Aplicativo com reconhecimento de imagem para deficientes visuais

Pedro Angelo Catalini
Pontifícia Universidade Católica de Campinas, Brasil
e-mail: pedro.ac@puccampinas.edu.br
Fernando Ernesto Kintschner
Pontifícia Universidade Católica de Campinas, Brasil e-mail: fek@puc-campinas.edu.br
Denise Helena Lombardo Ferreira Pontifícia Universidade Católica de Campinas, Brasil e-mail: lombardo@puc-campinas.edu.br

Resumo— A inclusão de pessoas com especificidades na sociedade tem se tornado um grande desafio. A visão é um elemento extremamente relevante na relação com os objetos, com o espaço e com os outros e a sua ausência pode afetar negativamente a qualidade de vida. A detecção de objetos pode ser aplicada em diversas formas, como por exemplo, carros autônomos, indústria 4.0, controle de pessoas, entre outros. Tendo em vista o número elevado de deficientes visuais no Brasil, esta pesquisa tem como objetivo a aplicação de técnicas de visão computacional com o intuito de apresentar uma proposta de um aplicativo de reconhecimento de imagens para deficientes visuais. O Sistema de Informação utiliza o Método de Gestão de Projetos Ágeis baseado em Scrum da Engenharia de Software. A solução final utiliza um modelo com os pesos da rede quantizado, no formato inteiro de 8 bits. Isso faz com que os modelos utilizados sejam extremamente leves, com tamanho em disco de menos de 5Mb. A partir das técnicas utilizadas foi possível constuir um aplicativo de reconhecimento de imagens que despertou o interesse de diversos usuários, tendo em vista o elevado número de downloads do aplicativo.

Palavras-chave— Aplicativo; Deficiência visual; Machine learning

I. INTRODUÇÃO

Segundo o Instituto Brasileiro de Geografia e Estatística, no Brasil há 6,5 milhões de deficientes visuais, dentre eles, 582 mil são cegos [1]. Entretanto, esse número só tende a aumentar pelo fato de que a distribuição de pessoas cegas nas faixas etárias ser desigual, isto é, quanto mais velho, maior é a chance de adquirir cegueira. Em geral, 80% das pessoas cegas no mundo são maiores de 50 anos [2]. Tendo em vista que a população idosa acima de 60 anos deve dobrar no Brasil até 2048 [3], provavelmente a demanda por tecnologias assistivas também deverá aumentar.

A tecnologia assistiva no Brasil muitas vezes é sinônimo de alto custo, como por exemplo os óculos OrCam MyEye [4], um dispositivo dotado de câmeras acopladas em óculos que tem como objetivo auxiliar o deficiente visual em tarefas do dia-a-dia, como por exemplo ler textos, identificar rostos e objetos. No Brasil essa tecnologia continua inacessível, considerando que 46% das pessoas com deficiência recebem menos que um salário mínimo [5].

Os smartphones modernos contam com softwares que permitem a sua utilização por deficientes visuais, como leitores de tela e navegação por gestos, o que implica na procura por novos usuários com deficiência para realizar tarefas do dia-a-dia.

Contudo, o crescente interesse de usuários com deficiência visual pelos smartphones gera uma demanda por tecnologias assistivas por meio de novos aplicativos. Esses aplicativos visam melhorar a qualidade de vida e proporcionar autonomia para o deficiente visual.

Inúmeros problemas enfrentados pelos deficientes visuais podem ser resolvidos com o auxílio de softwares, tais como identificação de objetos levando em consideração suas características visuais; localização de objetos no espaço físico; identificação do conteúdo de imagens recebidas pela internet; falta de audiodescrição em filmes e falta de integração entre tecnologias assistivas.

O problema abordado neste trabalho refere-se sobre a dificuldade de localização de objetos. Nesse sentido, implementou-se um aplicativo capaz de sinalizar para o usuário com deficiência visual a localização de objetos por meio de vibrações.

II. DESENVOLVIMENTO DO APLICATIVO

O aplicativo de reconhecimento de imagens foi desenvolvido para o sistema operacional Android [6] com a utilização de técnicas de visão computacional, sendo capaz de detectar 82 classes de objetos. Dessa forma foi utilizada a biblioteca de código aberto *TensorFlowLite* [7] que possibilita rodar modelos em formato para *TensorFlowLite* (tflite) em smartphones.

Dois modelos foram utilizados, o primeiro ssd_mobilenet_v1_1_metadata_1 [8] foi obtido no repositório público do TensorFlow, tendo o seu treinamento já realizado com o dataset COCO [9]. Esse modelo é responsável pela detecção de 80 classes de objetos presentes no aplicativo. O segundo modelo utilizado foi o ssd_mobilenet_v2_quantized_coco. Na sequência foi realizado o processo de *Fine Tuning* [10] para a detecção de dois novos objetos: tomada e interruptor. Também foi necessária a conversão desse modelo para formato tflite.

Ambos os modelos utilizam pesos da rede neural quantizados para um inteiro de 8 bits, por consequência, o modelo final ocupa um espaço em disco de menos de 5 Mb e

permite um processamento mais rápido quando comparado com outras técnicas.

III. ARQUITETURA DO SISTEMA

A arquitetura do sistema foi criada segundo o paradigma do baixo acoplamento e da alta coesão entre os módulos implementados. A Fig. 1 representa o diagrama de arquitetura concebido para o aplicativo, tendo como principal destaque os itens de complexidade que agregam relevância ao artefato computacional desenvolvido.

O diagrama da Fig. 1 descreve o processo de adição de novos objetos no aplicativo, sendo necessária a adição de novos modelos treinados que contemplam determinadas categorias de novos objetos. Cada modelo adicionado da ssd_mobilenet_v2_quantized_coco pode suportar até 1.000 classes de objetos diferentes [11].

O Aplicativo pode ser separado em dois principais componentes: interface gráfica acessível e núcleo de processamento da aplicação. A arquitetura do sistema também contempla a necessidade de um administrador, responsável pela criação de novos modelos treinados com o intuito de adicionar novos objetos na lista do aplicativo.

A interface gráfica é composta por módulos que comportam funcionalidades específicas, sendo as principais: seleção de objetos por comando de voz; seleção de objetos pela interface gráfica e seleção de configurações gerais. Todos esses itens fornecem uma interface gráfica acessível.

Os principais módulos que o administrador necessita para a geração de modelos treinados são: um banco de imagens e a biblioteca do Tensorflow que contempla programas para o treinamento do modelo.

Os principais itens da arquitetura que compõem o núcleo da aplicação são: banco de modelos; seleção do modelo e detecção de objeto com *TensorFlowLite*.

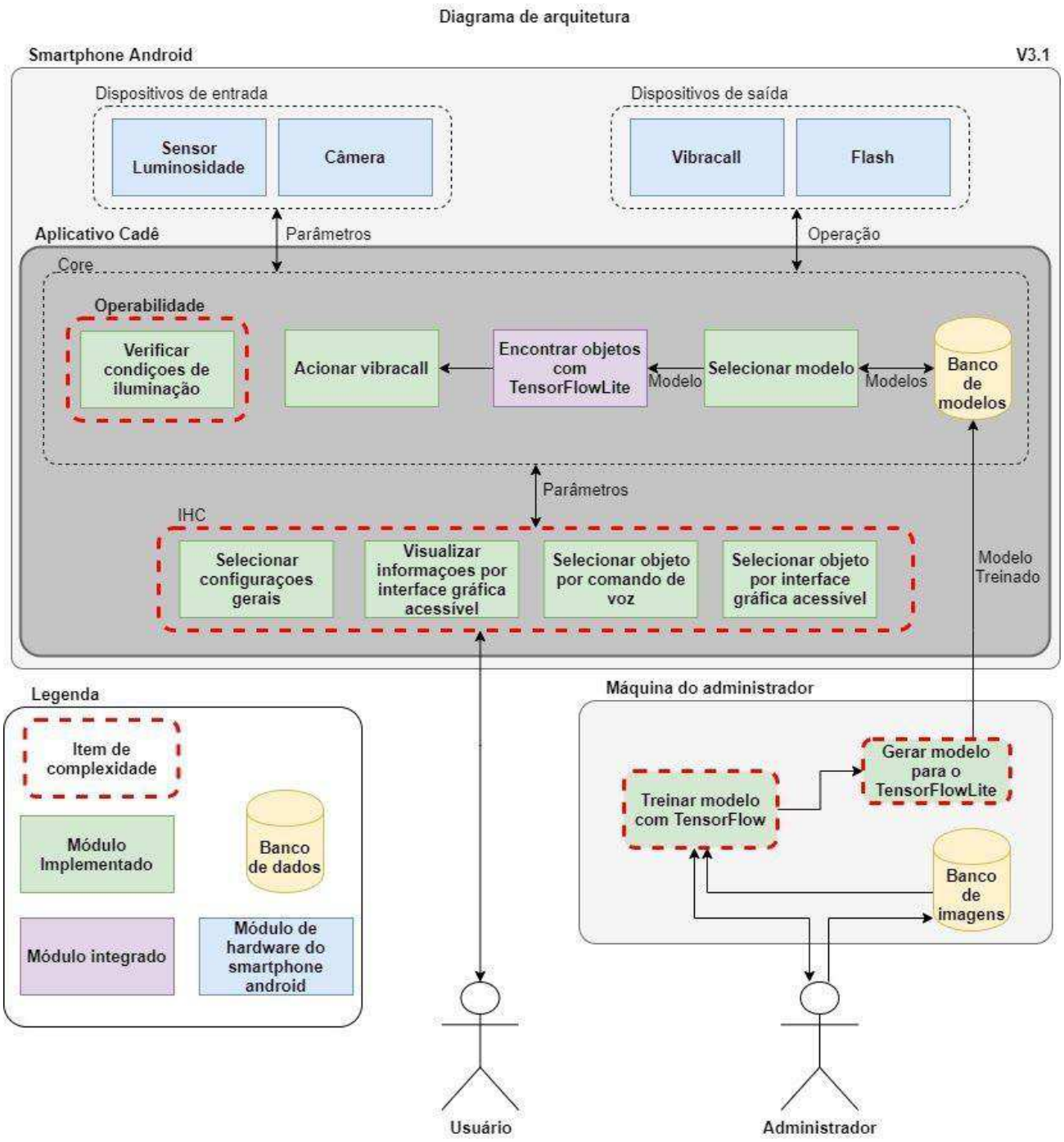

Fig. 1 – Diagrama de Arquitetura.
Fonte: Elaboração própria.

Banco de modelos

O banco de modelos pode comportar N modelos já treinados para o reconhecimento de determinadas classes de objetos. O modelo utilizado no aplicativo (ssd_mobilenet_v2_quantized_coco) pode suportar até 1.000 classes de objetos [11].

Seleção de modelo

A seleção do modelo é feita com base no objeto selecionado pelo usuário no aplicativo. Dessa forma é feita uma comparação em listas pré-definidas para descobrir qual é o modelo responsável pela detecção do objeto.

Detecção de objeto com *TensorFlowLite*

Após a seleção do modelo o mesmo é carregado utilizando chamadas para a biblioteca do *TensorFlowLite*. Uma nova classe é instanciada com todos os parâmetros necessários para a execução deste modelo.

Interface gráfica acessível

A criação da interface gráfica levou em consideração elementos de interface simples e textos grandes com o intuito de facilitar a utilização por meio de pessoas com baixa visão, isto é, pessoas com a acuidade visual inferior a 20/60 [2]. Para pessoas cegas foi inserido legenda oculta em todos os elementos gráficos para tornar possível a sua utilização por meio de leitores de tela.

A Fig. 2 apresenta um exemplo de interface do aplicativo.

Fig. 2 – Interface aplicativo cadê?
Fonte: Elaboração própria.

Treinar modelo com TensorFlow

Para o treinamento do modelo foi utilizado uma técnica chamada de *Fine Tuning* que permite a utilização de um modelo pré treinado. Isso possibilita um treino mais rápido possível de ser realizado com poucos exemplos de imagens, tendo em vista que as primeiras camadas da rede referentes a convolução não precisam ser treinadas, pois foram treinadas com o dataset COCO, capazes de extrair uma série de características padrões das classes de objetos.

Dataset

Foi criado um dataset com 138 imagens de tomada e interruptores com 22 imagens para teste e 116 imagens para treinamento.

Conversão do modelo para TensorFlowLite

Para conversão do modelo foi utilizado scripts da biblioteca do TensorFlow disponível em seu repositório online [12].

Modelo utilizado

O Modelo utilizado (ssd_mobilenet_v2_quantized_coco) trata-se da junção da técnica *Single Shot Detector* [13] com um detector de características MobileNet V2 [14].

IV. RESULTADOS

A avaliação do aplicativo foi realizada pelo monitoramento da publicação de avaliações e de comentários do aplicativo no Google Play [15].

As métricas para constituir os resultados deste estudo foram: avaliações dos usuários; número de usuários ativos e nota atribuída ao aplicativo.

Avaliações Funcionais

Como forma de avaliação funcional foi verificado se o treino realizado com os objetos adicionais conseguiu de forma satisfatória reconhecer os objetos.

Análise dos resultados

De acordo com as informações disponibilizadas através do console do google é possível notar a quantidade de usuários que permanecem ativos. Até o momento estão ativos 272 usuários, é possível notar uma tendência futura de crescimento do aplicativo.

A avaliação manteve-se acima da média ao comparar os aplicativos semelhantes. Até o momento a nota obtida é de 4.36 (escala de 0 a 5).

O treinamento realizado com o modelo conseguiu de forma satisfatória reconhecer os novos objetos adicionados (interruptor e tomada). Embora com poucos exemplos no banco de imagens foi possível diferenciar um interruptor de uma tomada. Contudo, é possível melhorar o reconhecimento dos objetos, utilizando técnicas como *Data Argumentation* [16] e a utilização de outros exemplos desses objetos no banco de imagens.

A Fig. 3 apresenta o padrão de detecção de objetos.

Fig. 3 – Demonstrativo de funcionamento com tomadas e interruptores. Fonte: Elaboração própria.

V. CONCLUSÃO

Esta pesquisa teve como principal objetivo a aplicação de técnicas de visão computacional com o intuito de promover e aprimorar habilidades do cotidiano de pessoas com deficiência visual.

A crescente procura pelo aplicativo tem demonstrado a sua utilidade para os usuários com deficiência visual.

REFERÊNCIAS

[1] Estatísticas da deficiência visual. Fundação Dorina Nowill para cegos. (2010). Disponível em: <https://www.fundacaodorina.org.br/a-fundacao/deficiencia-visual/estatisticas-da-deficiencia-visual/>. Acesso em: 27 set. 2020.

[2] As condições de saúde ocular no Brasil. (2019). Disponível em: <http://www.cbo.com.br/novo/publicacoes/condicoes_saude_ocular_brasil2019.pdf>. Acesso em: 27 set. 2020.

[3] Vander, O número de idosos no Brasil deve dobrar. (2018). Disponível em: <http://moa.org.br/blog/ate-2042-o-numero-de-idosos-no-brasil-deve-dobrar/>. Acesso em: 27 set. 2020

[4] OrCam MyEye Para pessoas cegas e com deficiência visual. (2020). Disponível em: <https://www.orcam.com/pt/myeye2/>. Acesso em: 27 set. 2020.

[5] IBGE: 46% das pessoas com deficiência recebem até 1 salário. Terra. (2012). Disponível em: <https://www.terra.com.br/economia/terra-da-diversidade/ibge-46-das-pessoas-com-deficiencia-recebem-ate-1-salario,de88b920548da310VgnCLD200000bbcceb0aRCRD.html>. Acesso em: 27 set. 2020.

[6] Android. Disponível em: <https://www.android.com/intl/pt-BR_br/>. Acesso em: 27 set. 2020.

[7] TensorFlowLite. Disponível em: <tensorflow.org/lite/guide>. Acesso em: 27 set. 2020.

[8] ssd_mobilenet_v1. Disponível em: <https://tfhub.dev/tensorflow/lite-model/ssd_mobilenet_v1/1/metadata/2>. Acesso em: 29 set. 2020.

[9] COCO dataset. Disponível em: <https://cocodataset.org/>. Acesso em: 27 set. 2020.

[10] Fine Tuning Disponível em: <https://www.tensorflow.org/guide/keras/transfer_learning>. Acesso em: 27 set. 2020.

[11] MobileNetV2 model architecture. Disponível em: <https://keras.rstudio.com/reference/application_mobilenet_v2.html>. Acesso em: 29 set. 2020.

[12] Repositório TensorFlow Disponível em: <https://github.com/tensorflow/tensorflow>. Acesso em: 27 set. 2020.

[13] Liu W. et al. (2016) SSD: Single Shot MultiBox Detector. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, v. 9905. Springer, Cham. https://doi.org/10.1007/978-3-319-46448-0_2

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L. Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks. (2018). IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018, pp. 4510-4520. doi: 10.1109/CVPR.2018.00474.

[15] Google Play. Disponivel em: <https://play.google.com/store/>. Acesso em: 27 set. 2020.

[16] Data augmentation. Disponível em: <https://www.tensorflow.org/tutorials/images/data_augmentation>. Acesso em: 27 set. 2020.