modelos de <i>Machine Learning</i> são treinados e exportados?</b>	<b>30</b>
<b>3.3.3</b>	<b>Como os modelos treinados são importados nas extensões?</b>	<b>32</b>
<b>3.3.4</b>	<b>Quais são as características funcionais dessas extensões?</b>	<b>34</b>
3.4	DISCUSSÃO	36
<b>4</b>	<b>DESENVOLVIMENTO</b>	<b>38</b>
4.1	ANÁLISE DE CONTEXTO E DEFINIÇÃO	38
4.2	ANÁLISE DE REQUISITOS	38
<b>4.2.1</b>	<b>Requisitos funcionais</b>	<b>38</b>
<b>4.2.2</b>	<b>Requisitos não funcionais</b>	<b>39</b>
4.3	ARQUITETURA DA EXTENSÃO TMIC E SUA EVOLUÇÃO	40
4.4	PROPRIEDADES E BLOCOS LÓGICOS DA EVOLUÇÃO DA EXTENSÃO	42
4.5	PREPARAÇÃO E INSTALAÇÃO DO AMBIENTE DE DESENVOLVIMENTO	45
4.6	DESENVOLVIMENTO DA EVOLUÇÃO DA EXTENSÃO	47
<b>4.6.1</b>	<b>Uso de arquivos locais para carregamento do modelo de classificação</b>	<b>47</b>
<b>4.6.2</b>	<b>Classificação de arquivos da galeria do dispositivo</b>	<b>52</b>

4.6.3	Uso da câmera frontal do dispositivo . . . . .	55
4.6.4	Testes da evolução . . . . .	56
5	<b>DESENVOLVIMENTO DO MATERIAL DIDÁTICO E DO APLICATIVO</b>	
	<b>EXEMPLO . . . . .</b>	<b>57</b>
5.1	DESENVOLVIMENTO DO MATERIAL DIDÁTICO . . . . .	57
5.2	DESENVOLVIMENTO DO APLICATIVO EXEMPLO . . . . .	59
5.2.1	Uso de arquivos locais para carregamento do modelo de classificação . . . . .	60
5.2.2	Classificação de arquivos da galeria do dispositivo . . . . .	61
5.2.3	Uso da câmera frontal do dispositivo . . . . .	63
6	<b>CONCLUSÃO . . . . .</b>	<b>65</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>66</b>
	<b>APÊNDICE A – CÓDIGO FONTE . . . . .</b>	<b>71</b>
	<b>APÊNDICE B – ARTIGO DA MONOGRAFIA . . . . .</b>	<b>72</b>

## 1 INTRODUÇÃO

*Machine Learning* é uma área de conhecimento em que se busca ensinar a uma máquina o reconhecimento de padrões por meio da apresentação de modelos exemplos, permitindo à máquina gerar conhecimento sobre estes mais rapidamente do que implementando manualmente regras de conhecimento (JANIESCH; ZSCHECH; HEINRICH, 2021).

Dentre as principais aplicações do *Machine Learning* está a classificação de imagens, que consiste em alimentar uma máquina com entradas de imagens rotuladas, ensinando-a e tornando-a capaz de classificar imagens desconhecidas e ainda não rotuladas baseando-se em seus aprendizados. A classificação de imagens pode ser usada em diversas áreas, como na medicina, na classificação de imagens dermatoscópicas para a detecção de melanoma (SANTOS, 2020), ou na agrimensura, na classificação de telhados para planejamento e cadastro urbano (CHAMMA *et al.*, 2021); entre outras. Assim, *Machine Learning* é uma área de conhecimento já presente e atuante no cotidiano da sociedade (LAWRENCE, 2019).

O impacto das áreas de IA e *Machine Learning* resulta também em reações do mercado de trabalho de tecnologia da informação e, conseqüentemente, nos sistemas de educação internacionais, que buscam formar profissionais para suprir as demandas desse mercado. Diante disso, países como a China já anunciaram programas de ensino de IA para estudantes do ensino médio, buscando ser a liderança nesta área até o ano de 2030 (JING, 2018).

Embora o ensino sobre IA e *Machine Learning* nas instituições de ensino superior seja presente, ele também se mostra necessário na educação básica. No Brasil, a computação passou a ser tratada como um tema escolar a partir de 2010. O currículo proposto pela Sociedade Brasileira de Computação prevê, na educação de nível médio, o ensino de conceitos como *Machine Learning* e IA apenas com caráter introdutório, evitando o aprofundamento e contrariando estratégias tomadas pelos demais países. Assim, *Machine Learning* ainda não é um tópico abordado de forma geral no ensino médio no Brasil (GRESSE VON WANGENHEIM; MARQUES; HAUCK, 2020).

Diante desta demanda, estão surgindo esforços para trazer esses e outros tópicos de computação para jovens em seus últimos anos de estudos antes da faculdade, como é o caso dos cursos de *Machine Learning* da iniciativa Computação na Escola, do INCoD/INE/UFSC, que visa ensinar conceitos básicos de *Machine Learning* e suas aplicações, guiando o aluno à criação e implantação de modelos de reconhecimento de imagens no MIT App Inventor.

Nesse estágio educacional, tipicamente ferramentas visuais são utilizadas para ensinar o desenvolvimento de modelos de *Machine Learning* (GRESSE VON WANGENHEIM *et al.*, 2021). Uma destas ferramentas é o Google Teachable Machine, uma

plataforma web para o ensino de *Machine Learning* e *Deep Learning* por meio da criação de modelos de classificação de imagens. Em 2020, o Google Teachable Machine foi utilizado em mais de duzentos países, gerando mais de 125 mil modelos de classificação. Assim, o Google Teachable Machine se mostra uma ferramenta popular, proporcionando uma oportunidade para qualquer interessado criar seu próprio modelo de classificação sem nenhum conhecimento aprofundado de *Machine Learning* ou de programação (CARNEY et al., 2020).

Outro exemplo de ambiente visual baseado em blocos é o App Inventor, uma plataforma online que ensina os conceitos básicos de programação de modo intuitivo e motivador, permitindo aos seus usuários programarem aplicativos para dispositivos móveis (FINIZOLA et al., 2014). Nesse contexto, o App Inventor também permite a criação de extensões que podem posteriormente serem implementadas nos aplicativos desenvolvidos na plataforma. Na área de classificação de imagens com modelos de *Machine Learning*, exemplos de extensões para o MIT App Inventor são o PIC, uma extensão que permite aos usuários treinarem modelos de classificação de imagens por meio de uma plataforma própria e usá-los em aplicativos criados no App Inventor (TANG; UTSUMI; LAO, 2019), e o TMIC, que permite o uso de modelos treinados no Google Teachable Machine hospedados na nuvem (OLIVEIRA, 2022).

Em relação ao TMIC, identificou-se a necessidade de melhorias desta extensão, apresentadas por Oliveira (2022) como sugestões de trabalhos futuros, como o suporte para importação de modelos de *Machine Learning* exportados do Google Teachable Machine em outros formatos, acesso à câmera frontal para capturar imagens, entre outros. Vê-se, portanto, a oportunidade de implementar uma evolução desta extensão que abranja mais possibilidades de uso da mesma, removendo suas limitações atuais e implementando as sugestões de trabalhos futuros do autor.

## 1.1 OBJETIVOS

### **Objetivo Geral**

O objetivo geral deste trabalho é criar uma nova versão da extensão TMIC com as funcionalidades adicionais de carregar modelos de classificação de imagens a partir de arquivos locais do modelo de *Machine Learning* treinado, classificar imagens da galeria do dispositivo e utilizar a câmera frontal do dispositivo. Além disso, o trabalho também visa desenvolver um material didático para ensinar o uso da nova versão do TMIC desenvolvida. Assim, espera-se aprimorar a extensão existente, expandindo suas capacidades e fornecendo recursos adicionais que permitam aos usuários uma maior flexibilidade e interação com a tecnologia de classificação de imagens utilizando o MIT App Inventor. O material didático desenvolvido busca fornecer um guia claro para auxiliar os usuários no uso efetivo da nova versão do TMIC, facilitando assim o

processo de aprendizado e aplicação prática dessa tecnologia.

### **Objetivos Específicos**

- O1. Elaborar a fundamentação teórica sobre classificação de imagens com *Machine Learning* por meio da plataforma Google Teachable Machine, exportação de modelos treinados, e a criação de extensões no App Inventor.
- O2. Levantar o estado da arte em relação a extensões do App Inventor que permitam a implementação de modelos de classificação de imagem.
- O3. Desenvolver uma versão do TMIC capaz de construir modelos de classificação de imagens a partir de arquivos locais do modelo de *Machine Learning* treinado, classificar imagens da galeria e utilizar a câmera frontal do dispositivo para classificar imagens.
- O4. Desenvolver material didático na forma de slides para ensinar o uso da extensão desenvolvida, acompanhado de um projeto no MIT App Inventor com exemplos de aplicação das novas funcionalidades (aplicativo exemplo).

## 1.2 METODOLOGIA DE PESQUISA

O presente trabalho adotou diferentes metodologias de pesquisa para alcançar cada um dos objetivos expostos na Seção 1.1.

### **Etapa 1 - Fundamentação teórica**

Por meio de uma análise e síntese da literatura, são apresentados conceitos fundamentais de IA, *Machine Learning*, classificação de imagens por meio do Google Teachable Machine, exportação de modelos treinados e a criação de extensões para o MIT App Inventor.

- A1.1. Sintetizar conceitos IA, *Machine Learning*, Redes Neurais e *Deep Learning*;
- A1.2. Sintetizar conceitos de classificação de imagens e exportação de modelos treinados com o Google Teachable Machine;
- A1.3. Sintetizar conceitos do MIT App Inventor e a criação de extensões para o mesmo.

### **Etapa 2 - Levantamento do estado da arte em relação a extensões do App Inventor semelhantes**

Para levantar o estado da arte, foi realizada uma revisão sistemática por meio do processo proposto por Petersen et al. (2008) sobre trabalhos existentes relacionados

a extensões para o MIT App Inventor para a implantação de modelos de *Machine Learning*, identificando e analisando todos os resultados considerados relevantes de acordo com os critérios de aceitação.

A2.1. Definir o protocolo de busca;

A2.2. Executar a busca;

A2.3. Extrair e analisar as informações.

### **Etapa 3 - Desenvolvimento da evolução da extensão TMIC**

A evolução da extensão foi desenvolvida seguindo um processo de desenvolvimento de software que engloba a análise dos requisitos, modelagem, implementação e documentação, com o objetivo de aprimorar a qualidade do software em cada etapa.

A3.1. Análise de requisitos: levantamento dos requisitos funcionais e não-funcionais da extensão desenvolvida;

A3.2. Modelagem da arquitetura do sistema: análise e definição da arquitetura da extensão desenvolvida;

A3.3. Implementação: implementar a evolução da extensão desenvolvida, contemplando os requisitos definidos;

A3.4. Documentação: detalhar o desenvolvimento da extensão desenvolvida.

### **Etapa 4 – Desenvolvimento do material didático**

Com o objetivo de atualizar o conteúdo do curso para incluir as novas funcionalidades desenvolvidas neste trabalho, desenvolveu-se um material didático na forma de apresentação de slides e um aplicativo exemplo.

A4.1. Desenvolvimento dos slides do tutorial.

A4.2. Desenvolvimento do aplicativo exemplo.

## **1.3 ESTRUTURA DO DOCUMENTO**

No Capítulo 2 deste trabalho é apresentada a fundamentação teórica, com uma síntese dos conceitos de IA (incluindo *Machine Learning*, redes neurais e *Deep Learning*), Google Teachable Machine e MIT App Inventor. No Capítulo 3, é feito o levantamento o estado da arte das extensões para MIT App Inventor que permitem a implementação de modelos de classificação de imagens utilizando *Machine Learning*. No Capítulo 4 é detalhado todo o processo de desenvolvimento da evolução da

extensão TMIC, desde o levantamento dos requisitos funcionais e não funcionais até o a documentação do código das funcionalidades implementadas. No Capítulo 5, é documentada a atualização da apresentação de slides para incluir as novas funcionalidades da extensão, bem como a criação do aplicativo exemplo. Na conclusão, no Capítulo 6, apresenta-se os objetivos alcançados neste trabalho, bem como o impacto de seu resultado e sugestão de trabalhos futuros.



## 2 FUNDAMENTAÇÃO TEÓRICA

### 2.1 INTELIGÊNCIA ARTIFICIAL, MACHINE LEARNING, REDES NEURAIS ARTIFICIAIS E *DEEP LEARNING*

#### 2.1.1 Inteligência Artificial

O objetivo dos estudos de IA é de estudar e, eventualmente, imitar e reproduzir a habilidade humana de tomada de decisão, automatizando assim processos complexos onde essa habilidade é requisitada (JANIESCH; ZSCHECH; HEINRICH, 2021). Russell (2010) divide oito definições de IA entre dois eixos: o primeiro eixo divide as definições entre aquelas que focam no processo de raciocínio do sistema (como o sistema pensa) e aquelas que focam no comportamento do sistema (como o sistema se comporta), enquanto o segundo eixo divide as definições entre aquelas que comparam a performance do sistema em relação a um humano e aquelas que comparam a performance do sistema em relação a um conceito ideal de inteligência, que Russel chama de "racional". As oito definições podem ser vistas no Quadro 1.

Quadro 1 – Classificação das definições de IA

<b>Sistemas que pensam como humanos</b>	<b>Sistemas que pensam racionalmente</b>
"O novo e excitante esforço de fazer com que os computadores pensem... máquinas com a mente, no sentido pleno e literal"(HAUGELAND, 1985)	"O estudo das faculdades mentais através do uso de modelos computacionais"(CHARNIAK, 1985)
"[A automatização de] atividades que se associam ao pensamento humano, atividades como tomada de decisões, resolução de problemas, aprendizagem..."(BELLMAN, 1978)	"O estudo dos cálculos que tornam possível perceber, raciocinar e agir"(WINSTON, 1992)
<b>Sistemas que agem como humanos</b>	<b>Sistemas que agem racionalmente</b>
"A arte de criar máquinas que performam tarefas que exigem inteligência quando feitas por pessoas."(KURZWEIL et al., 1990)	"A inteligência computacional é o estudo da concepção de agentes inteligentes"(POOLE; GOEBEL; MACKWORTH, 1998)
"O estudo de como fazer com que os computadores façam coisas em que, neste momento, as pessoas são melhores"(RICH; KNIGHT, 1991)	"IA... está preocupada com o comportamento inteligente em artefatos"(NILSSON, 1998)

Fonte: Russell (2010)

Estudos iniciais de IA visaram resolver problemas que podiam ser descritos por regras formais e matemáticas. Este tipo de problema, embora seja um difícil desafio intelectual para a mente humana, se mostra fácil de se resolver com um computador. Os verdadeiros desafios para a IA se encontram nos problemas que a mente humana consegue resolver de forma intuitiva, como o reconhecimento de imagens ou vozes, por exemplo. Assim, para resolver este tipo de problema, a área de IA estudou novas formas de se adquirir conhecimento sobre os problemas as serem resolvidos (GOODFELLOW; BENGIO; COURVILLE, 2016).

### 2.1.2 *Machine Learning*, Redes Neurais e *Deep Learning*

Buscando novas formas de aprender sobre os problemas a serem resolvidos por seus modelos, *Machine Learning* surge como uma subárea da IA que visa permitir um computador aprender sobre o ambiente ao seu redor por meio de informações e exemplos do o próprio problema.

Janiesch, Zschech e Heinrich (2021) comparam o processo de aprendizado de uma máquina com o de uma criança: é mais fácil ensinar a uma criança a diferença entre um carro esportivo e um carro comum por meio de exemplos dos dois tipos de automóvel, ao invés de tentar elaborar regras que definam o que é um carro esportivo e o que é um carro comum. Da mesma forma, modelos de *Machine Learning* buscam aprender relações e padrões entre um conjunto de dados por meio de exemplos destes dados em vez de tentarem definir regras extensas e complexas para classificá-los.

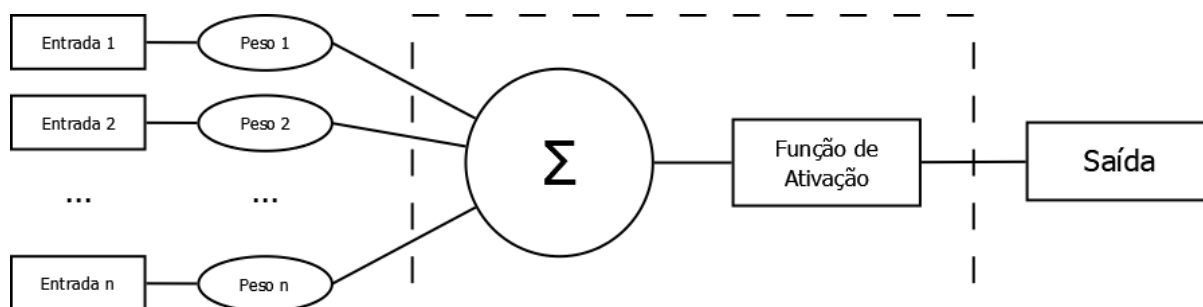
Uma das abordagens utilizadas por *Machine Learning* para resolução de problemas é o uso de redes neurais artificiais, uma área da computação motivada pelo estudo da mente humana.

O cérebro é um sistema de processamento de informações capaz de trabalhar com dados incompletos e difusos e interpretá-los, como ler um texto escrito em uma caligrafia inédita, identificar faces e pessoas conhecidas em ambientes desconhecidos ou reconhecer padrões (BAILER-JONES; GUPTA; SINGH, 2001).

O neurônio biológico é o dispositivo computacional elementar do sistema nervoso. Em uma rede neural, neurônios são ligados uns aos outros por seus axônios. Desta forma, um neurônio possui diversas entradas de informação (os axônios de outros neurônios ligados a si) e gera uma saída (através de seu próprio axônio). Informações recebidas pelo neurônio pelas chamadas conexões sinápticas podem ser excitatórias ou inibitórias: uma conexão excitatória contribui para a criação de um impulso nervoso no axônio do neurônio; uma conexão inibitória faz o oposto, inibindo o impulso nervoso (KOVÁCS, 2002).

Semelhante ao neurônio biológico, o neurônio artificial, apresentado pela primeira vez por McCulloch e Pitts (1943), é uma estrutura com várias entradas de informação e uma saída binárias. As entradas são ponderadas e podem ser excitatórias ou inibitórias, contribuindo para um resultado verdadeiro ou falso na saída do neurônio artificial dependendo de sua função de ativação (KOVÁCS, 2002). A Figura 1 mostra uma representação simples do neurônio biológico proposto por McCulloch e Pitts.

Figura 1 – Representação do neurônio biológico proposto por McCulloch e Pitts.



Fonte: adaptado de Kovács (2002).

Em redes neurais artificiais atuais, neurônios são dispostos em camadas. A primeira camada, que recebe os primeiros estímulos, é chamada de camada de entrada, que envia seus estímulos resultantes para uma segunda camada e assim sucessivamente até a última camada de neurônios, chamada de camada de saída. Camadas entre a camada de entrada e a de saída são comumente chamadas de camadas ocultas (KOVÁCS, 2002).

Utilizando redes neurais artificiais de múltiplas camadas, também conhecidas como redes neurais profundas, modelos de *Machine Learning* pode desenvolver algoritmos capazes de aprender representações de dados cada vez mais complexas e abstratas. Esta área é conhecida *Deep Learning*.

Uma das principais vantagens do *Deep Learning* é a capacidade de aprender automaticamente a partir dos dados, sem a necessidade de extrair manualmente as características relevantes. Isso é possível pelo ao processo de treinamento, no qual o modelo é alimentado com exemplos rotulados e ajusta os pesos das conexões entre os neurônios para minimizar a diferença entre as saídas previstas e os rótulos reais (GOODFELLOW; BENGIO; COURVILLE, 2016).

As redes neurais profundas têm sido aplicadas com sucesso em uma ampla variedade de tarefas, como reconhecimento de imagens, processamento de linguagem natural, tradução automática, reconhecimento de voz, entre outras. Esses avanços foram impulsionados pelo aumento na disponibilidade de grandes conjuntos de dados e pelo desenvolvimento de técnicas de otimização que permitem ajustar os pesos das conexões de forma eficiente (GOODFELLOW; BENGIO; COURVILLE, 2016).

## 2.2 CLASSIFICAÇÃO DE IMAGENS COM *DEEP LEARNING* USANDO O GOOGLE TEACHABLE MACHINE

Apesar da crescente presença *Machine Learning* na vida humana, poucas pessoas possuem conhecimento desta área ou dispõem dos recursos necessários para

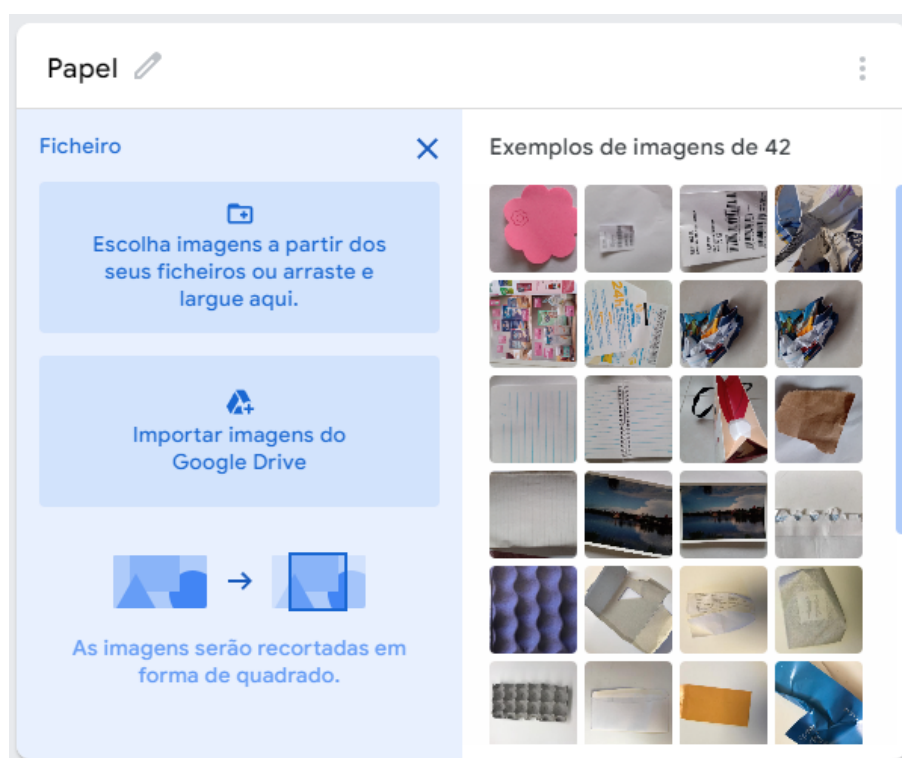
treinar seus próprios modelos. Diante disto, alternativas de ferramentas com o objetivo de viabilizar o treinamento de modelos de *Deep Learning* começaram a surgir.

Criado pela Google em 2017, o Google Teachable Machine é uma destas ferramentas, que permite a criação de modelos de classificação com *Deep Learning* por meio de uma interface de usuário intuitiva e inclusiva, não necessitando de conhecimento da área para utilizar a plataforma. Usando o Google Teachable Machine, um usuário pode fazer o *upload* ou a captura em tempo real de coleções de dados, como imagens ou áudio, treinar um modelo, avaliar seu desempenho e, por fim, exportá-lo, disponibilizando-o online ou transferindo-o na forma de arquivos (CARNEY et al., 2020).

Esta ferramenta permite a criação rápida de um modelo de classificação de dados pelo uso do método de treinamento chamado *transfer learning*, que se baseia na transferência de aprendizado de modelos previamente treinados para o treinamento do novo modelo, adaptando-o para novos cenários (OLIVAS et al., 2009). Assim, no Google Teachable Machine, novos modelos podem ser feitos a partir de um modelo pré treinado, podendo assim facilmente se adaptar a pequenos novos conjuntos de dados.

Para treinar um modelo de classificação com o Google Teachable Machine, o usuário primeiramente define as classes que deseja que o modelo reconheça. Para cada classe criada, é possível fazer o upload de arquivos existentes ou realizar uma captura em tempo real. A Figura 2 mostra uma classe de um projeto de classificação de imagens do Google Teachable Machine alimentada com imagens de objetos de papel.

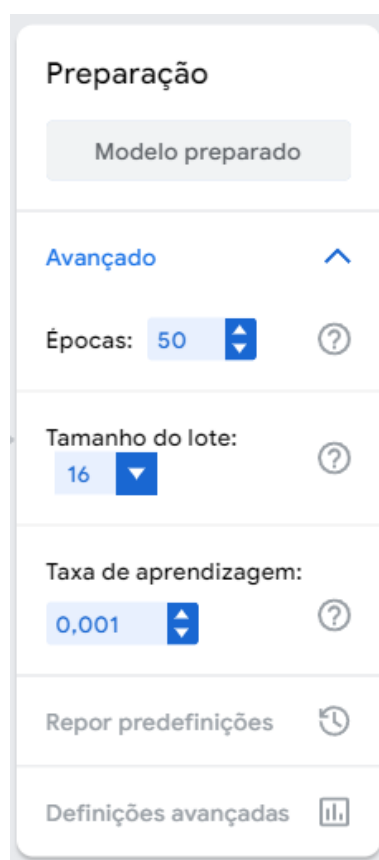
Figura 2 – Classe de um projeto de classificação de imagens do Google Teachable Machine alimentada com imagens de objetos de papel.



Fonte: <https://teachablemachine.withgoogle.com/>

Uma vez definidas as classes, é necessário treinar o modelo. A interface permite ao usuário fazê-lo de forma simplificada, como mostra a Figura 3. Neste componente, o usuário pode alterar critérios como quantidade de gerações, tamanho do lote e taxa de aprendizagem.

Figura 3 – Componente para teste do modelo de classificação de imagens treinado no Google Teachable Machine.



Fonte: <https://teachablemachine.withgoogle.com/>

Por fim, uma vez que o usuário possui o modelo treinado, ele pode testá-lo e exportá-lo. Para testá-lo, o Teachable Machine disponibiliza um componente, mostrado na Figura 4, pelo qual pode-se fazer o upload de imagens testes ou testar o modelo com exemplares de imagem ou áudio em tempo real, via uma webcam ou microfone do dispositivo.

Figura 4 – Componente para teste do modelo de classificação de imagens treinado no Google Teachable Machine.



Fonte: <https://teachablemachine.withgoogle.com/>

O Google Teachable Machine disponibiliza três opções de exportação:

A primeira opção, TensorFlow.js, é recomendada para projetos web. Caso opte-se por fazer o upload do modelo, o Teachable Machine passa a hospedá-lo online, criando um link para que qualquer pessoa possa acessá-lo. Caso opte-se pelo download do modelo, o Teachable Machine baixa um arquivo compactado contendo três arquivos:

- a) *metadata.json*: contém metadados do modelo, como versões das bibliotecas utilizadas, metadata do nome do usuário e modelo, a lista das classes e suas labels, e o tamanho das imagens utilizadas no modelo treinado;
- b) *model.json*: especifica a topologia do modelo;

c) *model.weights.bin*: arquivo contendo os pesos do modelo.

A segunda opção, TensorFlow, é recomendada para projetos em que pretende-se trabalhar com TensorFlow nativo, como Python. Nesta opção, o usuário pode fazer o download do modelo na forma de um modelo keras .h5 ou de um modelo TensorFlow Savedmodel.

A terceira opção de exportação, TensorFlow Lite, é recomendada para projetos de dispositivos móveis ou que usem EdgeTPU. Nesta opção, o usuário pode fazer o download do seu modelo nos formatos tfile floating point, Quantized tfile ou EdgeTPU compiled tfile. Este último é usado em dispositivos da Coral, uma plataforma de IA criada pela Google (RAMOS, 2020).

Por fim, o usuário também pode salvar o projeto integralmente em seu Google Drive por meio do menu lateral do Teachable Machine. As opções de exportação salvam apenas o modelo treinado, enquanto salvar o projeto como um todo inclui os dados utilizados para treinar o modelo. O projeto salvo no Google Drive do usuário tem formato .tm, que pode ser reaberto no Teachable Machine a qualquer momento.

A Tabela 1 mostra todos os formatos de exportação disponibilizados pelo Google Teachable Machine, bem como o tamanho do arquivo do modelo treinado para cada formato.

Tabela 1 – Formatos de exportação disponíveis pelo Google Teachable Machine e o tamanho de cada arquivo resultante.

<b>Formato de exportação</b>	<b>Tamanho do arquivo</b>
Tensorflow.js	2.195KB
Tensorflow (Keras)	2.397KB
Tensorflow (SavedModel)	7.347KB
Tensorflow Lite (vírgula flutuante)	2.043KB
Tensorflow Lite (quantizado)	775KB
Tensorflow Lite (EdgeTPU)	901KB

### 2.3 MIT APP INVENTOR

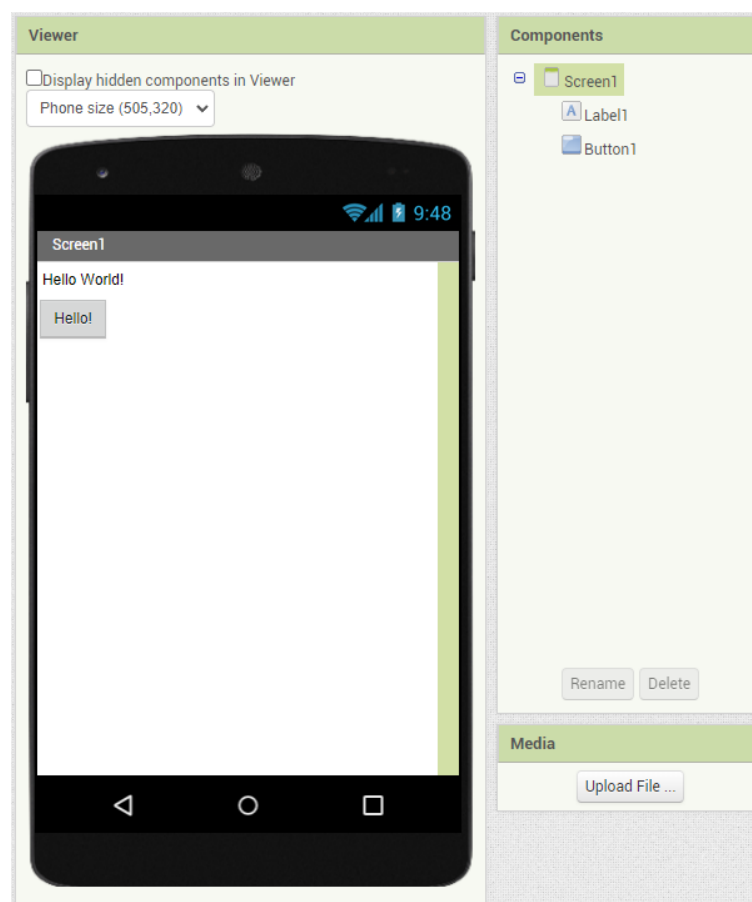
O App Inventor é um ambiente de desenvolvimento de aplicativos móveis lançado publicamente pela Google em 2010 e mantido pela Massachusetts Institute of Technology desde 2011, adotando o nome de MIT App Inventor. Nele, usuários podem criar aplicativos para dispositivos Android e iOS por meio de blocos lógicos que se conectam para formar operações lógicas. Através desta interface visual simples, programadores inexperientes podem criar suas primeiras aplicações funcionais rapidamente, que podem ser distribuídas, baixadas e vendidas como qualquer outro aplicativo móvel (PATTON; TISSENBAUM; HARUNANI, 2019).

O App Inventor é composto por duas abas: *Designer* e o editor de blocos. Na aba *Designer*, desenvolvedores contam com uma coleção de componentes para cons-



truírem a interface visual de suas aplicações. A plataforma disponibiliza componentes visíveis e invisíveis para o usuário. A Figura 5 mostra uma tela simples implementada com alguns componentes disponibilizados nativamente com o App Inventor: uma legenda, intitulada “Label1” e mostrando o texto “Hello World!”, e um botão, intitulado “Button1” e contendo o rótulo “Hello!”.

Figura 5 – Exemplo de tela construída com componentes do App Inventor Designer.



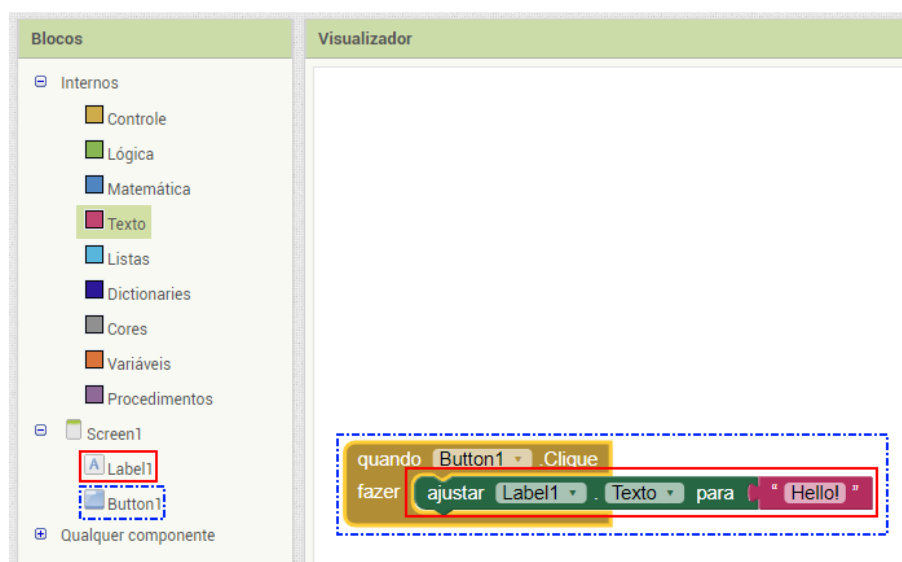
Fonte: <http://ai2.appinventor.mit.edu/>

No editor de blocos é feita a programação dos aplicativos. Em vez de escrever linhas de código, utiliza-se blocos visuais que representam estruturas (como laços de repetição, listas, estruturas condicionais, funções e operadores matemáticos ou lógicos) para implementar instruções a serem executadas pela aplicação (POKRESS; VEIGA, 2013).

O editor de blocos também permite aos usuários referenciar os componentes da aplicação adicionados pelo *Designer*. Em um menu lateral, todos os componentes da tela estão disponíveis como blocos que podem ser conectados a outros blocos internos, permitindo ler e escrever dados neste componente e gerando assim interação entre a aplicação e o usuário. A Figura 6 mostra uma instrução simples disparada

toda vez que o botão “Button1” for pressionado, mudando o texto da legenda “Label1” “Hello!”.

Figura 6 – Exemplo de instrução montada com blocos lógicos de componentes adicionados pelo do App Inventor Designer.



Fonte: <http://ai2.appinventor.mit.edu/>

Apesar dos componentes disponibilizados nativamente pelo App Inventor na sua aba *Designer*, a plataforma recebe constantemente novas sugestões de componentes a serem adicionados. Muitas destas sugestões não são acatadas por diversos motivos: o componente sugerido pode ter um propósito muito específico e ser destinado para poucos usuários, ou pode não receber a atenção do time de desenvolvimento do App Inventor pois este tem outras prioridades. A solução da plataforma para estas demandas é habilitar seus usuários a criarem extensões, para que possam assim atender suas demandas específicas e sem depender da disponibilidade ou julgamento do time de desenvolvimento do App Inventor.

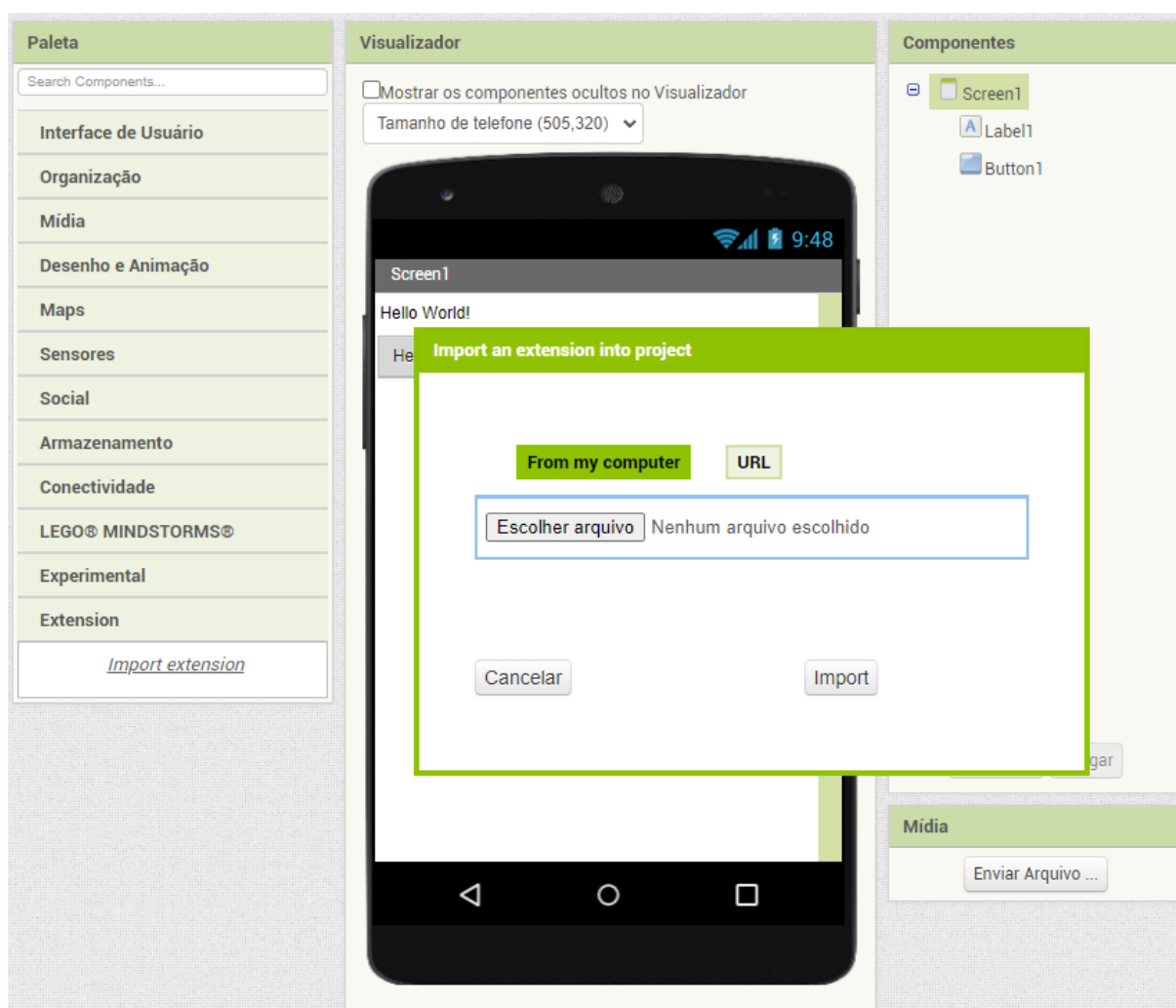
### 2.3.1 Criação de extensões para o App Inventor

Extensões para o App Inventor são arquivos do formato *aix*, que podem posteriormente ser exportados e compartilhados por qualquer plataforma (embora o MIT possua um repositório disponível para desenvolvedores publicarem e divulgarem suas extensões). Atualmente, apenas extensões invisíveis ao usuário final podem ser desenvolvidas, isto é, componentes que não aparecem na tela para interagir diretamente com o usuário via interface gráfica.

Extensões são programadas com a linguagem de programação Java, podendo-se usar bibliotecas Java de fontes terceiras. Uma vez que o desenvolvimento de uma

extensão é terminada, ela pode ser exportada e disponibilizada no formato de arquivo *.aix*. Na aba *Designer* do App Inventor, há um espaço onde pode-se importar arquivos *.aix* para o projeto, mostrado na Figura 7. Após importar a extensão por este espaço, ela aparecerá disponível como um componente dentro da divisória “Extensão” da paleta. Ao adicionar a extensão na tela, esta também poderá ser utilizada como bloco na aba de editor de blocos do App Inventor.

Figura 7 – Importação de arquivos de extensão aix para o projeto do App Inventor.



Fonte: <http://ai2.appinventor.mit.edu/>

### 3 ESTADO DA ARTE

Fez-se um levantamento do estado da arte sobre quais extensões existem para implantar modelos de *Machine Learning* no MIT App Inventor seguindo a proposta de mapeamento sistemático de Petersen et al. (2008).

#### 3.1 DEFINIÇÃO DO PROTOCOLO DE REVISÃO

Com o objetivo de descobrir quais extensões de App Inventor existem para implantar modelos de *Machine Learning* de classificação de imagens, foram definidas as seguintes perguntas de análise:

PA1. Quais extensões existem?

PA2. Em quais ambientes de desenvolvimento os modelos de *Machine Learning* são treinados e exportados?

PA3. Como os modelos treinados são importados nas extensões?

PA4. Quais são as características funcionais dessas extensões?

**Critérios de inclusão/exclusão:** foram incluídas apenas as extensões do MIT App Inventor específicas para importar modelos de *Machine Learning* voltados à classificação de imagens, enquanto outras extensões destinadas a diferentes tarefas, como reconhecimento de fala, foram excluídas da análise.

**Critérios de qualidade:** foram incluídos apenas artigos ou materiais que apresentam informações suficientes sobre extensões para o MIT App Inventor.

**Fontes dos dados:** foi realizado um amplo levantamento de dados, abrangendo todos os materiais e artigos publicados em inglês e disponíveis no Scopus. Dado o caráter emergente da área de estudo, nem todas as extensões desenvolvidas possuem necessariamente artigos científicos publicados, fazendo-se necessário o uso de ferramentas de busca como o Google, visto sua capacidade de indexar informações diversas fontes (HADDAWAY et al., 2015). Também foram considerados os fóruns MIT Media Lab e da comunidade App Inventor para buscar extensões relevantes.

**Definição da *string* de busca:** a *string* de busca foi adaptada para cada fonte de dados, como mostra o Quadro 2.

Quadro 2 – Fontes de busca e *strings* de busca correspondentes.

Fonte	String de busca
SCOPUS	TITLE-ABS-KEY ( ( "machine learning"OR "artificial intelligence"OR "deplearning") AND ( "App Inventor"OR "kodular"OR "thinkable") ) AND ( LIMIT-TO ( SUBJAREA , "COMP" ) )
Google Scholar	"machine learning""artificial intelligence" "App Inventor" "kodular" "thinkable"
Google	"machine learning""artificial intelligence" "App Inventor" "kodular" "thinkable"
MIT media lab	"machine learning", "artificial intelligence", "image", "recognizer"
MIT App Inventor Community	"machine learning", "artificial intelligence", "image", "recognizer"

Fonte: Autor

Para o MIT media lab (<https://appinventor.mit.edu/explore/research>), o Pura Vida Apps (<https://puravidaapps.com/extensions.php>) e o MIT App Inventor Community (<https://community.appinventor.mit.edu/c/extensions>), a *string* de busca não foi utilizada pois não existem ferramentas de busca por *string* nestes ambientes. Em vez disso, utilizou-se as ferramentas de busca nativas do *browser* para pesquisar pelas palavras-chave listadas na tabela para estas fontes. Por se tratarem de fóruns exclusivos do MIT App Inventor, palavras-chave como "App inventor", "kodular" e "thinkable", presentes nas outras search *strings*, não foram julgadas necessárias.

### 3.2 EXECUÇÃO DA BUSCA

A pesquisa foi realizada em maio de 2022. Algumas das fontes não possuíam alternativas apropriadas de filtragem de resultados ou aplicação de *string* de busca, como o Pura Vida Apps e o MIT Media Labs. Para estas fontes, a filtragem foi realizada utilizando a ferramenta de busca do *browser* pelas palavras chaves destacadas no Quadro 2 para cada uma. A Tabela 2 mostra os resultados da execução da busca em suas diversas fases. Para o Pura Vida Apps e o MIT Media Labs, os sub resultados abaixo do nome de cada fonte representam a quantidade de vezes que a palavra chave pesquisada é encontrada, que pode aparecer diversas vezes para um mesmo resultado (por exemplo, a palavra "*image*" pode aparecer tanto no título e na descrição de um resultado de busca do Pura Vida Apps). Assim, o método para contagem dos resultados totais de busca difere do método de contagem dos resultados de cada palavra chave.

Tabela 2 – Resultados da execução da busca.

Fonte	Busca	Analisados	Potencialmente relevantes	Relevantes
SCOPUS	20	20	7	0
Google Scholar	4	4	4	1
Google	419	419	7	1
MIT Media Labs	89	89	7	3
Pura Vita Apps	103	103	3	3
MIT App Inventor Community	563	563	1	0
Total (sem duplicatas)				5

Fonte: dados coletados pelo autor

A diferença entre a soma dos resultados relevantes (9) e o total sem duplicatas (5) se dá pelo fato de que uma das extensões encontradas, o PIC, foi encontrado diversas vezes em múltiplas fontes da Tabela 2, gerando redundância.

Na primeira fase de análise, avaliou-se os títulos dos resultados para considerá-los potencialmente relevantes e obteve-se um total de 28 artefatos a serem analisados. Na segunda fase, o conteúdo dos artefatos foi avaliado de acordo com os critérios de inclusão/exclusão e foram desconsiderados artefatos julgados irrelevantes. Principais causas de exclusão de artefatos nesta etapa incluem:

- Artefatos que, embora tivessem título que o classificasse como potencialmente relevante, não eram pertinentes, estes em sua maioria sendo discussões inacabadas em fóruns ou páginas de conteúdos comerciais (cursos e prestações de serviços);
- Artefatos que, de algum modo, não se adequaram aos critérios de inclusão e exclusão (somente extensões para App Inventor que importam modelos de *Machine Learning* para classificação de imagens). Exemplos são o Personal Audio Classifier (BHATIA et al., 2020) e o Artificial Intelligence Image Classification App 2 (ARTIFICIAL. . . , 2021): um por se tratar de uma extensão de classificação de áudio (não de imagem) e outro por não se tratar de uma extensão para o App Inventor, respectivamente;
- Artefatos que se tornaram obsoletos ou deixaram de receber suporte de seus desenvolvedores, como é o caso da extensão do Microsoft Image Recognizer by Thinkable.

Na última fase de análise, avaliou-se o conteúdo dos títulos obtidos na segunda fase, ainda utilizando-se dos critérios de inclusão/exclusão e de qualidade definidos na Seção 3.1. Como resultado, foram identificadas 5 extensões relevantes, apresentadas na Seção 3.3, no Quadro 3.

### 3.3 RESULTADOS DA REVISÃO

#### 3.3.1 Quais extensões existem?

As extensões para o MIT App Inventor encontradas no levantamento do estado da arte seguindo os critérios definidos na Seção 3.1 são apresentadas no Quadro 3. Além das extensões listadas, foram encontrados diversos tópicos em fóruns e páginas de discussão abordando o tema e expondo uma demanda da comunidade por uma extensão para a tarefa de classificação de imagem. No total, foram encontradas cinco extensões que satisfizeram todos os critérios da seção 3.1.

Quadro 3 – Extensões existentes.

Extensão	Descrição	Referência
AWS AI Services App Inventor Extension	Extensão do MIT App Inventor que permite o uso de serviços de IA da Amazon Web Services, como Amazon Polly, Amazon Translate e Amazon Rekognition para conversões text to speech, tradução de textos e reconhecimento de objetos e textos em imagens.	ÖZGÜN (2019)
LookExtension	Extensão do App Inventor com uma rede neural compilada em si para reconhecimento de imagens.	APPINVENTOR-EXTENSIONS:.. (2014)
ML4K AppInventor Extension	Extensão do App Inventor para uso de <i>Machine Learning</i> , ensinando modelos por meio de textos e imagens enviados pelo site do <i>Machine Learning for Kids</i> .	Corry (2018)
PIC	Uma interface web para treino, teste e análise de modelos de classificação de imagem acompanhada de uma extensão para o MIT App Inventor para o uso destes modelos em aplicativos móveis.	Tang, Utsumi e Lao (2019)
TMIC	Extensão do App Inventor para a implantação de modelos de <i>Machine Learning</i> voltados à classificação de imagens treinados no Teachable Machine.	Oliveira (2022)

Fonte: Elaborado pelo autor com base nas referências citadas.

#### 3.3.2 Em quais ambientes de desenvolvimento os modelos de *Machine Learning* são treinados e exportados?

O Quadro 4 apresenta, para cada extensão do Quadro 3, o ambiente de treinamento dos modelos de *Machine Learning* e o formato dos modelos exportados.

Quadro 4 – Ambientes de treinamento e formato de exportação dos modelos de classificação de imagens para as extensões encontradas.

Extensão	Ambiente de treinamento do modelo	Formato de exportação do modelo
AWS AI Services App Inventor Extension	Modelo não é treinado nem exportado pelo usuário. A extensão usa blocos para consultar online o web service Amazon Rekognition, da Amazon.	
LookExtension	Extensão nativa do MIT App Inventor. Modelo não é treinado nem exportado pelo usuário.	
ML4K AppInventor Extension	Página web própria (machinelearningforkids.co.uk/)	Modelo não é exportado pelo usuário. Uma chave API é utilizada para identificar o modelo treinado, mantido e acessado online pela extensão.
PIC	Página web própria (classifier.appinventor.mit.edu/)	Arquivo compactado .mdl contendo 4 arquivos com informações sobre o modelo.
TMIC	Google Teachable Machine	Arquivo Tensorflow.js a ser armazenado na nuvem. Extensão usa link compartilhável gerado para acessar ao modelo treinado.

Fonte: Elaborado pelo autor.

As extensões AWS AI Services App Inventor Extension e LookExtension não permitem ao usuário treinar e exportar modelos customizados de *Machine Learning*.

A extensão AWS AI Services App Inventor Extension usa o bloco *DetectLabels*, que possui uma imagem como parâmetro, para analisar esta imagem com o serviço online de classificação de imagem da Amazon, o Amazon Rekognition. Uma vez classificada, o bloco retorna na variável *labels* as etiquetas de classificação da imagem em forma de uma *string* separada por vírgulas. A imagem parâmetro pode ser obtida por meio de uma variável de imagem ou por meio do componente Câmera do MIT App Inventor.

A extensão LookExtension disponibiliza blocos como o *ClassifyImageData* (que assim como AWS AI Services App Inventor Extension recebe uma imagem de parâmetro) para classificar imagens com um modelo de *Machine Learning* próprio, não treinado pelo usuário.

A extensão ML4K AppInventor Extension possui uma página *web* própria (machinelearningforkids.co.uk/) onde usuários podem criar modelos de *Machine Learning* para reconhecer, dentre várias opções, imagens, treinando-os por meio de câmeras ou importando dados de treinamento. Estes modelos são mantidos *online* pelo projeto *Machine Learning for Kids*. Para acessá-los pela extensão, insere-se a chave API do projeto nos blocos da extensão quando requisitada. A página também disponibiliza um link por onde o usuário pode importar a extensão no App inventor com a chave API já configurada.

Para o treinamento de seus modelos de *Machine Learning*, o PIC dispõe uma página web própria (classifier.appinventor.mit.edu/) onde o usuário alimentar o modelo



a partir de sua câmera ou importando dados de treinamento. Uma vez satisfeito com o modelo, o usuário pode baixá-lo como um arquivo compactado no formato .mdl, que contém dentro de si outros quatro arquivos: `model_labels.json`, `model.weights.bin`, `model.json` e `transfer_model.json`, contendo, respectivamente, as etiquetas criadas pelo usuário, pesos, topologia do modelo e informações sobre o transfer model usado durante o treinamento do modelo (TANG; UTSUMI; LAO, 2019)

Por fim, a extensão TMIC pode ser utilizada para a implantação de modelos de *Machine Learning* treinados com o Google Teachable Machine. Uma vez satisfeito, o usuário pode subí-lo para a nuvem, fazendo-o ser hospedado pelo próprio Teachable Machine e obtendo um link para acessá-lo. O TMIC utiliza desse link para consultar o modelo treinado e classificar imagens no aplicativo criado no MIT App Inventor (OLIVEIRA, 2022).

### 3.3.3 Como os modelos treinados são importados nas extensões?

O Quadro 5 apresenta como são importados os modelos treinados em cada extensão do Quadro 3.

Quadro 5 – Métodos de importação do modelo de classificação de imagem de cada extensão.

<b>Extensão</b>	<b>Método de importação de modelos treinados</b>
AWS AI Services App Inventor Extension	<i>Não permite importação de novos modelos treinados pelo usuário.</i>
LookExtension	<i>Não permite importação de novos modelos treinados pelo usuário.</i>
ML4K AppInventor Extension	API Key do modelo armazenado em nuvem pela <i>Machine Learning for Kids</i> .
PIC	Upload de arquivo .mdl baixado pela página própria do PIC.
TMIC	Link de acesso ao modelo treinado armazenado em nuvem pela Google Teachable Machine.

Fonte: Elaborado pelo autor.

Por possuírem modelos fixos e imutáveis, as extensões AWS AI Services App Inventor Extension e LookExtension não preveem a importação de novos modelos em suas extensões.

A extensão ML4K AppInventor Extension prevê a utilização apenas de modelos treinados em sua plataforma. Cada modelo é considerado um projeto, que possui uma chave única (API Key) que deve ser referenciada dentro dos blocos lógicos da extensão quando requisitada. A Figura 8 mostra uma lógica feita com os blocos da extensão, onde o bloco rosa aponta, como exemplo, onde a API Key deve ser inserida.

Figura 8 – Bloco lógico da extensão ML4K AppInventor Extension.

```
when ImagePicker1 .AfterPicking
do
  set ML4K1 . Key to "YOUR API KEY HERE"
  set ImagePicker1 . Text to ImagePicker1 . Selection
  call ML4K1 .ClassifyImage
  path ImagePicker1 . Selection

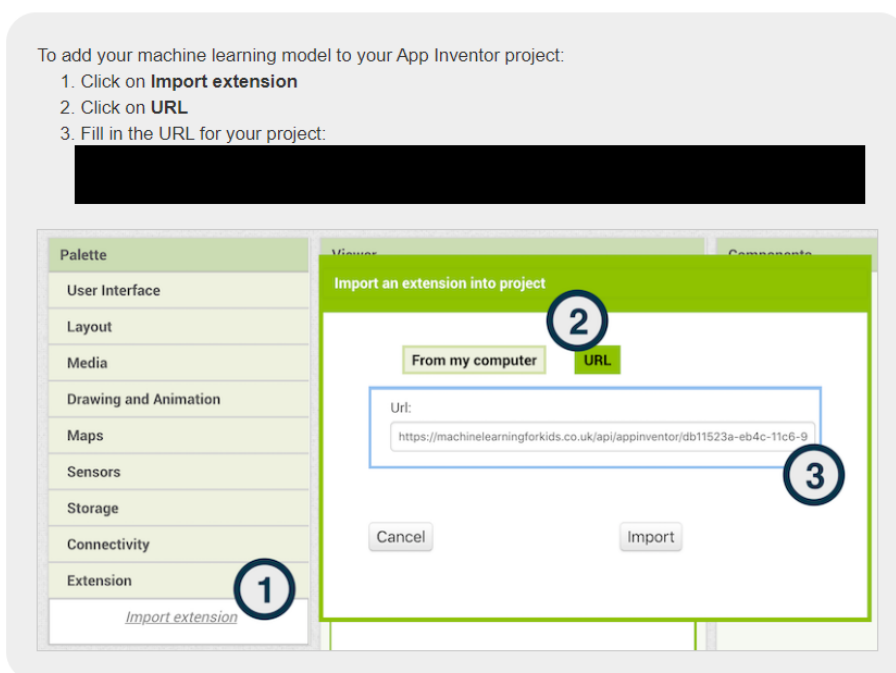
when ML4K1 .GotClassification
  data classification confidence
do
  set Label1 . Text to get classification

when ML4K1 .GotError
  data error
do
  set Label1 . Text to get error
```

Fonte: Corry (2018)

O *Machine Learning for Kids* também facilita o processo disponibilizando um link em cada projeto que, ao ser inserido MIT App Inventor, resulta na importação da extensão com a API key já configurada para consultar o modelo treinado referente àquele projeto. As instruções para importação do modelo desta forma se encontram na Figura 9.

Figura 9 – Instruções para importação da extensão ML4K ApplInventor Extension para o MIT App Inventor com a API Key configurada.



Fonte: <https://machinelearningforkids.co.uk/>

O PIC possui suporte para leitura dos arquivos .mdi gerados pela sua página web para treinamento de modelos de *Machine Learning*. Tang, Utsumi e Lao (2019) explicam que como este arquivo é importado ao MIT App Inventor como uma propriedade da extensão PIC, o *backend* da extensão recebe o caminho em que o arquivo está localizado e o salva internamente para acessar o modelo sempre que necessário.

Por fim, a extensão TMIC recebe como parâmetro o link de acesso ao modelo de *Machine Learning* treinado e armazenado na nuvem pelo Google Teachable Machine. Uma das propriedades do TMIC é o *UrlModel*, responsável por guardar este link de acesso.

### 3.3.4 Quais são as características funcionais dessas extensões?

A Tabela 3 apresenta as características funcionais das extensões do Quadro 3.

Tabela 3 – Fontes de busca e *strings* de busca correspondentes.

<b>Extensão</b>	<b>Blocos lógicos</b>	<b>Permite modelos treinados pelo usuário</b>	<b>Funciona sem acesso à internet</b>
AWS AI Services App Inventor Extension	23	Não	Não
LookExtension	9	Não	Sim
ML4K ApplInventor Extension	14	Sim	Não
PIC	16	Sim	Sim
TMIC	6	Sim	Não

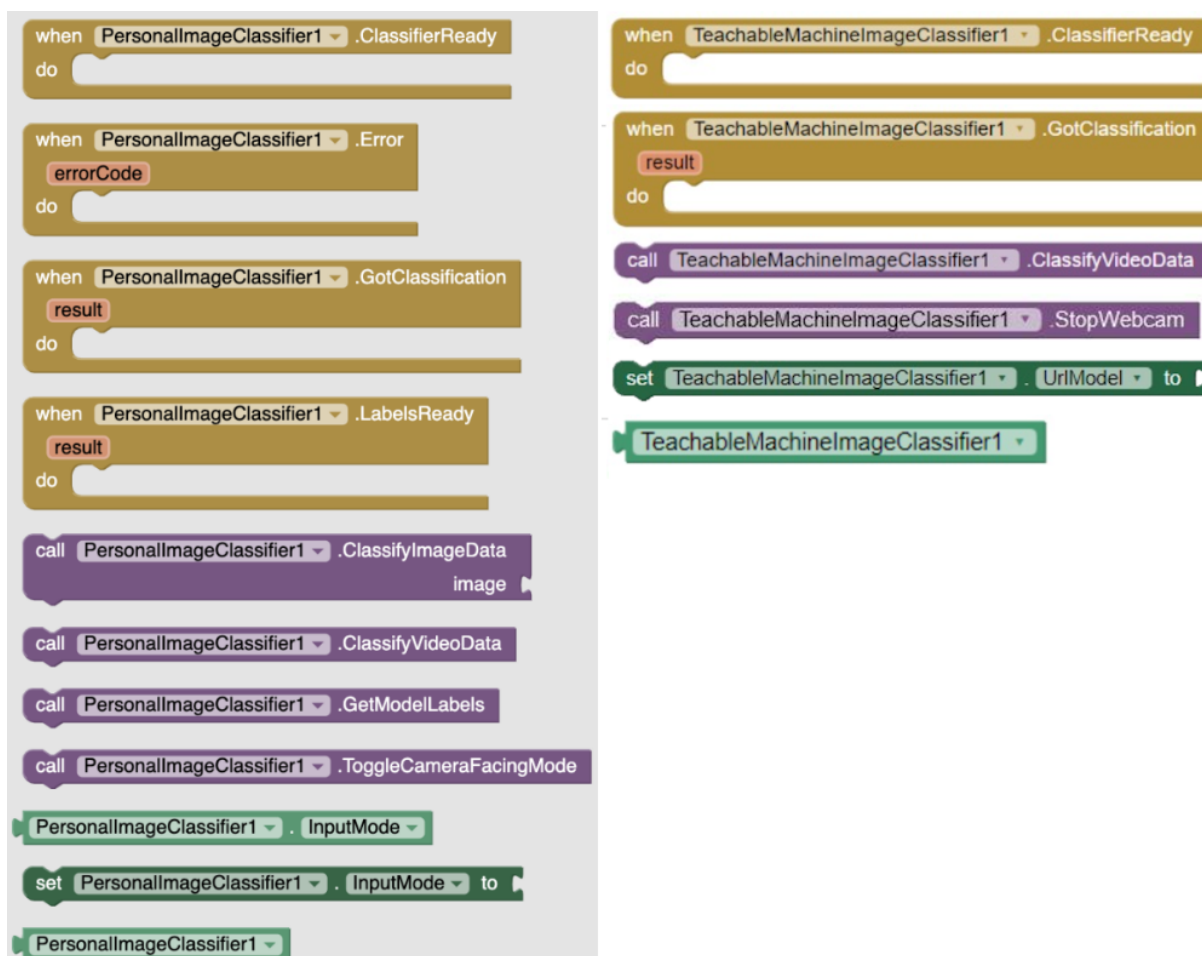
Fonte: Dados coletados pelo autor.

As extensões AWS AI Services App Inventor Extension e LookExtension não permitem que o usuário utilize modelos customizados, oferecendo em vez disso um modelo pré treinado para classificar imagens entre uma vasta coleção de rótulos. Embora estes modelos sejam potentes e tenham a capacidade de identificar uma gama maior de objetos, por serem fixos, eles também não podem ser substituídos por modelos treinados para propósitos mais específicos, capazes de classificar objetos com mais precisão entre uma coleção menor de rótulos.

As extensões AWS AI Services App Inventor Extension, ML4K ApplInventor Extension e TMIC precisam estar conectadas à rede para acessarem seus modelos treinados de *Machine Learning* armazenados na nuvem para realizarem a classificação de imagens.

Os blocos lógicos das extensões PIC e TMIC são apresentados na Figura 10.

Figura 10 – Blocos lógicos da extensão PIC, à esquerda, e TMIC, à direita.



Fonte: Oliveira (2022)

### 3.4 DISCUSSÃO

Como resultado do levantamento do estado da arte, foram encontradas ao total cinco extensões de classificação de imagens para o MIT App Inventor que cumprem os critérios de inclusão/exclusão e de qualidade definidos na Seção 3.1. Destas, destacam-se a AWS AI Services App Inventor Extension, o PIC e o TMIC pela documentação abundante. A AWS AI Services App Inventor Extension possui uma página wiki que pode ser encontrada no repositório do Github do projeto, ao passo de que o PIC e o TMIC possuem documentações completas em formas de publicações acadêmicas por Tang (2019) e Oliveira (2022). Embora não haja muita documentação para as outras duas extensões, estas possuem tutoriais online e materiais didáticos que facilitam a compreensão de como usá-las, além de disponibilizarem seu código em repositórios *open-source* permitindo um estudo mais avançado.

As extensões AWS AI Services App Inventor Extension e LookExtension não

permitem a importação de modelos treinados pelo próprio usuário, o que pode ser considerado uma desvantagem, visto que não é possível implementar modelos mais especializados, que o usuário confia. Além disso, ao impossibilitar que o usuário treine seu próprio modelo de *Machine Learning*, inibe-se a oportunidade de aprendizado sobre a área de *Machine Learning* pelo mesmo. No entanto, reconhece-se que o propósito destas duas extensões não é esse. A AWS AI Services App Inventor Extension possibilita o uso de um modelo potente de classificação de imagens da Amazon, enquanto que o LookExtension é a iniciativa do MIT App Inventor de implementação rápida e simples de modelos de classificação de imagem em aplicativos criados na plataforma.

As extensões LookExtension e PIC são as únicas que permitem que o usuário realize a classificação de imagens sem necessitar conexão à internet, ao passo que a AWS AI Services App Inventor Extension, a ML4K AppInventor Extension e o TMIC precisam constantemente consultar seus modelos de *Machine Learning* que estão armazenados na nuvem. Embora armazenar o modelo treinado fora do aplicativo permita que o mesmo não ocupe tanto espaço de memória do dispositivo, fazê-lo gera essa dependência de conexão à rede, o que pode ser uma desvantagem.

De todas as cinco extensões encontradas, apenas o PIC permite a classificação de imagens sem conexão à internet e com um modelo de *Machine Learning* treinado pelo próprio usuário. Este treinamento, no entanto, ocorre numa plataforma própria do projeto, e exporta o modelo num formato próprio, impedindo que usuário com modelos treinados em outros lugares possam importá-los para a extensão.

**Ameaças à validade.** A fim de minimizar ameaças à validade dos resultados do levantamento do estado da arte, identificou-se e atuou-se em cima de ameaças potenciais e foram aplicadas estratégias de mitigação. Para evitar a omissão de artefatos relevantes, foram consultadas várias fontes. Para cada fonte, quando possível, foi utilizada uma *string* de busca construída para ser a mais abrangente possível, considerando inclusive resultados para plataformas derivadas do MIT App Inventor, como Kodular ou Thunkable.

Por fim, ameaças à seleção de estudos e extração de dados foram mitigadas por meio do fornecimento de uma definição detalhada dos critérios de inclusão/exclusão, além da revisão dos resultados pela orientadora.

## 4 DESENVOLVIMENTO

O presente capítulo descreve o desenvolvimento da evolução da extensão TMIC proposta neste trabalho.

### 4.1 ANÁLISE DE CONTEXTO E DEFINIÇÃO

A extensão TMIC desenvolvida por Oliveira (2022) tem como objetivo permitir aos desenvolvedores importar modelos de classificação de imagem treinados no Google Teachable Machine para seus projetos no MIT App Inventor. O público alvo da extensão são alunos entre 10 e 14 anos que não possuem conhecimento prévio sobre *Machine Learning* e que estejam realizando o curso "*Machine Learning para Todos!*" da iniciativa Computação na Escola, para que possam aplicar seus conhecimentos recém adquiridos (OLIVEIRA, 2022). A partir desta proposta, professores, estudantes e interessados poderão treinar seus próprios modelos de classificação de imagem na Google Teachable Machine e utilizá-los para o desenvolvimento de aplicativos no MIT App Inventor.

Atualmente, para utilizar um modelo treinado desta forma, a extensão prevê que o modelo esteja hospedado na nuvem pela própria Google Teachable Machine, e que o link para o modelo seja fornecido para a extensão por meio de suas propriedades. Uma vez que a extensão tenha acesso ao modelo treinado, ela estará apta a classificar imagens da câmera traseira do aplicativo e fornecer os resultados de sua classificação.

Por tratar-se de uma evolução do TMIC, o público alvo e a proposta da evolução desenvolvida neste trabalho mantêm-se os mesmos da extensão original, com o objetivo de apenas aumentar as possibilidades de implementação da mesma, reduzindo as limitações identificadas.

O objetivo da evolução, portanto, é implementar estas melhorias no TMIC, expandindo as possibilidades de uso da extensão original. Para possibilitar a classificação de imagens sem conexão à internet, a extensão deve passar a permitir o carregamento do modelo de classificação de imagens a partir de arquivos locais disponibilizados por desenvolvedores durante o desenvolvimento do aplicativo. Para a utilização da câmera frontal e permitir a classificação de imagens da galeria do dispositivo, será necessário desenvolver novos blocos lógicos e propriedades para a extensão.

### 4.2 ANÁLISE DE REQUISITOS

#### 4.2.1 Requisitos funcionais

Os requisitos funcionais da evolução da extensão TMIC são mostrados no Quadro 6.

Quadro 6 – Requisitos funcionais.

ID	Requisito	Descrição
RF01	Importar modelos de classificação de imagens treinados no Google Teachable Machine	A extensão deve possibilitar a importação de arquivos de modelos de classificação de imagens treinados no Google Teachable Machine que permita sua execução sem conexão com a internet.
RF02	Executar modelos de classificação de imagens treinados no Google Teachable Machine.	A extensão deve possibilitar a execução do modelo de <i>Machine Learning</i> importado para a classificação de imagens.
RF03	Abrir a câmera frontal do dispositivo.	A extensão deve permitir a utilização da câmera frontal do dispositivo móvel para captura de imagens.
RF04	Solicitar permissão de utilização da câmera frontal do dispositivo.	A extensão deve solicitar ao usuário a permissão para utilização da câmera frontal do do dispositivo ao tentar abri-la.
RF05	Classificar imagens capturadas pela câmera frontal do dispositivo.	A extensão deve permitir, por meio do modelo de classificação de imagens treinado no Google Teachable Machine e importado pelo usuário, a classificação de imagens obtidas pela câmera frontal do dispositivo.
RF06	Disponibilizar os resultados da classificação de imagem obtidas pelo modelo importado.	A extensão deve disponibilizar os resultados da classificação de imagem realizada pelo modelo de <i>Machine Learning</i> 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.
RF07	Utilizar fotos salvas no celular para classificar a imagem.	A extensão deve permitir usar uma foto salva no celular para realizar a classificação, possibilitando também um pré-processamento da foto (cortar, rotacionar etc.)

Fonte: Elaborado pelo autor

#### 4.2.2 Requisitos não funcionais

Os requisitos não funcionais da evolução da extensão TMIC descritos no Quadro 7.

Quadro 7 – Requisitos não funcionais.

ID	Requisito	Descrição
RNF01	<i>Back end</i> da extensão desenvolvida em Java.	O <i>back end</i> da extensão deve ser desenvolvido na linguagem de programação Java utilizando o <i>framework</i> de desenvolvimento do App Inventor.
RNF02	<i>Front end</i> da extensão desenvolvida em Javascript	O <i>front end</i> da extensão deve ser desenvolvido na linguagem de programação Javascript, utilizando também a linguagem de marcação HTML e a linguagem de estilização CSS.
RNF03	Sistema operacional Android	A extensão deve ser implementada em aplicativos para o sistema operacional Android.
RNF04	Independência de conexão à internet.	A extensão deve permitir que o aplicativo desenvolvido classifique imagens sem necessidade de conexão com a internet.
RNF05	Internacionalização	Os blocos de programação da extensão devem estar disponíveis em português do Brasil e inglês

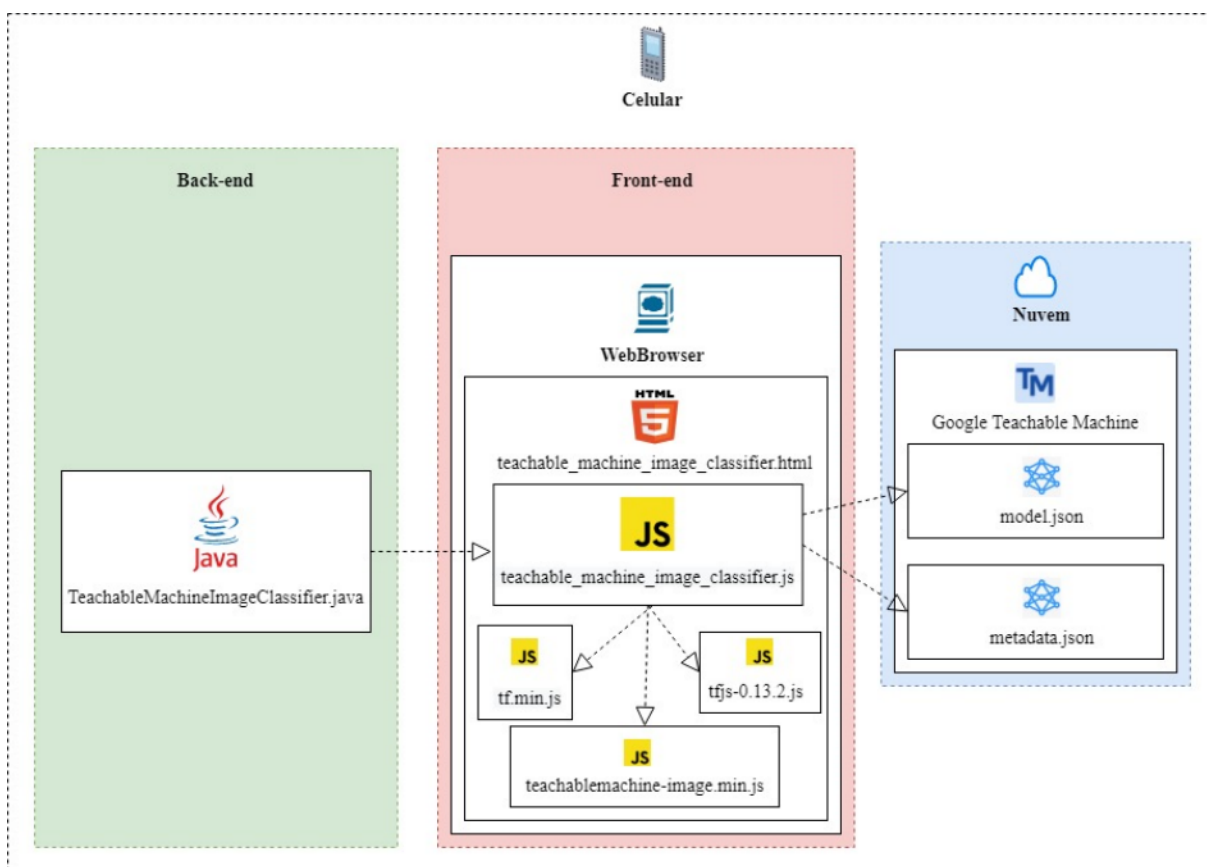
Fonte: Elaborado pelo autor



### 4.3 ARQUITETURA DA EXTENSÃO TMIC E SUA EVOLUÇÃO

A extensão TMIC foi desenvolvida baseando-se na extensão PIC, que é dividida em *back end* e *front end*. Consequentemente, o TMIC possui uma arquitetura semelhante, representada na Figura 11.

Figura 11 – Arquitetura da extensão original TMIC



Fonte: Oliveira (2022)

O *front end* da extensão TMIC é composto pelos seguintes arquivos (OLIVEIRA, 2022):

***teachable\_machine\_image\_classifier.html*** Arquivo HTML responsável por exibir a câmera dentro de uma página web que irá ser renderizada no Navegador Web do aplicativo. É neste arquivo que são carregados os outros arquivos Javascript que compõem o *front end*, responsáveis por executar os *frameworks* que irão realizar o controle do funcionamento da câmera do celular, carregar o modelo e a predição das imagens.

***teachable\_machine\_image\_classifier.js*** Arquivo Javascript responsável por carregar o modelo do *Google Teachable Machine*, executar a predição da imagem

capturada da câmera do celular e retorná-la para o *back end*.

***teachablemachine-image.min.js*** *Framework* do GTM que possui as funções necessárias para realizar a classificação da imagem a partir do modelo fornecido.

***tfjs-0.13.2.js*** Arquivo Javascript com funções do *framework* TensorFlow.js, utilizado pelo *framework* do *Google Teachable Machine* para realizar as classificações.

***tf.min.js*** Arquivo Javascript com funções do *framework* TensorFlow.js, utilizado pelo *framework* do *Google Teachable Machine* para realizar as classificações.

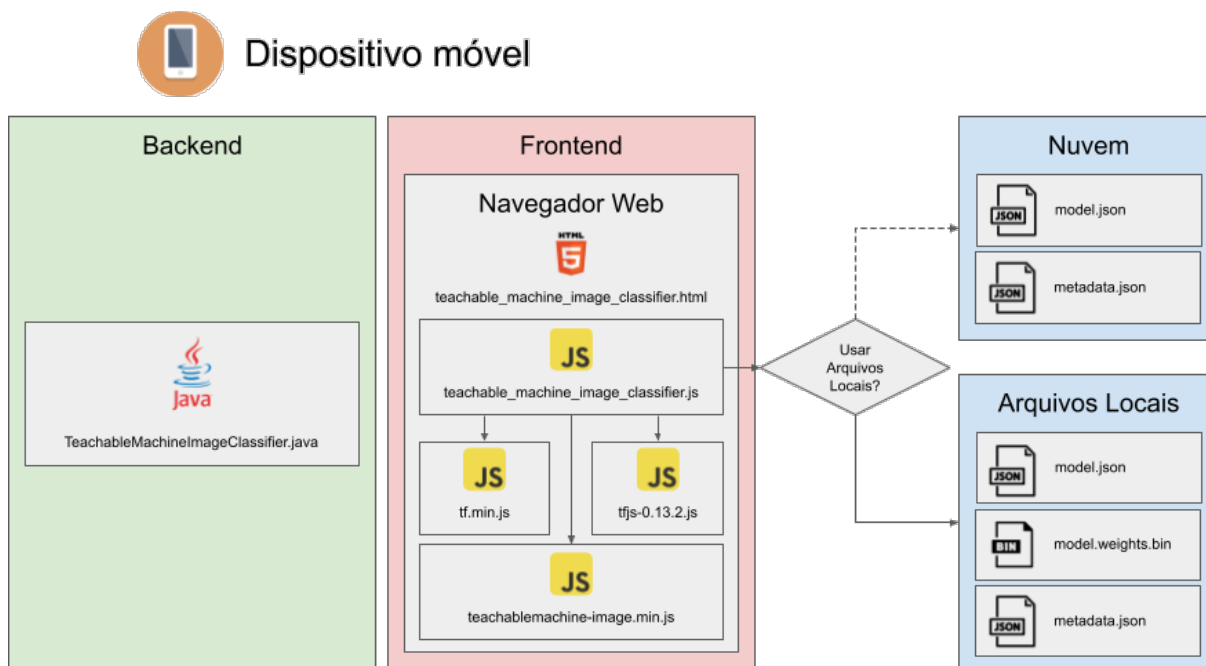
O *back end* da extensão TMIC é composto por um único arquivo (OLIVEIRA, 2022):

***TeachableMachineImageClassifier.java*** Classe Java responsável por inicializar a extensão e definir a funcionalidade dos blocos. As funções desta classe correspondem aos blocos lógicos e propriedades homônimas da extensão presentes no App Inventor. Essa classe também é responsável por se comunicar com o *front end*, enviando e *back end* recebendo dados dele.

O desenvolvimento da evolução do TMIC procurou preservar a arquitetura da extensão original, realizando apenas alterações nos arquivos existentes para que os requisitos da evolução fossem contemplados.

Considerando que a origem dos arquivos do modelo de classificação de imagem é considerada na arquitetura da extensão, e que os requisitos funcionais da evolução demandam o uso de arquivos locais para o carregamento do modelo, a única alteração na arquitetura é a inclusão da possibilidade dos arquivos do modelo de classificação de imagens serem arquivos locais, como mostra a Figura 12.

Figura 12 – Arquitetura da evolução da extensão



Fonte: Adaptado pelo autor de Oliveira (2022)

#### 4.4 PROPRIEDADES E BLOCOS LÓGICOS DA EVOLUÇÃO DA EXTENSÃO

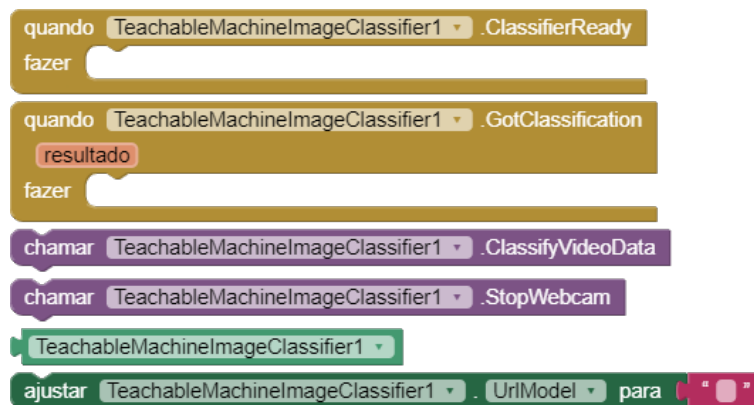
A presente seção apresenta as propriedades e os blocos lógicos da evolução da extensão TMIC.

Extensões do MIT App Inventor possuem propriedades, que são as variáveis relacionadas ao funcionamento da extensão. Algumas destas propriedades podem ser vistas no painel "Propriedades" na aba *Designer* do MIT App Inventor, onde podem ter seus valores iniciais definidos pré compilação.

Extensões também possuem blocos lógicos, que são os meios pelos quais desenvolvedores podem alterar o estado de propriedades da extensão em tempo de execução.

A Figura 13 apresenta os blocos lógicos da extensão original TMIC. Os blocos *ClassifierReady* e *GotClassification* provêm da extensão PIC, na qual o TMIC é baseado (OLIVEIRA, 2022).

Figura 13 – Blocos lógicos da extensão TMIC.



Fonte: Autor.

O Quadro 8 descreve a funcionalidade de cada propriedade e bloco lógico da extensão original TMIC.

Quadro 8 – Blocos e propriedades da extensão TMIC.

Nome	Tipo	Funcionalidade
UrlModel	Propriedade	URL do modelo treinado no <i>Google Teachable Machine</i> .
WebView	Propriedade	Referência a um componente NavegadorWeb ( <i>WebView</i> ) presente na tela.
<i>ClassifierReady</i>	Bloco lógico	Evento disparado após a extensão terminar de carregar o modelo de classificação de imagem por meio da URL fornecida.
<i>GotClassification</i>	Bloco lógico	Evento disparado pelo bloco <i>ClassifyVideoData</i> após o término da classificação da imagem, retornando a lista de predições pra cada classe do modelo.
<i>ClassifyVideoData</i>	Bloco lógico	Inicia a classificação da imagem mostrada pela câmera do dispositivo (mostrada no componente NavegadorWeb). Ao terminar a classificação, dispara o bloco <i>GotClassification</i> .
<i>ClassifyVideoData</i>	Bloco lógico	Para a captura de imagem pela câmera do dispositivo.
<i>TeachableMachineImageClassifier</i>	Bloco lógico	Retorna uma instância específica da extensão.
<i>set UrlModel to</i>	Bloco lógico	Altera o valor da propriedade <i>UrlModel</i> em tempo de execução.

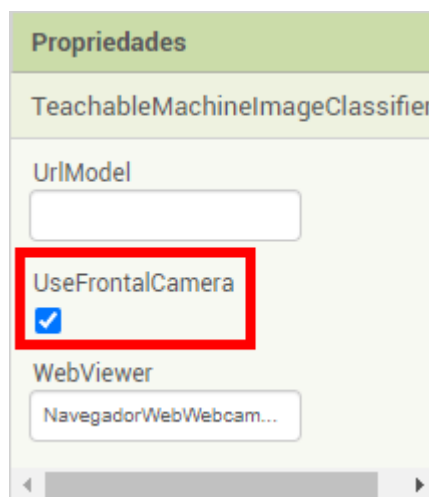
Fonte: Adaptado de Oliveira (2022).

Visando atender os requisitos definidos no Quadro 6, a evolução desenvolvida da extensão TMIC traz duas novas propriedades visíveis no painel da aba *Designer* do MIT App Inventor e três novos blocos lógicos.

Nas propriedades, foi adicionado o novo campo *UseFrontalCamera*, responsável por fazer a extensão utilizar ou não a câmera frontal do dispositivo. A Figura 14 mostra

o painel da evolução da extensão com todas as propriedades.

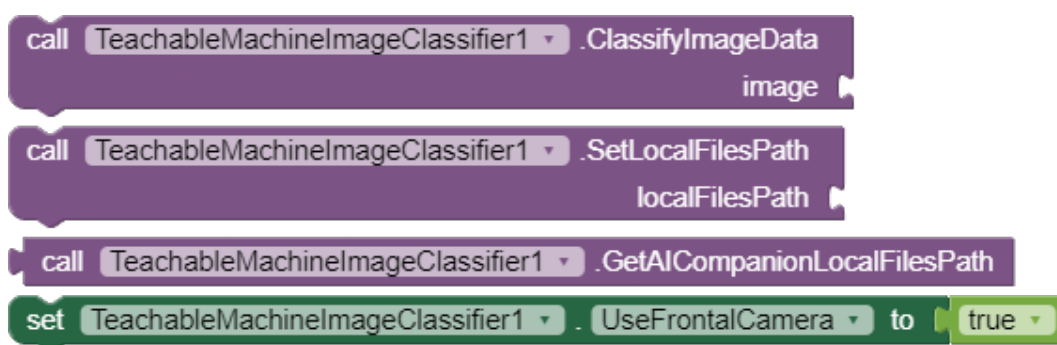
Figura 14 – Propriedades da evolução da extensão TMIC, destacando a nova propriedade *UseFrontalCamera*



Fonte: <http://ai2.appinventor.mit.edu/>

A Figura 15 mostra os quatro blocos lógicos criados durante o desenvolvimento da evolução da extensão.

Figura 15 – Blocos lógicos da evolução da extensão TMIC.



Fonte: Autor.

Estes blocos são:

***ClassifyImageData*** Recebe como parâmetro *image* o caminho para uma imagem. O modelo, então, classifica esta imagem, em vez da imagem reproduzida pela câmera do dispositivo.

***SetLocalFilePath*** Define o caminho para os arquivos locais do modelo de classificação para que a extensão possa fazer a classificação sem acesso à internet. Caso

este bloco não seja acionado, ou o valor do parâmetro *localFilesPath* seja nulo, a extensão tentará classificar imagens com o modelo hospedado online por meio da Url providenciada na propriedade *UrlModel*, da forma que a extensão original faz atualmente.

**GetAICompanionLocalFilesPath** Retorna o caminho do diretório onde o *App Inventor Companion* salva os arquivos carregados no painel *Media* na plataforma *MIT App Inventor*. Este bloco deve ser passado como parâmetro para o bloco *SetLocalFilesPath* enquanto se estiver usando o *App Inventor Companion* para depurar o aplicativo em desenvolvimento.

**Set UseFrontalCamera to** Bloco para definir o valor da propriedade *UseFrontalCamera*. Este bloco é criado a partir da existência da propriedade, e a alteração do valor da mesma não é recomendada durante o uso da extensão.

Um resumo destas propriedades e blocos lógicos resultantes do desenvolvimento da evolução do TMIC pode ser encontrado no Quadro 9.

Quadro 9 – Novos blocos e propriedades da extensão TMIC.

Nome	Tipo	Funcionalidade
<i>UseFrontalCamera</i>	Propriedade	<i>Checkbox</i> que, quando ticada, faz com que a extensão use a câmera frontal para captura de imagem.
<i>ClassifyImageData</i>	Bloco lógico	Recebe como parâmetro o caminho para uma imagem e inicia sua classificação. Ao terminar a classificação, dispara o bloco <i>GotClassification</i> .
<i>SetLocalFilesPath</i>	Bloco lógico	Bloco lógico para definir o caminho onde estão armazenados os arquivos locais do modelo de classificação de imagens.
<i>GetAICompanionLocalFilesPath</i>	Bloco lógico	Bloco lógico que retorna o caminho do diretório onde o <i>App Inventor Companion</i> guarda <i>assets</i> .
<i>Set UseFrontalCamera to</i>	Bloco lógico	Bloco lógico para definir o valor da propriedade <i>UseFrontalCamera</i> em tempo de execução.

Fonte: Elaborado pelo autor.

## 4.5 PREPARAÇÃO E INSTALAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Para o desenvolvimento da evolução da extensão TMIC, seguiu-se os passos da instalação da extensão descritos por Oliveira (2022).

A evolução da extensão foi desenvolvida no sistema operacional Windows 11 Home (64 bits). Para o desenvolvimento do código, foi instalada a IDE IntelliJ 2023.1 Community Edition.

Para o desenvolvimento de uma extensão para o MIT App Inventor, é necessário baixar o código fonte do próprio App Inventor, disponível em <https://github.com/mit-cml/appinventor-extensions>. Uma Após feito isso, baixou-se o código fonte da extensão

TMIC, disponível em <https://codigos.ufsc.br/gqs/tmic>. A pasta *src* do código fonte do TMIC foi copiada para a pasta *appinventor-extensions/appinventor/components* (onde também há uma pasta *src*, resultando na união das duas).

Em seguida, instalou-se o Java SE Development Kit 8 e o Apache Ant 1.10.1 (mesma versão usada para o desenvolvimento do TMIC), que é utilizado para a compilação da extensão em um arquivo *.aix*. Para a utilização do Apache Ant, foi necessário configurar algumas variáveis de ambiente, mostradas no Quadro 10.

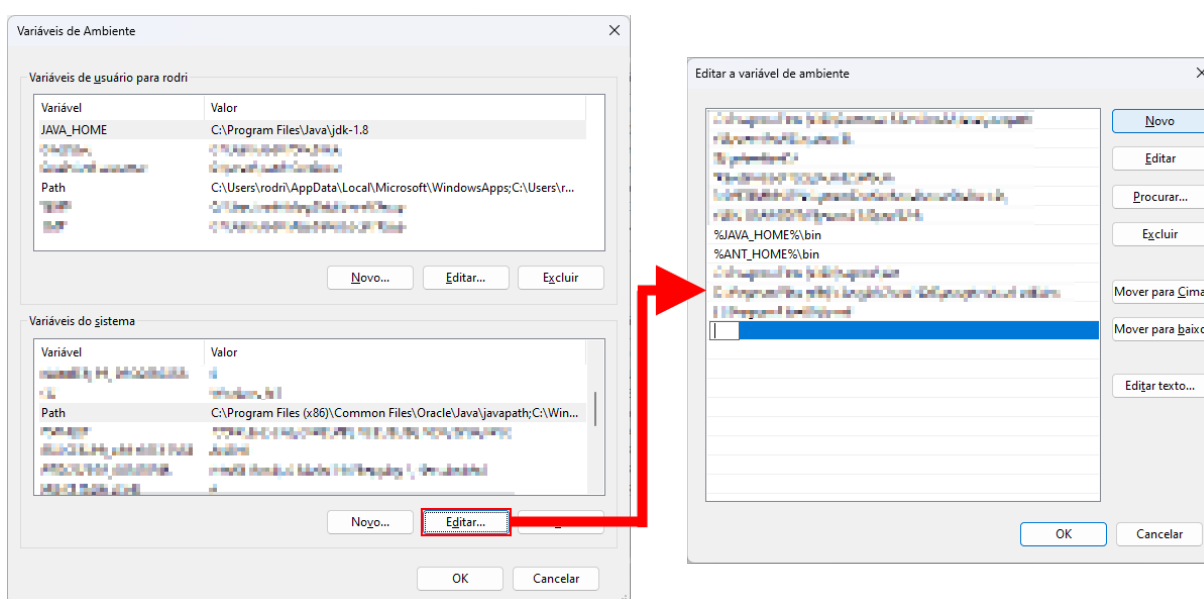
Quadro 10 – Variáveis de ambiente.

Variável de ambiente	Valor
_JAVA_OPTIONS	-Xmx1024m
ANT_HOME	Caminho da pasta bin dos arquivos do Apache (ex: C:/Program Files/Apache/apache-ant-1.10.1)
ANT_OPTIONS	-Xmx256M
JAVA_HOME	Caminho da pasta de instalação do Java (ex: C:/Program Files/Java/jdk-1.8)
CLASSPATH	%ANT_HOME%/lib;%JAVA_HOME%/lib
Path	É necessário adicionar "%ANT_HOME%/bin;%JAVA_HOME%/bin"ao final do <i>Path</i> atual.

Fonte: Elaborado pelo autor.

No Windows, para adicionar valores ao *Path*, deve-se acessar as variáveis de ambiente e editar a variável *Path*, adicionando as entradas "%ANT\_HOME%/bin"e "%JAVA\_HOME%/bin"clikando no botão "Novo". Este processo é ilustrado na Figura 16.

Figura 16 – Passos para alterar o *Path* nas variáveis de ambiente do Windows.

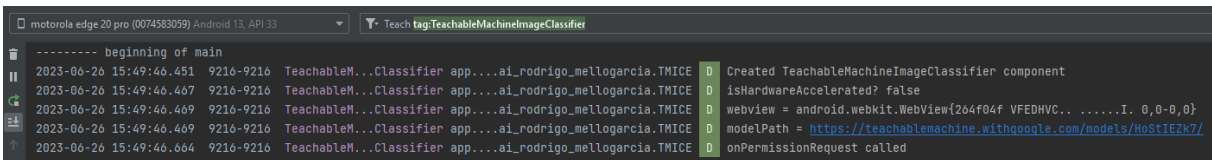


Fonte: Autor.

Com as variáveis de ambiente configuradas, é possível compilar a extensão e gerar o arquivo `.aix`, que é usado para importá-la na plataforma MIT App Inventor. Para isso, deve-se executar o comando `ant extensions` na pasta `/appinventor-extensions/appinventor` do código fonte do App Inventor. O arquivo `.aix` será gerado na pasta `/appinventor-extensions/appinventor/components/build/extensions`.

Para a depuração do código, utilizou-se do *Logcat*, uma ferramenta de linha de comando que despeja um registro de mensagens do sistema, incluindo mensagens escritas com a classe *Log*. Algumas interfaces de desenvolvimento já possuem o *Logcat* nativamente, como é o caso do Android Studio. Conectando o dispositivo físico onde a extensão está sendo executada ao computador, é possível ver no *Logcat* mensagens de erro e *logs* implementados no código, como mostra a Figura 17.

Figura 17 – *Logcat* mostrando mensagens do código com a classe *Log*.



```
----- beginning of main
2023-06-26 15:49:46.451  9216-9216  TeachableM...Classifier app...ai_rodrigo_mellogarcia.TWICE  D  Created TeachableMachineImageClassifier component
2023-06-26 15:49:46.467  9216-9216  TeachableM...Classifier app...ai_rodrigo_mellogarcia.TWICE  D  isHardwareAccelerated? false
2023-06-26 15:49:46.469  9216-9216  TeachableM...Classifier app...ai_rodrigo_mellogarcia.TWICE  D  webView = android.webkit.WebView{264f04f VFEDHVC.. ....I. 0,0-0,0}
2023-06-26 15:49:46.469  9216-9216  TeachableM...Classifier app...ai_rodrigo_mellogarcia.TWICE  D  modelPath = https://teachablemachine.withgoogle.com/models/5oSt1Ezk7/
2023-06-26 15:49:46.664  9216-9216  TeachableM...Classifier app...ai_rodrigo_mellogarcia.TWICE  D  onPermissionRequest called
```

Fonte: Autor.

## 4.6 DESENVOLVIMENTO DA EVOLUÇÃO DA EXTENSÃO

### 4.6.1 Uso de arquivos locais para carregamento do modelo de classificação

A presente subseção descreve o desenvolvimento da funcionalidade que contempla os requisitos funcionais RF01 e RF02 da Quadro 6.

Atualmente, a extensão original TMIC utiliza a biblioteca *Teachable Machine Library* para carregar o modelo de classificação de imagens. A partir da URL fornecida pelo desenvolvedor na propriedade *UriModel*, a função *classifyVideoData* do arquivo do *front end* `teachable_machine_image_classifier.js` deriva as duas URLs necessárias para a função *load* carregar o modelo de classificação de imagem. A Figura 18 mostra a função *classifyVideoData*. As URLs necessárias para a função *load* são derivadas nas linhas 47 e 48, enquanto que a função em si é chamada na linha 54.



Figura 18 – Função *UrlModel* na classe *TeachableMachineImageClassifier.java*

```
40 // Load the model and classify image
41 async function classifyVideoData() {
42
43     if(urlModel == null) {
44         alert("URL has not been set!");
45     }
46
47     const modelURL = urlModel + "model.json";
48     const metadataURL = urlModel + "metadata.json";
49
50     // load the model and metadata
51     // Refer to tmImage.loadFromFiles() in the API to support files from a file picker
52     // or files from your local hard drive
53     // Note: the pose library adds "tmImage" object to your window (window.tmImage)
54     model = await tmImage.load(modelURL, metadataURL);
55     maxPredictions = model.getTotalClasses();
56
57     // predict can take in an image, video or canvas html element
58     let result = [];
59
60     const prediction = await model.predict(webcamPredict.canvas);
61     for (let i = 0; i < maxPredictions; i++) {
62         result.push([prediction[i].className, prediction[i].probability.toFixed(2) * 100]);
63     }
64
65     TeachableMachineImageClassifier.reportResult(JSON.stringify(result));
66 }
```

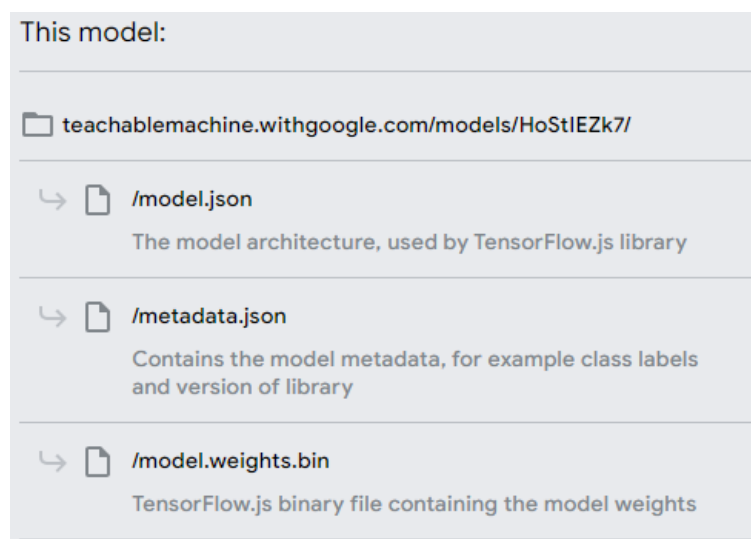
Fonte: Autor.

A biblioteca *Teachable Machine Library* também possui uma função para carregar o modelo a partir de arquivos locais chamada *loadFromFiles*, que recebe como parâmetros três arquivos que compõem o modelo:

- **model.json**: a arquitetura do modelo, usado pela biblioteca *TensorFlow.js*.
- **metadata.json**: contém os metadados do modelo, como as classes do modelo.
- **model.weights.bin**: arquivo binário que contém os pesos do modelo.

Ao hospedar um modelo de classificação de imagem na rede, o Google Teachable Machine também disponibiliza na mesma página estes três arquivos para serem baixados em um painel mostrado pela Figura 19.

Figura 19 – Painel para *download* dos arquivos do modelo de classificação de imagem, disponível na página do Google Teachable Machine do modelo.



Fonte: <https://teachablemachine.withgoogle.com/>

Para o contexto da evolução da extensão TMIC, espera-se que o desenvolvedor disponibilize estes arquivos previamente para o aplicativo para que a extensão carregue o modelo de classificação de imagens. Para isso, optou-se em usar o próprio painel Mídia do MIT App Inventor, visto que é algo nativo da própria plataforma e que desenvolvedores já estão familiares.

Uma vez que os arquivos sejam carregados pelo painel Mídia do MIT App Inventor, é necessário acessá-los por meio do *front end*.

Idealmente, estes arquivos devem ser acessados de forma discreta, dispensando quaisquer ações do usuário final, o que traz um problema: por questões de segurança, Javascript não consegue acessar arquivos locais sem permissão do usuário, geralmente sendo feito por meio de um componente HTML *input* ou semelhantes. A solução, portanto, foi criar novos arquivos com o construtor *File*, passando o conteúdo dos arquivos originais para os arquivos criados.

É necessário, então, acessar os arquivos do modelo de classificação por meio do *back end* (ou seja, por meio da classe *TeachableMachineImageClassifier.java*). Para isso, foram criadas as funções *readModelFile*, *readMetadataFile* e *readWeightsFile* na classe *JsonObject*, que é a classe do *TeachableMachineImageClassifier.java* que contém as funções acessíveis no *front end*. Cada uma destas funções lê um dos três arquivos e retorna o conteúdo do mesmo codificado em Base64, para garantir a integridade do conteúdo entre as linguagens Java e Javascript. A Figura 20 mostra a função *readModelFile*, sendo que as funções *readMetadataFile* e *readWeightsFile* são similares.

Figura 20 – Função *readModelFile* da classe *JsObject*.

```
@JavascriptInterface
public String readModelFile() {
    File modelFile = new File( pathname: assetsPath + "/model.json");

    byte[] fileContent;
    try {
        fileContent = Files.readAllBytes(modelFile.toPath());
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    return Base64.getEncoder().encodeToString(fileContent);
}
```

Fonte: Autor.

No *front end*, cria-se três novos arquivos homônimos aos arquivos originais, que são então passados como parâmetro para a função *loadFromFiles* da biblioteca *Teachable Machine Library* (Figura 21). Os parâmetros para a criação dos arquivos são:

- O conteúdo do arquivo original, retornado pela função do *JsObject*, decodificado de volta para binário (*base64ToBin*).
- O nome do novo arquivo, que deve necessariamente ser "model.json", "metadata.json" ou "model.weights.bin".
- O tipo MIME do arquivo, que para os arquivos JSON é "application/json" e para o arquivo binário é "application/octet-stream".

Figura 21 – Função *loadFromFiles* carregando o modelo a partir de arquivos criados.

```
const modelFile = new File([base64ToBin(TeachableMachineImageClassifier.readModelFile()),
    "model.json", {type: "application/json"}]);
const metadataFile = new File([base64ToBin(TeachableMachineImageClassifier.readMetadataFile()),
    "metadata.json", {type: "application/json"}]);
const weightsFile = new File([base64ToBin(TeachableMachineImageClassifier.readWeightsFile()),
    "model.weights.bin", {type: "application/octet-stream"}]);

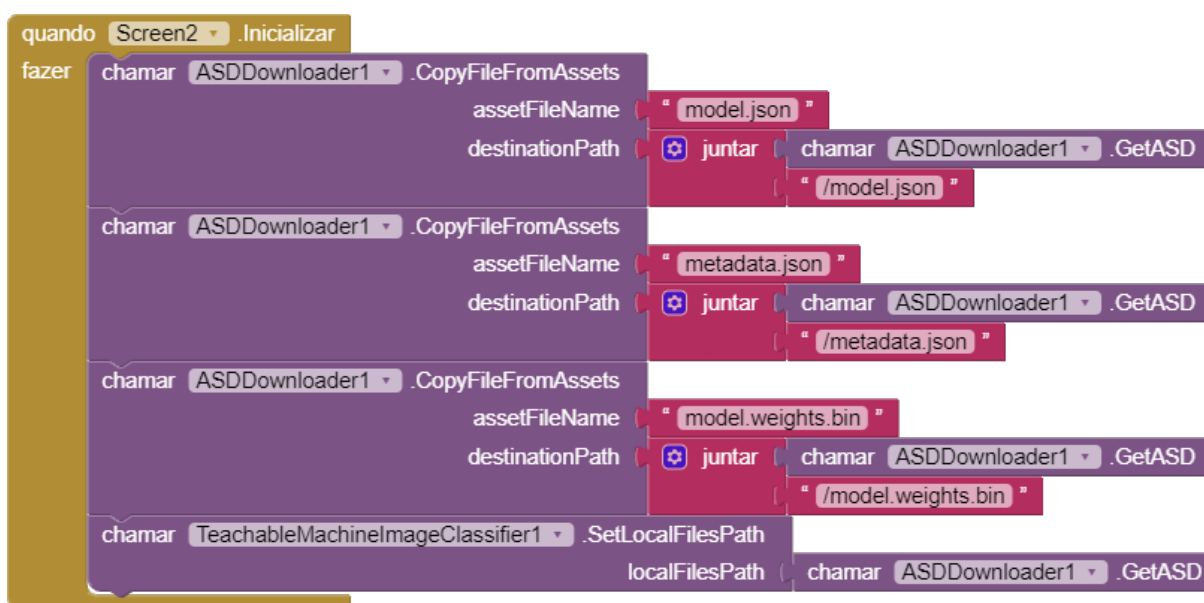
model = await tmImage.loadFromFiles(modelFile, weightsFile, metadataFile);
```

Fonte: Autor.

Se o desenvolvedor estiver depurando seu aplicativo com a ajuda do *AI Companion*, os arquivos carregados no painel Mídia já estarão disponíveis na pasta específica

do aplicativo (App Specific Directory - ASD). No caso do aplicativo compilado (.apk), os arquivos da Mídia estão no armazenamento interno do aplicativo, sendo necessário movê-los para a pasta específica do aplicativo. A extensão *ASDDownloader* (KARAN, 2023) é capaz de fazer isso com a combinação de seus blocos lógicos *CopyFileFromAssets* e *GetASD*, como mostra a Figura 22. É necessário também informar para a extensão TMIC o caminho para o diretório específico do aplicativo por meio do bloco lógico *SetLocalFilePath*.

Figura 22 – Copiando os arquivos locais do armazenamento interno para o diretório específico do aplicativo com o *ASDDownloader*.



Fonte: Autor.

Ao atribuir um valor para o bloco lógico *SetLocalFilePath*, a extensão define como verdadeiro uma variável interna "useLocalFiles", que é quem define como a extensão carrega o modelo de classificação de imagens, como mostra a Figura 23.

Figura 23 – Nodo de decisão do método de carregamento do modelo de classificação de imagem.

```

if (TeachableMachineImageClassifier.useLocalFiles()) {
  const modelFile = new File([base64ToBin(TeachableMachineImageClassifier.readModelFile())], "model.json", {type:
  const metadataFile = new File([base64ToBin(TeachableMachineImageClassifier.readMetadataFile())], "metadata.json
  const weightsFile = new File([base64ToBin(TeachableMachineImageClassifier.readWeightsFile())], "model.weights.b

  model = await tmImage.loadFromFiles(modelFile, weightsFile, metadataFile);
} else {
  if(urlModel == null) {
    alert("URL has not been set!");
  }

  const modelURL = urlModel + "model.json";
  const metadataURL = urlModel + "metadata.json";

  model = await tmImage.load(modelURL, metadataURL);
}

```

Fonte: Autor.

Como o caminho para o diretório específico do AICompanion e do aplicativo compilado é diferente (o primeiro termina em ".../files/assets/" enquanto que o outro termina apenas em ".../files"), criou-se também um bloco *GetAICompanionLocalFilesPath*, que retorna o caminho para o diretório específico do AICompanion. Assim, para ler arquivos locais e depurar o projeto com o AICompanion, é necessário este bloco como parâmetro para o bloco lógico *SetLocalFilesPath*, como mostra a Figura 24.

Figura 24 – Bloco *SetLocalFilesPath* recebendo como parâmetro o bloco *GetAICompanionLocalFilesPath*

```

quando Screen2 .Inicializar
fazer chamar TeachableMachineImageClassifier1 .SetLocalFilesPath localFilesPath
chamar TeachableMachineImageClassifier1 .GetAICompanionLocalFilesPath

```

Fonte: Autor.

#### 4.6.2 Classificação de arquivos da galeria do dispositivo

A presente subseção descreve o desenvolvimento da funcionalidade que contempla o requisito funcional RF07 da Quadro 6.

Atualmente, a extensão original TMIC classifica apenas a imagem mostrada no momento pela câmera do dispositivo. Na Figura 18, a linha 60 mostra a função *model.predict*, responsável por realizar a classificação de uma imagem que é passada como parâmetro para a função. Neste caso, *webcamPredict.canvas* passa a imagem capturada pela câmera no momento. Para classificar uma imagem da galeria, por-

tanto, é necessário passar esta imagem como parâmetro para a função *model.predict* (Figura 25).

Figura 25 – Criando uma imagem para passar para a função *model.predict*.

```
const imageFile = new File([base64ToBin(TeachableMachineImageClassifier.readImageFile()), "selectedImage.jpg", {type: "image/jpeg"}]);
const bitmapImage = await createImageBitmap(imageFile);

const prediction = await model.predict(bitmapImage);
for (let i = 0; i < maxPredictions; i++) {
  result.push([prediction[i].className, prediction[i].probability.toFixed(2) * 100]);
}

TeachableMachineImageClassifier.reportResult(JSON.stringify(result));
```

Fonte: Autor.

Na classe *TeachableMachineImageClassifier.java*, foi criada a função *readImageFiles* (Figura 26), que, assim como as funções que lêem os arquivos que compõem o modelo de classificação de imagem, recebe como parâmetro o caminho para o arquivo de imagem a ser lido e retorna o conteúdo do mesmo, codificado em Base64.

Figura 26 – Função *readImageFiles*.

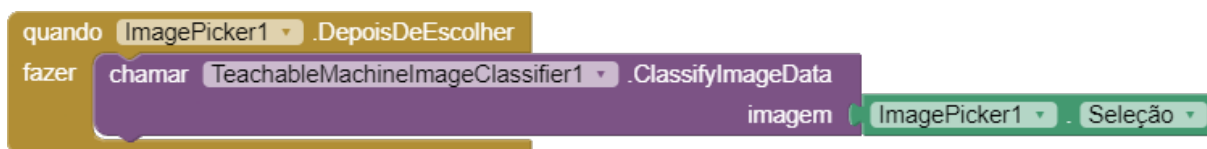
```
@JavascriptInterface
public String readImageFile() {
  File weightsFile = new File(imagePath);

  byte[] fileContent;
  try {
    fileContent = Files.readAllBytes(weightsFile.toPath());
  } catch (IOException e) {
    throw new RuntimeException(e);
  }

  return Base64.getEncoder().encodeToString(fileContent);
}
```

Fonte: Autor.

Foi criado o bloco lógico *classifyImageData*, que recebe como parâmetro o caminho para a imagem a ser classificada. A extensão *ImagePicker* do MIT App Inventor abre a galeria do dispositivo e retorna o caminho para a imagem selecionada. Desta forma, utilizando estes dois blocos, como mostra a Figura 27, desenvolvedores podem possibilitar a um usuário classificar uma imagem da galeria do dispositivo.

Figura 27 – Blocos lógicos do *ImagePicker* e *classifyImageData*.

Fonte: Autor.

Por fim, é necessário permitir à função *classifyVideoData*, do arquivo do *front end teachable\_machine\_image\_classifier.js*, saber quando classificar a imagem da câmera e quando classificar a imagem da galeria. Para isso, criou-se uma variável interna *isClassifyingFromGallery*, que é definida como "verdadeiro" quando o bloco *ClassifyImageData* é chamado, e como "falso" quando o bloco *ClassifyVideoData* é chamado, como mostra a Figura 28.

Figura 28 – Atribuição da variável *isClassifyingFromGallery*.

```

@SimpleFunction(description = "Performs the classification")
public void ClassifyVideoData() {
    isClassifyingFromGallery = false;
    assertWebView( method: "ClassifyVideoData");
    webview.evaluateJavascript( script: "classifyVideoData(\"\" + \"\");", resultCallback: null);
}

@SimpleFunction(description = "Classifies a given image")
public void ClassifyImageData(String image) {
    if (image == null) {
        Log.d(LOG_TAG, msg: "Couldn't find a file");
        return;
    }

    isClassifyingFromGallery = true;
    imagePath = image;

    assertWebView( method: "ClassifyVideoData");
    webview.evaluateJavascript( script: "classifyVideoData(\"\" + \"\");", resultCallback: null);
}

```

Fonte: Autor.

Dentro da função *classifyVideoData*, esta variável define como a extensão classificará a imagem, como mostra a Figura 29.

Figura 29 – Nodo de decisão da imagem a ser classificada.

```

if (TeachableMachineImageClassifier.isClassifyingFromGallery()) {
  const imageFile = new File([base64ToBin(TeachableMachineImageClassifier.readImageFile())], "selectedImage.jpg", {type: "image/jpeg"});
  const bitmapImage = await createImageBitmap(imageFile);

  const prediction = await model.predict(bitmapImage);
  for (let i = 0; i < maxPredictions; i++) {
    result.push([prediction[i].className, prediction[i].probability.toFixed(2) * 100]);
  }

  TeachableMachineImageClassifier.reportResult(JSON.stringify(result));
} else {
  const prediction = await model.predict(webcamPredict.canvas);
  for (let i = 0; i < maxPredictions; i++) {
    result.push([prediction[i].className, prediction[i].probability.toFixed(2) * 100]);
  }

  TeachableMachineImageClassifier.reportResult(JSON.stringify(result));
}

```

Fonte: Autor.

### 4.6.3 Uso da câmera frontal do dispositivo

A presente subseção descreve o desenvolvimento da funcionalidade que contempla os requisitos funcionais RF03, RF04 e RF05 da Quadro 6.

Atualmente, a extensão original TMIC renderiza a câmera traseira do aplicativo no componente NavegadorWeb por meio da biblioteca *Teachable Machine Library*. Na função *webcam.setup*, um valor pode ser atribuído para a propriedade *facingMode*. Caso nenhum valor seja atribuído, é usado o valor padrão "environment", o que resulta no uso da câmera traseira do dispositivo. Caso o valor "user", seja atribuído, a câmera frontal passa a ser utilizada.

Criou-se uma propriedade *UseFrontalCamera* na classe *TeachableMachineImageClassifier.java*, mostrada na Figura 30.

Figura 30 – Propriedade *UseFrontalCamera*

```

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN)
@SimpleProperty
public void UseFrontalCamera(final boolean useFC) { useFrontalCamera = useFC; }

```

Fonte: Autor.

Essa variável é usada no arquivo *teachable\_machine\_image\_classifier.js* do *front end* para atribuir o valor do parâmetro *facingMode* da função *webcam.setup*, como mostra a Figura 31.



Figura 31 – configuração da câmera do dispositivo.

```
webcam = new tmImage.Webcam(500, 500, flip); // width, height, flip
var camera = (TeachableMachineImageClassifier.useFrontalCamera() ? "user" : "environment");
await webcam.setup({ facingMode: camera }); // request access to the webcam. "environment" = back cam, "user" = frontal cam.
```

Fonte: Autor.

#### 4.6.4 Testes da evolução

Implementadas as novas funcionalidades, realizou-se uma série de testes para averiguar o correto funcionamento das mesmas. Todos os testes foram realizados no aplicativo exemplo, cujo desenvolvimento é descrito na Seção 5.2. Para todos os testes, definiu-se o valor da propriedade *UseFrontalCamera* como verdadeiro, com o objetivo de testar o funcionamento do acesso à câmera frontal. Em todos os testes, a imagem da câmera frontal foi mostrada com sucesso.

Primeiramente, configurou-se os blocos *SetLocalFilePath* e *GetAICompanionLocalFilePath*, como mostra a Figura 24, para testar se a extensão lê arquivos locais enquanto se está depurando o projeto com a ajuda do AICompanion. Também foi atribuído um valor nulo para a propriedade *UrlModel* (campo deixado vazio), para garantir que a classificação só pudesse ser proveniente de arquivos locais. O aplicativo conseguiu realizar a classificação da imagem com sucesso, comprovando que o AICompanion conseguiu carregar o modelo através dos arquivos locais fornecidos.

Em seguida, configurou-se os blocos como mostra a Figura 22 para testar se a extensão, enquanto compilada em um arquivo .apk, lê arquivos locais. Ainda com valor nulo atribuído para a propriedade *UrlModel* e o dispositivo em "modo avião", que impede qualquer conexão com a internet, o aplicativo conseguiu realizar a classificação da imagem com sucesso, comprovando que o aplicativo compilado conseguiu carregar o modelo através dos arquivos locais fornecidos.

Seguindo a configuração de blocos apresentada na Figura 37, tanto o AICompanion quanto o aplicativo compilado conseguiram realizar a classificação de uma imagem da galeria.

Por fim, realizou-se, utilizando a evolução da extensão, um teste das funcionalidades da extensão original, com o objetivo de garantir que nenhuma delas foi comprometida no desenvolvimento do presente trabalho. Como resultado, verificou-se que a evolução do TMIC ainda é possível de utilizar a câmera traseira do celular e carregar o modelo de classificação de imagens através da *Url* fornecida na propriedade *UrlModel*.

## 5 DESENVOLVIMENTO DO MATERIAL DIDÁTICO E DO APLICATIVO EXEMPLO

Devido a todas as alterações que a evolução do TMIC traz em relação à original, fez-se necessário o desenvolvimento de um novo material didático, ensinando os alunos a usarem as novas funcionalidades da extensão. Junto ao material, foi desenvolvido um projeto exemplo de aplicativo no MIT App Inventor, com exemplos de implementação dos novos blocos lógicos e uso das novas propriedades. O presente capítulo descreve a elaboração destes dois artefatos.

Os arquivos disponibilizados para compor o material didático estão listados no Quadro 11.

Quadro 11 – Componentes do material didático desenvolvido

<b>Material</b>	<b>Tipo</b>	<b>Descrição</b>
Tutorial de implantação do TMIC atualizado	Apresentação de Slides	Apresentação de slides ensinando a implementação da extensão TMIC em um projeto no App Inventor e o uso de seus blocos.
Aplicativo exemplo	Arquivo .aia	<i>Wireframe</i> do aplicativo desenvolvido no App Inventor, mencionado no material didático, o qual os alunos utilizarão como base para o tutorial do curso.
Extensão TMIC (evolução)	Arquivo .aix	Arquivo da evolução da extensão TMIC, resultado deste trabalho.

Fonte: Autor.

### 5.1 DESENVOLVIMENTO DO MATERIAL DIDÁTICO

Feitas as evoluções na extensão TMIC, é necessário atualizar a apresentação de slides utilizada no material didático para que este ensine as novas funcionalidades desenvolvidas neste trabalho.

No tutorial, foram adicionados slides ensinando os alunos da iniciativa a utilizar as novas funcionalidades desenvolvidas neste trabalho, como a classificação de imagens da galeria e o uso de arquivos locais para carregar o modelo de classificação de imagens. Alguns slides precisaram ser atualizados para mostrar as novas propriedades da extensão, como mostra a Figura 32.

Figura 32 – Slide do tutorial atualizado para mencionar a nova propriedade *UseFrontalCamera*



Fonte: Autor.

O tutorial atual ensina alunos a classificar imagens com o modelo hospedado na internet. Para mostra a funcionalidade de construir modelos de classificação a partir de arquivos locais, criou-se uma sessão de slides mostrando o passo a passo de como implementar esta funcionalidade. A Figura 33 mostra um desses slides, ensinado alunos a importarem os arquivos pelo painel Mídia do MIT App Inventor.

Figura 33 – Slide do tutorial ensinando a importar arquivos para o projeto pelo painel Mídia

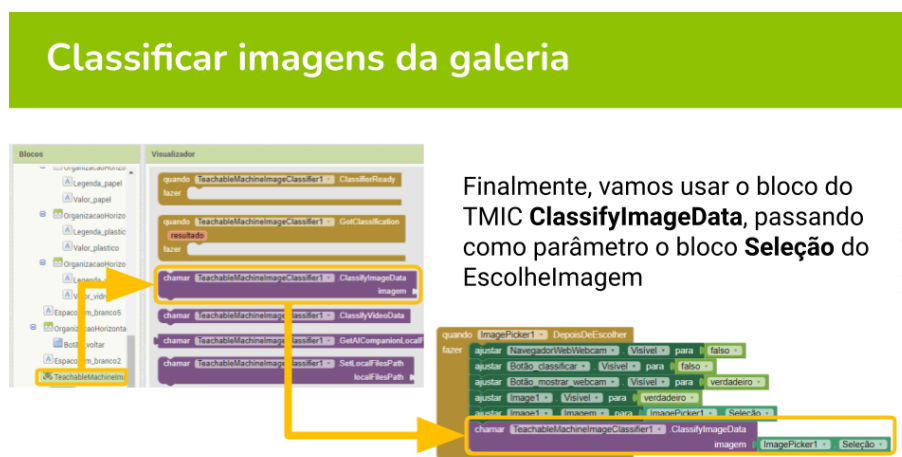


Fonte: Autor.

Ensinou-se também a classificar imagens da galeria do dispositivo utilizando a extensão TMIC em conjunto com o componente *ImagePicker*. A Figura 34 mostra um

dos slides desta parte do tutorial, que ensina a utilizar o bloco do TMIC *ClassifyImage-Data*.

Figura 34 – Slide do tutorial ensinando a classificar a imagem escolhida da galeria



Fonte: Autor.

## 5.2 DESENVOLVIMENTO DO APLICATIVO EXEMPLO

Baseado no aplicativo exemplo feito por Oliveira (2022), foi desenvolvido um projeto no MIT App Inventor que implementa todas as novas funcionalidades da evolução do TMIC, com o objetivo do mesmo servir como um exemplo de aplicação dos novos blocos lógicos e propriedades da extensão. Este aplicativo é composto por duas telas: a primeira sendo uma tela introdutória e a segunda sendo a que usa a extensão de fato. Assim, as alterações para comportar as novas funcionalidades que a evolução da extensão traz foram feitas na segunda tela do aplicativo. As duas telas do aplicativo original são mostradas na Figura 35.

Figura 35 – Telas do aplicativo de classificação de lixo (primeira tela à esquerda, segunda tela à direita)



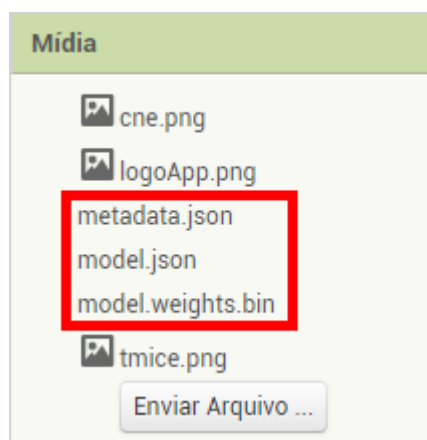
Fonte: Autor

### 5.2.1 Uso de arquivos locais para carregamento do modelo de classificação

Para este aplicativo, foi utilizado o modelo de classificação de imagens treinado para classificar lixo, disponível no tutorial de implantação do modelo de *Machine Learning* com TMIC da iniciativa Computação na Escola.

Para utilizar arquivos locais para o carregamento do modelo de classificação, primeiramente é necessário carregar estes arquivos no painel Mídia do MIT App Inventor, como mostra a Figura 36. Os arquivos de modelo, metadados e pesos do modelo precisam impreterivelmente ser chamados "*model.json*", "*metadata.json*" e "*model.weights.bin*", respectivamente.

Figura 36 – Arquivos do modelo de classificação de imagem carregados no painel Mídia do MIT App Inventor.



Fonte: <http://ai2.appinventor.mit.edu/>

Em seguida, foram usados o bloco *CopyFileFromPath* e *GetASD* da extensão ASDDownloader (KARAN, 2023) para copiar os arquivos do armazenamento interno do dispositivo para a pasta específica do aplicativo, onde a extensão terá acesso a eles. Depois, o valor do bloco *GetASD* também foi passado para o bloco da extensão *SetLocalFilePath*. Ao acionar este bloco, a extensão passa a utilizar os arquivos locais para o carregamento do arquivo. A Figura 22 da Seção 4.6 mostra o resultado final.

Caso se deseje depurar o aplicativo com o AICompanion, o bloco lógico *GetASD* do ASDDownloader deve ser substituído pelo bloco *GetAICompanionLocalFilePath*, do TMIC.

### 5.2.2 Classificação de arquivos da galeria do dispositivo

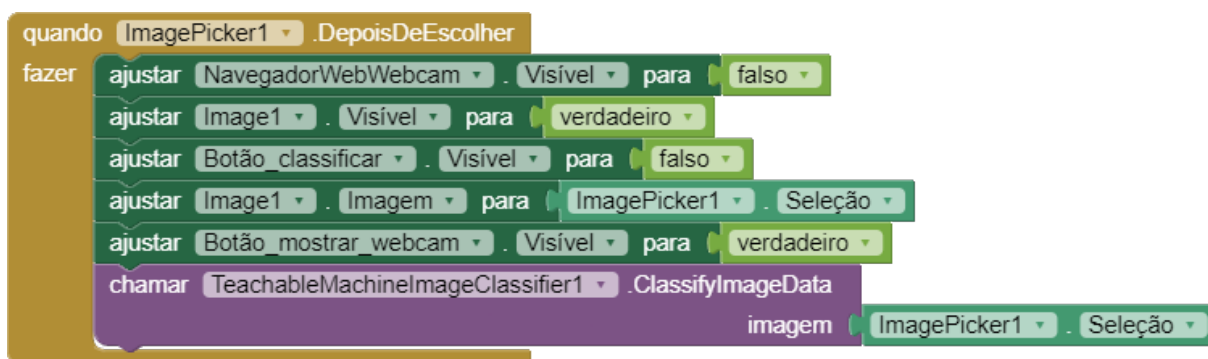
Para a demonstração desta funcionalidade, adicionou-se à tela:

1. Um componente **ImagePicker** chamado "*ImagePicker1*" que, ao clicado, abre a galeria do dispositivo;
2. Um componente **Imagem** chamado "*Image1*", para mostrar a imagem selecionada da galeria. Este componente é inicializado invisível para o usuário;
3. Um componente **Botão** chamado "*botao\_mostrar\_webcam*", para voltar a mostrar a câmera do dispositivo.

No bloco *DepoisDeEscolher* do ImagePicker (1), que é acionado assim que uma foto da galeria é selecionada, foram escondidos o navegador web (que mostra a câmera do dispositivo) e o botão "Classificar", e foram revelados o componente de imagem (2) e o botão "Mostra Webcam"(3). No componente de imagem, também foi

carregada a imagem selecionada pelo ImagePicker, para facilitar a compreensão dos resultados da classificação. Finalmente, foi passado o caminho da imagem selecionada para o bloco *ClassifyImageData* da extensão, que realiza a classificação da imagem. A Figura 37 mostra a interação destes blocos.

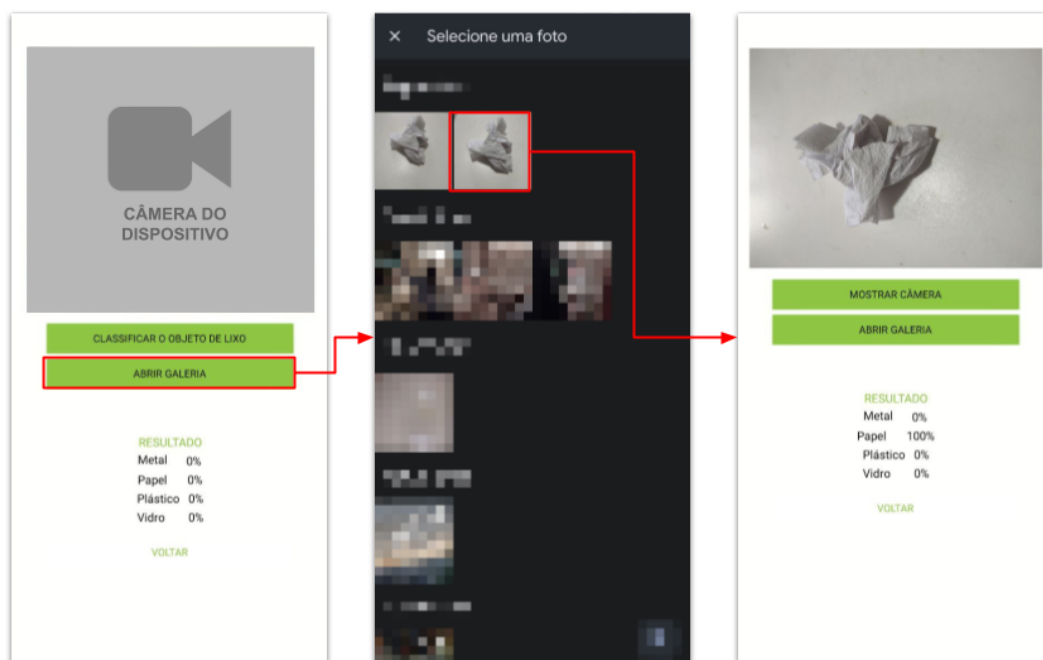
Figura 37 – Blocos lógicos para a classificação de imagens da galeria do dispositivo



Fonte: Autor.

As telas resultantes da implementação desta funcionalidade são mostradas pela Figura 38.

Figura 38 – Telas do aplicativo exemplo mostrando o fluxo de classificação de imagens da galeria

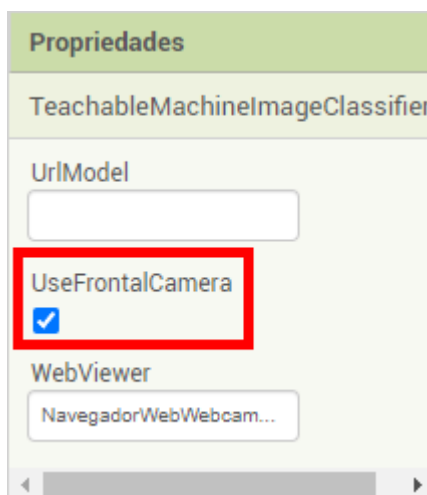


Fonte: Autor.

### 5.2.3 Uso da câmera frontal do dispositivo

Para habilitar a câmera frontal do dispositivo, é necessário ticar a propriedade *UseFrontalCamera* da extensão, como mostra a Figura 39.

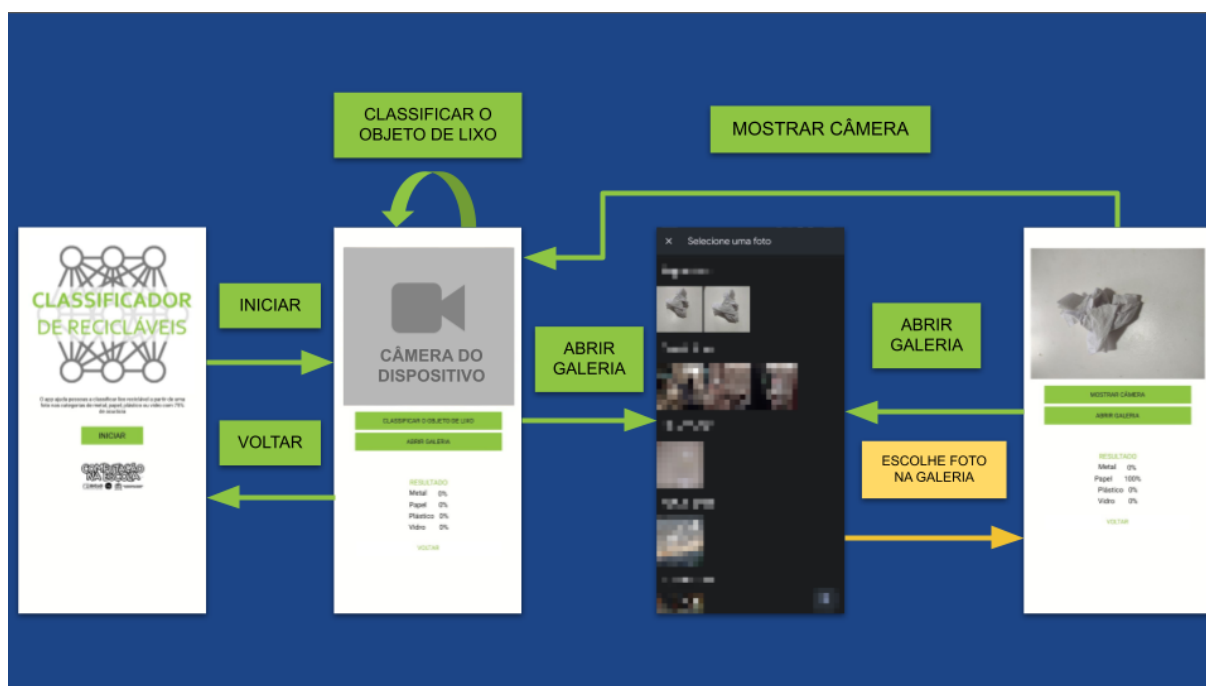
Figura 39 – Seleção da propriedade *UseFrontalCamera* da extensão TMIC



Fonte: <http://ai2.appinventor.mit.edu/>

O fluxo da versão final do aplicativo exemplo é mostrado na Figura 40.

Figura 40 – Fluxograma do aplicativo exemplo



Fonte: Autor.



Com isto, concluiu-se o desenvolvimento do aplicativo exemplo da evolução da extensão TMIC. Assim, exportou-se o projeto no formato .aia para ser incluído no material didático.

## 6 CONCLUSÃO

No presente trabalho, desenvolveu-se uma evolução da extensão TMIC, implementando novas funcionalidades que ampliam sua possibilidade de uso. Primeiramente, elaborou-se a fundamentação teórica em relação ao treinamento e exportação de modelos de classificação de imagens com o Google Teachable Machine, e a criação de extensões para o MIT App Inventor (O1). A partir do levantamento do estado da arte (O2), concluiu-se que o TMIC ainda é a única extensão para o MIT App Inventor que possibilita a importação de modelos de classificação de imagem treinados no Google Teachable Machine, porém apresentando limitações. No desenvolvimento da evolução da extensão TMIC (O3), foram desenvolvidas alterações para permitir que a extensão carregue modelos de classificação de imagem a partir de arquivos locais armazenados no dispositivo, classifique imagens armazenadas na galeria de fotos do dispositivo e utilize a câmera frontal do dispositivo para a classificação de imagens. Por fim, desenvolveu-se também um aplicativo exemplo, mostrando as novas funcionalidades do TMIC, além de um conjunto de slides para ser usado como material didático para ensinar o uso da nova evolução da extensão (O4).

Visto que o TMIC é, no momento deste trabalho, a única extensão disponível para o MIT App Inventor que possibilita a importação de modelos de classificação de imagem treinados no Google Teachable Machine, e que as alterações feitas nesta nova versão ampliam suas possibilidades de uso, este trabalho proporciona a alunos mais alternativas para exercer conhecimentos de IA, *Machine Learning* e classificação de imagens, podendo utilizar qualquer câmera do dispositivo para classificar imagens, classificar fotos da sua galeria, tiradas previamente, e classificar imagens mesmo sem a possibilidade de conexão com a internet.

Entre as dificuldades encontradas no decorrer do trabalho está a padronização de um local de armazenamento para os arquivos do modelo de classificação de imagens dentro do dispositivo, visto que este altera caso o usuário esteja depurando seu projeto por meio de um arquivo compilado *.apk* ou usando o aplicativo *AICompanion*. Dificuldades com esta geram a dependência de outras extensões, como o *ASDDownloader*. Outra dificuldade foi a realização da internacionalização dos blocos da extensão, mantendo a descrição dos mesmos apenas em inglês.

Para futuros trabalhos, recomenda-se desenvolver uma versão do TMIC capaz de mover arquivos do armazenamento interno do dispositivo para o diretório específico do aplicativo sem a necessidade de outras extensões e com seus blocos lógicos internacionalizados.

## REFERÊNCIAS

APPINVENTOR-EXTENSIONS: Source Code of extensions published for MIT app inventor. [S.l.: s.n.], out. 2014. Disponível em:

<https://github.com/mit-cml/appinventor-extensions>.

ARTIFICIAL Intelligence Image Classification app 2. [S.l.]: TM Software, Inc., mai.

2021. Disponível em: <https://appinventor.tmssoftwareinc.com/en/archives/2437>.

BAILER-JONES, Coryn AL; GUPTA, Ranjan; SINGH, Harinder P. An introduction to artificial neural networks. **arXiv preprint astro-ph/0102224**, 2001.

BELLMAN, R. **An Introduction to Artificial Intelligence: Can Computers Think?**

[S.l.]: Boyd & Fraser Publishing Company, 1978. ISBN 9780878350667. Disponível em: <https://books.google.com.br/books?id=84xQAAAAAAAJ>.

BHATIA, Nikhil et al. **Using transfer learning, spectrogram audio classification, and MIT app inventor to facilitate machine learning understanding**. 2020. Diss. (Mestrado) – Massachusetts Institute of Technology.

CARNEY, Michelle et al. Teachable machine: Approachable Web-based tool for exploring machine learning classification. In: EXTENDED abstracts of the 2020 CHI conference on human factors in computing systems. [S.l.: s.n.], 2020. P. 1–8.

CHAMMA, Willian Douglas Sbitkowski; BATISTELLA, Danielli;

CRISIGIOVANNI, Enzo Luigi; SILVA VICTORINO, Heloísa da;

LIMA, Vanderlei Aparecido de. Aprendizado de máquina aplicado em imagens de satélite para classificação de telhados Machine learning applied to satellite imagery for rooftop classification. **Brazilian Journal of Development**, v. 7, n. 7, p. 72558–72576, 2021.

CHARNIAK, Eugene. **Introduction to artificial intelligence**. [S.l.]: Pearson Education India, 1985.

CORRY, Kyle. **ML4K-ai-extension: Use machine learning in AppInventor, with easy training using text, images, or numbers through the Machine Learning for Kids website**. [S.l.: s.n.], jul. 2018. Disponível em:

<https://github.com/kylecorry31/ML4K-AI-Extension>.

FINIZOLA, Antonio; RAPOSO, Ewerton; PEREIRA, Maelso; GOMES, Wesclley; ARAÚJO, Ana; SOUZA, Flávia. O ensino de programação para dispositivos móveis utilizando o MIT-App Inventor com alunos do ensino medio. In: ANAIS do XX Workshop de Informática na Escola. Dourados: SBC, 2014. P. 337–341. DOI: 10.5753/cbie.wie.2014.337.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep learning**. [S.l.]: MIT press, 2016.

GRESSE VON WANGENHEIM, Christiane; HAUCK, Jean CR; PACHECO, Fernando S; BERTONCELI BUENO, Matheus F. Visual tools for teaching machine learning in K-12: A ten-year systematic mapping. **Education and Information Technologies**, Springer, v. 26, n. 5, p. 5733–5778, 2021.

GRESSE VON WANGENHEIM, Christiane; MARQUES, Livia S; HAUCK, Jean C R. **Machine Learning for All – Introducing Machine Learning in K-12**. [S.l.]: SocArXiv, ago. 2020. DOI: 10.31235/osf.io/wj5ne.

HADDAWAY, Neal Robert; COLLINS, Alexandra Mary; COUGHLIN, Deborah; KIRK, Stuart. The role of Google Scholar in evidence reviews and its applicability to grey literature searching. **PloS one**, Public Library of Science San Francisco, CA USA, v. 10, n. 9, e0138237, 2015.

HAUGELAND, John. **The very idea**. [S.l.]: Cambridge, MA: MIT Press, 1985.

JANIESCH, Christian; ZSCHECH, Patrick; HEINRICH, Kai. Machine learning and deep learning. **Electronic Markets**, Springer, v. 31, n. 3, p. 685–695, 2021.

JING, Meng. **China looks to school kids to win the global AI race**. 2018. Disponível em: <https://www.scmp.com/tech/china-tech/article/2144396/china-looks-school-kids-win-global-ai-race>. Acesso em: 31 mar. 2021.

KARAN. **ASD downloader extension - download and manage files in ASD**. [S.l.: s.n.], mai. 2023. Disponível em: <https://community.appinventor.mit.edu/t/free-asd-downloader-extension-download-and-manage-files-in-asd/84771>.

KOVÁCS, Z.L. **Redes Neurais Artificiais**. [S.l.]: Livraria da Física, 2002. ISBN 9788588325142.

KURZWEIL, Ray; RICHTER, Robert; KURZWEIL, Ray; SCHNEIDER, Martin L. **The age of intelligent machines**. [S.l.]: MIT press Cambridge, 1990. v. 580.

LAWRENCE, Ainsley. **Machine Learning & How It Affects Our Daily Lives**. 2019. Disponível em: <https://www.illinoisscience.org/2019/11/how-machine-learning-affects-our-daily-lives/>. Acesso em: 31 mar. 2021.

MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, p. 115–133, 1943.

NILSSON, Nils J. **Artificial intelligence: a new synthesis**. [S.l.]: Morgan Kaufmann, 1998.

OLIVAS, Emilio Soria; GUERRERO, Jos David Mart; MARTINEZ-SOBER, Marcelino; MAGDALENA-BENEDITO, Jose Rafael; SERRANO, L et al. **Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques: Algorithms, methods, and techniques**. [S.l.]: IGI global, 2009.

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. Florianópolis, SC, 2022.

ÖZGÜN, Ceyhun. **Home**. [S.l.: s.n.], fev. 2019. Disponível em: <https://github.com/ceyhunozgun/awsAIServicesAppInventorExtension/wiki>.

PATTON, Evan W; TISSENBAUM, Michael; HARUNANI, Farzeen. MIT app inventor: Objectives, design, and development. **Computational thinking education**, Springer Singapore, p. 31–49, 2019.

PETERSEN, Kai; FELDT, Robert; MUJTABA, Shahid; MATTSSON, Michael. Systematic mapping studies in software engineering. In: 12TH International Conference on Evaluation and Assessment in Software Engineering (EASE) 12. [S.l.: s.n.], 2008. P. 1–10.

POKRESS, Shaileen Crawford; VEIGA, José Juan Dominguez. MIT App Inventor: Enabling personal mobile computing. **arXiv preprint arXiv:1310.2830**, 2013.

POOLE, David I; GOEBEL, Randy G; MACKWORTH, Alan K. **Computational intelligence**. [S.l.]: Oxford University Press New York, 1998. v. 1.

RICH, E.; KNIGHT, K. **Artificial Intelligence**. [S.l.]: McGraw-Hill, 1991. (Artificial Intelligence Series). ISBN 9780071008945.

RUSSELL, Stuart J. **Artificial intelligence a modern approach**. [S.l.]: Pearson Education, Inc., 2010.

SANTOS, Julio Cezar Alves dos. **Classificação de imagens dermatoscópicas com machine learning para a detecção de melanoma**. 2020. Diss. (Mestrado) – Universidade Católica de Brasília.

TANG, Danny; UTSUMI, Yuria; LAO, Natalie. Pic: A personal image classification webtool for high school students. In: PROCEEDINGS of the 2019 IJCAI EduAI Workshop. [S.l.: s.n.], 2019.

WINSTON, Patrick Henry. **Artificial intelligence**. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1992.

# **Apêndices**

## **APÊNDICE A – CÓDIGO FONTE**

O código fonte do trabalho pode ser acessado em <https://codigos.ufsc.br/gqs/tmic>



## **APÊNDICE B – ARTIGO DA MONOGRAFIA**

A seguir, encontra-se o artigo do presente trabalho em formato SCB.

# Evolução do Desenvolvimento de uma Extensão para a Implementação de Modelos de *Machine Learning* em Aplicativos com o App Inventor

Rodrigo de Mello Garcia<sup>1</sup>, Christiane Gresse von Wangenheim<sup>1</sup>, Jean C. R. Hauck<sup>1</sup>

<sup>1</sup>Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis / SC, Brasil

rodrigo.mello.garcia@grad.ufsc.br, c.wangenheim@ufsc.br, jean.hauck@ufsc.br

**Abstract.** *This paper presents a proposal of evolution of TMIC (Teachable Machine Image Classifier), an extension for MIT App Inventor capable of loading and implementing image classification models trained on Google Teachable Machine. The new version described in this paper brings new features to the extension, allowing it to load image classification models from local files located on the mobile device where the application is installed, classify images in the device's photo gallery, and the use of the device's front camera for image classification.*

**Keywords:** *Machine Learning, Google Teachable Machine, MIT App Inventor, Extension*

**Resumo.** *Este artigo apresenta uma proposta de evolução do TMIC (Teachable Machine Image Classifier), uma extensão para o MIT App Inventor capaz de carregar e implementar modelos de classificação de imagem treinados no Google Teachable Machine. A nova versão descrita neste trabalho traz novas funcionalidades à extensão, permitindo-a carregar modelos de classificação de imagens a partir de arquivos locais localizados no dispositivo móvel onde a aplicação está instalada, classificar imagens presentes na galeria de fotos do dispositivo e a utilização da câmera frontal do dispositivo para a classificação de imagens.*

**Palavras-chave:** *Machine Learning, Google Teachable Machine, MIT App Inventor, Extensão.*

## 1. Introdução

*Machine Learning* é uma área de conhecimento em que se busca ensinar a uma máquina o reconhecimento de padrões por meio da apresentação de modelos exemplos, permitindo à máquina gerar conhecimento sobre estes mais rapidamente do que implementando manualmente regras de conhecimento [Janiesch et al. 2021].

Dentre as principais aplicações do *Machine Learning* está a classificação de imagens, que consiste em alimentar uma máquina com entradas de imagens rotuladas, ensinando-a e tornando-a capaz de classificar imagens desconhecidas e ainda não rotuladas baseando-se em seus aprendizados. A classificação de imagens pode ser usada em diversas áreas, como na medicina, na classificação de imagens dermatoscópicas para a detecção de melanoma [Santos 2020], ou na agrimensura, na classificação de telhados

para planejamento e cadastro urbano [Chamma et al. 2021]; entre outras. Assim, *Machine Learning* é uma área de conhecimento já presente e atuante no cotidiano da sociedade [Lawrence 2019].

O impacto das áreas de IA e *Machine Learning* resulta também em reações do mercado de trabalho de tecnologia da informação e, conseqüentemente, nos sistemas de educação internacionais, que buscam formar profissionais para suprir as demandas desse mercado. Diante disso, países como a China já anunciaram programas de ensino de IA para estudantes do ensino médio, buscando ser a liderança nesta área até o ano de 2030 [Jing 2018].

Embora o ensino sobre IA e *Machine Learning* nas instituições de ensino superior seja presente, ele também se mostra necessário na educação básica. No Brasil, a computação passou a ser tratada como um tema escolar a partir de 2010. O currículo proposto pela Sociedade Brasileira de Computação prevê, na educação de nível médio, o ensino de conceitos como *Machine Learning* e IA apenas com caráter introdutório, evitando o aprofundamento e contrariando estratégias tomadas pelos demais países. Assim, *Machine Learning* ainda não é um tópico abordado de forma geral no ensino médio no Brasil [Gresse von Wangenheim et al. 2020].

Diante desta demanda, estão surgindo esforços para trazer esses e outros tópicos de computação para jovens em seus últimos anos de estudos antes da faculdade, como é o caso dos cursos de *Machine Learning* da iniciativa Computação na Escola, do INCoD/INE/UFSC, que visa ensinar conceitos básicos de *Machine Learning* e suas aplicações, guiando o aluno à criação e implantação de modelos de reconhecimento de imagens no MIT App Inventor.

Nesse estágio educacional, tipicamente ferramentas visuais são utilizadas para ensinar o desenvolvimento de modelos de *Machine Learning* [Gresse von Wangenheim et al. 2021]. Uma destas ferramentas é o Google Teachable Machine, uma plataforma web para o ensino de *Machine Learning* e *Deep Learning* por meio da criação de modelos de classificação de imagens. Em 2020, o Google Teachable Machine foi utilizado em mais de duzentos países, gerando mais de 125 mil modelos de classificação. Assim, o Google Teachable Machine se mostra uma ferramenta popular, proporcionando uma oportunidade para qualquer interessado criar seu próprio modelo de classificação sem nenhum conhecimento aprofundado de *Machine Learning* ou de programação [Carney et al. 2020].

Outro exemplo de ambiente visual baseado em blocos é o App Inventor, uma plataforma online que ensina os conceitos básicos de programação de modo intuitivo e motivador, permitindo aos seus usuários programarem aplicativos para dispositivos móveis [Finizola et al. 2014]. Nesse contexto, o App Inventor também permite a criação de extensões que podem posteriormente serem implementadas nos aplicativos desenvolvidos na plataforma. Na área de classificação de imagens com modelos de *Machine Learning*, exemplos de extensões para o MIT App Inventor são o PIC, uma extensão que permite aos usuários treinarem modelos de classificação de imagens por meio de uma plataforma própria e usá-los em aplicativos criados no App Inventor [Tang et al. 2019], e o TMIC, que permite o uso de modelos treinados no Google Teachable Machine hospedados na nuvem [Oliveira 2022].

Em relação ao TMIC, identificou-se a necessidade de melhorias desta extensão, apresentadas por Oliveira como sugestões de trabalhos futuros, como o suporte para importação de modelos de *Machine Learning* exportados do Google Teachable Machine em outros formatos, acesso à câmera frontal para capturar imagens, entre outros. Vê-se, portanto, a oportunidade de implementar uma evolução desta extensão que abranja mais possibilidades de uso da mesma, removendo suas limitações atuais e implementando as sugestões de trabalhos futuros do autor.

O objetivo geral deste trabalho é criar uma nova versão da extensão TMIC com as funcionalidades adicionais de carregar modelos de classificação de imagens a partir de arquivos locais do modelo de *Machine Learning* treinado, classificar imagens da galeria do dispositivo e utilizar a câmera frontal do dispositivo. Assim, espera-se aprimorar a extensão existente, expandindo suas capacidades e fornecendo recursos adicionais que permitam aos usuários uma maior flexibilidade e interação com a tecnologia de classificação de imagens utilizando o MIT App Inventor. O material didático desenvolvido busca fornecer um guia claro para auxiliar os usuários no uso efetivo da nova versão do TMIC, facilitando assim o processo de aprendizado e aplicação prática dessa tecnologia.

## **2. Análise de Contexto e Definição da Evolução**

A extensão TMIC tem como objetivo permitir aos desenvolvedores importar modelos de classificação de imagem treinados no Google Teachable Machine para seus projetos no MIT App Inventor. O público alvo da extensão são alunos entre 10 e 14 anos que não possuem conhecimento prévio sobre *Machine Learning* e que estejam realizando o curso "Machine Learning para Todos!" da iniciativa Computação na Escola, para que possam aplicar seus conhecimentos recém adquiridos [Oliveira 2022]. A partir desta proposta, professores, estudantes e interessados poderão treinar seus próprios modelos de classificação de imagem na Google Teachable Machine e utilizá-los para o desenvolvimento de aplicativos no MIT App Inventor.

Atualmente, para utilizar um modelo treinado desta forma, a extensão prevê que o modelo esteja hospedado na nuvem pela própria Google Teachable Machine, e que o link para o modelo seja fornecido para a extensão por meio de suas propriedades. Uma vez que a extensão tenha acesso ao modelo treinado, ela estará apta a classificar imagens da câmera traseira do aplicativo e fornecer os resultados de sua classificação.

Por tratar-se de uma evolução do TMIC, o público alvo e a proposta da evolução desenvolvida neste trabalho mantêm-se os mesmos da extensão original, com o objetivo de apenas aumentar as possibilidades de implementação da mesma, reduzindo as limitações identificadas.

O objetivo da evolução, portanto, é implementar estas melhorias no TMIC, expandindo as possibilidades de uso da extensão original. Para possibilitar a classificação de imagens sem conexão à internet, a extensão deve passar a permitir o carregamento do modelo de classificação de imagens a partir de arquivos locais disponibilizados por desenvolvedores durante o desenvolvimento do aplicativo. Para a utilização da câmera frontal e permitir a classificação de imagens da galeria do dispositivo, será necessário desenvolver novos blocos lógicos e propriedades para a extensão.

Os requisitos funcionais da evolução da extensão TMIC são mostrados na Tabela 1.

**Tabela 1. Requisitos Funcionais**

ID	Descrição
<b>RF01</b>	A extensão deve possibilitar a importação de arquivos de modelos de classificação de imagens treinados no Google Teachable Machine que permita sua execução sem conexão com a internet.
<b>RF02</b>	A extensão deve possibilitar a execução do modelo de <i>Machine Learning</i> importado para a classificação de imagens.
<b>RF03</b>	A extensão deve permitir a utilização da câmera frontal do dispositivo móvel para captura de imagens.
<b>RF04</b>	A extensão deve solicitar ao usuário a permissão para utilização da câmera frontal do do dispositivo ao tentar abri-la.
<b>RF05</b>	A extensão deve permitir, por meio do modelo de classificação de imagens treinado no Google Teachable Machine e importado pelo usuário, a classificação de imagens obtidas pela câmera frontal do dispositivo.
<b>RF06</b>	A extensão deve disponibilizar os resultados da classificação de imagem realizada pelo modelo de <i>Machine Learning</i> 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.
<b>RF07</b>	A extensão deve permitir usar uma foto salva no celular para realizar a classificação, possibilitando também um pré-processamento da foto (cortar, rotacionar etc.)

Os requisitos não funcionais são mostrados na Tabela 2.

**Tabela 2. Requisitos Funcionais**

ID	Requisito	Descrição
<b>RNF01</b>	<i>Back end</i> da extensão desenvolvida em Java.	O <i>back end</i> da extensão deve ser desenvolvido na linguagem de programação Java utilizando o <i>framework</i> de desenvolvimento do App Inventor.
<b>RNF02</b>	<i>Front end</i> da extensão desenvolvida em Javascript	O <i>front end</i> da extensão deve ser desenvolvido na linguagem de programação Javascript, utilizando também a linguagem de marcação HTML e a linguagem de estilização CSS.
<b>RNF03</b>	Sistema operacional Android	A extensão deve ser implementada em aplicativos para o sistema operacional Android.
<b>RNF04</b>	Independência de conexão à internet.	A extensão deve permitir que o aplicativo desenvolvido classifique imagens sem necessidade de conexão com a internet.
<b>RNF05</b>	Internacionalização	Os blocos de programação da extensão devem estar disponíveis em português do Brasil e inglês

A extensão TMIC foi desenvolvida baseando-se na extensão PIC, que é dividida em *back end* e *front end*. Conseqüentemente, o TMIC possui uma arquitetura semelhante.

O desenvolvimento da evolução do TMIC procurou preservar a arquitetura da extensão original, realizando apenas alterações nos arquivos existentes para que os requisitos da evolução fossem contemplados.

Considerando que a origem dos arquivos do modelo de classificação de imagem é considerada na arquitetura da extensão, e que os requisitos funcionais da evolução demandam o uso de arquivos locais para o carregamento do modelo, a única alteração na arquitetura é a inclusão da possibilidade dos arquivos do modelo de classificação de imagens serem arquivos locais, como mostra a Figura 1.

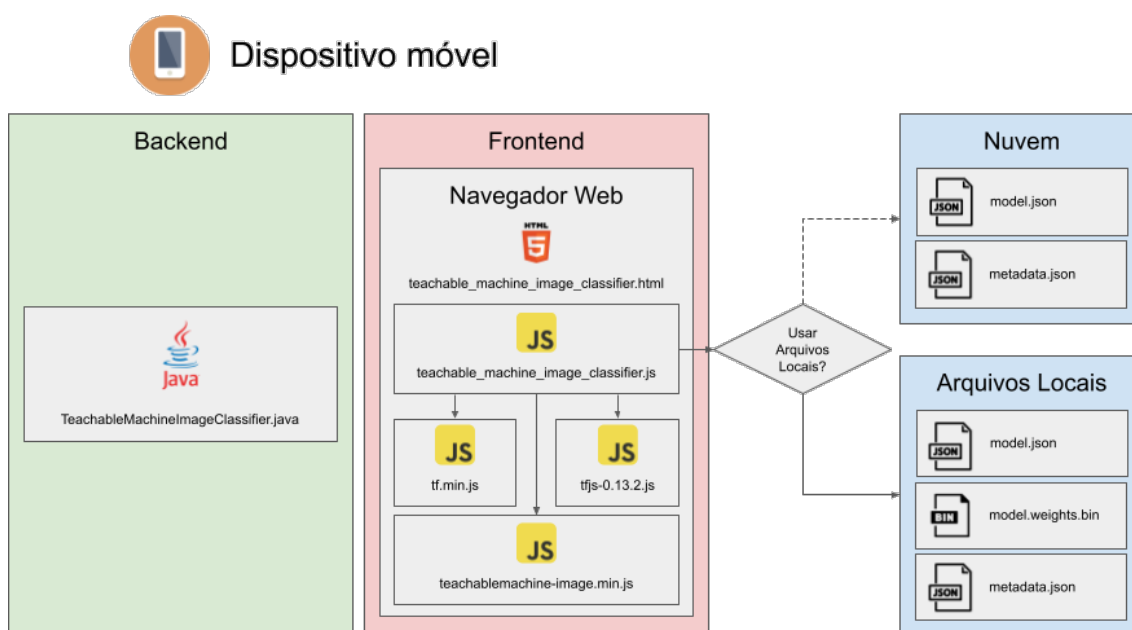


Figura 1. Arquitetura da evolução da extensão

### 3. Desenvolvimento

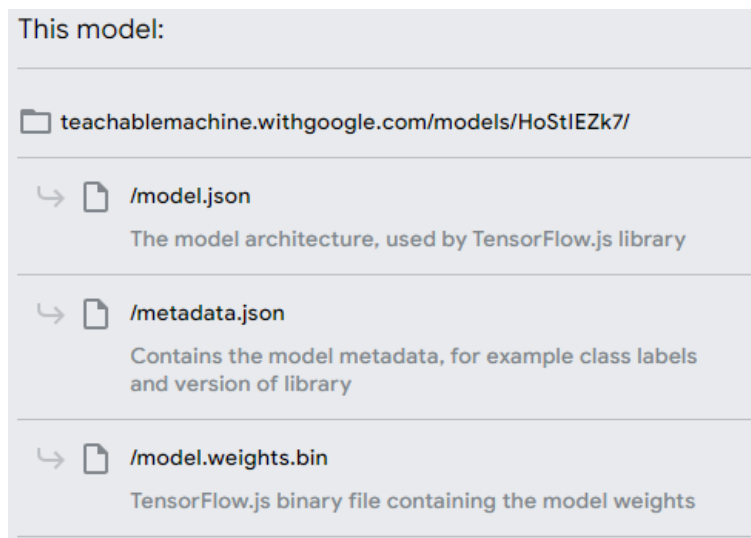
#### 3.1. Uso de arquivos locais para carregamento do modelo de classificação

Atualmente, a extensão original TMIC utiliza a biblioteca *Teachable Machine Library* para carregar o modelo de classificação de imagens. A partir da URL fornecida pelo desenvolvedor na propriedade `UrlModel`, a função `classifyVideoData` do arquivo do *front end* `teachable_machine_image_classifier.js` deriva as duas URLs necessárias para a função `load` carregar o modelo de classificação de imagem.

A biblioteca *Teachable Machine Library* também possui uma função para carregar o modelo a partir de arquivos locais chamada `loadFromFiles`, que recebe como parâmetros três arquivos que compõem o modelo:

- **model.json:** a arquitetura do modelo, usado pela biblioteca *TensorFlow.js*.
- **metadata.json:** contém os metadados do modelo, como as classes do modelo.
- **model.weights.bin:** arquivo binário que contém os pesos do modelo.

Ao hospedar um modelo de classificação de imagem na rede, o Google Teachable Machine também disponibiliza na mesma página estes três arquivos para serem baixados em um painel mostrado pela Figura 2.



**Figura 2.** Painel para *download* dos arquivos do modelo de classificação de imagem, disponível na página do Google Teachable Machine do modelo.

Para o contexto da evolução da extensão TMIC, espera-se que o desenvolvedor disponibilize estes arquivos previamente para o aplicativo para que a extensão carregue o modelo de classificação de imagens. Para isso, optou-se em usar o próprio painel Mídia do MIT App Inventor, visto que é algo nativo da própria plataforma e que desenvolvedores já estão familiares.

Uma vez que os arquivos sejam carregados pelo painel Mídia do MIT App Inventor, é necessário acessá-los por meio do *front end*. Idealmente, estes arquivos devem ser acessados de forma discreta, dispensando quaisquer ações do usuário final, o que traz um problema: por questões de segurança, Javascript não consegue acessar arquivos locais sem permissão do usuário, geralmente sendo feito por meio de um componente HTML *input* ou semelhantes. A solução, portanto, foi criar novos arquivos com o construtor *File*, passando o conteúdo dos arquivos originais para os arquivos criados.

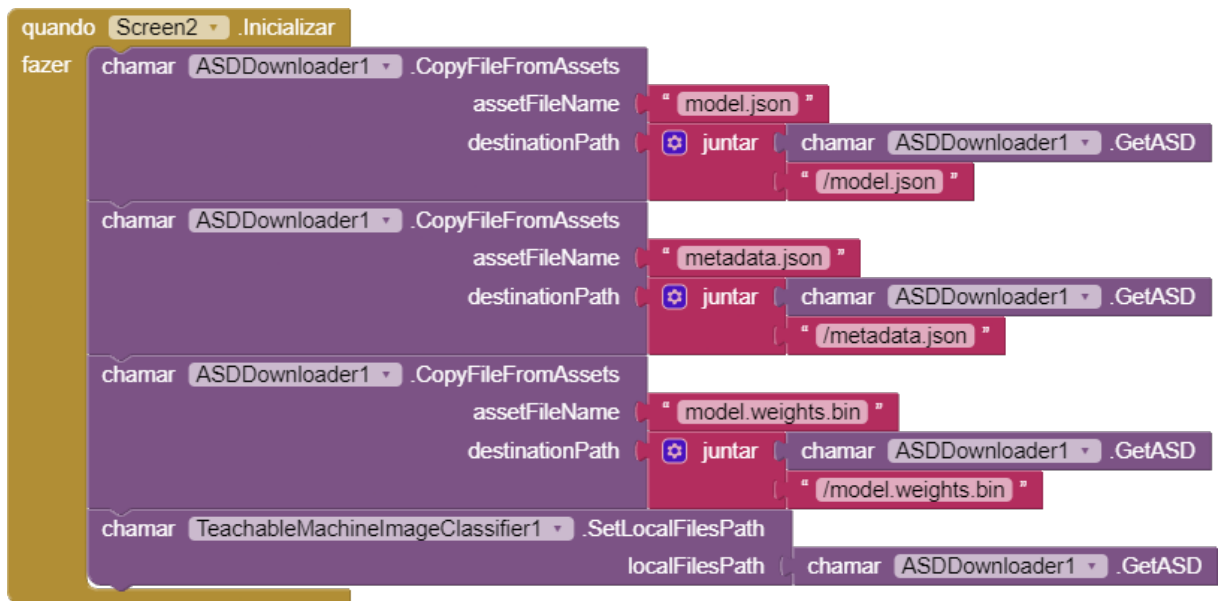
É necessário, então, acessar os arquivos do modelo de classificação por meio do *back end* (ou seja, por meio da classe *TeachableMachineImageClassifier.java*). Para isso, foram criadas as funções *readModelFile*, *readMetadataFile* e *readWeightsFile* na classe *JsonObject*, que é a classe do *TeachableMachineImageClassifier.java* que contém as funções acessíveis no *front end*. Cada uma destas funções lê um dos três arquivos e retorna o conteúdo do mesmo codificado em Base64, para garantir a integridade do conteúdo entre as linguagens Java e Javascript.

No *front end*, cria-se três novos arquivos homônimos aos arquivos originais, que são então passados como parâmetro para a função *loadFromFiles* da biblioteca *Teachable Machine Library*. Os parâmetros para a criação dos arquivos são:

- O conteúdo do arquivo original, retornado pela função do *JsonObject*, decodificado de volta para binário (*base64ToBin*).
- O nome do novo arquivo, que deve necessariamente ser "model.json", "metadata.json" ou "model.weights.bin".
- O tipo MIME do arquivo, que para os arquivos JSON é "application/json" e para o

arquivo binário é "application/octet-stream".

Se o desenvolvedor estiver depurando seu aplicativo com a ajuda do *AICompanion*, os arquivos carregados no painel Mídia já estarão disponíveis na pasta específica do aplicativo (App Specific Directory - ASD). No caso do aplicativo compilado (.apk), os arquivos da Mídia estão no armazenamento interno do aplicativo, sendo necessário movê-los para a pasta específica do aplicativo. A extensão *ASDDownloader* [Karan 2023] é capaz de fazer isso com a combinação de seus blocos lógicos *CopyFileFromAssets* e *GetASD*, como mostra a Figura 3. É necessário também informar para a extensão TMIC o caminho para o diretório específico do aplicativo por meio do bloco lógico *SetLocalFilePath*.



**Figura 3. Copiando os arquivos locais do armazenamento interno para o diretório específico do aplicativo com o *ASDDownloader*.**

Ao atribuir um valor para o bloco lógico *SetLocalFilePath*, a extensão define como verdadeiro uma variável interna "useLocalFiles", que é quem define como a extensão carrega o modelo de classificação de imagens.

Como o caminho para o diretório específico do *AICompanion* e do aplicativo compilado é diferente (o primeiro termina em ".../files/assets/" enquanto que o outro termina apenas em ".../files"), criou-se também um bloco *GetAICompanionLocalFilePath*, que retorna o caminho para o diretório específico do *AICompanion*. Assim, para ler arquivos locais e depurar o projeto com o *AICompanion*, é necessário este bloco como parâmetro para o bloco lógico *SetLocalFilePath*.

### 3.2. Classificação de arquivos da galeria do dispositivo

Atualmente, a extensão original TMIC classifica apenas a imagem mostrada no momento pela câmera do dispositivo.

Na classe *TeachableMachineImageClassifier.java*, foi criada a função *readImageFiles*, que, assim como as funções que lêem os arquivos que compõem o modelo de



classificação de imagem, recebe como parâmetro o caminho para o arquivo de imagem a ser lido e retorna o conteúdo do mesmo, codificado em Base64.

Foi criado o bloco lógico *classifyImageData*, que recebe como parâmetro o caminho para a imagem a ser classificada. A extensão *ImagePicker* do MIT App Inventor abre a galeria do dispositivo e retorna o caminho para a imagem selecionada. Desta forma, utilizando estes dois blocos, como mostra a Figura 4, desenvolvedores podem possibilitar a um usuário classificar uma imagem da galeria do dispositivo.

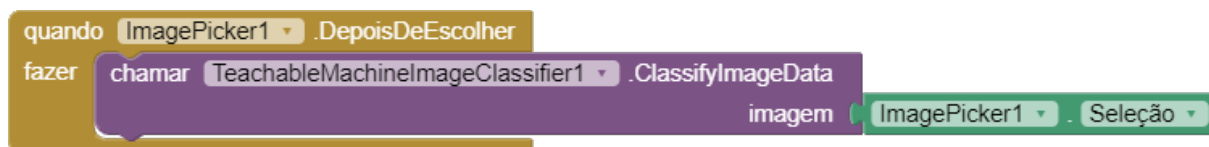


Figura 4. Blocos lógicos do *ImagePicker* e *classifyImageData*.

Por fim, é necessário permitir à função *classifyVideoData*, do arquivo do *front end teachable\_machine\_image\_classifier.js*, saber quando classificar a imagem da câmera e quando classificar a imagem da galeria. Para isso, criou-se uma variável interna *isClassifyingFromGallery*, que é definida como "verdadeiro" quando o bloco *ClassifyImageData* é chamado, e como "falso" quando o bloco *ClassifyVideoData* é chamado.

### 3.3. Uso da câmera frontal do dispositivo

Atualmente, a extensão original TMIC renderiza a câmera traseira do aplicativo no componente NavegadorWeb por meio da biblioteca *Teachable Machine Library*. Na função *webcam.setup*, um valor pode ser atribuído para a propriedade *facingMode*. Caso nenhum valor seja atribuído, é usado o valor padrão "environment", o que resulta no uso da câmera traseira do dispositivo. Caso o valor "user", seja atribuído, a câmera frontal passa a ser utilizada.

Por fim, criou-se uma propriedade *UseFrontalCamera* na classe *TeachableMachineImageClassifier.java*, que é usada no arquivo *teachable\_machine\_image\_classifier.js* do *front end* para atribuir o valor do parâmetro *facingMode* da função *webcam.setup*.

## 4. Testes da Evolução

Implementadas as novas funcionalidades, realizou-se uma série de testes para averiguar o correto funcionamento das mesmas. Para todos os testes, definiu-se o valor da propriedade *UseFrontalCamera* como verdadeiro, com o objetivo de testar o funcionamento do acesso à câmera frontal. Em todos os testes, a imagem da câmera frontal foi mostrada com sucesso.

Primeiramente, configurou-se os blocos *SetLocalFilePath* e *GetAICompanionLocalFilePath* para testar se a extensão lê arquivos locais enquanto se está depurando o projeto com a ajuda do *AICompanion*. Também foi atribuído um valor nulo para a propriedade *UrlModel* (campo deixado vazio), para garantir que a classificação só pudesse ser proveniente de arquivos locais. O aplicativo conseguiu realizar a classificação da imagem com sucesso, comprovando que o *AICompanion* conseguiu carregar o modelo através dos arquivos locais fornecidos.

Em seguida, configurou-se os blocos para testar se a extensão, enquanto compilada em um arquivo .apk, lê arquivos locais. Ainda com valor nulo atribuído para a propriedade *UrlModel* e o dispositivo em "modo avião", que impede qualquer conexão com a internet, o aplicativo conseguiu realizar a classificação da imagem com sucesso, comprovando que o aplicativo compilado conseguiu carregar o modelo através dos arquivos locais fornecidos. Tanto o AICompanion quanto o aplicativo compilado conseguiram realizar a classificação de uma imagem da galeria.

Por fim, realizou-se, utilizando a evolução da extensão, um teste das funcionalidades da extensão original, com o objetivo de garantir que nenhuma delas foi comprometida no desenvolvimento do presente trabalho. Como resultado, verificou-se que a evolução do TMIC ainda é possível de utilizar a câmera traseira do celular e carregar o modelo de classificação de imagens através da *Url* fornecida na propriedade *UrlModel*.

## Referências

- Carney, M. et al. (2020). Teachable machine: Approachable web-based tool for exploring machine learning classification. In *Extended abstracts of the 2020 CHI conference on human factors in computing systems*, pages 1–8.
- Chamma, W. D. S., Batistella, D., Crisigiovanni, E. L., da Silva Victorino, H., and de Lima, V. A. (2021). Aprendizado de máquina aplicado em imagens de satélite para classificação de telhados machine learning applied to satellite imagery for rooftop classification. *Brazilian Journal of Development*, 7(7):72558–72576.
- Finizola, A., Raposo, E., Pereira, M., Gomes, W., Araújo, A., and Souza, F. (2014). O ensino de programação para dispositivos móveis utilizando o mit-app inventor com alunos do ensino medio. In *Anais do XX Workshop de Informática na Escola*, pages 337–341, Porto Alegre, RS, Brasil. SBC.
- Gresse von Wangenheim, C., Hauck, J. C., Pacheco, F. S., and Bertoneceli Bueno, M. F. (2021). Visual tools for teaching machine learning in k-12: A ten-year systematic mapping. *Education and Information Technologies*, 26(5):5733–5778.
- Gresse von Wangenheim, C., Marques, L. S., and Hauck, J. C. R. (2020). Machine learning for all – introducing machine learning in k-12.
- Janiesch, C., Zschech, P., and Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, 31(3):685–695.
- Jing, M. (2018). China looks to school kids to win the global ai race.
- Karan (2023). Asd downloader extension - download and manage files in asd.
- Lawrence, A. (2019). Machine learning & how it affects our daily lives.
- Oliveira, F. P. d. (2022). Tmic-uma extensão do app inventor para a implantação de modelos de ml voltados a classificação de imagens treinados no teachable machine.
- Santos, J. C. A. d. (2020). Classificação de imagens dermatoscópicas com machine learning para a detecção de melanoma. Master's thesis, Universidade Católica de Brasília.
- Tang, D., Utsumi, Y., and Lao, N. (2019). Pic: A personal image classification webtool for high school students. In *Proceedings of the 2019 IJCAI EduAI Workshop*.