VisionSpeaker: Aplicativo Android capaz de reconhecer objetos para auxiliar deficientes visuais

Leonardo Felipe Mendes Lopes[1], **Carlos Roberto dos Santos Junior**[1]

[1]Instituto Federal de Educação, Ciência e tecnologia - Campus Hortolândia (IFSP)
CEP 13183-250 – Hortolândia – SP – Brasil

leonardo.lopes@aluno.ifsp.edu.br[1], carlos.rsantos@ifsp.edu.br[1]

Abstract. *People with visual impairment may face problems to perform daily tasks, due to the struggle to identify near objects. Taking this situation into account, this paper presents VisionSpeaker, an Android application that identifies and speaks to the user the name of the object that the device's camera is being pointed to. A convolutional neural network was trained in order to build the artificial intelligence model used in this project. This project was applied to an university environment, focusing on aiding visually impaired students to make object recognition easier. After testing the application, it was identified that the model reached a performance rate of 67.66%.*

Resumo. *Pessoas com deficiência visual podem enfrentar problemas ao realizar tarefas do dia a dia causados pela dificuldade em identificar objetos próximos. Considerando esta situação, este artigo apresenta o VisionSpeaker, um aplicativo Android que identifica e fala ao usuário o nome do objeto que a câmera do dispositivo está sendo apontada. Foi realizado o treinamento de uma rede neural convolucional para a construção do modelo de inteligência artificial utilizado neste trabalho. Este projeto foi aplicado a um ambiente de universidade, focando em auxiliar estudantes com deficiência visual para facilitar o reconhecimento dos objetos. Após testes com o aplicativo, foi identificado que o modelo atingiu uma taxa de desempenho de 67,66%.*

1. Introdução

Deficiência visual é um dos maiores problemas de saúde no mundo. Ocorre quando alguma doença afeta o sistema visual e uma ou mais de suas funções da visão [OMS 2021a]. A visão de uma pessoa pode ser classificada de acordo com a acuidade visual, uma forma de medir o nível de detalhe que uma pessoa consegue ver um objeto a uma determinada distância, em relação a uma pessoa com visão normal [Babic e Junior 2020]. A acuidade visual é expressa por meio de uma fração, onde o numerador indica a distância em metros que uma pessoa consegue identificar detalhes de um objeto, e o denominador indica qual distância uma pessoa com visão normal poderia identificar o mesmo nível de detalhe [Babic e Junior 2020]. Existem diferentes categorias que são usadas para classificar esta condição: suave, moderada, severa e cegueira. Cada uma destas categorias ocorre quando a acuidade visual é menor que 6/12, 6/18, 6/60 e 3/60, respectivamente [OMS 2021b].

De acordo com o censo demográfico de 2010 realizado pelo IBGE [IBGE 2010], em que foram pesquisadas informações sobre moradores de aproximadamente 67,5 milhões de domicílios em todo o território nacional, cerca de 18,8% da população residente possui deficiência visual.

A Pesquisa de Informações Básicas Municipais (Munic) realizada com dados coletados referentes ao ano de 2019 pelo IBGE, indica que houve forte redução da quantidade de municípios no país que promovem programas, ações ou medidas para pessoas com deficiência, pois de 2014 a 2019 este percentual caiu de 85,1% para 61,7% [IBGE 2020].

Diante dos dados apresentados, é possível observar que existe a necessidade de tecnologias assistivas para deficientes visuais que possuam um custo acessível e facilidade em seu uso. Com objetivo de ajudar a suprir esta necessidade, foi desenvolvido um aplicativo Android [Google 2008] para auxiliar deficientes visuais a identificar objetos comuns que podem ser encontrados em uma universidade.

O aplicativo desenvolvido neste trabalho não se trata de um produto completo, ou seja, ainda não é adequado para ser utilizado por usuários reais, que possuem alguma deficiência visual. A partir do desenvolvimento do aplicativo foi possível testar uma metodologia de detecção de objetos, podendo ser reaproveitada no desenvolvimento de algum trabalho futuro que viabilize o uso por usuários reais ao elaborar uma interface mais adequada.

Para simplificar os testes e servir como base para futuros projetos, a interface do aplicativo foi projetada para funcionar de forma simples, bastando apenas abri-lo e apontar a câmera traseira do dispositivo para frente, assim todos os objetos que estão no campo capturado pela câmera e foram previamente inseridos na base de dados serão identificados, reproduzindo um áudio que informa o nome do objeto e a direção em que se encontra. Caso necessário, para melhor compreensão do áudio, pode-se utilizar fones de ouvido durante a execução do aplicativo.

O desenvolvimento deste trabalho envolveu a montagem de uma base de dados com imagens de alguns objetos de uma sala de aula. Após montada, a base de dados foi utilizada para fazer o treinamento de uma rede neural, criando um modelo de inteligência artificial que pode reconhecer os objetos selecionados. Para interface com o usuário, foi desenvolvido um aplicativo Android [Google 2008], que alimenta o modelo com as imagens capturadas pela câmera do dispositivo e processa seu retorno, informando por áudio ao usuário quais objetos foram detectados e a direção em que cada um se encontra, que pode ser à esquerda, à direita ou à frente.

O restante deste artigo está organizado da seguinte forma: a seção 2 mostra trabalhos similares que também utilizam visão computacional [Szeliski 2010]. A seção 3 apresenta alguns conceitos que foram utilizados para desenvolvimento deste trabalho, para melhor compreensão na leitura do artigo. A seção 4 apresenta detalhes sobre a metodologia utilizada. A seção 5 detalha aspectos sobre o desenvolvimento do trabalho. A seção 6 apresenta os resultados que foram levantados após testes, analisando métricas relevantes para avaliação do desempenho do aplicativo. A seção 7 apresenta a conclusão levantada após o desenvolvimento e testes. A seção 8 finaliza o artigo apresentando possíveis trabalhos futuros que podem ser desenvolvidos com base no trabalho descrito por este artigo.

2. Trabalhos Correlatos

Esta seção apresenta alguns trabalhos que foram utilizados como referência para o desenvolvimento deste trabalho. Estes trabalhos contribuíram para a escolha de alguns conceitos, tecnologias ou técnicas, que foram utilizados neste trabalho.

2.1. Um sistema treinável de detecção de pedestres

No trabalho de [Papageorgiou et al. 2000], foi desenvolvido um sistema de detecção de objetos que automaticamente aprende a detectar objetos de uma determinada classe. No caso desse projeto, foi aplicada a detecção de pedestres.

Como imagens de pedestres variam muito em relação à cor, textura e fundo, foi necessário observar diferenças de intensidade em pequenas regiões locais. Para isso, foi utilizado o *framework wavelets de Haar* [Mallat 1987].

Para a realização do treinamento, foi utilizada uma base de dados com imagens de pedestres, e dessas imagens foram escolhidas as características mais úteis para classificar pedestres. Para cada imagem, foram escolhidas manualmente 29 características.

O treinamento foi realizado utilizando um classificador de Máquina de Vetor de Suporte (SVM) [Vapnik 1995]. Foram utilizadas 1848 imagens de pedestres e 7189 imagens de cenas que não contêm pedestres, para que o classificador possa diferenciar cenas que contêm ou não pedestres.

Para verificar a performance do sistema de detecção, foi feita uma análise utilizando uma curva de Característica de Operação do Receptor (ROC), gráfico usado para analisar a taxa de verdadeiros positivos em relação a taxa de falsos positivos [Krzanowski e Hand 2009]. Para cada análise, foram feitas variações na cor das imagens, número de características e grau do polinômio do classificador. Pela análise da curva ROC [Krzanowski e Hand 2009], foi identificado que a performance é mais afetada pelo número de características utilizadas. O sistema que utilizou mais características teve uma performance superior, porém ficou mais lento para classificar as imagens.

O trabalho mostrou que a visão computacional [Szeliski 2010] possui diversas aplicações que podem auxiliar pessoas em certas tarefas. Na conclusão, são citadas algumas dessas aplicações, como sistemas de assistência automotiva, indexação em base de dados de vídeos e imagens, e vigilância.

Com o conhecimento do potencial que a visão computacional [Szeliski 2010] possui para tornar diversas tarefas mais eficientes, foi considerado que poderia ser utilizada para auxiliar deficientes visuais a identificar objetos utilizados em seu cotidiano.

2.2. Detecção eficiente de múltiplos objetos e navegação inteligente utilizando inteligência artificial, para deficientes visuais

No trabalho de [Joshi et al. 2020], foi desenvolvido um dispositivo que auxilia a navegação de pessoas com deficiência visual. Os objetos são detectados e informados ao usuário em tempo real. O dispositivo também é capaz de detectar informações de distância, utilizando um sensor de distância. Quando as imagens são capturadas pela câmera, é desenhado um retângulo em volta do objeto e o usuário é informado por meio de áudio sobre quais objetos foram capturados.

Foi montada uma base de imagens baseando-se em objetos relevantes no cotidiano de pessoas com deficiência visual. Foram geradas diversas imagens de cada objeto em diferentes condições, como variações em ângulos, rotação e iluminação. Foi utilizada a técnica de transferência de aprendizado [Yang et al. 2020], utilizando um modelo YOLOv3 [Redmon e Farhadi 2018]. Para cada classe de objetos foram coletadas 650 imagens, utilizando 150 para o teste e 500 para o treinamento.

O sistema foi montado utilizando um Processador Digital de Sinais (DSP) de placa única, microprocessador utilizado para processar sinais de áudio ou vídeo [Angoletta 2008]. No processador DSP [Angoletta 2008] foram conectados um sensor de distância, uma câmera e um fone de ouvido.

Após realizados os testes, foi identificado que esse sistema chegou a uma acurácia média de 95,19% para detecção de objetos. Foram realizados testes com 36 pessoas, sendo 20 deficientes visuais e 16 pessoas vendadas. Após o uso, as pessoas informaram que o dispositivo desenvolvido é confortável e fácil de ser utilizado.

A metodologia apresentada no artigo serviu como referência para a construção da metodologia deste trabalho. Assim como o trabalho de referência, foi desenvolvido um sistema que por meio de uma câmera, imagens são capturadas e em seguida são reproduzidos áudios que informam o deficiente visual quais são os objetos visíveis pela câmera.

Além disso, foi utilizado o conhecimento sobre técnicas que, segundo os autores são eficientes no reconhecimento de objetos, mesma tarefa realizada pelo aplicativo desenvolvido neste trabalho. Foram utilizadas as seguintes técnicas em comum:

- Augmentação de imagens: o trabalho mostrou que aumentar a variedade de condições em imagens, como rotação, brilho e mudança de ângulos pode contribuir positivamente para o desempenho do modelo, pois a base de dados é ampliada;
- Transferência de aprendizado [Yang et al. 2020]: mostrou que ao utilizar um modelo pré-treinado para reaproveitar seu aprendizado é possível contribuir para o aumento do desempenho do modelo;
- Redes neurais convolucionais [Goodfellow et al. 2016]: no trabalho, foi utilizada uma rede neural convolucional para treinamento do modelo. A utilização deste tipo de rede neural mostrou-se eficiente no reconhecimento de objetos.

2.3. *Third Eye*: um aplicativo para auxiliar deficientes visuais utilizando *Machine Learning*

No trabalho de [V.K et al. 2019], é proposto o aplicativo *Third Eye* para Android [Google 2008], destinado a deficiente visuais, com o objetivo de torná-los mais independentes em seu cotidiano.

O aplicativo possui três módulos: usuário com deficiência visual, cuidador e administrador.

O módulo do usuário com deficiência visual possui as seguintes funcionalidades:

- Reconhecimento de objetos: os objetos são capturados pela câmera do *smartphone* e é reproduzido em áudio qual objeto foi detectado por meio de fone de ouvido. Foi utilizado o TensorFlow [Abadi et al. 2015] para realizar o reconhecimento de objetos;
- Reconhecimento de texto: o texto é capturado pela câmera do *smartphone* e reproduzido em áudio por meio de fone de ouvido. Foi utilizada a tecnologia de Reconhecimento Óptico de Caracteres (OCR) [Eikvil 1993];
- Busca de objeto: quando o nome do objeto é falado pelo usuário, o aplicativo procura o objeto na proximidade e informa o usuário;
- Solicitação de localização: o usuário pode solicitar a sua localização atual por meio da fala. O aplicativo detecta qual é a localização e informa o usuário.

O módulo de cuidador é responsável por monitorar os usuários com deficiência visual. O cuidador pode se cadastrar no aplicativo e cadastrar diversos deficientes visuais para monitorar. É possível acompanhar a localização atual e receber um alerta quando o deficiente visual ultrapassa uma determinada área.

O módulo de administrador é responsável por controlar o sistema inteiro. Ele pode visualizar todas as informações sobre os cuidadores e os deficientes visuais cadastrados.

O objetivo deste aplicativo foi desenvolver uma aplicação de baixo custo para auxiliar deficientes visuais em atividades do cotidiano, como navegação, compras e detecção de texto.

O trabalho demonstrou viabilidade de implementar um aplicativo de reconhecimento de objetos que executa em um dispositivo Android [Google 2008], que nativamente possui ferramentas que tornam a implementação possível, como a câmera, que é utilizada para capturar as imagens, a funcionalidade *text-to-speech*, que converte texto em fala, e a capacidade de processar modelos de inteligência artificial, necessário para realizar os testes. Estas características indicam que é possível utilizar um dispositivo Android [Google 2008] para realizar testes com o modelo construído neste trabalho.

A utilização do TensorFlow [Abadi et al. 2015] para construção do modelo também indicou viabilidade para desenvolvimento deste trabalho, além de indicar compatibilidade com dispositivos Android [Google 2008] ao utilizar esta tecnologia.

Além disso, foi utilizada uma rede neural convolucional [Goodfellow et al. 2016] para treinamento. Os autores do trabalho afirmam ser a técnica mais popular para classificações de imagens com alto desempenho.

3. Referencial Teórico

Esta seção possui informações sobre alguns conceitos relacionados a este trabalho, que são importantes para o entendimento do artigo.

3.1. Visão Computacional

Visão computacional é a área da inteligência artificial que é responsável pelo treinamento de computadores para serem capazes de identificar o mundo visual, a partir de técnicas matemáticas. O objetivo da visão computacional é descrever imagens e reconstruir suas propriedades, como forma, iluminação e distribuição de cores [Szeliski 2010].

Atualmente a visão computacional possui diversas aplicações no mundo real, como reconhecimento de caracteres, inspeção de máquinas, medicina e dispositivos de segurança em veículos [Szeliski 2010].

Este conceito é fundamental para o desenvolvimento deste trabalho, pois a parte principal desenvolvida, que se trata do desenvolvimento da rede neural e treinamento do modelo de inteligência artificial, baseia-se em conceitos de visão computacional.

3.2. Redes Neurais Artificiais

Redes neurais artificiais são estruturas que simulam o funcionamento de neurônios de um cérebro humano no processo de aprendizagem de padrões [de Medeiros 2018].

Uma rede neural é composta por neurônios, que são organizados em camadas. Qualquer rede neural possui pelo menos uma camada de entrada e uma camada de saída, podendo também possuir um número indeterminado de camadas ocultas, situadas entre a entrada e a saída [de Medeiros 2018]. As camadas são interligadas por um grande número de conexões associadas a pesos, que armazenam o conhecimento representado no modelo e ponderam a entrada recebida por cada neurônio [de Brittos Valdati 2020]. A Figura 1 apresenta um exemplo de estrutura de uma rede neural artificial.

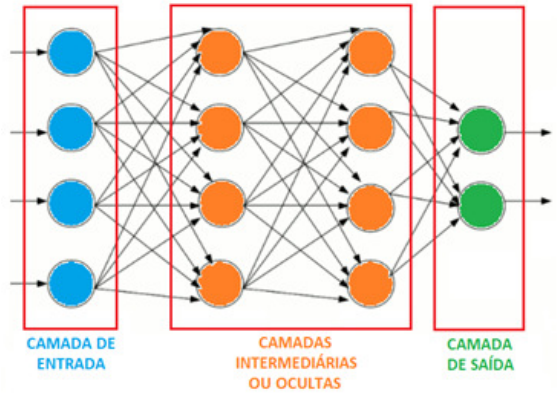

Figura 1. Estrutura de uma rede neural artificial. Fonte: [Viceri 2020]

O processo de aprendizado ocorre durante o treinamento da rede, realizado a partir de uma base de dados, que pode ser supervisionado ou não supervisionado. Quando supervisionado, são fornecidos à rede valores esperados para cada entrada, com o objetivo de orientar o treinamento. Quando não supervisionado, a rede aprende de forma auto-organizada, sem a necessidade de fornecer quais são os valores esperados [de Medeiros 2018].

3.3. Transferência de Aprendizado

O processo de transferência de aprendizado consiste em reutilizar um modelo que já foi previamente treinado em uma base de dados similar e mais genérica [Yang et al. 2020].

Esta técnica permite que um novo modelo, que será utilizado para uma tarefa diferente, possa ser treinado mais rapidamente se comparado com o tempo de treinamento ao montar uma nova rede neural [de Medeiros 2018], pois no momento que o novo modelo está sendo treinado diversas características mais genéricas já foram aprendidas [Yang et al. 2020].

Além do treinamento levar menos tempo, de forma geral o desempenho do modelo também é melhorado em bases de dados pequenas [Yang et al. 2020].

3.4. Redes Neurais Convolucionais

Redes neurais convolucionais são frequentemente utilizadas para realizar processamento de estruturas em formato tabular, como imagens, que podem ser interpretadas como matrizes de duas dimensões de *pixels* [Goodfellow et al. 2016].

Ao utilizar este tipo de rede neural [de Medeiros 2018], uma operação chamada de convolução é realizada nos dados. A convolução permite que a rede aprenda melhor

quais são as principais características de uma imagem, além de preservar alto nível de detalhes [Goodfellow et al. 2016].

Assim, a utilização de redes neurais convolucionais pode proporcionar maior assimilação das características mais importantes das imagens, por isso são muito utilizadas no campo da visão computacional [Szeliski 2010]. Foi utilizada uma rede neural convolucional no desenvolvimento deste trabalho.

4. Arquitetura Proposta

A Figura 2 mostra um diagrama que foi criado para representar a arquitetura proposta para o desenvolvimento do trabalho. Foram identificados três passos que ocorrem durante a execução, que são executados na ordem apresentada.

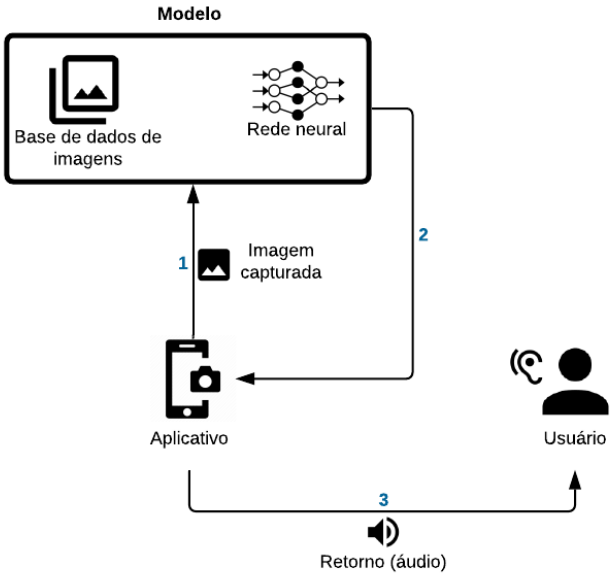

Figura 2. Fluxo de funcionamento do aplicativo

A execução do aplicativo se resume nos seguintes passos:

1. O usuário aponta a câmera do celular Android [Google 2008] em direção ao objeto, e em seguida o aplicativo alimenta a imagem capturada ao modelo de inteligência artificial, que foi treinado com uma base de dados que contém imagens de cada objeto que o aplicativo pode reconhecer;
2. O modelo retorna ao aplicativo o resultado do reconhecimento. Se trata de uma lista de probabilidades indicando qual é a probabilidade de cada classe existente na base de dados ser o objeto capturado;
3. O aplicativo processa a saída retornada pelo modelo e com base na classe com maior probabilidade, reproduz em forma de áudio o nome e direção em que o objeto se encontra, em relação ao usuário.

A Figura 3 apresenta as etapas que foram realizadas para a construção do modelo.

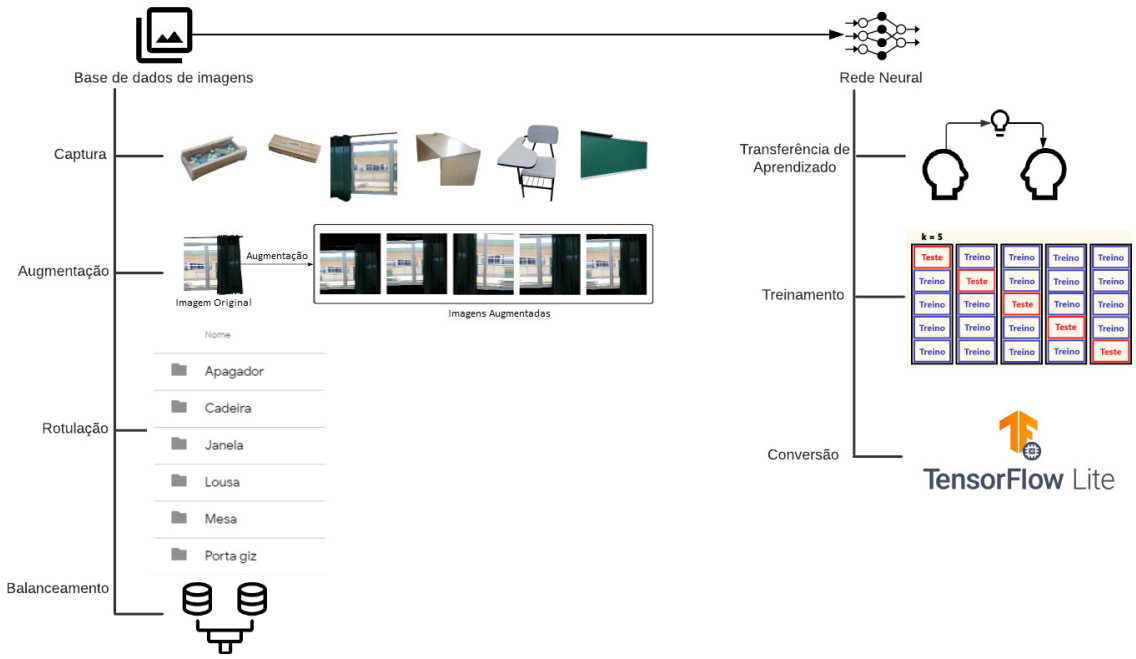

Figura 3. Etapas para construção do modelo

4.1. Base de Dados

Para que o treinamento da rede neural fosse viável e feito de forma eficiente, foi necessário delimitar um espaço onde fosse possível limitar a quantidade de objetos. Foram coletadas imagens de alguns objetos existentes em uma sala de aula padrão do IFSP Câmpus Hortolândia. Assim, foi possível obter uma amostra de uma porção de objetos existentes em uma universidade, ao mesmo tempo que foi reduzida a complexidade da aplicação, pois trabalhar com os inúmeros objetos que podem existir em uma universidade tornaria o escopo do projeto de alta complexidade.

Para montar a base de dados, primeiramente foram gravados vídeos, um para cada objeto que foi escolhido para constituir a base de dados, mostrando-o por diversos ângulos, distâncias e alturas diferentes. A partir destes vídeos, foram divididos os *frames* (quadros) de cada um, em diversas imagens de formato jpg. Nesta etapa, algumas imagens capturadas que não mostram com clareza o objeto foram eliminadas da base de dados, para evitar um impacto negativo no desempenho do modelo.

Para cada imagem gerada foi removido o fundo, pois isso permite um foco maior nas características dos objetos durante o treinamento. Foi utilizada a API Remove BG [Kaleido] para remover automaticamente o fundo de imagens. Em alguns casos, a API não é capaz de separar o fundo da imagem, então foi necessário remover manualmente o fundo destas imagens.

4.1.1. Augmentação de Imagens

Augmentação é um processo que são aplicadas transformações nas imagens com o objetivo de ampliar artificialmente a base de dados, de forma que sejam geradas novas imagens

com base nas originais, com variações como mudanças de orientação, distância e iluminação. Foi utilizada a biblioteca Keras [Chollet et al. 2015] para realizar este processo.

Esta etapa contribui positivamente para o desempenho do modelo em situações reais, pois as imagens capturadas durante um teste real são diferentes das imagens utilizadas durante o treinamento. Com maior variedade das imagens, é possível se aproximar um pouco mais das situações reais que ocorrem durante o uso do aplicativo, diminuindo a probabilidade de ocorrer *overfitting*, quando o modelo fica muito dependente dos dados de treinamento e não se adapta bem a dados reais [Ying 2019].

A Figura 4 mostra algumas imagens em que foi aplicado o processo de augmentação.

Figura 4. Exemplos de augmentação. Nestas imagens foram aplicadas variações de distância e orientação horizontal

4.1.2. Rotulação de Imagens

O processo de rotulação de imagens consiste em indicar nos dados de treinamento, para cada imagem existente na base de dados, qual é a classe correspondente.

Primeiramente, foram separadas todas as imagens da base de dados em pastas de mesmo nome do objeto correspondente, organizadas por ordem alfabética. Dentro de cada pasta foram armazenadas todas as imagens correspondentes ao objeto indicado em seu nome. A Figura 5 mostra como as pastas foram organizadas.

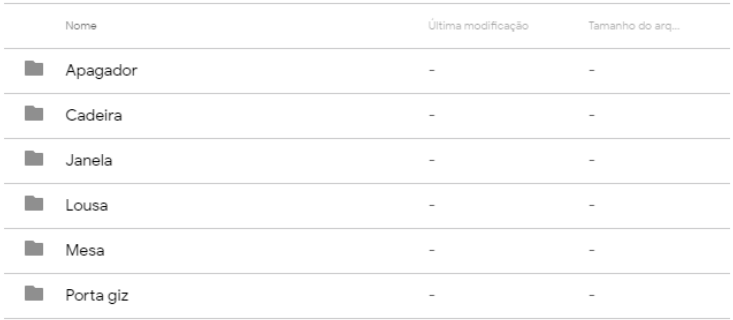

Figura 5. Organização de pastas

A partir destas imagens, foi gerado um arquivo csv com a rotulação de todas as imagens existentes na base de dados. A representação de cada classe no arquivo foi

realizada atribuindo um índice para cada uma, começando em 0 e incrementando em 1 sucessivamente até a última classe. Em uma coluna foi indicado o nome do arquivo da imagem e em outra coluna foi indicado o índice. A Tabela 1 mostra os índices utilizados para cada classe, e a Figura 6 mostra alguns exemplos de linhas do arquivo csv gerado.

Tabela 1. Índices correspondentes à cada classe

Classe	Índice
Apagador	0
Cadeira	1
Janela	2
Lousa	3
Mesa	4
Porta giz	5

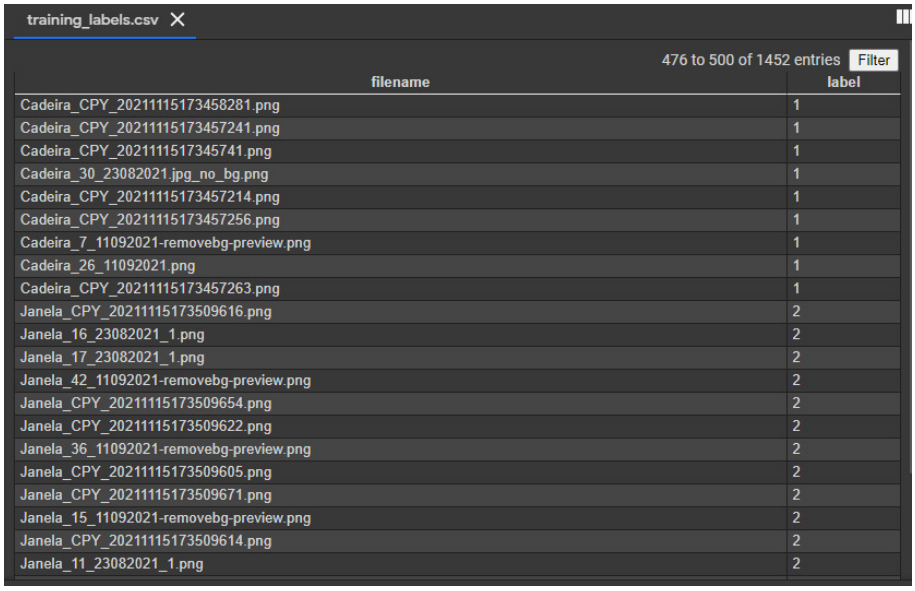

Figura 6. Exemplos de linhas do arquivo csv

4.1.3. Balanceamento de Imagens

Após organizadas as imagens na respectiva pasta da classe correspondente, foi verificado que houve diferença entre o número de imagens das classes. Para diminuir o impacto negativo que esta diferença poderia causar no desempenho do modelo, foi elaborado um *script*, executado antes do treinamento, que executa os seguintes passos:

1. Replica aleatoriamente as imagens de todas as classes;
2. Verifica qual é a classe que possui maior número de imagens;
3. Replica aleatoriamente as imagens das outras classes, que possuem menos imagens, até que o número de imagens de cada uma se iguale a classe com mais imagens.

A Tabela 2 mostra a quantidade de imagens de cada classe antes do balanceamento. Após o balanceamento, cada classe ficou com 242 imagens.

Tabela 2. Quantidade de imagens sem balanceamento

Classe	Nº de Imagens
Apagador	41
Cadeira	107
Janela	70
Lousa	51
Mesa	45
Porta giz	33

4.2. Rede Neural

Esta seção apresenta as etapas que foram realizadas para a construção da rede neural [de Medeiros 2018], envolvendo sua criação, treinamento, conversão e parâmetros utilizados.

4.2.1. Transferência de Aprendizado

Para construção da rede neural [de Medeiros 2018], foi utilizado o conceito de transferência de aprendizado [Yang et al. 2020].

A técnica de transferência de aprendizado utilizada neste trabalho foi realizada utilizando o modelo pré-treinado Inception-v3 [Szegedy et al. 2015]. O Inception-v3 [Szegedy et al. 2015] foi treinado por uma rede neural convolucional [Goodfellow et al. 2016] em uma base de dados com um total de 1000 classes diferentes de objetos.

Na transferência de aprendizado, para que seja possível reconhecer as classes que correspondem aos objetos específicos ao escopo do trabalho, além de utilizar as camadas do modelo pré-treinado também é necessário adicionar novas camadas à rede. Portanto, a rede neural completa foi constituída pela junção das primeiras camadas do Inception-v3 [Szegedy et al. 2015] com novas camadas criadas para que o modelo seja adequado às classes existentes na base de dados utilizada.

4.2.2. Treinamento

O treinamento da rede neural foi realizado por meio da biblioteca TensorFlow [Abadi et al. 2015].

Durante o treinamento foi utilizada a técnica de validação *cross-validation* [Hastie et al. 2017]. Esta técnica consiste em dividir o número de registros em uma base de dados por um certo número inteiro, e treinar o modelo pela quantidade de vezes igual ao número escolhido. A cada iteração em que o modelo é treinado, uma seção diferente da base de dados é utilizada para fazer a validação, ou seja, avaliar o desempenho do modelo. No final, a taxa de acertos do modelo é a média aritmética entre as taxas de acertos resultantes da validação de cada iteração do treinamento. A principal vantagem em se utilizar esta técnica é que o modelo tende a se adaptar melhor a situações de uso reais, diminuindo

a ocorrência de *overfitting*, que ocorre quando o modelo não generaliza bem ao processar os dados de validação, que não foram observados durante o treinamento [Ying 2019]. A Figura 7 representa o funcionamento da técnica de *cross-validation* [Hastie et al. 2017].

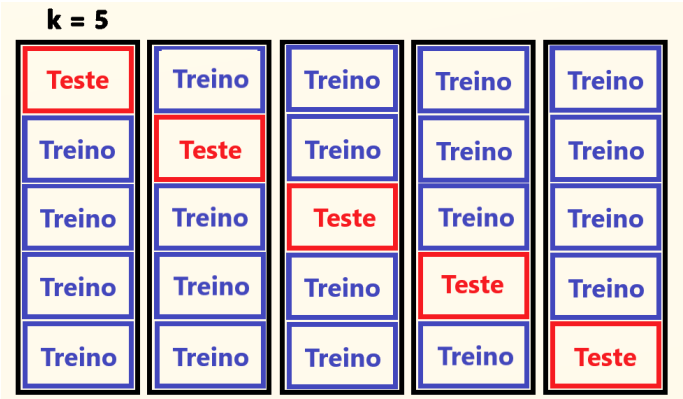

Figura 7. Processo de *cross-validation*. Fonte: [Tech 2021]

4.2.3. Conversão

Após treinamento da rede neural, foi obtido um modelo que pode ser utilizado para reconhecer objetos. Porém, para que funcione em um dispositivo Android [Google 2008] de forma eficiente, antes de utilizá-lo, foi convertido para um arquivo de formato tflite, utilizado pelo TensorFlow Lite [Google 2019]. Esta conversão foi realizada utilizando a biblioteca TensorFlow [Abadi et al. 2015].

O TensorFlow Lite [Google 2019] é um *framework* que permite a utilização de forma otimizada de modelos de inteligência artificial em dispositivos móveis, como *smartphones* Android [Google 2008] ou iOS [Apple 2007] e dispositivos IoT, como Raspberry Pi [Pi 2012].

4.2.4. Parâmetros

Ao treinar uma rede neural [de Medeiros 2018], existem diversos parâmetros que devem ser especificados, pois são usados durante o treinamento. Alterações nesses parâmetros afetam o desempenho do modelo, positivamente ou negativamente, dependendo de cada situação. Para determinar quais são os melhores valores para os parâmetros em uma determinada situação, é necessário realizar testes, que são usados para levantar o desempenho do modelo ao utilizar certos valores de parâmetros.

Neste trabalho foi necessário ajustar os seguintes parâmetros:

- Tamanho da imagem: para utilizar uma rede neural [de Medeiros 2018] que receba imagens na camada de entrada, é necessário redimensionar todas as imagens para um tamanho fixo. No caso deste trabalho, todas as imagens foram redimensionadas para o tamanho 300x300, que após realização de testes trouxe bom resultados;

- Número de épocas: em cada época, toda a base de dados é utilizada no treinamento da rede neural [Keras 2015]. Nas primeiras épocas, o desempenho do modelo aumenta, até chegar em um ponto que se torna estável ou começa a diminuir. Por meio de testes é necessário encontrar um número de épocas ideal para determinada situação. Ao utilizar um número muito baixo, é possível que ocorra *underfitting*, quando o modelo não é treinado o suficiente, porém se o número for muito alto, é possível que ocorra *overfitting*, quando o modelo se adapta muito aos dados de treinamento e não generaliza bem com os dados de validação [Google 2017b]. No caso deste trabalho, o modelo foi treinado por 7 épocas.
  A Figura 8 mostra a evolução da taxa de desempenho do modelo ao decorrer das épocas. Pode-se observar que no início a taxa de desempenho de treinamento e de validação possuíam um valor que sofreu aumento, com maior variação durante a primeira época. Ao decorrer das próximas épocas houve uma variação menor, até que houve uma estabilização ao se aproximar da sétima época.

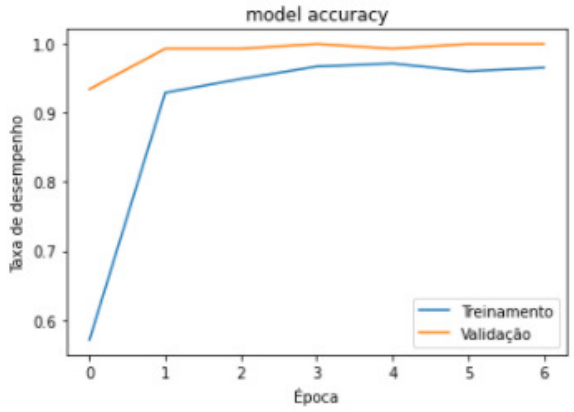

Figura 8. Taxa de desempenho ao decorrer das épocas

- Parâmetros de augmentação: existem alguns parâmetros que são responsáveis por controlar quais transformações serão aplicadas às imagens durante o processo de augmentação. Foram especificados os seguintes parâmetros [Keras]:
  - Faixa de deslocamento na largura: parâmetro responsável por determinar o intervalo que deve ser aplicado o deslocamento horizontal da posição da imagem. No trabalho foi utilizado o valor 0,1, que significa que a imagem será deslocada para a esquerda e para a direita em até 10% da largura total da imagem. A Figura 9 mostra um exemplo da aplicação desta transformação em uma imagem;

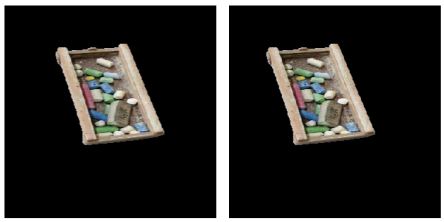

Figura 9. A imagem à esquerda não possui deslocamento e a imagem à direita mostra o resultado após a aplicação do deslocamento para a esquerda

– Faixa de deslocamento na altura: parâmetro responsável por determinar o intervalo que deve ser aplicado o deslocamento vertical da posição da imagem. No trabalho foi utilizado o valor 0,1, que significa que a imagem será deslocada para cima ou para baixo em até 10% da altura total da imagem. A Figura 10 mostra um exemplo da aplicação desta transformação em uma imagem;

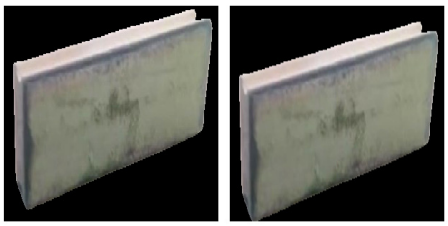

Figura 10. A imagem à esquerda não possui deslocamento e a imagem à direita mostra o resultado após a aplicação de deslocamento para baixo

– Faixa de *zoom*: durante a augmentação, caso este parâmetro seja especificado, são aplicadas aleatoriamente diferentes intensidades de *zoom* em cada imagem. Este parâmetro determina em qual intervalo devem ser aplicadas as intensidades de *zoom*. No trabalho foi utilizado o valor 0,3, que significa que pode ser aplicado zoom de até 30% nas imagens. A Figura 11 mostra um exemplo da aplicação desta transformação em uma imagem;

Figura 11. A imagem à esquerda não possui zoom e a imagem à direita mostra o resultado após a aplicação de *zoom*, que aproximou a imagem

– Modo de preenchimento: parâmetro utilizado para definir qual deve ser o comportamento caso faltem *pixels* em alguns pontos após a aplicação de transformações na imagem. No trabalho este parâmetro foi definido de forma que, caso haja regiões com *pixels* não existentes, essas regiões são preenchidas com o mesmo valor do pixel existente mais próximo;
– Inversão horizontal: parâmetro que define se as imagens devem ser invertidas horizontalmente. A Figura 12 mostra um exemplo da aplicação desta transformação em uma imagem.

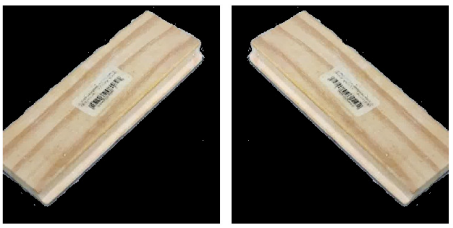

Figura 12. A imagem à esquerda não possui inversão horizontal e a imagem à direita mostra o resultado após a aplicação da inversão horizontal

- Normalização: define o número que cada *pixel* da imagem deve ser multiplicado, após a aplicação de todas as transformações da augmentação, para normalização dos *pixels*. No trabalho foi utilizado o valor 1/255. Após cada *pixel* ser multiplicado por este número, seu valor será alterado para um valor entre 0 e 1, pois cada *pixel* em uma imagem tem seu valor original entre 0 e 255;
- Otimizador: o otimizador é um algoritmo que ajusta os valores dos pesos da rede neural [de Medeiros 2018]. A escolha do otimizador determina o tempo de treinamento e o desempenho do modelo [Choi et al. 2020]. Neste trabalho foi utilizado o otimizador Adam [Kingma e Ba 2017], que é computacionalmente eficiente e tem poucos requisitos de memória, além de ter mostrado ser bem sucedido em tarefas de visão computacional [Szeliski 2010] utilizando redes neurais convolucionais [Goodfellow et al. 2016], segundo os desenvolvedores do algoritmo;
- Função de perda: a função de perda é uma forma de cálculo da perda, definida como a distância entre o retorno da rede neural [de Medeiros 2018] e o valor esperado, é utilizada como uma forma de avaliar a qualidade do retorno da rede [Chollet 2018]. Neste trabalho foi utilizada a função Entropia Cruzada Categórica Esparsa [Google], recomendada em problemas de classificação que possuem duas ou mais classes;
- Parâmetros de *cross-validation* [Pedregosa et al. 2011]: existem alguns parâmetros que são responsáveis por controlar como será realizado o processo de *cross-validation*. Foram especificados os seguintes parâmetros:
  - Número de divisões: indica o número que será utilizado para dividir a base de dados. Neste trabalho foi utilizado o valor 5, que significa que a base de dados foi dividida em 5 partes e a rede neural [de Medeiros 2018] foi treinada em 5 iterações, e em cada iteração foi utilizada uma parte diferente da base de dados para ser utilizada na validação;
  - Embaralhamento: define se os dados devem ser embaralhados antes de ser realizada a divisão. Neste trabalho foi configurado para ser realizado o embaralhamento.

Em redes neurais convolucionais [Goodfellow et al. 2016] é possível definir alguns parâmetros, que influenciam a estrutura da rede neural [de Medeiros 2018]. Os principais parâmetros que podem ser definidos são os seguintes:

- Tamanho dos filtros: define a dimensão da matriz do filtro em cada camada. Um filtro é uma matriz de valores inicialmente aleatórios, utilizado para realização da operação de convolução. Durante o treinamento, os valores dos filtros são atualizados com objetivo de melhorar o desempenho do modelo [Goodfellow et al. 2016];

- Comprimento de passada: define o tamanho do passo que deve ser realizado entre as operações de convolução em uma camada convolucional [Goodfellow et al. 2016].

Os valores utilizados para os parâmetros de convolução foram os valores padrões já definidos no modelo Inception-v3 [Szegedy et al. 2015]. São utilizados filtros de tamanho 3x3 e tamanhos de passada que variam entre 1 e 2 [Szegedy et al. 2015].

4.3. Tecnologias Utilizadas

A Tabela 3 apresenta as tecnologias que foram utilizadas no trabalho.

Tabela 3. Tecnologias utilizadas

Nome	Descrição
Python 3.7.12	Linguagem de programação de alto nível de propósito geral. Foi utilizada para implementar a construção do modelo de inteligência artificial, envolvendo por exemplo, manipulação das imagens que compõem a base de dados, construção da rede neural, treinamento e obtenção das métricas que indicam o desempenho do modelo
Google Colab	Serviço na nuvem que permite executar código Python. É possível utilizar gratuitamente o *hardware* dos servidores do Google, o que pode reduzir significativamente o tempo de treinamento de redes neurais
TensorFlow 2.7.0	Plataforma de código aberto que possui diversas ferramentas de inteligência artificial, como a criação e treinamento de redes neurais. Neste trabalho, os processos de criação da rede neural, treinamento e validação do modelo envolveram a utilização desta tecnologia.
Keras 2.7.0	API utilizada em conjunto com o TensorFlow para criação e treinamento de redes neurais. Fornece diversas ferramentas que foram utilizadas no desenvolvimento do trabalho, como a definição da estrutura, os tipos e configurações de cada camada que compõe a rede neural, definição de parâmetros utilizados durante o treinamento e augmentação da base de dados, e modelos pré-treinados que podem ser utilizados para aplicar transferência de aprendizado
TensorFlow Lite 2.7.0	Permite executar modelos de inteligência artificial de forma otimizada para dispositivos móveis, como Android, iOS, e dispositivos IoT
RemoveBG	Utilizada para remover de forma automatizada os fundos das imagens da base de dados
Scikit-learn 1.0.2	Biblioteca compatível com a linguagem de programação Python que fornece diversas ferramentas que foram utilizadas no trabalho, como cálculo de métricas que indicam o desempenho do modelo, e definições de *cross-validation*
Pandas 1.1.15	Biblioteca compatível com a linguagem de programação Python que possui ferramentas para análise e manipulação de dados. Foi utilizada para manipular arquivos csv, com o objetivo de utilizá-los para obter métricas a partir da biblioteca Scikit-learn
Android 11	Sistema operacional baseado em Linux desenvolvido pela Google, utilizado principalmente em dispositivos móveis. Foi utilizada como plataforma para realizar testes com o modelo desenvolvido
Android Studio 2020.3.1	Ferramenta desenvolvida pela Google para desenvolvimento de aplicativos destinados a dispositivos Android
Java 11.0.11	Linguagem de programação de alto nível de propósito geral. Neste trabalho, foi utilizada para implementar o aplicativo Android
ML Kit 16	Biblioteca que possui diversos recursos relacionados a inteligência artificial para serem utilizados em aplicativos Android ou iOS. Oferece uma interface de alto nível para utilização de modelos customizados de detecção de objetos
Text-to-speech	Tecnologia que converte texto em fala. Faz parte das tecnologias disponíveis nativamente no Android. Foi utilizada para o desenvolvimento da funcionalidade de falar o nome dos objetos identificados

5. Desenvolvimento do Aplicativo

Esta seção apresenta detalhes sobre o desenvolvimento do aplicativo Android [Google 2008], como seu fluxo e as implementações que foram realizadas.

5.1. Fluxo

Ao abrir o aplicativo, são procurados objetos na imagem capturada pela câmera do *smartphone*. No momento que algum objeto é detectado, a região que ele se encontra na imagem é alimentada ao modelo de detecção de objetos. Caso o percentual de detecção seja maior ou igual ao limiar configurado no aplicativo, o aplicativo irá falar o nome do objeto, caso contrário, será falado "Objeto Desconhecido". Em seguida, inicia-se o fluxo novamente. Este ciclo se repete até que o usuário feche o aplicativo.

O fluxograma da Figura 13 representa a execução do aplicativo.

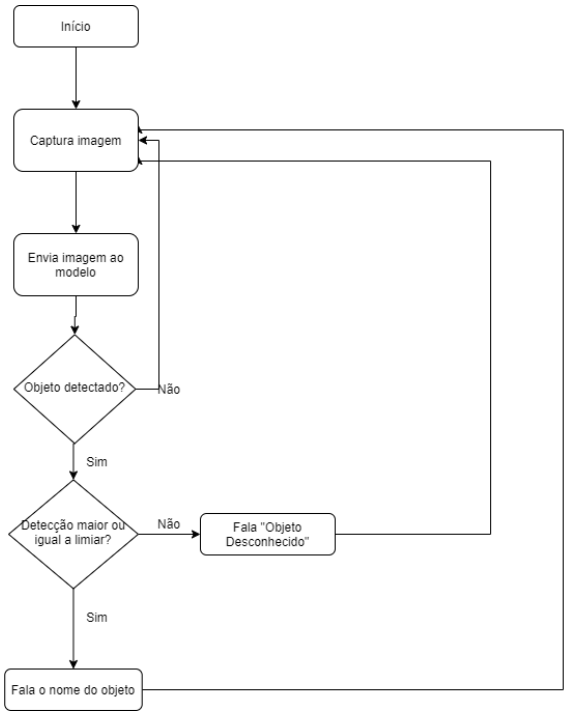

Figura 13. Fluxo do Aplicativo

5.2. Desenvolvimento

O aplicativo Android [Google 2008] foi desenvolvido por meio da IDE Android Studio [Google 2014], utilizando a linguagem de programação Java [Arnold et al. 2005]. Para integração com o modelo de inteligência artificial treinado no formato do TensorFlow Lite [Google 2019], foi utilizada a biblioteca ML Kit [Google 2020a].

Foi utilizado como base um repositório do GitHub [Google 2020b] com amostras dos recursos da biblioteca ML Kit [Google 2020a], para facilitar o desenvolvimento. O repositório possui um exemplo de visão computacional [Szeliski 2010], no qual foram feitas modificações conforme as necessidades do trabalho.

Principais modificações realizadas:

- Interface simples: a interface do aplicativo foi desenvolvida de forma que, no momento que o usuário abre o aplicativo, os objetos capturados pela câmera já são identificados e falados ao usuário. Não é necessário selecionar alguma opção ou fazer alguma configuração para utilizar o aplicativo, tornando sua utilização simples;

- Retorno dos nomes das classes: foi implementada a funcionalidade de retornar, na tela do aplicativo e em áudio, o nome do objeto que foi identificado. Para que isso funcione corretamente, foi necessário criar um arquivo txt que serviu como mapeamento do nome de cada classe com seu respectivo índice, onde cada linha corresponde a um índice. O aplicativo retorna ao usuário o nome da classe que está na linha correspondente ao índice retornado pelo modelo após a identificação de certo objeto;

- Objeto desconhecido: quando um objeto é reconhecido com uma porcentagem de detecção menor que um certo limiar, que possui um valor padrão de 80%, é informado ao usuário que existe um objeto desconhecido. Esta implementação foi feita, pois durante a utilização do aplicativo, podem existir situações que o modelo não identifica muito bem um objeto, que pode acontecer por má visualização de certo objeto. Nestas situações, existe possibilidade de ser identificado um objeto incorreto. O valor padrão do limiar foi escolhido após testes com o aplicativo, cujo resultado indicou que o valor foi suficiente para limitar a identificação dos objetos em situações em que certo objeto não é capturado corretamente pela câmera;

- Fala: foi implementada a funcionalidade de fala no aplicativo. O nome e a direção do objeto que está sendo detectado no momento são falados, para que um usuário de baixa visão saiba quais objetos existem em sua volta.

6. Resultados

Os testes foram realizados no IFSP - Câmpus Hortolândia, considerando os objetos que a rede neural foi treinada. O aplicativo foi testado apontando a câmera em frente ao corpo e locomovendo-se, com o objetivo de simular a utilização por um usuário real com baixa visão.

Para validação do desempenho do modelo durante o uso do aplicativo, foi realizada uma implementação de salvamento de *logs* em arquivos de formato csv. O aplicativo salva informações a cada detecção realizada, e cada linha contém o horário e qual objeto foi identificado no momento. O aplicativo salva diversos *logs* por segundo, então as informações do arquivo foram agrupadas para facilitar a análise, de forma que fossem removidos os registros duplicados, além de limpar o arquivo deixando apenas informações necessárias. A Figura 14 mostra exemplos de linhas do arquivo sem agrupamento, e a Figura 15 mostra exemplos de linhas do arquivo com agrupamento.

1 to 10 of 4764 entries Filter

1633779348949	2021.10.09 08:35:48.949	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:48 - Cadeira a frente - 99.99974%
1633779349066	2021.10.09 08:35:49.066	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a direita - 99.55357%
1633779349067	2021.10.09 08:35:49.067	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a direita - 99.55357%
1633779349090	2021.10.09 08:35:49.090	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a direita - 99.62557%
1633779349092	2021.10.09 08:35:49.092	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a direita - 99.62557%
1633779349112	2021.10.09 08:35:49.112	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a frente - 99.99854%
1633779349091	2021.10.09 08:35:49.091	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a direita - 99.62557%
1633779349110	2021.10.09 08:35:49.110	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a frente - 99.99854%
1633779349113	2021.10.09 08:35:49.113	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a frente - 99.99854%
1633779349111	2021.10.09 08:35:49.111	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a frente - 99.99854%
1633779349191	2021.10.09 08:35:49.191	DEBUG	PRETTY_LOGGER	09/10/2021 08:35:49 - Cadeira a frente - 99.99974%

Show 10 per page 1 2 10 100 400 470 477

Figura 14. Exemplos de linhas do arquivo csv sem agrupamento

1 to 10 of 436 entries Filter

data_hora	nome	porcentagem	counts
09/10/2021 08:35:48	Cadeira a frente	99.99974	1
09/10/2021 08:35:49	Cadeira a direita	99.55357	8
09/10/2021 08:35:49	Cadeira a direita	99.62557	10
09/10/2021 08:35:49	Cadeira a frente	99.99854	12
09/10/2021 08:35:49	Cadeira a frente	99.99974	5
09/10/2021 08:35:51	Cadeira a frente	97.96404	9
09/10/2021 08:35:52	Cadeira a direita	99.841705	45
09/10/2021 08:35:52	Cadeira a direita	99.99883	29
09/10/2021 08:35:52	Cadeira a frente	97.96404	33
09/10/2021 08:35:52	Cadeira a frente	99.98629	31

Show 10 per page 1 2 10 40 44

Figura 15. Exemplos de linhas do arquivo csv com agrupamento

Para ser possível a análise dos registros do arquivo, também foi necessário gravar a tela do dispositivo Android [Google 2008] durante o uso do aplicativo. Assim foi possível identificar, acompanhando o vídeo, em quais momentos foi detectado um objeto corretamente. Para cada detecção, foi anotado qual era o objeto real apontado pela câmera.

Com as informações de qual objeto foi identificado pelo aplicativo e qual foi o objeto real em cada detecção, foi possível identificar algumas métricas, que foram calculadas por meio da biblioteca scikit-learn [Pedregosa et al. 2011], disponível para a linguagem de programação Python [Van Rossum e Drake 2009]. Foram identificadas as seguintes métricas: acurácia, precisão, *recall* e F1-score. A acurácia é referente a taxa de acertos geral, pode ser calculada pela seguinte fórmula [Géron 2019]:

$$\text{Acurácia} = \frac{\text{Total de Acertos}}{\text{Total de Acertos} + \text{Total de Erros}}$$

As outras métricas envolvidas são calculadas a partir dos valores de verdadeiros positivos (VP), falsos positivos (FP) e falsos negativos (FN) referentes aos resultados de cada classe, e podem ser calculadas por meio das seguintes fórmulas [Géron 2019]:

$$\text{Precisão} = \frac{\text{Verdadeiros Positivos (VP)}}{\text{Verdadeiros Positivos (VP)} + \text{Falsos Positivos (FP)}}$$

$$Recall = \frac{\text{Verdadeiros Positivos (VP)}}{\text{Verdadeiros Positivos (VP)} + \text{Falsos Negativos (FN)}}$$

$$\text{F1-score} = \frac{2 * \text{Precisão} * Recall}{\text{Precisão} + Recall}$$

Após realizados os cálculos, foi observada uma acurácia [Géron 2019] de 67,66%, número que ainda pode ser melhorado para que seja possível a utilização do aplicativo por um usuário real, pois erros de detecção podem apresentar riscos ao usuário. Além disso, ainda também é necessário realizar testes supervisionados com usuários reais, para garantir a segurança durante o uso do aplicativo.

A Tabela 4 mostra as métricas calculadas para cada classe a partir dos testes.

Tabela 4. Métricas de cada classe

Classe	Precisão	*Recall*	F1-score
Apagador	50%	26%	35%
Cadeira	90%	87%	89%
Janela	82%	84%	83%
Lousa	100%	37%	54%
Mesa	48%	21%	29%
Porta giz	70%	70%	70%

6.1. Análise das Métricas

Apesar de não ter ocorrido com todas as classes, foi observada uma relação entre a quantidade de imagens antes de ser aplicado o balanceamento e as métricas. As duas classes com os melhores valores de métricas, cadeira e janela, possuem o maior número de imagens. As classes lousa, apagador e mesa tiveram baixo desempenho e possuem um número baixo de imagens. A única classe que não seguiu esta relação foi porta-giz, que tem o menor número de imagens entre as classes, mas foi observado um bom valor em suas métricas. Assim, foi observado que replicar as imagens não foi uma boa solução para resolver o problema da quantidade diferente de imagens, seria mais eficiente capturar um número igual de imagens para cada classe no momento da coleta.

Para as classes em que foi observado um valor baixo da precisão [Géron 2019], como por exemplo o apagador, a solução seria verificar quais objetos foram confundidos com apagador, e coletar mais imagens desses objetos, diversificando ainda mais os ângulos e distâncias capturados, para que o modelo possa identificar mais características destes objetos.

Quando observado um valor baixo de *recall* [Géron 2019], como por exemplo a lousa, a solução seria coletar mais imagens de lousa, para que o modelo possa aprender mais características de lousa e diminua o número de vezes que identifica um objeto diferente.

6.1.1. Apagador

Teve um valor de precisão [Géron 2019] e *recall* [Géron 2019] baixos e com alta diferença entre eles, indicando que o modelo, além de classificar incorretamente imagens de

apagador, também classificou como apagador objetos que não eram realmente um apagador.

6.1.2. Cadeira

Todas as suas métricas possuem um valor elevado e equilibrado, indicando que o modelo desempenhou bem com esta classe.

6.1.3. Janela

Todas as suas métricas possuem um valor elevado e equilibrado, indicando que o modelo desempenhou bem com esta classe.

6.1.4. Lousa

Teve valores de precisão [Géron 2019] e *recall* [Géron 2019] com a maior diferença entre as classes, com 100% de precisão [Géron 2019] e um valor baixo de *recall* [Géron 2019]. Isso indica que o modelo classificou incorretamente imagens de lousa em diversos momentos, porém em nenhum momento classificou como lousa um objeto que não era realmente uma lousa.

6.1.5. Mesa

Teve um valor de precisão [Géron 2019] e *recall* [Géron 2019] baixos e com alta diferença entre eles, indicando que o modelo, além de classificar incorretamente imagens de mesa, também classificou como mesa objetos que não eram realmente uma mesa.

6.1.6. Porta-giz

Todas as suas métricas possuem um valor elevado e equilibrado, indicando que o modelo desempenhou bem com esta classe.

7. Conclusão

A visão computacional [Szeliski 2010] possui grande capacidade de auxiliar pessoas com baixa visão. Esta área proporciona diversas possibilidades de soluções que podem ser aplicadas para esta utilidade. O aplicativo desenvolvido neste projeto mostrou que é possível auxiliar pessoas de baixa visão com um baixo custo.

Para que o trabalho possa ser utilizado por usuários reais, ainda é necessário realizar melhorias na base de dados, com o objetivo de melhorar os resultados para que seja possível utilizá-lo com mais segurança.

Durante o desenvolvimento deste projeto, foi possível exercitar diversos conhecimentos:

- Inteligência artificial
  - Python [Van Rossum e Drake 2009]: manipulação de arquivos, como acesso, cópia, deleção e conceitos básicos da linguagem
  - TensorFlow [Abadi et al. 2015]: criação e treinamento de rede neural [de Medeiros 2018]
  - Keras [Chollet et al. 2015]: parâmetros de pré-processamento da rede, camadas da rede neural [de Medeiros 2018]
  - Google Colab [Google 2017a]: utilização básica da ferramenta, utilização de comandos Linux para manipulação de arquivos, como copiar, mover, remover e compactar/descompactar arquivos
  - Redes neurais [de Medeiros 2018]: redes neurais convolucionais [Goodfellow et al. 2016], influência dos ajustes dos parâmetros utilizados pela rede [de Medeiros 2018]
  - Base de dados: montagem de base de dados para treinamento
- Desenvolvimento Android [Google 2008]
  - Java [Arnold et al. 2005]: conceitos básicos da linguagem
  - ML Kit [Google 2020a]: utilização do ML Kit [Google 2020a] com modelos de reconhecimento de objetos
  - Programação orientada a objetos [Sintes 2002]

8. Trabalhos Futuros

Neste trabalho foi possível realizar testes preliminares de uma aplicação que envolve visão computacional [Szeliski 2010], executada em dispositivos Android [Google 2008]. Como não foram realizados testes com deficientes visuais, não é possível garantir que o aplicativo esteja pronto para ser utilizado por usuários reais. A metodologia utilizada neste trabalho pode ser reaproveitada no desenvolvimento de trabalhos futuros.

Um possível trabalho futuro é o desenvolvimento de uma aplicação similar que execute em algum dispositivo IoT, pois permitiria maior mobilidade ao ser utilizado em conjunto com objetos comuns, como óculos ou alguma peça de roupa, o que facilitaria o uso por deficientes visuais.

Após realização de testes com o aplicativo Android [Google 2008] desenvolvido neste trabalho, foi concluído que é viável o desenvolvimento para dispositivos IoT, utilizando tecnologias similares.

No desenvolvimento de um trabalho futuro, seria necessário realizar melhorias na base de dados e testes com deficientes visuais, para melhorar os resultados e verificar a viabilidade do trabalho poder ser utilizado por usuários reais. Também podem ser realizadas alterações nos valores dos parâmetros utilizados, com o objetivo de otimizá-los, o que poderia melhorar os resultados.

Referências

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I.,

Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/. Acesso em: 25/12/2021.

Angoletta, M. (2008). Digital signal processor fundamentals and system design.

Apple (2007). ios. https://www.apple.com/br/ios/ios-15. Acesso em: 25/12/2021.

Arnold, K., Gosling, J., Holmes, D. (2005). *The Java programming language*. Addison Wesley Professional, 4ª ed.

Babic, M. Junior, R. S. (2020). *Oftalmologia*. Atheneu, 1ª ed.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., Dahl, G. E. (2020). On empirical comparisons of optimizers for deep learning.

Chollet, F. (2018). *Deep Learning with Python*. Manning, 1ª ed.

Chollet, F. et al. (2015). Keras. https://keras.io. Acesso em: 31/12/2021.

de Brittos Valdati, A. (2020). *Inteligência artificial - IA*. Contentus, 1ª ed.

de Medeiros, L. F. (2018). *Inteligência artificial aplicada: uma abordagem introdutória*. Editora Intersaberes, 1ª ed.

Eikvil, L. (1993). Ocr - optical character recognition.

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Google. Sparse categorica crossentropy. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy. Acesso em: 30/01/2022.

Google (2008). Android. https://www.android.com/. Acesso em: 20/12/2021.

Google (2014). Android studio. https://developer.android.com/studio. Acesso em: 31/12/2021.

Google (2017a). Google colab. https://colab.research.google.com/. Acesso em: 31/12/2021.

Google (2017b). Overfit and underfit. https://www.tensorflow.org/tutorials/keras/overfit_and_underfit. Acesso em: 29/01/2022.

Google (2019). Tensorflow lite. https://www.tensorflow.org/lite. Acesso em: 25/12/2021.

Google (2020a). Ml kit. https://developers.google.com/ml-kit/. Acesso em: 31/12/2021.

Google (2020b). Mlkit samples. https://github.com/googlesamples/mlkit. Acesso em: 28/02/2021.

Géron, A. (2019). *Mãos à obra: aprendizado de máquina com Scikit-Learn & TensorFlow*. Alta Books, 1ª ed.

Hastie, T., Tibshirani, R., Friedman, J. (2017). *The Elements of Statistical Learning*. Springer, 2ª ed.

IBGE (2010). Censo demográfico 2010: características gerais da população, religião e pessoas com deficiência. https://biblioteca.ibge.gov.br/visualizacao/periodicos/94/cd_2010_religiao_deficiencia.pdf. Acesso em: 20/11/2021.

IBGE (2020). Pessoas com deficiência: adaptando espaços e atitudes. https://biblioteca.ibge.gov.br/visualizacao/livros/liv101770.pdf. Acesso em: 20/11/2021.

Joshi, R., Yadav, S., Dutta, M., Travieso, C. (2020). Efficient multi-object detection and smart navigation using artificial intelligence for visually impaired people. *Entropy*, 22:941.

Kaleido. Remove bg. https://www.remove.bg/api. Acesso em: 21/12/2021.

Keras. Imagedatagenerator. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator. Acesso em: 29/01/2022.

Keras (2015). Model training apis. https://keras.io/api/models/model_training_apis/. Acesso em: 29/01/2022.

Kingma, D. P. Ba, J. (2017). Adam: A method for stochastic optimization.

Krzanowski, W. J. Hand, D. J. (2009). *ROC Curves for Continuous Data*. CRC Press, 1ª ed.

Mallat, S. G. (1987). A theory for multiresolution signal decomposition: The wavelet representation.

OMS (2021a). Icd-11 for mortality and morbidity statistics. https://icd.who.int/browse11/l-m/en#/http%3a%2f%2fid.who.int%2ficd%2fentity%2f30317704. Acesso em: 18/11/2021.

OMS (2021b). Icd-11 for mortality and morbidity statistics. https://icd.who.int/browse11/l-m/en#/http%3a%2f%2fid.who.int%2ficd%2fentity%2f1103667651. Acesso em: 18/11/2021.

Papageorgiou, C., Evgeniou, T., Poggio, T. (2000). A trainable pedestrian detection system. *Proceedings of Intelligent Vehicles*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pi, F. R. (2012). Raspberry pi. https://www.raspberrypi.com/. Acesso em: 31/12/2021.

Redmon, J. Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv*.

Sintes, T. (2002). *Aprenda Programação Orientada a Objetos em 21 dias*. Pearson, 1ª ed.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567.

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

Tech, D. (2021). cross-validation. `https://didatica.tech/o-pacote-caret-linguagem-r/`. Acesso em: 31/12/2021.

Van Rossum, G. Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer, 2ª ed.

Viceri (2020). Rede neural. `https://viceri.com.br/insights/arquiteturas-de-redes-neurais-convolucionais-para-reconhecimento-de-imagens/`. Acesso em: 31/12/2021.

V.K, M. S., Dileep, M., Rameez, K. M., Soni, A., K, S. K. (2019). Third eye - an app for blind assistance u sing machine learning. *International Journal of Computer Science Trends and Technology*.

Yang, Q., Zhang, Y., Dai, W., Pan, S. J. (2020). *Transfer Learning*. Cambridge University Press, 1ª ed.

Ying, X. (2019). An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168:022022.

Documento Digitalizado Público

Anexo I (Artigo) - Leonardo Felipe Mendes Lopes - HT1720007

Assunto: Anexo I (Artigo) - Leonardo Felipe Mendes Lopes - HT1720007
Assinado por: Carlos Junior
Tipo do Documento: Outro
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Carlos Roberto dos Santos Junior**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 04/04/2022 10:30:22.

Este documento foi armazenado no SUAP em 04/04/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 934826
Código de Autenticação: db0824d631