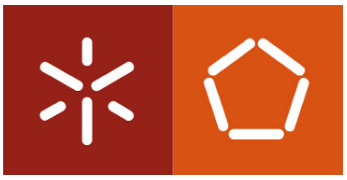


Universidade do Minho

Mestrado em Engenharia Informática

Perfil de Sistemas Inteligentes

Sensorização e Ambiente

Rui Moreira (PG50736)
Bernardo Saraiva (PG50259)
José Gonçalves (PG50519)

18 de Março de 2023

Sistema de Deteção de Quedas

Conteúdo

Capítulo 1

Introdução

O seguinte projeto foi desenvolvido no âmbito da Unidade Curricular de Sensorização e Ambiente, constituinte do perfil de Sistemas Inteligentes, onde a principal motivação foi implementar um sistema de sensorização, tirando partido da integração de sensores físicos e/ou virtuais, focando em domínios emergentes como a Sensorização Móvel, a Computação Humanizada e as Cidades Inteligentes. Este sistema desenvolvido deve ser capaz de obter dados e gerar informação útil no contexto do ambiente onde se encontra inserido.

Naturalmente, existem inúmeros problemas nos grandes centros urbanos, muitos destes, relacionados com saúde e bem-estar da população. Nos últimos anos, a preocupação com a segurança e bem-estar dos idosos e pessoas com mobilidade reduzida tem aumentado significativamente. As quedas são uma das principais causas de lesões graves nesses grupos e podem ter consequências físicas e psicológicas graves.

Tendo isto em conta, neste trabalho prático pretende-se desenvolver um sistema de deteção de quedas que possa fornecer tranquilidade e segurança tanto para os utilizadores quanto para seus familiares e cuidadores, permitindo que a pessoa continue a viver de forma independente, com a certeza de que há um suporte disponível em caso de necessidade.

Capítulo 2

Contextualização e Objetivos

Atualmente, grande parte da população possui *smartphones* ou qualquer outro tipo de dispositivos móveis modernos, o que se traduz em dispositivos mais completos e com maior quantidade de sensores, para além de mais precisos. Deste modo, um dos grandes objetivos passa por tirar o melhor proveito destes sensores no uso quotidiano dos dispositivos.

Uma aplicação móvel de deteção de quedas é uma solução tecnológica projetada para monitorizar movimentos e identificar quedas em tempo real por meio de sensores embutidos em dispositivos móveis como *smartphones* ou *smartwatches*, aproveitando os acelerómetros, giroscópios e outros sensores presentes nos dispositivos móveis para identificar padrões de movimento característicos de uma queda. Esta tecnologia tem como objetivo proporcionar segurança e assistência imediata em casos de acidentes. Ao detetar uma queda, a aplicação móvel aciona um alerta e efetua uma chamada de emergência automática para o serviço nacional de socorro. Essa solução é acessível e fácil de usar, pois aproveita os dispositivos móveis que a maioria dos cidadãos já possuem. Com a deteção e resposta rápida a quedas, a aplicação móvel contribui para a segurança e tranquilidade dos utilizadores, proporcionando uma intervenção médica imediata em casos de emergência.

Para isto é preciso, inicialmente, recolher dados para construir um modelo eficaz, para ser importado numa aplicação final com este objetivo.

Capítulo 3

Recolha de Dados

Nesta fase inicial de recolha dos dados, foi importante decidir quais os dados mais relevantes a coletar num contexto de deteção de quedas. Após uma breve pesquisa, o grupo chegou à conclusão que seriam os dados provenientes de 3 sensores, **acelerómetro**, **giroscópio** e **magnetómetro**, que dariam origem a 3 tipos:

- **AccelerometerData -** contém 5 campos (*id, valuex, valuey, valuez* e *accuracy*)
- **GyroscopicData -** contém 4 campos (*id, valuex, valuey* e *valuez*)
- **OrientationData -** contém 4 campos (*id, valuex, valuey* e *valuez*). Estes dados são obtidos através dos sensores acelerómetro e magnetómetro da seguinte maneira:

Primeiro obtém-se *rotationMatrix* através da função *SensorManager.getRotationMatrix* que recebe como argumentos os dados do acelerómetro e magnetómetro.

Após isto, aplicou-se a função *SensorManager.getOrientation* que recebe como argumento a *rotationMatrix*, e devolve os valores *azimuth*, *pitch* e *roll* (colocados no *OrientationData* como '*valuex*','*valuey*','*valuez*' respetivamente) .

De modo a perceber estes últimos valores do *OrientationData*, apresenta-se na figura 3.1 uma ilustração das coordenadas utilizadas na orientação.

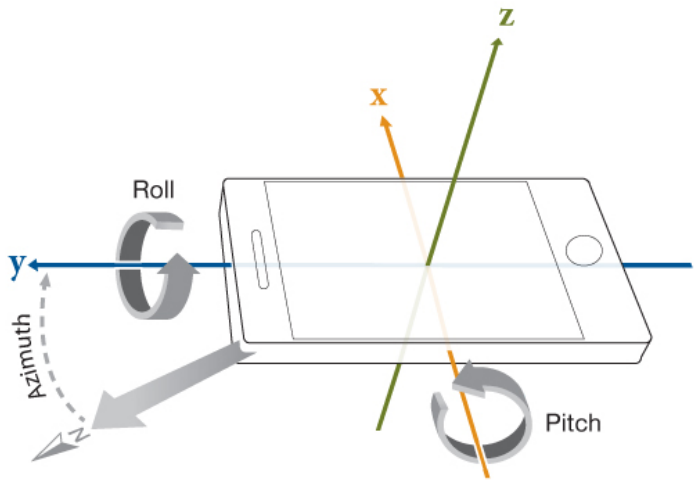


Figura 3.1: Figura explicativa das coordenadas utilizadas para orientação

Para recolher os dados, decidiu-se produzir uma aplicação *Android* onde isto fosse possível e ainda, armazená-los numa *CloudStore* no *Firebase*.

O resultado final desta aplicação é apresentado abaixo.

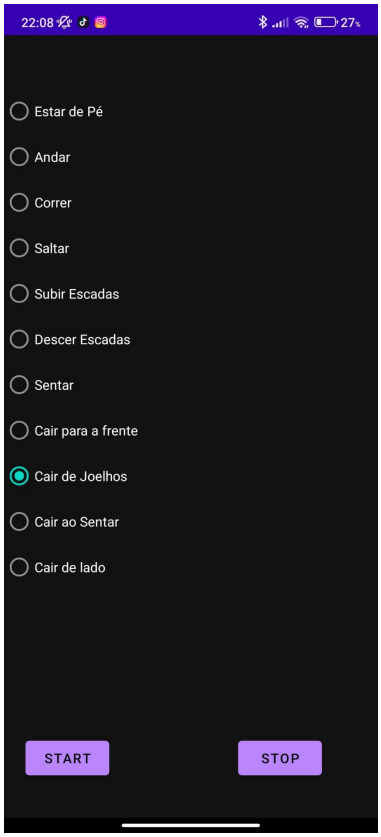


Figura 3.2: App Recolha de Dados

Como é possível ver na Figura acima ,3.2, esta aplicação tem 9 botões que permite ao utilizador selecionar a *label* dos dados que vai recolher. As *labels* escolhidas foram as seguintes:

- Estar de pé - STD (*Standing*)
- Andar - WAL (*Walking*)
- Correr - JOG (*Jogging*)
- Saltar - JUM (*Jumping*)
- Subir escadas - STU (*Stairs up*)
- Descer escadas - STN (*Stairs down*)
- Sentar - CH (*Sit chair*)
- Cair para a frente - FOL (*Fall forward lying*)
- Cair de joelhos - FKL (*Fall front knees lying*)
- Cair ao sentar - BSC (*Back-sitting-chair*)
- Cair de lado - SDL (*Sideward lying*)

Deciciu-se adotar as 9 *labels* apresentadas acima para, posteriormente, tentar incorporar um *dataset* externo (que continha, entre outras, estas *labels*) com o objetivo de ter mais dados para treinar o modelo de *machine learning*, que será detalhado mais à frente no relatório na secção 4.1.

A *app* contém ainda dois botões com nomes muito intuitivos, um de *'start'* para iniciar a recolha de dados e outro de *'stop'* para parar.

Nesta fase inicial de recolha de dados, 3 utilizadores utilizaram esta aplicação e recolheram dados 5 vezes para cada *label*(durante, aproximadamente, 10 segundos), dando no total, 135 testes (3x5x9).

Capítulo 4

Modelo Machine Learning

De modo a permitir o desenvolvimento de um modelo suficientemente capaz de prever quedas, foram utilizadas técnicas de *Machine Learning* que permitem treinar e aprimorar o modelo, de forma a que a eficácia seja tão elevada quanto possível para que o Sistema de Deteção de Quedas possa ser fiável.

4.1 Tratamento de Dados

Após recolhidos os dados, tal como explicado no capítulo 3, exportou-se os dados guardados no *Firebase* para um ficheiro *json*. Posteriormente, através de um *script python*, converteu-se estes dados num ficheiro *csv* já com as colunas pretendidas para o *Dataset*:

- **testId:** Identificador do número correspondente ao teste que um dado registo foi obtido
- **sampleNo:** Valor correspondente ao número de sequencia de um teste
- **acc_x:** Valor correspondente ao eixo do x do acelerómetro
- **acc_y:** Valor correspondente ao eixo do y do acelerómetro
- **acc_z:** Valor correspondente ao eixo do z do acelerómetro
- **gyro_x:** Valor correspondente ao eixo do x do giroscópio
- **gyro_y:** Valor correspondente ao eixo do y do giroscópio
- **gyro_z:** Valor correspondente ao eixo do z do giroscópio
- **azimuth:** Coordenada esférica relativa à orientação do dispositivo
- **pitch:** Coordenada esférica relativa à orientação do dispositivo
- **roll:** Coordenada esférica relativa à orientação do dispositivo
- **label:** Campo correspondente ao tipo de atividade, em que é atribuida uma label dependedo da atividade registada

De modo a alcançar um modelo mais eficaz, procurou-se complementar os dados recolhidos com os de um novo, chamado Dataset Mobifall. Deste modo, procurou-se atribuir mais observações ao modelo, tornando-o mais generalizável e robusto.

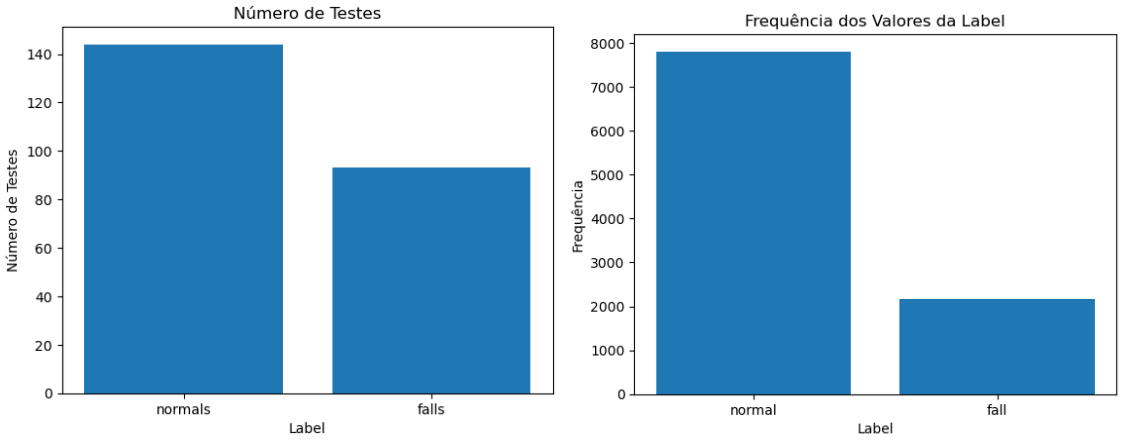
Este dataset, que inicialmente vinha em vários ficheiros com as diferentes labels, foi trabalhado de modo a que se apresentasse em um único *dataset* com o mesmo formato do que foi gerado através dos dados recolhidos.

Porém com a junção deste *dataset* aos dados recolhidos pelo grupo, o modelo acabou por não reproduzir os melhores resultados. Isto pode ser explicado, pelo facto de existir uma grande diferença entre os sensores utilizados em ambos os testes. O modelo desenvolvido com estes dados revelava bons resultados para os dados do *dataset* externo, porém, quando testado com os dados recolhidos pelo grupo, produziu péssimos resultados, obtendo uma *accuracy* inferior a 20%.

Assim, acabou-se por descartar este *dataset* externo e recolher mais alguns dados para fortalecer o modelo. Além disto, posteriormente, para tornar mais simples o treino do modelo, converteu-se as 9 *labels* em apenas 2: *Fall* e *Normal*.

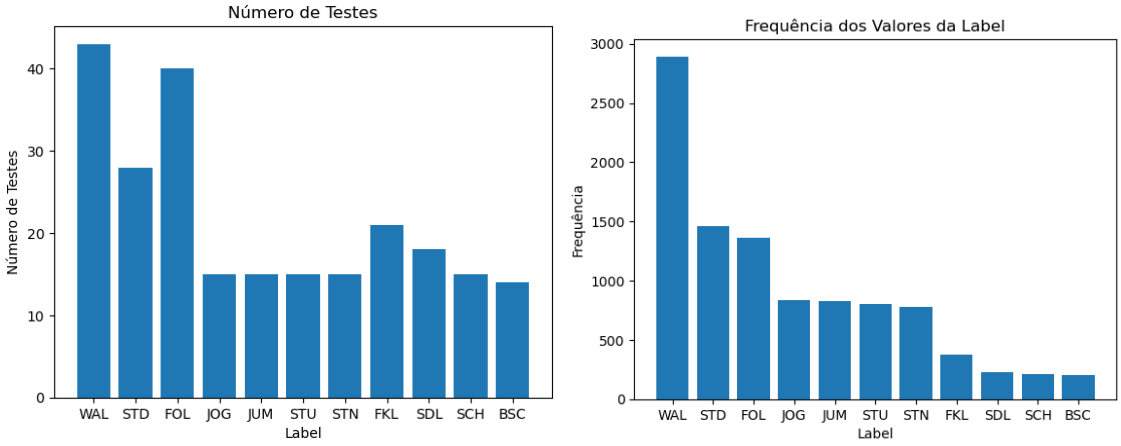
4.2 Visualização dos Dados

De modo a permitir uma perceção estatística dos dados utilizados, foram aplicadas algumas técnicas de visualização, as quais serão demonstradas e explicadas em seguida.



(a) Nº de testes para cada uma das *labels* (b) Quantidade de Dados para cada uma das *labels*

Figura 4.1



(a) Nº de testes para cada tipo de atividade (b) Quantidade de testes para cada tipo de atividade

Figura 4.2

Com o objetivo de verificar a quantidade de testes efetuados para cada *label*, foi gerado o gráfico da Figura 4.1a, que indica que o número de testes classificados como *fall* foi cerca de 90,

e o número de testes classificados como *normal* foi cerca de 145, estando assim o modelo mais familiarizado com atividades normais.

Pela a análise à Figura 4.1b e 4.2b, é possível concluir que cada teste do tipo "*Fall*"recolhe menos dados dos sensores em comparação aos testes do tipo "Normal", visto que, na recolha dos dados do tipo "*Fall*", foi tido o cuidado de iniciar a recolha no instante mesmo no início da queda e parar a recolha no fim da mesma, resultando assim, em menos dados recolhidos por cada teste.

Através de uma análise em maior detalhe para as 9 *labels* definidas inicialmente, é possível verificar pela Figura 4.2a que as que contêm maior número de testes são *WAL* (andar a pé), *FOL* (Cair para a frente) e *STD* (estar de pé). Isto deve-se ao facto de, numa segunda fase terem sido recolhidos mais dados para compensar a não utilização do *dataset* externo.

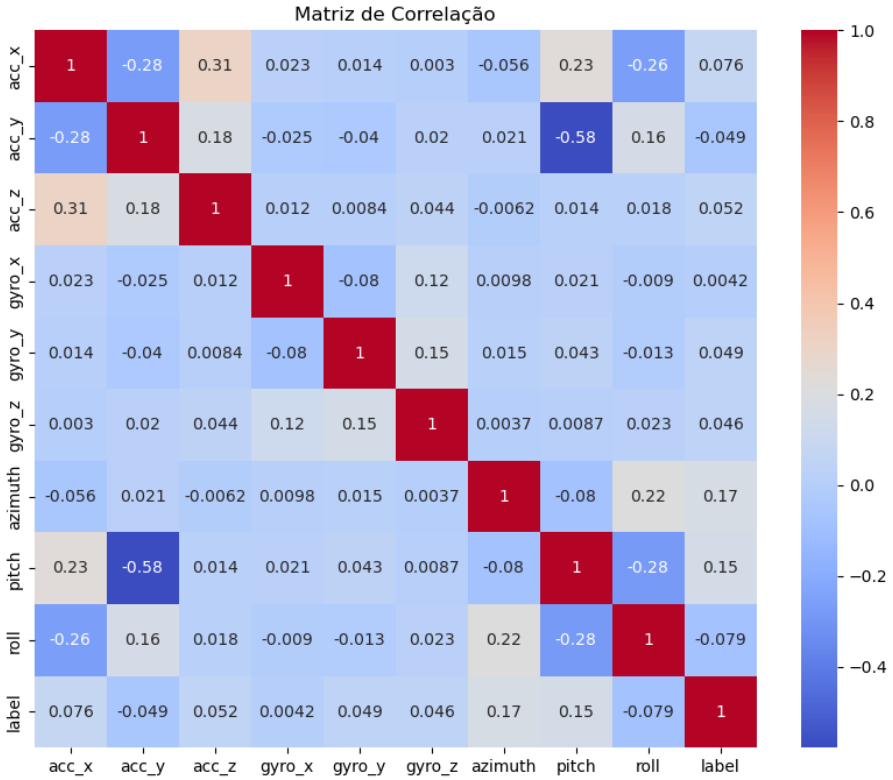


Figura 4.3: Matriz de correlação

Quanto à análise da matriz de correlação, não foram verificadas grandes correlações, excetuando o caso de *acc_y* com *pitch*, que é um caso relativamente expectável, uma vez que os dados *azimuth*, *pitch* e *roll* são inferidos através dos valores do acelerómetro e magnetómetro . Apesar de ser expectável verificar alguma correlação de algum dos atributos com a 'label', esta correlação acaba por não acontecer, e isto pode ser explicado pelo facto de um *sample* apenas não ser capaz de evidenciar o movimento.

4.3 Arquitetura do Modelo concebido

Nesta fase, o grupo deparou-se com a adversidade de uma linha do *dataset* (com um valor do acelerómetro, giroscópio e orientação) não possuir propriamente informação suficiente para o modelo prever o movimento. Era necessário ter um conjunto de linhas, ou seja, uma sequência dos vários valores no tempo, para o modelo conseguir extrair informação relevante para o objetivo proposto.

Assim, implementou-se uma técnica de *sliding window* (janela deslizante), que consiste no

processamento de dados sequencias, permitindo analisar sequências de dados ao longo do tempo, para extrair características relevantes e possibilitar a inferência do movimento, através da formação de janelas deslizantes.

Uma janela deslizante é caracterizada por valores como tamanho da janela e tamanho do salto, pelo que se considerou que os melhores valores seriam:

- Tamanho da janela: 30
- Tamanho do salto: 10

Isto cria uma sobreposição entre as janelas adjacentes, pois cada janela compartilha parte dos dados com a janela anterior.

No que diz respeito aos modelos de *machine learning*, estes foram implementados de modo a classificar os dados recolhidos como 'Fall' ou 'Normal'. Após alguns testes, chegou-se à conclusão que o modelo com melhor accuracy (85%) seria uma rede neuronal convolucional, com 25 épocas, learning rate de 0.001 e com as seguintes camadas:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 29, 8, 16)         80

 dropout (Dropout)           (None, 29, 8, 16)         0

 conv2d_1 (Conv2D)           (None, 28, 7, 32)         2080

 dropout_1 (Dropout)         (None, 28, 7, 32)         0

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 64)                401472

 dropout_2 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 2)                 130

=================================================================
Total params: 403,762
Trainable params: 403,762
Non-trainable params: 0
_________________________________________________________________
```

Figura 4.4: Resumo da Arquitetura do modelo

Capítulo 5

Aplicação Final

Já na fase de desenvolvimento do produto final, decidiu-se desenvolver uma aplicação em *Android Studio*, que estivesse constantemente a coletar dados até que seja detetada uma queda e, por consequente, ligar para a linha de emergência. Aquando da deteção de uma queda, a aplicação ativa um *timer* para permitir ao utilizador cancelar a chamada de emergência, sendo que em caso contrário, a aplicação assume que o utilizador precisa realmente de ajuda e efetua automaticamente a chamada.

Para isto, foi necessário integrar o modelo *machine learning* na aplicação no formato TensorFlow Lite (*.tflite*)

A aplicação tem a seguinte ordem de execução:

1. Como é possível verificar pela Figura 5.1, a app começa por apresentar a informação de que está a recolher dados para prever o movimento.

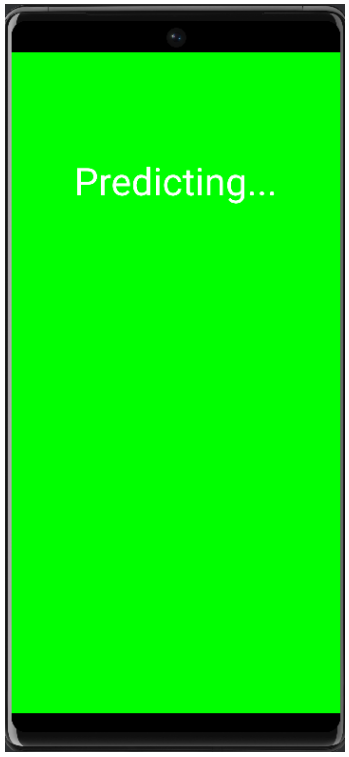
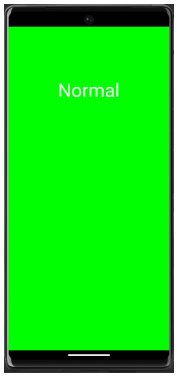
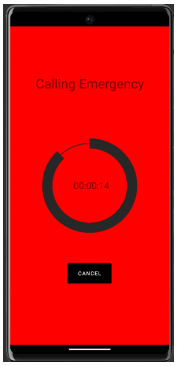


Figura 5.1: Menu Inicial

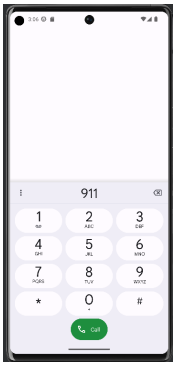
2. Depois de recolher dados suficientes (30 *samples*), o modelo prevê e se der como resultado '*Normal*' aparece como na Figura 5.2a. Se der '*Fall*', a *app* apresenta o *timer* (Figura 5.2b) e, no fim desse *timer*, se o utilizador não cancelar, liga para a linha de emergência (Figura 5.2c).



(a) Predict Normal



(b) Predict Fall - Timer



(c) Chamada de emergência

Figura 5.2

3. Tanto na situação do fim da chamada ou do utilizador cancelar a chamada, a *app* volta ao passo 1. para continuar a recolha de dados.

Capítulo 6

Análise crítica aos Resultados obtidos

Nesta fase, começou-se por fazer uma análise crítica aos resultados obtidos do modelo de *machine learning* desenvolvido.

Os dados de teste foram utilizados também como dados de validação, tendo sido possível obter uma *accuracy* de 85% nestes, como é possível verificar no gráfico abaixo.

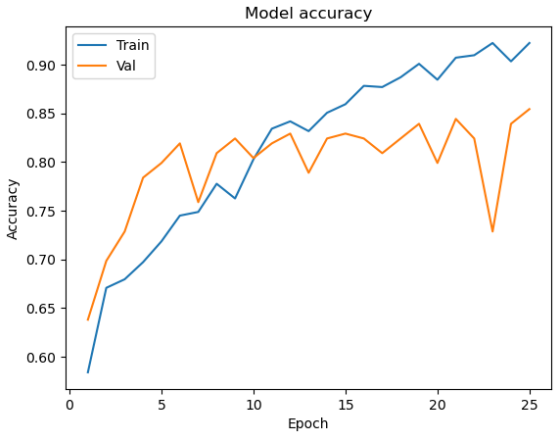


Figura 6.1: Curva de Aprendizagem

Analisando a Figura 6.2, é possível concluir que o modelo previu 58 quedas em 73, errando assim em 21%. Por outro lado, o modelo previu em 112 atividades normais em 126, errando 11%.

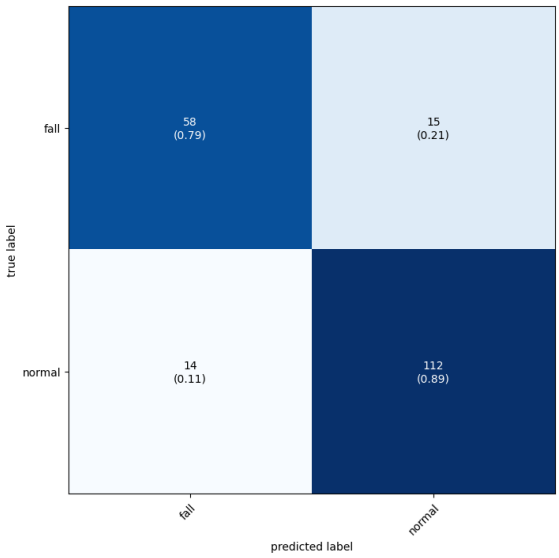


Figura 6.2: Matriz de Confusão

Passando aos testes à eficácia da aplicação final, ao simular os vários movimentos abordados, foi possível verificar que a aplicação mantém uma assertividade e uma consistência relativamente elevada na classificação dos acontecimentos, o que revela um aspeto bastante positivo.

Capítulo 7

Conclusão

A realização do presente trabalho prático permitiu consolidar e aprofundar os conhecimentos abordados ao longo do semestre, nomeadamente na área de utilização de sensores, assim como no desenvolvimento da aplicação *Android*. Permitiu também desenvolver competências como as utilizadas no manuseamento de redes neuronais e com o pipeline que envolveu a recolha através dos sensores dos dispositivos móveis, tratamento e utilização dos dados.

Considera-se o resultado final bastante satisfatório, no sentido em que, para além de concretizar o objetivo inicial, fá-lo com uma fiabilidade relativamente alta. Apesar de tudo, considera-se que a presença de mais dados seria uma vantagem para a precisão do modelo, conforme tentado sem êxito com um *dataset* externo.

Em suma, considera-se que o resultado é bastante positivo, cumprindo os objetivos propostos com clareza, e que, com algumas melhorias poderá ser algo útil e frequentemente utilizado pela população a correr em background nos seus smartphones.