Reconhecimento Assistivo de Objetos em Android com TensorFlow Lite

Assistive Object Recognition on Android with TensorFlow Lite

João Pedro Piccino Marafiotti[1], **Tiago Henrique Trojahn**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP)
Câmpus São Carlos – São Carlos, SP – Brasil

joao.marafiotti@gmail.com, tiagotrojahn@ifsp.edu.br

Abstract. *This paper presents an Android application for assistive object recognition in educational environments using on-device TensorFlow Lite inference. YOLOv8n and YOLO26n models were evaluated considering detection metrics, inference time, execution on emulator and physical device, and real images. The application provides text, speech, and vibration feedback and includes single-image and semi-continuous CameraX modes. Results support the technical feasibility of the prototype but reveal limitations under uncontrolled conditions. No formal evaluation with blind or visually impaired users was conducted.*

Resumo. *Este artigo apresenta uma aplicação Android para reconhecimento assistivo de objetos em ambientes educacionais com inferência local em TensorFlow Lite. Modelos YOLOv8n e YOLO26n foram avaliados considerando métricas de detecção, tempo de inferência, execução em emulador e dispositivo físico e imagens reais. A aplicação fornece feedback por texto, voz e vibração e inclui modos de imagem única e semi-contínuo com CameraX. Os resultados sustentam a viabilidade técnica do protótipo, mas evidenciam limitações em condições não controladas. Não foi realizada avaliação formal com usuários cegos ou com deficiência visual.*

1. Introdução

O reconhecimento automático de objetos é uma área da visão computacional voltada à identificação e localização de elementos presentes em imagens. Essa tarefa é relevante em diferentes aplicações, pois permite que sistemas computacionais interpretem parte do conteúdo visual de uma cena e indiquem quais objetos estão presentes nela [Szeliski 2010, Redmon et al. 2016]. Quando aplicada a dispositivos móveis, essa tecnologia pode contribuir para soluções assistivas, especialmente para pessoas cegas ou com deficiência visual, que muitas vezes dependem de informações não visuais para compreender melhor o ambiente ao seu redor.

Em ambientes educacionais, como salas de aula, laboratórios, bibliotecas e espaços de estudo, a identificação de objetos comuns, como mesas, cadeiras, livros, mochilas, quadros, réguas, canetas e notebooks, pode apoiar atividades cotidianas e favorecer maior autonomia. Aplicações móveis com reconhecimento de objetos e feedback

acessível têm sido investigadas como alternativas de apoio a pessoas cegas ou com deficiência visual, especialmente quando os resultados são comunicados por áudio, vibração ou interfaces compatíveis com leitores de tela [Hisham et al. 2020, Ashiq et al. 2022, Klann et al. 2024].

Este artigo apresenta uma aplicação Android para reconhecimento assistivo de objetos em ambientes educacionais, com inferência local no próprio dispositivo móvel. A abordagem utiliza modelos de detecção de objetos exportados para TensorFlow Lite, de modo que o processamento da imagem ocorra no dispositivo, sem depender continuamente de um servidor externo. Essa decisão é relevante em aplicações assistivas, pois pode reduzir a dependência de conexão remota, evitar o envio constante de imagens para processamento externo e tornar o fluxo de uso mais adequado a situações móveis.

A contribuição deste trabalho não está na proposição de uma nova arquitetura de detecção de objetos, mas na integração e avaliação de um fluxo móvel assistivo com inferência local em um domínio educacional específico. Em relação às soluções discutidas na literatura, o trabalho combina, em uma mesma aplicação Android, execução *on-device*, feedback por texto, voz e vibração, comparação de modelos YOLO em TensorFlow Lite, avaliação em imagens do conjunto de dados e em capturas reais e um modo de detecção semi-contínua com CameraX. Assim, as principais contribuições são: (i) implementação de uma aplicação Android de reconhecimento de objetos com processamento local; (ii) integração de mecanismos de feedback acessível; (iii) avaliação técnica e funcional da aplicação em diferentes condições de execução e captura; e (iv) implementação e avaliação funcional de um modo semi-contínuo voltado a reduzir a dependência de uma única captura corretamente enquadrada.

2. Trabalhos Relacionados

Aplicações de visão computacional voltadas a pessoas cegas ou com deficiência visual têm sido estudadas em diferentes contextos, incluindo reconhecimento de objetos, detecção de obstáculos, navegação assistida, leitura de elementos específicos do ambiente e apoio à mobilidade. A revisão apresentada em [Hisham et al. 2020] aborda aplicações móveis de reconhecimento de objetos para pessoas com deficiência visual, destacando a importância de interfaces acessíveis e da comunicação dos resultados por meios não visuais. De forma semelhante, o estudo descrito em [Ashiq et al. 2022] propõe um sistema baseado em redes neurais convolucionais para reconhecimento e rastreamento de objetos, evidenciando o potencial de modelos de aprendizagem profunda em aplicações assistivas.

Outros trabalhos abordam tarefas mais específicas. O reconhecimento de cédulas de dinheiro em Real é investigado em [Sausen and Frozza 2022] como forma de auxiliar pessoas com deficiência visual, enquanto propostas de detecção de obstáculos e apoio à navegação são exploradas em [Almeida et al. 2024, Wang et al. 2024]. A revisão sistemática apresentada em [Klann et al. 2024] reforça que o desenvolvimento dessas soluções envolve não apenas o desempenho dos modelos, mas também aspectos de usabilidade, acessibilidade e adequação ao contexto de uso.

Além das aplicações assistivas, a literatura também apresenta modelos de visão computacional voltados a cenários com restrições computacionais. A família MobileNets foi proposta com foco em redes neurais eficientes para aplicações de visão em dispositivos com menor capacidade de processamento [Howard et al. 2017]. O modelo Single

Shot MultiBox Detector (SSD), por sua vez, tornou-se uma referência em detecção de objetos em uma única etapa [Liu et al. 2016]. Embora sejam abordagens anteriores às versões mais recentes da família YOLO, esses trabalhos são relevantes por evidenciarem a preocupação com eficiência, latência e viabilidade de execução em dispositivos com recursos limitados.

No contexto de aplicações móveis, um aplicativo Android com reconhecimento de imagem para pessoas com deficiência visual utilizando TensorFlow Lite e SSD-MobileNet foi apresentado em [Catalini et al. 2021]. Esse tipo de solução é relevante para este trabalho por aproximar reconhecimento visual, execução móvel e interação acessível.

Em relação aos trabalhos correlatos, este trabalho enfatiza a integração e a avaliação técnica de um protótipo Android com inferência local para objetos de ambientes educacionais. A aplicação combina modelos YOLO em TensorFlow Lite, feedback por texto, voz e vibração, avaliação em dispositivo físico e imagens reais e um modo semi-contínuo com CameraX. Assim, a contribuição está na integração e avaliação desse conjunto de recursos, e não na proposição de um novo modelo de visão computacional.

3. Metodologia

3.1. Arquitetura e fluxos da aplicação

A aplicação Android foi organizada em torno de dois fluxos principais de uso: reconhecimento por imagem única e detecção semi-contínua com câmera aberta. No fluxo de imagem única, o usuário captura uma foto ou seleciona uma imagem para teste. Em seguida, a imagem é pré-processada com *letterbox*, enviada ao modelo local em TensorFlow Lite, pós-processada e apresentada ao usuário por meio de texto, voz e vibração.

No fluxo semi-contínuo, a câmera permanece aberta dentro da aplicação por meio do CameraX [Android Developers 2026a], e quadros são analisados periodicamente. Cada quadro selecionado passa por extração periódica, pré-processamento, inferência local e geração de feedback acessível. Esse fluxo busca reduzir a dependência de uma única captura correta, permitindo que o enquadramento seja ajustado ao longo do tempo.

A Figura 1 apresenta os dois principais fluxos da aplicação: o reconhecimento por imagem única e o modo semi-contínuo com CameraX. Em ambos os casos, a inferência é realizada localmente no dispositivo, utilizando um modelo YOLO26n exportado para TensorFlow Lite.

Uma versão preliminar cliente-servidor evidenciou limitações relacionadas à dependência de conexão e ao envio externo de imagens, motivando a adoção da inferência local com TensorFlow Lite [TensorFlow 2026] na versão discutida neste artigo.

3.2. Conjunto de dados

O conjunto de dados utilizado foi o *Objects in the Classroom*, disponível na plataforma Kaggle [Putra 2026]. Sua escolha é justificada por conter objetos comuns em ambientes educacionais, como salas de aula, laboratórios e espaços internos de estudo. Essa delimitação permitiu avaliar e comparar a abordagem em um domínio educacional específico.

O conjunto contém 20 classes: *table*, *chair*, *whiteboard*, *bookshelf*, *clock*, *wall-magazine*, *trash-can*, *eraser*, *sharpener*, *pen*, *book*, *ruler*, *scissor*, *fan*, *laptop*, *remote-*

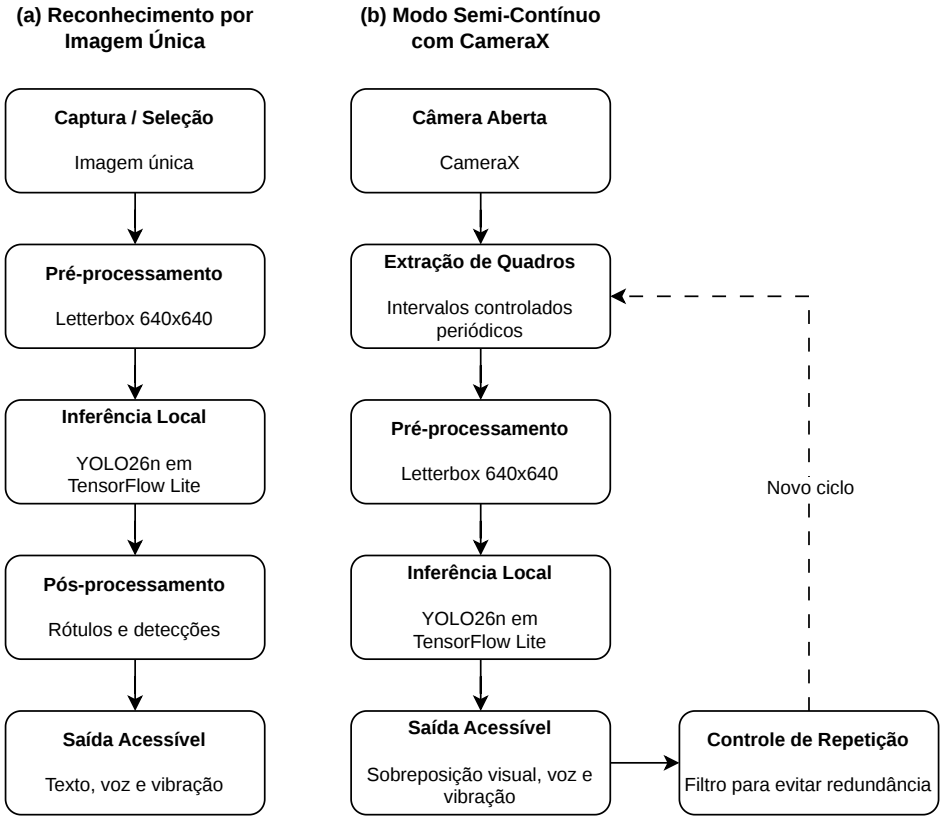


Figura 1. Fluxos principais da aplicação Android proposta: reconhecimento por imagem única (a) e modo semi-contínuo com CameraX (b).

control, *bag*, *pants*, *shoes* e *hat*. As classes foram mantidas em inglês durante o treinamento e traduzidas para português na interface do aplicativo, de modo que o usuário recebesse mensagens como “caneta”, “borracha”, “tesoura” ou “mochila”.

A Tabela 1 apresenta a divisão do conjunto de dados utilizada no treinamento e avaliação dos modelos. Todas as imagens analisadas possuem resolução de 640×640 pixels.

Tabela 1. Divisão do conjunto de dados utilizado.

Conjunto	Imagens	Resolução	Finalidade
Treinamento	3200	640×640	Ajuste dos modelos
Validação	640	640×640	Acompanhamento do treinamento
Teste	160	640×640	Avaliação final

Além da quantidade de imagens, foi verificada a distribuição das instâncias anotadas por classe. No total, foram identificadas 5273 instâncias anotadas. Essa verificação foi importante para confirmar que todas as classes estavam representadas no conjunto utilizado. As fotos reais capturadas posteriormente foram usadas apenas para testes funcionais e qualitativos da aplicação, não para retreinamento dos modelos.

3.3. Modelos avaliados

Foram avaliadas as variantes leves YOLOv8n e YOLO26n. O YOLOv8n foi adotado como referência inicial por sua ampla documentação e suporte no ecossistema Ultralytics [Ultralytics 2026b], enquanto o YOLO26n foi incluído como alternativa mais recente [Jocher et al. 2026]. As variantes "n" foram escolhidas pelo menor custo computacional, característica relevante para execução em dispositivos móveis. A comparação busca investigar o comportamento dos modelos no mesmo domínio e após exportação para TensorFlow Lite, e não estabelecer superioridade geral entre as arquiteturas.

A etapa inicial de ajuste do YOLOv8n foi realizada a partir de pesos pré-treinados, com imagens de entrada de 640×640 pixels, durante 50 épocas, utilizando o framework Ultralytics em ambiente Google Colab com GPU Tesla T4. O YOLO26n também foi ajustado ao mesmo conjunto de dados, seguindo configuração equivalente de entrada e treinamento, permitindo comparar os modelos no mesmo domínio antes da exportação para TensorFlow Lite.

Para a execução local, os modelos foram exportados para TensorFlow Lite. Foram integrados ao aplicativo três modelos: YOLO26n Float32, YOLO26n Float16 e YOLOv8n Float32. A versão Float32 utiliza representação numérica em precisão simples, enquanto a versão Float16 utiliza meia precisão, reduzindo o tamanho do modelo e o uso de memória, embora possa exigir validação adicional quanto à estabilidade e ao desempenho em diferentes dispositivos [Google AI Edge 2026].

O YOLO26n Float32 foi adotado como modelo padrão final por apresentar uma relação adequada entre tempo de inferência, estabilidade e compatibilidade com os fluxos implementados no aplicativo. O YOLO26n Float16 foi mantido como alternativa experimental, e o YOLOv8n Float32 foi preservado como referência comparativa.

3.4. Aplicativo Android e recursos de acessibilidade

O aplicativo Android foi desenvolvido com foco em um fluxo simples de interação. No fluxo principal, o usuário tira uma foto e recebe o resultado por meio de texto, voz e vibração. No modo de teste, é possível selecionar imagens, trocar entre modelos locais e comparar seus resultados em diferentes cenários de avaliação. O código-fonte da aplicação está disponível publicamente em repositório GitHub.[1]

Foram integrados recursos de acessibilidade por texto, Text-to-Speech [Android Developers 2026b], vibração e descrições semânticas para leitores de tela. No fluxo baseado em imagem única, o detector utiliza limiar mínimo de confiança de 0,40. Detecções com confiança igual ou superior a 0,70 são tratadas como de maior confiança, enquanto valores iguais ou superiores a 0,40 e inferiores a 0,70 são comunicados como possíveis detecções. Nesse fluxo, múltiplas classes podem ser anunciadas quando pertencem à faixa de confiança considerada, e a ausência de reconhecimento seguro também é comunicada ao usuário.

O feedback tátil do fluxo de imagem única utiliza padrões distintos: uma vibração de 90 ms para detecções de maior confiança, duas vibrações de 70 ms separadas por 90 ms para possíveis detecções e uma vibração de 180 ms quando não há reconhecimento

[1] https://github.com/joaomarafiotti/blind-assistance-app/releases/tag/v1.0-ic-final

considerado seguro. A resposta falada é simplificada e não informa a porcentagem exata de confiança, enquanto a interface visual mantém informações como modelo utilizado, confiança e tempo aproximado de inferência.

No modo CameraX, foi adotada uma configuração específica de feedback, com limiar mínimo de 0,30 e três faixas: confiança baixa para valores iguais ou superiores a 0,30 e inferiores a 0,50, média para valores iguais ou superiores a 0,50 e inferiores a 0,80 e alta a partir de 0,80. Quando existem múltiplas detecções, apenas a de maior confiança é utilizada para o anúncio por voz e para a vibração. Os padrões táteis são, respectivamente, uma vibração de 40 ms para confiança baixa, duas vibrações de 50 ms separadas por 90 ms para confiança média e uma vibração de 70 ms para confiança alta. Na ausência de detecção, o CameraX atualiza apenas o status visual, sem emitir fala ou vibração.

3.5. Modo de detecção semi-contínua

Além do fluxo baseado em uma única foto, foi implementado um modo de detecção semi-contínua com CameraX. Esse modo foi motivado por uma limitação prática: em uma aplicação assistiva, um usuário cego pode ter dificuldade para enquadrar corretamente um objeto, manter o foco ou capturar a imagem em boas condições de iluminação.

No modo semi-contínuo, a câmera permanece aberta dentro do aplicativo e quadros são analisados periodicamente. Cada quadro selecionado é convertido para imagem manipulável, pré-processado com *letterbox*, enviado ao modelo YOLO26n Float32 em TensorFlow Lite e apresentado ao usuário por meio de sobreposição visual, voz e vibração.

O modo é denominado semi-contínuo porque não processa todos os quadros em tempo real. Uma nova análise é permitida aproximadamente a cada 3,5 segundos. Para reduzir feedback redundante, existe ainda um intervalo global mínimo de 4,5 segundos entre mensagens faladas e um intervalo de 10 segundos para que a mesma classe seja anunciada novamente. Esses valores são parâmetros de implementação do protótipo e não foram sistematicamente otimizados.

4. Resultados e Discussão

Esta seção apresenta os principais resultados obtidos nas avaliações da aplicação proposta. Foram consideradas métricas usuais de detecção de objetos, como precisão, revocação e *mean Average Precision* (mAP). Neste trabalho, mAP50 indica a mAP calculada com limiar de 0,50 de *Intersection over Union* (IoU), enquanto mAP50-95 considera a média de limiares de IoU entre 0,50 e 0,95, seguindo práticas comuns na avaliação de detectores de objetos [Ultralytics 2026a, Padilla et al. 2021].

4.1. Avaliação dos modelos

O modelo YOLOv8n foi ajustado ao conjunto de dados *Objects in the Classroom* e utilizado como referência inicial de comparação. Em seguida, o YOLO26n foi avaliado como alternativa para execução em dispositivo móvel. A Tabela 2 apresenta os resultados principais dos modelos avaliados nos formatos PyTorch e TensorFlow Lite. Os tempos de inferência correspondem às médias observadas em um dispositivo físico Samsung S25 FE.

Na implementação Android, o campo de tempo de inferência cronometra exclusivamente a chamada de execução do modelo pelo interpretador TensorFlow Lite, não incluindo *letterbox*, preparação do tensor de entrada, pós-processamento, sobreposição visual ou geração de feedback. Também não foram configurados explicitamente número de *threads* ou *delegate* no interpretador. Como o protocolo original das médias agregadas apresentadas na Tabela 2 não preservou informações suficientes sobre número de repetições e aquecimento, esses valores são tratados como tempos médios observados, e não como um *benchmark* rigorosamente padronizado.

Tabela 2. Comparação dos modelos avaliados.

Modelo	Precisão	Revocação	mAP50	mAP50-95	Tempo
YOLOv8n PyTorch	0,866	0,870	0,900	0,746	–
YOLO26n PyTorch	0,889	0,831	0,899	0,734	–
YOLOv8n TFLite Float32	0,897	0,850	0,874	0,715	137,2 ms
YOLO26n TFLite Float32	0,869	0,837	0,894	0,727	114,1 ms

No formato PyTorch, o YOLOv8n apresentou maior revocação e maior mAP50-95, enquanto o YOLO26n apresentou maior precisão e desempenho geral próximo. Após a exportação para TensorFlow Lite, o YOLO26n Float32 apresentou menor tempo médio de inferência e valores superiores de mAP50 e mAP50-95 em relação ao YOLOv8n TFLite Float32 nesta avaliação. Por esse motivo, o YOLO26n Float32 foi priorizado na integração com inferência local no aplicativo Android.

A versão YOLO26n Float16 também foi integrada como alternativa experimental para verificar a compatibilidade da execução local com representação em meia precisão. Em uma comparação funcional reduzida no dispositivo físico Samsung S25 FE, com apenas três objetos reais, os três modelos avaliados acertaram 3/3 casos. O YOLO26n Float32 apresentou confiança média de 92,7% e tempo médio de 138,7 ms; o YOLO26n Float16 apresentou 93,0% e 145,3 ms; e o YOLOv8n Float32 apresentou 91,0% e 194,3 ms. Devido ao tamanho reduzido dessa avaliação, esses resultados não constituem um *benchmark* suficiente para concluir superioridade entre Float32 e Float16. Por esse motivo, o YOLO26n Float16 é tratado apenas como alternativa experimental, enquanto o YOLO26n Float32 permaneceu como modelo padrão da aplicação.

4.2. Avaliações funcionais em emulador e dispositivo físico

Além das métricas dos modelos, foram conduzidas avaliações funcionais para verificar o comportamento da aplicação nos fluxos implementados. A inferência local foi avaliada em emulador Android, em dispositivo físico Samsung S25 FE e em um fluxo de captura real após a implementação do pré-processamento com *letterbox*. A Tabela 3 resume os principais resultados.

Para as avaliações funcionais, foi considerado "caso correto" quando a classe esperada foi detectada corretamente. A categoria "parcialmente correto" foi utilizada apenas quando o resultado não correspondia exatamente à classe esperada, mas apresentava uma detecção semanticamente relacionada ao alvo ou a uma parte relevante da cena. Por envolver interpretação qualitativa, essa categoria é apresentada separadamente e não é contabilizada como acerto na taxa de acerto simples, que constitui a medida principal

reportada. Quando indicado, uma segunda taxa incluindo o caso parcial é apresentada apenas como informação complementar.

Tabela 3. Resumo das avaliações funcionais da aplicação com inferência local.

Cenário	Casos	Corretos	Parciais	Erros	Sem det.	Taxa
Emulador Android	20	17	1	1	1	85% / 90%
Dispositivo físico	10	8	–	1	1	80%
Captura real após letterbox	7	3	–	1	3	42,9%

Essas avaliações possuem caráter funcional e exploratório e não devem ser interpretadas como estimativas estatísticas de acurácia ou generalizadas para outros dispositivos. Os tempos também representam valores aproximados de cenários distintos, e não um *benchmark* controlado entre ambientes equivalentes.

Na avaliação em emulador, a aplicação executou inferência local sem depender de servidor, alcançando 85% de taxa de acerto simples e 90% quando considerado o caso parcialmente correto. Os testes no dispositivo físico também confirmaram a viabilidade técnica da execução local em hardware real. A validação após o uso de *letterbox* confirmou o fluxo principal no dispositivo, mas também evidenciou limitações quando o sistema depende de uma única foto.

4.3. Avaliação com fotos reais

Para investigar o comportamento da aplicação em condições mais próximas de uso real, foram realizadas avaliações com fotos capturadas em dispositivo físico. A primeira avaliação utilizou fotos reais não controladas, com variação de iluminação, distância, fundo, ângulo e composição. A segunda avaliação utilizou fotos reais controladas, com melhor iluminação, maior centralização do objeto e menor quantidade de elementos ao redor. A Tabela 4 apresenta os resultados considerando apenas objetos pertencentes às classes do conjunto de dados.

Tabela 4. Avaliação com fotos reais pertencentes às classes do conjunto de dados.

Cenário	Casos	Corretos	Parciais	Erros	Sem det.	Taxa
Não controlado	16	5	1	4	6	31,25% / 37,5%
Controlado	11	8	–	3	0	72,7%

A condição não controlada apresentou taxa de acerto simples de apenas 31,25%, substancialmente inferior aos 72,7% obtidos nas fotos controladas. A diferença reforça um *gap* entre o conjunto de treinamento e condições mais próximas de uma captura cotidiana. Entre os fatores observados nas falhas estão iluminação, distância, tamanho do objeto, fundos complexos, enquadramento e objetos parcialmente cortados. Também ocorreram confusões entre categorias visualmente ou semanticamente próximas, como entre livro/caderno e *wall-magazine*, e entre mochila/boné e *shoes*. Assim, o desempenho no conjunto de teste não deve ser diretamente extrapolado para condições reais de utilização.

4.4. Teste funcional do modo CameraX

O modo CameraX foi implementado para reduzir a dependência de uma única foto, mantendo a câmera aberta e analisando quadros periodicamente conforme os intervalos descritos na metodologia. Cada observação registrada correspondeu a uma execução do ciclo de detecção, incluindo captura do quadro, inferência local e geração de feedback visual, falado e tátil. Uma detecção foi considerada semanticamente correta quando a classe retornada correspondia ao objeto-alvo presente na cena avaliada. A Tabela 5 apresenta o resumo do teste funcional.

Tabela 5. Teste funcional do modo CameraX.

Métrica	Resultado
Observações registradas	10
Detecções semanticamente corretas	10
Tempo médio de inferência aproximado	214,2 ms
Menor tempo observado	111 ms
Maior tempo observado	467 ms
Confiança média aproximada	60,9%

O teste do CameraX é uma avaliação funcional e qualitativa, e não uma estimativa ampla de acurácia. Os tempos reportados correspondem ao tempo de inferência do modelo, conforme definido na Seção 4.1, e não ao tempo fim-a-fim do ciclo CameraX. As 10 observações demonstraram o funcionamento do ciclo de câmera, inferência e feedback sem travamentos observados, mas a amostra reduzida impede generalizações. A confiança média de 60,9% também exige cautela, pois detecções incorretas comunicadas ao usuário podem produzir informação inadequada. O modo deve, portanto, ser entendido nesta etapa como uma demonstração funcional da redução da dependência de uma única captura.

5. Limitações

O conjunto utilizado possui apenas 20 classes e é proveniente de uma base pública internacional, podendo não representar plenamente ambientes educacionais brasileiros. Objetos fora dessas classes podem ser associados incorretamente a categorias conhecidas, e a queda para 31,25% de acerto simples em fotos não controladas evidencia o *gap* entre o domínio de treinamento e condições mais próximas do uso cotidiano.

As avaliações funcionais também possuem amostras reduzidas e foram executadas em apenas um dispositivo físico, o Samsung S25 FE. Portanto, os resultados não devem ser generalizados para outros aparelhos ou condições de uso. Também não foram realizadas análises detalhadas por classe ou matriz de confusão. Uma avaliação mais completa deveria ainda utilizar protocolo padronizado de latência, incluindo número de repetições, aquecimento, *threads*, *delegate* e delimitação de pré e pós-processamento.

Consumo de bateria, memória, aquecimento e uso prolongado não foram avaliados sistematicamente. Além disso, não houve validação formal com usuários cegos ou com deficiência visual; assim, os resultados demonstram viabilidade técnica e funcional do protótipo, mas não eficácia assistiva, usabilidade ou segurança em uso real.

Por fim, a aplicação deve ser entendida como ferramenta experimental de auxílio ao reconhecimento de objetos, e não como sistema de navegação ou substituto de recursos de mobilidade. O CameraX também não fornece orientação espacial ativa e ainda requer avaliações mais amplas e estratégias mais robustas de tratamento de confiança e múltiplas detecções.

6. Conclusão

Este trabalho apresentou uma aplicação Android para reconhecimento assistivo de objetos em ambientes educacionais com inferência local via TensorFlow Lite. A principal contribuição está na integração e avaliação técnica, em um mesmo protótipo móvel, de modelos YOLO *on-device*, feedback por texto, voz e vibração e um modo semi-contínuo com CameraX.

Os resultados demonstram a viabilidade técnica da execução local, mas também evidenciam limitações importantes em condições reais de captura. Em particular, as amostras reduzidas e a ausência de validação com usuários cegos ou com deficiência visual impedem conclusões sobre eficácia assistiva; o protótipo deve ser entendido como base técnica para estudos posteriores, e não como sistema de navegação.

Como trabalhos futuros, pretende-se ampliar e diversificar o conjunto de dados, realizar análises por classe e avaliações com usuários finais, testar diferentes dispositivos e investigar consumo de recursos. Também se pretende aprofundar a comparação entre Float32 e Float16 e aprimorar o CameraX quanto ao tratamento de confiança, múltiplas detecções e orientação de enquadramento.

Referências

Almeida, C., Filho, P., Rosal, I., Gonçalves, R., Aguilar, P., Nunes, M. S. M., and Martins, M. (2024). Proposta de tecnologia assistiva para auxiliar pessoas com deficiência visual na detecção de obstáculos. In *Anais do Seminário Integrado de Software e Hardware (SEMISH)*, pages 288–299.

Android Developers (2026a). CameraX overview. `https://developer.android.com/media/camera/camerax`. Acesso em: 31 maio 2026.

Android Developers (2026b). TextToSpeech. `https://developer.android.com/reference/android/speech/tts/TextToSpeech`. Acesso em: 31 maio 2026.

Ashiq, F., Asif, M., Ahmad, M. B., Zafar, S., Masood, K., Mahmood, T., Mahmood, M. T., and Lee, I. H. (2022). CNN-Based Object Recognition and Tracking System to Assist Visually Impaired People. *IEEE Access*, 10:14819–14834.

Catalini, P. A., Kintschner, F. E., and Ferreira, D. H. L. (2021). Aplicativo com reconhecimento de imagem para deficientes visuais. In *Brazilian Technology Symposium (BTSym)*.

Google AI Edge (2026). Post-training float16 quantization. `https://developers.google.com/edge/litert/conversion/tensorflow/quantization/post_training_float16_quant`. Acesso em: 31 maio 2026.

Hisham, Z. A. N., Faudzi, M. A., Ghapar, A. A., and Rahim, F. A. (2020). A systematic literature review of the mobile application for object recognition for visually impaired people. In *Proceedings of the 8th International Conference on Information Technology and Multimedia (ICIMU)*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861. Disponível em: https://arxiv.org/abs/1704.04861. Acesso em: 31 maio 2026.

Jocher, G., Qiu, J., Liu, M., Lyu, S., Akyon, F. C., and Kalfaoglu, M. E. (2026). Ultralytics YOLO26: Unified real-time end-to-end vision models. arXiv preprint arXiv:2606.03748. Disponível em: https://arxiv.org/abs/2606.03748. Acesso em: 5 jun. 2026.

Klann, D., Fernandes, A. M. R., Silva, E. A., and Parreira, W. D. (2024). Explorando algoritmos de visão computacional em tecnologias assistivas: uma revisão sistemática da literatura. *Journal of Health Informatics*, 16(Especial).

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 21–37. Springer.

Padilla, R., Passos, W. L., Dias, T. L. B., Netto, S. L., and da Silva, E. A. B. (2021). A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3):279.

Putra, A. K. (2026). Objects in the classroom. Kaggle dataset. https://www.kaggle.com/datasets/aryakrisnaputra/objects-in-the-classroom. Acesso em: 31 maio 2026.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788.

Sausen, F. S. and Frozza, R. (2022). Aplicativo para auxiliar pessoas com deficiência visual no reconhecimento de cédulas de dinheiro em real com a técnica de redes neurais artificiais. *Revista Brasileira de Computação Aplicada*, 14(3):1–16.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer, London.

TensorFlow (2026). TensorFlow Lite. https://www.tensorflow.org/lite. Acesso em: 31 maio 2026.

Ultralytics (2026a). Performance metrics deep dive. https://docs.ultralytics.com/guides/yolo-performance-metrics/. Acesso em: 31 maio 2026.

Ultralytics (2026b). YOLOv8 documentation. https://docs.ultralytics.com/models/yolov8/. Acesso em: 31 maio 2026.

Wang, W., Jing, B., Yu, X., Sun, Y., Yang, L., and Wang, C. (2024). YOLO-OD: Obstacle detection for visually impaired navigation assistance. *Sensors*, 24(23):7621.