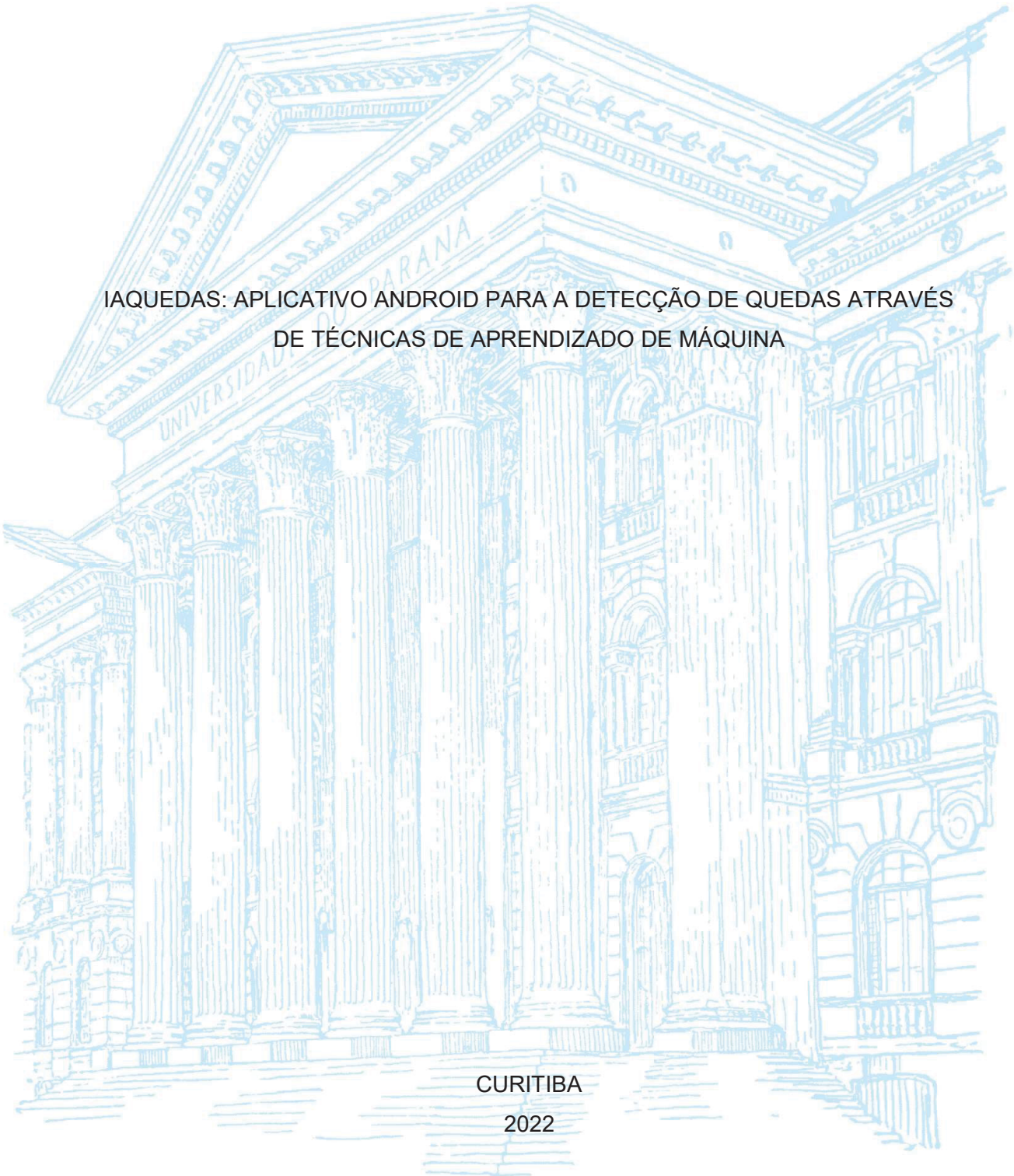
UNIVERSIDADE FEDERAL DO PARANÁ

GUSTAVO HENRIQUE SILVA



IAQUEDAS: APLICATIVO ANDROID PARA A DETECÇÃO DE QUEDAS ATRAVÉS DE TÉCNICAS DE APRENDIZADO DE MÁQUINA

CURITIBA

2022

GUSTAVO HENRIQUE SILVA

IAQUEDAS: APLICATIVO ANDROID PARA A DETECÇÃO DE QUEDAS ATRAVÉS DE TÉCNICAS DE APRENDIZADO DE MÁQUINA

TCC apresentado ao curso de Pós-Graduação em Inteligência Artificial Aplicada, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Especialista em Inteligência Artificial Aplicada.

Orientador: Prof. Dr. Jaime Wojciechowski.

CURITIBA

2022

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INTELIGÊNCIA ARTIFICIAL APLICADA da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **GUSTAVO HENRIQUE SILVA** intitulada: **IAQUEDAS: APLICATIVO ANDROID PARA A DETECCAO DE QUEDAS ATRAVES DE TECNICAS DE APRENDIZADO DE MAQUINA**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua __APROVAÇÃO__ no rito de defesa.

A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 15 de Março de 2022.

JAIME WOJCIECHOWSKI
Presidente da Banca Examinadora

RAZER ANTHOM NIZER ROJAS MONTAÑO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

AGRADECIMENTOS

Aos amigos e familiares, que durante os últimos anos compreenderam pacientemente minha ausência. À minha companheira Ana, que presenciou cada passo no processo de desenvolvimento deste trabalho e me ajudou a manter uma rotina de estudos constante.

“Na busca da inteligência artificial, não devemos sacrificar a humanidade, a criatividade e a ingenuidade que definem nossa inteligência humana”

(Tim Cook, 2018)

RESUMO

As quedas figuram entre os acidentes mais comuns em ambientes domésticos e de trabalho. Em alguns casos, elas são responsáveis por consequências que impactam diretamente a qualidade de vida dos indivíduos, principalmente a dos mais velhos. Eventualmente, as quedas podem ocorrer quando o indivíduo está sozinho, em local isolado, ou impossibilitado de pedir ajuda. Evidentemente, quanto mais rápido o atendimento em casos graves, melhores as chances de recuperação e menor o impacto na qualidade de vida do paciente. Com base nessas constatações, nota-se também a necessidade de soluções que aumentam as chances de atendimento em casos de queda. Neste contexto, este trabalho apresenta o aplicativo iaQuedas e documenta seu processo de desenvolvimento. Este aplicativo possui as funcionalidades da detecção de possíveis eventos de quedas e notificação de um contato de confiança via SMS. A detecção é realizada através da apresentação dos dados do sensor acelerômetro para uma rede neural artificial do tipo LSTM, especializada na classificação de movimentos. O treinamento do modelo de inteligência artificial é realizado com dados da base MobiAct 2.0, a qual possui amostras de atividades diárias e de quedas. As predições realizadas para os dados de validação ao fim do treinamento apresentam alta acurácia. Durante testes do aplicativo no dia-a-dia, quedas também são devidamente detectadas. Contudo, casos de falso-positivo na detecção de quedas são mais frequentes do que o esperado. O resultado final é um aplicativo que cumpre com seus objetivos, com espaço para melhorias em trabalhos futuros.

Palavras-chave: Detecção de Quedas. Cuidados de Saúde. Desenvolvimento de Software. Aplicativo Mobile. Inteligência Artificial. LSTM.

ABSTRACT

Falls are among the most common accidents in domestic and work environments. In some cases, they are responsible for consequences that directly impact the individual's quality of life, especially the elderly. Eventually, falls may occur when the person is alone, isolated or unable to ask for help. Evidently, for severe incidents, the faster the medical care the better are the recovery chances, and smaller is the impact on the patient's quality of life. Based on these findings, the need of solutions to increase the chances of proper care in case of fall becomes clear. In this context, this work presents the app iaQuedas and documents its development process. This app has the functionalities of presumable fall detection and its notification to a trusted contact via SMS. The detection is made through the presentation of accelerometer data to an artificial neural network of type LSTM, specialized on movement classification. The artificial intelligence model training is made with data from MobiAct 2.0 dataset, which has samples of daily-living activities and falls. The predictions made for the validation data at the end of the training shows high accuracy. During the tests in a daily usage context, falls are also properly detected. However, false-positive cases on fall detection are more frequent than expected. The final result is an application that fulfills its goals, but still has room for improvements.

Keywords: Fall Detection. Healthcare. Software Development. Mobile App. Artificial Intelligence. LSTM.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS OU SIGLAS

AM	- Aprendizado de máquina
IA	- Inteligência artificial
RNA	- Rede neural artificial
RNR	- Rede neural recorrente
LSTM	- Long short-term memory
VP	- Verdadeiro positivo
VN	- Verdadeiro negativo
FP	- Falso positivo
FN	- Falso negativo
SVM	- Support vector machine
K-NN	- K-nearest neighbors
UML	- Unified Modeling Language
SO	- Sistema operacional
JVM	- Java Virtual Machine
IDE	- Integrated development environment
SMS	- Short message service
RAM	- Random-access memory

SUMÁRIO

1 INTRODUÇÃO

As quedas são um importante problema de saúde pública, e nesse contexto, são definidas como eventos em que determinado indivíduo se desloca involuntariamente ao chão (WHO, 2021). Esse evento se consuma dada a incapacidade de recuperação de postura em tempo hábil (BUKSMAN et al., 2008), e ocorre pela conjunção de fatores intrínsecos e extrínsecos ao sujeito que sofre a queda (ALMEIDA et al., 2012; LEAMON; MURPY, 1995). Entre os fatores intrínsecos, destacam-se os problemas de equilíbrio, fraqueza, tonturas e vertigens, hipotensão postural, desordem visual, síncope e outros (RUBENSTEIN; JOSEPHSON, 2002). Por outro lado, problemas relacionados ao ambiente, como baixa iluminação, tapetes e degraus soltos, figuram como importantes fatores extrínsecos (ALMEIDA et al., 2012).

Este evento pode ocorrer com pessoas de quaisquer idades, mas as consequências geralmente são mais sérias quando vivenciado por pessoas idosas (LAYNE; POLLACK, 2004; LEAMON; MURPHY, 1995). Nesses casos, há um maior risco de morte, injúria e incapacitação, sendo que este último aspecto pode também gerar um grande custo social, devido ao risco da redução de independência e autonomia do idoso (FABRÍCIO et al., 2004). Este grupo de indivíduos também tende a sofrer mais quedas, algo comumente atribuído pelas mudanças fisiológicas que ocorrem naturalmente durante o processo de envelhecimento e também pela maior incidência de doenças clínicas, como a osteoporose (GARG, 1991).

Vários estudos indicam que as quedas são o acidente mais recorrente em ambientes domésticos (EVCI et al., 2006; GRAHAM; FIRTH, 1992; PIFFER et al., 2021). Sobretudo, esse mal não é exclusivo aos lares, pois também ocorrem com frequência em ambientes de trabalho, sendo constantemente reportados como um dos principais tipos de acidente (JENSEN et al., 2005; KEMMLERT; LUNDHOLM, 2001). Ademais, de acordo com o Departamento de Trabalho dos Estados Unidos da América, 15% das fatalidades no trabalho ocorrem por acidentes envolvendo quedas (OSHA, 2016).

Acidentes com quedas são comuns e podem ocorrer em diferentes circunstâncias. Porém, o que há de comum na maioria das vezes, é o padrão de movimentos que o corpo sofre. Tudo se inicia com a aceleração abrupta, seguido do impacto, e por fim, o período de inatividade. Isso permite que esse tipo de acidente

seja detectado através de tecnologias que monitoram e analisam padrões de movimento.

Este trabalho apresenta o desenvolvimento do iaQuedas, um aplicativo de smartphone que realiza o monitoramento dos movimentos do usuário e aplica técnicas de inteligência artificial para detectar possíveis eventos de queda. Uma vez detectados, este sistema possibilita o envio de uma mensagem automática para um contato de confiança, informando a data, hora e localização aproximada da queda.

1.1 JUSTIFICATIVA

Em emergências médicas, especialmente em quadros traumáticos graves, a rapidez e eficiência no atendimento médico possuem grande importância, tanto para a sobrevivência da vítima, quanto para evitar sequelas (DESLANDES, et al., 2006; MAYER, 1979). Um estudo sobre os casos de lesão da medula espinhal ocorridos entre 2004 à 2008 no estado australiano de Nova Gales do Sul, revelou que pelo menos 30% dessas lesões ocorreram por conta de quedas. Também se demonstrou que quanto maior o tempo entre a ocorrência e o atendimento médico, maior o risco de complicações. Nesse mesmo estudo, nota-se que para pessoas mais velhas, a identificação da lesão na medula espinhal ocorreu de forma mais lenta, sendo, portanto, mais um agravante para esse grupo (MIDDLETON et al., 2011).

Existem vários fatores que podem atrasar o atendimento médico, como a distância até a unidade hospitalar, a existência de fila no atendimento emergencial e o trânsito congestionado (MAYER, 1979). Contudo, é importante ressaltar que esse tempo também pode se estender devido a situações em que o paciente está impossibilitado ou possui dificuldades para procurar ajuda, como por exemplo durante um episódio de negação, um mecanismo psicológico que pode atuar como uma defesa contra picos de ansiedade em situações de emergência (OLIN; HACKETT, 1964). Esse tipo de incidência aliada ao fato de que 40 a 60% das quedas ocorrem na ausência de testemunhas (UNGAR et al. 2013), demonstram a importância na busca de soluções que ajudem a atenuar as más consequências associadas às quedas.

Além disso, devido a um sentimento de insegurança diante do perigo iminente de quedas, o idoso pode passar a restringir algumas de suas atividades por medo de sofrer uma queda. Este fenômeno é conhecido por *fear of falling*, e afeta diretamente a qualidade de vida dos mais velhos (LACHMAN et al., 1998).

Estima-se que atualmente existam cerca de 6,3 bilhões de usuários de *smartphone* distribuídos pelo mundo (STATISTA, 2021). Sabe-se também que mais de 70% desse mercado utiliza aparelhos que rodam o sistema operacional Android (STATCOUNTER, 2021).

Estudos realizados provam que quedas podem ser detectadas através do monitoramento de dados de diversos tipos de sensores, em especial dos acelerômetros (LINDEMANN et al., 2005). Estes sensores estão largamente presentes nos *smartphones* Android, que permitem que desenvolvedores de aplicativos os utilizem para o desenvolvimento de funcionalidades que envolvem movimento (ANDROID, 2019). Além disso, trabalhos correlatos a este, já demonstraram que é possível e viável desenvolver ferramentas para a detecção de quedas utilizando *smartphones* Android.

1.2 OBJETIVOS

1.2.1 Objetivo geral

Este trabalho tem como objetivo o desenvolvimento de um aplicativo de *smartphone* que realize a detecção e notificação de quedas.

1.2.2 Objetivos específicos

- Aplicar uma metodologia ágil durante o desenvolvimento do *software*;
- Utilizar algoritmos de inteligência artificial e dados de sensores do *smartphone* para realizar a detecção de quedas;
- Contribuir com a comunidade através da criação de uma ferramenta que possa aliviar os danos causados pelas quedas acidentais em idosos e trabalhadores.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho descreve o referencial teórico, conceitos que foram necessários para o desenvolvimento do aplicativo iaQuedas, bem como trabalhos relacionados que serviram como fonte de inspiração no capítulo 2.

Em seguida, no capítulo 3, são apresentados os materiais e métodos que foram utilizados durante a etapa de desenvolvimento. Nesse capítulo, são descritos os detalhes da base de dados utilizada, os tratamentos realizados nos dados, os detalhes do modelo de aprendizado de máquina e seu treinamento, e os requisitos do sistema. Além disso são descritos os recursos físicos e versões de *software* utilizadas para o desenvolvimento do aplicativo.

O capítulo 4 é dedicado aos resultados. Há um relatório das etapas de desenvolvimento, se discorre sobre o desempenho do modelo de inteligência artificial e também sobre os aspectos do aplicativo Android. Também são apresentadas as capturas de tela e a descrição do *workflow* existente entre elas.

Por fim, o capítulo 5 desenvolve as considerações finais, onde são apresentados os pontos de melhoria e recomendações para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

O iaQuedas é um aplicativo para smartphone Android que realiza detecção de quedas através do monitoramento de dados de aceleração utilizando inteligência artificial. Portanto, para a sua realização, é necessário o entendimento de conceitos de detecção de quedas e aprendizado de máquina, os quais são apresentados durante este capítulo.

Como o trabalho cobre o processo de desenvolvimento do aplicativo, existem também subcapítulos que tratam de conceitos de engenharia de software, plataformas, linguagens de programação, bibliotecas e ambientes de desenvolvimento integrado.

2.1 SENSORES E TÉCNICAS PARA DETECÇÃO DE QUEDAS

Em uma solução para a detecção de quedas utilizando *smartphones*, é necessário que exista evidência de tal evento. Essa evidência deve, portanto, ser capturada em tempo real através de sensores presentes no *smartphone*, e posteriormente processada para a tarefa de detecção de queda.

O guia de desenvolvimento oficial para aplicações Android, revela que a maioria dos aparelhos que utilizam essa plataforma, possui sensores integrados, os quais são capazes de medir movimentação, orientação e condições ambientais. Esses sensores podem disponibilizar leituras de alta precisão e acurácia (ANDROID, 2019).

Estudos relacionados apresentam soluções de detecção de quedas que utilizam sensores iguais ou semelhantes aos disponíveis em aparelhos *smartphones*, como por exemplo os acelerômetros e giroscópios (BOURKE et al., 2007; BOURKE et al., 2006; BOURKE, LYONS, 2008; ÖZDEMIR; BARSHAN, 2014).

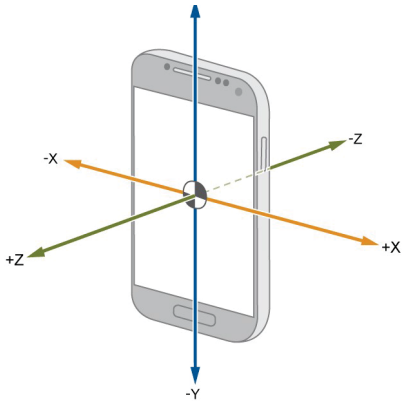
2.2 ACELERÔMETROS E GIROSCÓPIOS

O acelerômetro é um tipo de sensor que possui a capacidade de medir sua aceleração através de sinais elétricos. Esses sinais são gerados por estruturas microscópicas de cristal, presentes em sua construção, que ao serem estressadas devido às forças de aceleração, produzem uma tensão mensurável (JUNIOR et al., 2019; THOMAZINI; ALBUQUERQUE, 2011).

Esse tipo de sensor está presente nos *smartphones*, e é capaz de gerar sinais para os eixos X, Y e Z, conforme ilustrado na FIGURA 1. Essa capacidade permite deduzir a velocidade em que o dispositivo está se movendo e para qual direção aponta. Essa característica permite que seus dados sejam explorados em soluções que envolvem monitoramento físico (SIMAS et al., 2019).

O giroscópio, por sua vez, destina-se a medir a velocidade angular e é capaz de medir as mudanças de estado na inclinação (STEVAN; SILVA, 2015). Esses dados podem ser usados para efetuar correções na estabilização da câmera e detectar inclinações no aparelho *smartphone* (SIMAS et al., 2019).

FIGURA 1 - EIXOS DE LEITURA DO ACELERÔMETRO EM UM SMARTPHONE



FONTE: Mathworks (2021).

2.2.1 Magnitude do vetor de aceleração

Vavoulas et al (2013), sugere como um primeiro passo na tarefa de detecção de quedas usando acelerômetro, a obtenção do valor de magnitude do vetor de aceleração para cada leitura do sensor. Essa informação sintetiza a leitura dos sinais gerados pelo acelerômetro, por meio do cálculo da raiz quadrada da soma dos quadrados do vetor de aceleração, conforme representado na equação (1).

$$M_{ac} = \sqrt[2]{x^2 + y^2 + z^2} \quad (1)$$

Esse valor obtido em cada leitura pode então ser utilizado como um primeiro filtro na detecção de um possível evento de queda. Em uma abordagem sugerida por Sposaro e Tyson (2009), um possível evento de queda pode ser considerado quando os valores limiares de magnitude mínima 5,5, e máxima 16, ocorrem ambos em uma janela de leituras ao longo de 0,5 segundo.

2.3 APRENDIZADO DE MÁQUINA

O aprendizado de máquina (AM), é uma habilidade relacionada à área da inteligência artificial (IA). Essa subárea se caracteriza pela capacidade de seus algoritmos na tarefa de aprender características e criar uma representação do conhecimento presente em determinado conjunto de dados (FACELI et al., 2021).

Os dados utilizados na construção de um modelo de AM, devem ser compostos por atributos preditivos, que descrevem as características do objeto de estudo. Em muitos casos, há também o atributo alvo, que rotula a unidade de atributos preditivos, usando para isso uma classe ou valor numérico. Nestes casos, o processo de AM é denominado aprendizado supervisionado, pois são disponibilizados os dados que caracterizam o objeto, juntamente com uma informação complementar (atributo alvo) que o identifica quantitativa ou qualitativamente (LENZ et al., 2020).

Os detalhes sobre como o processo de aprendizado ocorre, diferem para cada técnica. Contudo, a utilização de um conjunto de dados para treinamento é comum entre todas. Esses dados devem representar o problema de forma generalizada o bastante, para que o modelo seja capaz de classificar ou estimar o atributo alvo corretamente ao ser confrontado com atributos preditivos inéditos (COPPIN, 2010).

Normalmente, os algoritmos de aprendizado supervisionado são usados para dois tipos de objetivos: a classificação, que se destina a apontar uma classe nominal para dados preditivos e a regressão, onde busca-se estimar um valor numérico contínuo.

Outra modalidade popular de aprendizagem é a não supervisionada. Esse tipo de técnica dispensa o atributo alvo, isto é, não é necessário que os dados estejam rotulados. Nesse caso, um dos objetivos desse tipo de aprendizado é que o algoritmo consiga organizar os dados em grupos com base em suas semelhanças (LENZ et al., 2020). Outra finalidade comum é a descoberta de padrões, regras e correlações com

algoritmos de associação. Essa categoria de aprendizado é utilizada para explorar bases de dados e revelar padrões (SILVA; PERES; BOSCARIOLI, 2016).

A área de AM possui também outras categorias de técnicas e algoritmos, como por exemplo o aprendizado por reforço. Esse tipo de técnica é muito comum no desenvolvimento de jogos e na robótica. Se utiliza o aprendizado por reforço, para que um modelo aprenda as sequências de decisões necessárias para atingir determinado objetivo. O aprendizado ocorre através do método da tentativa e erro, aplicando recompensas e penalidades ao algoritmo, de modo que com o passar do tempo, se maximize a recompensa total (RUSSEL; NORVIG, 2010).

2.3.1 Redes neurais artificiais

Um dos algoritmos mais populares na área de AM são as redes neurais artificiais (RNA). Essa técnica inspira-se na estrutura biológica do sistema nervoso, e tem por objetivo simular a capacidade de aprendizagem do cérebro humano.

A seguir são apresentados conceitos importantes para o entendimento dessa técnica.

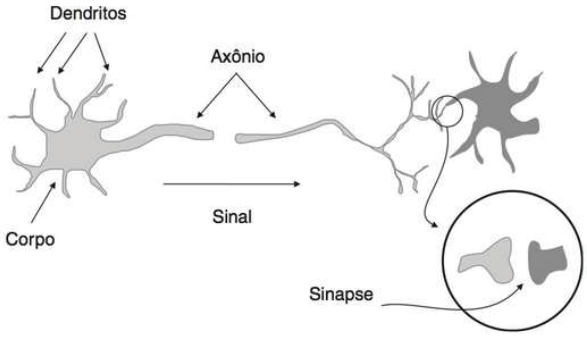
2.3.1.1 O neurônio biológico

A principal célula que compõe o sistema nervoso é o neurônio, o qual se distingue das demais células por ter a capacidade de reagir a estímulos. Isso confere ao neurônio o poder de transmitir impulsos nervosos a outros neurônios. A estrutura básica do neurônio biológico, ilustrada na FIGURA 2, pode ser resumida em:

- dendritos - prolongamentos que recebem estímulos nervosos;
- corpo - unidade que concentra os estímulos de todos os dendritos e após processá-los, gera um novo estímulo;
- axônio - prolongamento que transmite o estímulo gerado pelo corpo para outras partes (normalmente para dendritos de outros neurônios);
- sinapses - ponto de contato entre o axônio de um neurônio com dendritos de

outros neurônios.

FIGURA 2 - ESTRUTURA SIMPLIFICADA DE UM NEURÔNIO BIOLÓGICO



FONTE: Coppin (2010).

Essas células se conectam entre si e formam complexas redes, que permitem que os humanos aprendam e realizem várias tarefas desafiadoras ao mesmo tempo (FACELI et al., 2021).

2.3.1.2 O neurônio artificial

Apresentado ainda na primeira metade do século passado por McCulloch e Pitts (1943), o conceito de neurônio artificial busca simular matematicamente o funcionamento do neurônio biológico, resumidamente descrito na seção anterior.

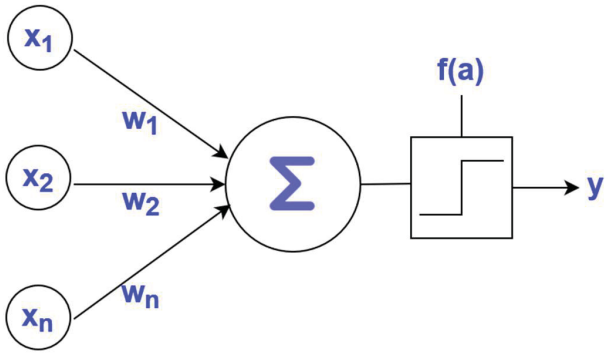
Nesse primeiro entendimento, um neurônio recebe entradas binárias, que podem ser excitatórias ou inibitórias. Quando se trata de um sinal excitatório, o valor de entrada é multiplicado por 1, e quando o sinal é inibitório, o valor da entrada é multiplicado por -1. Esses valores são somados, e o resultado é submetido a uma função de ativação degrau. Isto é, se o resultado ultrapassar um valor previamente determinado, a saída é igual a 1, e, caso contrário, igual a 0.

Mais tarde, uma importante evolução do conceito de neurônio artificial foi apresentada por Rosenblatt (1958). Esse modelo, batizado de *perceptron*, oferece a possibilidade de apresentação de dados numéricos contínuos, que são combinados através de uma soma ponderada por pesos definidos para cada conexão.

Conforme ilustrado na FIGURA 3, o *perceptron* recebe a entrada de dados $x_{1,\ \ldots\ ,}x_n$, que por sua vez são multiplicadas pelos pesos individuais $w_{1,\ \ldots\ ,}w_n$. Esses

resultados são somados e por fim processados por uma função de ativação do tipo degrau $f(a)$, que determina uma resposta de saída binária y.

FIGURA 3 - O NEURÔNIO ARTIFICIAL



FONTE: O autor (2022).

2.3.1.3 Funções de ativação

As funções de ativação são componentes chave na construção de um neurônio artificial. Elas cumprem o papel de processar o resultado da soma ponderada das conexões e determinar o sinal de saída do neurônio. Ao longo dos anos, a literatura apresentou diversas funções de ativação com características distintas (LIMA; PINHEIRO; SANTOS, 2014).

A função degrau, também conhecida por limiar, foi utilizada nas primeiras concepções de neurônio artificial e sua saída se limita aos valores 0 e 1. A equação (2) descreve essa função φ, com um valor limiar 0 e com v sendo o valor da soma dos sinais.

$$\varphi(v) = \begin{cases} 1 \; se \; v \geq 0 \\ 0 \; se \; v < 0 \end{cases} \tag{2}$$

A função descrita acima é aplicável apenas para modelos simples e possui limitações em relação aos problemas que é capaz de resolver. Conforme o estudo das redes neurais evoluiu para modelos mais complexos, foi necessário encontrar funções de ativação que oferecessem características de não linearidade e que possibilitassem o ajuste de pesos em arquiteturas de multicamadas. Para isso, funções como a

sigmoide foram introduzidas (SILVA et al, 2019). A função sigmoide possui a característica de gerar saídas contínuas entre 0 e 1, ou seja, o seu resultado é sempre positivo. Essa função pode ser descrita conforme a equação (3), onde a entrada líquida da unidade é representada por z.

$$\varphi(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

Dependendo do problema, é desejável que o sinal de saída represente também valores negativos. Para isso, uma das opções é a função tangente hiperbólica. Essa função é capaz de gerar saídas contínuas entre -1 e 1, e é semelhante à função sigmoide. Ela pode ser representada pela equação (4).

$$\varphi(z) = \frac{2}{1 + e^{-2\,z}} - 1 \tag{4}$$

Outra opção amplamente utilizada é a função de ativação linear retificada, mais conhecida como ReLU, que oferece as características de baixa complexidade e melhor desempenho em relação às anteriores. Essa função pode ser descrita conforme a equação (5).

$$\varphi(z) = \begin{cases} z \; se \; z \geq 0 \\ 0 \; se \; z < 0 \end{cases} \tag{5}$$

Para neurônios artificiais que tratam problemas de classificação, é comum a utilização da função *softmax*. Conforme demonstra a equação (6), essa função de ativação possui a característica de apresentar como saída um vetor com K posições contendo números contínuos entre 0 e 1, onde cada elemento na posição i representa a probabilidade dos dados de entrada pertencerem à determinada classe. Esse cálculo é realizado com base nos sinais z_i, recebidos pela função de ativação. Dessa maneira, a soma de todos os elementos desse vetor de saída resulta em 1 (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$\varphi(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{6}$$

2.3.1.4 A rede neural artificial

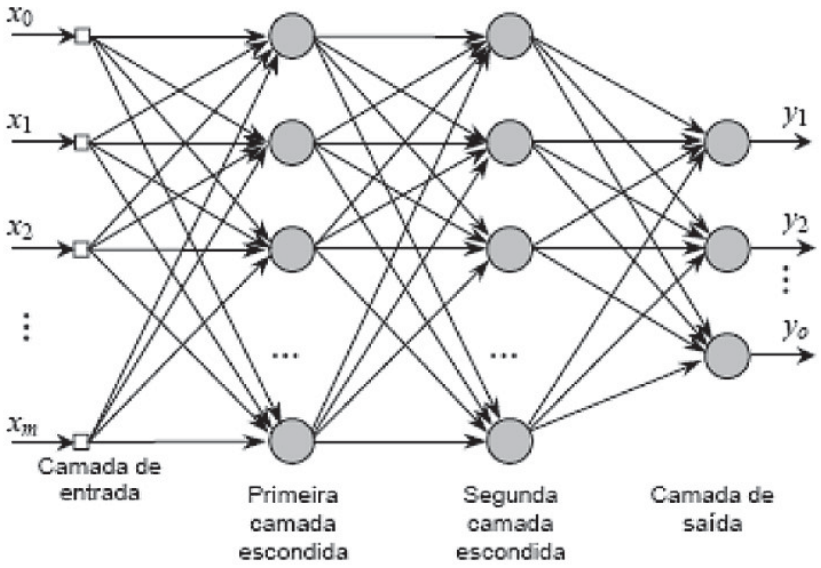
Assim como no sistema nervoso, estruturas análogas ao neurônio artificial desenvolvido por Rosenblatt (1958), podem ser conectadas entre si, formando redes, que ao serem estimuladas com dados em uma camada de entrada, são capazes de produzir sinais em uma camada de saída. Esse arranjo é conhecido como rede neural artificial.

Além das camadas de entrada e de saída, sua arquitetura é geralmente formada por uma ou mais camadas ocultas. Essas camadas representam um grupo de neurônios conectados aos elementos de outra camada, conforme é ilustrado na FIGURA 4. Assim como na estrutura do *perceptron*, cada conexão entre os neurônios possui um valor de peso ajustável, que juntamente da função de ativação, influenciam diretamente no valor do sinal repassado para os neurônios da camada seguinte (LIMA; PINHEIRO; SANTOS, 2014).

Na camada de saída, são introduzidos os neurônios responsáveis por apresentar o resultado do processamento. Esses componentes são construídos com base no tipo do resultado esperado. Por exemplo: se o problema se tratar de uma regressão, normalmente haverá apenas um neurônio com uma função de ativação que estimará um número contínuo. Se for o caso de uma classificação binária, a camada de saída também pode ser construída com apenas um neurônio, mas com uma função de ativação que estime um valor entre 0 e 1. Já para problemas de classificação com mais de duas classes, a camada de saída poderá ter uma quantidade de neurônios equivalentes ao número de categorias, com uma função de ativação que resulte em um vetor contendo uma distribuição de probabilidade, como a *softmax*.

Existem diversas arquiteturas de RNA, sendo a mais conhecida, a rede neural *feedforward*, que recebe esse nome em razão do seu processamento ocorrer em apenas um sentido, da camada de entrada para a camada de saída. Outra arquitetura popular é rede neural recorrente, chamada assim por conta da existência de retroalimentação em suas camadas, isto é, a saída de um neurônio pode estar ligada à entrada de neurônios precedentes, ou deste mesmo neurônio antes do sinal prosseguir para a camada seguinte.

FIGURA 4 - RNA COM MAIS DE UMA CAMADA OCULTA



FONTE: Olivera et al. (2010).

2.3.1.5 As redes neurais recorrentes

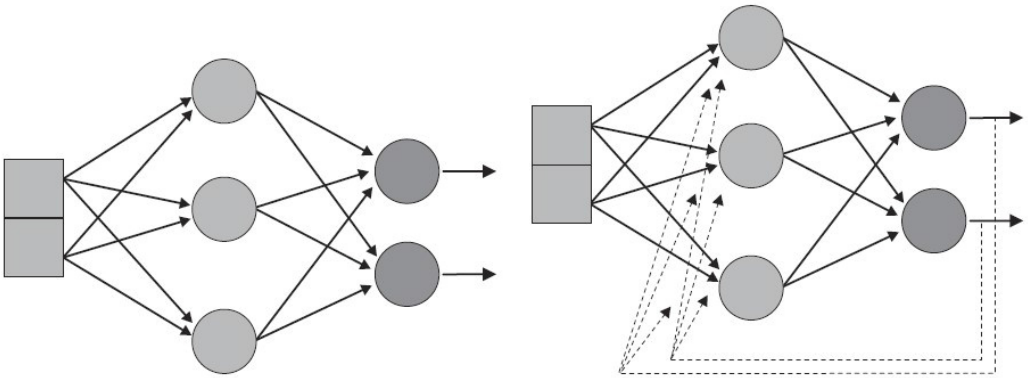
Frequentemente, o treinamento de RNA utiliza conjuntos de dados em que cada amostra é constituída por um número fixo de atributos. Por exemplo, no conjunto de dados da flor íris (ANDERSON, 1936), cada amostra possui um atributo alvo discriminando a espécie de íris, e medições da altura e largura das pétalas e sépalas. Um modelo preditivo baseado em redes neurais clássicas, para esse problema, não enfrentaria um grande desafio, afinal cada atributo poderia ser submetido a um nó da camada de entrada, o qual posteriormente repassaria a saída para neurônios das demais camadas, contribuindo singularmente para o processo de aprendizagem via ajuste de pesos.

Contudo, para alguns problemas, as amostras são representadas por séries temporais, fazendo com que as características a serem aprendidas não estejam sintetizadas em variáveis singulares, mas sim inerentes às variações que ocorrem ao decorrer da série e a ordem em que valores aparecem historicamente. Nesses casos, uma RNA clássica pode não ser o suficiente para alcançar êxito no processo de

aprendizagem, uma vez que nessa abordagem não existem mecanismos que memorizem aspectos da sequência.

As redes neurais recorrentes (RNR), por sua vez, podem ser utilizadas com maior êxito para realizar tarefas que envolvam informações sequenciais (FACELI, et al., 2021). Isto porque seu funcionamento se dá através da existência de retroalimentação entre as camadas, e estados ocultos que realizam uma espécie de memorização. De modo simplificado, pode-se dizer que as RNR se valem do reprocessamento das saídas, que por sua vez interagem e modificam os estados ocultos, os quais guardam informações relevantes de entradas anteriores da sequência, garantindo que o valor de saída final esteja baseado em toda a sequência (GRUS, 2021). A FIGURA 5 ilustra essa topologia à direita, em comparação com uma RNA clássica à esquerda, também conhecida como *feedforward*.

FIGURA 5 - TOPOLOGIA DE RNA CLÁSSICA E RNR



FONTE: Faceli et al. (2011).

Contudo, as RNR são suscetíveis a um problema conhecido como desaparecimento de gradiente. Esse fenômeno ocorre quando o cálculo de gradiente resulta em valores muito baixos durante a atualização dos pesos via algoritmo *back-propagation*. Devido ao grande número de pesos envolvidos em um modelo de RNR, a tendência é que o valor de gradiente diminua a cada multiplicação, dificultando a atualização dos pesos e atrasando a convergência do modelo. Além disso, o cálculo de atualização de pesos nas RNR está ligado aos passos da série temporal de entrada, e nesse contexto, o desaparecimento de gradiente significa que, durante o aprendizado, valores do início da sequência perdem relevância em detrimento de

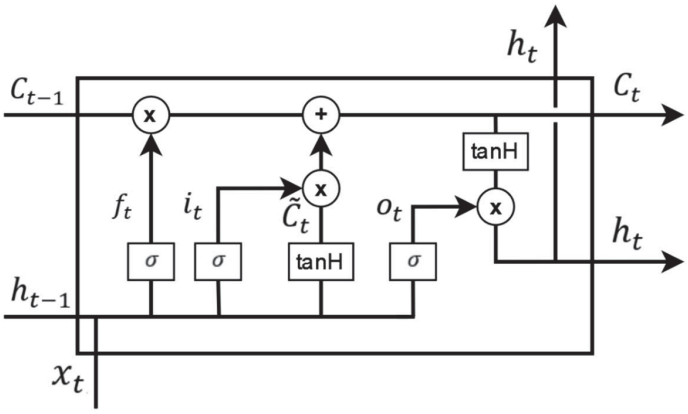
entradas mais recentes. Em outras palavras, RNR não são tão eficientes para aprender aspectos de longo prazo.

2.3.1.6 LSTM

Para minimizar o fenômeno do desaparecimento de gradiente, outras arquiteturas de RNR foram criadas, como é o caso das redes *Long Short-Term Memory,* do inglês, memória de curto prazo longa, mais conhecidas simplesmente por LSTM (SEJNOWSKI, 2019).

Esse tipo de RNR propõe uma arquitetura mais robusta, baseada em células que também possuem um estado, o qual é capaz de memorizar informações relevantes da série temporal. O controle dessa memorização é baseado em estruturas conhecidas como *gates*, que definem quais informações devem ser adicionadas, mantidas ou esquecidas durante o processamento das séries temporais. Para isso, existem 3 *gates* na célula LSTM, denominados por *forget*, *input* e *output*. Essas estruturas são construídas com funções de ativação sigmoide, a qual resulta em valores entre 0 e 1. O resultado obtido dessas funções define a quantidade de informação que deve ser propagada para as operações seguintes (GOODFELLOW; BENGIO; COURVILLE, 2016). A FIGURA 6 ilustra a estrutura de uma célula LSTM.

FIGURA 6 - REPRESENTAÇÃO DE CÉLULA LSTM



FONTE: O autor (2022).

No primeiro passo, a célula LSTM calcula quais informações devem ser esquecidas do estado através do *forget gate*. Para isso, o estado oculto h_{t-1} e o dado

de entrada x_t são ponderados por um peso w_t. Esse valor é submetido a uma função sigmoide σ, resultando em um valor f_t, conforme demonstra a equação (7). O estado oculto h_{t-1} é o valor de saída h_t no passo anterior.

$$f_t = \sigma(w_t[h_{t-1}, x_t]) \tag{7}$$

Em seguida, a célula LSTM decide quais novas informações devem ser adicionadas ao estado da célula, e isso é realizado em duas etapas. Primeiramente aplica-se a função sigmoide do *input gate* ao valor do estado oculto e ao dado de entrada ponderados pelo peso w_i. Esse processamento resulta em um valor i_t, conforme descreve a equação (8). Em seguida, o dado de entrada e estado oculto são ponderados por um peso w_c e processados por uma função do tipo tangente hiperbólica, resultando em um valor candidato $\tilde{C}_t$. Esse procedimento é descrito pela equação (9).

$$i_t = \sigma(w_i[h_{t-1}, x_t]) \tag{8}$$

$$\tilde{C}_t = tanH(w_c[h_{t-1}, x_t]) \tag{9}$$

Para a atualização do estado da célula, aplica-se uma multiplicação entre o valor do estado C_{t-1} e f_t, obtido previamente pelo *forget gate*. Em seguida, é realizada uma soma entre esse valor e o produto do valor candidato $\tilde{C}_t$ e i_t, previamente obtido pelo *input gate*. Essa operação é representada pela equação (10).

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t \tag{10}$$

Na última etapa de processamento, a LSTM calcula o valor de saída da célula. Para isso, a função sigmoide do *output gate* obtém o valor de O_t ao ser aplicada ao valor do estado oculto e ao dado de entrada, ponderados pelo peso w_o, conforme demonstra a equação (11). Em seguida, o estado da célula C_t é submetido a uma função do tipo tangente hiperbólica, e por fim, o resultado é multiplicado pelo valor de O_t, gerando a saída h_t, conforme descreve a equação (12).

$$o_t = \sigma(w_o[h_{t-1}, x_t]) \quad (11)$$

$$h_t = tanH(C_t)o_t \quad (12)$$

Esses mecanismos garantem que haja um aprendizado mais consistente, tanto para características de curto prazo, quanto para características de longo prazo, minimizando o fenômeno do desaparecimento de gradiente.

2.3.1.7 Treinamento de uma RNA

Para que uma RNA adquira a capacidade de classificação ou regressão, é necessário que os pesos das conexões entre os neurônios estejam ajustados para resolver o problema em questão.

Normalmente, as conexões da RNA são iniciadas com valores de pesos aleatórios, e durante a etapa de treinamento, esses valores são ajustados de modo que se aproximem de uma combinação eficaz para o problema. Esse procedimento se dá através da apresentação dos dados preditivos aos neurônios da camada de entrada, seguido do processamento da RNA e verificação do resultado na camada de saída. Ao fim de cada execução (época), são mensurados os erros cometidos, e como esse resultado está diretamente ligado aos pesos das conexões, são realizados ajustes equivalentes, com o objetivo de diminuir os erros e melhorar o desempenho nas próximas execuções (GOLDSCHMIDT; PASSOS; BEZERRA, 2015). Pode-se dizer que a conjuntura final dos pesos representa o conhecimento adquirido por uma RNA após a conclusão do treinamento (LIMA; PINHEIRO; SANTOS, 2014).

2.3.1.8 Algoritmo back-propagation

Normalmente utiliza-se o algoritmo *back-propagation* para realizar o treinamento de uma RNA. Esse algoritmo aplica o método de descida de gradiente, que consiste na otimização iterativa de valores dos parâmetros que minimizam uma função de interesse. No caso do problema utilizado no *back-propagation*, a função a ser minimizada é uma função de erro, também conhecida por função de custo, enquanto os parâmetros a serem ajustados são os pesos das conexões dos neurônios (SEJNOWSKI, 2019).

Em uma RNA *feedforward* de 3 camadas, onde I é a camada de entrada, J uma camada oculta, e K a camada de saída, o erro de um neurônio k da camada de saída na iteração n pode ser representado pela equação (13), onde d_k é a saída desejada e y_k é a saída proporcionada pelo neurônio k.

$$e_k(n) = d_k(n) - y_k(n) \tag{13}$$

A entrada líquida desse neurônio pode ser expressa pela equação (14), onde p é o número de elementos da camada oculta, j a polarização do neurônio k, w_{kj} é determinado peso da conexão entre o neurônio k e o neurônio da camada anterior, e y_j é o sinal de saída de determinado neurônio da camada anterior.

$$u_k = \sum_{j=0}^{p} w_{kj}(n) y_j(n) \tag{14}$$

A saída y_k é obtida pela aplicação da função de ativação na entrada líquida do neurônio, conforme a equação (15).

$$y_k = \varphi(u_k) \tag{15}$$

O ajuste de pesos que caracteriza o algoritmo de *back-propagation* é realizado em w_{kj} e deve ser proporcional ao gradiente do erro em relação aos pesos, que após a aplicação da regra de cadeia pode ser representado pela equação (16).

$$\frac{\partial(n)}{\partial w_{kj}(n)} = -e_k(n)\varphi'_k(u_k(n))y_j(n) \tag{16}$$

Contudo, o cálculo do gradiente resulta na direção de máximo crescimento do erro em relação à variação que ocorre nos pesos, portanto, deseja-se obter a direção contrária, que pode ser representada com a aplicação da regra delta. Essa operação pode ser observada nas equações (17) e (18).

$$\Delta w_{kj}(n) = \delta_k(n)\, y_j(n) \tag{17}$$

$$\delta_k(n) = e_k(n)\varphi'_k\big(u_k(n)\big). \tag{18}$$

O valor de $\Delta w_{kj}(n)$ deve ser, portanto, utilizado para a atualização do peso da conexão entre determinado neurônio da camada oculta com determinado neurônio da camada de saída.

Após o ajuste realizado nos pesos que conectam os neurônios da camada de saída, o algoritmo *back-propagation* deve realizar o ajuste de pesos das camadas ocultas. Para isso, deve-se obter gradiente do erro em relação aos pesos que conectam os neurônios da camada I com os neurônios da camada J. Esse valor pode ser representado pela equação (19).

$$\frac{\partial(n)}{\partial w_{ji}(n)} = \left(-\sum_k \delta_k(n) w_{kj}(n)\right) \varphi'_j(u_j(n)) y_i(n) \tag{19}$$

Portanto, com a aplicação da regra delta, o valor para a atualização dos pesos pode ser obtido pelas equações (20) e (21).

$$\Delta w_{ji}(n) = -\frac{\partial(n)}{\partial w_{ji}(n)} = \delta_j(n) y_j(n) \tag{20}$$

$$\delta_j(n) = \varphi'_j(u_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{21}$$

Os cálculos para a atualização dos pesos entre neurônios da camada I e J podem ser estendidos para problemas que contenham mais camadas, de modo que os ajustes sejam realizados até que se atinja a camada de entrada. Ao término desse processo, os passos são repetidos, iniciando pela apresentação de dados na camada de entrada para que então se obtenha novos valores de erro (LIMA; PINHEIRO; SANTOS, 2014).

2.3.1.9 Taxa de aprendizado

A taxa de aprendizado é um parâmetro que pode ser utilizado durante a atualização dos pesos no algoritmo de *back-propagation*. Esse valor exerce forte influência no tempo de convergência da rede, pois ele é responsável por ponderar a alteração dos pesos em cada iteração do algoritmo de aprendizado. Se essa taxa for muito pequena, é possível que se gaste muito tempo para atingir um bom conjunto de pesos. Por outro lado, se a taxa tiver um valor alto, é provável que existam grandes oscilações entre os valores de erro das iterações (FACELI et al., 2021).

A equação (22) demonstra a aplicação de uma taxa de aprendizado α para definir a atualização do peso w_{ji}.

$$w_{ji}(n+1) = w_{ji}(n) + \alpha\, \Delta w_{ji}(n) \tag{22}$$

2.3.1.10 Funções de perda

O treinamento de uma RNA depende da definição de uma função de perda, também conhecida como função de custo ou função de erro. O resultado desse cálculo demonstra o desempenho do modelo após cada iteração do treinamento. Existem diversas funções de perda que se aplicam para diferentes cenários.

Para problemas de regressão, comumente se utiliza a função do erro quadrático médio, também conhecido por MSE, do inglês *mean squared error* (SILVA et al., 2019). A MSE pode ser descrita conforme a equação (23), onde y representa o valor esperado, $\hat{y}$ o resultado obtido e n o número de observações.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2 \tag{23}$$

Em problemas de classificação, pode-se utilizar a função da entropia cruzada. Essa função de perda analisa a distância de determinado vetor de probabilidades obtidas em um neurônio de saída em relação ao resultado esperado. Esse tipo de valor de saída, pode ser obtido com a utilização de uma função de ativação *softmax*.

A função da entropia cruzada é representada pela equação (24), onde $p(x)$ é o valor de probabilidade esperado para uma classe x, e $q(x)$ o valor obtido.

$$CE(p,q) = -\sum_{x} p(x)\ log\ q(x) \tag{24}$$

Para o uso da função de perda de entropia cruzada para problemas com mais de uma classe, é recomendável que o atributo alvo esteja representado em notação *one-hot encoding*. Essa técnica de representação dos dados consiste na transformação de determinado atributo em um vetor com valores binários. Por exemplo, para as classes da base de dados da flor Iris, as combinações possíveis em notação *one-hot encoding* podem ser representados pelas linhas da TABELA 1.

TABELA 1 - REPRESENTAÇÃO ONE-HOT ENCODING DE IRIS

Classe	Primeira posição	Segunda posição	Terceira posição
Setosa	1	0	0
Virginica	0	1	0
Versicolor	0	0	1

FONTE: O autor (2022).

2.3.2 Separação dos dados de treinamento

Durante o treinamento de um modelo preditivo, como uma RNA, é recomendável que os dados utilizados sejam diferentes daqueles que serão aplicados na etapa de avaliação, onde se verificam o desempenho e capacidade preditiva do modelo. A avaliação do modelo se beneficia de dados inéditos devido a um fenômeno conhecido por *overfitting* (SILVA; PERES; BOSCARIOLI, 2016).

O *overfitting* ocorre quando o modelo está excessivamente ajustado para acertar o atributo alvo do conjunto de dados utilizado na etapa de treinamento, mas não apresenta o mesmo alto desempenho com dados inéditos. Uma das ações que ajudam a evitar esse fenômeno é a separação dos dados em subconjuntos de treino e validação.

Essa separação é frequentemente realizada através da técnica *holdout*¸ que consiste em escolher e separar entradas do conjunto de dados aleatoriamente, até que se satisfaça uma porcentagem previamente definida. Tradicionalmente, são separados 70% dos dados para treinamento, enquanto os 30% restantes são destinados à avaliação.

2.3.3 Métricas de avaliação para modelos de classificação

Normalmente após o treinamento de um modelo de AM supervisionado, é promovida uma etapa de avaliação para que se mensure a qualidade do classificador. Para isso, um conjunto de dados supervisionados inéditos é submetido ao modelo, que realizará a predição dos atributos alvo com base no conhecimento obtido. O resultado de cada predição é comparado com o atributo alvo original, e deste modo verificam-se os erros e acertos (MONARD; BARANAUSKAS, 2003).

2.3.3.1 Taxa de erro e acurácia

O desempenho do classificador pode ser representado através das taxas de erro e acurácia.

A taxa de erro $err(\hat{f})$ é calculada através do número de classificações incorretas em um conjunto de dados com n entradas. Esse cálculo pode ser representado conforme a equação (25), onde $I = 1$ quando a classe predita $\hat{f}(x_i)$ é diferente do atributo alvo original y_i, e $I = 0$ quando $\hat{f}(x_i)$ é igual a y_i (FACELI et al., 2021).

$$err\left(\hat{f}\right) = \frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{f}(x_i)\right) \tag{25}$$

O valor da taxa de erros pode variar entre 0 e 1 e quanto mais próximo de 0, melhor é a qualidade do modelo. A acurácia, por sua vez, é uma informação complementar, e representa a taxa de acertos. O seu cálculo pode ser realizado subtraindo a taxa de erro por 1 (FACELI et al., 2021), conforme ilustra a equação (26).

$$ac(\hat{f}) = 1 - err(\hat{f}) \tag{26}$$

2.3.3.2 Matriz de confusão

Outra forma de avaliar um classificador é através da confecção da matriz de confusão. Essa ferramenta agrupa detalhes sobre o desempenho do teste, demonstrando em um quadro a relação de erros e acertos para cada uma das classes. Nessa matriz, as linhas representam as classes reais, enquanto as colunas

representam os valores preditos. As células da diagonal principal possuem o número total de acertos para cada classe, enquanto cada uma das demais células apresentam a soma de classificações erradas estratificadas por classe (FACELI et al, 2021).

Em um problema de detecção de quedas, havendo as classes “queda” e “atividade normal”, a matriz de confusão possui dimensão 2x2. A FIGURA 7 ilustra um resultado fictício para uma amostragem de 106 entradas, em que 47 delas são exemplos de queda e 59 são exemplos de atividade normal.

FIGURA 7 - EXEMPLO DE MATRIZ DE CONFUSÃO

		VALOR PREDITO	
		QUEDA	ATIV. NORMAL
VALOR REAL	QUEDA	45 (VP)	2 (FN)
	ATIV. NORMAL	4 (FP)	55 (VN)

FONTE: O autor (2022).

Neste exemplo simples, convém-se nomear os dois acertos possíveis de verdadeiro positivo (VP) para representar as quedas, e verdadeiro negativo (VN) para representar as atividades normais. De maneira oposta, os erros são denominados falso positivo (FP) para atividades normais detectadas como quedas e falso negativo (FN) para quedas que foram interpretadas como atividades normais.

2.3.3.3 Sensibilidade e especificidade

Com os valores postos em uma matriz de confusão, outras métricas relevantes podem ser inferidas. Como por exemplo a sensibilidade e especificidade do modelo.

A sensibilidade demonstra a efetividade que um classificador possui para acertar os exemplos positivos. Este valor é calculado conforme a equação (27). Na tarefa de detecção de quedas, quanto mais próxima de 1, maior o número de quedas

classificadas corretamente (MONARD; BARANAUSKAS, 2003). Contudo, uma sensibilidade igual a 1 pode não significar que o modelo cumpre seu papel perfeitamente, afinal um classificador enviesado que identifica todo e qualquer evento como queda, também forneceria uma sensibilidade igual a 1.

De modo semelhante, a especificidade demonstra a capacidade de um modelo para acertar os exemplos negativos. Seu resultado é obtido pela equação (28) (MONARD; BARANAUSKAS, 2003). No contexto da detecção de quedas, uma baixa especificidade significaria um alto número de alarmes falsos.

Essas métricas se complementam, e considerá-las na avaliação do modelo é algo importante, afinal, dependendo da prioridade da solução, uma sensibilidade mais alta é naturalmente preferível (POWERS, 2011), sendo este o caso da detecção de quedas. Isto é, para que o detector de quedas prospere em seu objetivo de diminuir os riscos relacionados às quedas desassistidas, é mais prudente que se busque uma alta sensibilidade, mesmo que isso custe a ocorrência de alguns alarmes falsos.

$$sensibilidade = \frac{VP}{VP+F} \tag{27}$$

$$especificidade = \frac{VN}{VN+F} \tag{28}$$

2.4 TRABALHOS RELACIONADOS

Diversos trabalhos destinados à detecção de quedas foram propostos até o momento. Essas soluções baseiam-se na leitura individual ou conjunta de sensores presos à diferentes partes do corpo do sujeito, integrados a *smartphones* ou a dispositivos *wearables*. Entre esses sensores, acelerômetros e giroscópios figuram entre os mais utilizados (ISLAM et al., 2019). Alguns trabalhos se diferenciam ao explorar com sucesso a captura de sons do microfone (CHEFFENA, 2016).

2.4.1 Detecção por threshold

Em geral, a técnica mais utilizada para a detecção de um evento de queda é o *threshold*. Esse método se baseia no monitoramento dos sinais provenientes do acelerômetro ou do giroscópio, seguido da comparação com valores limiares,

previamente estabelecidos, que caracterizam uma queda. Esses valores são definidos através do estudo de dados coletados em eventos de queda e atividades comuns.

Bourke et al. (2006) e (2007) propuseram algoritmos de *threshold* que utilizam dados de sensores acelerômetros presos ao tronco e coxa do indivíduo, e conseguiram a façanha de atingir 100% de detecção das quedas simuladas no estudo. Outro exemplo de sucesso é apresentado por Bourke e Lyons (2008), que também atinge a perfeição utilizando *threshold*, mas com a leitura de sensores giroscópios. Lee e Tseng (2019) também apresentam a aplicação de um algoritmo de *threshold* com bons resultados, mas o faz utilizando o sensor de acelerômetro integrado aos *smartphones*.

Özdemir e Barshan (2014) apontam limitações na utilização de *thresholds* para a detecção de quedas, uma vez que não existem valores limiares universais, isto é, um evento de queda gera sinais de acelerômetro e giroscópio diferentes para cada indivíduo. Isso coloca em xeque a qualidade desses algoritmos em aplicações mais realistas, ou seja, fora do ambiente controlado de um laboratório.

2.4.2 Detecção por técnicas de aprendizado de máquina

A fim de superar as limitações da técnica de *threshold*, alguns estudos propõem abordagens menos determinísticas para a detecção de quedas. Essas abordagens se caracterizam pelo uso de algoritmos de aprendizado de máquina sobre dados coletados pelo mesmo tipo de sensores apresentados anteriormente.

Vallabh et al. (2016) realizaram experimentos com a leitura do acelerômetro em diferentes algoritmos de aprendizado de máquina e conseguiram melhores resultados com o uso do algoritmo SVM, atingindo mais de 87% de acurácia na detecção de quedas. Özdemir e Barshan (2014) por sua vez, realizaram um estudo capturando dados de vários acelerômetros presos em diferentes partes do corpo, e ao testar diversas técnicas, conseguiram a maior acurácia para a detecção de quedas com o algoritmo K-NN, com 99% de acertos.

Abbate et al. (2012) demonstram que é possível utilizar a técnica de *threshold* em conjunto com algoritmos de aprendizado de máquina para a detecção de queda. Assim como Tsinganos e Skodras (2017), um sistema monitora os dados do acelerômetro do *smartphone* em tempo real, aplicando verificações de *threshold*, que detectam eventos de possível queda. Esse evento inicial, dispara uma classificação

pelo modelo de aprendizado de máquina, que é o responsável por confirmar que o evento realmente trata-se de uma queda. Desse modo, o *threshold* atua como um filtro que reduz a carga de processamento, e por consequência, o consumo de bateria. Por fim, esses dois trabalhos também apresentam um sistema de notificação, o qual é acionado após a expiração de uma tela de confirmação de falso-positivo. Essa etapa adicional evita que classificações de queda espúrias disparem notificações indevidas. Para Tsinganos e Skodras (2017), o melhor resultado foi alcançado com um algoritmo K-NN, com 95% de acurácia.

2.5 ENGENHARIA DE SOFTWARE

Assim como ocorre na elaboração de qualquer produto, normalmente, um aplicativo é concebido através de processos que utilizam técnicas de engenharia de *software*. Essa área da engenharia conta com vários métodos que auxiliam na definição, modelagem e planejamento de projetos de sistemas. A sua aplicação adequada conduz à produção de sistemas de alta qualidade (PRESSMAN; MAXIM, 2016).

Contudo, é evidente que uma ampla utilização dos artefatos consagrados da engenharia de *software* exige tempo e esforço consideráveis. Portanto, para garantir a viabilidade de um projeto, é sempre necessário encontrar o equilíbrio entre a aplicação dessas técnicas, o tempo e a mão de obra disponível.

Conquanto, alguns processos são imprescindíveis para qualquer projeto de *software*. Este é o caso das técnicas relacionadas à especificação de requisitos, isto é, a definição das características que permeiam todo o conceito do *software*. Também se destaca nesse quesito, a produção de diagramas, a qual é comumente realizada seguindo o padrão UML, do inglês, *Unified Modeling Language*. Esse padrão para a elaboração de projetos de *software* recomenda uma série de diagramas para representar e documentar os detalhes de funcionamento e características esperadas de um *software* (FOWLER, 2011).

2.5.1 Especificação de requisitos

O processo de desenvolvimento de um sistema envolve diversas etapas. Segundo Peters e Pedrycsz, uma das mais importantes é a elaboração de uma

definição formal que seja capaz de sintetizar as especificações e características do projeto, ou seja, tudo o que o sistema deve ou não fazer (2001). Esse procedimento está compreendido na área de engenharia de requisitos e tem por objetivo final a produção de um documento que cubra aspectos técnicos e gerais do *software*. Deseja-se que esse documento possa ser consultado e compreendido tanto por quem não possui conhecimento técnico em desenvolvimento de *software* quanto para os profissionais que vão construir de fato o produto (SOMMERVILLE, 2003).

2.5.2 Diagrama de casos de uso

Frequentemente, a definição dos requisitos não é o suficiente para prover uma visão completa dos processos e interações existentes em um projeto de *software*. Para tal tarefa, uma das soluções mais utilizadas é a representação do sistema em diagramas de casos de usos.

Esse diagrama é constituído por atores e casos de uso. Os atores podem ser pessoas, equipamentos, sensores e entre outras entidades que interagem com o sistema. Normalmente, os atores são colocados nas extremidades do diagrama e são representados por figuras que os ilustrem, por exemplo, um boneco palito quando o ator se trata de um humano. Já os casos de uso são as ações existentes no sistema. Essas ações são representadas por figuras de elipse com uma frase curta que sintetiza o procedimento (REINEHR, 2020).

Os atores são conectados aos casos de uso que interagem. Os casos de uso também podem estar conectados entre si, com uma seta de linha pontilhada. As relações podem indicar que um caso de uso ocorre obrigatoriamente após o outro, nesse caso, a seta é identificada com a expressão *include*. Outro tipo comum de conexão é quando determinado caso de uso ocorre opcionalmente ou eventualmente após o outro, nesse caso, a expressão *extend* ilustra o relacionamento (SILVA; VIDEIRA, 2001).

2.5.3 Diagrama de atividades

Outra importante ferramenta definida pelo padrão UML é o diagrama de atividades. Ele é uma espécie de fluxograma, e, portanto, é capaz de representar visualmente regras de negócio e modos específicos de cumprir tarefas.

O padrão UML define uma vasta série de símbolos e regras para compor o diagrama de atividades, mas de modo geral, os elementos mais importantes são as ações, representadas por elipses contendo uma frase curta, e as decisões, representadas por losangos que dividem o fluxo da atividade com setas, contendo a condição que leva a cada mudança de fluxo (FOWLER, 2011).

2.5.4 Diagrama de classes

O padrão UML também atua em um contexto mais próximo do processo de construção do sistema, isto é, a programação. Este é o caso do diagrama de classes, que auxilia no planejamento de sistemas que utilizam o paradigma da programação orientada a objetos. De modo simplificado, pode-se dizer que essa técnica de programação separa as entidades que compõem os processos do sistema em objetos. Isto é, abstrações que possuem os atributos e funções que caracterizam a entidade. As classes são a descrição formal dos objetos, portanto são elas que carregam a definição dos atributos e toda a lógica das funções, que serão utilizadas pelos objetos.

Nesse tipo de diagrama, as classes são representadas por retângulos contendo o nome, seus atributos e funções. Também pode ser descrita a relação entre as classes e sua multiplicidade, ou seja, a quantidade de objetos que estão associados nas relações (PRESSMAN; MAXIM, 2016).

2.5.5 Metodologias de desenvolvimento

As etapas que envolvem a construção de um sistema, são cumpridas com base em algum modelo de projeto que melhor se enquadre à situação. Existem diversos modos de fazer, que são formalmente chamados de metodologias de desenvolvimento. Elas indicam a forma com que o *software* será documentado, desenvolvido, testado e implantado.

As metodologias existem para organizar o processo e proporcionar benefícios de qualidade. Tonsig as define como “um roteiro de trabalho, constituído em geral de macro-etapas com objetivos funcionais na construção de um *software*, onde também é possível visualizar-se a interdependência existente entre as macro-etapas” (2003, p. 56).

2.5.5.1 Scrum

Scrum é uma metodologia de gerenciamento de projetos muito utilizada na área de desenvolvimento de *software*.

Essa metodologia é baseada na entrega de resultados após curtos ciclos conhecidos por *sprints*. No planejamento de uma *sprint*, uma série de objetivos ou tarefas são colocados como meta de entrega, e ao final do período, se espera que todos os objetivos sejam cumpridos. Deste modo, o projeto é entregue de forma incremental.

Em cada dia da *sprint*, os membros da equipe Scrum se reúnem para reportar evoluções e anunciar dificuldades. Essas reuniões possuem a característica de serem curtas.

As tarefas utilizadas na *sprint* são retiradas de uma pilha de objetivos, ordenada por prioridade, conhecida como *Product Backlog*. Pode-se dizer que os requisitos do *software* estão fragmentados no *Backlog* no formato de tarefas. À medida que novas necessidades são identificadas, o *Backlog* é realimentado com novas tarefas.

Nessa metodologia existe uma separação de papéis. Há os membros da equipe, que são as pessoas que trabalham de forma auto gerenciada, e que são os responsáveis diretos por cumprir as tarefas da *sprint*; O *product owner*, que é o responsável por definir e priorizar o *Product Backlog*; E o scrum *master*, que atua como um membro da equipe responsável por solucionar problemas vivenciados pela equipe e garantir que a metodologia Scrum esteja sendo executada da maneira correta (COHN, 2011).

2.5.5.2 Scrum Solo

A Scrum Solo é uma metodologia que se inspira e utiliza de elementos da Scrum para cenários onde há apenas um desenvolvedor de *software*. Nessa técnica, as *sprints* possuem no máximo 1 semana. Além disso, alguns aspectos da Scrum, como a reunião diária e a existência de alguns papéis, são dispensados (PAGOTTO et al. 2016).

2.6 PLATAFORMAS E LINGUAGENS DE PROGRAMAÇÃO

Para atingir os objetivos descritos no presente projeto, algumas linguagens de programação são necessárias para o tratamento de dados do acelerômetro, criação do modelo de AM e desenvolvimento da aplicação para *smartphone*.

2.6.1 Android

O Android é um sistema operacional (SO) popular, que faz parte das operações da Google desde 2005, quando a empresa que idealizou a marca foi absorvida (MEIO BIT, 2018). Existem SO Android para *smartphone*, *tablet*, *smart tv*, dispositivos *wearables* e computadores. Até o presente momento, 24 versões do Android foram lançadas e o site oficial divulga que ainda existe uma expectativa de longa vida para a marca (ANDROID, 2021a).

Hoje é o sistema operacional de *smartphones* mais utilizado do mundo (STATCOUNTER, 2021), estando presente em mais de 24 mil modelos de *smartphones* e *tablets* das mais variadas fabricantes (ANDROID, 2021a).

Escolher o sistema operacional Android como plataforma-alvo no desenvolvimento de uma nova aplicação de *smartphone* é uma movimentação natural, dadas as impressionantes estatísticas que comprovam a popularidade do sistema.

2.6.2 Java

O Java é uma das linguagens de programação oficiais para o desenvolvimento de aplicações Android (ANDROID, 2021b). Ela foi lançada em 1995 pela empresa Sun Microsystems, que mais tarde seria adquirida pela Oracle (JAVA, 2021). Atualmente o Java é divulgado como a linguagem de programação moderna mais popular do mundo (ORACLE, 2021). Essa informação é reforçada segundo pesquisas do índice TIOBE (2021), que demonstram que desde o ano de 2001 o Java figura entre as 3 linguagens mais utilizadas.

Um dos aspectos que contribuiu para a popularização da linguagem Java é a capacidade de operar em diferentes SO. Isso é possível pois todo programa desenvolvido em Java é executado em uma máquina virtual conhecida por JVM. Esse computador lógico opera em um computador físico real, e é capaz de interpretar o

programa escrito em Java (MANZANO; AFFONSO COSTA, 2014). Portanto, uma premissa para a execução de um sistema desenvolvido em Java, é a existência de uma JVM, o que é o caso de todas as versões do SO Android.

2.6.3 Python

O Python é uma popular linguagem de programação que vem sendo desenvolvida e utilizada desde a década de 90. Conhecida por ser simples e intuitiva sem deixar de lado a robustez, o Python possui uma grande e engajada comunidade de desenvolvedores pelo mundo (BANIN, 2018).

Na área de desenvolvimento de soluções baseadas em inteligência artificial, o Python é amplamente utilizado. A razão para essa forte presença no mercado pode ser atribuída à popularidade e simplicidade da linguagem, e também pela existência de muitas bibliotecas que fornecem soluções para tratamento de dados e desenvolvimento de modelos de inteligência artificial.

2.7 BIBLIOTECAS

Linguagens de programação podem ter suas capacidades aumentadas com o uso de bibliotecas. Uma biblioteca pode ser definida como um conjunto de funções e estruturas de dados previamente desenvolvidas. O uso desse tipo de recurso facilita e acelera os processos de desenvolvimento de *software*.

2.7.1 Pandas

A pandas é uma biblioteca para Python que fornece uma maneira otimizada de importar, analisar e manipular dados de diversos tipos de fonte. Além das estruturas de dados, a biblioteca ainda facilita a aplicação de diversas operações de álgebra relacional nos dados importados (PANDAS, 2021).

2.7.2 TensorFlow

A biblioteca TensorFlow foi criada pela equipe de pesquisa de inteligência artificial da Google, tendo sua primeira versão lançada em 2015. Ela fornece uma

grande variedade de ferramentas para a criação e treinamento de modelos de AM (TENSORFLOW, 2021).

Há também uma versão para dispositivos móveis, chamada de TensorFlow Lite. Essa biblioteca permite que modelos de AM criados com a TensorFlow, sejam convertidos em um formato compatível com a TensorFlow Lite, fazendo com que os modelos já treinados possam realizar predições em um contexto Java, portanto, em dispositivos Android.

2.7.3 Keras

A Keras, por sua vez, é outra biblioteca de código aberto escrita em Python. Ela disponibiliza interfaces que simplificam a utilização de outras bibliotecas destinadas a tarefas de aprendizado de máquina, como por exemplo o TensorFlow. O foco do Keras é a produtividade, pois simplifica ainda mais a criação dos modelos de aprendizado de máquina utilizando o TensorFlow (KERAS, 2021).

2.8 AMBIENTE DE DESENVOLVIMENTO INTEGRADO

As IDE, do inglês *Integrated Development Environments*, ou ambientes de desenvolvimento integrados, são editores de códigos de programação que fornecem um ambiente organizado com ferramentas que facilitam o desenvolvimento de *software*, aumentando a produtividade do desenvolvedor.

Entre as vantagens, há a análise em tempo real para erros de sintaxe, destaque em palavras-chave para melhorar a legibilidade, sugestão de código e atalhos de teclado. Também facilitam a compilação do código, publicação do sistema, possibilitam a execução de testes, entre outras facilidades (REDHAT, 2021).

3 MATERIAIS E MÉTODOS

O desenvolvimento do iaQuedas se deu em duas fases bem definidas. Primeiramente, foi realizada a criação de um modelo de aprendizado de máquina, e em seguida foi realizado o desenvolvimento de uma aplicação Android para consumir esse modelo.

Durante a primeira fase, cumpriu-se o providenciamento de uma base de dados compatível com o problema proposto. Em seguida, foi realizada uma etapa de estudo e pré-processamento dos dados. Finalmente, foi elaborada a modelagem e treinamento de um modelo de aprendizado de máquina.

A segunda fase foi o momento de desenvolvimento do aplicativo Android. Iniciou-se com uma etapa de engenharia de software, onde houve o levantamento de requisitos e confecção de diagramas. Em seguida, foi realizado o desenvolvimento da aplicação, que por fim utilizou o modelo de AM treinado na primeira fase.

Os subcapítulos a seguir tratam dos detalhes que permearam essas duas etapas principais do desenvolvimento.

3.1 FONTE DE DADOS

Para atingir os objetivos deste trabalho utilizando técnicas de AM, foi necessário dispor de uma fonte de dados contendo leituras de sensores acelerômetros em situações de queda e atividades normais. Isto porque as técnicas de AM dependem desse conhecimento prévio para construírem seus modelos.

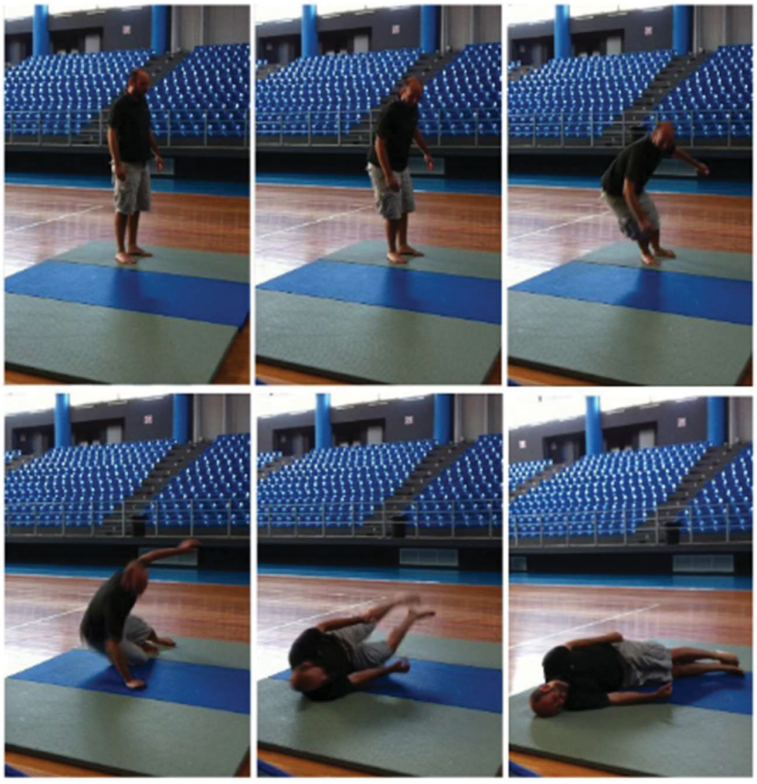
Conforme já mencionado anteriormente, outros trabalhos já exploraram a detecção de queda a partir dos dados de acelerômetro. Em alguns casos, a base de dados foi criada pelos próprios autores através de simulações de quedas. E em outros casos, utilizaram-se de bases disponibilizadas por outros trabalhos.

3.1.1 A base de dados MobiAct 2.0

Dentre as bases de dados disponíveis publicamente e via solicitação, neste trabalho decidiu-se por utilizar a MobiAct *2.0,* gentilmente disponibilizada pela Universidade Mediterrânea Helênica.

A MobiAct 2.0 é fruto de uma pesquisa realizada no Instituto Tecnológico Educacional de Creta, Grécia (CHATZAKI et al., 2017). Essa pesquisa reuniu 66 voluntários com idades entre 20 e 47 anos, os quais realizaram simulações de 4 tipos de quedas em um colchonete com 5cm de espessura, como demonstrado na FIGURA 8. Além das quedas, também foram realizadas simulações de atividades normais do dia a dia, como andar, sentar, pular e entre outras. Também foram coletados dados de algumas rotinas específicas que misturam atividades normais do dia a dia, como os cenários de ida ao trabalho, exercitar-se, voltar para casa e entre outros.

FIGURA 8 - SIMULAÇÃO DE QUEDA REALIZADA PARA A MOBIACT 2.0



FONTE: Vavoulas et al. (2013).

Os autores coordenaram as simulações, e para a obtenção dos dados de atividades normais, selecionaram movimentos corriqueiros e movimentos que possuíam semelhanças com eventos de quedas. Durante a captura dos movimentos, o voluntário teve a liberdade de posicionar o *smartphone* em qualquer bolso e direção,

essa decisão foi tomada para que as leituras representassem situações de uso do *smartphone* no dia a dia.

A MobiAct 2.0 é constituída por dados provenientes de leituras do módulo LSM330DLC de um *smartphone* Samsung Galaxy S3. Deste módulo eletrônico, foram coletados dados dos sensores acelerômetro, giroscópio e de orientação, sendo este último um sensor lógico, obtido ao cruzar dados dos sensores acelerômetro e magnetômetro.

Para a obtenção dos dados, houve o desenvolvimento de uma aplicação Android para capturar os dados brutos do sensor, o qual foi configurado para realizar o maior número de leituras possível. Também foi coletado um valor de *timestamp* para cada leitura.

A MobiAct 2.0 é baseada em trabalhos anteriores, como a MobiFall (VAVOULAS et al., 2013), MobiFall 2.0 (VAVOULAS et al., 2014) e MobiAct (VAVOULAS et al., 2016).

3.1.2 Características da MobiAct 2.0

A MobiAct 2.0 foi projetada para auxiliar o estudo de métodos para a detecção de certas atividades cotidianas e quedas, portanto, conta com uma variedade de categorias de amostras, as quais são demonstradas no QUADRO 1.

QUADRO 1 - CATEGORIAS DE AMOSTRAS DO MOBIACT 2.0

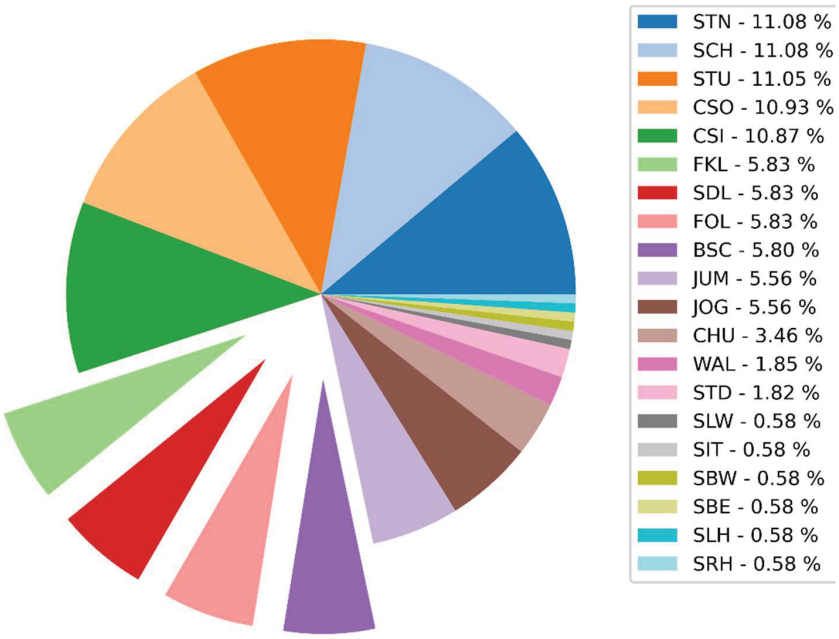
Cód.	**Descrição da atividade**
STD	Permanecendo em pé com movimentos sutis.
WAL	Andando normalmente.
JOG	Correndo.
JUM	Pulando.
STU	Subindo escada (10 degraus).
STN	Descendo escada.
SCH	Transição da atividade STD para SIT.
SIT	Permanecendo sentado em uma cadeira com movimentos sutis.
CHU	Transição da atividade SIT para STD.
CSI	Entrando no carro.
CSO	Saindo do carro.
LYI	Deitado (categoria derivada do período de inatividade pós-queda).
FOL	Queda para frente com auxílio das mãos para suavizar impacto.
FKL	Queda para frente com primeiro impacto nos joelhos.
BSC	Queda para trás em tentativa de sentar em uma cadeira.
SDL	Queda para um dos lados, com perna flexionada.
SLH	Rotina “saindo de casa” (STD, WAL, STN, WAL, STD, CSI, SIT, CSO, STD).
SBW	Rotina “permanecendo no trabalho” (STD, WAL, STU, WAL, WAL, STD, WAL, SCH, SIT).

SLW	Rotina “saindo do trabalho” (SIT, CHU, WAL, STD, WAL, STN, WAL, STD, CSI, SIT, CSO, STD).
SBE	Rotina “fazendo exercícios” (STD, WAL, JOG, WAL, STD, JUM, STD, WAL, STD).
SRH	Rotina “voltando para casa” (STD, CSI, SIT, CSO, STD, WAL, STU, WAL, STD, WAL, SCH).

FONTE: O autor (2022).

Ao analisar seu conteúdo, pode-se constatar que a base MobiAct 2.0 não é completamente balanceada, isto é, existem variações no número total de amostras de cada categoria, conforme demonstrado no GRÁFICO 1.

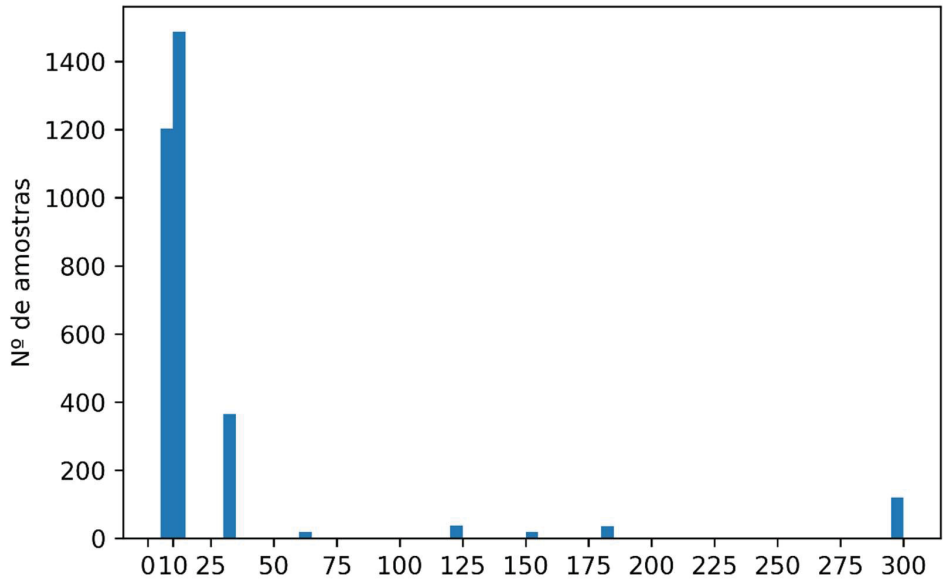
GRÁFICO 1 – COMPOSIÇÃO DO MOBIACT 2.0 POR CATEGORIA



FONTE: O autor (2022).

Outro aspecto do MobiAct 2.0 é a diferença na duração das amostras em cada categoria, variando de 5 minutos, em categorias como STD e WAL, para 6 segundos, como nas categorias SCH, CSI e CSO. O GRÁFICO 2 ilustra a distribuição das durações das amostras na base de dados.

GRÁFICO 2 – HISTOGRAMA DE DURAÇÃO DAS AMOSTRAS EM SEGUNDOS



FONTE: O autor (2022).

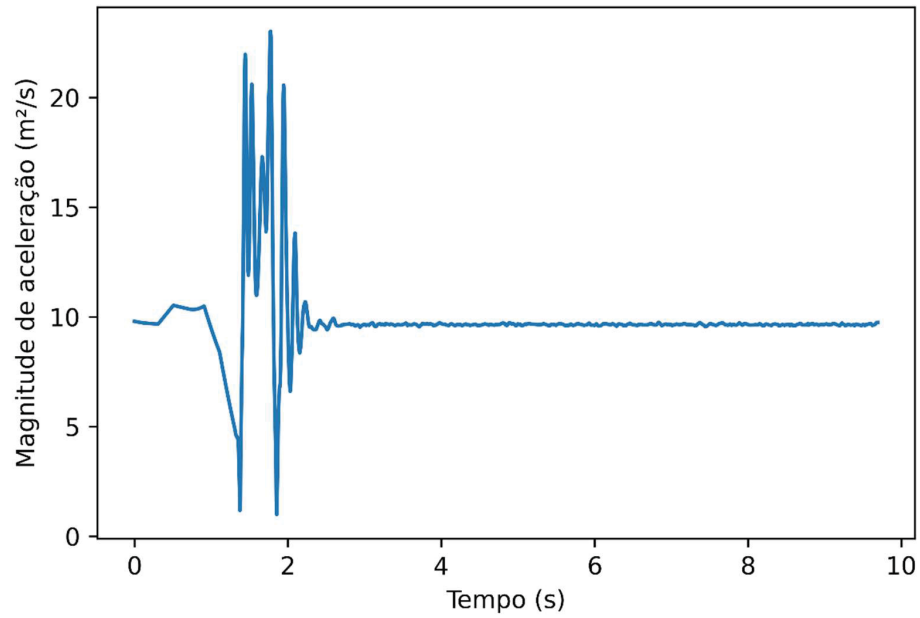
3.1.3 Pré-processamento

Dadas as características da base MobiAct 2.0, algumas técnicas de pré-processamento foram aplicadas para viabilizar a sua utilização.

A primeira etapa foi a remoção dos dados irrelevantes para este trabalho, como os arquivos contendo coletas do sensor de giroscópio e de orientação. Em seguida, calculou-se a magnitude de aceleração para cada leitura do acelerômetro. Outra transformação realizada ocorreu no campo de *timestamp*, que foi recalculado para representar seu tempo relativo na janela de amostragem. Ao fim, foram selecionadas apenas as duas colunas obtidas através da transformação dos dados: o tempo relativo e a magnitude de aceleração.

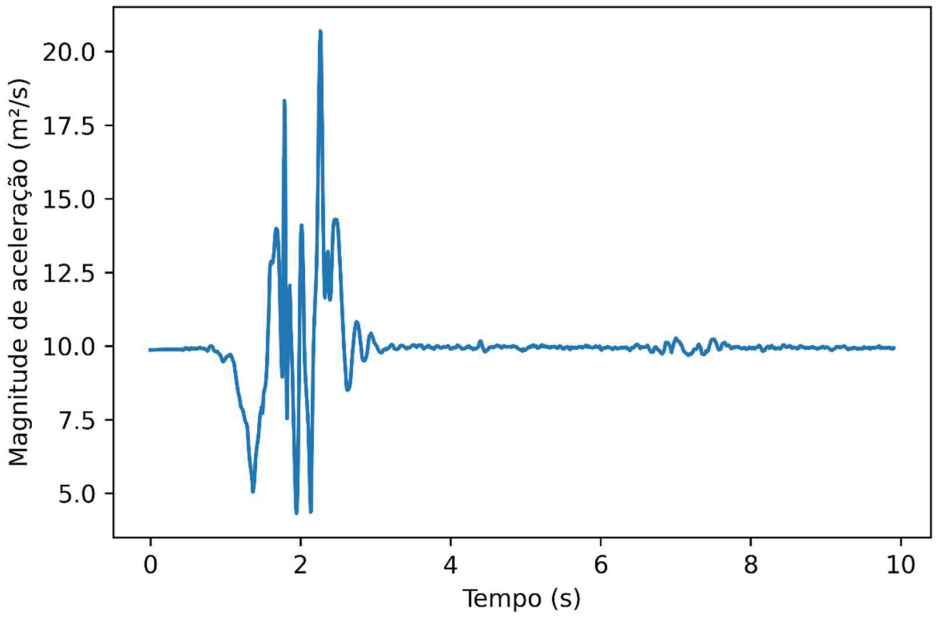
Após essa primeira intervenção na base de dados, foi possível observar a evolução da aceleração em cada categoria através de gráficos de linha. Os gráficos 3, 4, 5 e 6 representam um exemplo de série temporal de aceleração para cada tipo de queda, já os gráficos 7, 8, 9 e 10 são exemplos de atividades diárias.

GRÁFICO 3 – EXEMPLO DE AMOSTRA DE QUEDA “FOL”



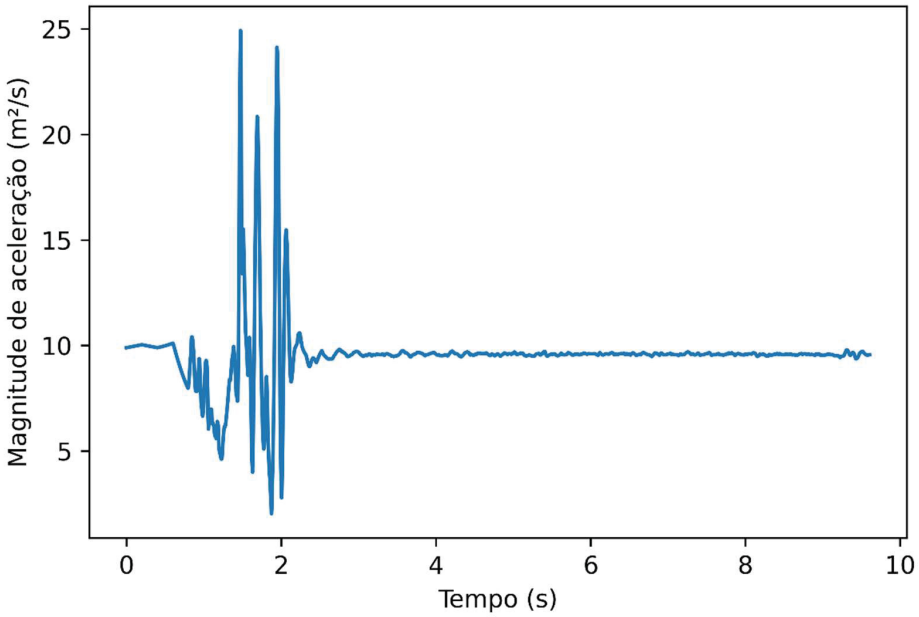
FONTE: O autor (2022).

GRÁFICO 4 – EXEMPLO DE AMOSTRA DE QUEDA “FKL”



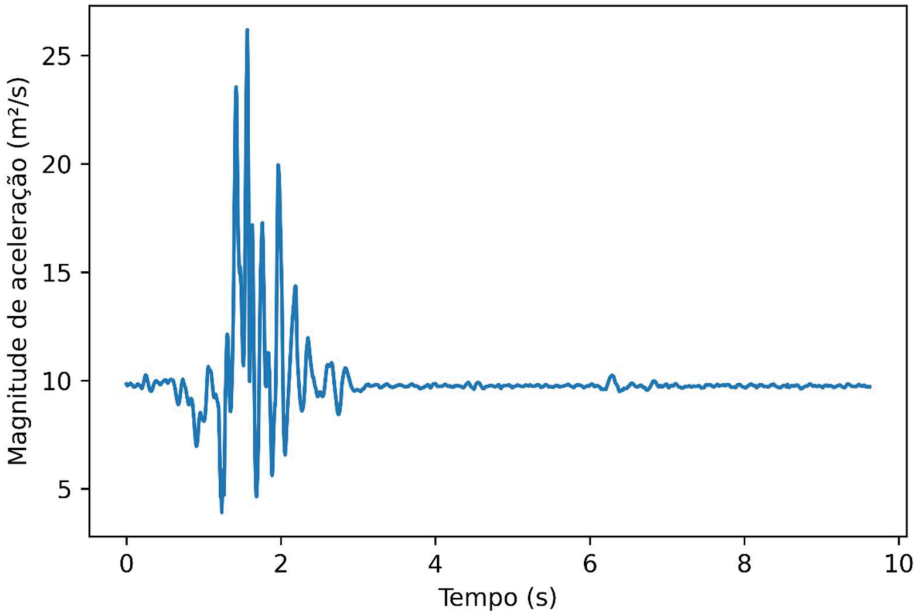
FONTE: O autor (2022).

GRÁFICO 5 – EXEMPLO DE AMOSTRA DE QUEDA “BSC”



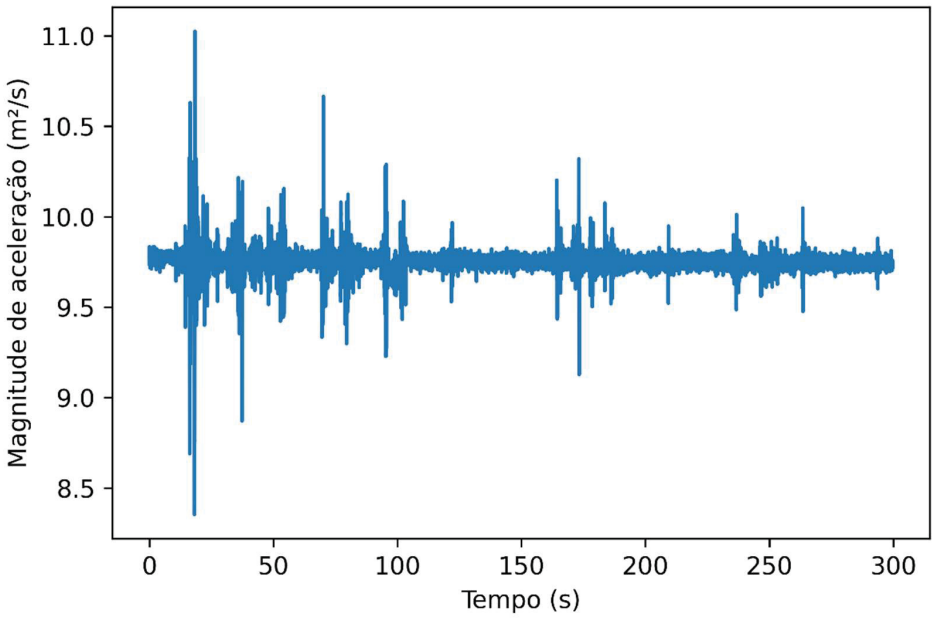
FONTE: O autor (2022).

GRÁFICO 6 – EXEMPLO DE AMOSTRA DE QUEDA “SDL”



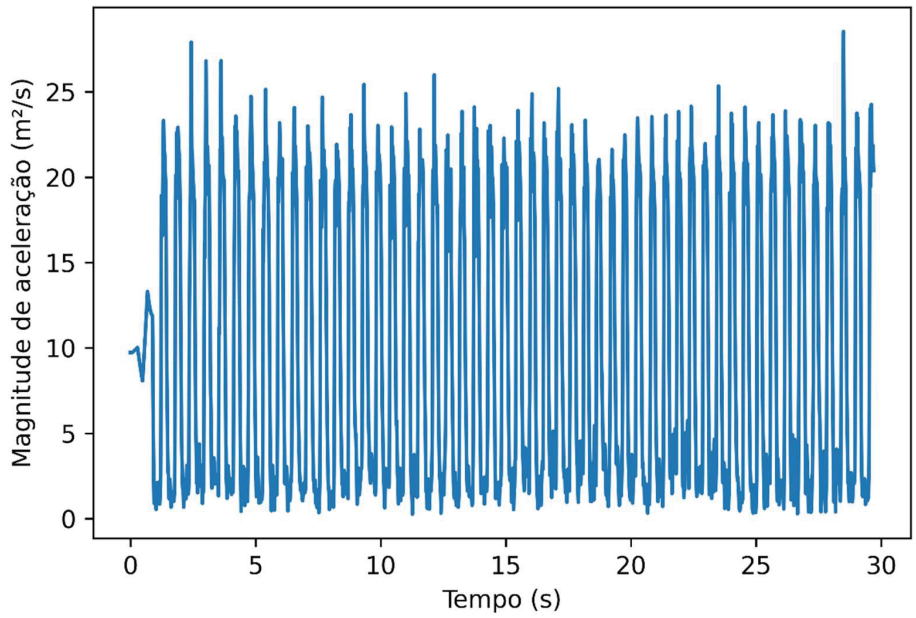
FONTE: O autor (2022).

GRÁFICO 7 – EXEMPLO DE AMOSTRA DE ATIVIDADE “STD” (EM PÉ)



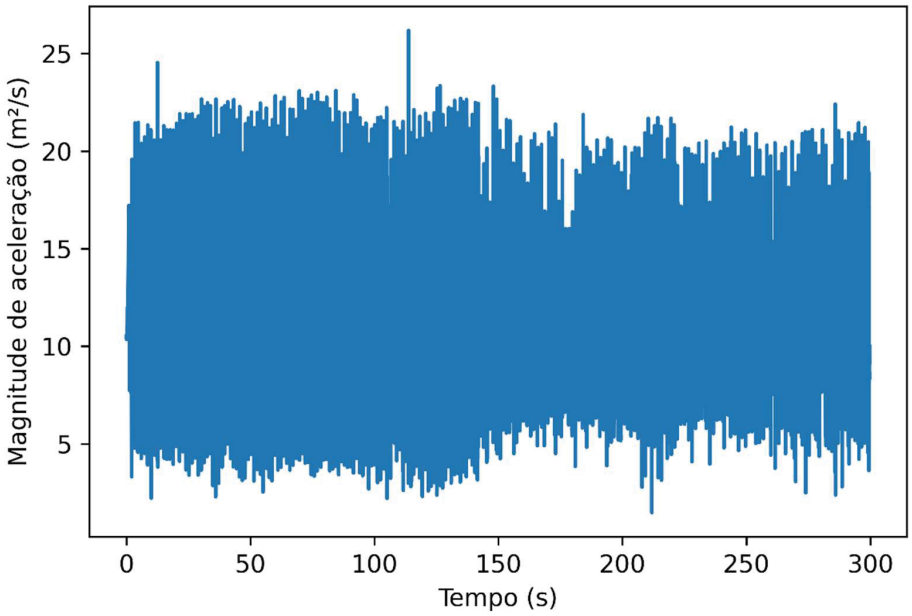
FONTE: O autor (2022).

GRÁFICO 8 – EXEMPLO DE AMOSTRA DE ATIVIDADE “JUM” (PULANDO)



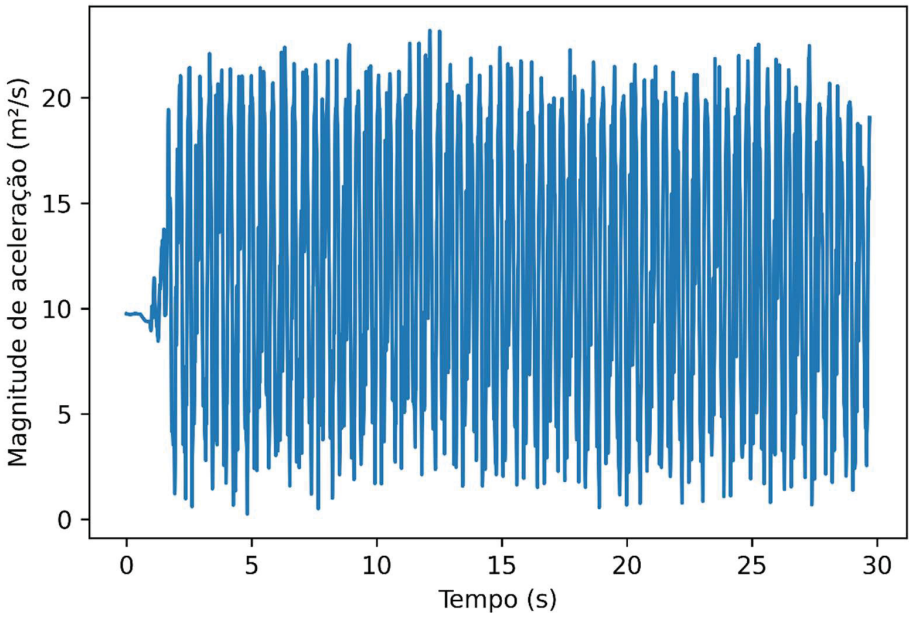
FONTE: O autor (2022).

GRÁFICO 9 – EXEMPLO DE AMOSTRA DE ATIVIDADE “WAL” (ANDANDO)



FONTE: O autor (2022).

GRÁFICO 10 – EXEMPLO DE AMOSTRA DE ATIVIDADE “JOG” (CORRENDO)



FONTE: O autor (2022).

A representação visual das amostras aponta grandes semelhanças entre as categorias de queda, além disso, evidencia que as características das atividades normais diferem intensamente das quedas. Também é possível notar que, nas amostras de longa duração, o movimento que dá nome à categoria se repete ao longo do tempo, possibilitando o fatiamento dessas séries temporais para fins de normalização e balanceamento dos dados.

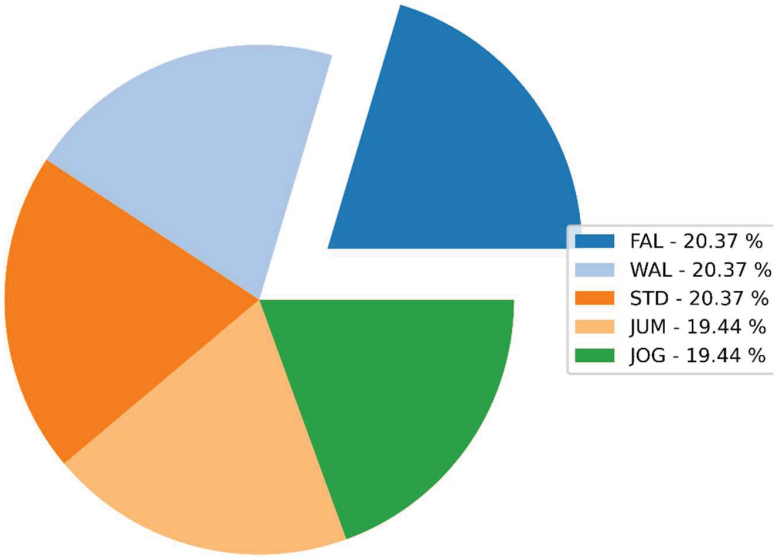
Essas conclusões levaram à decisão de criar uma categoria de queda genérica chamada FAL, a qual englobou os quatro tipos de queda. Também se decidiu pela diminuição do tempo das amostras de queda, desprezando os 4 segundos finais, que via de regra, eram períodos de inatividade. Esta última etapa foi realizada para que o posterior processo de confirmação de queda via rede neural tivesse maior celeridade. Também porque supõe-se que no mundo real, períodos de inatividade tão longos e uniformes podem não ocorrer após uma queda.

Assim que a categoria FAL foi criada, ela se tornou a maior em termos de quantidade de amostras, somando um total de 767 quedas. Para obter uma base de dados balanceada, se decidiu pelo abandono de algumas categorias que possuíam uma pequena quantidade de amostras e uma curta duração. Por outro lado, algumas categorias possuíam duração maior que 12 segundos, oferecendo, portanto, a possibilidade de fatiamento da série temporal até que a quantidade de amostras atingisse os patamares da categoria FAL. Este foi o caso das categorias STD e WAL, que após este processamento ficaram com a mesma quantidade de amostras. Já as categorias JOG e JUM alcançaram 732 amostras.

Por fim, a base de dados estava normalizada em relação à duração das séries e balanceada em relação ao número de amostras para cada categoria, como demonstra o GRÁFICO 11.

Também houve uma separação nessa base dados utilizando a técnica *holdout*, portanto, 70% dos dados foi reservado para ser utilizado durante o treinamento e 30% para o processo de validação.

GRÁFICO 11 – COMPOSIÇÃO DA BASE APÓS PRÉ-PROCESSAMENTO



FONTE: O autor (2022).

3.2 MODELO DE APRENDIZADO DE MÁQUINA

Para o modelo de confirmação de queda baseado em inteligência artificial, decidiu-se utilizar o algoritmo de redes neurais recorrentes do tipo LSTM da biblioteca Keras.

Apesar da configuração de taxa de amostragem do sensor funcionar satisfatoriamente na maioria das vezes, o sistema operacional Android não garante que o acelerômetro realize as leituras com alta precisão em relação à taxa de amostragem. Portanto, essa configuração possui um caráter maior de recomendação ao sensor do que um ajuste determinístico. Isso fez com que fosse necessária a adição de uma máscara da biblioteca Keras para padronizar o número de passos da série temporal antes de submeter os dados ao algoritmo LSTM. O valor escolhido para a máscara foi 1201, o qual se trata do número de passos da série mais longa após o pré-processamento da base de dados.

Após a padronização, a camada de entrada do modelo ficou pronta para receber dados de séries temporais de 1201 passos contendo dados de magnitude de aceleração.

A simplicidade do modelo foi priorizada, portanto a camada oculta foi criada com apenas 16 unidades LSTM, sem nenhum mecanismo adicional de redes neurais para o aprendizado.

Foram mantidas as funções de ativação originais das células LSTM, isto é, a função tangente hiperbólica foi utilizada para a camada de saída das células, enquanto funções sigmoides foram utilizadas nos processamentos internos para a atualização dos estados.

A camada de saída da rede LSTM foi criada com 5 neurônios artificiais, um para cada categoria de atividade treinada. A função de ativação *softmax* foi configurada para a camada de saída. Deste modo, cada neurônio gerou um valor numérico que representava a probabilidade de o conjunto de dados ser classificado com o atributo alvo do neurônio. A inferência do resultado de cada classificação foi realizada verificando qual era a categoria relacionada ao neurônio de saída com o maior valor de probabilidade.

3.2.1 Treinamento

O treinamento foi configurado para ocorrer em 100 épocas de 165 passos. Isto é, em cada época, foram apresentados lotes de 16 amostras antes de estimar o erro e reajustar os pesos. A função de erro designada para o treinamento foi a entropia cruzada, e os atributos alvo esperados foram transformados para o formato *one-hot encode*. O valor 0,001 foi definido como taxa de aprendizado.

Também foi criada uma função para armazenar o conjunto de pesos da época onde houvesse o menor erro durante o treinamento. Assim, pôde-se recuperar a melhor versão do modelo para a avaliação e utilização no aplicativo Android.

A FIGURA 9 demonstra o resumo da arquitetura do modelo gerado pela biblioteca Keras. A quantidade total de parâmetros e pesos ajustáveis foi de 1237, um número pequeno quando comparado a soluções modernas para problemas de AM.

FIGURA 9 - RESUMO DA ARQUITETURA DO MODELO DE AM

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
masking (Masking)            (None, 1201, 1)           0
_________________________________________________________________
lstm (LSTM)                  (None, 16)                1152
_________________________________________________________________
dense (Dense)                (None, 5)                 85
=================================================================
Total params: 1,237
Trainable params: 1,237
Non-trainable params: 0
```

FONTE: O autor (2022)

3.3 REQUISITOS DO APLICATIVO

Para o desenvolvimento deste trabalho, optou-se por fazer uma documentação de requisitos simples. Para cada requisito, foi atribuído um código, uma descrição e uma prioridade. Essa coleta inicial foi documentada no QUADRO 2.

QUADRO 2 - REQUISITOS DO APLICATIVO

Cód.	Descrição	Prioridade
R-001	O nome do aplicativo deve ser iaQuedas.	Alta
R-002	O logotipo deve representar uma pessoa caindo e um ponto de exclamação vermelho.	Baixa
R-003	O aplicativo deve possuir uma tela de configuração para informar o nome do usuário e o número de um contato de confiança.	Alta
R-004	Os campos de nome de usuário e número do contato de confiança devem ser obrigatórios.	Alta
R-005	O aplicativo deve solicitar automaticamente qualquer permissão especial necessária.	Alta
R-006	O aplicativo deve levar o usuário automaticamente para a tela de configuração, caso os dados obrigatórios não estejam preenchidos.	Média
R-007	O aplicativo não deve permitir ao usuário voltar para a tela inicial sem preencher os campos obrigatórios.	Média
R-008	O campo de configuração de nome deve ser de formato livre (letras, números e símbolos).	Alta
R-009	O campo do número do contato de confiança deve ser numérico.	Alta
R-010	A tela principal deve possuir instruções de funcionamento do aplicativo.	Baixa
R-011	A tela principal deve mostrar o logotipo e uma mensagem de boas-vindas.	Baixa
R-012	Os textos utilizados na tela inicial devem ser claros e sem detalhes técnicos.	Média
R-013	Emojis devem ser utilizados para complementar os textos.	Baixa
R-014	A tela principal deve possuir um botão para sair do aplicativo.	Baixa
R-015	A tela principal deve possuir um botão para interromper o monitoramento (quando operante).	Baixa

R-016	A tela principal deve possuir um botão para retomar o monitoramento (quando interrompido).	Baixa
R-017	A tela principal deve possuir uma navegação para a tela de configurações.	Alta
R-018	Quando as configurações obrigatórias estiverem preenchidas, o aplicativo deve monitorar os dados do acelerômetro em tempo real e aplicar testes dos valores limiares.	Alta
R-019	Quando o teste de limiar for positivo, o aplicativo deve aplicar o teste contra o modelo de AM.	Alta
R-020	O aplicativo deve emitir um som e vibração repetidamente em caso de detecção de queda pelo modelo de AM.	Média
R-021	O som reproduzido durante a detecção de quedas deve ser livre de direitos autorais.	Alta
R-022	Em caso de detecção de queda, o monitoramento dos dados de acelerômetro deve entrar em pausa.	Alta
R-023	Uma tela de ação deve aparecer no caso de detecção de queda.	Alta
R-024	A tela de ação deve possuir um texto que informe a suspeita de queda.	Alta
R-025	A tela de ação deve possuir um contador regressivo centralizado de 30 segundos.	Baixa
R-026	A tela de ação deve possuir um botão para cancelar a ação (indicação de alarme-falso).	Alta
R-027	O toque no botão de cancelar a ação deve fazer o aplicativo retornar à tela principal e retomar o monitoramento do acelerômetro.	Alta
R-028	A tela de ação deve possuir um botão para notificar a queda.	Alta
R-029	O toque no botão de notificar queda deve disparar o envio de uma SMS para o contato de confiança.	Alta
R-030	A SMS deve informar que houve um possível evento de queda.	Alta
R-031	A SMS deve informar o nome do usuário e o horário de ocorrência.	Alta
R-032	Quando possível, a SMS deve conter um *link* para o serviço Google Maps contendo a geolocalização aproximada da ocorrência.	Baixa
R-033	A SMS deve deixar claro que é uma notificação automática.	Alta
R-034	A SMS deve ser enviada utilizando o serviço de mensagem do celular do usuário.	Média
R-035	Caso não haja interação com a tela de ação em 30 segundos, a rotina de notificação de quedas deve ser disparada.	Alta
R-036	Após a notificação via SMS, o aplicativo deve navegar automaticamente para a tela inicial e o monitoramento do acelerômetro deve ser retomado.	Alta
R-037	O monitoramento do acelerômetro deve funcionar com a tela bloqueada.	Baixa
R-038	O monitoramento do acelerômetro deve funcionar com o aplicativo em segundo plano.	Baixa
R-039	O aplicativo deve ser simples e de fácil utilização.	Baixa
R-040	O aplicativo deve funcionar sem acesso à internet.	Média

FONTE: O autor (2022).

3.3.1 Confecção de diagramas

Alguns diagramas do padrão UML foram confeccionados para documentar e auxiliar no planejamento do iaQuedas.

Para complementar os requisitos e ilustrar com mais clareza as interações planejadas para o aplicativo, foi criado um diagrama de casos de uso. Ao todo foram mapeados 12 casos de uso e 3 atores, conforme demonstrado no APÊNDICE 1.

Também foi criado um diagrama de atividades para planejar a coleta e verificação dos dados do acelerômetro. Este diagrama foi adicionado ao APÊNDICE 2, e foi criado porque essa é, evidentemente, a atividade mais complexa do aplicativo.

Por fim, foi criado um diagrama de classes para orientar o desenvolvimento do aplicativo Android. Este diagrama pode ser consultado no APÊNDICE 3.

3.4 FERRAMENTAS PARA O DESENVOLVIMENTO

Uma série de artefatos físicos e de *software* foram necessários para atingir os objetivos deste trabalho, a qual é descrita nas seções abaixo.

3.4.1 Recursos físicos

Para o desenvolvimento da aplicação Android e do modelo de AM, foi utilizado um computador Lenovo Ideapad S145 com um processador Intel Core i7 de 3.9 GHz.

Já para os testes do aplicativo Android, foi utilizado um *smartphone* Xiaomi Redmi Note 8. Este aparelho possui um módulo de sensores ICM-42605, da marca TDK-Invensense.

3.4.2 Recursos de software

No computador utilizado para o desenvolvimento do aplicativo e do modelo de AM, utilizou-se o SO Ubuntu 20.04. Por sua vez, no *smartphone* utilizado para os testes do aplicativo, utilizou-se o sistema operacional Android 9 (Android Pie).

O Python versão 3.7.12 foi utilizado no desenvolvimento do modelo de aprendizado de máquina, primeiramente com a preparação dos dados da base MobiAct 2.0 com a biblioteca pandas versão 1.1.5. As bibliotecas utilizadas para modelar o algoritmo foram a Keras versão 2.4.3 e TensorFlow versão 2.7.0.

Para o desenvolvimento do aplicativo Android, foi utilizada a versão 8 do Java. Também foi utilizada a biblioteca Tensorflow Lite na versão 2.4.0. Esta biblioteca foi um ponto chave para o desenvolvimento da aplicação proposta neste trabalho, uma vez que a modelagem do algoritmo de AM foi realizada em Python, mas o modelo foi efetivamente utilizado no ambiente Android.

3.4.3 IDE

Para realizar o desenvolvimento da aplicação Android, foi utilizada a IDE Android Studio versão 2020.3.1. Já para o tratamento dos dados, desenvolvimento e treinamento do modelo de AM, foi utilizado a IDE Spyder versão 4.2.5.

4 RESULTADOS

Os resultados deste trabalho são apresentados a seguir. Inicialmente são documentadas as características da metodologia de desenvolvimento utilizada e as atividades realizadas durante o projeto. Também são apresentados os valores das métricas alcançadas nas predições do conjunto de teste e indicativos do desempenho obtido durante a etapa de aprendizado. Além disso, as capturas de tela do aplicativo Android com breves explicações do seu fluxo de funcionamento estão reportadas neste capítulo.

4.1 DESENVOLVIMENTO DO PROJETO

Neste trabalho, a metodologia Scrum Solo foi utilizada para a confecção da maior parte da escrita deste trabalho e desenvolvimento da aplicação.

O período de cada *sprint* foi definido para 5 dias, de segunda a sexta-feira, sendo que aos sábados e domingos, fazia-se a revisão e planejamento da próxima *sprint*, além da adição de novas tarefas ao *backlog* quando necessário.

Ao todo foram planejadas e executadas 18 *sprints*, entre os dias 05/07/2021 e 05/11/2021. A maioria das tarefas realizadas até a *sprint* 8 foram planejadas inicialmente, com o objetivo de escrita dos 3 primeiros capítulos deste trabalho. A conclusão da *sprint* 17 marcou o fim do desenvolvimento e dos testes manuais do iaQuedas. Por sua vez, a *sprint* 18 marcou a conclusão do desenvolvimento do projeto utilizando a metodologia Scrum.

A seguir, as *sprints* são retratadas com suas datas de ocorrência e a descrição resumida das tarefas realizadas durante o período.

4.1.1 Sprint 1

A *sprint* 1 ocorreu entre os dias 05/07 e 09/07. Nesta etapa inicial, foi realizada uma pesquisa de contextualização do problema das quedas. Também foram definidos os objetivos gerais e específicos deste trabalho.

4.1.2 Sprint 2

A *sprint* 2 ocorreu entre os dias 12/07 e 16/07. Durante esse período, foram escritas as seções de contextualização e justificativa do capítulo de introdução. Nessa etapa, os objetivos foram refinados e escritos.

4.1.3 Sprint 3

A *sprint* 3 foi realizada entre os dias 19/07 e 23/07. Nesses dias, foram pesquisados trabalhos relacionados, e, portanto, as técnicas existentes para a detecção de quedas. Também se pesquisou detalhes sobre o funcionamento de sensores acelerômetros.

4.1.4 Sprint 4

A *sprint* 4 aconteceu entre os dias 26/07 e 30/07. Foram nesses dias que se decidiu as técnicas que seriam utilizadas para a detecção de quedas. Neste período também foi iniciada a pesquisa sobre as ferramentas que seriam utilizadas.

4.1.5 Sprint 5

A *sprint* 5 ocorreu entre os dias 02/08 e 06/08. Neste período iniciou-se a escrita da fundamentação teórica, primeiramente sobre os sensores eletrônicos, em seguida sobre as técnicas para a detecção de queda. No último dia de *sprint* foi realizado um trabalho de revisão da introdução.

4.1.6 Sprint 6

A *sprint* 6 ocorreu entre os dias 09/08 e 13/08. Nesses dias foram incluídos textos na seção de fundamentação teórica, especificamente os que descrevem as linguagens de programação e bibliotecas que seriam utilizadas. Ao fim do período também foi realizado um trabalho de revisão da fundamentação teórica.

4.1.7 Sprint 7

A *sprint* 7 ocorreu entre os dias 16/08 e 20/08. Durante esta semana, iniciou-se a escrita da seção de materiais e métodos.

4.1.8 Sprint 8

A *sprint* 8 ocorreu entre os dias 23/08 e 27/08. Nesta semana realizou-se uma revisão geral deste documento, incluindo um trabalho de formatação.

4.1.9 Sprint 9

A *sprint* 9 ocorreu entre os dias 30/08 e 03/09. Durante essa semana foi realizada uma pesquisa sobre as fontes de dados disponíveis para a realização deste trabalho. Após a escolha da fonte, uma solicitação para a utilização do MobiAct 2.0 foi enviada para a Universidade Helênica Mediterrânea.

4.1.10 Sprint 10

A *sprint* 10 ocorreu entre os dias 06/09 e 10/09. Durante esses dias foram confeccionados o logotipo e o ícone do iaQuedas. Além disso foi realizada a escolha do alerta sonoro que seria utilizado na tela de ação.

4.1.11 Sprint 11

A *sprint* 11 ocorreu entre os dias 13/09 e 17/09. Nesse período, o ambiente de desenvolvimento foi preparado, com a instalação dos IDE Android Studio e Spyder. Também foi criado o projeto do aplicativo no IDE Android Studio e um repositório para o versionamento do código no site GitHub.

4.1.12 Sprint 12

A *sprint* 12 ocorreu entre os dias 20/09 e 24/09. Durante esses dias, foi realizada a coleta da assinatura do orientador em um acordo de uso para a obtenção da MobiAct 2.0. Em paralelo, foi iniciado o desenvolvimento dos requisitos que envolvem a tela de configuração e a solicitação de permissões especiais.

4.1.13 Sprint 13

A *sprint* 13 ocorreu entre os dias 27/09 e 01/10. Nesse período, foi desenvolvido o serviço de monitoramento do acelerômetro, incluindo a lógica para o armazenamento em memória dos valores de aceleração e a verificação dos limiares para detecção de queda. Também foi desenvolvida a tela principal, contendo o texto explicativo, os botões de saída, interrupção e retomada do monitoramento da aceleração.

4.1.14 Sprint 14

A *sprint* 14 ocorreu entre os dias 04/10 e 08/10. Durante essa sprint, a tela de ação foi criada, contendo todos os seus casos de uso para cancelamento e confirmação de queda. Também foi implementada a lógica para a construção e envio da SMS com a notificação de queda. Ao fim dessa *sprint*, havia uma versão funcional do aplicativo sem a verificação do evento no modelo de inteligência artificial.

4.1.15 Sprint 15

A *sprint* 15 ocorreu entre os dias 11/10 e 15/10. No início dessa *sprint*, a base de dados foi disponibilizada para *download*. Ao decorrer da semana, foram realizadas a exploração e a preparação da base de dados. A escrita das seções de características e pré-processamento da MobiAct 2.0 ocorreu durante esse mesmo período, em paralelo com o trabalho realizado com os dados.

4.1.16 Sprint 16

A *sprint* 16 ocorreu entre os dias 18/10 e 22/10. Durante essa semana foram realizadas a modelagem e treinamento do algoritmo de aprendizado de máquina. Nesse período também se realizou a avaliação do desempenho do algoritmo com o conjunto de dados de teste.

4.1.17 Sprint 17

A *sprint* 17 ocorreu entre os dias 25/10 e 29/10. Durante esse período foi feita a conversão do algoritmo para o formato compatível com a biblioteca TensorFlow Lite. Também foi realizada a integração do algoritmo de inteligência artificial ao serviço de monitoramento do acelerômetro. Após essa *sprint*, pôde-se iniciar os testes do aplicativo com todas as funcionalidades descritas na análise de requisitos.

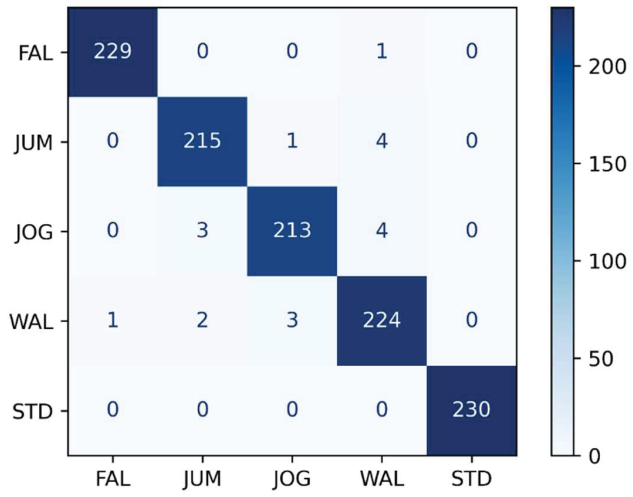
4.1.18 Sprint 18

A *sprint* 18 ocorreu entre os dias 01/11 e 05/11. Essa foi a última *sprint* do projeto, e durante esse período foram realizados os testes manuais do aplicativo. Muitas correções foram realizadas, conforme encontravam-se falhas e erros no aplicativo. Por fim, foi realizada a escrita da seção de resultados.

4.2 MÉTRICAS DO MELHOR MODELO

O modelo selecionado após a etapa de treinamento alcançou um ótimo desempenho na tarefa de classificação para o conjunto de testes. A taxa de erros foi de apenas 0,0168 enquanto a acurácia foi de 0,9831. O GRÁFICO 12 demonstra a matriz de confusão obtida para essas predições.

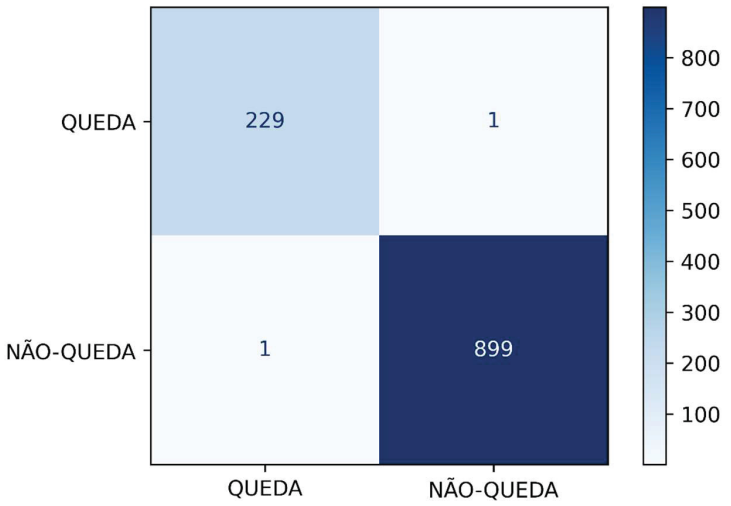
GRÁFICO 12 - MATRIZ DE CONFUSÃO PARA AS 5 CATEGORIAS



FONTE: O autor (2022).

Uma vez que o objetivo final do modelo foi a detecção de quedas, a avaliação do modelo pôde ser reinterpretada como um problema de duas classes: queda e não-queda. Ao considerar essa abordagem, os resultados melhoram. O GRÁFICO 13 demonstra a matriz de confusão reinterpretada para um problema de duas categorias, enquanto a TABELA 2 destaca as principais métricas alcançadas.

GRÁFICO 13 – MATRIZ DE CONFUSÃO PARA 2 CATEGORIAS



FONTE: O autor (2022).

TABELA 2 - RESUMO DAS MÉTRICAS ALCANÇADAS

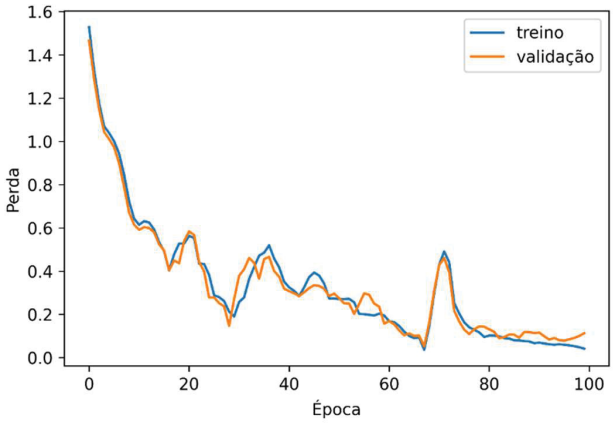
Métrica	Valor
Acurácia	0,9982
Taxa de erros	0,0017
Sensibilidade	0,9956
Especificidade	0,9988

FONTE: O autor (2022).

4.3 DESEMPENHO DO APRENDIZADO

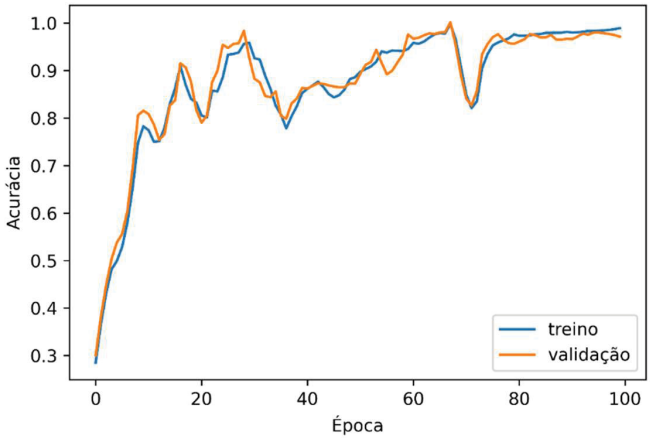
O GRÁFICO 14 e o GRÁFICO 15 demonstram a evolução do aprendizado do modelo, respectivamente através das curvas de erro e de acurácia. Percebe-se que o modelo já alcançava resultados de acurácia acima de 90% antes da vigésima época. Nota-se também que há certa instabilidade no aprendizado antes da octogésima época.

GRÁFICO 14 - EVOLUÇÃO DO VALOR DE ERRO



FONTE: O autor (2022).

GRÁFICO 15 - EVOLUÇÃO DA ACURÁCIA



FONTE: O autor (2022).

Os pesos selecionados foram os obtidos ao fim da nonagésima quinta época, a qual alcançou um valor de erro de 0,06747 no conjunto de testes.

4.4 APLICATIVO ANDROID

4.4.1 Desempenho

O desempenho do aplicativo para a detecção de quedas, ao decorrer de simulações, se demonstrou satisfatório. Porém, casos de falso-positivo durante movimentos corriqueiros foram mais comuns do que o esperado, especialmente durante a utilização do aplicativo no interior de automóveis em movimento. De todo modo, a tela de confirmação de queda sempre permitiu que houvesse o cancelamento da notificação em tempo hábil. Na maioria das vezes, as notificações geradas via SMS chegavam em menos de 1 minuto. Porém, devido a instabilidades nos serviços de telefonia, por vezes, houveram atrasos no recebimento das mensagens.

4.4.2 Usabilidade

O pequeno número de telas e botões desenvolvidos para o iaQuedas, garantiu que o aplicativo fosse simples em termos de usabilidade. Foram desenvolvidos mecanismos para assegurar que o monitoramento iniciasse somente após o concedimento das permissões especiais e da configuração do nome de usuário e contato de confiança. Além disso, um texto explicativo foi adicionado à tela principal, para facilitar o entendimento das funcionalidades do iaQuedas.

4.4.3 Identidade visual e telas

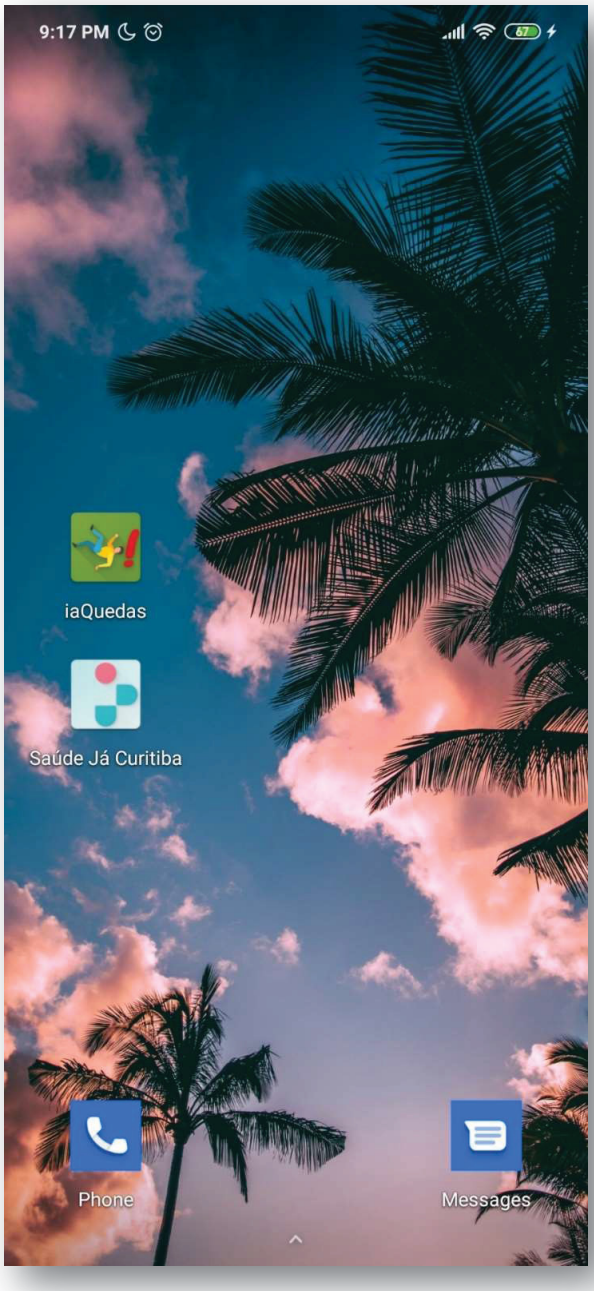
As telas foram criadas com foco no minimalismo, e seguiram o estilo Material Design. As cores principais foram o branco e verde. A FIGURA 10 demonstra o logotipo criado a partir da junção de *cliparts* livres de direitos autorais. Variações desse logotipo foram utilizadas na tela inicial e no ícone de inicialização.

FIGURA 10 - LOGOTIPO DO IAQUEDAS

FONTE: O autor (2022).

A FIGURA 11 demonstra a visualização do ícone do iaQuedas na tela inicial de um sistema Android.

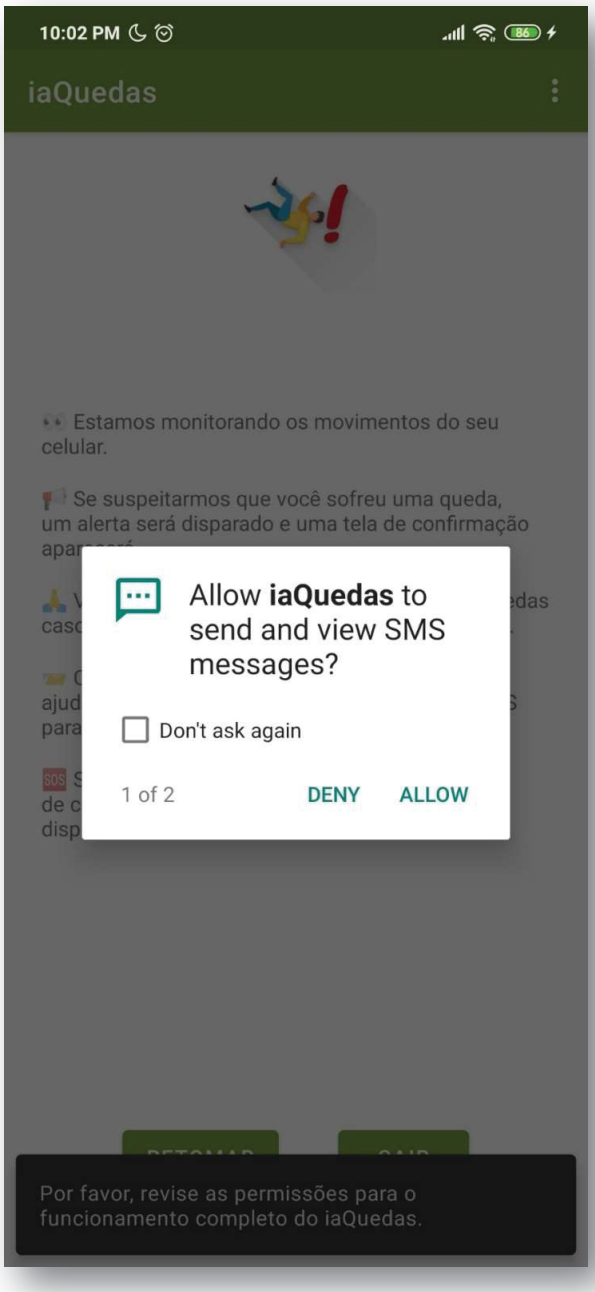
FIGURA 11 - ÍCONE DO IAQUEDAS NA TELA INICIAL DO ANDROID



FONTE: O autor (2022).

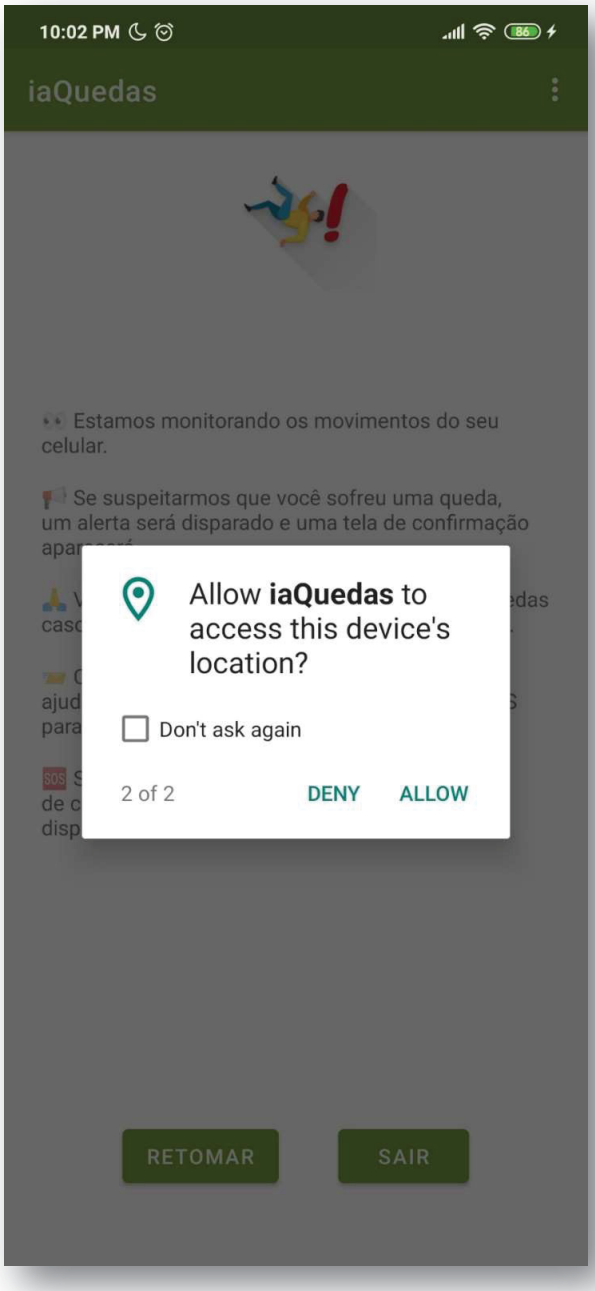
Já na FIGURA 12 e FIGURA 13 são apresentadas as solicitações de permissões especiais que ocorrem durante o primeiro acesso ao aplicativo.

FIGURA 12 - SOLICITAÇÃO DE PERMISSÃO PARA ENVIO DE SMS



FONTE: O autor (2022).

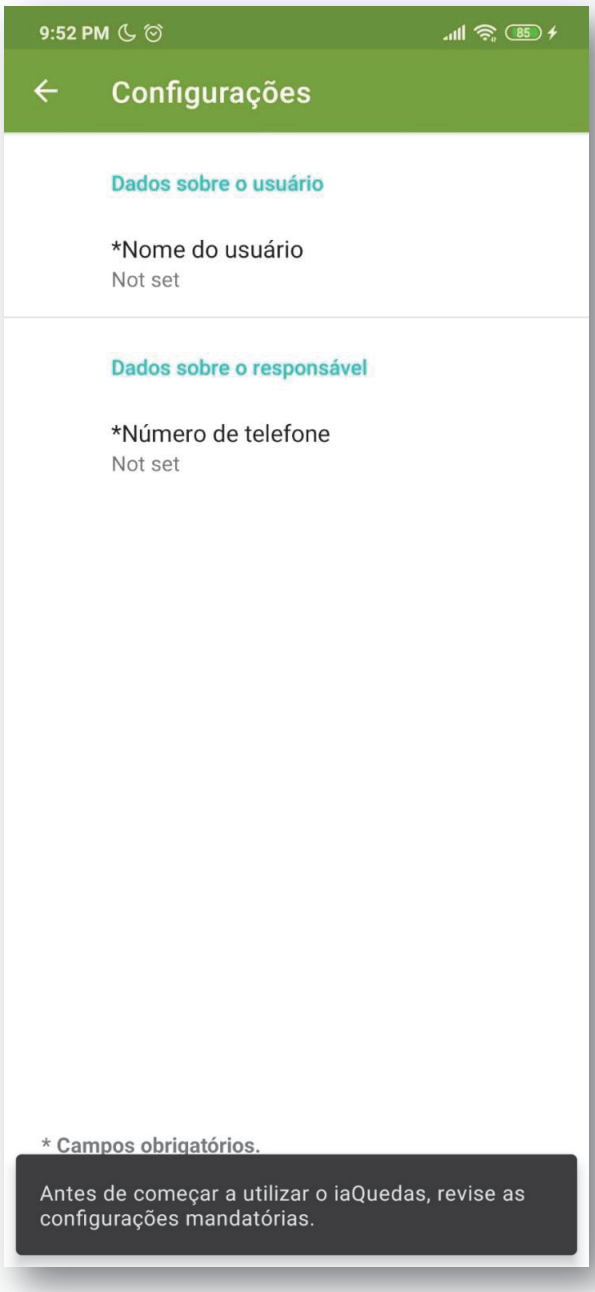
FIGURA 13 - SOLICITAÇÃO DE PERMISSÃO PARA ACESSO À LOCALIZAÇÃO



FONTE: O autor (2022).

Posteriormente, o preenchimento das configurações mandatórias é solicitado na tela de configurações, conforme demonstra a FIGURA 14.

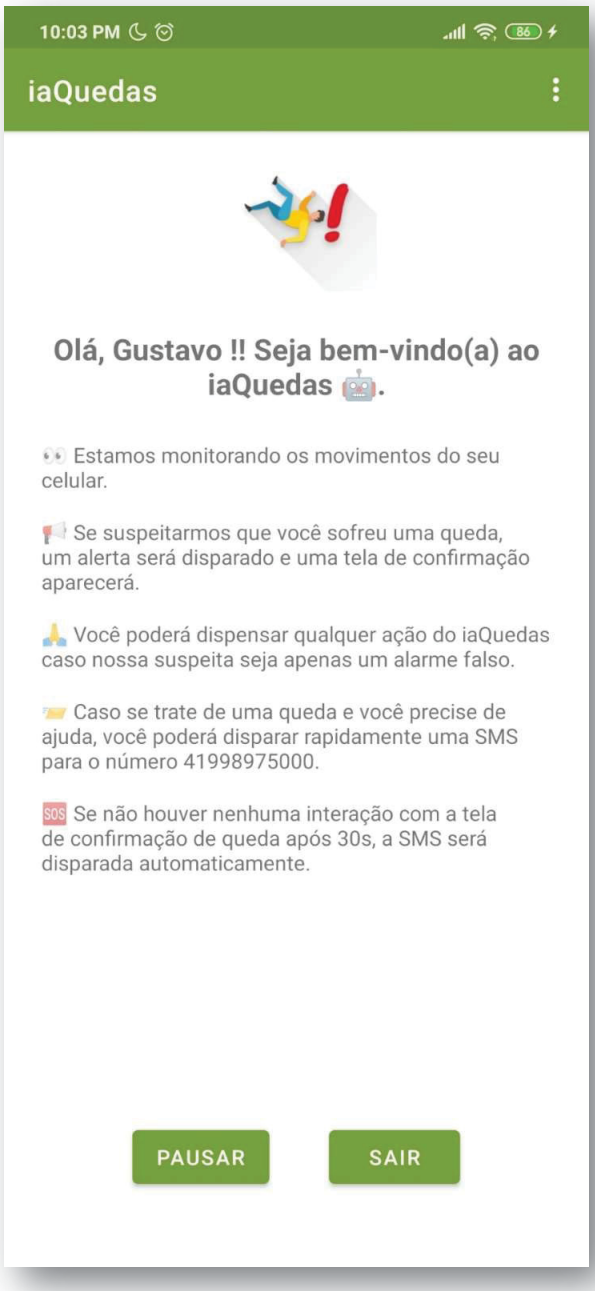
FIGURA 14 - TELA DE CONFIGURAÇÕES



FONTE: O autor (2022).

Após a preparação que ocorre nas telas anteriores, o aplicativo carrega a tela principal, a qual contém informações sobre o funcionamento do iaQuedas. A tela principal é demonstrada na FIGURA 15.

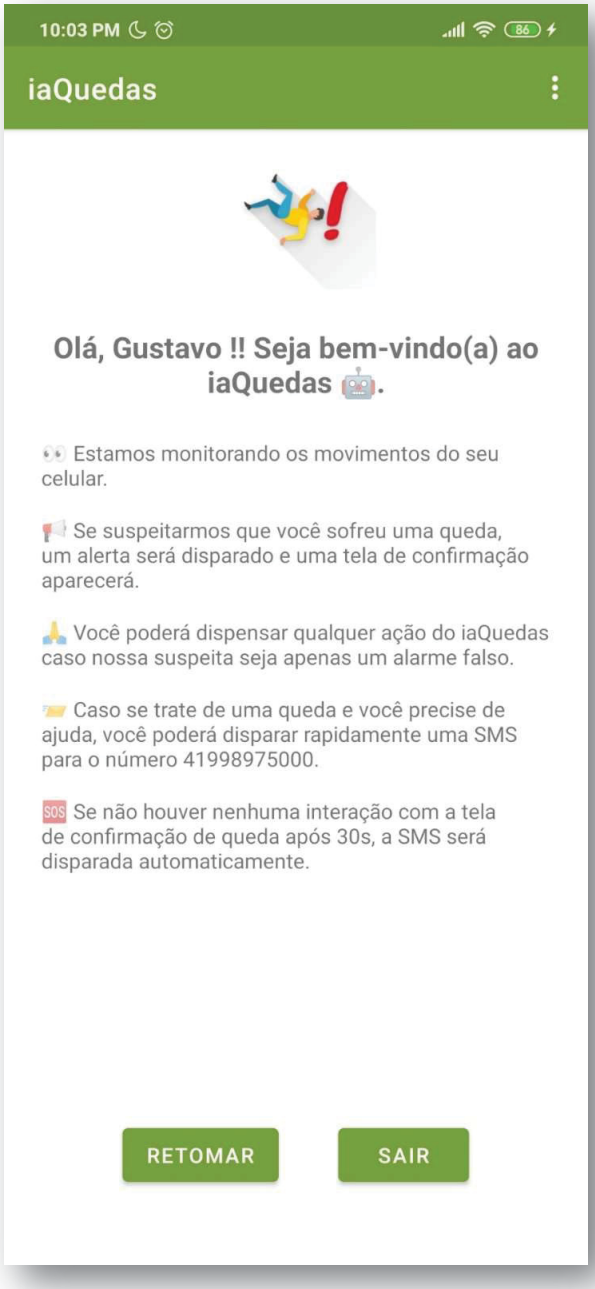
FIGURA 15 - TELA PRINCIPAL



FONTE: O autor (2022).

Na tela principal também é possível interromper o monitoramento, a FIGURA 16 exibe a tela principal neste estado.

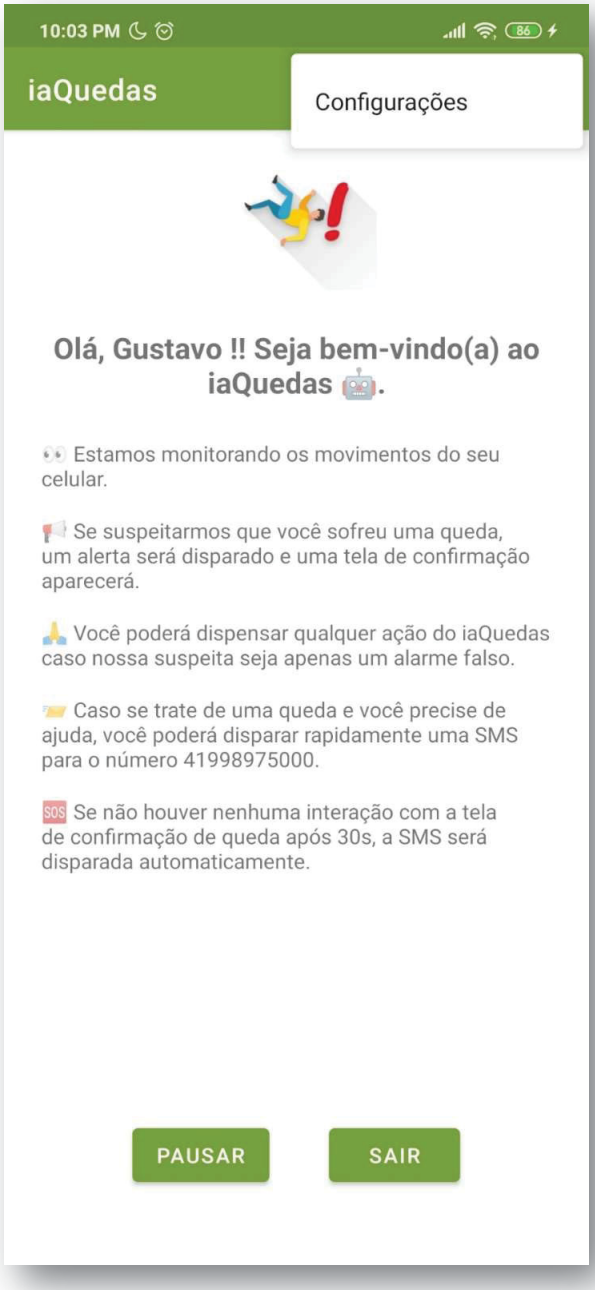
FIGURA 16 - TELA PRINCIPAL COM MONITORAMENTO INTERROMPIDO



FONTE: O autor (2022).

Outra possibilidade é apresentada na FIGURA 17, onde há a opção para navegar até a tela de configurações. Essa opção aparece ao tocar no ícone de três pontos verticais, visível na FIGURA 15 e FIGURA 16.

FIGURA 17 - TELA INICIAL COM NAVEGAÇÃO PARA AS CONFIGURAÇÕES



FONTE: O autor (2022).

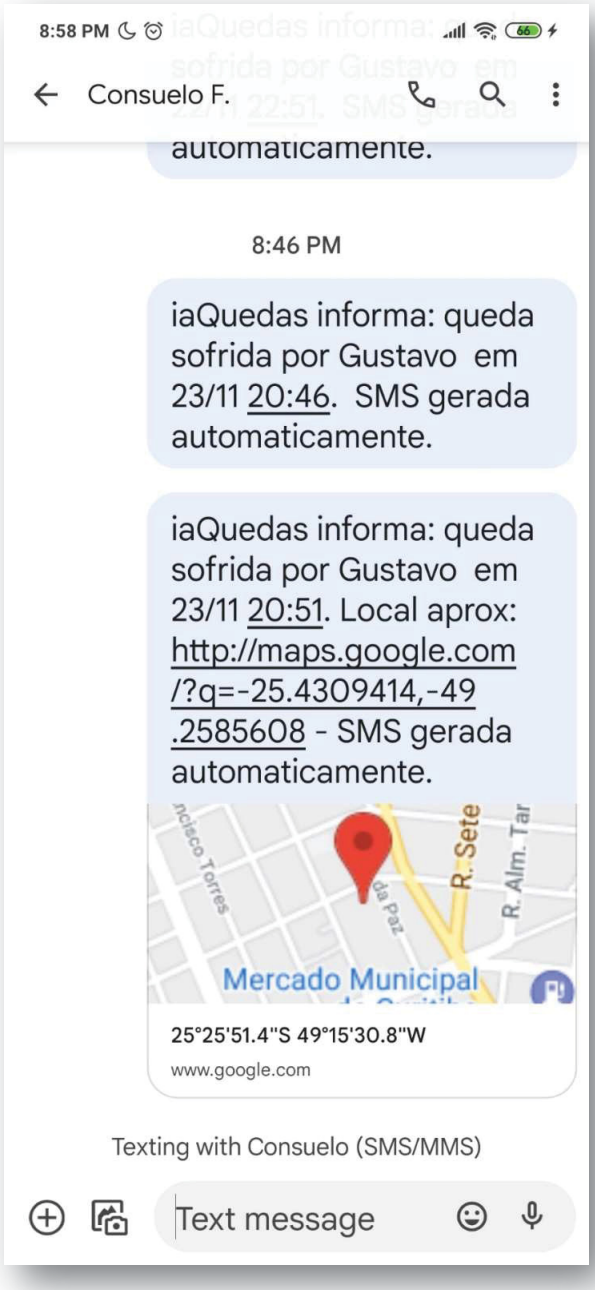
O aplicativo, ao detectar suspeições de queda via *threshold* e modelo de AM, carrega a tela de ação para quedas, demonstrada na FIGURA 18.

FIGURA 18 - TELA DE AÇÃO PARA QUEDA

FONTE: O autor (2022).

Quando o desfecho da tela de ação é a notificação de queda, uma mensagem SMS é enviada ao contato de confiança. Exemplos de mensagens de notificação de queda são demonstradas na FIGURA 19, e contém a data, horário, nome do usuário e possivelmente, sua localização.

FIGURA 19 - NOTIFICAÇÕES DE SMS COM E SEM LOCALIZAÇÃO



FONTE: O autor (2022).

5 CONSIDERACOES FINAIS

Assim como demonstrado nos estudos relacionados, o presente trabalho evidencia a praticabilidade existente na tarefa de detecção de quedas através do monitoramento e verificação dos dados de acelerômetros. Se abordada através da metodologia adequada, essa tarefa pode ser cumprida com boa qualidade em termos de acurácia, precisão e sensibilidade. Portanto, soluções que envolvem detecção de atividades através da aceleração são promissoras, principalmente ao considerar os avanços científicos que ocorrem diariamente na área da IA.

Uma vez que a problemática das quedas desassistidas é algo que ainda ameaça a todos, principalmente os mais velhos, pode-se afirmar que iniciativas como esta são importantes para promover melhorias de qualidade de vida. Um pequeno aumento nas chances de atendimento rápido para vítimas de acidentes ilustra o tipo de contribuição que tecnologias baseadas em IA são capazes de promover para a sociedade.

Um aspecto especialmente importante deste trabalho, foi a decisão de construir uma solução monolítica no ambiente Android, permitindo que o iaQuedas não dependesse de servidores ou acesso à internet para seu funcionamento. Isso garante a redução dos custos de publicação e manutenção da solução, já que não há a necessidade de uma infraestrutura de servidores de alta disponibilidade. A consequência final dessa decisão de arquitetura, é a viabilidade para a distribuição gratuita do aplicativo.

5.1 Pontos de melhoria

Ao decorrer das etapas de desenvolvimento e teste do iaQuedas, notaram-se alguns aspectos problemáticos. O principal é a fragilidade das notificações de quedas baseadas apenas em mensagens SMS. Para que isso funcione apropriadamente, conta-se com três premissas:

1. Os *smartphones* possuem número de telefone ativo.
2. O número do usuário é capaz de enviar SMS.
3. A rede de telefonia está disponível e operante no local da queda e na localidade do contato de confiança.

É evidente que problemas de rede e infraestrutura tendem a atrapalhar o funcionamento de qualquer solução baseada em *software*, contudo, uma vez que não se atenda qualquer uma das três premissas acima, o iaQuedas perde sua funcionalidade mais importante.

Outro problema, mas de menor gravidade, é o disparo da tela de ação em decorrência de falsos-positivos. Quando frequentes, esses casos atrapalham a experiência do usuário. Esse tipo de problema pode ser atribuído à diferença dos dados utilizados para o treinamento *versus* os dados do mundo real.

Em estado de funcionamento, incontáveis movimentos não-catalogados pelas pesquisas da MobiAct 2.0 podem disparar a verificação via IA. Nessas situações, as inferências realizadas pelo modelo de IA correm o risco de apresentar resultados incorretos, afinal, não houve treinamento para aqueles padrões de movimento. Além disso, cada pessoa possui sua maneira de se movimentar durante suas atividades diárias. As simulações realizadas durante a construção da MobiAct 2.0 representam uma boa generalização, mas que se limita à amostra de sujeitos pesquisados.

5.2 Recomendações para trabalhos futuros

Em um eventual prosseguimento deste trabalho, a principal prioridade deve ser o saneamento dos dois problemas apresentados na seção anterior.

Para aumentar a robustez na notificação de quedas, recomenda-se a adição de métodos de contingência, como o disparo de ligações com mensagens de voz geradas automaticamente, envio de e-mails, SMS, mensagens de WhatsApp e outros tipos de mensagens automáticas através de serviços da internet. Também é evidente a necessidade de possibilitar a inclusão de mais de um contato de confiança, para que as chances de entrega da notificação aumentem.

Outra recomendação é a utilização do aplicativo iaQuedas para a realização de novas pesquisas, onde se investiguem os movimentos não-catalogados que geram casos de falsos-positivos. Isso poderia orientar a criação de novas bases de dados com enfoque na tarefa de detecção de quedas. Para a criação dessas novas bases de dados, recomenda-se a mobilização de um grupo maior de voluntários e com maior amplitude etária, incluindo, se possível, idosos. A utilização de diferentes *smartphones* com sensores de diferentes marcas e modelos também poderia contribuir no enriquecimento da qualidade dos novos dados.

Após a resolução dos problemas mais relevantes, trabalhos futuros poderiam explorar aspectos para o aumento do nível de personalização da detecção de queda. Uma possibilidade é a calibragem do algoritmo de *threshold* com base nos atributos físicos do usuário. Outra possível funcionalidade é o retreinamento do modelo de IA com base em capturas que foram detectadas como possíveis eventos de queda. Para casos de falso-positivo, o aplicativo poderia apresentar uma tela adicional, questionando o usuário sobre como ele classificaria a atividade que gerou tal evento. Essa captura seria enviada para um servidor, onde ocorreria o retreinamento do modelo, que ao ser concluído, poderia ser disponibilizado para o usuário em formato de atualização do aplicativo. Ao passar do tempo, espera-se que tal funcionalidade melhore a qualidade na detecção de queda, uma vez que o modelo estaria treinado para reconhecer os padrões de movimento do usuário.

Além de melhorias relacionadas às técnicas baseadas na aceleração, a detecção de quedas poderia se beneficiar com a combinação de técnicas que utilizam outros sensores presentes no *smartphone*, como capturas de microfone, câmera, giroscópio, e entre outros.

REFERÊNCIAS

ABBATE, S. *et al*. A smartphone-based fall detection system. **Pervasive and Mobile Computing**, v. 8, n. 6, p. 883-899, dez. 2012. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S1574119212000983. Acesso em: 1 jul. 2021. https://doi.org/10.1016/j.pmcj.2012.08.003.

ALMEIDA, S. T. *et al.* Análise de fatores extrínsecos e intrínsecos que predispõem a quedas em idosos. **Revista da Associação Médica Brasileira**, v. 58, n. 4, p. 427-433, ago. 2012. Disponível em: https://www.scielo.br/j/ramb/a/XvWsksqJrStvztGSmMV8TGf. Acesso em: 1 jul. 2021. https://doi.org/10.1590/S0104-42302012000400012.

ANDERSON, E. The Species Problem in Iris. Annals of the Missouri Botanical Garden. **Annals of the Missouri Botanical Garden**. v. 23, p. 457-509, 1936. Disponível em: http://biostor.org/reference/11559. Acesso em 11 nov. 2021. https://doi.org/10.2307/2394164

ANDROID. Facts. 2021a. Disponível em: **https://www.android.com/everyone/facts/**. Acesso em 24 ago. 2021.

ANDROID. Android developers. Guide. Application Fundamentals. 2021b. Disponível em: https://developer.android.com/guide/components/fundamentals?hl=en. Acesso em: 25 ago. 2021.

ANDROID. Android Developers. Docs. Guides. Sensors Overview, 2019. Disponível em: https://developer.android.com/guide/topics/sensors/sensors_overview. Acesso em: 11 jul. 2021.

BANIN, S. L. **Python 3** - Conceitos e Aplicações - Uma abordagem didática. São Paulo: Editora Saraiva, 2018.

BOURKE, A. K. et al. An Optimum Accelerometer Configuration and Simple Algorithm for Accurately Detecting falls. In: IASTED International Conference on BIOMEDICAL ENGINEERING, 4. 2006, Innsbruck. **Anais**... Innsbruck: BioMED, 2006. p. 156-160.

BOURKE, A. K. et al. Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm. **Gait & Posture**, v. 26, n. 2, p. 194-199, jul-2007. Disponível em: https://www.sciencedirect.com/science/article/pii/S0966636206001895. Acesso em: 12 jul. 2021. https://doi.org/10.1016/j.gaitpost.2006.09.012

BOURKE, A. K; LYONS, G. M. A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. **Medical Engineering & Physics**, v. 30, n. 1, p. 84-90, jan-2008. Disponível em: https://www.sciencedirect.com/science/article/pii/S1350453306002657. Acesso em: 12 jul. 2021. https://doi.org/10.1016/j.medengphy.2006.12.001

BUKSMAN, S. *et al*. Quedas em Idosos: Prevenção. **Associação Médica Brasileira e Conselho Federal de Medicina**, 2008. Disponível em: http://www.projetodiretrizes.org.br/projeto_diretrizes/082.pdf. Acesso em: 28 jun. 2021.

FACELI, K. et al. **Inteligência Artificial** - Uma Abordagem de Aprendizado de Máquina. Rio de Janeiro: LTC, 2021.

CHATZAKI, C. et al. Human Daily Activity and Fall Recognition Using a Smartphone's Acceleration Sensor. **Communications in Computer and Information Science**. v. 736, p. 100-118. jul. 2017. www.doi.org/10.1007/978-3-319-62704-5_7.

CHEFFENA, M. Fall Detection Using Smartphone Audio Features. IEEE Journal of Biomedical and Health Informatics, v. 20, n. 4, p. 1073-1080, jul. 2016. Disponível em: https://ieeexplore.ieee.org/abstract/document/7093113/. Acesso em: 16 jul. 2021. https://doi.org/10.1109/JBHI.2015.2425932.

COHN, M. **Desenvolvimento de Software com Scrum** – Aplicando métodos ágeis com sucesso. Porto Alegre: Bookman, 2011.

COPPIN, B. **Inteligência Artificial**. Rio de Janeiro: LTC, 2010.

DAI, J. et al. PerFallD: A pervasive fall detection system using mobile phones. In: PerFallD: A pervasive fall detection system using mobile phones, 8., 2010. Mannheim, Alemanha. **Anais**... Mannheim, 2010. p. 292-297.

DESLANDES, S. F. *et al.* Caracterização diagnóstica dos serviços que atendem vítimas de acidentes e violências em cinco capitais brasileiras. **Ciência e saúde coletiva**, v. 11, p. 1279-1290, 2006. Disponível em: https://www.scielo.br/j/csc/a/rPTMGjCbnkDbZ7ykDTDxP4M. Acesso em 7 jul. 2021. https://doi.org/10.1590/S1413-81232006000500017.

EVCI, E. D. *et al*. Home Accidents in the Elderly in Turkey. **Tohoku Journal of Experimental Medicine**, v. 209, n. 4, p. 291-301, 2006. Disponível em: https://www.jstage.jst.go.jp/article/tjem/209/4/209_4_291/_article/-char/ja. Acesso em: 4 jul. 2021.

FABRÍCIO, SCF. *et al*. Causas e conseqüências de quedas de idosos atendidos em hospital público. **Revista de Saúde Pública**, vol. 38, n. 1, p. 93-99, fev. 2004. Disponível em: https://www.scielo.br/j/rsp/a/sHxR7CbcsVqpXvQsrfnWPtJ. Acesso em: 2 jul. 2021. https://doi.org/10.1590/S0034-89102004000100013.

FOWLER, M. **UML Essencial**. São Paulo: Bookman, 2011.

GARG, A. Ergonomics and the Older Worker: An Overview. **Experimental Aging Research**, v. 17, n. 3, p. 143-155. 1991. Disponível em: https://www.tandfonline.com/doi/abs/10.1080/03610739108253894. Acesso em: 1 jul. 2021. https://doi.org/10.1080/03610739108253894.

GOLDSCHMIDT, R.; PASSOS, E.; BEZERRA, E. **Data Mining** – Conceitos, técnicas, algoritmos, orientações e aplicações. São Paulo: Elsevier, 2015.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge: The MIT Press, 2016.

GRAHAM, H. J.; FIRTH, J. Home accidents in older people: role of primary health care team. **British Medical Journal**, v. 304, p. 30-32, jul. 1992. Disponível em: https://www.bmj.com/content/305/6844/30. Acesso em 6 jul. 2021. https://doi.org/10.1136/bmj.305.6844.30.

GRUS, J. **Data Science do Zero**. Rio de Janeiro: Editora Alta Books, 2021.

ISLAM, M. M. et al. Review on fall detection systems using data from Smartphone Sensors**. Ingénierie des Systèmes d'Information**, v. 24, n. 6, dez. 2019, p. 569-576. Disponível em: https://www.iieta.org/journals/isi/paper/10.18280/isi.240602. Acesso em 20 ago. 2021.

JAVA. O que é a Tecnologia Java e porque preciso dela?. 2021. Disponível em: https://www.java.com/pt-BR/download/help/whatis_java.html. Acesso em: 25 ago. 2021.

JENSEN, O. C. *et al.* Non-Fatal Occupational Injuries Related to Slips, Trips and Falls in Seafaring. **American Journal of Industrial Medicine**, v. 42, n. 2, p. 161-171, fev-2005. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/ajim.20119. Acesso em: 9 jul. 2021. https://doi.org/10.1002/ajim.20119.

KEMMLERT, K.; LUNDHOLM, L. Slips, trips and falls in different work groups — with reference to age and from a preventive perspective. **Applied Ergonomics**, v. 32, n. 2, p. 149-153, abr. 2001. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S000368700000051X. Acesso em: 29 jun. 2021. https://doi.org/10.1016/S0003-6870(00)00051-X.

KERAS. 2021. Disponível em: https://keras.io/. Acesso em: 26 ago. 2021.

LACHMAN, M. E. *et al*. Fear of Falling and Activity Restriction: The Survey of Activities and Fear of Falling in the Elderly (SAFE). **The Journals of Gerontology: Series B**. v. 53b, n. 1, p. 43-50, jan-1998. Disponível em: https://academic.oup.com/psychsocgerontology/article/53B/1/P43/586909. Acesso em: 15 jul. 2021. https://doi.org/10.1093/geronb/53B.1.P43.

LAYNE, L. A.; POLLACK, K. M. Nonfatal occupational injuries from slips, trips, and falls among older workers treated in hospital emergency departments, United States 1998. **American journal of industrial medicine**, v. 46, n. 1, p. 32-41, jul. 2004. Disponível em: https://pubmed.ncbi.nlm.nih.gov/15202123/. Acesso em: 30 jun. 2021. https://doi.org/10.1002/ajim.20038.

LEAMON, T. B.; MURPHY, P. L. Occupational slips and falls: more than a trivial problem. **Ergonomics**, v. 38, n.3, p. 487-498. 1995. Disponível em:

https://www.tandfonline.com/doi/abs/10.1080/00140139508925120. Acesso em: 30 jun. 2021. https://www.tandfonline.com/doi/abs/10.1080/00140139508925120.

LEE, J. -S.; TSENG, H. -H. Development of an Enhanced Threshold-Based Fall Detection System Using Smartphones With Built-In Accelerometers. **IEEE Sensors Journal**, v. 19, n. 18, p. 8293-8302, set-2019. Disponível em: https://ieeexplore.ieee.org/abstract/document/8721091. Acesso em 13 jul. 2021. https://doi.org/10.1109/JSEN.2019.2918690.

LENZ, M. L. et al. **Fundamentos de Aprendizagem de Máquina**. Porto Alegre: Grupo A, 2020.

LIMA, I.; PINHEIRO, C. A. M.; SANTOS, F. A. O. **Inteligência Artificial**. São Paulo: Elsevier, 2014.

LINDEMANN, U. *et al*. Evaluation of a fall detector based on accelerometers: A pilot study, **Medical and Biological Engineering and Computing**. v. 43, p. 548-551, out-2005. Disponível em: https://link.springer.com/article/10.1007/BF02351026. Acesso em: 15 jul. 2021. https://doi.org/10.1007/BF02351026.

MANZANO, J. A. N. G.; AFFONSO COSTA, R. **Programação de Computadores com Java**. São Paulo: Editora Érica, 2014.

MATHWORKS. Documentation. Accelerometer. 2021. Disponível em: https://www.mathworks.com/help/supportpkg/android/ref/accelerometer.html. Acesso em: 25 ago. 2021.

MAYER, J. D. Emergency Medical Service: Delays, Response Time and Survival. **Medical Care**, v. 17, n. 8, p. 818–827, ago. 1979. Disponível em: https://www.jstor.org/stable/3764282. Acceso em: 29 jun. 2021.

MCCULLOCH, W. S.; PITTS, W. A. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, p. 115-133, 1943. Disponível em: https://link.springer.com/article/10.1007%2FBF02478259. Acesso em: 15 ago. 2021.

MEIO BIT. **13 anos atrás, Google comprava uma pequena empresa chamada Android Inc.** 2018. Não paginado. Disponível em: **https://tecnoblog.net/meiobit/389558/13-anos-atras-google-comprava-uma-pequena-empresa-chamada-android-inc/**. Acesso em 24 ago. 2021.

MIDDLETON, P. M. *et al*. The pre-hospital epidemiology and management of spinal cord injuries in New South Wales: 2004–2008. **Injury**, v. 43, n. 4, p.480-485, dez. 2011. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S002013831100581X. Acesso em: 7 jul. 2021. https://doi.org/10.1016/j.injury.2011.12.010.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos Sobre Aprendizado de Máquina. **Sistemas Inteligentes Fundamentos e Aplicações**. Barueri-SP: Manole Ltda, 2003.

OLIN, H. S; HACKETT, T. P. The Denial of Chest Pain in 32 Patients With Acute Myocardial Infarction. **JAMA**, v. 190, n. 11, p. 977-981, dez. 1964. Disponível em: https://jamanetwork.com/journals/jama/article-abstract/1165764. Acesso em: 8 jul. 2021. https://doi.org/10.1001/jama.1964.03070240023006.

OLIVEIRA, A. C. S. et al. Aplicação de redes neurais artificiais na previsão da produção de álcool. **Ciência e Agrotecnologia**, v. 34, n. 2, p. 279-284, abr. 2010. Disponível em: https://www.scielo.br/j/cagro/a/HWFRGpHsMrQkShH8WDXggbM/. Acesso em: 20 ago. 2021. https://doi.org/10.1590/S1413-70542010000200002.

ORACLE. Oracle Java. 2021. Disponível em: https://www.oracle.com/java/. 2021. Acesso em: 25 ago. 2021.

OSHA - OCCUPATIONAL SAFETY AND HEALTH ADMINISTRATION, U.S. Department of Labor. Recomendações de segurança para trabalho na indústria naval, 2016. Disponível em: https://www.osha.gov/sites/default/files/housekeeping.pdf. Acesso em: 4 jul. 2021.

ÖZDEMIR, A. T; BARSHAN, B. Detecting Falls with Wearable Sensors Using Machine Learning Techniques. **Sensors**, v. 14, n. 6, p. 10691-10708, jun. 2014. Disponível em: https://www.mdpi.com/1424-8220/14/6/10691. Acesso em 14 jul. 2021. https://doi.org/10.3390/s140610691.

PAGOTTO et al. Scrum solo. In: Iberian Conference on Information Systems and Technologies (CISTI), 11., 2016, Grã-Canária, Espanha. **Anais...** Grã-Canária: 2016.

PANDAS. 2021. Disponível em: https://pandas.pydata.org/. Acesso em: 26 ago. 2021.

PETERS, J.; PEDRYCZ, W. **Engenharia de Software**: teoria e prática. Rio de Janeiro: Campus, 2001.

PIFFER, S. et al. Home accidents in the province of Trento. Ten years of observations regarding admissions to the emergency and first aid department. **Annali di Igiene : Medicina Preventiva e di Comunita**. v. 33, n. 2, p. 152-162, mar-abr. 2021. Disponível em: https://europepmc.org/article/med/33570087. Acesso em 6 jul. 2021. https://www.doi.org/10.7416/ai.2021.2421

POWERS, D. M. W. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. **Journal of Machine Learning Technology**, v. 2, n.1, p. 37-63, 2011.

PRESSMAN, R.; MAXIM, R. **Engenharia de Software**. Porto Alegre: Grupo A, 2016.

PUHL, F.L.P. et al. **Robótica**. Porto Alegre: SAGAH, 2019.

REDHAT. AMBIENTE DE DESENVOLVIMENTO INTEGRADO, O que é IDE?. 2021. Disponivel em: https://www.redhat.com/pt-br/topics/middleware/what-is-ide. Acesso em 27 ago. 2021.

REINEHR, S. **Engenharia de Requisitos**. Porto Alegre: Grupo A, 2020.
ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, v. 65, n. 6, p. 386-408, 1958.

RUBENSTEIN, L. Z.; JOSEPHSON, K. R. The epidemiology of falls and syncope. **Clinics in Geriatric Medicine**, v. 18, n. 2, p. 141-158, mai. 2002. Disponível em: https://www.geriatric.theclinics.com/article/S0749-0690(02)00002-2/. Acesso em: 29 jun. 2021. https://doi.org/10.1016/S0749-0690(02)00002-2.

RUSSEL, S; NORVIG, P. **Artificial Intelligence** - A Modern Approach. New Jersey: Pearson, 2010.

SEJNOWSKI, T. J. **A Revolução do Aprendizado Profundo**. Rio de Janeiro: Alta Books, 2019.

SILVA, A. M. R; VIDEIRA, C. A. E. **UML, Metodologias e ferramentas CASE**. Lisboa: Centro Atlântico, 2001.

SILVA, F. M. et al. **Inteligência Artificial**. Porto Alegre: SAGAH, 2019.

SILVA, L. A; PERES, S. M; BOSCARIOLI, C. **Introdução à Mineração de Dados Com Aplicações em R**. Rio de Janeiro: Elsevier, 2016.

SIMAS, V. L. et al. **Desenvolvimento para dispositivos móveis** - Volume 2. Porto Alegre: SAGAH, 2019.

SOMMERVILLE, I. **Engenharia de Software**. São Paulo: Pearson Addison Wesley, 2007.

STATCOUNTER - GLOBAL STATS. Mobile Operating System Market Share Worldwide. 2021. Disponível em: https://gs.statcounter.com/os-market-share/mobile/worldwide. Acesso em: 10 jul. 2021.

STATISTA. Statistics. Technology & Telecommunications. Number of smartphone users from 2016 to 2021. Number of smartphone users from 2016 to 2021. 2021. Disponível em: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. Acesso em: 10 jul. 2021.

STEVAN, L. S.; SILVA, R. A. **Automação e Instrumentação Industrial com Arduino** - Teoria e Projetos. São Paulo: Editora Érica, 2015.

TENSORFLOW. 2021. Disponível em: https://www.tensorflow.org/. Acesso em: 26 ago. 2021.

THOMAZINI, D.; ALBUQUERQUE, P.U.B. **Sensores Industriais** - Fundamentos e Aplicações. São Paulo: Editora Saraiva, 2011.

TIOBE. Tiobe Index. 2021. Disponível em: https://www.tiobe.com/tiobe-index/. Acesso em: 25 ago. 2021.

TONSIG, S. L**. Engenharia de Software**. São Paulo: Futura, 2003.

TSINGANOS, P.; SKODRAS, A. A smartphone-based fall detection system for the elderly. In: INTERNATIONAL SYMPOSIUM ON IMAGE AND SIGNAL PROCESSING AND ANALYSIS, 10., 2017, Liubliana. **Anais...** Liubliana, Eslovênia, 2017. p. 53-58.

TYSON, G. SPOSARO, F. iFall: An android application for fall monitoring and response. In: Annual International Conference of the IEEE Engineering in Medicine and Biology Society. 2009. Minneapolis, EUA. **Anais…** Minneapolis: 2009.

UNGAR, A. *et al*. Fall prevention in the elderly. **Clinical Cases in Mineral and Bone Metabolism**. v. 10, n. 2, p. 91-95, mai-ago. 2013. Disponível em: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3797008/. Acesso em: 8 jul. 2021.

VALLABH, P. et al. Fall Detection Using Machine Learning Algorithms. In: International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 24., 2016. Split, Croácia. **Anais**... Split: FESB - University of Split, 2016. p. 1-9.

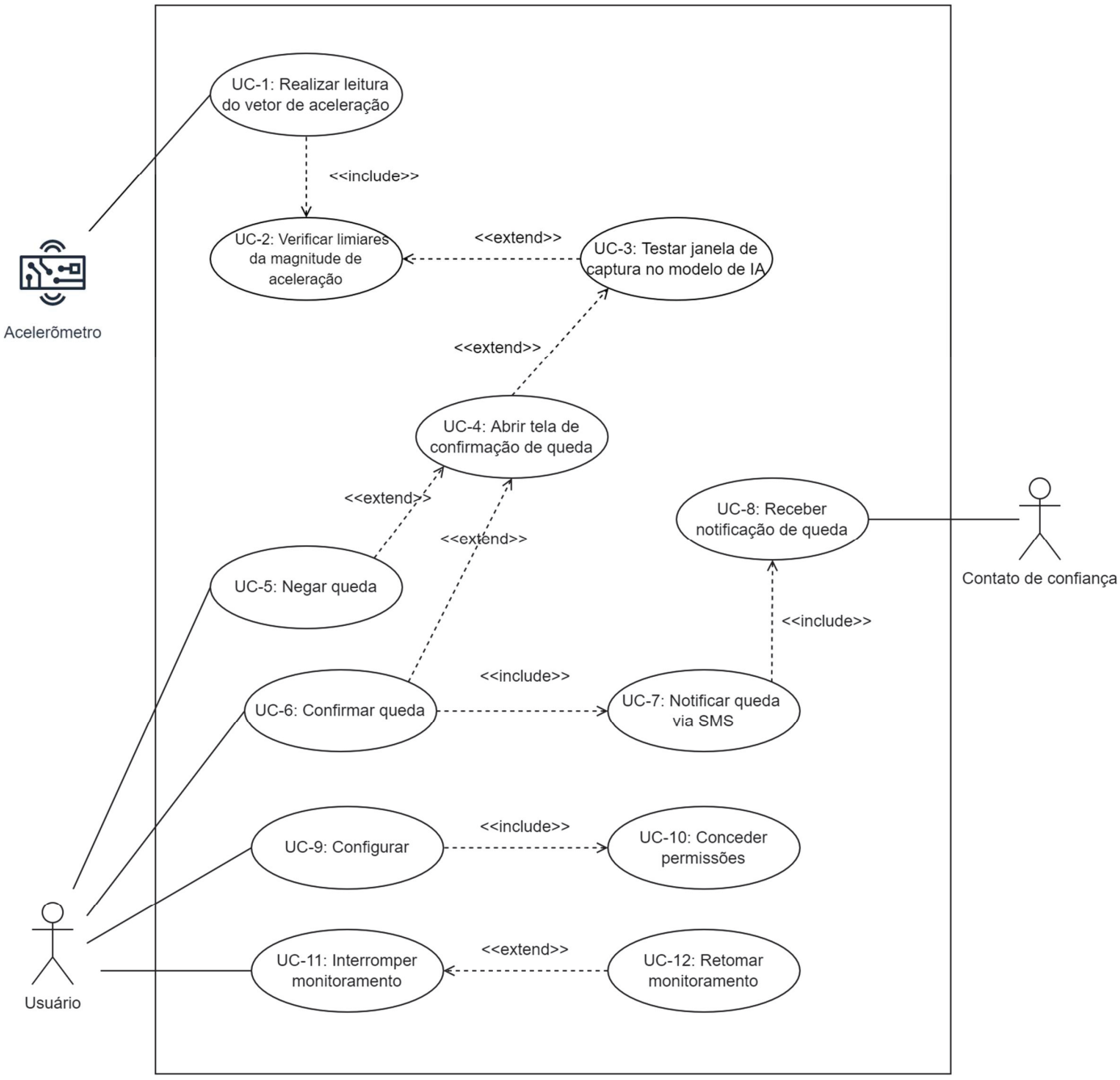
VAVOULAS, G. et al. The MobiFall dataset: An initial evaluation of fall detection algorithms using smartphones. In: IEEE International Conference on BioInformatics and BioEngineering, 13. 2013. Chania, Grécia. **Anais…** Chania: Minoa Palace Resort and Spa, nov. 2013. p. 1-4.

VAVOULAS, G. et al. The MobiFall Dataset: Fall Detection and Classification with a Smartphone. **International Journal of Monitoring and Surveillance Technologies Research**. v. 2, n. 1, p. 44-56, 2014. www.doi.org/10.4018/ijmstr.2014010103.

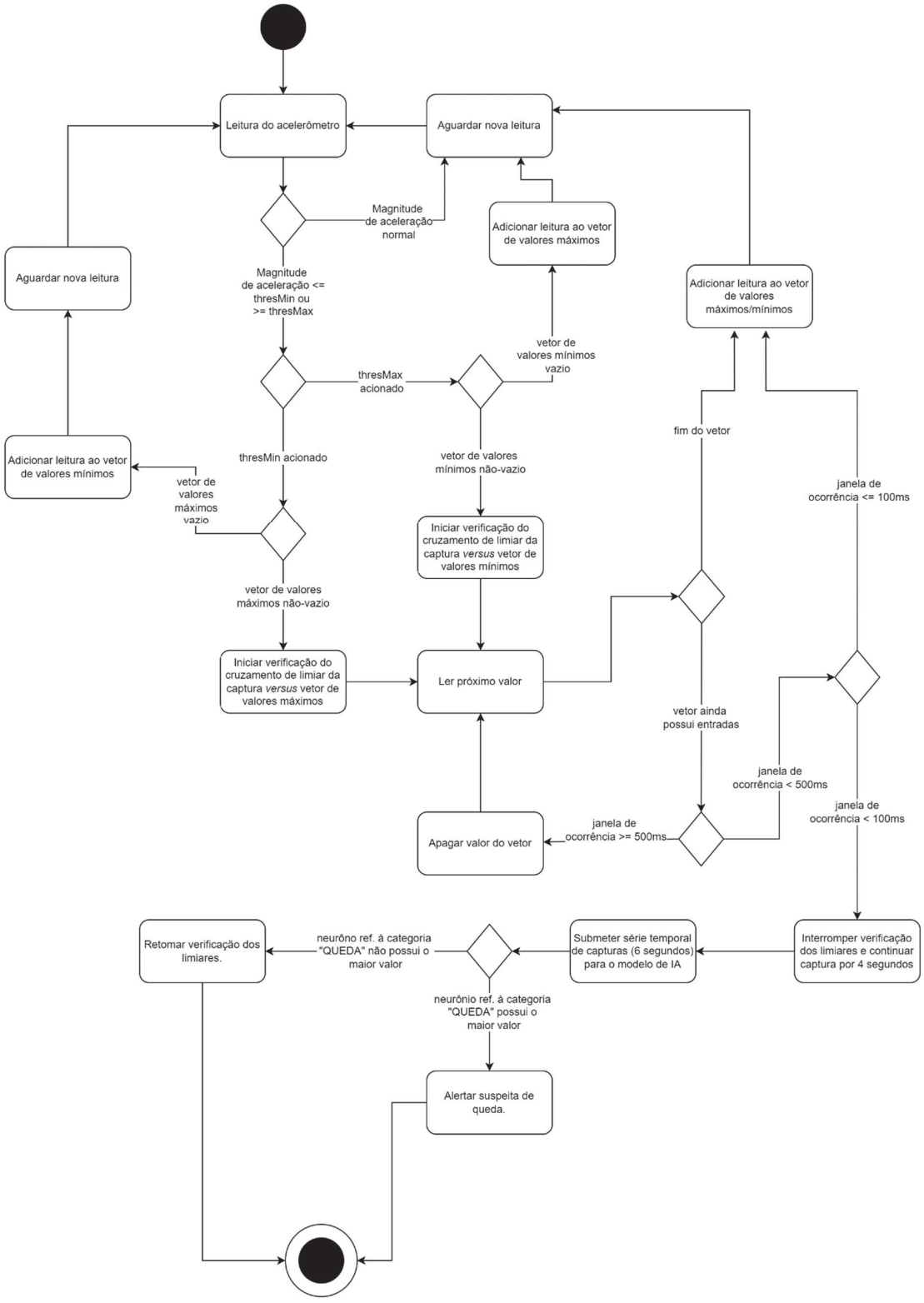
VAVOULAS, G. et al. The MobiAct Dataset: Recognition of Activities of Daily Living using Smartphones. In: International Conference on Information and Communication Technologies for Ageing Well and e-Health, 2. 2016. Roma, Itália. **Anais…** Roma: Hotel Barceló Aran Mantegna, 2016. p. 143-151.

WHO - WORLD HEALTH ORGANIZATION. Newsroom. Fact sheets. Falls, 2021. Disponível em: **https://www.who.int/news-room/fact-sheets/detail/falls**. Acesso em: 28 jun. 2021.

APÊNDICE 1 – DIAGRAMA DE CASOS DE USO



APÊNDICE 2 – DIAGRAMA DE ATIVIDADES: MONITORAMENTO E DETECÇÃO DE QUEDAS



APÊNDICE 3 – DIAGRAMA DE CLASSES